ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

PRISM:

RIFRANGERE DISTRIBUZIONI DI PROBABILITÀ IN RIASSUNTI ASTRATTIVI MEDIANTE STRATEGIE DI DECODING

Elaborato in
Programmazione

Relatore
Prof. Antonella Carbonaro

Co-relatore
Dott. Giacomo Frisoni

Presentata da
David Cohen

Seconda Sessione di Laurea
Anno Accademico 2022 – 2023

PAROLE CHIAVE

Decoding Strategy

Transformer

Text Summarization

Natural Language Generation

Natural Language Processing

"In any system, including human ones, the element with the widest range of variability will be the one that determines the outcome."

Abstract

Negli ultimi quattro anni la summarization astrattiva è stata protagonista di una evoluzione senza precedenti dettata da nuovi language model neurali, architetture transformer-based, elevati spazi dimensionali, ampi dataset e innovativi task di pre-training. In questo contesto, le strategie di decoding convertono le distribuzioni di probabilità predette da un modello in un testo artificiale, il quale viene composto in modo autoregressivo. Nonostante il loro cruciale impatto sulla qualità dei riassunti inferiti, il ruolo delle strategie di decoding è frequentemente trascurato e sottovalutato. Di fronte all'elevato numero di tecniche e iperparametri, i ricercatori necessitano di operare scelte consapevoli per ottenere risultati più affini agli obiettivi di generazione. Questa tesi propone il primo studio altamente comprensivo sull'efficacia ed efficienza delle strategie di decoding in task di short, long e multi-document abstractive summarization. Diversamente dalle pubblicazioni disponibili in letteratura, la valutazione quantitativa comprende 5 metriche automatiche, analisi temporali e carbon footprint. I risultati ottenuti dimostrano come non vi sia una strategia di decoding dominante, ma come ciascuna possieda delle caratteristiche adatte a task e dataset specifici. I contributi proposti hanno l'obiettivo di neutralizzare il gap di conoscenza attuale e stimolare lo sviluppo di nuove tecniche di decoding.

Introduzione

Oggi è possibile automatizzare la generazione di riassunti tramite tecniche di *Automatic Text Summarization* [1] o semplicemente *Text Summarization*, che si basano principalmente su soluzioni di deep learning. In particolare, l'architettura *Transformer* [2] ha rivestito un ruolo fondamentale nell'intero ambito di downstream task del *Natural Language Processing* (NLP). La forte crescita del settore ha permesso lo studio di modelli neurali sempre più complessi e di tecniche sempre più efficaci ed efficienti. Nella summarization astrattiva [3], il modello deve imparare a comprendere il contenuto di un testo per parafrasarlo, quindi combinare i contenuti in una nuova forma, mantenendone il suo significato. La generazione del riassunto avviene in modo auto-regressivo (token dopo token). Ad ogni ciclo, nella predizione, il modello—sulla base dei pesi appresi durante il training—inferisce la probabilità di ciascun token del vocabolario. Sulla base di tale distribuzione di probabilità, viene operata la scelta effettiva in accordo a un criterio desiderato, noto anche come **strategia di decoding**. Possiamo considerarlo come ad un *modulo* (o interfaccia), che si pone alla fine di ogni modello neurale NLG con lo scopo di generare testo con caratteristiche desiderate. A seconda di come questo modulo è costruito, la conoscenza racchiusa nei parametri di un modello può essere tradotto in molteplici output. Metaforicamente, questo processo richiama il comportamento di un **prisma** (Figura 1), che rinfrange le distribuzioni di probabilità in riassunti diversi dipendenti dall'angolo e dal materiale.

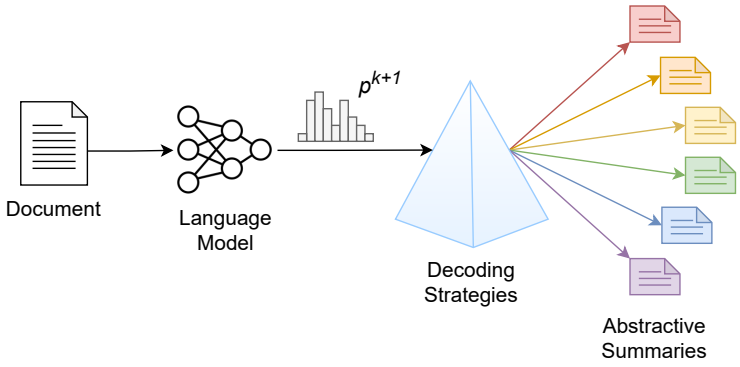


Figura 1: Inizialmente il documento viene codificato da un language model restituendo una distribuzione di probabilità. Le strategie di decoding (il prisma) restituiscono dei riassunti astrattivi diversi in base all'angolazione scelta e al materiale utilizzato.

Nonostante nuovi modelli siano costantemente proposti in letteratura, i contributi avanzati dalla comunità si focalizzano principalmente sul proporre nuove architetture (sempre più complesse), caratterizzate da alti spazi dimensionali, nuovi dataset e task di pre-training. In tal contesto, il ruolo della strategia di decoding viene spesso *trascurato e sottovalutato* [4]. Le ricerche attualmente presenti tra cui Holtzman et al. [4], Wiher et al. [5] e Ippolito et al. [6], paragonano le strategie più tradizionali (i.e. beam search, greedy e sampling) oppure ne introducono di nuove, ma non vi sono studi approfonditi sull'impatto in termini di performance di una strategia piuttosto che un'altra. Di fronte all'elevato numero di tecniche e iperparametri, *i ricercatori necessitano di operare scelte consapevoli per ottenere risultati più affini agli obiettivi di generazione*. Il campo della *Natural Language Generation* (NLG) rischia di *stagnarsi* se l'evoluzione dei modelli abilitata dai progressi del deep learning non è accompagnata anche da un'evoluzione consapevole delle strategie di decoding, che richiedono di essere approfondite e paragonate. Serve evidenziare che, a parità di modello, la strategia adottata può *ribaltare* completamente il testo predetto.

Il contributo di questa tesi è proporre uno studio comprensivo a larga scala focalizzandosi su determinati obiettivi.

- Vengono proposte le più *note strategie di decoding in un ampio spettro di iper-parametri* testato per la prima volta.
- Le strategie vengono testate su *tutti i task di summarization astrattivi esistenti*, tra cui **single, long e multi-document**;

- Per ogni task si prende il considerazione il *modello allo stato dell'arte*;
- Vengono impiegati *i più diffusi dataset pubblici con caratteristiche differenti*, tra cui dimensione, contenuto, forma e dominio;
- Vengono messi a confronto le predizioni con i riassunti target attraverso *molteplici metriche NLG*, *tempi di esecuzione* e *carbon footprint*.

Lo scopo è comprendere e analizzare se esista una strategia migliore in assoluto, oppure ciascuna può essere più efficace ed efficiente a seconda del dataset e task di summarization.

L'intera sperimentazione segue la vision della **Green NLP** [7]. In questo modo verrà aggiunta una metrica molto importante, in grado di determinare l'emissione di CO_2 per ogni strategia di decoding utilizzata. Questo permette di prendere visione degli impatti ambientali che le attività di origine antropica hanno sul *climate change* e, quindi, sul *surriscaldamento globale*. Un altro parametro importante che viene valutato per ogni task sono le risorse necessarie per la generazione di testo, quindi vengono prese in considerazioni le singole prestazioni in termini di *tempo e workload*.

La tesi si sviluppa nei seguenti capitoli:

- **Capitolo 1** - Tratta le principali nozioni teoriche: Automatic Text Summarization, architettura Transformers e i più noti modelli basati su di essa. Introduce le principali strategie di decoding adottate spiegando il loro funzionamento. Viene proposto un breve richiamo sui dataset le metriche di valutazione e la funzione del campionamento stratificato per una maggiore comprensione del progetto svolto.
- **Capitolo 2** - Introduce le principali problematiche esistenti dimostrando l'attuale gap presente. Propone uno studio con lo scopo di neutralizzare le problematiche esposte. Presenta le principali caratteristiche dell'ambiente di sviluppo, tra cui dataset, modelli e metriche scelti. Esplicita e motiva l'adozione del campionamento stratificato a discapito di altri.
- **Capitolo 3** - Mostra i principali risultati ottenuti attraverso grafi e tabelle. Confronta i risultati ottenuti e le loro possibili applicazioni.

Indice

Elenco delle figure

Capitolo 1

Framework teorico

Questo capitolo presenterà le nozioni teoriche che verranno utilizzate nei capitoli successivi. Inizialmente, viene introdotto il concetto base di **Automatic Text Summarization**, il suo funzionamento e le sue derivazioni. Verranno poi trattati i principali **task di Text Summarization** in relazione ai loro **dataset** più conosciuti e utilizzati. Alla fine, si parlerà dell'architettura **Transformer**, del suo funzionamento e dei punti focali che l'hanno resa lo scheletro dei modelli allo stato d'arte per NLP.

1.1 Automatic Text Summarization

La scrittura manuale di un riassunto (o sintesi) è un task il cui tempo di esecuzione dipende fortemente dalla lunghezza dell'input e dalla sua complessità. In generale, è uno dei task più impegnativi dell'NLP. Una definizione di riassunto è data dal Prof. Radev, ricercatore nel dipartimento di informatica alla Yale University: [1]:

> *A summary is a reductive transformation of a source text into a summary text by extraction or generation.*

Attualmente sono presenti dei modelli che ricevono in input dei testi e generano in output dei riassunti, attraverso tecniche di Automatic Summarization. Esse possono essere applicate—oltre che a elementi testuali—anche a immagini, video o dati multi-modali [10].

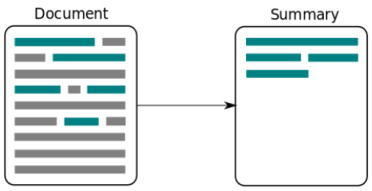


Figura 1.1: Da documento a riassunto. Esempio di Summarization estrattiva con selezione di blocchi di testo (in **verde acqua**) riproposti in un nuovo documento.

Alcune delle principali caratteristiche nell'utilizzo di metodologie automatiche nella generazione di riassunti sono:

- un notevole risparmio di tempo nella produzione di sintesi;
- la possibilità di produrre riassunti (accurati) di documenti —o un insiemi di essi— di lunghezza considerevole tramite long-document summarization [3];
- l'agevolazione nelle ricerche di documenti sintetizzati in domini specifici che risulterebbero costose da ottenere manualmente;
- il rimpiazzo di ricerche manuali nei domini per cui ottenere dei riassunti è altamente complesso (e.g. legale o biomedico) [1];
- la restituzione di output, più o meno formale rispetto a quello che si ottiene da riassunto svolto da un essere umano [3].

Le due tipologie più importanti nel task di text summarization definiscono la tipologia di riassunto prodotto. Esso può essere di due tipi: *estrattivo* o *astrattivo*. Il primo definisce un riassunto come l'unione di un subset scelto di frasi dal testo di input, il secondo prevede la generazione di testo — con lunghezza minima e massima regolabili — che non coincida 1:1 con il testo di partenza. Nel dettaglio:

- **Extractive Text Summarization**: dato il testo integrale, il task etichetta le frasi più importanti e informative e le compone per creare un unico riassunto. L'implementazione risulta semplice ma poco elastica per possibili variazioni. Il task estrattivo utilizza principalmente l'equazione $TF-IDF$ riportata nell'Equazione 3.1, che combina la misurazione della frequenza con cui un termine viene usato in un documento (TF) e la misurazione della frequenza con cui tale termine appare in tutte quelle raccolte (IDF):

$$w_{i,j} = tf_{i,j} \cdot log(N/df_i) \tag{1.1}$$

Tale che:
$tf_{i,j}$ è il numero di occorrenze di i in j
df_i è il numero di frasi contenenti i
N è il numero totale di frasi.

- **Abstractive Text Summarization**: dato il testo integrale, il task raccoglie le frasi più importanti che mantengono inalterato il contesto per poi successivamente rielaborarle e sintetizzarle. Questo consente un risultato personalizzato e più "human-written". Il task astrattivo risulta molto più elastico ma anche più complesso rispetto al precedente perché richiede l'utilizzo di un componente aggiuntivo chiamato *generatore*.

1.2 Extractive vs Abstractive Summarization

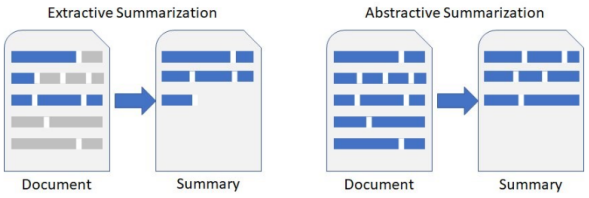


Figura 1.2: Vengono mostrate le principali differenze tra estrattiva (sinistra) ed astrattiva (destra). L'estrattiva mostra la permanenza del testo (in **blu**) anche nel riassunto, mentre l'astrattiva mostra la selezione intera del documento (in **blu**) creando un riassunto ex-novo.

L'approccio **extractive** segue un funzionamento comune per tutte le tecniche utilizzate.

1. Viene costruita una *rappresentazione* del testo che contiene le informazioni più importanti che può essere svolta:
   - identificando le parole che meglio descrivono il contesto;
   - utilizzando un insieme di feature per il calcolo del ranking. Quest'ultima può essere fatta tramite l'utilizzo di un grafo ad albero dove nelle foglie ci sono le frasi, mentre gli archi indicano la similarità tra esse. Oppure calcolata tramite tecniche di machine learning.
2. Ad ogni frase si assegna un *ranking*, quindi si attribuisce un peso (o punteggio).

3. Il riassunto finale è composto da un numero prefissato di frasi con punteggio maggiore. La selezione delle frasi può avvenire tramite diverse scelte algoritmiche, tra cui *greedy*, *minimizzazione* della ridondanza delle singole parole o frasi, oppure per *massima frequenza*.

L'approccio **abstractive** cambia il suo funzionamento in base alla tecnica utilizzata che può essere:

- **Sematic-Based**: viene costruita una rappresentazione semantica del testo basata sui concetti e sul significato del testo. In base alle informazioni estratte più rilevanti viene generato il riassunto.
- **Structured-Based**: la rappresentazione viene data in input ad un sistema di Natural Language Generation per generare il riassunto.
- **Deep Learning with Neural Networks**: l'estrazione di informazioni rilevanti dal testo utilizza delle feature tramite reti neurali. Nel tempo si sono sviluppate molte tipologie di reti neurali tra cui Convolutional Neural Networks (CNN) [11] e reti Sequence-to-sequence tra cui Recurrent Neural Networks (RNN) [12], Long-short term memory (i.e. LSTM) [13], Gated Recurrent Unit (GRU) [14] e modelli transformer-based [2].

I modelli basati su architettura **Transformer** sono attualmente considerati dalla comunità scientifica lo stato dell'arte in numerosi task downstream della NLP, e in particolare della NLG, tra cui spicca l'Abstractive Text Summarization.

1.3 Single-Document Summarization

Il task di single-document summarization (SDS) elabora singoli documenti. SDS utilizza sia tecniche estrattive che astrattive per la generazione di testo. Se viene utilizzata l'estrattiva, ci sarà la necessità di evitare documenti troppo lunghi per non avere ridondanze.

Il "target" è di generare un riassunto informativo Sum da un unico documento D. D consiste di M_i frasi $\{s_i | i \in [1, M_i]\}$. La variabile s_i è riferita alla i-esima del singolo documento.

SDS, in base alla dimensionalità del documento, si suddivide a sua volta in due sotto-tipologie.

- **Short-document**: sono principalmente documenti di lunghezza contenuta. Contengono singole frasi o paragrafi. Le fonti principali sono recensioni, articoli e documenti frammentati.
- **Long-document** [15]: sono principalmente documenti ampi. Contengono almeno centinaia di parole. Le fonti principali derivano da documenti giudiziari, cronache e ricerche scientifiche.

– I dataset più utilizzati per i task di short-document summarization sono:

- **Xsum** (Extreme Summarization) [16]: consiste in 226,711 articoli della BBC dal 2010 al 2017. Comprende un ampio range di domini tra cui: news, politica, sport, famiglia, educazione, scienza, salute, tecnologia e arte.
- **CNN/DailyMail** [17]: consiste in centinai di migliaia di articoli della CNN e del "The Daily Mail".

– I dataset più utilizzati in task di long-document-summarization sono:

- **Arxiv** [18]: raccoglie 1,5 milioni di articoli in pre-stampa, ospitando ricerche scientifiche di vari ambiti tra cui: fisica, matematica e informatica. Ognuna contiene feature di testo, figure, autori, citazioni, categorie e meta-dati.
- **Pubmed**: consiste in 19,717 pubblicazioni scientifiche prese dal database di Pubmed.
- **BillSum** [19]: consiste in una collezione di dati per la sintesi degli atti del Congresso degli Stati Uniti e dello stato della California.
- **BigPatent** [20]: raccoglie 1,3 milioni di record di documenti di brevetti statunitensi insieme a riepiloghi astrattivi scritti da persone.

1.4 Multi-document summarization

Il task di multi-document summarization (MDS) è un particolare task di single-document summarization. La principale differenza, risiede nella capacità di elaborare e relazionare più testi in un unico riassunto, mantenendo inalterato il contesto ed evitando ridondanze.

Il "target" è la generazione di un riassunto informativo Sum da una collezione di documenti D. D denota l'insieme dei documenti $\{d_i | i \in [1, N]\}$, e N è il numero di documenti totali. Ogni documento d_i consiste in M_i frasi $\{s_{i,j} | j \in [1, M_d]\}$.

La variabile $s_{i,j}$ è riferita alla j-esima frase nell'i-esimo documento.

Il task MDS evita l'utilizzo di tecniche estrattive per la similarità dei documenti da riassumere. MDS ricopre un vasto range di applicazioni, tra cui stampe e notiziari [21], in ambito medico [22] e tanti altri settori. Inoltre, viene utilizzato per la generazione di riassunti web come Wikipedia [23].

I dataset MDS comparati a quelli di SDS sono nettamente superiori di dimensionalità. I più famosi sono:

- **Oposum** [24]: è una collezione di recensioni multiple di sei domini di produzione appartenenti ad Amazon.
- **Duc&Tac**: sono dei riassunti di testi presi da competizioni a partire dal 2001 al 2007 con lo scopo di promuovere la ricerca nel campo della summarization.
- **WikiSum** [23]: è una collezione di document MDS astrattivi presi direttamente da Wikipedia.
- **Multi-News** [21]: è un large-scale dataset nel dominio di notizie e cronaca di giornale. Il dataset include 56216 articoli in coppia con i riassunti human-written. Gli autori hanno mostrato con confronti tra le loro versioni che il dataset possiede più disposizioni possibili nell'utilizzo.
- **Opinosis**: è un dataset di recensioni prese da 51 cluster collezionate da TripAdvisor, Amazon e Edmuns.
- **Rotten Tomatoes**: sono 3731 recensioni di film prese direttamente dal loro dominio.
- **HowSumm** [25]: è un large-scale dataset incentrato sulle query (qMDS). I dati sono composti principalmente dagli articoli del sito Web di WikiHow e dalle fonti citate al suo interno.

1.5 Transformer

L'architettura Transformer rappresenta un modello di rete neurale che ha rivoluzionato lo stato dell'arte in numerosi task, tra cui Natural Language Generation. Prima di addentrarci nella sua architettura vengono presentate le tecniche e i meccanismi che lo descrivono.

1.5.1 Il modello sequence-to-sequence

I *sequence-to-sequence* (Seq2Seq) sono modelli che hanno raggiunto dei traguardi considerevoli in moltissimi task, come machine translation, text summarization e composizioni di caption per immagini. Le reti seq2seq vennero introdotte per la prima volta da Google nel 2014 durante la conferenza annuale dei sistemi informativi di reti neurali (Sutskever et al. [26]) e successivamente da un gruppo di ricercatori dell'università di Montreal (Cho et al. [27]). Queste reti vengono utilizzate tutt'ora da Google Translate [28], famoso tool per la traduzione di frasi e paragrafi.

Ad alto livello la rete prende una serie di input, che possono essere parole o lettere e produce come output una successione di frasi (o item).

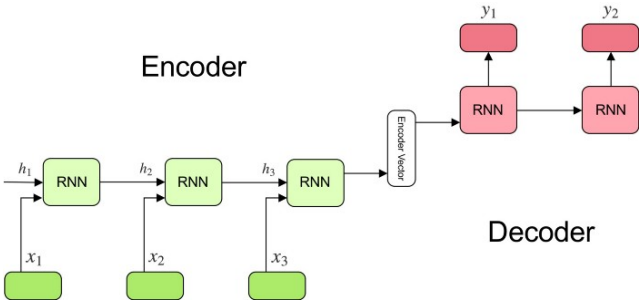


Figura 1.3: L'architettura del modello sequence-to-sequence. In **verde** l'encoder con x_i input e in **rosso** il decoder con le produzioni dei y_i risultati. L'encoder vector viene mostrato nell'elemento centrale, che viene successivamente passato al decoder.

Come mostrato nella Figura 1.3, è possibile individuare nell'architettura del modello sequence-to-sequence due componenti fondamentali:

- **Encoder** o **sequence-to vector network**: ha il compito di mappare le sequenze di input in una rappresentazione astratta creando un vettore n-dimensionale (denominato *encoder vector* o *embedding*). Durante la codifica del testo, aggiorna l'*hidden state* della rete con le nuove informazioni acquisite. L'ultimo hidden state corrisponde al contesto.
- **Decoder** o **vector-to-sequence network**: ha il compito di decodificare l'embedding e restituire una sequenza di testo. In pratica, esegue il procedimento inverso dell'encoder: riceve il contesto considerandolo come primo hidden state, lo elabora e produce come output una sequenza di parole.

