UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

REZUMAREA AUTOMATĂ A TEXTELOR ÎN LIMBA ROMÂNĂ

Absolvent
Petcu Mircea

Coordonator științific
Prof. dr. Radu Ionescu

București, iunie 2024

Rezumat

Timpul limitat pe care îl avem la dispoziție ne constrânge cantitatea de informație pe care o putem citi și asimila într-un interval limitat de timp. Pentru rezolvarea acestei probleme, trebuie să venim cu soluții eficiente pentru a cuprinde cele mai importante informații în cel mai scurt timp.

În această lucrare, propun o metodă de rezumare de texte pentru limba română bazată pe Învățare Automată. Principala cerința în Învățare Automata o reprezintă volumul mare de date necesar pentru antrenarea modelelor. În acest context, am realizat un set de date ce constă în știrile și rezumatele corespunzătoare unor publicații online de presă, cu ajutorul căruia am antrenat un model bazat pe arhitectura Transformer, pe care îl rafinez ulterior cu Direct Preference Optimization, în funcție de preferințele umane.

Problema modelelor actuale de limbaj este faptul că numărul de token-uri care se pot procesa deodată este limitat din considerente de memorie și putere computațională. Prin urmare, deseori se recurge la trunchierea textului la un număr anume de token-uri, astfel, pierzând din informație. Așadar, propun o metodă de preprocesare cu TextRank pentru păstrarea celor mai importante propoziții in procesul de trunchiere.

Abstract

The limited time at our disposal constrains our quantity of information that we can read and assimilate in a limited amount of time. To solve this problem, we need to come up with effective solutions to capture the most important information in the shortest time.

In this paper, we propose a text summarization method for Romanian language based on Machine Learning. The main requirement in Machine Learning represents the large amount of data required to train the models. Hence, we created a data set that consists of the news and their summaries from online press publications, with which we trained an architecture-based Transformer model, which we later refine with Direct Preference Optimization, with respect to human preferences.

The problem with current language models is that the number of tokens that can be processed at once is limited due to memory and computational power considerations. Thus, it is often used to truncate the text to a specific number of tokens, losing information. So, we propose a preprocessing method with TextRank to preserve those more important sentences in the truncation process.

Cuprins

Listă de figuri

Listă de tabele

Capitolul 1

Introducere

1.1 Scopul și motivația

Mă consider o persoană cu o dorință mare de cunoaștere. În același timp, consider că nu am timpul necesar pentru a afla tot ce mă interesează. Așadar, trebuie sa găsesc alternative eficiente pentru a rezolva aceasta problemă de constrângeri. În acest sens, încerc sa extrag cele mai bune informații dintr-o sursă, fără să fiu nevoit sa parcurg tot conținutul. Prima variantă la care mă gândesc cand încerc să abordez această problemă este sa extrag doar prima parte dintr-un text sau să fac o extragere vizuală de entități cu tot cu propozițiile în care se află.

Abordarea manuală nu este de cele mai multe ori si eficace in practica deoarece timpul necesar extragerii informației dintr-un text variază în funcție de lungimea pe care o are. Soluția acestei probleme o găsesc in aplicarea tehnicilor de Învățare Automată in vederea sumarizării automate. Astfel, scopul lucrării de licența va fi de a găsi o metodă propice pentru capturare celor mai importante informații dintr-un text și compresia acestuia cat mai intensă și eficientă, păstrând in același timp logica si coerența textului.

1.2 Rezumarea automată de texte

Rezumarea automată de texte, ca și abordare, poate fi împărțită in două secțiuni diferite: sumarizare extractivă și sumarizare abstractivă. Sumarizarea extractivă face referire la extragerea celor mai importante propoziții dintr-un text, astfel rezumatul rezultat din acest proces se va regăsi, fragmentat sau nu, in textul original. Sumarizarea abstractivă, pe de alta parte, se concentrează pe extragerea informației în sine si are libertatea de a modela rezumatul astfel încât textul rezultat sa aibă o coerență cât mai mare, o logică cât mai bine definită si informația sa fie expusă cât mai cuprinzător posibil, dar în același timp într-un format condensat.

De asemena, acest domeniu se mai împarte în două categorii: rezumarea unui singur

document sau a mai multor documente într-un singur sumar. Prima categorie se concentrează asupra sumarizării unui singur text într-unul simplificat, în timp ce cea de a doua se axează pe compresia mai multor texte într-un singur rezumat.

Tipul abordării mai poate fi clasificat și după mărimea textului ce trebuie simplificat. Astfel, pentru un text mai lung cum ar fi o carte de 200 de pagini, metoda pentru sumarizarea automată va fi cu totul diferită față de un text mai scurt, spre exemplu, un articol de știri sau un text și mai scurt, cum ar fi o recenzie.

1.3 Repere istorice

Rezumarea automată de texte este un subiect intens discutat care a fost cercetat în mai multe lucrări pe această temă. Pentru limba engleza există seturi de date consacrate cu sute de mii de exemple cum ar fi CNN/DailyMail [1] și XSum [2] care reprezintă o resursă valoroasă pentru modelele dezvoltate pe această limbă.

În mod clasic, rezumarea automată de texte, folosind Învățarea Automata, se realiza cu ajutorul Rețelelor Neuronale Recurente în special folosind modelul seq2seq [3] ce are o structura de tip encoder-decoder. Odată cu introducerea arhitecturii Transformer, acest domeniu a cunoscut o creștere semnificativă în progres. Datorită cantității mari de date pentru limbile cu resurse mari, s-au putut dezvolta și pre-antrena modele de limbaj supradimensionate și potente, unele dintre ele respectând niște arhitecturi îndreptate către această cerință [4]. Aceste modele au fost mai apoi ajustate specific pentru sumarizarea de texte cu ajutorul seturilor de date menționate anterior (sau a altor seturi de date).

1.4 Contribuția proprie

În această lucrare propun un set de date realizat în aceeași manieră ca și [5] de aproximativ 25 de mii de exemple extrase în mod automat, ce reprezintă perechi de știri si rezumatele acestora scrise în limba română. Prin intermediul acestor date, am antrenat în mod supervizat modele bazate pe arhitectura Transformer, am experimentat impactul preprocesării TextRank asupra calității rezumatelor generate, urmând ca în ultima parte a lucrării să experimentez alinierea celui mai bun model la preferințele umane, folosind Direct Preference Optimization. Totodată, vin cu o aplicație web care expune în mod practic cel mai bun model pentru inferență.

1.5 Structura lucrării

În cele ce urmează, voi prezenta fiecare capitol în parte, oferind informații descriptive despre ideea principală ce reiese din acestea.

1. Capitolul 1, prezentul capitol, introduce lucrarea de licență, oferind detalii despre scopul și motivația din spatele alegerii temei, prezentarea domeniului din care face parte, repere istorice relevante cu privire la această temă și prezintă pe scurt contribuția adusă de mine.

2. Capitolul 2 intră în detalii cu privire la conceptele teoretice care stau la baza lucrării de licență, tehnologiile folosite pentru realizarea acesteia și abordări recente cu privire la tema aleasă cu accent pe rezumarea automată a textelor în limba română.

3. În capitolul 3 se pot regăsi informații detaliate cu referire la construirea setului de date, precum colectarea știrilor și curățarea textelor.

4. Capitolul 4 expune procesul efectuării experimentului care studiază preprocesarea cu TextRank a datelor, înainte de propagarea acestora prin model.

5. Capitolul 5 prezintă antrenarea în mod supervizat a unor modele și evaluarea acestora, urmând ca finalul capitolului să fie reprezentat de construirea setului de date pentru antrenarea cu Direct Preference Optimization și efectuarea acesteia.

6. În capitolul 6 este prezentată aplicația web asociată inferenței prin modelul specializat pentru rezumare automată de texte.

7. Capitolul 7 transmite o concluzie dobândită după realizarea acestei lucrări, limitările întâlnite de-a lungul acesteia și posibile dezvoltări ulterioare.

Capitolul 2

Preliminarii

2.1 Concepte teoretice

2.1.1 Transformer

Piatra de temelie a marilor modele actuale de limbaj o reprezintă cu certitudine arhitectura Transformer. Această lucrare [6] revoluționară a adus cu sine un alt mod de a percepe atenția în contextul rețelelor neuronale. Față de precedentele arhitecturi care abordau problema procesării limbajului natural, ce erau bazate pe rețele neuronale recurente, acestea au marele avantaj de a putea paraleliza foarte multe operații și astfel pot să fie antrenate pe mai multe GPU-uri sau chiar mai multe cluster-e de GPU-uri (unități de procesare specializate pe operații simple ce pot paraleliza foarte multe operații dintr-o dată).

Unul dintre principalele avantaje pe care l-a adus această arhitectură este posibilitatea de a procesa mai multe token-uri deodată, fără să mai fie un proces secvențial că în cazul rețelelor recurente. Acest lucru este realizat în special prin reprezentările pozițiilor în format vectorial. În articolul original este prezentată ca variantă finală pentru încorporările pozițiilor cu o abordare pe bază de funcții sinusoidale. În esență, există mai multe valori date de funcții sinus și cosinus cu diferiți parametrii pentru a oferi o reprezentare a pozițiilor cât mai precisă. Acest tip de funcții a fost considerat potrivit datorită proprietăților sale matematice, cum ar fi mărginirea pentru stabilitatea interpolării cu reprezentările semantice sau nesaturarea funcției pentru valori mai mari sau mai mici. Acestea se adaugă la reprezentările semantice, care la rândul lor sunt reprezentate de un strat complet conectat ce are ca dimensiune a datelor de intrare lungimea vocabularului, iar numărul de neuroni este așa zisa dimensiune a modelului.

Mecanismul de atenție este unul revoluționar pentru domeniu și este format dintr-o suită de sub-matrici: cheie (key), interogare (query) și valoare (value). Atenția este efectuată pentru fiecare token în parte folosind informație din toate celelalte token-uri. Pe scurt, fiecare încorporare (embeddings) este ajustată cu o porțiune din celelalte încorporări

ale celorlalte token-uri.

Procedeul descris se realizează la nivelul unui cap (head) de atenție, dar în realitate pentru o performanță mai bună există mai multe capete care sunt concatenate între ele. Matricea obținută este înmulțită cu o alta de ieșire (output). Mecanismul de atenție descris mai sus se numește self-attention, însă există și cross-attention, care reprezintă atenția decoder-ului asupra encoder-ului, și atenție auto-regresivă, ce se diferențiază prin mascarea token-urile următoare din secvență, în cazul decoder-ului. Pe lângă reprezentările semantice de la început, reprezentările pozițiilor și mecanismul de atenție, în arhitectura Transformer există și straturi de normalizare pentru stabilizarea gradientului, și două straturi complet conectate pentru specializare cu o funcție de activare după fiecare dintre ele și încă un strat de normalizare la final.

Arhitectura originală Transformer este formată dintr-un encoder și un decoder, fiind un model pentru traducere automată. Astfel, un bloc de encoder este format de stratul de intrare, reprezentările pozițiilor, mecanismul self-attention, un strat de normalizare, două straturi complet conectate, și încă un strat de normalizare la final. Totodată, între aceste straturi există conexiuni reziduale pentru propagarea lină a gradientului, rețeaua fiind una adâncă. Blocul de Decoder este foarte asemănător cu blocul de Encoder. Diferență o face faptul că atenția este auto-regresivă și există și mecanismul de cross-attention, ce se folosește de reprezentările Encoder-ului, iar în capăt se află un strat complet conectat asemănător cu stratul pentru datelor de intrare ce are ca dimensiune de ieșire lungimea vocabularului. De asemenea, este de menționat că textul de ieșire este mutat spre dreapta pentru a putea fi prezis următorul token, nu același token. De obicei, se scaleaza cu mai multe blocuri de Encoder și Decoder pentru a crește capabilitățile modelului. Din această arhitectură s-au dezvoltat ulterior foarte multe modele puternice și specializate pentru diverse sarcini.

2.1.2 TextRank

TextRank [7] este un algoritm bazat pe grafuri pentru rezumare extractivă de texte și extragere de cuvinte cheie. Acesta este foarte influențat de PageRank [8], algoritm pentru navigarea eficientă pe internet. Algoritmul funcționează în următorii pași: setează nodurilor grafului cu unități de text specifice pentru cerința în speță, conectează nodurile prin muchii, reprezentate de relațiile dintre aceste unități, iterează formula de scor până la convergență (scorurile a două iterații consecutive nu se mai modifică aproape deloc) și sortează nodurilor în funcție de scorurile obținute la final. În cazul sumarizării extractive, nodurile sunt reprezentate de propozițiile documentului ce trebuie sumarizat iar muchiile sunt reprezentate de o măsură de similaritate între propoziții, numărul de token-uri suprapuse dintre 2 noduri adiacente, normalizat cu numărul total de token-uri din fiecare nod, fiind propusă în lucrarea originală. De asemenea, în lucrare este menționat că se pot

utiliza și alte metrici. Se rulează algoritmul până la convergență și se retrag propozițiile în ordinea descrescătoare a scorului obținut.

2.1.3 Cosine Annealing

Cosine Annealing [9] este un learning rate scheduler (metodă de programare a ratei de învățare) care presupune scăderea treptată a ratei de învățare de la valoarea maximă setată la cea minimă, respectând traiectoria unei funcții cosinus. Această metodă poate avea mai multe cicluri, oscilând între valoarea maximă și minimă a ratei de învățare sau poate avea doar unul, sau chiar și jumătate de ciclu. Aceste creșteri ale ratei de învățare pot ajuta algoritmul de optimizare să treacă peste minime locale, iar în ultimi pași de antrenare, având o rată de învățare mai mică, poate ajuta ca funcția de pierdere să nu oscileze foarte mult în partea finală a antrenării și să găsească un optim mai bun.

2.1.4 Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension (BART)

Modelul BART [10] se bazează pe arhitectura Transformer descrisă anterior cu mici diferențe. Față de arhitectura originală, aceasta folosește reprezentări parametrizate ale pozițiilor, care sunt învățate în procesul de antrenare. De asemenea, acesta folosește activări de tip GELU [11] în loc de ReLU după straturile complet conectate de după mecanismul de atenție. Modelul de bază are 6 blocuri de Transformer (Encoder și Decoder), iar cel mare are 12 blocuri de Transformer. Arhitectura acestuia are marele avantaj de a realiza sarcini cu succes, atât discriminative, cât și generative. Autorii au realizat o suită de metodologii de pre-antrenare. Aceștia au utilizat:

- prezicerea token-urilor mascate
- ștergerea aleatoare ale unor token-uri, urmând ca modelul să decidă pe ce poziții se aflau acestea
- prezicerea unor secvențe de token-urilor mascate de lungime variabilă
- permutarea aleatoare a propozițiilor, modelul trebuind să le pună în ordine
- rotația documentului, având ca pivot un token aleator, modelul urmând să prezică token-ul de început al documentului

2.1.5 Multilingual Denoising Pre-training for Neural Machine Translation (mBART)

Varianta multilingvistică a modelului BART este o extindere a acestuia pentru mai multe limbi. Autorii propun în [12] mai multe modele pre-antrenate pe diferite mulțimi de limbi. În această lucrare, voi folosi modelul pre-antrenat pe 25 de limbi, incluzând limba română. Acesta a fost antrenat pe corpusul CC25, care include aproximativ 63 GB de date în limba română. Arhitectura modelului constă în 12 blocuri de Encoder și 12 blocuri de Decoder cu un strat de normalizare la finalul blocului. Dimensiunea latentă a modelului este de 1024 și are aproximativ 680 de milioane de parametrii.

2.1.6 Text-To-Text Transfer Transformer (T5)

Modelul T5 [13] se bazează pe arhitectura Transformer descrisă anterior, cu mici diferențe. Față de arhitectura originală, aceasta folosește reprezentări relative ale pozițiilor [14], ce ajută la antrenarea pe secvențe mai lungi de text dar, în același timp, adaugă un număr semnificativ de parametrii. De asemenea, T5 elimină parametrii de bias din straturile de normalizare.

Lucrarea a mai adus și un corpus numit „Colossal Clean Crawled Corpus" (C4), care este folosit pentru pre-antrenarea modelului. Modelul a fost atât pre-antrenat auto-supervizat (self-supervised) pe date neetichetate prin mascarea a `15%` din token-urile secvenței de intrare și prezicerea acestora de către rețea, cât și în mod supervizat, pentru mai multe sarcini simultan, folosind prefixe ce indică sarcina curentă. Numele vine din faptul că modelul tratează orice sarcină ca una de la text la text (text-to-text), inclusiv regresia.

2.1.7 Gradient checkpointing

Gradient checkpointing [15] este o tehnică ce ajută la antrenarea modelelor mari care nu se pot antrena cu resurse limitate din considerente de memorie. Această tehnică se bazează pe ștergerea unor anumite activări ale neuronilor pentru a salva memorie. Deși poate economisi foarte multă memorie, vine la pachet cu o creștere intensă în timpul necesar antrenării. Așadar, Gradient checkpointing găsește o strategie pentru a avea un echilibru între necesarul de memorie și necesarul de timp computațional. Astfel, se păstrează doar un subset de neuroni care își vor păstra activarea după propagarea înainte a rețelei neuronale. Doar nodurile critice (de articulație) își păstrează activarea după propagarea înainte, restul activărilor se șterg și se recalculează o dată cu propagarea înapoi, când sunt necesare. După pasul prin nodurile grafului computațional din propagarea înapoi, se șterg atât activările nodurilor critice, cât și a celorlalte noduri. Se păstrează în memorie doar activările nodurilor necesare în procesul de calcul al gradienților și a

nodurilor critice (până ce se trece de ele în propagarea înapoi).