Nel procedimento di encoding, ogni parola viene rappresentata tramite un vettore n-dimensionale. Il procedimento di conversione avviene tramite un algoritmo di **word embedding**. Il suo funzionamento, tramite un algoritmo, ha lo scopo di convertire il testo convertendolo in *token*. Esso, è un valore reale contenuto in una rappresentazione tabellare del vocabolario del modello. I token possono essere caratteri, parole o segni di punteggiatura. Il procedimento viene definito *tokenizzazione* è può essere fatto in maniera auto-regressiva, ossia token per token oppure può essere frammentato in parti più piccoli. L'idea è la possibilità di riuscire a rappresentare tutte le parole presenti nel vocabolario e distinguere correttamente quelle composte. Un esempio semplice è la parola *"notebook"* che può essere suddivisa in *"note"* e *"book"*. L'obiettivo è quello di mappare parole o frasi di un vocabolario—tramite diverse possibili strategie [29]— in vettori di numeri reali. Questo processo permette di rappresentare una elevata quantità di informazioni per ogni singola parola e può essere effettuato tramite modelli che vengono addestrati con la rete stessa oppure tramite reti pre-addestrate. È importante fissare durante la rappresentazione la lunghezza dell'embedding. Generalmente i valori più utilizzati sono 256, 512, 1024 [29].

I primi modelli seq2seq presentano grosse limitazioni. Le principali sono:

- la natura sequenziale delle reti neurali ricorrenti che impedisce la loro parallelizzazione e di conseguenza causa un calo a livello di performance;
- l'elevato peso agli ultimi elementi della sequenza che causa un'asimmetria nel processare l'embedding. Questo processo, infatti, va a "consumare" sempre l'ultimo elemento e viene definito *greedy search*.

Queste limitazioni hanno portato allo sviluppo di ottimizzazioni molto importanti, tra cui:

- **Attention** [30]: meccanismo che permette all'encoder di "focalizzarsi" maggiormente su determinate parole di input in modo che il decoder possa concentrarsi su di esse;
- **Bucketing**: tecnica che permette di raggruppare le frasi in base a dei range di lunghezza, in questo modo, far processare assieme quelle con dimensionalità simile da reti con lunghezza massima diversa, limitando così lo spreco di memoria;
- **Beam Search** [31]: la selezione di più parole viene rappresentata da un albero che varia in base alla quantità di "“beam” selezionati.

1.5.2 Meccanismo di Attention

Nella maggior parte dei task NLP è fondamentale riuscire a distinguere le dipendenze tra le parole del testo. Ottenere una buona predizione in un unico vettore può risultare complesso e in certi casi quasi impossibile. Un approccio di questo genere può funzionare solo se il modello possiede la capacità di *scegliere* e *focalizzare la propria attenzione* sulle parole più importanti e dal maggior valore informativo. Risulta necessario allineare le parole di input con quelle di output in maniera intelligente, definito *word alignment*. A tale scopo nasce un nuovo layer che prende il nome di **Attention**. L'obiettivo dell'attention è di allineare correttamente le parole calcolando per ognuna un punteggio corrispettivo (o *score*).

Un procedimento esemplificativo nel calcolo dell'attention può essere suddiviso nei seguenti passaggi.

- Per prima cosa, vengono definiti gli hidden state per l'encoder e decoder. Successivamente vengono calcolati i pesi facendo i prodotti scalari tra gli hidden state dei due componenti. I valori più alti saranno maggiormente influenzati alle previsioni successive.
- Il risultato viene passato ad una funzione *softmax*, che normalizza i punteggi in modo che siano esprimibili sotto forma di probabilità, ossia valori compresi fa 0 e 1.
- Ogni hidden state dell'encoder viene moltiplicato per il peso dell'attention—passato attraverso la softmax—ottenendo il vettore di allineamento. In questo modo si preserva il valore di ogni parola, diminuendo contemporaneamente l'importanza di quelle irrilevanti.
- Infine, si sommano i valori dei vettori pesati ottenendo come risultato il vettore di contesto.

1.5.3 Elementi dell'Attention

Il calcolo dell'attention si basa sulla computazione di tre diversi vettori: `query`, `key` e `value`. Per associare meglio il significato dei tre elementi consideriamo un esempio pratico che viene spesso proposto.

Quando si effettua una ricerca su un motore di ricerca, il motore mappa la richiesta, `query`, su un insieme di chiavi, `key`, associate alla possibili pagine candidate nel database. Le pagine che verranno mostrate saranno quelle con un punteggio più alto (`value`) rispetto alla richiesta (`query`) .

Quindi, nel modello, `query` rappresenta la richiesta, alla quale si associa una `key` e si assegna ad ogni posizione un `value`. Il più alto fra questi, indica il punto dove è più semplice trovare una corrispondenza con la richiesta `query` proposta.

Una delle formule più diffuse nel calcolo dell'Attention è la **Dot-Product Attention** [32]. Il suo funzionamento può essere suddiviso in tre step:

1. Nell'attention layer viene calcolato il prodotto scalare (*Dot-Product*) fra `query` e `key`.
2. Successivamente, si effettua il calcolo della softmax ottenendo score positivi tra $[0, 1]$, ossia una probabilità.
3. Infine, si moltiplicano le probabilità trovate con i `value`. Il risultato in output risulterà pesato per ogni parola o in combinazione ad altre.

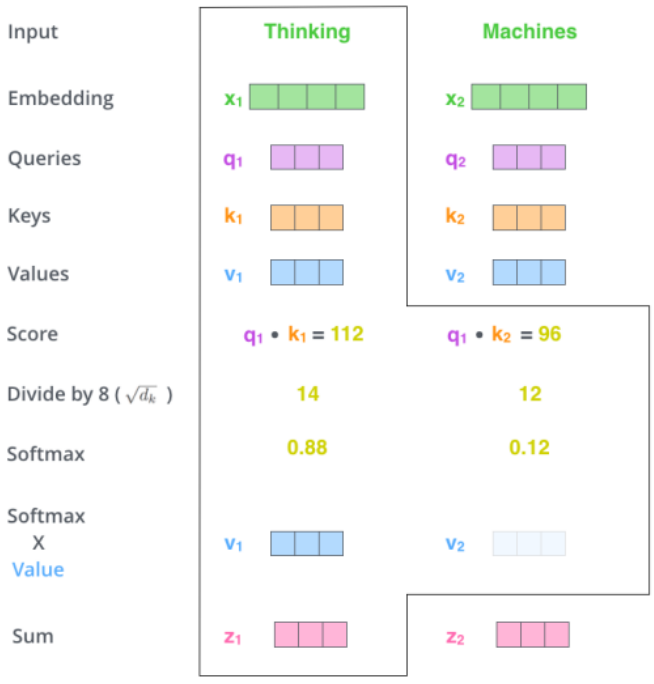


Figura 1.4: Rappresentazione del meccanismo di attention in uno spazio vettoriale in un task di machine translation. La sequenza di operazioni parte dall'input "Thinking", mostrando la creazione dell'embedding, delle `query`, `key` e `value`. Le successive operazioni mostrano la relazione con la parola "Machines".

Fonte: *https://jalammar.github.io/illustrated-transformer/*

Per comprendere meglio il meccanismo di attention facciamo un esempio sulla base del procedimento proposto.
Prendiamo in input *"Ciao Mondo!"*. La prima cosa da fare è il calcolo del punteggio rispetto a tutte le altre parole. Questo punteggio ci permetterà di capire quanto la parola sia in relazione a tutte le altre. Per calcolarlo viene fatto in primis il prodotto `query` $\cdot$ `key` della parola di cui vogliamo assegnare un punteggio. Successivamente, viene applicata la funzione di softmax alla matrice `score`, in modo da ottenere delle probabilità. Infine, vengono sommate tutte le parole con i loro relativi pesi producendo il valore di attention per la parola calcolata.

La *Dot-Product Attention* appena descritta, può essere scritta in forma matriciale e riassunta nella Formula 1.2:

$$Attention(Q, K, V) = softmax(QK^T) \cdot V \tag{1.2}$$

Tale che:
`query` $= Q : [L_Q, D_K]$
`key` $= K : [L_K, D_k]$
`value` $= V : [L_V, D_v]$
$L =$ lunghezza di query, key e value
$D =$ dimensione arbitraria di `query`, `key` e `value`

La Formula 1.2 presenta dei problemi. All'aumentare della dimensionalità di Q e K cambia la distribuzione tendendo sempre a valori più vicini a zero oppure ad uno. Per prevenire il problema bisogna effettuare una normalizzazione, dividendo per $\sqrt{d_k}$, ossia la radice quadrata della lunghezza di `query` $\cdot$ `key`:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V \tag{1.3}$$

La normalizzazione effettuata nella Formula 1.5 previene che il gradiente della funzione risulti estremamente piccolo o estremamente grande. Viene effettuato il prodotto fra matrici e successivamente diviso per la radice quadrata della lunghezza `query` $\cdot$ `key` denominato **Scaled Dot-Product Attention**.

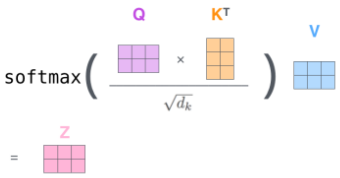


Figura 1.5: Funzione softmax in rappresentazione matriciale. Viene mostrato il prodotto delle matrici di query e key diviso per lo scaled-dot product attention e moltiplicato per value con il conseguente risultato Z.
Fonte: *https://jalammar.github.io/illustrated-transformer/*

Il calcolo dell'attention sotto forma matriciale permette l'utilizzo di *acceleratori hardware*, in questo modo si possono sfruttare Graphics Processor Unit (GPU) e Tensor Processor Unit (TPU). Questo permette la parallelizzazione delle istruzioni di prodotti matriciali, aumentando significativamente la velocità di calcolo.

1.5.4 Meccanismi di Attention

Nei paragrafi precedenti viene analizzato l'Attention in generale e applicato ad una sua variante denominata Encoder-Decoder Attention. Nel tempo sono state introdotte molte altre varianti rispetto alla versione originale.Di seguito, sono menzionate le principali.

- **Encoder-Decoder Attention**: esposto nei paragrafi precendenti. Dove il decoder considera Q e K provenienti da frasi differenti.
- **Casual Self Attention**: è una tipologia di attention utilizzata nei task di text summarization dove le predizioni delle frasi dipendono da quelle già predette. Quindi Q e K provengono dalla stessa frase, per questo prendono il nome di **Self-Attention**;
- **Bidirectional Self Attention**: è una tipologia nella quale le parole possono considerare quelle già elaborate e quelle ancora da eseguire. Quindi Q e K considerano le frasi precedenti e quelle successive.

1.5.5 L'architettura Transformer

Nel Giugno 2017, Google ha pubblicato un paper dal titolo *Attention Is All You Need*, Vaswani et al. [2] dove introduce il meccanismo di **self-attention** che riproduce la base-line per ogni modello Transformer-based allo stato dell'arte.

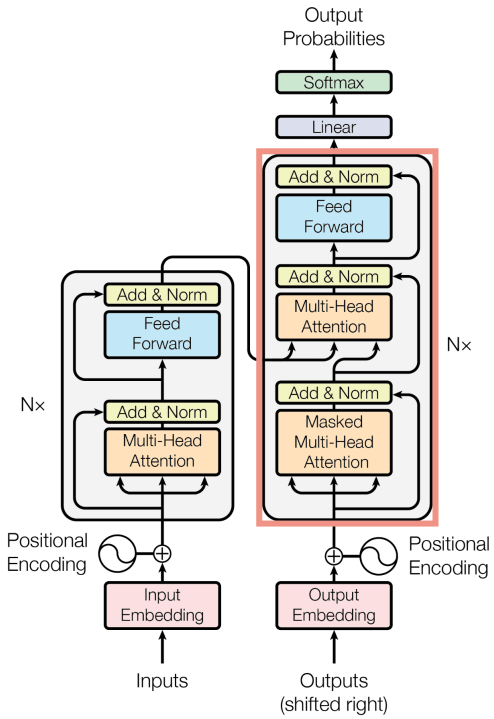


Figura 1.6: Scomposizione in layer e sublayer dell'architettura Transformer [2]. A sinistra l'encoder e destra il decoder (evidenziato in **arancione**).

L'architettura Transformer si basa sul meccanismo di encoding-decoding.

- L'**encoder** ha il compito di mappare le sequenze di input in una rappresentazione astratta continua creando un embedding in un vettore n-dimensionale. È composto da uno stack 6 layer identici. Ogni layer ha due sub-layer. Il primo è il *multi-head self-attention mechanism*, mentre il secondo è un *position-wise* completamente connesso alla rete feed-forward.

- Il **decoder** ha il compito di decodificare l'embedding e restituire una sequenza di testo. È anch'esso composto da uno stack di 6 layer identici. Oltre ai due sub-layer per ogni livello di encoder, viene inserito un terzo sub-layer che esegue *multi-head attention* sopra l'output dello stack di encoding.

Per rendere ancora meglio l'idea, prendiamo la Figure 1.7 che mostra, in un task di machine translation i layer e sublayer raggruppati ottenendo una versione più "compatta". In questo modo si comprende perfettamente il ciclo di esecuzione delle due parti.

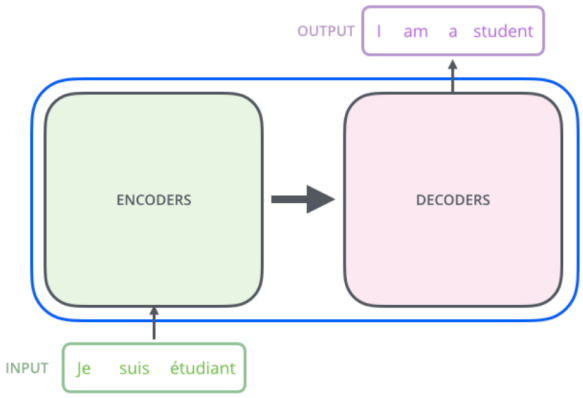


Figura 1.7: Rappresentazione compatta dell'architettura Transformer in un task di text translation. Presa la **parola** in input, viene generato l'embedding e passato al decoder che restituirà la **traduzione** generata.
Fonte: *https://jalammar.github.io/illustrated-transformer/*

1.5.6 Multi-Head Attention

Il **Multi-Head Attention** è un meccanismo aggiuntivo introdotto da Transformer che sostituisce il layer ricorrente delle seq2seq ed emula la ricorrenza delle sequenze. Il principio di base è quello di aggiungere un insieme di layer di self-attention in parallelo in cima a meccanismo di attention già presente, denominato *head*.

In pratica, dato lo stesso insieme di `query`, `key` e `value`, si richiede che il risultato contenga informazioni dei diversi componenti spaziali e aiuti il modello a concentrasi su posizioni differenti (Figura 1.8). Pertanto, può essere utile consentire al meccanismo di attention di utilizzare congiuntamente diversi sottospazi di rappresentazione per `query`, `key` e `value`. Ogni head è inizializzato in maniera casuale e viene utilizzato per proiettate gli input in diverse rappresentazioni di sotto-sequenze spaziali. Nell'architettura il numero di head totali è pari a 8; ne conseguono 8 differenti matrici di output. Dato che l'output deve essere una singola matrice, vengono concatenate assieme per poi essere moltiplicate con una matrice addestrata con l'intero modello.

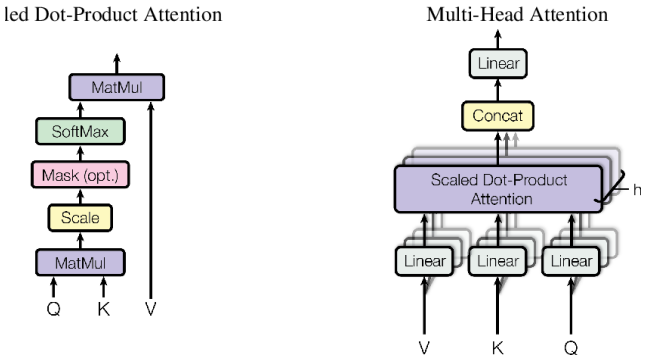


Figura 1.8: Confronto tra il meccanismo di dot-product attention con il layer multi-head attention. Il flusso input-output viene eseguita top-down, ossia dal basso verso l'alto. [8]

Fonte: *https://jalammar.github.io/illustrated-transformer/*

1.5.7 Positional Embedding

Il modello Transformer non presenta ricorrenza e convulsione. Per colmare questa mancanza è stato necessario trovare un metodo che permettesse la possibilità di aggiungere informazioni riguardo alle posizione relative o assolute all'interno dell'embedding. Una soluzione è stata quella di creare un vettore che venga sommato all'embedding di un token. Il vettore risultante rappresenta l'informazione spaziale e condividendo la stessa dimensione dell'input. Il vettore prende il nome di **Positional Embedding** (PE) [33]. Per calcolare i valori nel vettore si possono utilizzare diverse funzioni lineari, sia addestrabili che fisse. Nell'architettura Transformer originaria viene utilizzata una funzione che associa la funzione seno (1.5)– se lo step è pari – e coseno (1.4) – se lo step è dispari, definita come segue:

$$PE(pos, 2i+1) = cos(pos/(10000^{2*i}/d_{model})) \tag{1.4}$$

$$PE(pos, 2i) = sin(pos/(10000^{2*i}/d_{model})) \tag{1.5}$$

In questo modo si riesce ad ottenere sia la posizione assoluta dell'*embedding* che quella relativa. Come si può osservare in Figura 1.9, per ogni posizione si ottiene un *positional embedding* univoco.

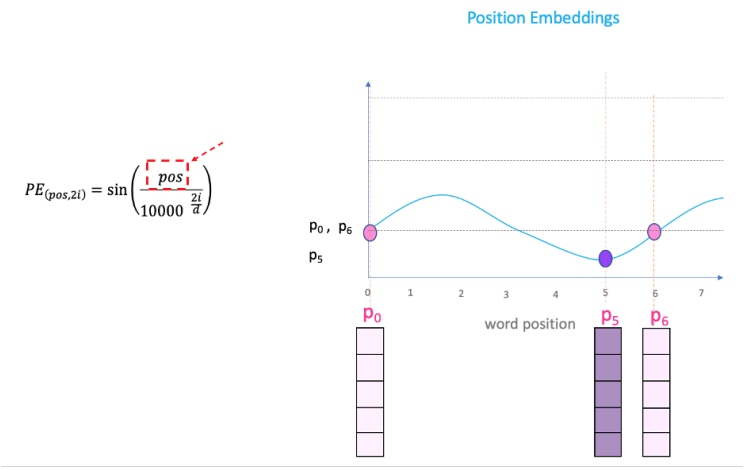


Figura 1.9: Matrice del *positional embedding*. A sinistra la formula la formula seno per indice pari. A destra un grafico che mostra le possibili associazioni alle posizioni associate al proprio posiziona embedding P_i.

1.6 Modelli pre-addestrati

Ad oggi, molti modelli basano le loro fondamenta sull'architettura Transformer. È possibile pre-addestrare modelli neurali per i task NLP. Un modello di questo genere può apprendere rappresentazioni in linguaggio universale ed evitare l'addestramento da zero [34].

Molti ricercatori e studiosi rilasciano costantemente nuovi modelli pre-addestrati sempre più complessi e performanti per l'abstractive text summarization. Attualmente, i principali sono:

- **BART** [35]: un modello proposto per migliorare i risultati in task di natura generativa, che possiede alcuni tra i risultati migliori nel campo dell'Abstractive Text Summarization.
- **PEGASUS** [36]: un modello Transformer encoder-decoder su enormi dati di testo. A differenza di altre architetture utilizza una funzione di costo più simile possibile a quella del down-stream task (i.e, task di fine-tuning) ponendo particolare enfasi su di esso.
- **BERT** [37]: un modello con il meccanismo del fine-tuning, progettato per pre-addestrare rappresentazioni deep bidirezionali. Sono presenti due step di addestramento: il pre-training e il fine-tuning.
- **T5** [38]: un modello "text-to-text" pre-addestrato su diversi task NLP individuabili tramite un prefisso (e.g., "summarize:" nel caso della Text

Summarization). Il suo punto di forza risiede nella maggiore varietà dei down-stream task.

1.7 Metriche di valutazione

Dati tutti i possibili metodi generativi di testo, per poterne valutare la bontà viene utilizzata una metrica valutativa. La metrica permette di rilevare il possibile potenziale di ciascuno confrontando i risultati ottenuti. In generale, la miglior valutazione per il testo generato rimane quella umana, anche detta "gold-standard". I criteri utilizzati come strumento comparativo di riassunti sono:

- **Rilevanza**: la consistenza del testo rispetto al documento sorgente.
- **Informatività**: l'individuazione e il focus sulle parola chiave del testo sorgente.
- **Fattualità**: il confronto dal documento di origine delle frasi che rendendole affermative, valutano presenza di allucinazioni nel testo.
- **copertura semantica**: il confronto della quantità di unità semantiche rispettate nel testo rispetto ai suoi riferimenti.
- **Adeguatezza**: il confronto fra possibili distorsioni tra il testo di output e quello di input.

I criteri valutativi vengono applicati in task di modelli neurali generando una valutazione automatica.

Le metriche di valutazione possono essere distinte in:

- **Matching**: la metrica esegue un matching esatto tra token di riferimento e token generato. Questo permette di cogliere aspetti importanti, come la rilevanza e l'adeguatezza. Al contrario ha difficoltà a rilevare la fattualità del riassunto completo poiché il match difficilmente riesce a distinguere rumori o allucinazioni nel testo.
- **Regressione**: la metrica utilizza modelli supervisionati per ottenere la fedeltà massima nella mappatura fra dati di input e quelli attesi in output. A conclusione del training, viene introdotto un layer regressivo con lo scopo di predire —con dei dati visti per la prima volta— il miglior risultato. L'obiettivo è quello di emulare il giudizio umano.

- **Ranking**: la metrica apprende una funzione che assegna un ranking a seconda di un criterio di punteggio basato sulle ipotesi migliori rispetto a quelle peggiori.

- **Generazione**: la metrica basa la sua valutazione su un costante confronto del testo in maniera auto regressiva durante la generazione della predizione.