2.1.8 Acumularea gradientului

Acumularea gradientului este o tehnică pentru antrenarea modelelor de Învățare Automată cu un batch mai mare decât ar încăpea în mod normal în memorie. Procedeul presupune propagarea înainte și înapoi a unui batch de mărime mai mică și stocarea gradienților calculați în memorie fară să se actualizeze parametrii înglobați în procesul de optimizare. O dată cu acumularea unui număr de iterații setat, se actualizează și valorile parametrilor, doar că sunt actualizați în concordanță cu gradienții exemplelor acumulate și nu doar a exemplelor din pasul curent. După actualizare, gradienții se șterg din memorie. Aceasta implică o economie de memorie datorită faptului că activările și datele de intrare nu sunt ținute în memorie deodată, ci secvențial.

2.1.9 Categorii de modele de limbaj bazate pe arhitectura Transformer

Ca arhitectură, modelele mari de limbaj se pot împărți în trei mari categorii:

- Encoder-Decoder: este forma pe care o avea și modelul Transfomer și este concepută dintr-un Encoder ce are atenție bidirecțională (self-attention), astfel poate să creeze legături atât pentru token-urile ce sunt poziționate înaintea token-ului curent, cât și după acesta, și dintr-un Decoder ce are atenție auto-regresivă (masked self attention), astfel token-urile poziționate după token-ul curent sunt mascate și nu influențează încorporarea acestuia. De asemenea, există și un mecanism de atenție între Encoder și Decoder numită Cross-Attention care ajută Decoder-ul cu informații încorporate în Encoder. Această arhitectură este de obicei utilizată în sarcini ca traducere automată și sumarizare abstractivă de texte, deoarece sunt concepute pentru generarea de secvențe de lungime diferită față de secvențele de intrare și îmbină atât calitățile Encoder-ului de a avea o înțelegere amplă și bidirecțională a textului de intrare, cât și calitățile Decoder-ului de generare auto-regresivă de text. Ca exemple pot aminti T5 , BART sau Transformer.

- Decoder-only: această arhitectură include doar Decoder-ul din arhitectura clasică și este potrivit pentru sarcinile ce constau în generarea foarte bună de texte, cum ar fi răspunderea la diverse întrebări primite sau generarea de text coerent. Exemple de modele sunt GPT2 [16] sau LLaMA [17].

- Encoder-only: aceste modele sunt specializate pe înțelegerea pe deplin a textului primit ca date de intrare și sunt îndreptate mai degrabă către sarcini de clasificare de texte sau recunoaștere de entități din texte. Aici, pot menționa ca exemple BERT [18] sau ALBERT [19].

2.1.10 Recall-Oriented Understudy for Gisting Evaluation (ROUGE)

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) reprezintă un set de metrici propus în [20] conceput special pentru a evalua calitatea rezumatelor generate în procesul de sumarizare automată. În esență, aceste metrici se concentrează pe calcularea suprapunerii rezumatului generat cu cel de referință, cu diverse formule pentru a induce o concluzie cât mai clară a calității rezumatului propus. Pachetul conține mai multe implementări ale acestei idei care denotă mai multe perspective:

- ROUGE-N reprezintă suprapunerea n-gramelor din rezumatul de referință raportat la cel generat. Aceasta se regăsește sub diferite forme, cele mai întâlnite fiind ROUGE-1 (suprapunerea peste uni-grame), ROUGE-2 (suprapunerea peste bi-grame), dar se poate extinde și la 3-grame sau mai mult. Se împarte în 3 categorii: precizia ce denotă numărul de n-grame regăsite atât în textul țintă, cât și în cel de referință raportat la numărul de n-grame din textul țintă, recall ce se raportează la numărul de n-grame din rezumatul de referință și f1-score, o combinație dintre cele două.
- ROUGE-L reprezintă cea mai lungă subsecvență, nu neapărat continuă, care se regăsește atât în rezumatul de referință, cât și în cel generat, împărțită la numărul de cuvinte/token-uri. Aceasta are trei sub-formule separate: precizia ce se referă la raportul dintre lungimea celei mai lungi subsecvențe și numărul de cuvinte din textul generat, recall, care față de precizie, se raportează la numărul de cuvinte din textul de referință și f1-score care reprezintă o combinație dintre cele două amintite.
- ROUGE-W ajustează ROUGE-L prin promovarea subsecvențelor mai lungi în favoarea celor mai scurte, astfel reprezintă o metrică mai adecvată pentru coerența textului.
- ROUGE-Lsum, care față de ROUGE-L, este o metrică care se calculează la nivel de propoziție, nu la nivelul întregului text.
- Plus alte variațiuni ale acestor metrici.

2.1.11 BERT Score

Autorii lucrării [21] admit că BERT Score aduce o mai bună încorporare semantică a calității rezumatelor generate. În această metodă, atât token-urile din rezumatul de referință, cât și cel generat se propagă înainte printr-un model de limbaj, de obicei BERT care va produce niște încorporări (embeddings) pentru ambele cazuri. Se calculează similaritatea cosinus între toate token-urile din textul de referință și cel generat. Formula finală încorporează și un parametru de inversul numărului de apariții a token-ului în document pentru a se face distincția între token-uri rare și cele mai frecvente. Pentru fiecare token din rezumatul de referință (în cazul recall) sau din rezumatul generat (în

cazul preciziei), se alege cel mai mare scor care se ajustează cu parametrul menționat anterior. În final, se deduce un scor care denotă cât mai bine calitatea rezumatului generat raportat la cel de referință.

2.1.12 AdamW

AdamW [22] vine cu o ajustare pentru algoritmul de optimizare Adaptive Moment Estimation (Adam) [23]. Adam adaugă la algoritmul clasic de Stochastic Gradient Descent atât momentum, care ia în calcul valorile gradienților anteriori, cât și rata de învățare adaptivă, care ajustează rata de învățare pentru fiecare parametru în calcul cu ajutorul valorilor gradienților anteriori, făcând astfel posibilă modificarea mai accentuată a parametrilor al căror gradient are o magnitudine mai mică, și diminuând modificarea acelora al căror gradient este mai mare. Astfel, se asigură o actualizare echilibrată a gradienților cu ajutorul ratei de învățare adaptive, iar oscilațiile în actualizările parametrilor sunt diminuate datorită momentului, ducând într-un final la o convergență mai rapidă. Deși se poate aplica regularizare și în cazul algoritmului Adam clasic, acesta normalizează penalizarea parametrilor cu termenul de adaptarea a ratei de învățare deoarece regularizarea este adunată gradientului de la pasul curent, astfel, dacă un parametru a avut în trecut mulți gradienți cu magnitudine mare, efectul benefic al regularizării vă fi suprimat de normalizarea menționată. AdamW aduce o modificare a algoritmului în ceea ce privește acest aspect, acesta scoate regularizarea parametrilor din raza de acțiune a normalizării înafară acesteia, astfel, că regularizarea este aplicată direct în loss ca un termen separat și este înmulțit doar cu rata de învățare.

2.1.13 Low-Rank Adaptation of Large Language Models (LoRA)

Această tehnică de antrenare a modelelor de Învățare Automată a fost propusă în [24] și constă în ajustarea unui subset mai mic de parametrii din totalul acestora. Aceasta se bazează pe ipoteza că nu toți parametrii sunt în egală măsură la fel de importanți, iar impactul pe care îl poate avea o antrenare ulterioară asupra unui model pre-antrenat poate fi încapsulat într-o reprezentare mult mai restrânsă a matricii de parametrii. Este aplicată în special pentru straturile complet conectate, unde numărul de parametrii este considerabil în comparație cu alte tipuri de straturi. Cum modelele bazate pe arhitectura Transformer au foarte multe straturi complet conectate, tehnica se aplică cu precădere în antrenarea acestor modele. Autorii au demonstrat că antrenarea modelelor cu această tehnică este cel puțin echivalentă cu antrenarea tuturor parametrilor modelelor sau a ultimului strat de clasificare de la final, dar cu o eficiență ridicată în ceea ce privește resursele hardware. LoRA se realizează prin înghețarea tuturor parametrilor pre-antrenați (parametrii nu mai sunt modificați prin scăderea gradientului înmulțit cu rata de învățare) și antrenarea a unor adapteri formați din două matrici, de exemplu A și B (numite și LoRA

adapters), ale căror înmulțire rezultă într-o matrice cu aceleași dimensiuni ca matricea parametrilor înghețată (W). Pentru a putea fi înmulțite matricile A și B, trebuie sa aibă aceeași dimensiune pentru una dintre cele două dimensiuni. Această dimensiune intrinsecă se numește rang. Practic, antrenarea se reduce la optimizarea unui subset mult mai mic de parametrii al căror număr este dat de rangul setat. Pe lângă avantajul evident pe care îl aduce în ceea ce privește antrenarea, aceasta mai aduce și alte beneficii. Asupra modelului pre-antrenat se pot aplica mai multe matrici (LoRA adapters) diferite, pentru fiecare cerință în parte, astfel, se păstrează parametrii modelului pre-antrenat și se modifică doar matricile pentru cerința respectivă.

2.1.14 Efficient Finetuning of Quantized LLMs (QLORA)

Autorii rafinează și mai mult tehnica de antrenare LoRA în [25] prin mai multe abordări. În primul rând, introduc un nou tip de date Normal Float 4, prin care asumă faptul că parametrii unei rețele neuronale respectă, de obicei, o distribuție normală cu o anumită deviație standard. Aceștia scalează, în primă fază, distribuția parametrilor la o distribuție normală standard cu deviația standard de 1. Pentru a evita valorile extreme ce cresc eroarea de cuantizare, se folosesc de cuantizarea pe blocuri, doar că mărimea blocurilor este dictată de cuantilele distribuției normale standard, făcând cât mai puțin posibilă încadrarea inegală a valorilor în blocurile de cuantizare. Pe lângă cuantizarea valorilor parametrilor din matricea înghețată la Normal Float 4, aceștia aplică cuantizare dublă prin cuantizarea constantei de cuantizare (reprezentată inițial în floating point 32) și aplică mutarea paginilor de memorie de pe GPU pe CPU pentru evitarea erorilor de memorie pline a plăcii grafice. Toate aceste inovații fac posibilă antrenarea modelelor foarte mari cu resurse hardware limitate.

2.1.15 Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (Învățarea prin consolidare prin intermediul recenziilor umane) sau RLHF [26] reprezintă o tehnică de aliniere a modelelor mari de limbaj la preferințele umane prin intermediul Reinforcement Learning. Abordarea constă în mai multe faze de dezvoltare:

1. Antrenarea în mod supervizat a unui model pentru una sau mai multe cerințe specifice.

2. Extragerea mai multor variante de răspuns (specifice pentru fiecare sarcină în parte) pentru un prompt și ordonarea într-un clasament al răspunsurilor de către niște adnotatori.

3. Antrenarea unui model, de obicei cu aceeași arhitectură ca modelul în speță, cu aceste clasamente pentru a fi capabil să ofere recompense modelui ajustat cu preferințe umane.

4. Utilizarea algoritmilor de Reinforcement Learning pentru ajustarea modelului principal, utilizând recompense de la modelul de recompense menționat anterior.

2.1.16 Direct Preference Optimization

Direct Preference Optimization [27] este o tehnică de aliniere a modelelor de limbă și nu numai, capabilă să ghideze modelele înspre preferințele umane, fară a fi nevoie să se antreneze un model pentru generarea recompensei și fară a se mai utiliza Reinforcement Learning (Învățare prin Consolidare). Această metodă are beneficiul de a nu fi afectată de instabilitatea în antrenare datorată Reinforcement Learning și totodată reprezintă o alternativă mai ieftină față de RLHF. Autorii lucrării au adus funcția obiectiv din metodologia de antrenare RLHF la o funcție de pierdere Cross-Entropy binară ce poate fi minimizată prin antrenarea rețelei neuronale cu gradienți. Metoda se bazează pe un set de date format din exemple ce conțin un text de intrare și o pereche de două posibile texte de ieșire, unul dintre cele două find adnotată că și câștigător, iar celălalt că și pierzător de niște adnotatori umani sau de un sistem. Funcția de pierdere calculată de autori reflectă întocmai deviația modelului față de răspunsul ales și de cel respins. DPO folosește un hiper-parametru de control al deviației modelului ajustat față de cel de la care s-a plecat numit beta. Beta mai mic se deduce într-o depărtare mai accentuată față de modelul de referință, iar o valoare mai mare a hiper-parametrului reflectă o deviație mai mică față de modelul de bază.

2.2 Tehnologii folosite

2.2.1 Python

Python [28] este un limbaj de programare interpretat, ce implementează programarea orientată pe obiecte, cu tipuri dinamice, aplicat în foarte multe domenii din lumea Software. Are avantajele de a asigura o dezvoltare rapidă aplicațiilor unde este folosit, iar sintaxa acestuia are un grad de dificultate scăzut, fiind accesibil pentru persoanele la început de drum în programare. Acesta este utilizat intens în data science și învățare automată datorită bibliotecilor numeroase și calitative pe care le conține.

2.2.2 Pytorch

Pytorch [29] este un framework open-source specializat în Învățarea Profunda a rețelelor neuronale (Deep Learning) bazat pe Python și Torch (bibliotecă pentru Învățare Automată

scrisă în limbajul Lua). Un mare avantaj adus de acesta este ușurință în crearea și optimizarea rețelelor neuronale în mod foarte eficient datorită grafului computațional dinamic de derivare din spatele acestuia, ce permite modificare arhitecturii modelor în timpul execuției.

2.2.3 Pandas,Numpy, Matplotlib și Seaborn

Pandas [30] este o bibliotecă din limbajul Python specializată în procesarea și analiza datelor în format tabelar. Aceasta aduce cu sine păstrarea datelor în structuri de date adecvate în timpul analizei și curățării. Numpy [31] este o bibliotecă din limbajul Python ce oferă operații matematice asupra matricilor unidimensionale și multidimensionale în mod foarte eficient. Matplotlib [32] și Seaborn [33] sunt biblioteci din Python ce oferă posibilitatea utilizatorului de a realiza diverse grafice pe baza datelor disponibile. Diferența dintre cele două fiind faptul că Seaborn oferă mai degrabă ușurință în scrierea codului și stilizare implicită graficelor, pe când Matplotlib oferă o flexibilitate mai mare dar necesită scrierea de mai mult cod. Acestea se pot utiliza și împreună în scopul lor comun.

2.2.4 MongoDB și Pymongo

MongoDB [34] este o bază de date nerelațională ce operează asupra documentelor. Este potrivită pentru stocarea ușoară și eficientă a datelor, în special cele non-tabelare. Pymongo [35], pe de altă parte, reprezintă legătura dintre baza de date MongoDB și programul scris în Python. Aceasta permite aplicarea operațiunilor asupra bazei de date, precum inserare, ștergere sau interogare, direct din codul Python.

2.2.5 Transformers, Accelerate, Peft, Evaluate și Trl

Transformers [36] este o bibliotecă open-source folosită în special pentru diverse sarcini de prelucrare a limbajului natural, dar nu numai. Aceasta oferă implementări ale modelor State Of The Art, iar multe dintre ele le oferă și pre-antrenate pentru a se ajusta ulterior prin fine-tuning, utilizând sau nu propria lor interfață de antrenare. Accelerate [37] este o bibliotecă care oferă un stil al codului asemănător Pytorch pentru antrenarea rețelelor neuronale în mod eficient. Aceasta pune la dispoziție un mod mai simplificat și intuitiv al antrenării modelelor pe mai multe plăci grafice (GPU) sau pe TPU (Tensor Processing Unit). Peft (Parameter-Efficient Fine-Tuning) [38] este o bibliotecă de la Hugging Face ce facilitează ajustarea modelor pre-antrenate asupra unei cerințe (fine-tuning) în mod eficient. Aceasta se foloste de diverse tehnici printre care și LoRA și QLoRA. TRL [39] este o bibliotecă pentru antrenarea modelelor de limbaj prin antrenare supervizat, Reinforcement Learning, DPO etc. Evaluate [40] este o bibliotecă specializată in evaluarea modelelor de Învățare Automată. Din această bibliotecă, am folosit în cadrul

lucării de licență metricile ROUGE [41] si BERT Score [42], cea din urmă am utilizat-o fără termenul idf.

2.2.6 Flask și Bootstrap

Flask [43] este un framework Python construit pentru realizarea de aplicații web cu ușurință. Acesta înglobează biblioteci precum jinja [44] și werkzeug [45], oferind o modalitatea de dezvoltarea mai prietenoasă cu programatorul. Bootstrap [46] este un framework ce facilitează realizarea frontend-ului aplicației cu ușurință, acesta îmbină mai multe componente stilizate de HTML si CSS.