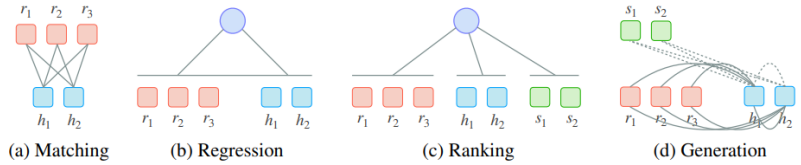


Figura 1.10: Raffigurazione delle metriche valutative in task NLP [9].

1.8 Green AI

Ad oggi, i calcoli necessari per le ricerche nel campo di Deep Learning hanno una crescita che raddoppia —mese dopo mese— con un aumento stimato di 300.000 volte dal 2012 al 2018 [39]. Questi calcoli producono un'enorme quantità di carbonio che porta ad avere un grosso impatto sull'ambiente. Nasce nell'istituto di Allen un nuovo trend, che prende il nome di Green AI (Green Artificial Intelligence), il quale cerca di trovare delle soluzioni per ridurre l'impatto energetico delle attuali tecnologie. Questa tendenza solleva molte interessanti direzioni di ricerca che aiutano a superare le sfide dell'inclusività della Red AI. Con *"Red"* si intende la propensione a guardare solamente il risultato finale, adoperando ampi dati e modelli complessi per condurre un'elevata quantità di esperimenti trascurando il tempo di computazione richiesto e di conseguenza l'emissione di CO_2 rilasciata.

Uno dei più grandi sostenitori del Green AI è Google, che intende alimentare i propri data center attraverso fonti rinnovabili. Una delle principali cause di consumo energetico dei data center è causato dalla necessità di raffreddare i componenti. Per affrontare il problema, attualmente i ricercatori utilizzano reti neurali DeepMind per sfruttare al massimo l'energia eolica.

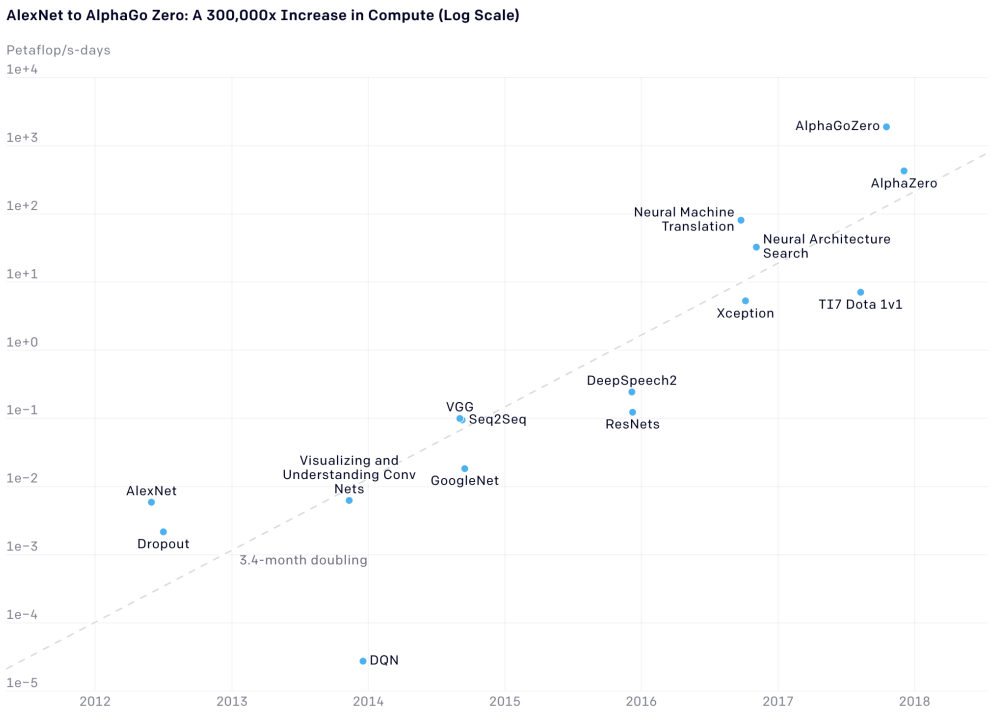


Figura 1.11: Il grafico mostra l'elevata crescita di calcolo utilizzata per addestrare i modelli. Aumentata di circa 300.000 volte in 6 anni. Nel grafico (in **blu**) vediamo il posizionamento dei modelli in relazione alla quantità di calcolo (petaflop al giorno) in anni.

Fonte: *Ai and Compute [39]*

Attraverso l'introduzione di misure preventive è possibile ridurre le emissioni di CO_2 e aumentare l'efficienza dei sistemi. Questo potrebbe portare ad una graduale scomparsa della Red AI. Esistono algoritmi di ottimizzazione degli iper-parametri che possono ridurre la spesa computazionale richiesta per raggiungere un determinato livello di prestazione (Bergstra et al. [40]), (Li et al. [41]) e semplici accorgimenti che possono ridurre i costi computazionali come l'interruzione anticipata durante un addestramento di modelli o generazione di testo quando non si hanno ulteriori guadagni.

1.9 Strategie di decoding

Nel seguente paragrafo vengono trattate tutte le possibili strategie adottate per la generazione di testo, spiegando il loro funzionamento in dettaglio. In questo modo, si potrà comprendere meglio il divario che presenta ogni ogni strategia di decoding.

1.9.1 Greedy Search

L'algoritmo **Greedy Search** (i.e. ricerca avida) è ampiamente adottato in diversi ambiti. Spesso, viene utilizzato come primo approccio alla modellazione di reti neurali legali alla text generation e machine translation per la sua notevole velocità di inferenza.

L'algoritmo genera una singola sequenza da sinistra a destra, scegliendo il token con probabilità più alta come parola successiva. Quindi, è possibile visualizzarlo come una strategia che mangia in maniera "avida"—dall'inglese greedy— la parola x_i più probabile dalla distribuzione di probabilità in maniera auto regressiva.

L'output $\hat{x} = \{\hat{x}_1, \hat{x}_2, ...\hat{x}_n\}$ può essere rappresentato nella Formula 1.6:

$$\arg\max_{x} \left(P(x|x_{1:i-1})\right) \tag{1.6}$$

Nonostante la bassa complessità computazionale dell'algoritmo, le sequenze selezionate da quest'ultimo possono essere tutt'altro che ottimali sotto la distribuzione complessiva data dal modello. Basti pensare che non abbiamo la certezza che la parola più probabile sia necessariamente la parola giusta. Di conseguenza questa strategia viene utilizza come primo approccio e testing per la generazione di testo in modo tale da avere risultati tangibili in tempi ridotti.

Un esempio pratico e dimostrativo dell'algoritmo è dato dalla Figura 1.14.

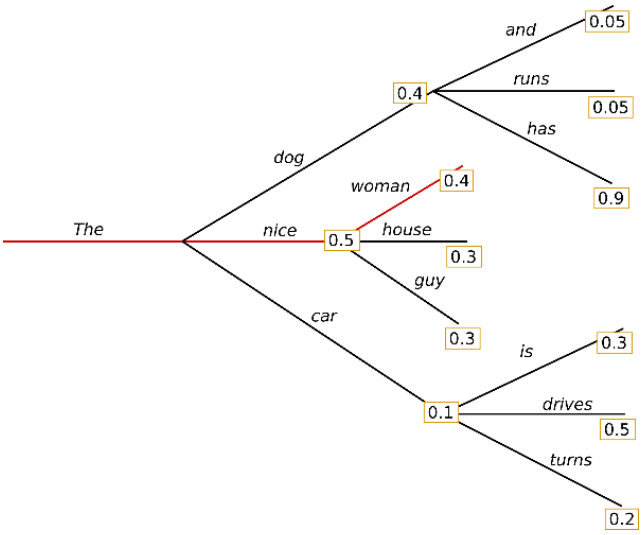


Figura 1.12: Partendo dalla parola "L'illustrazione mostra l'algoritmo greedy search, evidenziando in **rosso** le scelte effettuate ad ogni step. *The*", l'algoritmo sceglie in maniera "avida" la parola successiva con probabilità più alta. Come seconda parola viene scelta "*nice*" e infine "*woman*". ("*The*", "*nice*", "*woman*") ha una probabilità totale di $0.5 \times 0.4 = 0.20$.

Fonte: *https://huggingface.co/blog/how-to-generate*

1.9.2 Beam Search

La ricerca Greedy presenta diverse limitazioni nella scelta del token. La strategia **Beam Search** nasce come estensione euristica della ricerca "*best-fist*" o avida di Greedy. La principale differenza è la capacità di selezionare dei *cammini* (o ipotesi) multipli ad ogni step. Quindi è in grado di scavare in profondità per ricercare la frase con probabilità totale maggiore. L'obiettivo quindi, è ricercare la strada x migliore tra le y_i alternative possibili. Possiamo rappresentarlo nella formula 1.7:

$$\arg\max_{y} \prod_{i \leq |y|}^{n} (P(y_i|x, y_{<i})) \tag{1.7}$$

Il numero di alternative, possiamo esprimerlo sotto forma di iper-parametro che prende il nome di `beam size` o *beam width*. Se il suo valore ha dimensione uguale ad 1, allora ci riduciamo ad una ricerca Greedy.

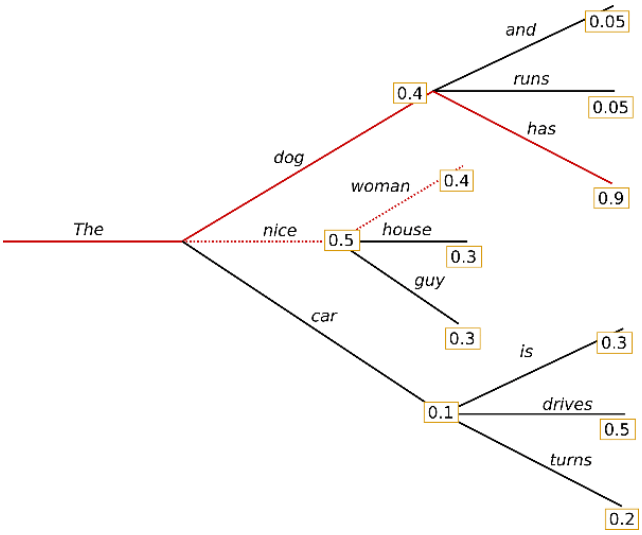
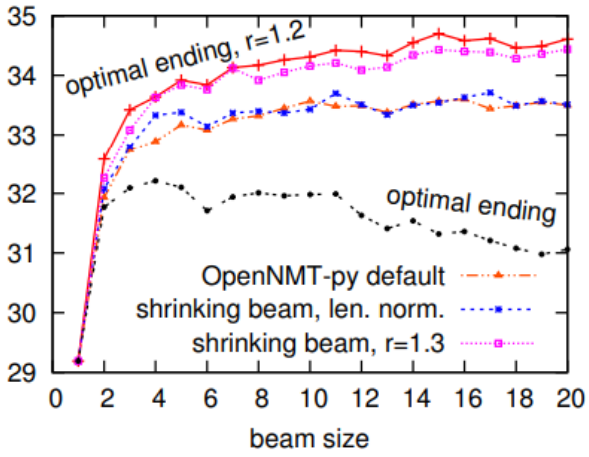


Figura 1.13: L'illustrazione mostra l'algoritmo beam search con parametro `beam_size` = 2, evidenziando in **rosso** le scelte effettuate ad ogni step. Nello step iniziale, oltre a tenere conto della migliore ipotesi ("The", "nice"), l'algoritmo tiene traccia della seconda ipotesi migliore ("The", "dog"). Seguendo questo percorso, nello step successivo, trova le sequenze di parole ("The", "dog", "has") con una probabilità di 0.36 che risulta più alta di ("The", "nice", "woman"), ossia con probabilità pari a 0.2.

Fonte: *https://huggingface.co/blog/how-to-generate*

La scelta della dimensionalità del parametro ha avviato tantissimi studi e ricerche alla scoperta del numero di beam ottimale [42]. Uno studio comprensivo sul beam search del 2018 nell'Università di Oregon Huang et al. [43] ha mostrato tramite il confronto su un ampio spettro di task NLP con la metrica BLEU interessanti proprietà:

- con n beam molto piccoli l'algoritmo tende a soffrire di una mancanza di diversità tra le n ipotesi circostanti;
- con $n \geq 3$ la generazione genera del testo che secondo i criteri della ricerca può essere definito "accettabile";
- con n compreso tra $10 \leq n \leq 17$ si ottengono dei risultati approssimativamente "ideali";
- con $n \geq 17$ abbiamo un lieve aumento, ma costante della qualità di generazione del testo.



(a) BLEU vs. beam size

Figura 1.14: Il grafico riporta il punteggio BLEU rispetto alla dimensione del beam tramite sui vari task NLG.

La complessità del Beam Search con il crescere del `beam_size` ha una crescita logaritmica. Questo lo rende ottimo quando la lunghezza della generazione desiderata è prevedibile. Al contrario, è sconsigliato l'utilizzo di questa strategia con generazione a tempo indeterminato nella quale la lunghezza dell'output desiderata può variare notevolmente, ad esempio nella generazione di dialoghi e storie.

1.9.3 Diverse Beam Search

Una variante del Beam Search è il **Diverse Beam Search** (DBS). Alcuni studi tra cui Vijayakumar et al. [44] e Vijayakumar et al. [45], hanno mostrato come in task di machine translation e computer vision il Diverse Beam Search ha ottenuto risultati ottimi rispetto al classico Beam Search [45]. L'algoritmo è stato proposto nel 2018 dalla Virginia Tech Department con la pubblicazione: "*Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models*" [44]. L'idea alla base della nascita dell'algoritmo risiede nella scarsa *diversità* del Beam Search che lo porta:

- a discostarsi poco dalle ipotesi possibili, rendendo l'algoritmo spesso ripetitivo e computazionalmente dispendioso senza alcun significativo aumento delle prestazioni;
- nel caso di beam grandi, oltre ad avere gli n output desiderati, si hanno anche delle sequenze "garbage" che verranno scartate;

- ad avere notevoli problemi nel *rilevare l'ambiguità*. Ad esempio, ci sono diversi modi per descrivere un'immagine o rispondere correttamente in una conversazione ed è importante cogliere questa ambiguità trovando diverse ipotesi plausibili.

Ad alto livello, DBS elabora diverse alternative dividendo il bilancio dei beam in gruppi e forzando la *diversità* tra di essi. In questo modo si ha un trade-off tra l'esplorazione dello spazio di generazione (più gruppi con meno beam) e la ricerca del miglior rendimento del massimo locale (meno gruppi con più beam).

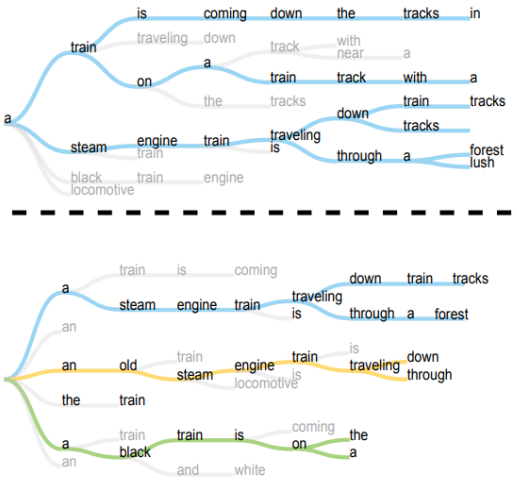


Figura 1.15: Il grafico mostra un confronto tra beam search (figura in alto) e diverse beam search (figura in basso). Notiamo subito che i percorsi del beam search non sono discostanti molto e che arrivati ad una certa profondità le soluzioni ottenute tendono ad essere simili tra loro. Diverse beam search mostra (con una differenziazione di colori) che la diversità ha un impatto elevato sui percorsi scelti e di conseguenza sui risultati ottenuti.

Fonte: *https://huggingface.co/blog/how-to-generate*

La definizione della funzione di diversità la possiamo esprimere con $\Delta(y_{|t|}, Y^g_{|t|})$ che permette di misurare la dissimilarità tra la sequenza $y_{|t|}$ e il gruppo $Y^g_{|t|}$. Mentre Δ può assumere diverse forme, per semplicità definiamo un'ampia classe che si decompone attraverso i beam all'interno di un gruppo. Scriviamo la formula generale 1.10:

$$\Delta(y_{|t|}\,,\, Y^g_{|t|}) = \sum_{b=1}^{B'} \delta(y_{|t|}, y^g_{b\,,\,|t|}) \tag{1.8}$$

Tale che:

- $\sum_{b=1}^{B'} \delta(y_{|t|}, y^g_{b,|t|})$ è la somma di tutti i gruppi di beam precedenti.
- $\delta(y_{|t|}, y^g_{b,|t|})$ rappresenta la funzione di dissimilarità.

Nell'ambito del text summarization è stato dimostrato che il Diverse Beam Search non solo riduce l'estrattività, ma migliora il punteggio ROUGE complessivo di un significativo 2% [46]. Quindi, la scelta di DBS rispetto a BS non porta ad un aumento di qualità assicurata rispetto alle generazioni di beam search. Sicuramente la variante porta ad avere più *diversità*. In conclusione la scelta di quale strategia adottare sarà strettamente correlata al tipo di documento da riassumere e dal target desiderato.

1.9.4 Temperature sampling

Il **Sampling** è un metodo non deterministico che seleziona casualmente una nuova parola (o token) da una distribuzione di probabilità, condizionata da parole generate in precedenza in maniera auto-regressiva. I metodi basati sul Sampling (o campionamento) possono produrre risultati inaspettati e più innovativi rispetto al Beam Search.

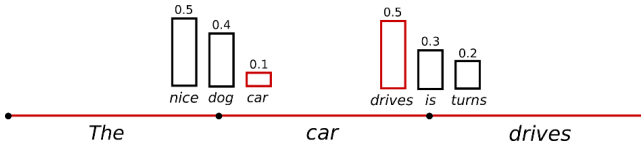


Figura 1.16: Il sampling ad ogni step estrae un token casuale dalla distribuzione di parole, dove ognuna ha associato il suo peso. In **rosso** la parola campionata. La parola *"macchina"* è campionata dalla distribuzione di probabilità condizionata $P(w \mid \text{"car"})$, seguito dal campionamento *"drivers"* ottenuta da $P(w \mid \text{"car"}, \text{"drivers"})$.

Fonte: *https://huggingface.co/blog/how-to-generate*

Formalmente, il Sampling sceglie casualmente la parola successiva w_t secondo la sua distribuzione di probabilità condizionata:

$$w_t \sim P(w|w_1 : w_{1:t-1}) \tag{1.9}$$

Un altro approccio comune alla generazione basato sul sampling consiste nel modellare una distribuzione di probabilità attraverso la **temperatura** t. L'utilizzo del semplice sampling non sempre e efficiente e può generare parole estremamente improbabili che possono risultare fuori dal contesto. Pertanto, viene comunemente usato in combinazione con il parametro temperatura `t`.

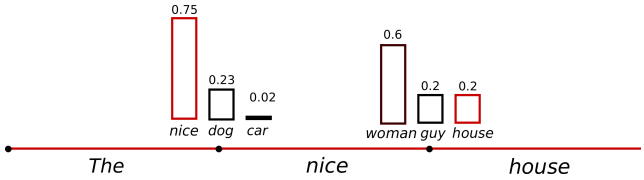


Figura 1.17: Il temperature sampling ad ogni step estrae un token casuale dalla distribuzione di parole *modificata*. In rosso la parola campionata. Con `t` = 0.7, la distribuzione viene 'schiacciata', abbassando i pesi dei singoli token. Si vede subito a sinistra, che il campionamento sopprime la parola *"cars"* portando ad una scelta "condizionata".

Fonte: *https://huggingface.co/blog/how-to-generate*

La funzione *softmax* viene riformulata rispetto a quella base, vista nel paragrafo 1.5.3 come:

$$P(x = V_l|x_{1:i-1}) = \frac{exp(u_l/t)}{\sum_{l^i} exp(u_l^i/t)} \tag{1.10}$$

Tale che $t \in [0, 1)$ distorce la distribuzione verso eventi ad alta probabilità e implicitamente abbassa la distribuzione dei token distanti. Questo permette di aumentare la probabilità di parole con probabilità elevate e diminuirla per le parole con probabilità trascurabile.

Nel caso particolare in cui `t` = 1, ci riconduciamo all'utilizzo del Sampling standard. È stato dimostrato che il Temperature Sampling ha ottenuto ottimi risultati nella traduzione inversa del testo [47].

1.9.5 Nucleus (Top-P sampling)

Il sampling **Nucleus** noto come campionamento *Top-T*, è stato introdotto nel paper *"the curios case of neural text degeneration"* [4] dimostrando per le strategie di decoding che ottimizzano l'output con alta probabilità—come il beam search—portano al testo a degenerare anche con modelli all'avanguardia come GPT-2-Large [48]. Nel dettaglio il problema deriva dall'insieme di decine di migliaia di token candidati con probabilità relativamente bassa che vengono sovra-rappresentati. La chiave risolutiva è l'utilizzo di una nuova strategia denominata Nucleus.

La scelta della generazione di testo campiona le parole successive dalle prime `k` scelte più probabili, invece di mirare a decodificare il testo massimizzando le probabilità. Il campionamento Nucleus (i.e. top-p sampling) può facilmente abbinare la perplessità di riferimento regolando il valore di `p`, evitando l'incoe-

renza causata dall'impostazione `k` sufficientemente alto da corrispondere alle statistiche distributive.

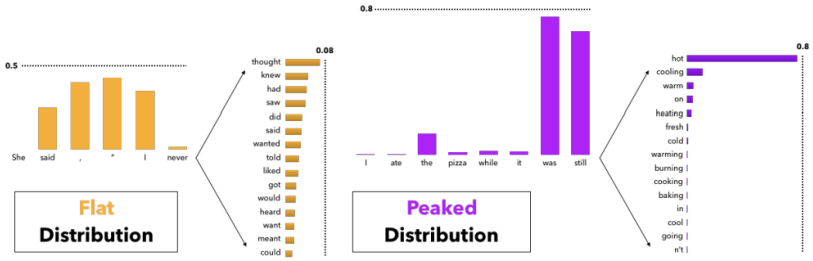


Figura 1.18: La figura mostra le differenze tra una massa di probabilità lineare (in **arancione**) e una massa di probabilità compressa tra un insieme `k` (in **viola**). Quella lineare ha una finestra di token più alta rispetto a quella a compressa, che concentra la maggior parte della massa di probabilità in alcuni token.

Fonte: *The curios case of text degeneration [4]*

L'idea chiave è usare la forma della distribuzione di probabilità per determinare l'insieme di token da cui campionare. Data una distribuzione $P(x|x_{1:i-1})$, definiamo `top-p` con il suo vocabolario $V^p \in V$ il più piccolo insieme possibile:

$$\sum_{x \in V^p} P(x|x_{1:i-1}) \geq p \tag{1.11}$$

Sia $p' = \sum_{x \in V^p} P(x|x_{1:i-1})$. La distribuzione originale viene ridimensionata a una nuova distribuzione, da quale viene campionata la parola successiva:

$$P'(x|x_{1:i-1}) = \begin{cases} P(x|x_{1:i-1})/p', & \text{se } x \in V^p \\ 0, & \textit{altrimenti} \end{cases} \tag{1.12}$$

In pratica questo significa selezionare i token con la probabilità più alta la cui massa di probabilità cumulativa supera la soglia prescelta `p`. La dimensione del set di campionamento verrà regolata dinamicamente in base alla forma della distribuzione di probabilità ad ogni step temporale. Quindi, per valori elevati di `p`, il *nucleo*, si occupa di selezionare il sottoinsieme del vocabolario della massa di probabilità.

L'analisi qualitativa mostra che Nucleus soddisfa tutti i criteri di distribuzione per le generazioni desiderabili. Nonostante tutto, presenta alcune limitazioni, mostrando l'incapacità di distinguere correttamente in maniera assoluta le

parole del documento sorgente. Il suo punto di forza, rimane la possibilità di catturare efficacemente la regione di fiducia dei modelli linguistici.

1.9.6 Top-k sampling

Il campionamento completamente casuale può introdurre molta diversità nelle parole scelte deteriorando la generazione. La restrizione del campionamento in una finestra di `k` più probabili candidati riducono il rischio di estrazioni di campioni a bassa probabilità.

Nel 2018 viene introdotto nel paper Fan et al. [49] una nuova strategia con il nome di **Top-K Sampling** diventando una procedura di campionamento alternativa molto popolare.

L'utilizzo di `top-k`, porta quindi a filtrare in una finestra di parole campionando in modo casuale tra le prime `k` parole selezionate.

Formalmente, data una distribuzione $P(x = V_l|x_{1:i-1})$, definiamo il vocabolario top-k $V^k \in V$ come la dimensione dell'insieme k che massimizza $\sum_{x \in V^k} P(x = V_l|x_{1:i-1})$. Prendiamo $p' = \sum_{x \in V^k} P(x = V_l|x_{1:i-1})$. La distribuzione viene ridimensionata come l'equazione 1.12 e il campionamento avviene basato su tale distribuzione.

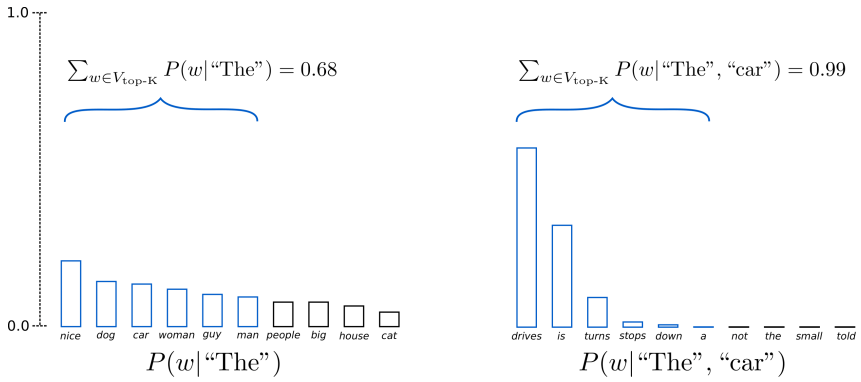


Figura 1.19: I due istogrammi mostrano il funzionamento di top-k sampling mostrando i primi due step iniziali del procedimento. L'ampiezza dell'istogramma è il peso del token mentre la finestra (o pool) di parole selezionate sono evidenziate in **blu**. Con `k` = 6, nel primo step viene viene limitato il campionamento in un pool di 6 token riducendo $\frac{1}{3}$ della distribuzione. Nel secondo step la finestra include quasi tutta la massa di probabilità. Questo porta ad eliminare (nel secondo step) con successo candidati inadeguati come: *“not”*, *“the”*, *“small”*, *“told”*

. Fonte: *https://huggingface.co/blog/how-to-generate*

La modifica della temperatura, in alcuni contesti rende la selezione della parola successiva 'piatta' su decine o centinaia di opzioni (i.e. sostantivi o verbi), mentre in altri contesti la maggior parte della massa di probabilità è concentrata in un piccolo insieme di token. Quindi se `k` è piccolo c'è il rischio di generare testo parzialmente blando o generico, mentre se `k` è grande la finestra `top-k` lo farà includere come candidato 'inappropriato' aumentando la probabilità di essere inseriti nel campione. Deduciamo che l'uso di una costante `k` non è sempre ottimale, ma come altre strategie, bisogna sceglierlo con criterio in base al contesto e al testo di riferimento.

In task di text summarization questa caratteristica porta 'incertezza', spingendo il suo impiego in riassunti che richiedono di presentare una nota di creativa.

1.10 Repetition penalty

Il problema della generazione di testo ripetitivo influisce negativamente su tutti i metodi proposti. Beam Search in particolare, soffre maggiormente di questo problema. Gli autori del paper CTRL Keskar et al. [50] hanno proposto un nuovo metodo di campionamento che *penalizza* le ripetizioni aggiornando i punteggi dei token precedentemente generati. Questo metodo è stato aggiungo come iper-parametro in moltissime librerie con il nome di `repetition_penalty`. La distribuzione di probabilità per il token successivo con repetition penality viene definita come:

$$P_i = \frac{exp(x_i/(T \cdot I(i \in g))}{\sum_j exp(x_j/(T \cdot I(j \in g)))} \tag{1.13}$$

Tale che:
$I(c) = \Theta$ se c è Vera, altrimenti 1.
$T > 0$ indica la temperatura.
$x_i \in R^d$ indica i punteggi per ogni token i del vocabolario.
g contiene una lista di token precedentemente generati.
$\Theta > 1$ indica il fattore di penalizzazione.

In conclusione, la funziona permette di ottenere un buon equilibrio tra poche ripetizioni e generazioni più precise.

1.11 Campionamento stratificato

Il **campionamento stratificato** è una tecnica di campionamento utilizzata per ottenere campioni che rappresentano al meglio la popolazione da studiare. Riduce la distorsione nella selezione dei campioni dividendo la popolazione in sottogruppi omogenei chiamati *strati* campionando casualmente i dati in ciascuno di essi. In questo modo, ogni campione viene prelevato da ogni strato in proporzione diretta alla dimensione dello strato rispetto alla popolazione, quindi ogni possibile campione ha la stessa probabilità di verificarsi. Nel machine learning, il campionamento stratificato viene comunemente utilizzato per creare dataset di test per valutare i modelli, soprattutto quando il dataset è significativamente grande e sbilanciato.

Figura 1.20: L'immagine mostra il procedimento di stratificazione. La popolazione viene *stratificata* in classi significative. Successivamente vengono mescolate ed estratte in maniera casuale per ogni strato

Fonte: *https://www.scribbr.com/methodology/stratified-sampling/*

In conclusione, i vantaggi che derivano dall'utilizzo di un campionamento stratificato sono molteplici.

- Il campionamento riflette accuratamente la popolazione studiata.
- Assicura che ogni sottogruppo all'interno della popolazione riceva un'adeguata rappresentazione all'interno del campione.
- Fornisce una migliore copertura della popolazione.

L'utilizzo di questa tecnica può risultare svantaggiosa quando ci troviamo davanti a gruppi complessi, dove non possiamo classificare con sicurezza ogni membro della popolazione in un sottogruppo.

Capitolo 2

Prism

All'inizio del capitolo vengono illustrate le principali problematiche legate alle analisi comparative sulle strategie di decoding attualmente esistenti. Nello specifico, vengono inclusi riferimenti ad alcuni degli studi più popolari, con l'obiettivo di mostrarne le criticità attuali. Successivamente, viene presentata l'analisi oggetto di questa tesi, descrivendo le strategie investigate, i parametri selezionati per ciascuna di esse, i dataset e i relativi modelli pre-addestrati, oltre che la configurazione sperimentale.

2.1 Il gap metodico

Ad oggi è stato raggiunto un progresso straordinario in moltissime aree della NLP caratterizzate da costante crescita e sviluppo. In particolare:

- architetture di reti neurali sempre più complesse e sofisticate.
- settore Big Data in costante crescita ha permesso la creazione di dataset sempre più voluminosi, in grado di raccogliere quantità di dati enormi. I dati spesso sono appartenenti allo stesso dominio per tipologia oppure descrivono una varietà di contenuti differenti.
- sviluppo e nascita di nuove strategie di decoding con l'obiettivo di rimanere al passo con le architetture e i modelli attuali. Molte strategie di decoding invece, vengono consolidate e perfezionate rilasciando delle nuove versioni e alternative che derivano da quelle pre-esistenti.

Tra le aree di sviluppo appena elencate ce n'è una in particolare che viene spesso trascurata: quella che riguarda le *strategie di decoding per la generazione di riassunti astrattivi.*

Attualmente, solo una piccola branca di ricerca analizza un *ampio spettro di strategie di decoding* con lo scopo di identificare l'impiego più idoneo di ciascuna di esse. Comprendere e verificare le strategie attuali non solo permette di avere una maggior comprensione su quali di esse performino al meglio, ma anche di avere una "scheda tecnica" che ne racchiuda le qualità. Ma a quale scopo?

Avere la strategia che si *"adatti"* meglio in base al tipo di task desiderato permette un miglioramento significativo nella qualità di generazione del testo. Questo si traduce in uno studio capace di guidare l'utente nella scelta della strategia più idonea, capace di condurre un qualunque modello NLG verso l'inferenza di riassunti di maggior qualità rispetto a dimensioni di proprio interesse (es. leggibilità, fattualità, coerenza, preservazione del contenuto informativo, correttezza grammaticale).

Un altro effetto scaturito dalla possibilità di usufruire di una "scheda tecnica", è la possibilità di confrontare—a priori dal contesto—quale strategia sia migliore rispetto ad un'altra, e quale invece risulti debole. In questo modo, c'è una reale comprensione delle strategie di decoding, decifrandone tutte le possibili sfumature. Ad esempio, dato uno sguardo generale a tutte le strategie è possibile cogliere se esse siano al passo con le tecniche di encoding e le architetture attuali, oppure risultino essere superate. La verifica richiede di prendere in considerazione i vincoli e le dipendenze utilizzate e il tipo di risultato che si vuole ottenere. Purtroppo, ad oggi, non ci sono molti studi che analizzano questa tematica e quelli già esistenti risultano incompleti. Come descritto, le ricerche—se prese singolarmente—presentano tutte delle mancanze:

- La carenza nella scelta degli iper-parametri.
- L'insufficienza nell'utilizzo di modelli per tutti i task di text-summarization.
- La focalizzazione su un singolo dataset, escludendo possibili alternative.
- La selezione a priori di iper-parametri basata sulla scelta dei creatori, senza realmente valutare se siano davvero quelli ottimali nel contesto scelto.
- Soluzioni accettabili ma arretrate, risultando così obsolete.

Citando le ricerche prese in considerazione, Holtzman et al. [4] esaminano 6 strategie di decoding con un elevato numero di configurazioni, senza però riportare con precisione i valori di iper-parametri esplorati, limitandone la replicabilità. Gli autori indagano 5 task NLP differenti; tuttavia, in ambito summarization, solo lo scenario long-document è preso in considerazione su due dataset distinti. Questa limitazione porta a non avere piena sicurezza dell'efficacia di una strategia in scenari short e multi-document. Inoltre, a

distanza di quattro anni dalla pubblicazione, è stato uno dei primi paper comparativi proposti. C'è quindi la possibilità di avere soluzioni e algoritmi deprecati o inconsistenti rispetto alle versioni implementative di librerie attuali (rendendo i confronti "unfair"). Un altro studio considerato è il paper Wiher et al. [5] che a differenza della ricerca precedente ricopre tantissime tipologie di task tralasciando —nel caso di text summarization— solo la multi-document. Inoltre, un possibile problema è la scarsa quantità di configurazioni eseguite sulle strategie selezionate. La ricerca risulta limitante per molti aspetti ma ottima nella scrupolosa selezione degli iper-parametri.

Per evitare di dilungarsi eccessivamente nella trattazione di tutto il set di studi preso in esame, è stato scelto di riunire i principali criteri di interesse in Tabella 2.1.

Fonte	**Strategie di Decoding**	**Parametri**	**Task**	**Documento**			**Co2**	**Runtime**
				Short	**Long**	**Multi**		
Holtzman et al. [4]	BeamSearch Greedy Sampling Top-K Top-P	$beam \in [5, 15]$ / $t \in [0.25, 1.0]$ $k \in [10^1, 10^4]$ $p \in [0.1.09999]$	Text Summ.	/	✓(x1)	/	/	/
Leblond et al. [51]	Sampling BeamSearch Greedy MonteCarlo BeamSearch Value-guided BeamSearch Sampling Reranking Sampling Reranking-Value	/ / / / / / / / /	Text Summ.	✓(x1)	/	✓(x1)	/	/
Wiher et al. [5]	BeamSearch Diverse BeamSearch Greedy Ancestral Top-K Top-P	$beam \in \{5\}, \{10\}$ $\lambda \in \{0.7\}$ / $a \in [30, 32]$ $k \in \{30\}$ $p \in \{0.85\}$	Text Summ. Machine Trans. Dialogue Uncoditional Gen. Story Gen.	/	✓(x1)	/	/	/
Ippolito et al. [6]	BeamSearch Top-G BeamSearch Iterative BS Diverse BeamSearch Noisy BeamSearch Clustered BeamSearch Sampling Top-P	$beam \in \{10\}$ $g \in \{3\}$ $i \in \{5\}$ $\lambda \in \{0.8\}$ $n \in \{0.3\}$ $c \in \{5\}$ $t \in [0.5, 10]$ $t \in \{10\}$	OpenEnded Dia. Image Capt.	/	/	/	/	/
Our study	Greedy BeamSearch Diverse BeamSearch Sampling Top-K Top-P	/ $beam \in [2, 10]$ $d \in [0.2, 1.0]$ $t \in \{0.8\}, \{1.0\}$ $k \in [30, 50]$ $p \in [0.4, 0.8]$	Text Summ.	✓(x1)	✓(x1)	✓(x1)	✓	✓

Tabella 2.1: Nell'ultima riga vengono presentate le caratteristiche del nostro studio. (...) riportano il numero di dataset testati per ogni tipologia di summarization.

Si evince immediatamente dalla tabella come ciascuno tenda a privilegiare

qualcosa, trascurando nel contempo qualcos'altro, che può essere: lo spettro di iper-parametri, il numero di configurazioni adottate, oppure le sperimentazioni eseguite sui vari task differenti dalla abstractive text summarization.

Il nostro obiettivo è stato quello di *colmare* questo gap riorganizzando l'indagine. Lo studio doveva assorbire le scoperte essenziali che abbiamo fino ad oggi fondendole in un'unica soluzione, che veniva quindi estesa anche a quelli parti che a parer nostro erano mancanti.

Il nostro studio, descritto nell'ultima riga della Tabella 2.1, alla voce (**our study**) ha selezionato tutti i task più importanti—ad esclusione delle varianti emergenti—selezionando quanti più iper-parametri possibili. Rispetto alle altre ricerche abbiamo cercato di ricoprire i range di valori ammissibili per ogni strategia di decoding. Li abbiamo testati su *tutti* i possibili task di summarization, che sono short-document, long-document e multi document.

Per rendere concreto il nostro studio abbiamo testato la predizione di ogni metrica con quante più metriche possibili prese dalla libreria (metric-verse) per avere quante più informazioni possibili. Infine, abbiamo tenuto conto anche:

- del tempo totale per ogni strategia;
- della quantità di CO2 emessa.

2.2 L'approccio adottato

In un primo momento ci siamo focalizzati sulla selezione dei principali task di abstractive summarization. Sono stati presi in considerazione i task di **Short-Document**, **Long-Document** e **Multi-Document**. Per elaborare i task scelti, abbiamo preso come riferimento la libreria **Transformer** di Hugging Face (HF)[1]. Il portale è ampiamente supportato da sviluppatori e ricercatori e presenta i contenuti più influenti nel dominio NLP. La libreria Transformer propone una quantità ingente di modelli e dataset che permettono una grande quantità di task tra cui NLP, computer vision, audio e multi-modale.

Per la scelta di modelli e dataset, ci siamo basati sulle risorse disponibili su HF tutelate e supportate da tantissimi sviluppatori, attualmente "gold standard" per la NLP. Relativamente ai task presi in considerazione, sono stati considerati dei modelli è stata funzionale ai dataset più rappresentativi dei task. In questo

[1] `https://huggingface.co/docs/transformers/main/en/index`

modo abbiamo ottenuto una coppia dataset-modello per ciascun task.

Partendo dai dataset quelli che abbiamo scelto sono:

- **XSum** (Extreme Summarization) [16] per i task di Short-Document. Possiede una banca dati di riassunti estremamente sintetici e astrattivi con contenuti di dominio generico.
- **ArXiv** [18] per i task di Long-Document. Grande collezione di articoli, paper e documenti di dominio scientifico.
- **Multi-news** per i task di Multi-Document. Una collezione di articoli di giornale e di cronaca. Il dataset è stato uno dei primi proposti e pubblicati per eseguire MDS.

Le tipologie di task e dataset permettono alla sperimentazione di essere ancora più variabile grazie alla lunghezza del riassunto e al contenuto dei documenti. Ma non è tutto, i dataset presi in considerazione sono di fatto inerenti a tre domini completamente diversi.

Riguardo ai modelli, ne abbiamo selezionati tre, tutti allo stato dell'arte:

- **PRIMERA-Large** [52], modello adottato principalmente per Multi-Document Summarization.
- **LED-Large** [53], uno dei migliori modelli per Long-Document Summarization.
- **BART-Large** [35], modello redditizio su molteplici task sequence-to-sequence, tra cui Short-Document Summarization.

Inoltre, l'architettura del modello LED è una variante del modello BART, che viene adattato per task di Long-Document Summarization. Mentre PRIMERA è una modifica/miglioria di LED adattata per task di Multi-Document Summarization. Di seguito nella Tabella 2.2 vengono riassunte le scelte appena citate per avere una visione più chiara della soluzione adottata.

In un secondo momento abbiamo scelto uno spettro di strategie di decoding il più ampio possibile mai considerato fino ad ora. Il criterio è stato quello di selezionare quelle attualmente più stabili, trascurando quelle emergenti ancora in fase di sviluppo e testing. In particolare per fare inferenza ci siamo focalizzati

Summarization Task	Modello	Dataset
Short Document	PRIMERA-Large	Multi-news
Long Document	LED-Large	ArXiv
Long Document	BART-Large	XSum

Tabella 2.2: Rappresentazione della scelta adottata per modelli e dataset riguardo a specifici task di abstractive summarization.

sul metodo della libreria Transformer *generate()*[2]

L'implementazione del metodo, ma in generale tutto il caso di studio è stata svolta tramite l'utilizzo di file e notebook **Python**. Per generare le predizioni di tutte le strategie di decoding è stata creata una funzione "generica" che racchiude tutti parametri in comune, che è stata poi assegnata alla funzione specifica.

```python
def generate_by_method(model, args, input_ids, attention_mask, global_attention_mask,
    tokenizer):
    base_param = dict(
        input_ids=input_ids,
        attention_mask=attention_mask,
        early_stopping=True,
        min_length=100,
        max_length=256,
        no_repeat_ngram_size=args.no_repeat_ngram_size
    )
    # decode the encoding model with the specified attributes given by the args
    if args.decoding_method == "beam-search":
        return generate_beam_search(model, args, global_attention_mask, base_param)
        # Continue for all decoding strategies
        ...
```

Figura 2.1: Funzione generale che racchiude le principali caratteristiche per ogni strategia, che viene poi assegnata alla funzione di interesse.

Successivamente, in base alla strategia di decoding adottata, viene selezionata la funzione di interesse, che viene specializzata tramite i propri iper-parametri. Inoltre, viene controllato se è presente *global attention*: un parametro necessario per gestire l'attention nei task di multi-document.

[2] https://huggingface.co/docs/transformers/main_classes/text_generation.

```python
# test method that decode the dataset with the specified model and tokenizer (given)
def generate_beam_search(model, args, global_attention_mask, param):
    param.update(dict(
        early_stopping=True,
        num_beams=args.num_beams,
        length_penalty=args.length_penalty,

        repetition_penalty=args.repetition_penalty
    ))
    if global_attention_mask is not None:
        param['global_attention_mask'] = global_attention_mask

    return model.generate(**param)
```

Figura 2.2: Definition and application of a *Scorer* object for the concurrent evaluation of multiple metrics.

Di seguito, vengono elencate e motivate le strategie di decoding, correlate agli iper-parametri, adottate nel caso di studio:

- **Beam Search** (§ 1.9.2), strategia cardine. Abbiamo selezionato un range di *beam* b con $b = (2, 3, 4, 5, 6, 7, 8, 9, 10)$. La finestra permette un trade-off tra qualità e prestazioni.