2.3 Abordări recente

Cea mai reprezentativă lucrare pentru rezumarea de texte în mod abstractiv în limba romană este [47]. Autorii au extras un set de date format din articole de știri și rezumatele aferente de la o publicție și au antrenat mai multe variante al modelului GPT2 [48], urmărind abordarea de token-uri de control pentru a specifica în mod concis modelului numărul de propoziții, de cuvinte a rezumatului generat, mărimea sumarului sau suprapunerea acestuia cu documentul de sumarizat.

O altă lucrare ce explorează, de data aceasta, sumarizarea extractivă pentru limba română este [49]. În cadrul acestei lucrări se abordează sumarizarea nesupervizata cu ajutorul modelului BERT. Spre deosebire de articolul menționat anterior, aici se axează pe extragerea celor mai importante propoziții pentru realizarea unui rezumat.

Capitolul 3

Realizarea setului de date

După cum am menționat în introducere, în prima parte am adunat noi date de la publicații online de știri pentru a completa setul de date. Astfel, am realizat mai multe scripturi de colectare a datelor pe care le-am rulat de-a lungul a mai multor zile. Cu acordul ziarelor online b365.ro [50], alephnews.ro [51] și buletin.de/București [52], am rulat scripturi de colectare pentru a aduna toate știrile și rezumatele aferente care se încadrau în anumite constrângeri. În acest sens, nu am urmărit în mod expres cantitatea datelor extrase, ci mai degrabă calitatea acestora. Astfel, am reușit să colectez aproximativ 25 de mii de exemple.

3.0.1 Procesul de colectare a datelor

Pentru fiecare publicație de știri de unde am extras datele, am realizat câte un script separat în limbajul de programare Python. M-am folosit de sitemap-ul site-ului pentru a fi mai ușor de extras toate știrile în ordine, fară să omit vreun articol relevant. Deoarece pe sitemap se regăseau și alte link-uri înafară de cele către articole de știri, am filtrat textul link-urilor. Astfel, nu am făcut cereri către link-urile ce conțineau imagini sau documente. Cererile către site-ul țintă le efectuez cu modulul requests [53] din Python. Pentru a nu suprasolicita serverul și pentru a nu fi blocat, schimb câmpul User Agent din header-ul cererii, aleator, cu ajutorul modului fake_useragent [54] și efectuez cererile la un interval diferit de timp, aleator la fiecare apelare, între 8 și 18 secunde. De asemenea, tot din considerente etice, atunci când cererea eșuează din diferite motive, trec mai departe la următorul link din listă.

Structura finală a setului de date respectă întocmai structura setului de la Readerbench, pentru a putea fi combinate cu ușurință. Fiecare exemplu conține: titlul articolului, categoria din care face parte, conținutul efectiv al articolului, rezumatul corespunzător, sursa/publicația de știri de unde a fost extras și link-ul către articolul respectiv. De menționat este faptul că, pentru publicațiile b365.ro și buletin.de/București, nu am reușit să stochez și categoria, astfel, câmpul există dar nu este completat. Scripturile pentru cele

două ziare locale din București au structură asemănătoare, în timp ce pentru alephnews.ro am ales altă abordare.

Atunci când cererea este terminată cu succes și primesc un cod de status 200, voi primi și conținutul paginii în html pe care îl parsez prin intermediul modulului Python beautifulsoup [55]. În această etapă, aplic primele constrângeri asupra conținutului pe care îl selectez. Așadar, articolul de știri este selectat doar în cazul în care se regăsesc toate câmpurile menționate anterior, cu excepția categoriei în cazul b365.ro și buletin.de/București.

Titlul nu trebuie să conțină simbolul '(P)' în textul acestuia deoarece am observat că articolele tip publicitate nu au structura dorită în acest context. De asemenea, am eliminat orice articol care conținea sintagma "powered_by" în acesta, pe același motiv.

În ceea ce privește sumarul, în cazul publicațiilor b365.ro și buletin.de/București am optat către selectarea primului paragraf. În cazul b365.ro, articolul era selectat ca făcând parte din setul de date doar dacă după primul paragraf urmează un subtitlu cu marcajul HTML <h2>. Pentru buletin.de/București, rezumatul este considerat valid pentru problema în speță numai dacă primul paragraf are tagul .

Conținutul efectiv al articolului este reprezentat de următoarele paragrafe de după rezumat. Dintre acestea, exclud rubrici de tipul "CITEȘTE ȘI:" , copyright, contact și bucăți din alte articole.

Pentru alephnews.ro am selectat bucata de cod din pagina html cu tagul <script> cu tipul "application/ld+json", mai exact al patrulea marcaj <script> cu acest tip. Din acest tag am selectat câmpurile "articleBody", "headline" și "description". Problema apare însă că rezumatul nu este întru-totul conținut în câmpul "description", ci mai degrabă este conținut în câmpul "articleBody". Așa că am selectat tot câmpul menționat și am extras și rezumatul din secțiunea body a paginii, care se regăsea ca fiind prima listă cu tagul <ul>. În câmpul "articleBody" este conținut atât rezumatul, cât și conținutul efectiv a știrii. Textul selectat de sumarizat l-am considerat ca fiind paragrafele ce urmează după sumar. Titlul l-am obținut din câmpul "headline", iar categoria am parsat-o din link-ul articolului.

In ideea de a avea o calitate ridicată a datelor, am mai aplicat următoarele filtre, și anume:

- rezumatul trebuie să conțină minimum 15 cuvinte (cuvintele sunt despărțite de un spațiu)
- pentru cazul buletin.de/București, pentru primele articole de pe site, primul paragraf nu era scris cu tagul ci cu tagul care în marea majoritate a cazurilor nu reprezintă un rezumat adecvat al articolului
- pentru a putea fi considerat un rezumat, am decis ca numărul de cuvinte din conținut să fie de minim două ori mai mare decât numărul de cuvinte din rezumat

De menționat este faptul că nu am adunat toate articolele de pe site-uri, ci din nou am aplicat anumite filtre. În cazul b365.ro, în primii ani ai publicației, primul paragraf nu reprezenta neapărat un sumar al articolului, ci mai degrabă următoarele paragrafe continuau ideea din primul paragraf, fară să reitereze ce este menționat în acesta. Pentru buletin.de/București, pentru primele articole, am considerat că articolele nu au neapărat un rezumat, ci, din nou ca în cazul b365.ro, primul paragraf face parte din conținutul știrii. În general, această regulă se aplică în special în cazul primului paragraf, atunci când are marcajul . În cazul alephnews.ro, nu au existat probleme în ceea ce privește calitatea rezumatelor, în schimb am selectat acele știri ce nu au fost incluse și în setul de date citat în introducere.

Toate aceste filtre și constrângeri le-am ales pe baza cercetărilor personale prin studiul amănunțit al articolelor de știri, structura site-ului, paginilor de știri și urmărind trendul care s-a modificat sau nu de-a lungul timpului pentru aceste elemente.

Din considerente de memorie, am păstrat datele într-o bază de date nerelațională MongoDB, păstrând toate câmpurile pe care le-am menționat, plus un id. Pentru operațiile ce țin de baza de date, m-am folosit de modulul pymongo din Python. Scripturile le-am realizat în așa fel în cât să poată fi oprite și reluate fară probleme, deoarece datele au fost adunate în decursul a mai multor zile. Procesul a luat loc preponderent noaptea pentru a nu pune presiune pe infrastructura site-urilor. Pentru datele de la b365.ro, am extras articolele din decembrie 2022 până la martie 2024, pentru buletin.de/București din iunie 2019 până în martie 2024, iar pentru alephnews.ro din august 2022 până în martie 2024.

3.0.2 Curățarea setului de date

O problemă majoră a modelelor bazate pe arhitectura Transformer o reprezintă inadaptabilitatea token-izatorul la domenii noi și termeni specifici sau simboluri pe care nu a fost antrenat. Așadar, pentru a evita posibila multitudine de token-uri necunoscute, am curățat setul de date de emoji-urile din textul acestora cu ajutorul modului demoji [56] din Python. În special datele extrase de pe alephnews.ro au avut o pondere mare de simboluri HTML. Astfel, am decis să elimin toate simbolurile HTML în afară de `"""`, pe care l-am înlocuit cu ghilimele pentru a marca începutul unui citat. Am eliminat orice marcaj de trecere la rand nou `"\n"` și am eliminat spațiile redundante.

Pentru datele de la Readerbench, am ales să elimin exemplele în care numărul de cuvinte din rezumat era mai mare sau egal decât numărul de cuvinte din documentul ce trebuie sumarizat. Astfel, am eliminat 269 de exemple pe care nu le-am considerat utile în această speță. Întocmai ca în cazul curățării datelor mele, am eliminat emoji-urile, simbolurile html, trecerea la rand nou și spatiile redundante. Având datele curățate și cu aceeași structură, am reușit să concatenez cu succes cele două seturi de date într-unul singur.

Capitolul 4

Antrenarea cu datele preprocesate de TextRank

4.1 Preprocesarea datelor

4.1.1 Analiza datelor

Pentru implementarea preprocesării cu TextRank, a trebuit să realizez o analiză minuțioasă a setului de date. Ideea preprocesării presupune micșorarea numărului de token-uri, pentru a nu mai fi nevoit să trunchiez datele de intrare, înainte de trecerea acestora prin model. O dată cu trunchierea datelor, pot pierde informație valoroasă pe care modelul ar fi putut să o folosească în procesul de generare a rezumatului. Deși modelul ales poate procesa date mai lungi decât pragul maxim pe care îl voi folosi, acest lucru implică o durată mai lungă a timpului de antrenare. Așadar, pe lângă procesul clasic de trunchiere, pe care îl voi experimenta, am să utilizez preprocesarea cu TextRank pentru micșorarea numărului de token-uri din datele de intrare.

În vederea studiului impactului asupra lungimii secvenței de intrare a modelului, am ales să limitez numărul de token-uri la maxim 768. Lucrul acesta îl realizez fie cu trunchiere, fie cu preprocesare cu TextRank. În ceea ce privește trunchierea, lucrurile stau ca atare deoarece trebuie să șterg toate token-urile ce urmează după pragul ales. În situația preprocesării, a trebuit să analizez mai în adâncime setul de date. Un dezavantaj al algoritmului îl reprezintă faptul că micșorarea textul se realizează la nivel de propoziții, nu de cuvinte. Astfel, trebuie să am o privire de ansamblu la cum voi aplică algoritmul.

În cele ce urmează, mă voi referi la numărul de cuvinte ca numărul total de cuvinte, semne de punctuație și numere deoarece am observat că media numărului de cuvinte (în cazul acesta) raportat la numărul de token-uri per document este mult mai centrat în medie față de cazul în care aș împărți textul numai după spații.

Am token-izat întregul setul de date pentru a putea evidenția relația dintre numărul de cuvinte și numărul de token-uri. Din figurile 4.1 și 4.2 se poate observa că numărul de

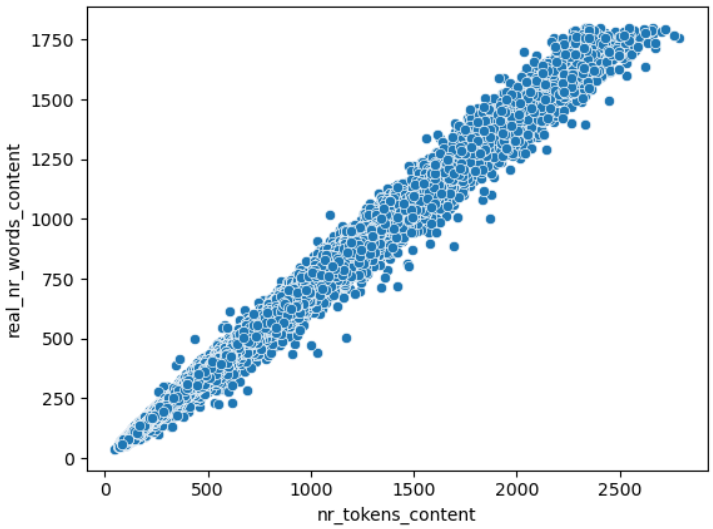

Figura 4.1: Pe axa orizontală sunt lungimile documentelor ce trebuie rezumate în token-uri, iar pe axa verticala se află lungimile documentelor ce trebuie rezumate în cuvinte. Graficul denotă corelația dintre cele două.

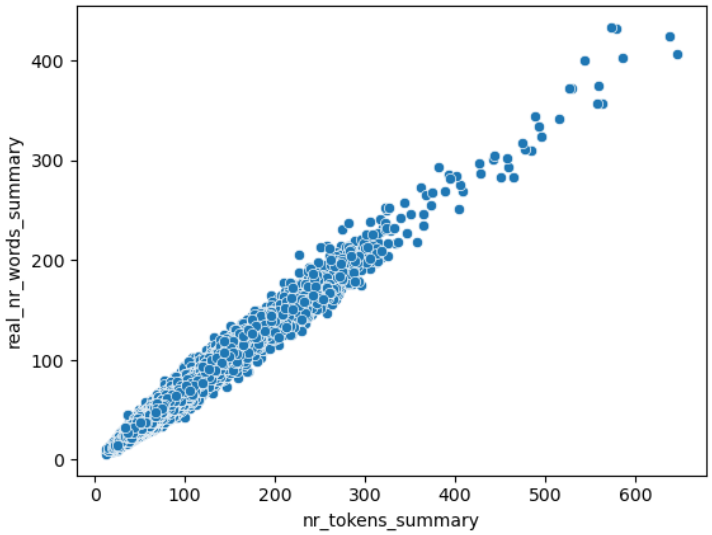

Figura 4.2: Pe axa orizontală sunt lungimile rezumatelor în token-uri, iar pe axa verticală se află lungimile rezumatelor în cuvinte. Graficul denotă corelația dintre cele două.

cuvinte este foarte corelat cu numărul de token-uri, atât pentru documentul ce trebuie sumarizat, cât și pentru rezumat. Totodată, corelația Pearson [57], calculată pentru aceste două entități, este de peste 0.99. Am analizat distribuția raportului dintre numărul de token-uri și numărul de cuvinte, iar această are media de aproximativ 1.38 token-uri la un cuvânt și deviația standard este de puțin peste 0.08. Regula se propagă și la un eșantion mai mic. Deși deviația standard tinde să se mărească odată cu micșorarea numărului de exemple selectate, aceasta respectă forma de clopot a distribuției normale, iar media este centrată în toate cazurile la aproximativ 1.38.

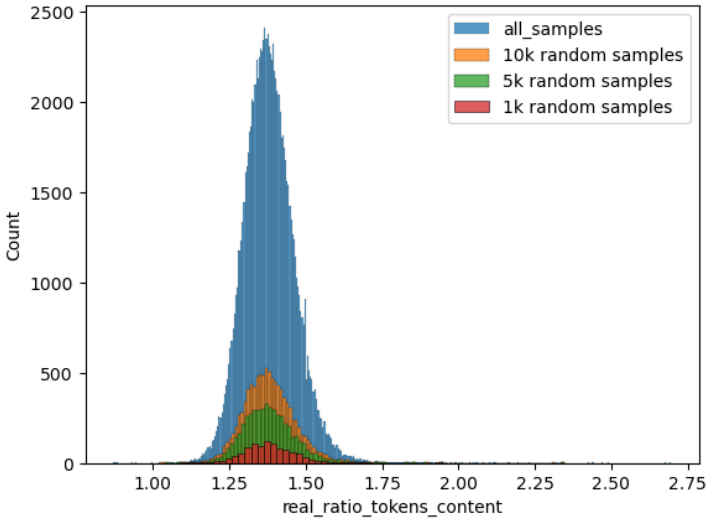

Figura 4.3: Graficul demonstrează rata de token-uri per cuvânt la nivelul textului ce trebuie rezumat pe diverse eșantioane de exemple, 1000, 5000, 10000 respectiv 100577.

Din figura 4.3, se poate observa că o dată cu creșterea numărului de exemple selectate, deviația standard scade, iar media rămâne centrată în 1.38. Într-adevar, există valori extreme, preponderent în partea dreaptă a distribuției, acestea sunt foarte puține și reprezintă acele exemple ce au în textul documentului multe cifre sau multe substantive proprii. În cazul substantivelor proprii, algoritmul de token-izare are dificultăți în procesare deoarece reprezintă niște cuvinte mai rare și astfel le împarte în mai multe token-uri, generând numărul mare de token-uri din exemplele respective. Spre exemplu, sunt doar 1021 de exemple ce au mai mult de 1.6 token-uri per cuvânt și doar 916 exemple cu mai puțin de 1.2 token-uri per cuvânt după aplicarea procesului de token-izare 4.4. Reiterând cele menționate anterior, raportul este bine cumulat în vecinătatea mediei, de unde trag concluzia că mă pot baza pe numărul de cuvinte în locul numărului de token-uri în cazul selectării exemplelor ce trebuie micșorate. Așadar, setez pragul de unde încep să selectez la 371 în cazul contextului de 512 token-uri și 556 în cazul contextului de 768 de token-uri.