- **Diverse Beam Search** (§ 1.9.3), variazione del Beam Search, è un'ottima alternativa in task di text summarization. Oltre a tenere in considerazione più alternative, si avvale anche del concetto di diversità, permettendo di creare ipotesi distinte tra loro. Abbiamo mantenuto invariata la finestra di *beam* b con $b = (2, 3, 4, 5, 6, 7, 8, 9, 10)$. La diversità d, denominata *diversity_penalty* nella libreria, ricopre intelligentemente tutto il dominio $D \in [0, 1]$ con una quantità moderata di parametri $d = (0.2, 0.4, 0.6, 0.8, 1)$.

- **Greedy** (sec.1.9.1), versione "debole" del Beam Search. È l'algoritmo più semplice di tutti, ma non per questo va escluso dallo studio. Può risultare ottimo per eseguire prime generazioni di confronto e verifica. Nonostante la sua semplicità, potrebbe portare a risultati inaspettati.

- **Temperature Sampling** (sec.**??**), estensione del Sampling Base. Permette di campionare un insieme di pesi in maniera casuale applicando un fattore t denominato temperatura. Abbiamo scelto i valori di t in modo da non ottenere valori infiniti o nulli. L'idea è stata quella di scegliere valori compresi tra il massimo del minimo ammissibile ed il minimo valore massimo che corrisponde all'estremo destro del dominio $D \in (0, 1]$, quindi tale che $t = [0.8, 0.9, 1.0]$.

- **top-k sampling** (sec.1.9.6), estensione del Sampling Base. Permette di campionare un insieme di pesi in maniera casuale limitando la scelta ad un pool di valori k. I valori per k sono stati scelti in modo da non appiattire o schiacciare troppo la distribuzione di probabilità, di conseguenza $k = [20, 30, 40] \notin [50]$ dato che 50 è un parametro di default del metodo *generate()*.
- **top-p sampling (nucleus)** (sec.1.9.5), estensione del top-k sampling. Oltre al funzionamento appena descritto aggiunge un paramento p che permette di selezionare un pool di valori k a partire da una distribuzione minima p. Anche p è stato scelto in modo da ricoprire intelligentemente il dominio $D \in [0, 1]$ focalizzandosi maggiormente sul centro. Di conseguenza $p = [0.4, 0.6, 0.8] \notin [1.0]$ dato che 1 è parametro di default del metodo *generate()*. Inoltre, abbiamo incluso il parametro di temperatura t, scelto con le medesime considerazioni fatte per la strategia di "temperature sampling", quindi tale che $t = [0.8, 0.9, 1.0]$. Questo permette di ottenere una strategia più completa e varia.

Nella Tabella 2.3 vengono riassunte tutte le strategie adottate, associate a tutti gli iper-parametri, appena descritte:

Iper-Parametri	**Greedy**	**Beam Search**		**Sampling**		
		Base	**Diverse**	**Base**	**Top-K**	**Top-P**
no_rep_ngram_size	$2, 3, 4, 5$	$2, 3, 4, 5$	$2, 3, 4, 5$	$2, 3, 4, 5$	$2, 3, 4, 5$	$2, 3, 4, 5$
early_stopping	/	✓	✓	/	/	/
diversity_penalty	/	$0.\{2, 4, 6, 8\}$	/	/	/	/
num_beams	/	2...10	2...10	/	/	/
temperature	/	/	/	$0.\{7, 8, 9\}, 1$	$0.\{7, 8, 9\}, 1$	$0.\{7, 8, 9\}, 1$
top_p	/	/	/	/	/	$0.\{4, 6, 8\}$
top_k	/	/	/	/	$20, 30, 40$	$20, 30, 40$
do_sample	/	/	/	✓	✓	✓

Tabella 2.3

Un iper-parametro non trascurabile che accomuna tutti i metodi descritti fin'ora è *no_repeat_ngram_size*, che viene rappresentato come valore intero. Evita (in particolare per il Beam Search), durante la generazione, la ripetizione di una parola, denominata *ngram* vincolando nella selezione un numero massimo di volte per il quale può presentarsi. In generale, viene utilizzato quando il dataset è molto grande o inconsistente. Nel nostro caso è stato utilizzato *no_repeat_ngram_size* di 2 e 3 per XSum essendo un dataset abbastanza sintetico, mentre per Multi-news e Arxiv è stato scelto *no_repeat_ngram_size* di $3, 4, 5$ in modo da assicurare la non ripetitività in vista di più casistiche

possibili. L'utilizzo di questo parametro conferisce—in fase di sperimentazione— 8 configurazioni minime per ogni strategia adottata.

2.3 Experiment setup

2.3.1 Il codice

Lo sviluppo primario, si è concentrato sulla preparazione dell'ambiente di decoding. Una parte del codice in questione è possibile visionarlo nel paragrafo precedente nel Codice 2.1 e il Codice 2.2. Essendoci focalizzati sul layer di decoding, per essere svolto, si ha la necessità di avere già a disposizione il layer di encoding con il rispettivo testo di riferimento. Quindi, prima di generale il testo, abbiamo la necessità di avere un insieme di elementi, tra cui modelli e dataset. È possibile definire una panoramica di sviluppo, distinguendo il funzionamento in tre macro blocchi in ordine di esecuzione:

- **Acquisizione del dataset e pulizia dei dati**. Innanzitutto, viene caricato il dataset localmente se disponibile, altrimenti viene ottenuto direttamente dalla repository di Hugging Face. Successivamente vengono escluse dal dataset le parti di training e validation lasciando esclusivamente la parte di test, necessaria per la decodifica. Infine, viene fatta una pulizia del testo in maniera da modificare ed eliminare le parti superflue o ambigue.

- **Caricamento del modello pre-addestrato**. In secondo luogo, viene selezionato —come per il dataset— il modello, caricato localmente se presente, oppure direttamente dalla repository di Hugging Face.

- **Generazione del testo e salvataggio della predizione**. Step cardine della sperimentazione. Viene definisco, in base al tipo di dataset se necessaria la *global attention*. Successivamente viene scelta la strategia e impostati gli iper-parametri definiti. Infine, a termine dell'inferenza, la predizione generata viene salvata sia localmente che in remoto.

L'esecuzione dei tre blocchi appena descritta è stata eseguita per ogni modello e dataset, in unione alla strategia e a tutti gli iper-parametri utilizzati. Le combinazioni totati di tutte le esecuzioni o semplicemente run sperimentate nel caso di studio sono state **856** escludendo escludendo quelle di test.

Di seguito nella tabella 2.4 vengono mostrate per ogni strategia il numero totale di run.

Strategia di decoding adottate	Numero totale di combinazioni
Beam Search	72
Diverse Beam Search	360
Greedy	8
Sampling	32
Top-K Sampling	96
Top-P Sampling	288

Tabella 2.4: Rappresentazione della scelta adottata per modelli e dataset riguardo a specifici task di abstractive summarization.

Per riuscire a supportare tutte le combinazioni possibili è stato necessario rendere il codice elastico a causa dell'elevata quantità di iper-parametri attraverso l'utilizzo di *argomenti*. Di conseguenza, per agevolare la sperimentazione, è stato generato un file di configurazione per ogni singola strategia, contenente tutti gli iper-parametri necessari.

È possibile reperire il progetto, compreso delle sue revisioni nella repository Github[3] del Dipartimento di Computer Science and Engineering (DISI).

2.3.2 Ambiente di sperimentazione

L'inferenza, per ogni strategia adottata, richiede un elevato impiego di risorse e tempo computazionale, per questo motivo è stata svolta in un cluster di 2 server appartenenti all'Ateneo con eguali caratteristiche. L'ambiente opera sul sistema operativo Ubuntu. Il **cluster**, appartenente all'Università di Bologna e messo a disposizione dal Dipartimento di Informatica - Scienza e Ingegneria (DISI) possiede le seguenti specifiche: 5 schede grafiche Nvidia RTX 3090 da 24Gb di RAM e 2 processori Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz. I driver Nvidia sono: NVIDIA-SMI 470.141.03, Driver Version: 470.141.03 e CUDA Version: 11.4.

L'Assegnazione dei task è gestita dallo Slurm Workload Manager[4] conosciuto anche come SLURM. Esso è un sistema di gestione dei cluster e pianificazione dei lavori open source, tollerante ai guasti e altamente scalabile per sistemi Linux. SLURM possiede tre funzioni chiave. Innanzitutto, assegna agli utenti l'accesso esclusivo e/o non esclusivo alle risorse per un certo periodo di tempo in modo che possano eseguire il lavoro. In secondo luogo, fornisce un framework

[3]https://github.com/disi-unibo-nlp/decoding-summ
[4]https://slurm.schedmd.com/

per avviare, eseguire e monitorare il lavoro sull'insieme dei nodi allocati. Infine, arbitra la contesa per le risorse gestendo una coda di lavoro in sospeso. I singoli lanci sbatch consentono di avviare attività sul cluster parallelamente. Utilizzando sbatch, l'attività viene gestita in background da SLURM e l'utente può disconnettersi, chiudere il terminale, ecc. senza conseguenze. Il lavoro non è più collegato alla shell in esecuzione.

2.3.3 Docker e librerie

Per realizzare un ambiente che renda gli esperimenti testabili e replicabili per tutti i task, è stato utilizzato **Docker**. Esso permette di ricreare un ambiente virtuale denominato *container* dove eseguire il codice. L'ambiente virtuale emula un sistema, isolando l'esecuzione. Questo permette di proteggere il server da possibili errori tediosi di *out of memory* facendo fallire il container e non l'intera macchina. Il container viene prodotto tramite un'immagine docker. Essa rappresenta lo scheletro padre per la creazione di nuove istanze container figlie. Questo, permette—soprattutto nel nostro caso di studio— di distribuire e ricalibrare le dipendenze necessarie all'ambiente di sperimentazione.

L'immagine rappresentata nella figura 2.3 contiene i principali framework per reti neurali, tra cui Transformer, *Pytorch* e *Tensorflow*.

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

WORKDIR /workspace

RUN pip3 install torch==1.170.2+cu113 torchvision==0.11.3+cu113 torchaudio==0.10.2+cu113
↪ -f https://download.pytorch.org/whl/cu113/torch_stable.html
RUN pip3 install git+https://github.com/huggingface/transformers
RUN pip3 install --upgrade nvidia-ml-py3
RUN pip3 install --upgrade sentence_transformers
RUN pip3 install --upgrade accelerate
RUN pip3 install --upgrade codecarbon
RUN pip3 install --upgrade streamlit
RUN pip3 install --upgrade wget
RUN pip3 install --upgrade tensorflow
RUN pip3 install --upgrade tensorflow-datasets
RUN pip3 install --upgrade scikit-learn
RUN pip3 install --upgrade nltk
RUN pip3 install --upgrade stqdm
RUN pip3 install --upgrade datasets
RUN pip3 install --upgrade wandb

COPY ./netrc /root/.netrc

```

Figura 2.3: Dockerfile utilizzato per la creazione dell'immagine e del suo container.

Inoltre, abbiamo selezionate due librerie molto interessanti, eseguite in ogni script di decoding:

- **Code Carbon**[5] è una libreria leggera che si integra perfettamente nel codice Python. Stima la quantità di anidride carbonica (CO2) emessa. Mostra quindi agli sviluppatori come possono ridurre le emissioni ottimizzando il loro codice. Il nostro contributo è stato calcolare l'emissione totale emessa per ogni run. Per implementare il tracking dell'emissione vengono richieste poche istruzioni. Nel codice 2.4 è possibile

```
tracker = EmissionsTracker()
tracker.start()
# ... other code
total_emission = tracker.stop()
```

Figura 2.4: Codice implementativo di CodeCarbon.

- **Weight&Biases**[6] (W&B), spesso riferita come "W and B", è una libreria che permette di essere integrata agilmente. La piattaforma è un servizio

[5]https://codecarbon.io/
[6]https://wandb.ai/site

Cloud che mette a disposizione numerosi strumenti utili, tra cui:

- monitoraggio degli esperimenti;
- dashboard collaborative;
- controllo di versione per modelli e dataset;
- tabelle interattive;
- ottimizzazione di iper-parametri.

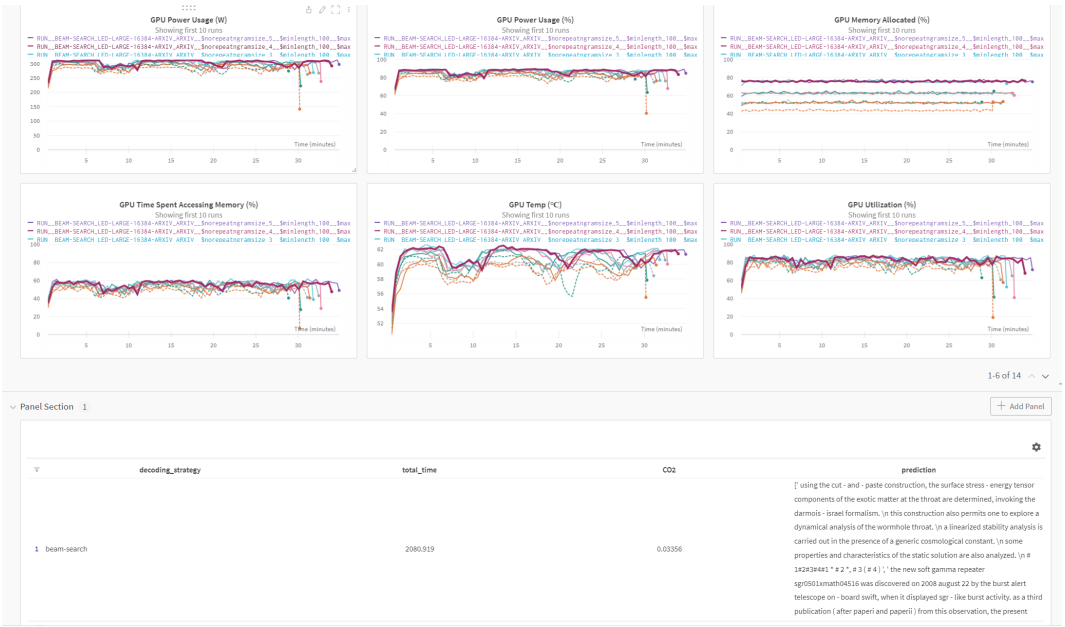


Figura 2.5: Dashboard W&B che mostra le statistiche di tutte le run eseguite per il Beam Search.

Nel nostro caso di studio, W&B ha permesso di avere una dashboard condivisa della sperimentazione in tempo reale. Per ogni run è stata popolata una tabella con i dati di nostro interesse, tra cui tipologia di decoding, tempo di esecuzione, carbonio emesso e predizione finale. Inoltre, abbiamo permesso l'utilizzo o meno della libreria tramite un argomento booleano.

2.3.4 Stratificazione dei dataset

Per evitare di limitare la scelta delle combinazioni totali proposte nell'analisi, a causa sia delle risorse limitate e per l'elevato tempo di esecuzione per ogni run

è stato scelto di trovare una soluzione che permette di ridurre i tempi, senza perdere la qualità della codifica. La soluzione più avanzata ma allo stesso tempo semplice da implementare è stato eseguire il **campionamento stratificato** sui dataset utilizzati. Come descritto nel paragrafo 1.11 del Capitolo 1, permette di ridurre drasticamente le dimensioni del dataset evitando di perdere informazioni importanti.

Lo sviluppo del codice e i primi test sono stati effettuati su notebook di Google Colaboratory o semplicemente Colab[7].

Per eseguire la stratificazione, abbiamo scelto come feature la **lunghezza dei documenti** e la **lunghezza dei riassunti** in modo da non creare sbilanci nella decodifica del documento originario. Per ogni dataset abbiamo classificato le lunghezze di documenti e riassunti in terzili. Ottenendo tre classi per ognuna, enumerabili in SHORT, MEDIUM e LARGE. Per il dataset multi-news è stata preso in considerazione il **numero di documenti** come feature aggiuntiva, classificata sempre in terzili.

L'Implementazione della funzione di stratificazione appartiene all'ecosistema di **R**.L'esecuzione è stata svolta all'interno di un file notebook Python, questo è reso possibile grazie alla funzionalità "R magic". Essa è una funzionalità della libreria rpy2[8] che permette di eseguire codice R incorporato in un processo Python tramite l'utilizzo del comando %%R nelle celle di codice desiderato. Il campione è stato impostato in modo da essere proporzionale al *10%* rispetto al numero totale del numero di osservazioni per classe campionando senza *rimpiazzo*. Di seguito nel blocco di codice 2.6 viene mostrata l'operazione di stratificazione svolta per tutti i dataset del nostro studio. Il risultato è stato successivamente visionato ed esportato per poi essere utilizzato nel codice principale di generazione del testo.

[7]https://colab.research.google.com/
[8]https://rpy2.github.io/

```
%%R -i data -i document_percentile -i summary_percentile -i DATASET
{
    if(DATASET == 'multi_news') {
        samples <- stratified(data, c(document_percentile,summary_percentile,'num_doc'),
        ↪  .1, replace=FALSE, keep.rownames=TRUE)
    } else {
        samples <- stratified(data, c(document_percentile,summary_percentile), .1,
        ↪  replace=FALSE, keep.rownames=TRUE)
    }
}

sampling = %R samples
```

Figura 2.6: Stratificazione tramite l'utilizzo del metodo *stratified* di R, richiamato tramite l'utilizzo di celle "R magic". Nella stratificazione vengono prese in considerazione la lunghezza del documento (document_percentile) e del riassunto (summary_percentile) e per il dataset multi-news il numero di documenti (num_doc).

2.4 Stima dei risultati

Per stimare sono state selezionate le principali metriche valutative in ambito NLP. Per valutare le predizioni delle strategie di decoding abbiamo selezionato un'ampia varietà di metriche. Nella tabella sono presenti quelle selezionate.

Metrica	**Dimensioni valutate**
Rouge	fluenza, adeguatezza
Perplexity	copertura semantica, informatività, rilevanza
Abstractness	copertura semantica, informatività, adeguatezza
Repetitiveness	copertura semantica, informatività
Accuracy	copertura semantica, informatività

Tabella 2.5: Prospettive di valutazione delle metriche.

2.4.1 Rouge

ROUGE, o Recall-Oriented Understudy for Gisting Evaluation [54] è un pacchetto di metriche basato sul recall di n-grammi per misurare la similarità fra riassunti. Ancora oggi, è una metrica di riferimento per molti task di NLP. La sua principale qualità è l'elevata velocità di stima. Questo le permette di essere utilizzata come primo approccio valutativo. ROUGE possiede al suo interno diversi tipi di metriche.

- **Rouge-N.** misura il numero di n-grammi coincidenti tra il testo di origine e il testo prodotto. In pratica, viene valutata la *similarità* tra i due testi. Il calcolo si basa sulle occorrenze occorrenze totali degli n-grammi di riferimento individuati nel riassunto candidato in rapporto agli n-grammi totali.
- **Rouge-L** misura la più lunga sotto-sequenza in comune (LCS) fra gli n-grammi di riferimento e quelli da valutare. Rouge-L dove la *precision* e la *recall* sono eseguite usando la lunghezza della sotto-sequenza.

  Rouge-L non controlla la consecutività delle corrispondenze fintanto che l'ordine delle parole è lo stesso.
- **Rouge-W** Risolve il limite di ROUGE-L, utilizzando una corrispondenza LCS ponderata che aggiunge una penalità, in modo tale da ridurre una corrispondenza ripetitiva.
- **Rouge-S** introduce una latenza rispetto ROUGE-N, dando la possibilità di considerare n-grams sovrapposti e non consecutivi.

ROUGE notoriamente inadatta a valutare testi fortemente astrattivi, dal momento che possiede una natura puramente lessicale. Essendo consapevoli dei suoi limiti non ci siamo fermati ad essa.

2.4.2 Perplexity

Dato un modello e una sequenza di testo di input, la **perplexity** misura quanto è probabile che il modello generi la sequenza di testo di input. Intuitivamente, può essere pensato come una valutazione della capacità del modello di prevedere in modo uniforme tra l'insieme di token specificati in un determinato corpus.

Abbiamo preso in considerazione questa metrica perchè principalmente vien utilizzata per valutare dei language model puri, dove l'obiettivo è quello di predire il token successivo. In pratica, fornisce una indicazione di quanto il testo costruito sia inerente da un punto di vista probabilistico, rispetto a tutti i documenti che il modello ha studiato durante la fase di training; quindi ci dà un'idea della "naturalezza" del testo.

Il calcolo consiste nell'utilizzare una strategia a finestra scorrevole (dimensione della finestra = lunghezza massima), in cui si sposta continuamente il contesto attraverso la sequenza (di un certo passo, ovvero non necessariamente 1 token alla volta), consentendo al modello di trarne vantaggio del contesto disponibile. Più piccolo è il passo, più contesto avrà il modello nel fare ogni previsione e migliore sarà in genere la perplessità segnalata.

2.4.3 Abstractness e Repetitiveness

Abstractivity e **Repetitivity** sono strettamente specifiche della text summarization.Sono metriche che riportano importanti statistiche legate al dominio del documento. Possono rilevarsi particolarmente utili per costruire un quadro completo rispetto alla similarità lessicale e quella semantica.

L'astrattività è la proporzione tra il numero totale di parole identiche identificate nel testo di riferimento e quello predetto diviso il numero totale di parole. Più il valore dell'astrattività è alto meno il testo è astrattivo.

La ripetitività è la proporzione tra la somma delle parole ripetute nel testo diviso la lunghezza totale del testo. Più il valore è elevato è maggiore sarà la ripetitività nel testo.

2.4.4 Accuracy

L'**accuracy** [55] (o precisione) indica la proporzione di previsioni corrette sul numero totale di casi elaborati. Quindi la precisione nella generazione del testo calcola l'accuratezza basata sui token. Può essere calcolato con:

$$Precisione = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.1}$$

Con:
TP: True positive
TN: True negative
FP: False positive
FN: False negative

2.4.5 Implementazione

Per implementare le metriche, ci siamo appoggiati sulla libreria NLG-Metricverse[9]. La libreria è stata sviluppata da Dipartimento di Ingegneria e Scienze Informatiche (DISI), supporta complessivamente 38 diverse metriche di valutazione e permette una agile implementazione di esse. Il calcolo della stima confronta il testo di riferimento ottenuto dalla *gold summary* associato al dataset stratificato con la predizione ottenuta. Il codice implementato, calcola ad ogni predizione il punteggio di tutte le metriche selezionate, salvando i risultati ottenuti.

[9]https://github.com/disi-unibo-nlp/nlg-metricverse

```
for index, row in decoded_frame.iterrows():
    evaluation = row.to_dict()
    scorer = NLGMetricverse(metrics=load_metric(METRIC_PATH + "rouge"))

    scores = scorer(predictions=row["predictions"], references=row["summary"],
    ↪  reduce_fn=REDUCTION_FUNCTION)
    evaluation['rouge'] = scores['rouge']
    scorer = NLGMetricverse(metrics=load_metric(METRIC_PATH + "perplexity"))

    scores = scorer(predictions=row["predictions"], references=row["summary"],
    ↪  reduce_fn=REDUCTION_FUNCTION)
    # ... other code
    evaluation['repetitiveness'] = scores['repetitiveness']

    scorer = NLGMetricverse(metrics=load_metric(METRIC_PATH + "accuracy"))
    scores = scorer(predictions=row["predictions"], references=row["summary"],
    ↪  reduce_fn=REDUCTION_FUNCTION)
    evaluation['accuracy'] = scores['accuracy']

    completed_frame = completed_frame.append(evaluation, ignore_index=True)
```

Figura 2.7: Il codice itera tutte le predizioni di una singola strategia di decoding inserendo in un dizionario "evaluation" i risultati ottenuti per ogni singola metrica. Infine, le valutazioni vengono salvate in un dataframe "completed_frame".

Capitolo 3

Risultati

Per avere una panoramica il più dettagliata possibile, sono stati elaborati dei grafici e delle tabelle che racchiudono le informazioni emerse con lo studio svolto.

3.1 Esito delle generazioni

L'esito di ogni generazione è rappresentata sotto forma di grafico a dispersione. Per comprendere meglio i risultati e rappresentarli in maniera strutturata, sono stati presi come riferimento gli score predetti dalle singole metriche di valutazione menzionate nella Tabella 2.5.

I risultati sono raggruppati per le metriche dello studio, divisi per strategia di decoding e poi suddivisi per ogni iper-parametro chiave in riferimento al dataset utilizzato.

Per separare ulteriormente i dati, ogni grafo mostra la dispersione dei valori per ogni parametro di `no_repeat_ngram_size`—due nel caso del dataset XSum, tre per Arxiv e Multi-news.

3.1.1 ROUGE

Per Rouge, il valore di riferimento è la media dei singoli risultati ottenuti tra Rouge-1 (r_1), Rouge-2 (r_2) e RougeL (r_L), divisi per la somma di $1 + \sigma_r^2$, dove ... In questo modo si ottiene un risultato più accurato, meno dipendente da un singolo risultato. Il valore ricavato è compreso tra $[0, 1]$.

$$\hat{R} = \frac{Avg(r_1, r_2, r_L)}{1 + \sigma_r^2} \tag{3.1}$$

Beam Search

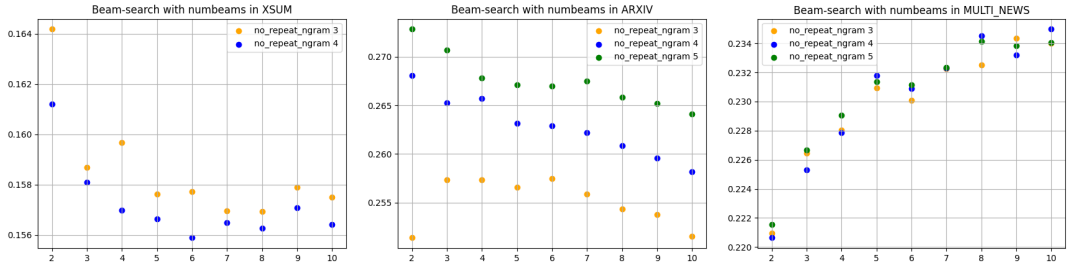


Figura 3.1: Beam Search con x=beams, y=rouge ($\hat{R}$).

Diverse Beam Search

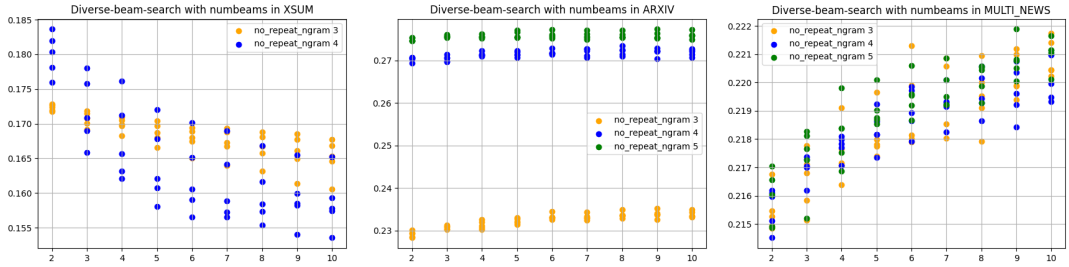


Figura 3.2: Diverse Beam Search con x=beams, y=rouge (avg).

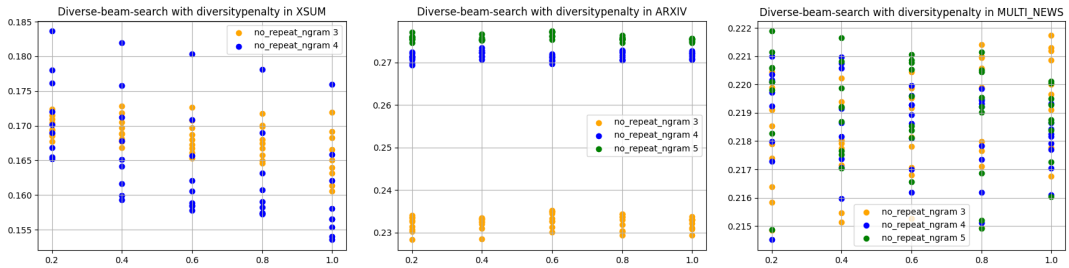


Figura 3.3: Diverse Beam Search con x=diversity penalty, y=rouge (avg).

Greedy

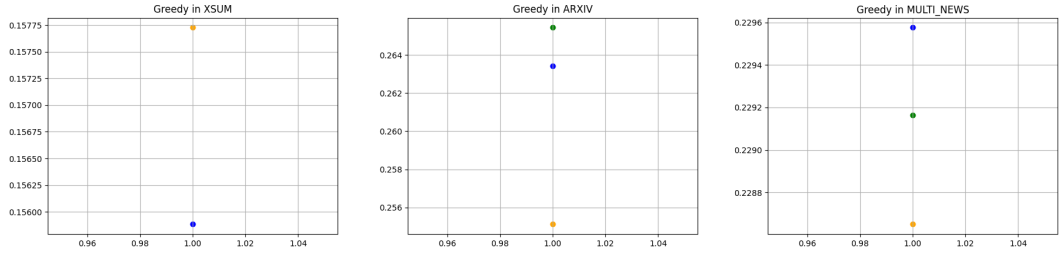


Figura 3.4: Greedy con x=costante 1, y=rouge (avg).

Sampling Temperature

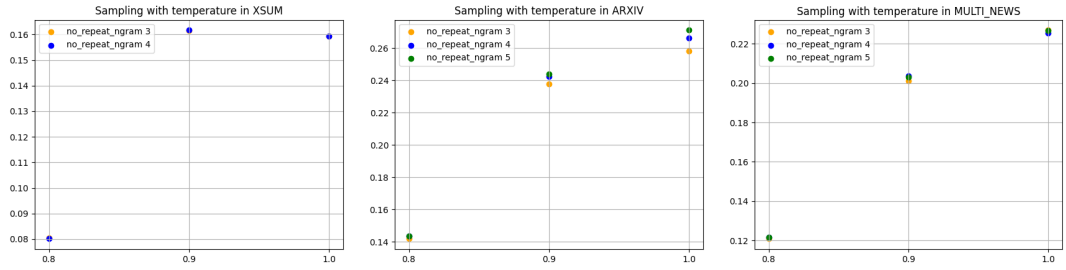


Figura 3.5: Sampling Temperature con x=temperatura, y=rouge (avg).

Sampling Top-K

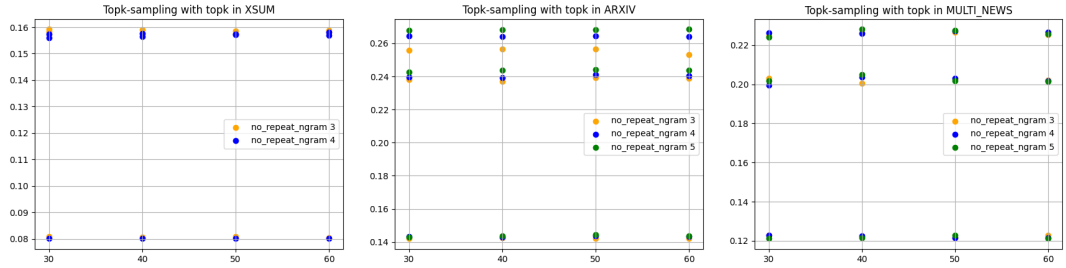


Figura 3.6: Sampling Top-K con x=topk, y=rouge (avg).

Top-P

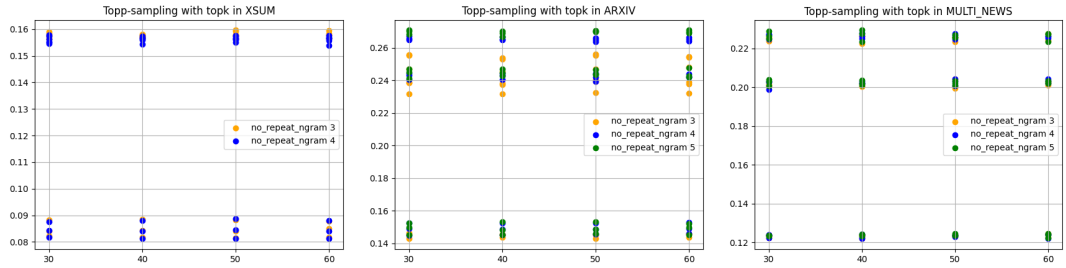


Figura 3.7: Sampling Top-P con x=topk, y=rouge (avg).

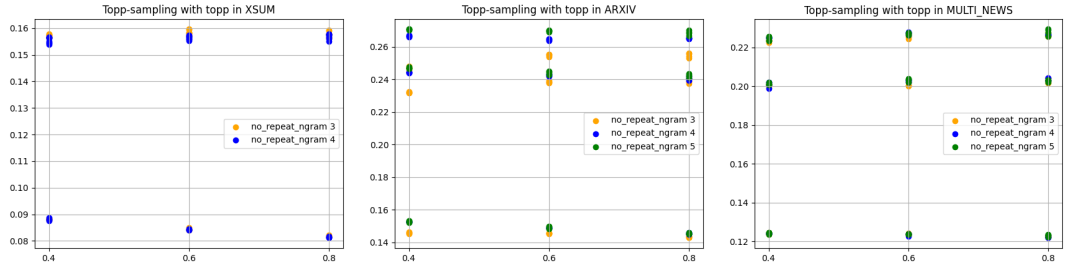


Figura 3.8: Sampling Top-P con x=topp, y=rouge (avg).

3.1.2 Perplexity

Il valore preso in considerazione per la perplessità è la media dei singoli punteggi ottenuti sul testo generato. Un valore basso indica che la distribuzione di

probabilità del testo è stata appresa correttamente dal modello addestrato.

BeamSearch

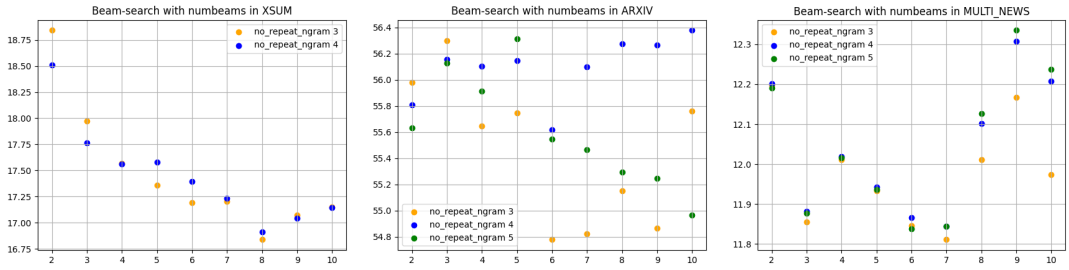


Figura 3.9: Beam Search con x=beams, y=perplexities (avg).

Diverse Beam Search

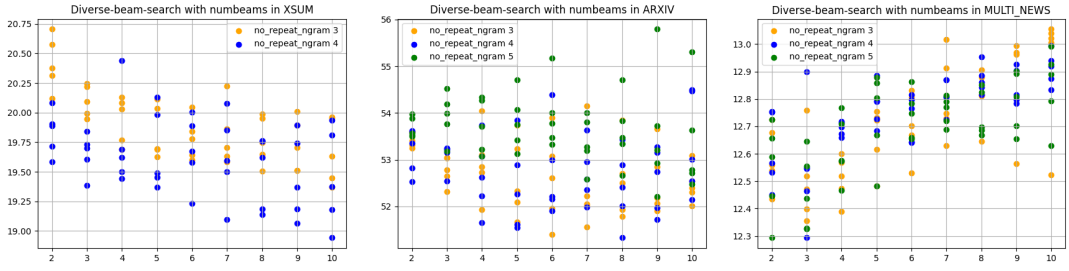


Figura 3.10: Diverse Beam Search con x=beams, y=perplexities (avg).

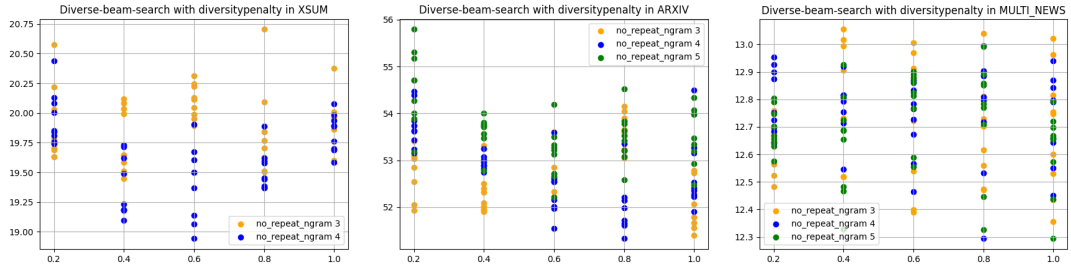


Figura 3.11: Diverse Beam Search con x=diversity penalty, y=perplexities (avg).

Greedy

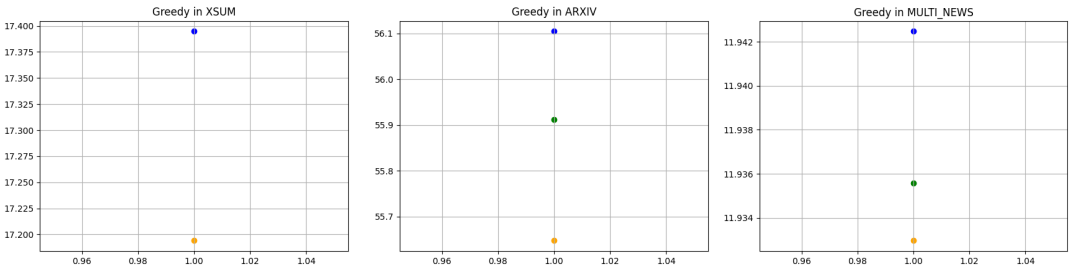


Figura 3.12: Greedy con x=costante 1, y=perplexities (avg).

Sampling Temperature

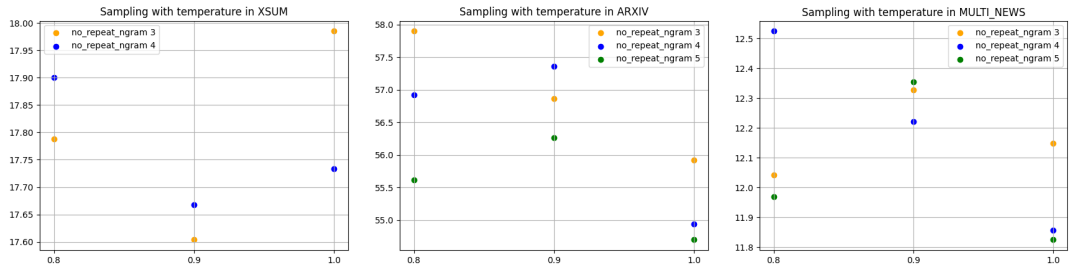


Figura 3.13: Sampling Temperature con x=temperatura, y=perplexities (avg).

Sampling Top-K

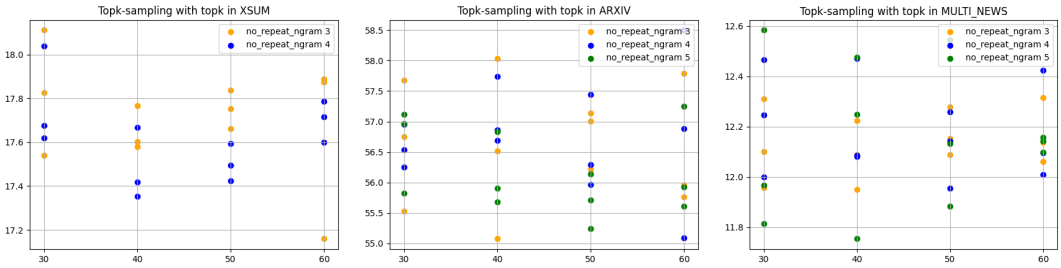


Figura 3.14: Sampling Top-K con x=topk, y=perplexities (avg).

Top-P

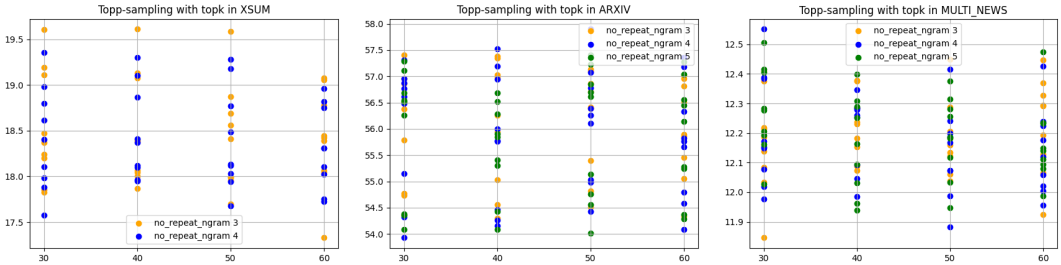


Figura 3.15: Sampling Top-P con x=topk, y=perplexities (avg).

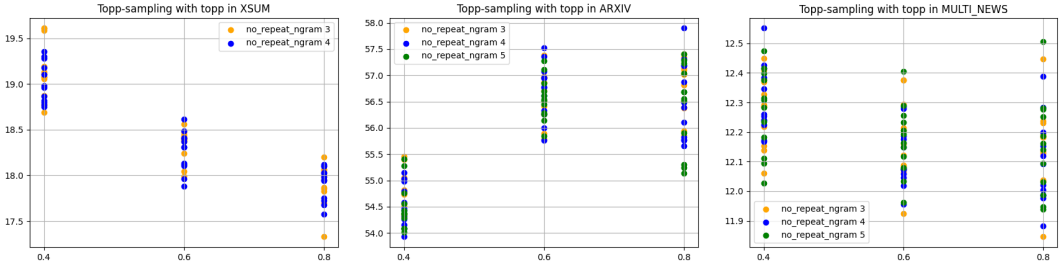


Figura 3.16: Sampling Top-P con x=topp, y=perplexities (avg).

3.1.3 Abstractness

Il valore di astrattività è un valore compreso tra $[0, 1]$. Un punteggio elevato indica che la predizione generata possiede una buona astrattività.

BeamSearch

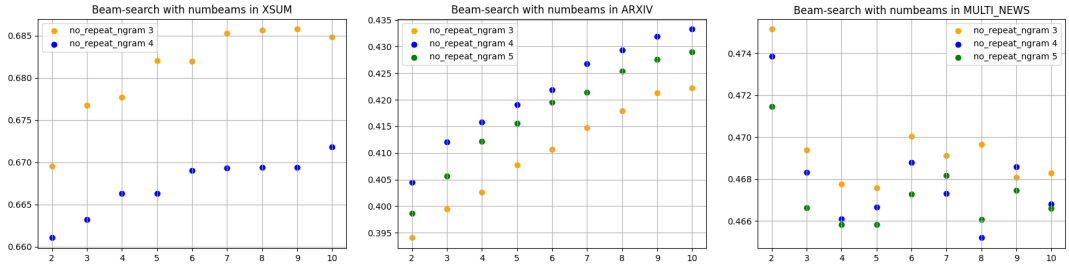


Figura 3.17: Beam Search con x=beams, y=abstractness (score).

Diverse Beam Search

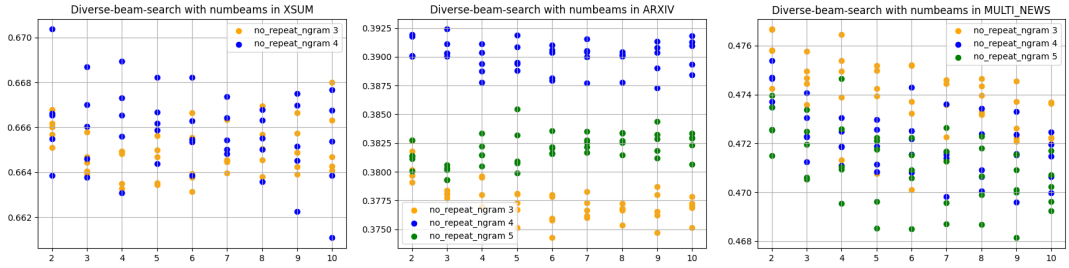


Figura 3.18: Diverse Beam Search con x=beams, y=abstractness (score).

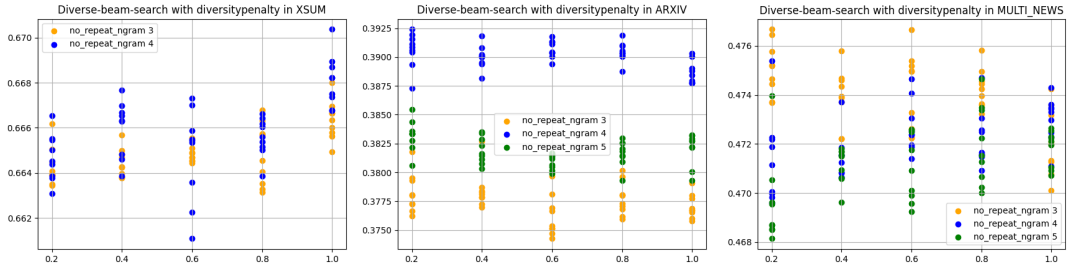


Figura 3.19: Diverse Beam Search con x=diversity penalty, y=abstractness (score).

Greedy

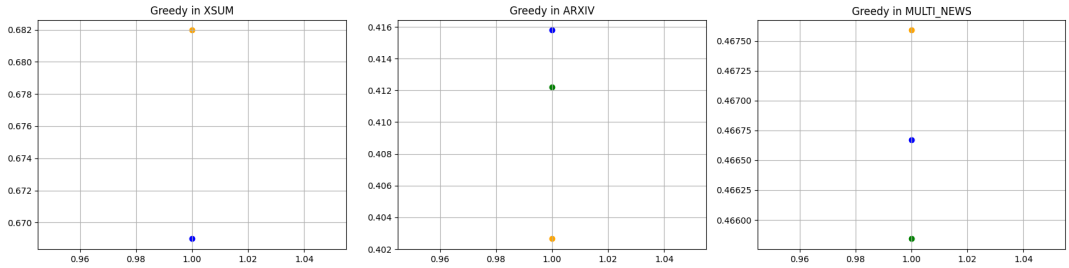


Figura 3.20: Greedy con x=costante 1, y=abstractness (score).

Sampling Temperature

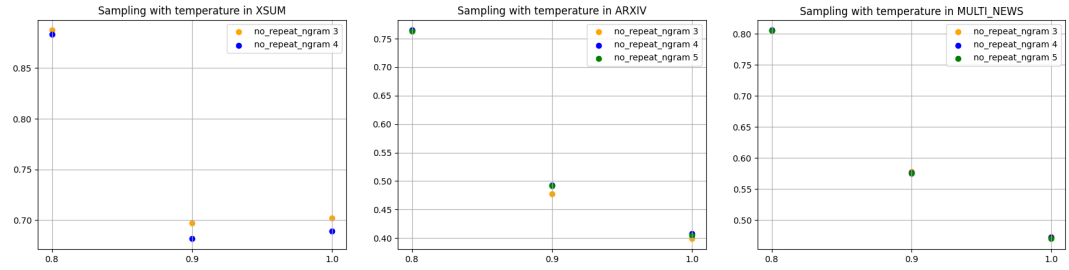


Figura 3.21: Sampling Temperature con x=temperatura, y=abstractness (score).

Sampling Top-K

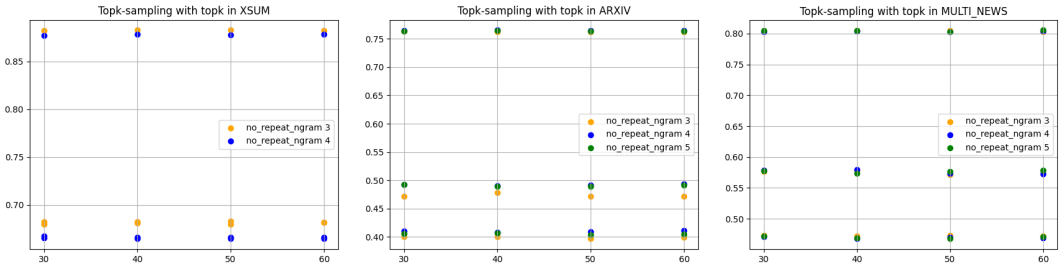


Figura 3.22: Sampling Top-K con x=topk, y=abstractness (score).

Top-P

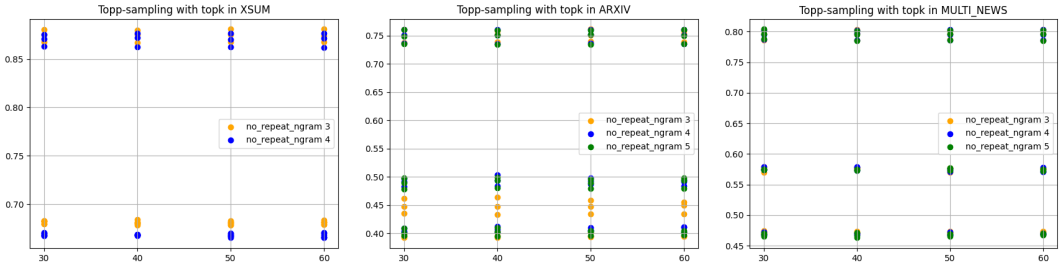


Figura 3.23: Sampling Top-P con x=topk, y=abstractness (score).

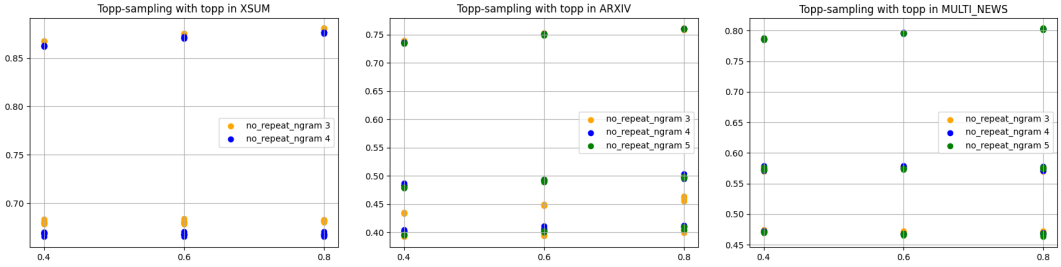
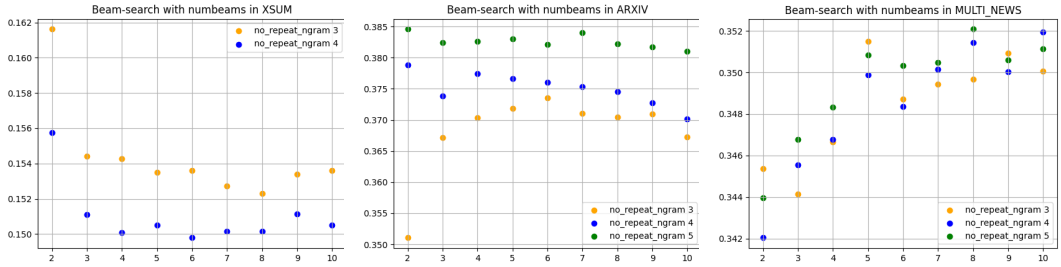


Figura 3.24: Sampling Top-P con x=topp, y=abstractness (score).

3.1.4 Accuracy

L'accuracy o precisione, è un valore compreso tra $[0, 1]$. Se elevato, indica una buona somiglianza del testo predetto con il riassunto di riferimento.

BeamSearch



]

Figura 3.25: Beam Search con x=beams, y=accuracy (score).

Diverse Beam Search

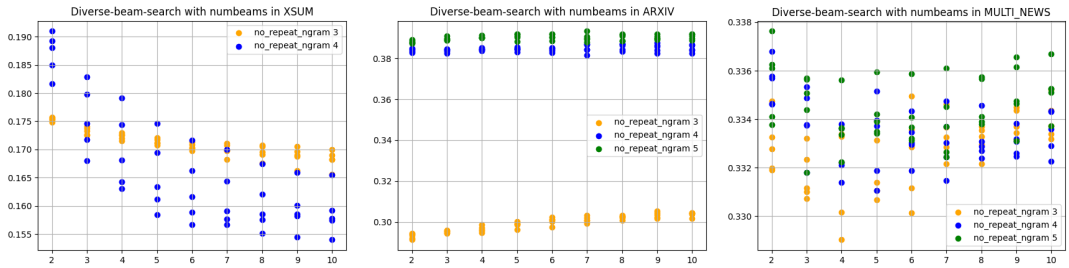


Figura 3.26: Diverse Beam Search con x=beams, y=accuracy (score).

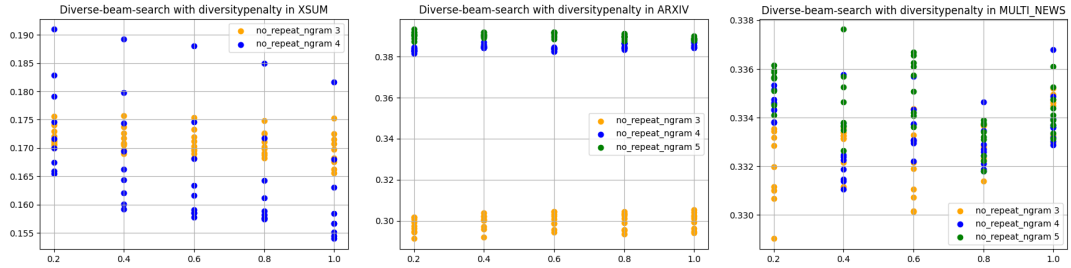


Figura 3.27: Diverse Beam Search con x=diversity penalty, y=accuracy (score).

Greedy

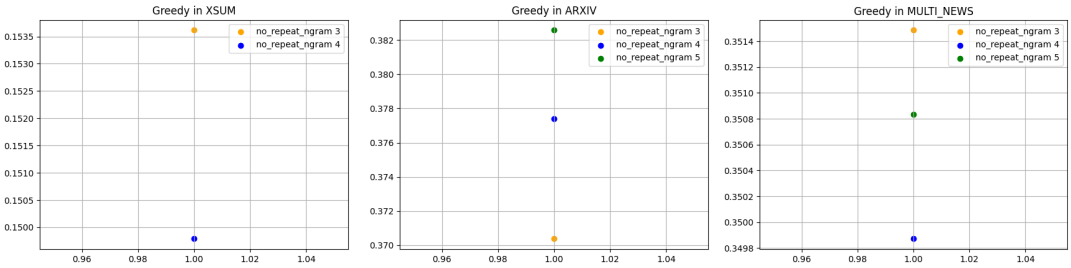


Figura 3.28: Greedy con x=costante 1, y=accuracy (score).

Sampling Temperature

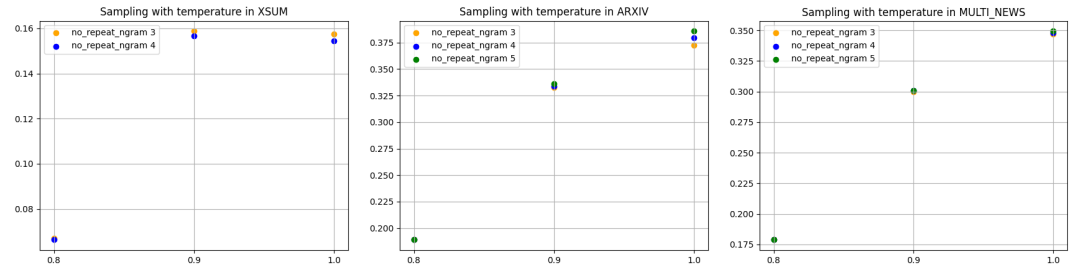


Figura 3.29: Sampling Temperature con x=temperatura, y=accuracy (score).

Sampling Top-K

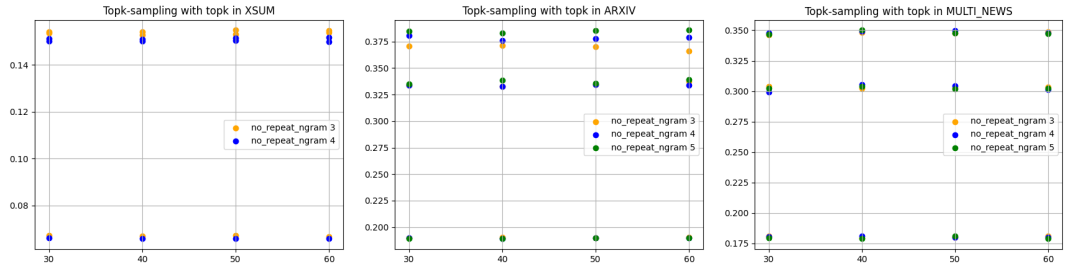


Figura 3.30: Sampling Top-K con x=topk, y=accuracy (score).

Top-P

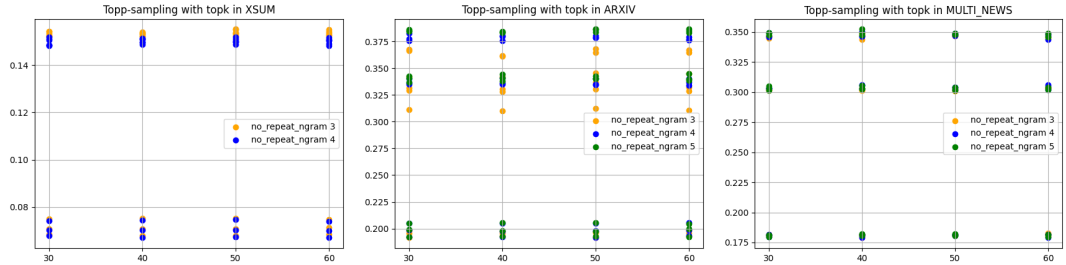


Figura 3.31: Sampling Top-P con x=topk, y=accuracy (score).

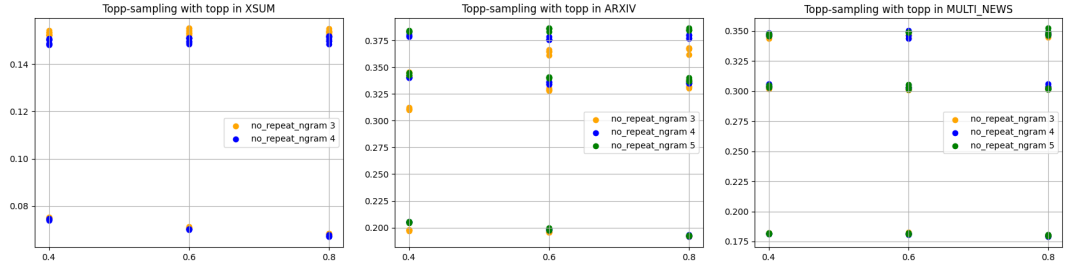


Figura 3.32: Sampling Top-P con x=topp, y=accuracy (score).

3.1.5 Repetitiveness

Il valore di ripetitività è il punteggio ottenuto applicando la sua formula. Più alto è il valore, più la predizione risulta essere ripetitività.

BeamSearch

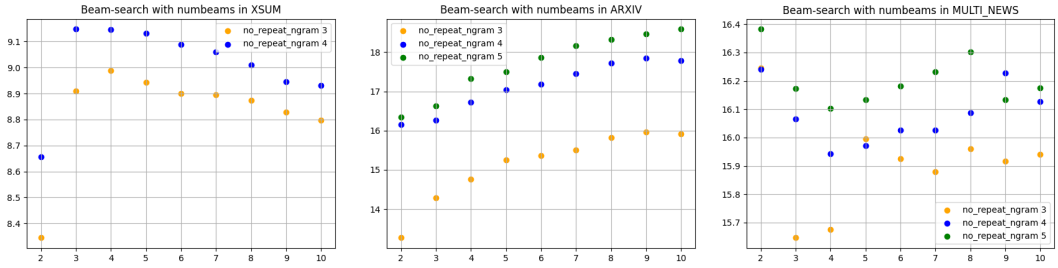


Figura 3.33: Beam Search con x=beams, y=repetitiveness (score).

Diverse Beam Search

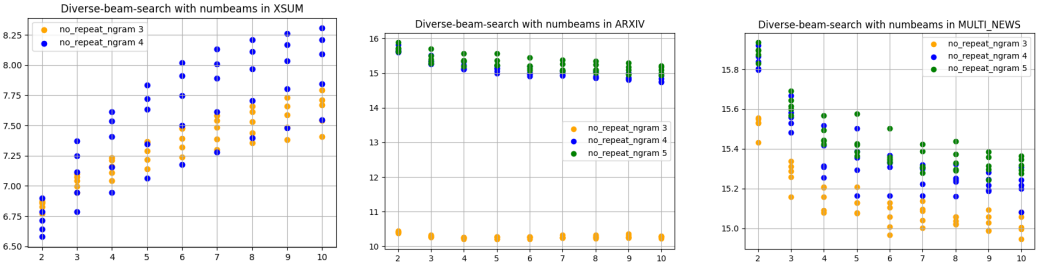


Figura 3.34: Diverse Beam Search con x=beams, y=repetitiveness (score).

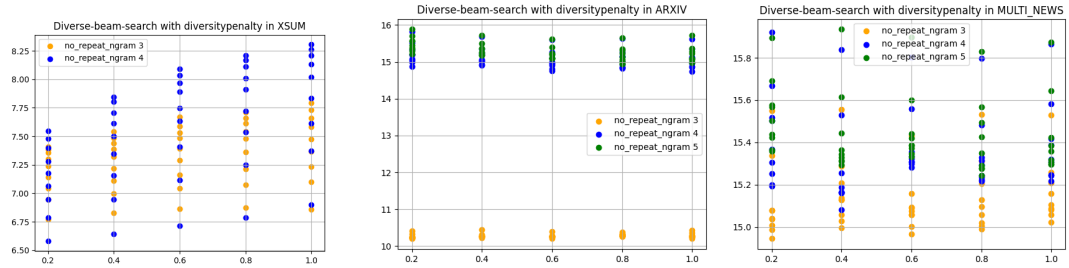


Figura 3.35: Diverse Beam Search con x=diversity penalty, y=repetitiveness (score).

Greedy

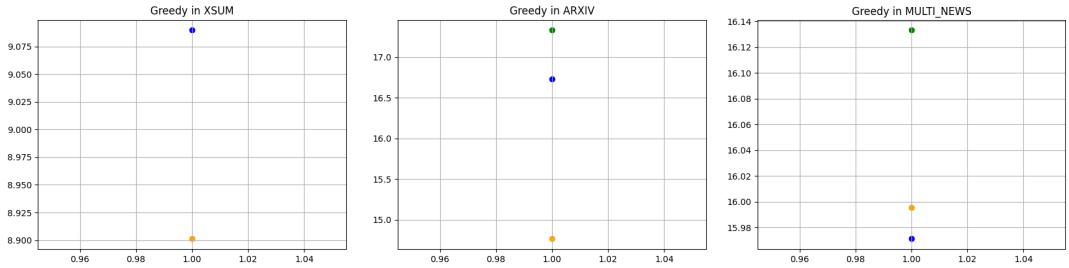


Figura 3.36: Greedy con x=costante 1, y=repetitiveness (score).

Sampling Temperature

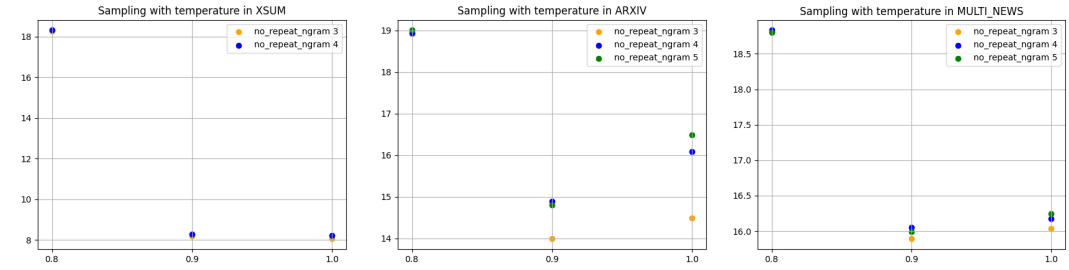


Figura 3.37: Sampling Temperature con x=temperatura, y=repetitiveness (score).

Sampling Top-K

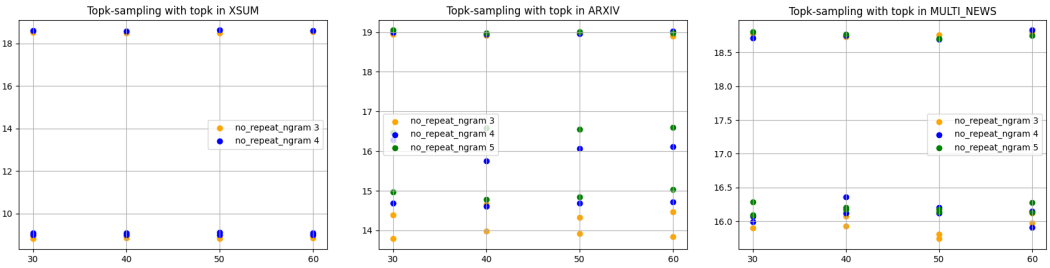


Figura 3.38: Sampling Top-K con x=topk, y=repetitiveness (score).

Top-P

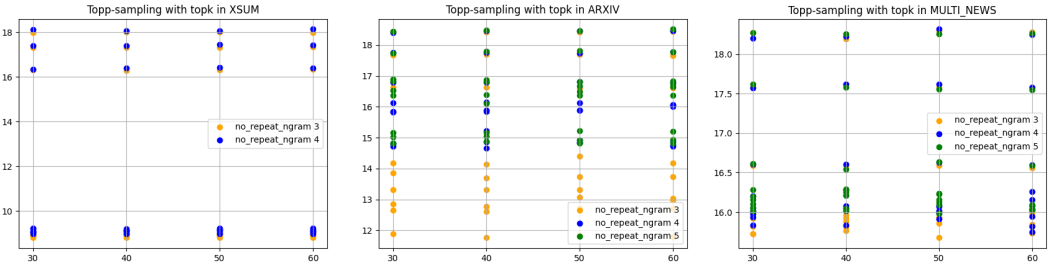


Figura 3.39: Sampling Top-P con x=topk, y=repetitiveness (score).

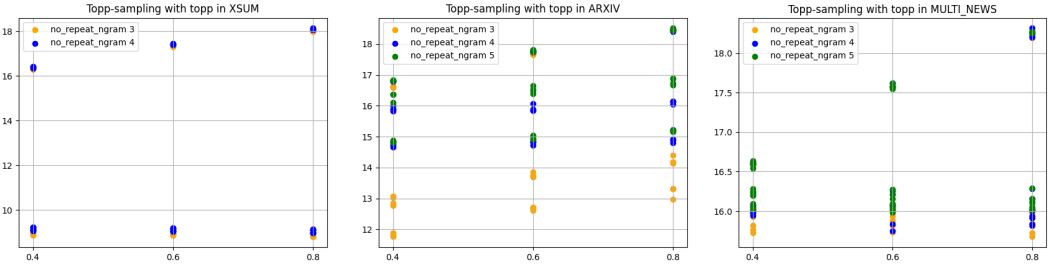


Figura 3.40: Sampling Top-P con x=topp, y=repetitiveness (score).

3.1.6 Riepilogo

Le generazioni sono state raggruppate e raffigurate in forma tabellare, mettendo in rilievo i risultati migliori per ogni metrica. Per ogni strategia, vengono mostrati gli iper-parametri e il dataset utilizzati.

La riga evidenziata mostra il risultato migliore ottenuto. Il valore **nrns** raffigura la dimensione di un n-gram che non può ripetersi per più di m volte durante la generazione (`no_repeat_ngram_size`).

ROUGE

Strategia	**Iper-Parametri**	**nrns**	**Dataset**	**Rouge (avg)**
Beam Search	beams=2	5	Arxiv	0.272836
Diverse Beam Search	beams=2, diversity=1.0	5	Arxiv	**0.277322**
Greedy	/	5	Arxiv	0.265433
Sampling	t=1.0	5	Arxiv	0.271179
Top-K Sampling	temp=1.0 topk=60	5	Arxiv	0.268477
Top-P Sampling	temp=1.0, topk=30, topp= 0.4	5	Arxiv	0.270832

Tabella 3.1: Risultati migliori con ROUGE.
$nrns = norepeatngramsize$

Perplexity

Strategia	**Iper-Parametri**	**nrns**	**Dataset**	**Perplexity (avg)**
Beam Search	beam=2	3	Multi-news	12.196444
Diverse Beam Search	beam=9, diversity=0.6	5	Multi-news	12.29375
Greedy	/	3	Multi-news	11.932968
Sampling	t=1.0	5	Multi-news	11.753861
Top-K Sampling	temp=0.8, topk=40	5	Multi-news	**11.753861**
Top-P Sampling	temp=0.8, topk=30, topp=0.8	3	Multi-news	11.845948

Tabella 3.2: Risultati migliori con Perplexity.
$nrns = norepeatngramsize$

Abstractivity