Obiectivul central al preprocesării îl reprezintă micșorarea exemplelor care nu încap în 371, respectiv 556 de cuvinte. Așadar, aplic algoritmul doar pe exemplele care nu încap în această constrângere, minimizând alterarea logicii textelor. Am ales ca pragul de unde

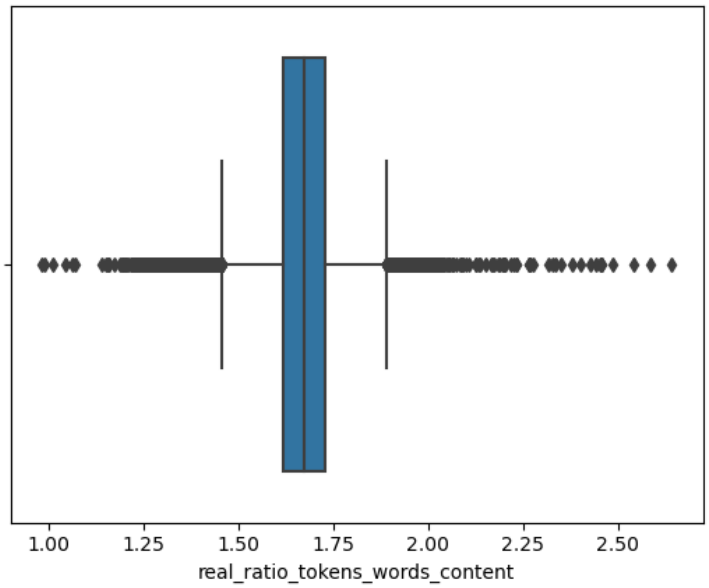

Figura 4.4: Graficul prezintă distribuția ratei de token-uri per cuvânt la nivelul textului ce trebuie rezumat printr-un boxplot din seaborn.

încep să aplic preprocesarea cu TextRank, pentru ambele cazuri, să fie majorat cu 27 de cuvinte, deoarece am considerat că aplicarea algoritmului pe documentele ce au între numărul de cuvinte dorit și pragul setat ar putea duce la o pierdere excesivă de informație prin micșorarea dimensiunii documentului. TextRank este aplicat la nivel de propoziții, nu de cuvinte, și prin urmare, trebuie să aproximez numărul de propoziții care trebuie eliminate pentru ca textul să încapă în contextul setat. Dacă voi selecta exemple pentru a fi preprocesate ce au doar câteva cuvinte peste pragul maxim, există posibilitatea foarte probabilă să elimin prea mult. Așadar, am decis să îmi iau o marjă de eroare de 27 de cuvinte, ce reprezintă media numărului de cuvinte per propoziție la nivelul întregului set de date.

4.1.2 Preprocesare cu TextRank

Pentru aplicarea algoritmului, mă folosesc de biblioteca pytextrank[58] și o adaug ca pipeline pentru spacy[59]. Varianta algoritmului pe care o folosesc cei care au creat această bibliotecă este cea ce implementează măsura de similaritate dintre propoziții din lucrarea ce a propus algoritmul.

Din observațiile proprii, am remarcat că în unele documente se repetă propoziții, acestea aflându-se în diferite poziții în textul original. Astfel, păstrez propozițiile cu pozițiile unde apar în textul original într-un dicționar Python de forma: textul propoziției ca și cheie, poziția propoziției în textul original ca și valoare. Adaug propoziția în dicționar doar o dată, pentru a nu avea duplicate în textul rezultat. Deși TextRank este considerat un algoritm de sumarizare extractivă, eu îl voi folosi pentru micșorarea textului, mai exact

îl voi aplica pe tot textul, păstrând toate propozițiile și stocându-le în ordinea importanței date de algoritm.

Având propozițiile sortate după importanța acestora conform TextRank, pot începe procesul de micșorare a textului. Pentru a evita necesitatea de a token-iza fiecare propoziție în parte pentru a număra token-urile, aleg să mă raportez la numărul de cuvinte. După cum am menționat în subsecțiunea anterioară, există o relație strânsă între numărul de token-uri și numărul de cuvinte, așa că setez pragul unde intenționez să ajung cu documentele la 371 de cuvinte, deoarece am în medie 1.38 token-uri per cuvânt, pragul real fiind de fapt 512 token-uri, analog și pentru cazul contextului de 768 de token-uri. Așadar, adaug câte o propoziție în ordinea importanței până când numărul de cuvinte din textul selectat depășește 371, respectiv 556 de cuvinte. Având textul micșorat, îl sortez după ordinea apariției propozițiilor în textul original, pentru a păstra logica acestuia.

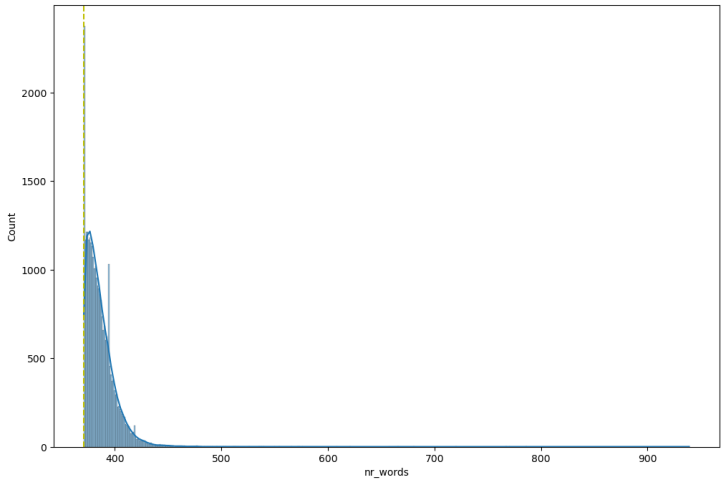

Figura 4.5: Distribuția textelor procesate cu TextRank până la 371 de cuvinte. Linia verde punctată marchează pragul de 371 de cuvinte.

După aplicarea algoritmului, ajung să am o statistică clară a rezultatelor obținute. Obțin o distribuție a numărului de cuvinte după preprocesare prezentată în figura 4.5, cu media de aproximativ 386 de cuvinte și deviația standard de aproximativ 15 cuvinte. Desigur, există și anumite valori extreme (figura 4.6), dar acestea sunt într-un număr limitat iar magnitudinea acestora a fost puternic influențată de procesare. Deși există unele valori care se abat de la distribuția expectată la început, marea masă a exemplelor este concentrată între 377 și 393 de cuvinte, 14901 de exemple din 28676 selectate pentru preprocesare se alfă în acest interval. Există o fracțiune din datele procesate care au rămas cu un număr însemnat de cuvinte. Dintre acestea, doar 1547 au cel puțin 411 cuvinte, respectiv 29 cu peste 496 de cuvinte. În cele din urma, textele au scăzut în lungime accentuat și s-au stabilizat în jurul intervalului dorit (figura 4.7).

Pentru a valida într-adevăr calitatea rezultatelor obținute, token-izez textele pe care le-am procesat fară să aplic padding sau trunchiere. Obțin o distribuție normală a numărului

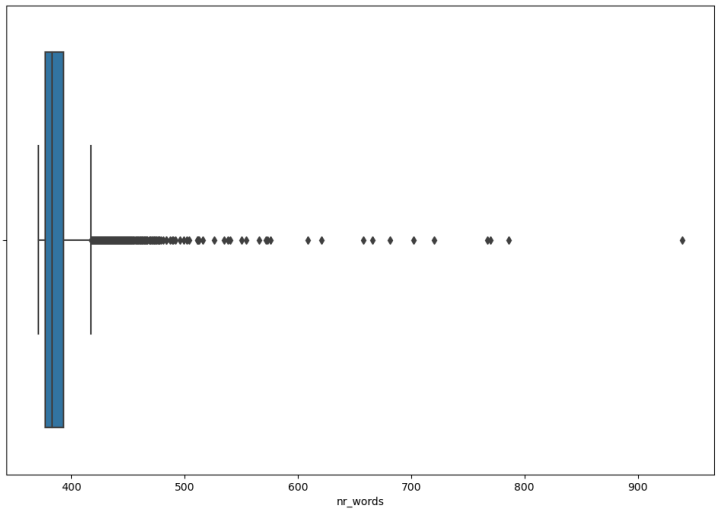

Figura 4.6: Distribuția textelor preprocesate cu TextRank pentru 512 token-uri. Graficul este la nivel de cuvinte. Este un boxplot din seaborn, pentru a se putea observa cuartilele și valorile extreme.

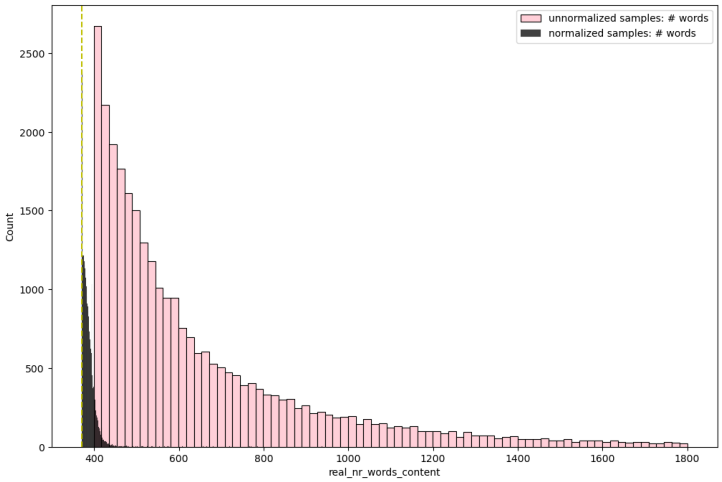

Figura 4.7: Graficul prezintă diferență dintre distribuția formată din lungimiile textelor în cuvinte înainte și după preprocesarea cu TextRank în cazul contextului de 512 token-uri. Culoarea neagra reprezintă lungimile textelor preprocesate, iar cea roz lumgimile textelor nepreprocesate. Linia verde punctată marchează pragul de 371 de cuvinte.

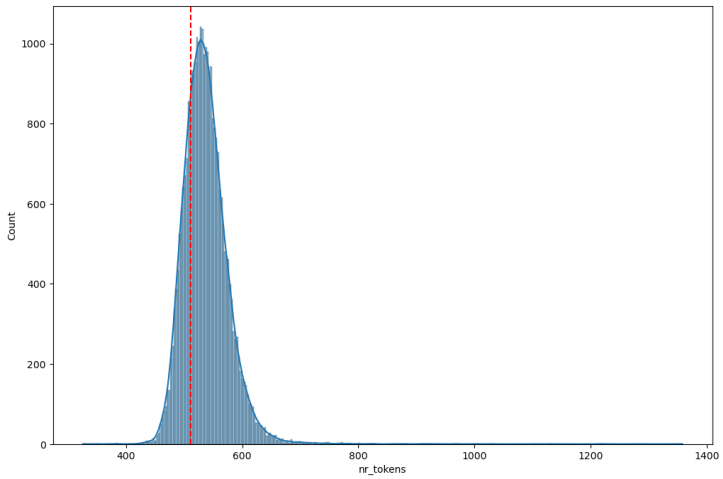

Figura 4.8: Distributia textelor procesate cu TextRank la nivel de token-uri in contextul de 512 token-uri. Linia verde punctata marcheaza pragul de 512 token-uri.

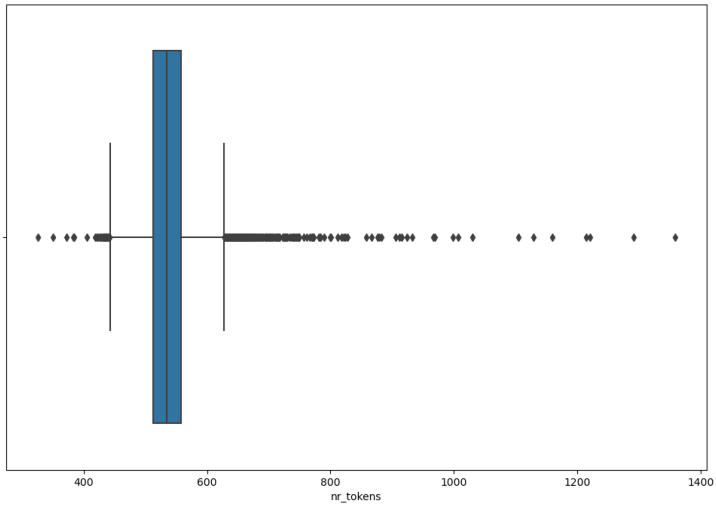

Figura 4.9: Distribuția textelor preprocesate cu TextRank pentru 512 token-uri. Graficul este la nivel de token-uri. Este un boxplot din seaborn, pentru a se putea observa cuartilele și valorile extreme.

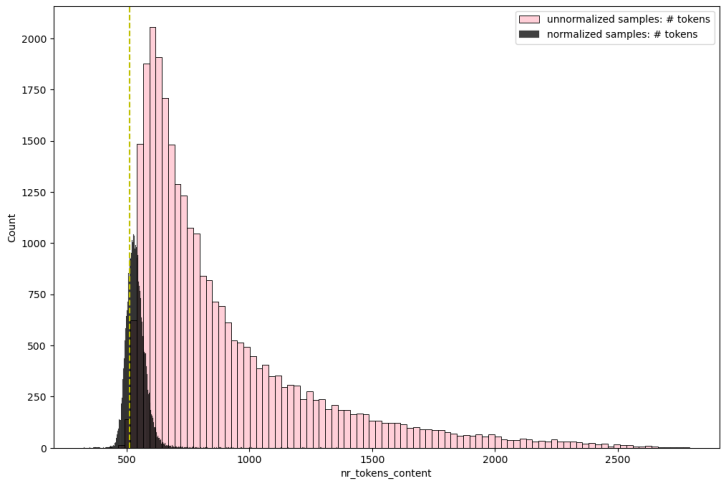

Figura 4.10: Graficul prezintă diferență dintre distribuția formată din lungimiile textelor în token-uri înainte și după preprocesarea cu TextRank în cazul contextului de 512 token-uri. Culoarea neagră reprezintă lungimile textelor preprocesate, iar cea roz lumgimile textelor nepreprocesate. Linia verde punctată marchează pragul de 512 token-uri.

de token-uri cu media la 537 token-uri per text, cu o deviație standard de aproximativ 39 (figura 4.8). Ca și în cazul cuvintelor, există exemple care nu se încadrează în intervalul țintit (figura 4.9), dar marea majoritate, 23085 de exemple au între 494 și 583 de cuvinte, dintre care 14587 au între 512 și 558 token-uri. Ca valori extreme pot aminti doar 145 de texte cu cel puțin 670 de token-uri, dintre care doar 3 cu peste 1215 token-uri. Texte pentru care am pierdut o parte din informație prin ștergere excesivă sunt doar 30, ce au cel mult 439 token-uri. Astfel, am reușit să selectez cele mai importante propoziții, conform algoritmului TextRank, fără să pierd prea multă informație (figura 4.10).

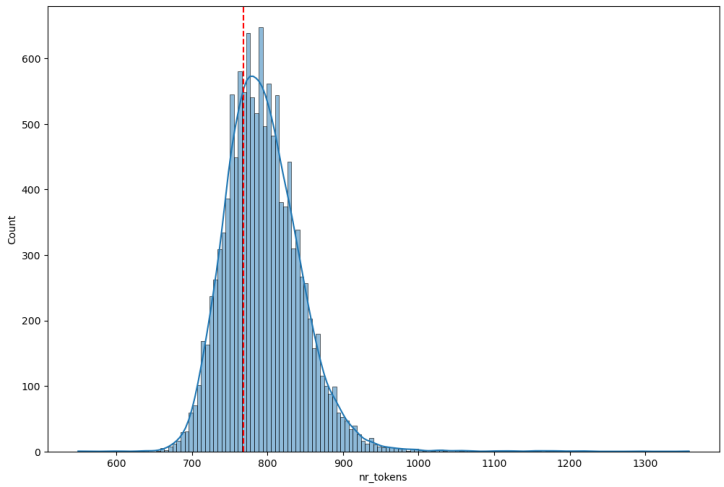

Figura 4.11: Distribuția textelor procesate cu TextRank la nivel de token-uri în contextul de 768 de token-uri. Linia verde punctată marchează pragul de 768 de token-uri.

De asemenea, în cazul pragului de 768 de token-uri, am obținut o distribuție cu media de 570 și deviația standard de 15 în cazul cuvintelor (figura 4.12) și o distribuție normală

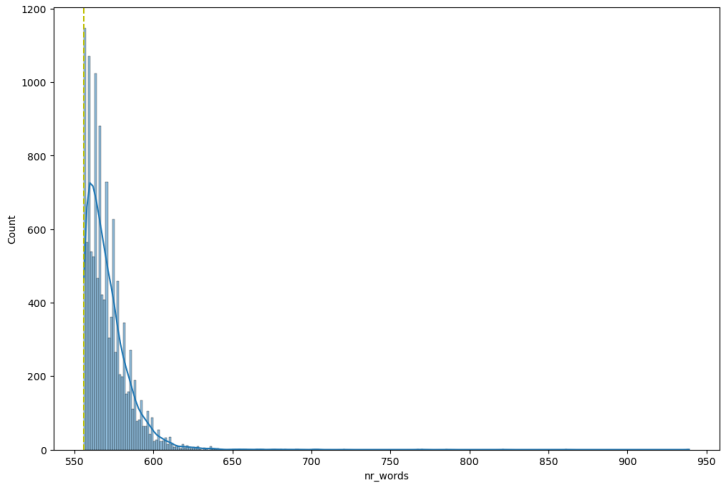

Figura 4.12: Distribuția textelor procesate cu TextRank până la 556 cuvinte. Linia verde punctată marchează pragul de 556 de cuvinte.