Strategia	Iper-Parametri	nrns	Dataset	Abstractivity (score)
Beam Search	beam=9	2	XSum	0.685752
Diverse Beam Search	beam=2, diversity=1.0	3	XSum	0.670371
Greedy	/	2	XSum	0.681966
Sampling	t=0.8	2	XSum	**0.887445**
Top-K Sampling	temp=0.8, topk=40	2	XSum	0.882577
Top-P Sampling	temp=0.8, topk=50, topp=0.8	2	XSum	0.880887

Tabella 3.3: Risultati migliori con Abstractivity.
$nrns = norepeatngramsize$

Repetitivity

Strategia	Iper-Parametri	nrns	Dataset	Repetitivity (score)
Beam Search	beam=6	2	XSum	8.344885
Diverse Beam Search	beam=2, diversity=0.2	3	XSum	**6.579189**
Greedy	/	2	XSum	8.901235
Sampling	t=0.8	3	Arxiv	18.923292
Top-K Sampling	temp=0.9, topk=50	2	XSum	8.808730
Top-P Sampling	temp=0.9, topk=60, topp=0.8	2	XSum	8.801764

Tabella 3.4: Risultati migliori con Repetitivity.
$nrns = norepeatngramsize$

Accuracy

Strategia	Iper-Parametri	nrns	Dataset	Accuracy (score)
Beam Search	beam=2	25	Arxiv	0.384569
Diverse Beam Search	beam=7, diversity=0.2	5	Arxiv	**0.393387**
Greedy	/	5	Arxiv	0.382578
Sampling	t=1.0	5	Arxiv	0.385617
Top-K Sampling	temp=1.0, topk=60	5	Arxiv	0.385685
Top-P Sampling	temp=1.0, topk=60 topp=0.6	5	Arxiv	0.386409

Tabella 3.5: Risultati migliori con Accuracy.
$nrns = norepeatngramsize$

3.1.7 Tempi di esecuzione

Sono stati riportati i tempi massimi e minimi per ogni strategia di decoding tenendo conto anche del dataset utilizzato.

Strategia	XSUM		ARXIV		MULTI-NEWS		T. medio
	Min	Max	Min	Max	Min	Max	
Beam Search	29min31s	53min28s	**19m06s**	34m53s	**18m47s**	25m59s	28m36s
Diverse Beam Search	23m42s	1h58m28s	24m55s	1h01m31s	27m43s	49m19s	40m31s
Greedy	24m43s	26m25s	22m41s	24m43s	20m27s	21m12s	**23m7s**
Sampling	**23m41s**	1h3m9s	20m11s	24m43s	18m12s	21m12s	32m13s
Top-K Sampling	24m18s	26m25s	24m47s	35m47s	23m29s	32m14s	29m53s
Top-P Sampling	25m33s	1h0m16s	25m20s	42m14s	25m43s	36m45s	36m45s

Tabella 3.6: In **grassetto** i tempi minimi e sottolineati i tempi massimi. T. medio equivale al Tempo medio di esecuzione.

3.1.8 Emissione di Co2 e energia generata

Sono stati riportati i valori totali di emissione di CO2 insieme all'energia generata. È stato possibile ottenere questi risultati grazie alla libreria `codecarbon` in grado di produrre ad ogni generazione un file *.csv* contenente le rilevazioni di energia, da cui è possibile calcolare le emissioni totali.

I valori totali ottenuti sono stati suddivisi nei due server del cluster utilizzati durante la sperimentazione.

Server	Energia generata	Emissioni di Co2
Faretra	155.3 kWh	**25.3 Kg**
cloudifaicdlw001	74.2 kWh	**11.8 Kg**

Tabella 3.7: A sinistra il nome esteso del server. Il valore in **grassetto** indica l'emissione di CO2 espressa in chilogrammi.

3.2 Findings

Una prima importante osservazione riguarda Beam Search, dove si osserva che non necessariamente l'aumento del numero di beam comporta un miglioramento della predizione. Si può notare nel nostro studio che nella generazione di short-document e long-document i risultati tendono a deteriorare con l'aumento del beam size. Questo dimostra l'incapacità dell'algoritmo di uscire da determinati cammini scelti, addentrandosi in un loop a cui non può sfuggire. Questa caratteristica—come è facile immaginarsi, all'aumentare del parametro, aumenta la ripetitività e diminuisce la perplessità. Questa informazione è stata rilevata anche nella ricerca Holtzman et al. [4]. Un nostro contributo aggiuntivo dimostra che nella generazione multi-document la situazione si capovolge. Dove all'aumentare dei beam aumenta la qualità del testo generato. Questo vuol dimostrare che durante la generazione è necessario tenere traccia di più token e

quindi avere una maggiore memoria predittiva quando si fa riferimento a più documenti non necessariamente collegati tra di loro.

La strategia Diverse Beam Search condivide le stesse caratteristiche del Beam Search per quanto riguarda la quantità di beam utilizzati. La situazione cambia quando si varia il parametro di penalità di diversità, mostrando un'oscillazione dei parametri che difficilmente è prevedibile a priori. Questo dimostra che non esiste una scelta empirica che garantisca un miglior risultato in senso assoluto. È possibile guadagnare qualità di predizione se scelto il parametro con criterio; soprattutto quando il documento presenta molte ridondanze oppure è molto simile nella sua struttura. In generale, avere una penalità di diversità bassa diminuisce inevitabilmente la ripetitività. Nel nostro studio, avere una elevata diversità ha portato a una migliore accuratezza e perplessità.

Il Greedy Search, nonostante la sua semplicità, porta a dei risultati modesti in rapporto alla sua velocità di generazione. Come si può intuire dalla natura dell'algoritmo, possiede la ripetitività più alta di tutti.

Il sampling si dimostra la strategia più stocastica di tutte, data dalla sua natura poco prevedibile. Questa caratteristica porta la strategia ad essere la migliore per quanto riguarda l'astrattività. Nel nostro studio gli estremi della temperatura hanno migliorato la perplessità. L'aggiunta di un ulteriore parametro k, che limita o amplifica la finestra dei token estratti, nel nostro caso non ha portato differenze significative. Questo potrebbe essere scaturito da un k troppo elevato.

Il Nucleus Sampling si differenzia principalmente dalle combinazioni del parametro topk e topp, andando ulteriormente a modificare la finestra di probabilità. La qualità di generazione è decisamente influenzata dalla combinazione di k e p. Un p basso con con qualsiasi valore di k trasforma l'output in un testo ripetitivo, declassandolo a un problema di sampling con temperatura bassa. Questo dimostra che una temperatura troppo bassa raramente porta a delle buone predizioni per tutti i tipi task di text summarization.

In tutte le strategie di decoding, evitare ripetizioni di n-gram nella generazione porta ad uno sbalzo non indifferente nella qualità di predizione. Ogni metrica presa in considerazione mostra che scegliere correttamente il **no-repeat-ngrams-size** comporti un miglioramento tassativo dell'output generato. Nel nostro studio, viene dimostrato che la limitazione dell'n-gram a non più di 3

ripetizioni per XSUM e non più di 5 per Arxiv e Multi-news aumenti fortemente le stime in tutte le metriche analizzate. Sulla base delle curve di performance registrate, i risultati ottenuti per Arxiv e Multi-news potrebbero migliorare ulteriormente mediante l'esplorazione di valori più elevati per il parametro in oggetto. Sorprendentemente, il **no-repeat-ngrams-size** si configura come uno degli iperparametri più impattanti sul riassunto astrattivo generato.

Un peso valutativo importante è il tempo di esecuzione di ogni strategia di decoding. Questo non solo permette di cogliere degli aspetti in più nella generazione, ma permette di ricondursi ad una maggiore consapevolezza nelle emissioni di CO2. Nel nostro studio è possibile comprendere dalla Tabella 3.6 che il Diverse Beam Search apporta il maggior consumo di tutti, mentre Greedy risulta il meno costoso. Ne consegue un trade-off tra qualità di generazione e prestazioni per Top-P Sampling, ma sopratutto Diverse Beam Search. Mentre la strategia Greedy risulta ottima se utilizzata come strategia iniziale valutativa per i suoi tempi di esecuzioni ridotti. Inoltre, un altro criterio correlato alle performance è la dimensione del dataset. La sua scelta, come naturalmente si può immaginare, influenza il tempo di predizione. Nel nostro caso di studio, il dataset XSUM nonostante sia decisamente più ristretto possiede il runtime maggiore di tutti.

Conclusioni

In questa tesi, viene proposto lo studio comparativo più comprensivo e aggiornato sull'efficacia e l'efficienza delle principali strategie di decoding per abstractive text summarization.

La realizzazione è stata affrontata attraverso i seguenti passi. In primo luogo sono state analizzate e valutate tutte le principali strategie di decoding. A seguito sono state presi in considerazione 3 modelli pre-addestrati allo stato dell'arte e 3 dataset legati a differenti task di abstractive text summarization, tra cui short-document, long-document e multi-document. Successivamente sono stati implementati ed eseguiti mediante la libreria Transformers di Hugging Face. Dalle generazioni, realizziamo un'analisi approfondita delle metriche valutative, temporali e di emissione di CO2.

I risultati ottenuti rivelano l'inesistenza di una strategia prevalente sulle altre. Questo dimostra che ogni strategia possiede determinate caratteristiche che si adattano meglio in base al task desiderato. Questo stimola la necessità di ponderare correttamente l'utilizzo di una strategia piuttosto che un'altra se si ricerca un risultato ottimale.

Il Beam Search e il Diverse Beam Search evidenziano una forte compatibilità nei task di multi-document, mentre nei task di short-document e long-document risultano apportare troppa ripetitività al testo all'aumentare dei beam. La strategia di Sampling, compresi di tutti i suoi iper-parametri dimostra di ottenere un risultato maggiormente inatteso, il quale potrebbe risultate il più umano di tutti nei task di short-document e multi-document. Un'altro aspetto importante è stato l'impatto causato dal parametro `no_ repeat_ngram_size` che risulta influire notevolmente tutti i task analizzati. Questo dimostra l'esigenza di dover prendere in analisi questo parametro per la generazione di testo.

In conclusione, la tesi mette in luce l'impatto qualitativo che si può ottenere nella scelta di una strategia di decoding nel generale predizioni di riassunti astrattivi. Queste informazioni possono contribuire ad un aumento nel settore

di ricerca che riguarda il decoding di task summarization portando a nuovi sviluppi con l'obiettivo di neutralizzare il divario esistente tra modelli e dataset in continua espansione.

Sviluppi futuri

Lo studio condotto innalza nuove questioni e quesiti che richiedono ulteriori approfondimenti. Inoltre, a causa dei tempi limitati, non è stato possibile analizzare determinati aspetti che potrebbero apportare un maggior contributo all'analisi. I principali obiettivi futuri sono di seguito riportati.

- Esplorare ulteriori variazioni del parametro k per la strategia Top-P Sampling e Top-K Sampling per verificare la natura poco influente del parametro.
- Aumentare il numero di `no_repeat_ngram_size` per i task di short e long document summarization nel tentativo di migliorare ulteriormente la qualità predittiva.
- Espandere lo spettro di modelli di abstractive text summarization per comprendere quanto siano correlati tra le tipologie di task e i dataset selezionati.
- Estrarre per ogni generazione le singole emissioni di CO2 e l'energia generata in modo da avere più dettagli sulle singole strategie.
- Aggiungere allo studio la strategia *Contrastive Beam Search*, il cui supporto è stato integrato nell'ultima settimana di tesi.
- Rivalutare le strategie che hanno ottenuto i migliori risultati non più singolo campione stratificato, ma sull'intero dataset per verificare se se effettivamente la scelta di una singola strategia possa determinare il raggiungimento di un nuovo stato dell'arte.

Ringraziamenti

Ringrazio chi mi è stato vicino, supportandomi e sopportandomi. Un ringraziamento anche ai miei colleghi: G. Frisoni e L.Ragazzi per avermi permesso di lavorare in questa meravigliosa ricerca ancora in continuo sviluppo.

Bibliografia

[1] Divakar Yadav, Jalpa Desai, and Arun Kumar Yadav. Automatic text summarization methods: A comprehensive review, 2022. URL `https://arxiv.org/abs/2204.01849`.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[3] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. A discourse-aware attention model for abstractive summarization of long documents. *arXiv preprint arXiv:1804.05685*, 2018.

[4] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration, 2019. URL `https://arxiv.org/abs/1904.09751`.

[5] Gian Wiher, Clara Meister, and Ryan Cotterell. On decoding strategies for neural text generators, 2022. URL `https://arxiv.org/abs/2203.15721`.

[6] Daphne Ippolito, Reno Kriz, Maria Kustikova, João Sedoc, and Chris Callison-Burch. Comparison of diverse decoding methods from conditional language models. *arXiv preprint arXiv:1906.06362*, 2019.

[7] Jingjing Xu, Wangchunshu Zhou, Zhiyi Fu, Hao Zhou, and Lei Li. A survey on green deep learning, 2021. URL `https://arxiv.org/abs/2111.05193`.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio,

Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

[9] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation, 2021. URL https://arxiv.org/abs/2106.11520.

[10] Anubhav Jangra, Adam Jatowt, Sriparna Saha, and Mohammad Hasanuzzaman. A survey on multi-modal summarization. *arXiv preprint arXiv:2109.05199*, 2021.

[11] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[12] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, mar 2020. doi: 10.1016/j.physd.2019.132306. URL https://doi.org/10.1016%2Fj.physd.2019.132306.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. URL https://arxiv.org/abs/1412.3555.

[15] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1419–1436, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.112. URL https://aclanthology.org/2021.naacl-main.112.

[16] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational

Linguistics. doi: 10.18653/v1/D18-1206. URL https://aclanthology.org/D18-1206.

[17] Vincent Chen. An examination of the cnn / dailymail neural summarization task. 2017.

[18] Colin B. Clement, Matthew Bierbaum, Kevin P. O'Keeffe, and Alexander A. Alemi. On the use of arxiv as a dataset, 2019. URL https://arxiv.org/abs/1905.00075.

[19] Anastassia Kornilova and Vladimir Eidelman. BillSum: A corpus for automatic summarization of US legislation. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 48–56, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5406. URL https://aclanthology.org/D19-5406.

[20] Eva Sharma, Chen Li, and Lu Wang. Bigpatent: A large-scale dataset for abstractive and coherent summarization, 2019. URL https://arxiv.org/abs/1906.03741.

[21] Xiachong Feng, Xiaocheng Feng, Bing Qin, Xinwei Geng, and Ting Liu. Dialogue discourse-aware graph convolutional networks for abstractive meeting summarization. *CoRR*, abs/2012.03502, 2020. URL https://arxiv.org/abs/2012.03502.

[22] Stergos Afantenos, Vangelis Karkaletsis, and Panagiotis Stamatopoulos. Summarization from medical documents: a survey. *Artificial Intelligence in Medicine*, 33(2):157–177, feb 2005. doi: 10.1016/j.artmed.2004.07.017. URL https://doi.org/10.1016%2Fj.artmed.2004.07.017.

[23] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018. URL https://arxiv.org/abs/1801.10198.

[24] Stefanos Angelidis and Mirella Lapata. Summarizing opinions: Aspect extraction meets sentiment prediction and they are both weakly supervised. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3675–3686, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1403. URL https://aclanthology.org/D18-1403.

[25] Odellia Boni, Guy Feigenblat, Guy Lev, Michal Shmueli-Scheuer, Benjamin Sznajder, and David Konopnicki. Howsumm: A multi-document summarization dataset derived from wikihow articles, 2021. URL https://arxiv.org/abs/2110.03179.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.

[27] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/v1/d14-1179. URL https://doi.org/10.3115/v1/d14-1179.

[28] Barak Turovsky. Found in translation: More accurate, fluent sentences in google translate. URL https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/.

[29] Felipe Almeida and Geraldo Xexéo. Word embeddings: A survey, 2019. URL https://arxiv.org/abs/1901.09069.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.

[31] Sam Wiseman and Alexander M. Rush. Sequence-to-sequence learning as beam-search optimization. *CoRR*, abs/1606.02960, 2016. URL http://arxiv.org/abs/1606.02960.

[32] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015. URL https://arxiv.org/abs/1508.04025.

[33] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[34] XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, sep 2020. doi: 10.1007/s11431-020-1647-3. URL https://doi.org/10.1007%2Fs11431-020-1647-3.

[35] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

[36] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization, 2019. URL https://arxiv.org/abs/1912.08777.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. URL https://arxiv.org/abs/1810.04805.

[38] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2019. URL https://arxiv.org/abs/1910.10683.

[39] Dario Amodei. Ai and compute, Jun 2021. URL https://openai.com/blog/ai-and-compute/.

[40] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[41] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2016. doi: 10.48550/ARXIV.1603.06560. URL https://arxiv.org/abs/1603.06560.

[42] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics, 2017. doi: 10.18653/v1/w17-3207. URL https://doi.org/10.18653%2Fv1%2Fw17-3207.

[43] Liang Huang, Kai Zhao, and Mingbo Ma. When to finish? optimal beam search for neural text generation (modulo beam size), 2018. URL https://arxiv.org/abs/1809.00069.

[44] Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R. Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models, 2016. URL https://arxiv.org/abs/1610.02424.

[45] Ashwin Vijayakumar, Michael Cogswell, Ramprasaath Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search for improved description of complex scenes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[46] André Cibils, Claudiu Musat, Andreea Hossman, and Michael Baeriswyl. Diverse beam search for increased novelty in abstractive summarization. *arXiv preprint arXiv:1802.01457*, 2018.

[47] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1045. URL https://aclanthology.org/D18-1045.

[48] 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.

[49] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[50] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation, 2019. URL https://arxiv.org/abs/1909.05858.

[51] Rémi Leblond, Jean-Baptiste Alayrac, Laurent Sifre, Miruna Pislar, Jean-Baptiste Lespiau, Ioannis Antonoglou, Karen Simonyan, and Oriol Vinyals. Machine translation decoding beyond beam search. *arXiv preprint arXiv:2104.05336*, 2021.

[52] Wen Xiao, Iz Beltagy, Giuseppe Carenini, and Arman Cohan. Primera: Pyramid-based masked sentence pre-training for multi-document summarization, 2021. URL https://arxiv.org/abs/2110.08499.

[53] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

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

[55] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.