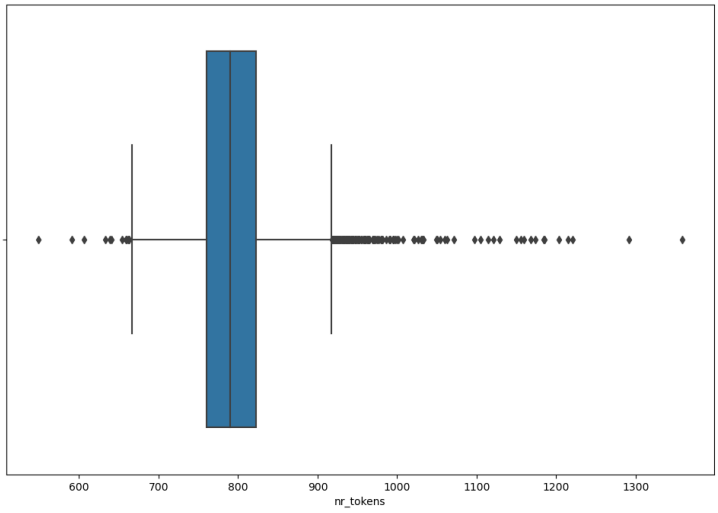

Figura 4.13: Distribuția textelor preprocesate cu TextRank pentru 768 token-uri. Graficul este la nivel de token-uri. Este un boxplot din seaborn, pentru a se putea observa cuartilele și valorile extreme.

cu media 793 și deviația de 50 în cazul token-urilor (figura 4.11). Din totalul de 12463 de exemple 6787 au intre 561 și 577 de cuvinte, iar 10036 din total au între 735 și 856 de token-uri. Ca și în cazul contextului mai mic, de 512 token-uri, valorile extreme există dar sunt într-un număr restrâns (figura 4.13).

4.1.3 Interpretarea rezultatelor preprocesării

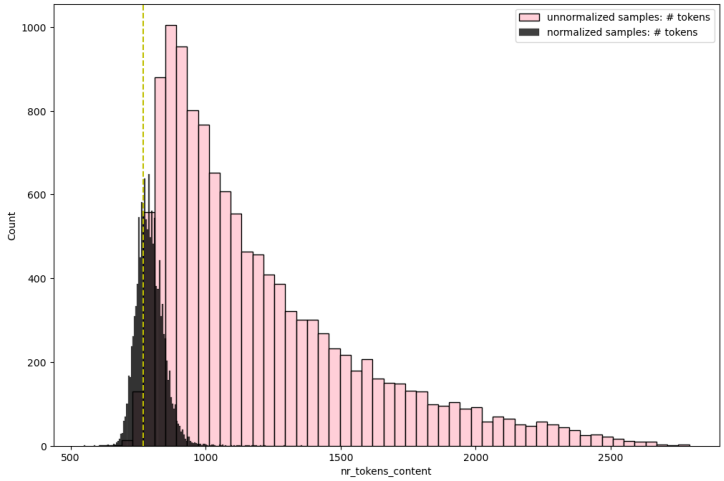

Figura 4.14: Graficul prezintă diferența dintre distribuția formată din lungimiile textelor în token-uri înainte și după preprocesarea cu TextRank în cazul contextului de 768 de token-uri. Culoarea neagră reprezintă lungimile textelor preprocesate, iar cea roz lumgimile textelor nepreprocesate. Linia verde punctată marchează pragul de 768 de token-uri.

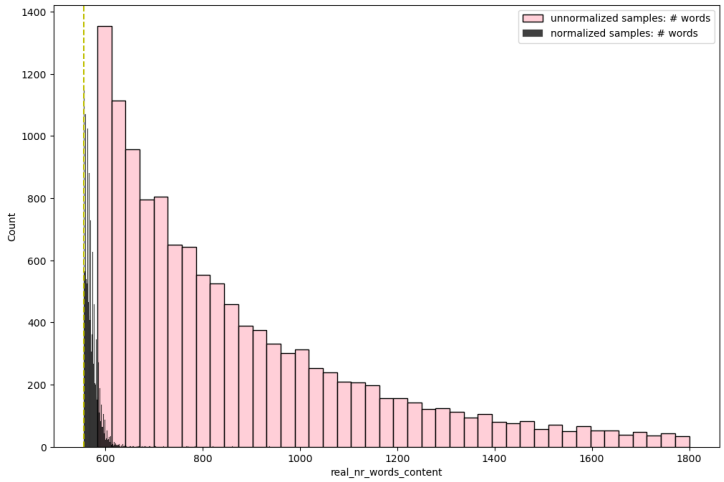

Figura 4.15: Graficul prezintă diferența dintre distribuția formată din lungimiile textelor în cuvinte înainte și după preprocesarea cu TextRank în cazul contextului de 768 de token-uri. Culoarea neagră reprezintă lungimile textelor preprocesate, iar cea roz lumgimile textelor nepreprocesate. Linia verde punctată marchează pragul de 556 de cuvinte.

În mod cert, cel mai mare avantaj al acestei abordări o reprezintă diferență majoră

față de distribuția textelor nenormalizate, de unde am plecat. Atât în ceea ce privește cuvintele (figurile 4.7 și 4.15) cât și token-urile (figurile 4.10 și 4.14), numărul de cuvinte sau de token-uri s-a micșorat foarte mult și s-a stabilizat în jurul intervalului asumat la început. Textele foarte mari s-au micșorat intens și chiar dacă nu se încadrează în limita a 512 token-uri sau 768 token-uri algoritmul a selectat cele mai importante propoziții, lăsând la final propozițiile cu mai puțină valoare, care ar putea fi șterse la procesul de trunchiere. Spre exemplu, cel mai mare document înainte de normalizare avea 2790 de token-uri, iar după normalizare, cel mai mare exemplu a ajuns să aibă 1358 token-uri în ambele cazuri. Astfel, se pot obține beneficii prin aplicarea algoritmului și pentru exemplele foarte mari.

4.2 Antrenarea pe date normalizate si nenormalizate cu TextRank

4.2.1 Modelul ales

Modelul pe care l-am ales pentru a desfășura experimentele este un MBart Large CC25, acesta a fost pre-antrenat pe 25 de limbi pentru traducere automată, dar poate fi utilizat și pentru rezumare monolingv conform [60]. După cum am menționat în capitolul 2, modelul are marele avantaj de a fi pre-antrenat pe aproape 63 GB de date în limba română iar numărul de parametrii este destul de mare făcându-l propice pentru sarcini mai complexe, cum ar fi sumarizarea abstractiva de texte.

O constrângere puternică pentru alegerea modelului pe care am realizat fine-tuning (antrenarea în continuare a unui model pre-antrenat, pentru o cerință specifică) este ca acesta să fie pre-antrenat pentru limba română, fapt ce mi-a micșorat semnificativ aria de selecție. Am ales acest model în special datorită arhitecturii sale. Acesta este o arhitectura bazata pe Transformer de tip Encoder-Decoder, fată de GPT2. Acesta beneficiază atât de partea auto-regresivă reprezentată de Decoder, capabilă să genereze text, cât și de partea de atenție bidirectională reprezentată de Encoder. Astfel, modelul poate avea o înțelegere mai bună a textului de intrare (documentul ce trebuie sumarizat), dar poate genera și auto-regresiv rezumatul corespunzător. Totodată, modelul poate primi secvențe de intrare și poate genera secvențe de lungimi diferite între ele, făcând-ul propice din punct de vederea al memoriei și al timpului necesar antrenării, spre deosebire de modelele Decoder-only ce trebuie să genereze o secvență de lungime egală cu secvența de intrare (eticheta în procesul de antrenare este reprezentată de rezumatul de referință concatenat la textul de intrare, mutate cu o poziție la dreapta), in timpul antrenării.

4.2.2 Împărțirea setului de date

Din totalul de 101575 de exemple rezervate pentru antrenarea supervizată, am alocat 84484 pentru antrenare, 7000 pentru subsetul de validare și 9093 pentru testare. Din cauza unităților de procesare limitate de care dispun, nu am putut să calculez metrici în timpul antrenării pentru a ține evidența calității modelului, deoarece acestea necesită ca modelul să genereze text, ceea ce ar fi încetinit foarte mult convergența. Așadar, am ales să aloc 7000 de exemple de validare pentru a avea o reprezentare cât mai reală a funcției de pierdere pentru date noi. De asemenea, atunci când aleg între diferite checkpoint-uri ale modelor mă bazez tot pe funcția de pierdere și din această cauză am ales 9093 de exemple pentru subsetul de testare. Din totalul datelor pe care le dețin elimin aproximativ 1000 de exemple cu peste 1800 de cuvinte deoarece nu fac obiectivul etapei de antrenare. Aceste exemple supradimensionate ar fi introdus prea mult zgomot în procesul de antrenare și implicit divergența sau convergența mult prea lentă.

4.2.3 Pregătirea datelor

Pentru a putea evidenția aportul pe care îl aduce preprocesarea cu TextRank asupra performanței modelului, aleg să limitez numărul maxim de token-uri per document ce urmează să fie sumarizat la 512, respectiv 768. Un prag mai mare ar însemnă ca preprocesarea să fie aplicată pe mult prea puține exemple pentru a avea un impact major. Spre exemplu, doar 7139 de exemple au peste 1024 de token-uri. Pentru toate exemplele ce nu se încadrează în această limită, aplic trunchiere la dreapta, iar pentru cele care sunt mai scurte de atât aplic padding tot la dreapta pentru a beneficia de paralelismul oferit de unitatea de procesare grafică (GPU). Înlocuiesc id-ul token-ul de padding cu -100 pentru a nu fi luat în considerare la calcularea funcției de pierdere, pentru ca modelul să nu fie recompensat sau penalizat în mod eronat atunci când generează padding și pentru a fi încurajat să producă text de calitate. Voi folosi atât pentru antrenare cât și pentru testare două seturi de date : unul preprocesat cu TextRank și unul neprepocesat.

4.2.4 Experimente și rezultate

Modelul ales nu este unul foarte mare comparativ cu modele precum versiuni de 7 sau 70 de miliarde de parametrii a Llama 2 [61], dar un impediment major al antrenări este necesarul mare de memorie solicitat de datele de antrenare. Atât documentele ce trebuie sumarizate cât și rezumatele aferente sunt în general mai mari decât exemplele dintr-un set de date pentru sarcina de question answering (răspunsuri la întrebările primite), spre exemplu. Din această cauză, antrenez modelul pentru rezumarea abstractiva a textelor cu QLORA pentru a putea folosi un batch de mărime 4. Totodată, folosesc gradient checkpointing și acumularea gradienților pentru cel puțin 4 pași de antrenare și în final

ajung să am un batch de minimum 16 exemple, ceea ce îmi oferă mai multă stabilitate în procesul de antrenare.

Am început cu un rang de 8 și alpha de 16 cu un weight decay de 0.001, cu o rată de învățare de 0.0001 pe un context de 768 de token-uri. Am folosit Cosine Annealing pentru learning rate scheduler (programarea ratei de învățare) și AdamW ca și algoritm de optimizare. În prima parte, am utilizat 8 pași de acumulare a gradientului, pe care i-am scăzut treptat la 6 și apoi la 4 deoarece, antrenarea era mult prea lentă ca să fie realizată în timp util. De asemenea, având în vedere complexitatea sarcinii de sumarizare abstractiva, și faptul că modelul are aproximativ 680 de milioane de parametrii, am optat către un rang mai mare față de cum e menționat ca fiind optim în lucrarea ce a introdus QLORA. Așa că, am trecut la un rang 16 cu alpha 32 și mai apoi la un rang de 32 cu alpha de 64, păstrând regularizarea la 0.001. Pentru rangul de 32, am ales să cresc weight decay la 0.01 și, desi în prima parte a avut rezultate mai slabe, pe parcurs a dus la o generalizare mai bună. De asemenea, am încercat cu rate de învățare mai mici și mari de 0.0001 (0.00001,0.00005 și 0.0003) și nu am obținut rezultate mai bune.

În toate experimentele, am ales să cuantizez modelul la normal float 4, să folosesc cuantizare dubla, iar tipul de date în care se desfășoară calculele l-am ales ca fiind brainfloat16 [62] pentru stabilitatea numerică în comparație cu floating point 16, și rapiditatea cu care se desfășoară calculele. De asemenea, am utilizat dropout pentru LoRA de 0.01, rslora [63] și am inițializat matricile de adaptare LoRA cum este specificat în lucrarea originală (prima matrice dintr-o distribuție Gaussiana iar cea de a doua cu 0). Am mai experimentat și antrenarea doar a ultimului strat de clasificare de la final, dar a adus rezultate mult mai slabe fata de antrenarea cu QLORA.

Așadar, cele mai bune rezultate, din perspectiva funcției de pierdere, le-am obținut cu:

- o rată de învățare de 0.0001
- Cosine Annealing cu 0.5 ciclii
- weight decay de 0.01
- mărimea batch-ului este de 4 cu 4 pași de acumulare
- 400 de pași pentru creșterea progresiva în mod linear de la 0 la rata de învățare setată (linear warmup) pentru a stabiliza modelul și pentru accelerarea convergenței cu AdamW
- rang 32 cu alpha 64 și dropout de 0.01 pentru LoRA
- cu rslora și inițializarea matricilor LoRA cu varianta originală

Cu acest spațiu al hiper-parametrilor am realizat următoarele experimente pentru a observa efectul preprocesării cu TextRank:

1. O variantă ce are un context de 512 token-uri obținut prin trunchierea la dreapta.
2. O variantă ce are un context de 768 token-uri obținut prin trunchierea la dreapta.
3. O variantă ce are un context de 512 token-uri obținut prin preprocesarea cu TextRank la aproximativ 371 de cuvinte și prin trunchierea la dreapta până la 512 token-uri.
4. O variantă ce are un context de 512 token-uri obținut prin preprocesarea cu TextRank la aproximativ 556 de cuvinte și prin trunchierea la dreapta până la 768 token-uri.

Pentru metricile calculate prin intermediul textului generat, am folosit doi algoritmi de decodare: Greedy și Beam Search. Pentru ambele modalități am ales să generez rezumate între 35 și 250 de token-uri deoarece am considerat că aproape toate rezumatele s-ar încadra în acest interval și am restricționat ca modelul să nu repete în rezumat n-gramele de lungime 5. Pentru Beam Search am folosit 5 beam-uri și early stopping. Am setat parametrul de penalizare a lungimilor secvențelor generate, în cazul Beam Search, la 1.5. Parametrii menționați anterior i-am ales pe baza testelor pe 400 de exemple din subsetul de validare.

4.2.5 Interpretarea rezultatelor

Rezultatele celor patru modele se pot observa în tabela 4.1 pentru metricile ROUGE și în tabela 4.2 pentru metricile BERT Score. Pentru evaluare am folosit ROUGE-1, ROUGE-2, ROUGE-L, ROUGE-Lsum și BERT-Score (ce cuprinde rata de precizie, de recall și f1-score, fără termenii de idf). Rezultatele arată că modelele antrenate cu textul procesat cu TextRank nu au performat mai bine decât cele antrenate pe datele normale și există o diferență mică între cele două abordări.

Acestă diferență se datorează cel mai probabil din cauza distribuției textelor, ce are un bias în expunerea conținutului relevant în special la începutul textului, de unde și diferență mică de performanță între contextul de 512 token-uri și cel de 768 token-uri, și din cauza faptului ca este posibil ca antrenarea să sufere în cazul în care preprocesarea TextRank ar elimina unele propoziții ce se bazează pe ideea altora. Practic, prin trunchiere, modelul primea ca date de intrare deja cea mai importantă informație, iar procesarea cu TextRank nu a adus un beneficiu pentru această categorie de texte, și anume articole de știri.

Model	Algoritm de decodare	ROUGE-1 (%)	ROUGE-2 (%)	ROUGE-L (%)	ROUGE-Lsum (%)
context 512	Greedy	34.94	16.54	24.73	30.18
	Beam Search	36.13	17.93	25.85	31.56
context 512 TextRank	Greedy	34.95	16.47	24.73	30.23
	Beam Search	35.78	17.44	25.39	31.20
context 768	Greedy	35.13	16.55	24.80	30.35
	Beam Search	**36.17**	**17.95**	**25.85**	**31.60**
context 768 TextRank	Greedy	35.11	16.73	24.99	30.40
	Beam Search	36.02	17.80	25.74	31.45

Tabela 4.1: În tabel sunt prezentate metricile ROUGE pentru fiecare variantă de model MBart Large antrenată: cu context de 512 de token-uri, cu context de 512 de token-uri cu preprocesare TextRank, cu context de 768 de token-uri și cu context de 768 de token-uri cu preprocesare TextRank.

Model	Algoritm de decodare	Precizie (%)	Recall (%)	F1-score (%)
context 512	Greedy	73.66	71.99	72.74
	Beam Search	72.76	72.77	72.72
context 512 TextRank	Greedy	73.34	72.12	72.68
	Beam Search	72.67	72.78	72.67
context 768	Greedy	**73.57**	72.11	72.77
	Beam Search	72.67	**72.84**	72.70
context 768 TextRank	Greedy	73.56	72.14	**72.78**
	Beam Search	72.67	72.77	72.67

Tabela 4.2: În tabel sunt prezentate metricile BERT Score pentru fiecare variantă de model MBart Large antrenată: cu context de 512 de token-uri, cu context de 512 de token-uri cu preprocesare TextRank, cu context de 768 de token-uri și cu context de 768 de token-uri cu preprocesare TextRank.

Capitolul 5

Alinierea la preferințele umane cu Direct Preference Optimization

Metricile calculate în capitolul anterior (ROUGE și BERT Score), deși pot reprezenta o evaluare destul de bună în primă fază, nu sunt foarte corelate cu o evaluare umană și depind în mare măsură de calitatea setului de date. Așadar, propun utilizarea metodei Direct Preference Optimization (DPO) pentru alinierea rezumatelor generate cu preferințele umane, luând în considerare aspecte precum ideea principală, corectitudinea gramaticii textului, parafrazarea sau coerența și logica textului. Metoda constă în etichetarea unui set de date format din perechi de două rezumate racordate la un document ce reprezintă textul original, unul dintre cele doua fiind adnotat ca fiind mai bun, iar celălalt mai slab. În prima etapă, am ales să antrenez în mod supervizat modele pentru sumarizare abstractivă de texte pentru ca modelul ales să fie capabil să genereze rezumate destul de bune înainte de a fi optimizat cu DPO.

5.1 Antrenare supervizată

Scopul expus în această etapă este acela de a antrena supervizat niște modele candidat pentru ca cel mai bun dintre ele să fie optimizat mai departe cu DPO. Luând în considerare că modelul MBART Large folosit în cadrul capitolului anterior a obținut rezultate destul de bune, îl voi alege ca și candidat în acest proces. Pe lângă acesta, antrenez și un model Flan-T5 Large [64], model pre-antrenat și pe limba română. Flan-T5 Large prezintă o arhitectură puțin diferita fata de MBART în principal datorită reprezentării pozițiilor care sunt, de data aceasta, relative, nu absolute ca și în cazul MBART. Pentru a scoate o performanță mai bună decât în capitolul anterior cu MBART, aleg un context de 1024 de token-uri pentru acesta, context în care se încadrează majoritatea datelor din setul de antrenare. Flan-T5, fiind un model mai mare, nu îl pot antrena cu acest context într-un timp rezonabil, așa că scad contextul până la 512 token-uri. La fel ca în cazul

experimentelor ce cuprindeau preprocesare cu TextRank, antrenez ambele modele cu QLORA pentru a încadra un batch de mărime 4.

5.1.1 MBART Large

Având un context mai mare, 1024 de token-uri, și timpul de execuție pentru antrenare a crescut considerabil, așadar, aleg să continui optimizarea de hiper-parametrii cu o variantă mai mică cu context de 512 token-uri (Pentru varianta cu context 1024 antrenare durează aproximativ 7 ore și jumătate per epoca, în timp ce pentru varianta de 512 durează 4 ore și jumătate per epoca). Am început cu cea mai bună combinație de hiper-parametrii din experimentele trecute: batch de mărime 4 cu 4 pasi de acumulare, rată de învățare de 0.0001, weight decay de 0.01, programator al ratei de învățare Cosine Annealing cu jumătate de ciclu și cu warmup pentru 400 de pasi de modificare a parametrilor. Pentru QLORA, am păstrat un rang de 32, alpha de 64, inițializarea propusă în lucrarea LoRA, rslora și dropout de 0.01, aplicând LoRA pe toate straturile complet conectate. În continuare păstrez cuantizarea normal float 4 și cuantizarea dublă.

În primul rand am schimbat programatorul ratei de învățare din Cosine Annealing în Linear Annealing [65] cu warmup deoarece am observat că loss-ul (funcția de pierdere) se plafonează în prima parte a jumătății de ciclu cosinus, moment în care rata de învățare scade mult mai lent decât la Linear Annealing și începe să scadă când și rata de de învățare scade mai accentuat, în ultima parte a semi-ciclului. Am obținut rezultate mai bune cu acest programator (learning rate scheduler), dar tot nu am ajuns la convergența pe care mi-am dorit-o, așa că am început să cresc rangul și alpha de la adapterii LoRA pentru a face modelul mai potent. Am început prin a creste treptat rangul și alpha, păstrând proporția ca alpha să fie de două ori mai mare decât rangul. Cu aceasta metoda am observat îmbunătățiri doar în prima parte a procesului de antrenare. Din această cauză am dedus că rangul aduce beneficiul de a învață mai bine distribuția, dar modelul se supra-specializează în final. Așadar, am decis să scad parametrul alpha pentru a minimiza impactul LoRA. Am obținut rezultate mai bune cu un rang și un alpha de 64, iar cu parametrii setați la 128 am obținut rezultate și mai bune. De la 256 pentru rang și alpha, modelul nu a mai obținut performanțe satisfăcătoare. Pentru Linear Annealing am încercat variante de start pentru rata de învățare de 0.00005 și 0.0003 dar nu au obținut rezultate mai bune decât rata de învățare de 0.0001. În momentul în care cresc rata de învățare, modelul intră în divergență, iar când o reduc, convergența este mult prea lentă. Nu am observat îmbunătățiri în generalizare prin antrenarea pentru mai mult de o epoca.

Varianta câștigătoare pentru spațiul hiper-parametrilor pentru acest model a rămas să fie:

- o rata de învățare de 0.0001
- Linear Annealing

- weight decay de 0.01
- mărimea batch-ului este de 4 cu 4 pasi de acumulare
- 400 de pasi pentru creșterea progresivă în mod linear de la 0 la rata de învățare setată (linear warmup)
- rang 128 cu alpha 128 pentru LoRA
- dropout 0.01 pentru Lora
- cu rslora și inițializarea matricilor LoRA cu varianta originală

5.1.2 Flan-T5 Large

Flan-T5 Large reprezintă un model mai mare decât MBART Large, acesta având 780 de milioane de parametrii, deci timpul de execuție pentru antrenate este mai mare, iar numărul de experimente pentru optimizarea hiper-parametrilor a fost mai mic. În cazul acestui model, am început cu un rang de 8 și alpha de 16 pentru LoRA, rata de învățare de 0.0001, weight decay de 0.01 și learning rate scheduler Cosine Annealing cu jumătate de ciclu și 400 de pasi de warmup. Cu hiper-parametrii setați inițial, modelul nu a putut învață distribuția datelor, așa că am crescut la un rang de 16, respectiv 32, păstrând regularizarea. De asemena, convergența era destul de lentă așa că am crescut rata de învățare la 0.0003 (cu o rata de învățare mai mare de 0.0003, modelul intra în divergență).

Pentru Flan-T5 Large, cel mai bun experiment a avut următorii hiper-parametrii:

- o rată de învățare de 0.0003
- Cosine Annealing cu 0.5 ciclii
- weight decay de 0.01
- mărimea batch-ului este de 4 cu 4 pasi de acumulare
- 400 de pasi pentru creșterea progresivă în mod linear de la 0 la rata de învățare setată (linear warmup)
- rang 32 cu alpha 64 pentru LoRA
- dropout 0.01 pentru Lora
- cu rslora și inițializarea matricilor LoRA cu varianta originală

5.1.3 Baseline

Ca baseline, folosesc modelul GPT2 Medium pre-antrenat pentru limba romana și ajustat pentru sumarizare abstractivă în limba română, citat în capitolul 2. Acesta reprezintă un baseline solid pentru a demonstra eficacitatea metodelor propuse. Modelul a fost antrenat pe un subset de date din setul total de date pe care îl folosesc, doar pe știri având că sursă publicația alephnews.ro. Au fost antrenat pe un context de 724 token-uri, context ce l-am setat și eu în momentul evaluării sale. Această variantă este cea mai bună dintre cele fară token-uri de control conform autorilor.

5.1.4 Generarea rezumatelor pentru calcularea metricilor de evaluare

Asemănător cu capitolul anterior, am utilizat metricile ROUGE și BERT Score pentru evaluarea rezumatelor generate de către modele. Pentru obținerea celor mai bune rezultate, a trebuit să realizez o optimizare a parametrilor de generare. Metodele pentru producerea rezumatelor au fost Greedy și Beam Search Multinomial-Sampling (nu este prezis cel mai probabil token, ci se alege dintr-o distribuție asupra vocabularului modelului). Pentru a găsi cea mai bună combinație de hiper-parametrii, am testat modelele pe un subset de 400 de exemple din setul de validare, de unde am dedus faptul că:

- numărul de token-uri pentru fiecare rezumat generat va fi între 35 și 250 de token-uri
- modelul trebuie limitat în a nu folosi o secvență de 5 token-uri consecutive de mai multe ori, pentru a nu se repeta
- pentru Beam Search setez parametru de penalizare la 1.5, pentru a încuraja modelul să producă rezumate puțin mai lungi
- echilibrul între performanță și resurse hardware l-am găsit fiind de 5 beam-uri (pentru Beam Search)
- pentru diversitate în rezumatele generate și pentru a încuraja abstractivitatea și parafrazarea textelor folosesc varianta cu sampling pentru Beam Search

Având parametrii optimi pentru generare, am evaluat modelele pe subsetul de testare de 9093 de exemple formate din documentul original și rezumatul de referință aferent. Pentru baseline nu am observat îmbunătățiri prin creșterea sau micșorarea parametrului de penalizare a lungimi, așadar l-am păstrat la 1.0. Totodată, contextul maxim, incluzând documentul și rezumatul, fiind de 1024, numărul maxim de token-uri pentru rezumat este setat la 200. Din rezultate obținute (tabela 5.1 si tabela 5.2), se poate observa că ambele modele au depășit baseline-ul, iar MBART este cel mai bun model din punctul de vedere al metricilor calculate. De asemenea, trebuie menționat faptul că modelul baseline s-a descurcat foarte bine, având în vedere că a fost antrenat pe un set de date de doua ori mai mic ce conținea știri doar dintr-o sursă.

Model	Algoritm de decodare	ROUGE-1 (%)	ROUGE-2 (%)	ROUGE-L (%)	ROUGE-Lsum (%)
GPT2 Medium (baseline)	Greedy	31.43	14.36	23.05	27.60
	Beam Search	33.3	15.76	24.05	29.26
MBART Large	Greedy	37.89	19.92	28.14	33.44
	Beam Search	**38.69**	**20.60**	**28.69**	**34.21**
Flan-T5 Large	Greedy	37.18	19.33	27.56	32.96
	Beam Search	37.85	19.82	27.79	33.52

Tabela 5.1: Metricile ROUGE pentru fiecare varianta de model MBart Large antrenata: cu context de 512 de token-uri, cu context de 512 de token-uri cu preprocesare TextRank, cu context de 768 de token-uri si cu context de 768 de token-uri cu preprocesare TextRank

Model	Algoritm de decodare	Precizie (%)	Recall (%)	F1-score (%)
GPT2 Medium (baseline)	Greedy	69.79	69.80	69.71
	Beam Search	69.97	71.20	70.52
MBART Large	Greedy	**74.06**	73.71	73.82
	Beam Search	73.77	**74.07**	**73.86**
Flan-T5 Large	Greedy	73.43	73.38	73.34
	Beam Search	73.04	73.91	73.42

Tabela 5.2: Metricile BERT Score pentru fiecare varianta de model MBart Large antrenata: cu context de 512 de token-uri, cu context de 512 de token-uri cu preprocesare TextRank, cu context de 768 de token-uri si cu context de 768 de token-uri cu preprocesare TextRank

5.2 Ajustare cu Direct Preference Optimization

5.2.1 Realizarea setului de date în concordonață cu preferințele umane

Metoda de de aliniere a modelelor la preferințele umane, Direct Preference Optimization, presupune un format anume de set de date prin intermediul căruia modelul politică trebuie să învețe. Pentru tema lucrării, rezumarea automată a textelor, un exemplu trebuie să aibă trei componente:

1. documentul ce trebuie sumarizat
2. un rezumat catalogat ca fiind mai bun

3. un rezumat clasificat ca fiind mai slab decât celălalt

Pentru realizarea setului de date aleg să iau drept soluții candidat pentru rezumatul mai bun și pentru rezumatul mai slab 5 rezumate generate de cel mai bun model antrenat (MBART Large cu context de 1024) produse folosind Beam Search cu sampling cu penalizare pentru lungimea secvenței între 1.5 și 2.0, deoarece am observat că modelul tinde să genereze rezumate mai scurte, și, în același timp, modelul nu are voie să genereze secvențe de 3 token-uri de mai multe ori și secvențe de token-uri din datele de intrare ce au lungimi intre 10 și 20 pentru a stimula abstractivitatea și parafrazarea rezumatelor.

Pe lange cele 5 rezumate, adaug că posibilă soluție și rezumatul de referință, obținut în mod automat în aceeași manieră ca în capitolul 3. Motivul pentru care decid să generez rezumate din modelul politică este dimensiunea redusă a setului de date rezervat pentru această abordare de doar 933 de exemple. Astfel, obținerea rezumatelor candidat din modele mai mari și mai potente, desi posibil mai calitative, nu ar duce la o generalizare mai bună tocmai din cauza setului de date foarte mic. Abordarea se bazează pe probabilitatea ca rezumatele să fie generate de modelul politica sau nu, deci probabilitatea pentru rezumatele mult mai calitative generate de modele mai capabile ar fi foarte mică la început și optimizarea modelului înspre acele rezumate ar fi predispusă la overfitting. Așadar, decid să selectez rezumate din spațiul expus la început.

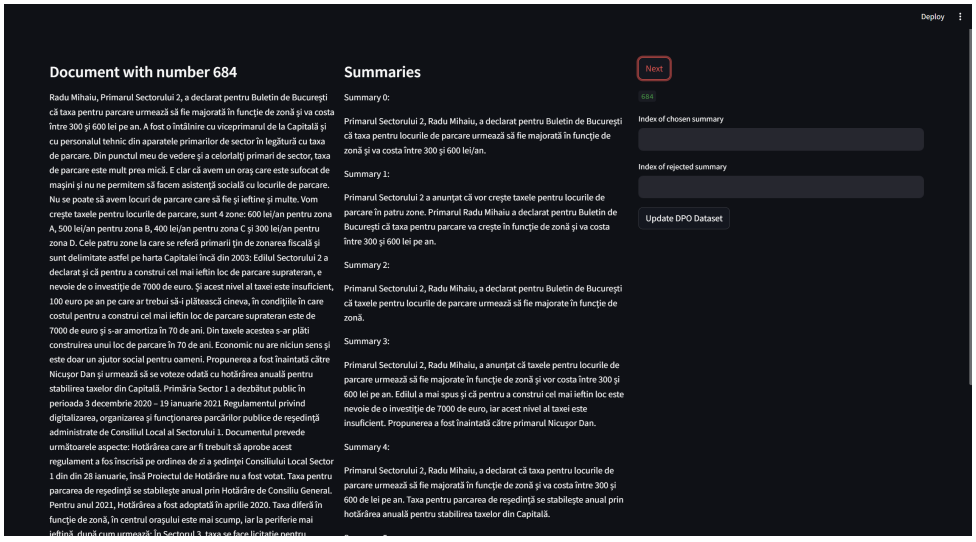

Figura 5.1: Figura ilustreaza aplicatia de selectare a datelor pentru Direct Preference Optimization

Pentru această etapă am realizat o aplicație în Streamlit [66] pentru a face procesul de selecție mult mai ușor. Am structurat pagina în 3 coloane pentru o citire mai lina a documentului și a rezumatelor. În prima coloană am așezat documentul ce are scopul de a fi rezumat, în coloana doi afișez toate cele 6 rezumate candidat, iar coloana 3 conține 2 textbox-uri unde sunt introduși indicii rezumatelor alese ca fiind mai bun, respectiv mai slab. De asemena, pe lange cele precizate, coloana 3 conține un buton pentru actualizarea

exemplului curent al setului de date pentru Direct Preference Optimization și un buton pentru trecerea la exemplul următor. În cazul rezumatelor care nu au atins rezultatele așteptate de către mine, au fost ușor ajustate prin adăugare, scoatere sau corectare a conținutului rezumatului ales ca fiind mai bun (figura 5.1)

5.2.2 Observații pentru rezumatele evaluate

Trecând manual prin aproximativ 900 de exemple, mi-am putut forma o opinie clară și obiectivă asupra rezumatelor generate. În general, modelul produce rezumate ce respectă foarte bine ideea principală a textului original iar greșelile gramaticale sunt destul de rare. Nivelul detaliilor relevante este unul bun, cu precădere în cazul documentelor mai scurte. În cazul documentelor mai lungi, nivelul detaliilor scade, ceea ce denotă că nu reușește să condenseze așa de bine ideile. Coerența și logica textelor generate sunt în mare parte destul de bune, modelul suferind în cazul în care este forțat să fie mai abstractiv și să condenseze informația, unde se pierd aceste calități. Cea mai mare problema rămâne să fie parafrazarea și nivelul de abstractivitate a sumarurilor generate, fapt ce se datorează setului de date unde în multe situații, propoziții din rezumatul de referință sunt doar extrase din documentul original sau sunt minim parafrazate. Cu toate acestea, modelul reușește câteodată să combine mai multe propoziții într-un singura, punând virgulă între ele sau eliminând din acestea cuvinte și expresii irelevante.

Intenția mea principală a fost să îmbunătățesc modelul acolo unde acesta nu performează așa de bine, și anume, în condensarea conținutului prin parafrazare, păstrând pe cât posibil coerența și logica textului. Pentru aceasta a trebuit să fac compromisuri în ceea ce privește erorile gramaticale și de coerența, pe care le-am corectat ulterior minimal. Astfel, intervenția mea a fost una redusă, fară să afectez prea tare scorurile de probabilitate a rezumatelor și încadrându-mă în caracteristicile setului de date liliputan. De asemenea, prin această abordare am încercat să elimin din bias-urile provenite din setul de date pentru antrenarea supervizată. Spre exemplu, în unele articole de la publicația b365.ro, textele aveau o doză de sarcasm, evidențiat prin repetarea semnelor de punctuație „.." sau „...", lucru ce se propagă și în unele texte generate de model. Totodată, am vrut să elimin sintagmele de forma „conform Agerpres", „afirmă Administrația Prezidențială", „potrivit Digi24" etc. Prin eliminarea acestor tipuri de sintagme, am dorit extinderea capabilităților modelului și pentru alte tipuri de texte, nu doar articole de știri.

5.2.3 Antrenarea cu Direct Preference Optimization

Pentru rafinarea modelului cu Direct Preference Optimization am utilizat librăria TRL de la Huggingface, mai exact Trainer-ul pentru DPO denumit DPOTrainer. Acesta implementează algoritmul și asigură diferite opțiuni de personalizare. Având modelul antrenat în mod supervizat cu QLORA, optez în a adăuga de două ori același LoRA

adapter utilizat în prima etapă a procesului, unul devenind referință, ce trebuie luat ca pivot în aceasta abordare, și celălalt devenind politica, ce trebuie optimizat prin Direct Preference Optimization. În acest fel, mă feresc de eroarea de decuantizare pe care o voi obține în cazul în care unesc modelul de bază cu LoRA adapters înainte de a aplica algoritmul. Setul de datele îl împart în două secțiuni: antrenare și testare, cu 793 de exemple pentru antrenare și 140 pentru testare.

Fiind într-o fază avansată a antrenării, folosesc o rată de învățare mai mică de 0.000005 cu Cosine Annealing și cu warmup linear pentru 10% din pașii de actualizare a parametrilor. Algoritmul de optimizare folosit este în continuare AdamW. Având un set de date mult mai mic, antrenez pentru 5 epoci cu batch de mărime 3 cu 8 pasi de acumulare a gradientului și salvez și evaluez modelul la fiecare 10 pasi de actualizare a parametrilor. Încep cu parametrul pentru deviația politicii de referință, beta la valoare 0.1. Nu am observat performanțe mai bune pentru o rată de învățare diferită și nici cu alt programator al ratei de învățare. Deoarece setul de date este de dimensiuni foarte reduse, există riscul crescut de overfitting, așadar, cresc treptat parametrul beta la 0.3, respectiv 0.5 și 0.7.

5.2.4 Rezultatele obținute cu Direct Preference Optimization

Criteriul de evaluare pentru interpretarea rezultatelor pe care l-am considerat cel mai potrivit a fost reprezentat de două metrici: diferența în medie dintre recompensele rezumatelor considerate mai bune și recompensele rezumatelor mai slabe, denumită în nomenclatura DPOTrainer HuggingFace rewards/margins, și media momentelor în care rezumatele mai bune au avut o recompensă mai mare decât cea a rezumatelor mai slabe, denumită rewards/accuracies. Din acest punct de vedere, cele mai bune rezultate pentru subsetul de testare le obțin cu beta de 0.5, pentru checkpoint-ul aferent la pasul de antrenare cu numărul 120. (vezi figura 5.2 și figura 5.3) Alegerea celei mai bune combinații de hiper-parametrii și a celui mai bun checkpoint pentru aceasta am făcut-o pe baza faptului că ambele metrice ar trebui să fie la un nivel ridicat în același timp. Având în vedere setul mic de date pentru evaluare, cea de-a doua metrică a oscilat destul de mult, fapt ce se poate observa în figurile prezentate.

5.2.5 Evaluarea și comparația cu modelul antrenat doar în mod supervizat

Antrenare cu Direct Preference Optimization a făcut ca modelul politică să devieze puțin față de distribuția pe care a fost antrenat in primă fază, astfel, pentru a verifica, într-adevăr efectul metodei aplicate, am realizat o serie de experimente prin care constrâng modelele să producă rezumate mai abstractive prin parametrul funcției generate() din mediul Huggingface, encoder_no_repeat_ngram_size. Parametrul constrânge modelul

Figura 5.2: Metrica rewards/margins pentru subsetul de evaluare pentru experimentele cu beta de 0.1, 0.3, 0.5 si 0.7.

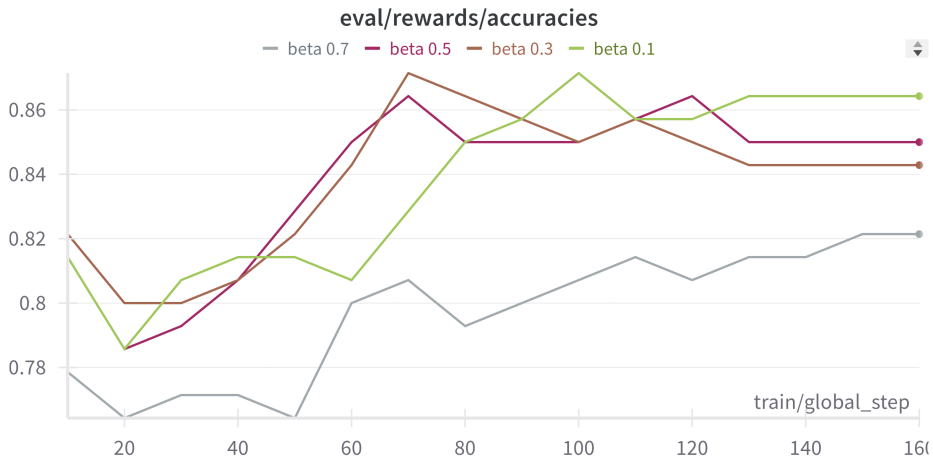

Figura 5.3: Metrica rewards/accuracies pentru subsetul de evaluare pentru experimentele cu beta de 0.1, 0.3, 0.5 si 0.7.

cu care se realizează inferență să nu folosească în textul generat o secvență de token-uri ce se află și în datele de intrare de cel puțin valoarea acestuia. În aceste condiții, am căutat prin intermediul unui subset de 400 de exemple din setul de validare rezervat pentru antrenare supervizata cei mai buni parametrii pentru modelul politică obținut și pentru modelul de referință. Modelul politică are rezultate mai bune cu un parametru de penalizare a lungimi aferent Beam Search, de 1.0, față de 1.5 pentru modelul referință, deoarece în momentul selecției datelor am preferat rezumatele mai lungi, ce cuprindeau mai multe detalii relevante pentru a construi un context mai bine definit sumarului. De asemenea, am testat modelul politică pentru a-l compara cu modelul referință și fară nicio constrângere cu privire la textul de intrare, dar am folosit, totuși, Beam Search cu Sampling pentru diversitate. În tabelele 5.3 și 5.4 se pot observa rezultatele obținute

pentru cele două metrici. Din acestea, se observă că modelul ajustat cu DPO se descurcă mai bine decât cel antrenat doar în mod supervizat atunci când este constrâns să fie mai abstractiv. Din propriile observații, ajustarea cu Direct Preference Optimization nu a reușit să crească într-un mod impresionant nivelul de abstractivitate, textele fiind aproximativ în aceeași măsură de abstractive și parafrazate, însă abordarea a adus beneficiul de a expune într-un mod mai corect detaliile atunci când este constrâns să fie mai abstractiv. În schimb, bias-urile provenite din setul de date de antrenare supervizată au fost ameliorate destul de bine. În exemplul din Anexa 1 .1 se remarcă faptul că rezumatele generate de modelul politică nu au sintagme de forma „, a afirmat" sau semne de punctuație redundante, iar informațiile expuse, în special când este restricționat mai tare să fie abstractiv, sunt mai corecte decât cele din modelul de referință. Așadar, pot afirma că abordarea cu Direct Preference Optimization a obținut rezultate bune, având în vedere setul mic de date.

Algoritm de decodare	**Model**	**ROUGE-1 (%)**	**ROUGE-2 (%)**	**ROUGE-L (%)**	**ROUGE-Lsum (%)**
Beam Search cu sampling	Referință	**38.69**	**20.60**	**28.69**	**34.21**
	Politică	38.29	20.08	27.54	33.97
Beam Search encoder_no_repeat_ngram_size 10	Referință	36.86	17.57	26.49	32.11
	Politică	38.00	17.73	26.25	33.09
Beam Search encoder_no_repeat_ngram_size 7	Referință	35.84	15.61	25.14	30.81
	Politică	37.25	15.94	25.03	31.96
Beam Search encoder_no_repeat_ngram_size 5	Referință	33.76	12.65	22.91	28.55
	Politică	35.55	13.11	23.06	29.95

Tabela 5.3: Metricile ROUGE pentru modelul politică și cel de referință pentru metodele de decodare menționate.

5.2.6 Inferență pe documente lungi

În continuarea evaluării performanțelor abordări cu DPO, am testat inferență modelului politică și a celui de referință pe 773 de texte lungi, cu peste 1800 de cuvinte. Totodată, am evaluat eficacitatea preprocesării TextRank pe texte foarte lungi, fără antrenarea în prealabil cu acest tip de date. Preprocesarea cu TextRank a fost realizată întocmai ca în capitolul 4. Am continuat cu aproximativ aceiași hiper-parametrii pentru procesul de generare: parametrul de penalizare a lungimii este setat la 1.5 pentru modelul de referință, respectiv la 1.0 la modelul politică, textele generate vor avea între 35 și 250 de token-uri, iar numărul de beam-uri este este setat la 4 în ambele cazuri. Din tabelele 5.5 si 5.6 reiese că, deși rezultatele nu sunt prea bune per total, modelul politică se descurcă puțin mai bine pe documentele lungi decat cel de referință, iar preprocesarea cu TextRank a avut

Algoritm de decodare	Model	Precizie (%)	Recall (%)	F1-score (%)
Beam Search cu sampling	Referință	73.77	74.07	**73.86**
	Politică	71.82	**75.00**	73.34
Beam Search encoder_no_repeat_ngram_size 10	Referință	**74.53**	72.99	73.71
	Politică	72.88	74.31	73.54
Beam Search encoder_no_repeat_ngram_size 7	Referință	74.38	72.56	73.42
	Politică	72.86	73.95	73.36
Beam Search encoder_no_repeat_ngram_size 5	Referință	73.74	71.76	72.70
	Politică	72.44	73.22	72.79

Tabela 5.4: Metricile BERT Score pentru modelul politică și cel de referință pentru metodele de decodare menționate.

un impact pozitiv asupra performanței, atât în privința poticii, cât și a referinței. Din observațiile proprii, textele folosite în acest set de testare deviază putin de la distribuția setului de date original (informația principală nu mai este expusă in majoritatea cazurilor la început), fapt pentru care preprocesarea TextRank oferă rezultate mai bune decât în capitolul 4.

Tip date	Model	ROUGE-1 (%)	ROUGE-2 (%)	ROUGE-L (%)	ROUGE-Lsum (%)
Date normale	Politică	25.13	5.87	15.16	21.96
	Referință	24.21	5.59	15.06	21.03
Preprocesare TextRank 1024	Politică	25.76	5.89	15.04	22.23
	Referință	24.47	5.25	14.47	21.03
Preprocesare TextRank 768	Politică	26.22	6.51	15.58	22.72
	Referință	25.56	6.39	15.57	22.07
Preprocesare TextRank 512	Politică	**26.89**	**7.18**	**15.94**	**23.32**
	Referință	26.04	7.07	15.93	22.55

Tabela 5.5: Metricile ROUGE calculate pentru modelele politică si de referință pentru texte foarte lungi pentru date nepreprocesate și preprocesate cu TextRank la 1024, 768, respectiv 512 token-uri.

Tip de date	Model	Precizie (%)	Recall (%)	F1-score (%)
Date normale	Politică	67.09	67.99	67.51
	Referință	67.67	67.34	67.47
Preprocesare TextRank 1024	Politică	67.38	68.53	67.92
	Referință	67.76	67.66	67.68
Preprocesare TextRank 768	Politică	67.60	68.76	68.14
	Referință	68.28	68.01	68.10
Preprocesare TextRank 512	Politică	67.82	**68.92**	**68.33**
	Referință	**68.52**	68.11	68.27

Tabela 5.6: Metricile BERT Score calculate pentru modelele politică si de referință pentru texte foarte lungi pentru date nepreprocesate și preprocesate cu TextRank la 1024, 768, respectiv 512 token-uri.

Capitolul 6

Aplicație Web

Pentru a oferi practicabilitate lucrării de licență, am realizat o aplicație web, unde am folosit limbajul de programare Python, framework-ul Flask pentru backend și Bootstrap pentru frontend. Baza de date folosită a fost MongoDB, conectarea realizându-se prin modulul pymongo din Python.

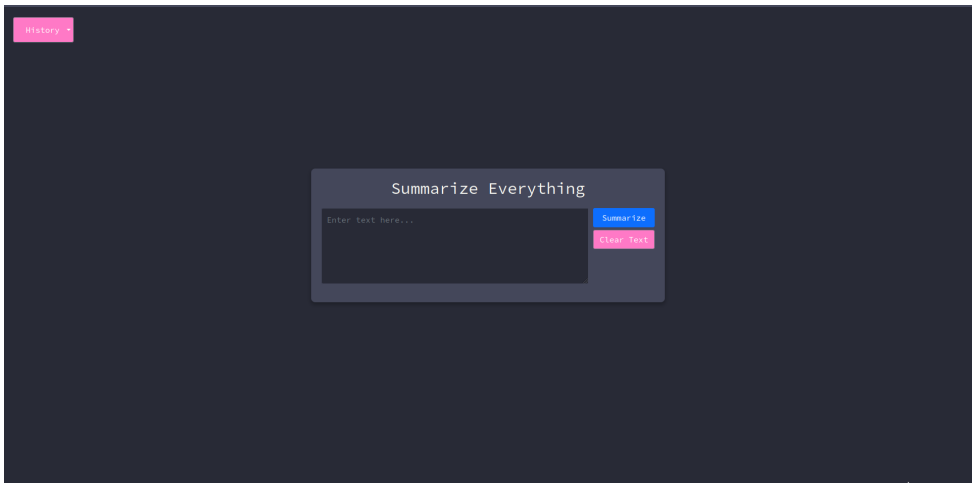

Figura 6.1: În figură este ilustrată pagina principală a aplicației.

Aplicația ilustrează principala funcționalitate a lucrării de licență, rezumarea automată de texte în limba română (figura 6.1).

Pentru acest lucru, obiectul principal al acesteia este reprezentat de un textbox, unde se poate introduce documentul ce se dorește a fi rezumat, un buton pentru ștergerea totală a textului de intrare și un buton ce declanșează generarea rezumatului de către model.

În plus de principala funcționalitate, utilizatorii mai au opțiunea de vizualizare ultimelor 5 prompt-uri efectuate cu ajutorul funcționalității de „History”.(figura 6.2) Datele sunt introduse în secțiunea „History” prin apăsarea butonului „Summarize”.

Pentru stocarea istoricului folosesc o colecție din baza de date MongoDB, unde sunt introduse textele documentului și rezumatului, și, automat, un id al intrării din colecție. Aceasta prezintă, o dată cu apăsarea butonului „History”, ultimele 5 prompt-uri și răspunsurile

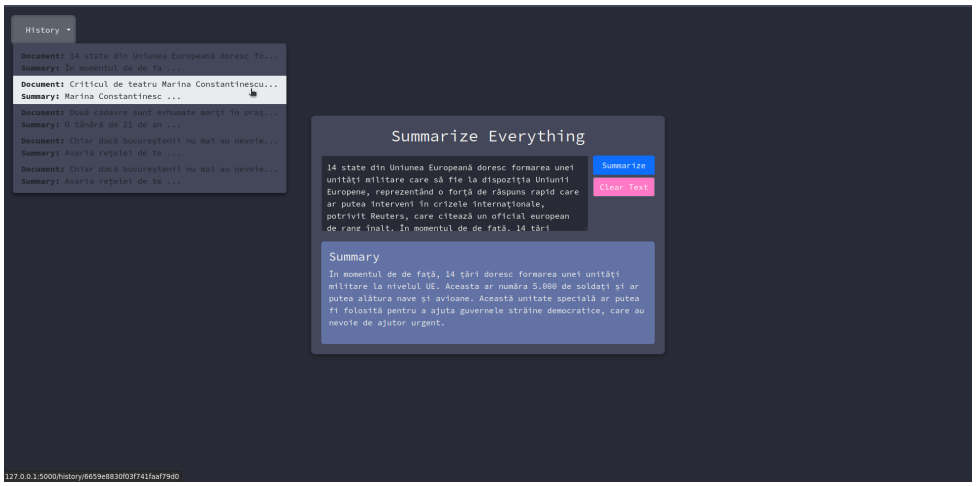

Figura 6.2: Figura ilustrază vizualizara istoricului în timp ce utilizatorul se află pe pagina principală a aplicației.

într-o formă scurată, primele 40 de caractere pentru document și primele 20 pentru rezumat (figura 6.3). Click-ul pe una dintre intrările istoricului direcționează utilizatorul către o pagina noua, pagina ce afișează în întregime documentul și rezumatul. Pe această pagină există și un buton de „Înapoi”, pentru ușurința utilizării și fluența aplicației.

Figura 6.3: Figura prezintă pagina de afișare a unui singur exemplu din istoric, în totalitate.

Capitolul 7

Concluzii si dezvoltări uleterioare

7.1 Limitări

Pe lângă limitările evidente ce țin de infrastructura hardware, lucrarea a fost condiționată și de constrângeri ce țin de altă natură. În primul rând, setul de date, deși este unul vast ce acoperă o diversitatea largă de articole de știri, conține texte cu bias-ul provenit de categoria din care fac parte: în majoritatea a cazurilor, cele mai importante informații sunt expuse la începutul documentului, din cauza aceasta modelelor antrenate le este dificil să asigure un rezumat cuprinzător și pe texte de altă natură cum ar fi texte legale lungi sau povestiri. Acest bias al setului de date a fost și principalul motiv pentru care preprocesarea cu TextRank nu a adus beneficii majore. Informația principală este deja, în mare parte, conținută la începutul textului. Pe de altă parte, preprocesarea TextRank a avut un impact pozitiv când am abordat strategia de inferență pe texte foarte lungi, distribuție puțin diferită față de cea a setului de date rezervat pentru antrenare supervizată. O altă limitare a setului de date o reprezintă abstractivitatea mai scăzută a acestuia. În destule cazuri, unele propoziții din rezumatul de referință sunt doar extrase din textul original sau parafrazate minimal, de aici și parafrazarea diminuată a sumarelor generate. De asemenea, setul de date fiind extras automat și fiind vast, există situații în care se găsesc erori gramaticale și dezacorduri în textele colectate. O altă barieră ce a avut un impact asupra dezvoltării lucrării o reprezintă lipsa evaluării umane. Metricile prezentate, deși sunt reprezentative în fazele incipiente și intermediare ale dezvoltării, pe măsură ce se înaintează în complexitate, nu mai reflectă în totalitate calitatea rezumatelor generate din cauza faptului că sunt foarte condiționate de calitatea celor de referință. Cel mai potrivit exemplu pentru acest argument este reprezentat de evaluarea modelului politică după antrenarea cu DPO, unde acesta s-a depărtat puțin de distribuția originală iar evaluarea cu metricile automate nu au mai reflectat în totalitate calitatea rezumatelor. În plus, deși abordare Direct Preference Optimization a avut rezultate respectabile, având în vedere mărimea setului de date, aceasta ar fi fost mult mai eficientă cu un set de

date considerabil mai mare. Tot în ceea ce privește setul de date pentru DPO, este foarte probabil ca textele alese ca fiind mai bune, respectiv mai slabe să aibă o doză de subiectivism din cauza faptului că au fost evaluate doar de mine, limitare ce ar putea fi depășită cu ajutorul mai multor adnotatori.

7.2 Dezvoltări ulerioare

Ca și dezvoltări ulterioare pot amintii cu siguranță, preprocesarea cu TextRank pe text de altă natură ce nu conțin informația principală la început. Desigur aceasta constă și în construirea unui set de date nou pentru acest scop. Totodată, se pot încercă și alte metrici de similaritate între propoziții pentru algoritmul TextRank. De asemenea, modelele prezentate în această lucrare nu au fost pre-antrenate în mod special pentru această sarcină, așadar, o dezvoltare ulterioară interesantă o poate reprezenta pre-antrenarea și specializarea pentru limba română a unui model cu arhitectură specifică pentru această sarcină, de exemplu arhitectura Pre-training with Extracted Gap-sentences for Abstractive Summarization (PEGASUS). Pentru partea de aliniere a rezumatelor la preferințele umane, s-ar putea extinde setul de date cu mult mai multe exemple etichetate de mai mulți adnotatori sau s-ar putea experimenta cu [67], ce necesită doar etichetarea exemplelor ca fiind bune sau nu, fapt ce ar diminua semnificativ procesul de adnotare.

7.3 Concluzii

Cum am precizat și în capitolul 1, prezenta lucrare se concentrează pe dezbaterea temei de rezumare automată de texte, prin intermediul următoarelor puncte atinse: realizarea unui set de date pentru antrenarea supervizată, studierea impactului preprocesarii cu TextRank asupra calității rezumatelor, îmbunătățirea rezumatelor generate folosind Direct Preference Optimization și realizarea unei aplicații web destinate prezentării temei.

Mi s-a părut interesant faptul că preferințele mele în ceea ce privește calitatea rezumatelor au reușit să fie expuse și în textele generate, chiar dacă numărul de exemple a fost unul mic, iar dimensiunea modelului antrenat a fost considerabilă. Aplicația web pe care am dezvoltat-o plusează cu practicabilitate metodelor și redă clar funcționalitatea temei abordate. Consider că am reușit să depășesc provocările survenite în procesul de dezvoltare a lucrări, cu privire la dimensiunea modelelor și a datelor, și am reușit să învăț să utilizez tehnici noi cum ar fi LoRA, QLORA, gradient checkpointing pentru a rezolva aceste probleme. În concluzie, pot să afirm că am obținut rezultate foarte bune cu abordările prezentate în această lucrare, luând în considerare limitările expuse.

Bibliografie

[1] Shashi Narayan, Shay B. Cohen și Mirella Lapata, „Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization", în *ArXiv* abs/1808.08745 (2018).

[2] Ramesh Nallapati, Bowen Zhou, Cicero Nogueira dos santos, Caglar Gulcehre și Bing Xiang, *Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond*, 2016, arXiv: 1602.06023 [cs.CL].

[3] Ilya Sutskever, Oriol Vinyals și Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, 2014, arXiv: 1409.3215 [cs.CL].

[4] Jingqing Zhang, Yao Zhao, Mohammad Saleh și Peter J. Liu, *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*, 2020, arXiv: 1912.08777 [cs.CL].

[5] ReaderBench, *ro-text-summarization*, accesat la data 01-06-2024, URL: https://huggingface.co/datasets/readerbench/ro-text-summarization.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser și Illia Polosukhin, *Attention Is All You Need*, 2023, arXiv: 1706.03762 [cs.CL].

[7] Rada Mihalcea și Paul Tarau, „TextRank: Bringing Order into Text", în *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, ed. de Dekang Lin și Dekai Wu, Barcelona, Spain: Association for Computational Linguistics, Iul. 2004, pp. 404–411, URL: https://aclanthology.org/W04-3252.

[8] Lawrence Page, Sergey Brin, Rajeev Motwani și Terry Winograd, „The PageRank Citation Ranking : Bringing Order to the Web", în *The Web Conference*, 1999, URL: https://api.semanticscholar.org/CorpusID:1508503.

[9] Ilya Loshchilov și Frank Hutter, *SGDR: Stochastic Gradient Descent with Warm Restarts*, 2017, arXiv: 1608.03983 [cs.LG].

[10] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov și Luke Zettlemoyer, *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*, 2019, arXiv: 1910.13461 [cs.CL].

[11] Dan Hendrycks și Kevin Gimpel, *Gaussian Error Linear Units (GELUs)*, 2023, arXiv: 1606.08415 [cs.LG].

[12] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis și Luke Zettlemoyer, *Multilingual Denoising Pre-training for Neural Machine Translation*, 2020, arXiv: 2001.08210 [cs.CL].

[13] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li și Peter J. Liu, *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*, 2023, arXiv: 1910.10683 [cs.LG].

[14] Peter Shaw, Jakob Uszkoreit și Ashish Vaswani, *Self-Attention with Relative Position Representations*, 2018, arXiv: 1803.02155 [cs.CL].

[15] Tim Salimans și Yaroslav Bulatov, *GitHub - cybertronai/gradient-checkpointing: Make huge neural nets fit in memory*, accesat la data 01-06-2024, URL: https://github.com/cybertronai/gradient-checkpointing.

[16] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei și Ilya Sutskever, „Language Models are Unsupervised Multitask Learners", în 2019, URL: https://api.semanticscholar.org/CorpusID:160025533.

[17] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave și Guillaume Lample, *LLaMA: Open and Efficient Foundation Language Models*, 2023, arXiv: 2302.13971 [cs.CL].

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee și Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019, arXiv: 1810.04805 [cs.CL].

[19] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma și Radu Soricut, *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*, 2020, arXiv: 1909.11942 [cs.CL].

[20] Chin-Yew Lin, „ROUGE: A Package for Automatic Evaluation of summaries", în Ian. 2004, p. 10.

[21] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger și Yoav Artzi, *BERTScore: Evaluating Text Generation with BERT*, 2020, arXiv: 1904.09675 [cs.CL].

[22] Ilya Loshchilov și Frank Hutter, *Decoupled Weight Decay Regularization*, 2019, arXiv: 1711.05101 [cs.LG].

[23] Diederik P. Kingma și Jimmy Ba, *Adam: A Method for Stochastic Optimization*, 2017, arXiv: 1412.6980 [cs.LG].

[24] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang și Weizhu Chen, *LoRA: Low-Rank Adaptation of Large Language Models*, 2021, arXiv: 2106.09685 [cs.CL].

[25] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman și Luke Zettlemoyer, *QLoRA: Efficient Finetuning of Quantized LLMs*, 2023, arXiv: 2305.14314 [cs.LG].

[26] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike și Ryan Lowe, *Training language models to follow instructions with human feedback*, 2022, arXiv: 2203.02155 [cs.CL].

[27] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning și Chelsea Finn, *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*, 2023, arXiv: 2305.18290 [cs.LG].

[28] accesat la data 01-06-2024, URL: https://www.python.org/doc/.

[29] accesat la data 01-06-2024, URL: https://pytorch.org/docs/stable/index.html.

[30] accesat la data 01-06-2024, URL: https://pandas.pydata.org/docs/.

[31] accesat la data 01-06-2024, URL: https://numpy.org/doc/.

[32] accesat la data 01-06-2024, URL: https://matplotlib.org/stable/index.html.

[33] accesat la data 01-06-2024, URL: https://seaborn.pydata.org/.

[34] *MongoDB Documentation*, accesat la data 01-06-2024, URL: https://www.mongodb.com/docs/.

[35] accesat la data 01-06-2024, URL: https://pymongo.readthedocs.io/en/stable/.

[36] accesat la data 01-06-2024, URL: https://huggingface.co/docs/transformers/index.

[37] accesat la data 01-06-2024, URL: https://huggingface.co/docs/accelerate/index.

[38] accesat la data 01-06-2024, URL: https://huggingface.co/docs/peft/en/index.

[39] accesat la data 01-06-2024, URL: https://huggingface.co/docs/trl/en/index.

[40] accesat la data 01-06-2024, URL: https://huggingface.co/docs/evaluate/en/index.

[41] accesat la data 01-06-2024, URL: https://huggingface.co/spaces/evaluate-metric/rouge.

[42] accesat la data 01-06-2024, URL: https://huggingface.co/spaces/evaluate-metric/bertscore.

[43] accesat la data 01-06-2024, URL: https://flask.palletsprojects.com/en/3.0.x/.

[44] accesat la data 01-06-2024, URL: https://documentation.bloomreach.com/engagement/docs/jinja-syntax.

[45] accesat la data 01-06-2024, URL: https://werkzeug.palletsprojects.com/en/3.0.x/.

[46] accesat la data 01-06-2024, URL: https://getbootstrap.com/docs/5.3/getting-started/introduction/.

[47] Mihai Niculescu, Stefan Ruseti și Mihai Dascalu, „RoSummary: Control Tokens for Romanian News Summarization", în *Algorithms* 15 (Dec. 2022), p. 472, DOI: 10.3390/a15120472.

[48] Mihai Niculescu, Stefan Ruseti și Mihai Dascalu, „RoGPT2: Romanian GPT2 for Text Generation", în Nov. 2021, pp. 1154–1161, DOI: 10.1109/ICTAI52525.2021.00183.

[49] Andreea - Nicoleta Dutulescu, Mihai Dascalu și Stefan Ruseti, „Unsupervised Extractive Summarization with BERT", în *2022 24th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2022, pp. 158–164, DOI: 10.1109/SYNASC57785.2022.00032.

[50] accesat la data 01-06-2024, URL: https://b365.ro/.

[51] accesat la data 01-06-2024, URL: https://alephnews.ro/.

[52] Redacția Buletin de București, *Home*, accesat la data 01-06-2024, URL: https://buletin.de/bucuresti/.

[53] accesat la data 01-06-2024, URL: https://docs.python-requests.org/en/v2.0.0/.

[54] accesat la data 01-06-2024, URL: https://fake-useragent.readthedocs.io/en/latest/.

[55] accesat la data 01-06-2024, URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[56] accesat la data 01-06-2024, URL: https://pypi.org/project/demoji/.

[57] Karl Pearson, „Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity, and Panmixia", în *Philosophical Transactions of the Royal Society A* 187 (), pp. 253–318, URL: https://api.semanticscholar.org/CorpusID:119875807.

[58] accesat la data 01-06-2024, URL: https://derwen.ai/docs/ptr/sample/.

[59] accesat la data 01-06-2024, URL: https://spacy.io/usage/spacy-101.

[60] accesat la data 01-06-2024, URL: https://huggingface.co/facebook/mbart-large-cc25.

[61] Hugo Touvron et al., *Llama 2: Open Foundation and Fine-Tuned Chat Models*, 2023, arXiv: 2307.09288 [cs.CL].

[62] Intel Corporation, „bfloat16 - Hardware Numerics Definition", în ().

[63] Damjan Kalajdzievski, *A Rank Stabilization Scaling Factor for Fine-Tuning with LoRA*, 2023, arXiv: 2312.03732 [cs.CL].

[64] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le și Jason Wei, *Scaling Instruction-Finetuned Language Models*, 2022, arXiv: 2210.11416 [cs.LG].

[65] accesat la data 01-06-2024, URL: https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.LinearLR.html.

[66] accesat la data 01-06-2024, URL: https://docs.streamlit.io/.

[67] Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky și Douwe Kiela, *KTO: Model Alignment as Prospect Theoretic Optimization*, 2024, arXiv: 2402.01306 [cs.LG].

Anexa 1

.1 Comparație între modelul rafinat cu Direct Preference Optimization și cel antrenat doar în mod supervizat

Document:

„Administraţia Joseph Biden este îngrijorată că Beijingul ar putea recurge la preluarea controlului asupra Taiwanului, în contextul în care preşedintele Chinei, Xi Jinping, pare din ce în ce mai dispus să asume riscuri pentru a lăsa o moştenire politică importantă. ”China pare să se îndrepte de la o perioadă în care s-a mulţumit cu statutul Taiwanului spre o perioadă în care liderii chinezi sunt din ce în ce mai nerăbdători şi mai pregătiţi să testeze limitele şi să cocheteze cu ideea unificării, afirmă un oficial de rang înalt de la Washington, citat de cotidianul Financial Times. Oficialul american a subliniat că Administraţia Joseph Biden a ajuns la această concluzie după analizarea comportamentului Chinei din ultimele două luni. ”În contextul în care ne pregătim pentru o perioadă în care Xi Jinping probabil va intra în al treilea mandat, există preocupări că percepe progresele semnificative în privinţa Taiwanului ca fiind importante pentru legitimitatea şi moştenirea sa politică. Pare a fi pregătit să asume mai multe riscuri, subliniază oficialul american, sub protecţia anonimatului. Douăzeci de avioane militare chineze au pătruns vineri în zona de securitate aeriană din jurul Taiwanului, aceasta fiind cea mai mare incursiune de acest fel din ultima perioadă. Zborurile militare au avut loc la o zi după ce Statele Unite şi Taiwanul au stabilit să intensifice cooperarea între unităţile Pazei de Coastă.”

Rezumat generat de modelul antrenat cu DPO cu parametrul encoder_no_repeat_ngram_size setat la 10:

„Administrația Joe Biden este îngrijorată că China ar putea prelua controlul asupra Taiwanului, în contextul în care președinte Xi Jinping pare din ce în ce mai dispus să își asume riscuri. Oficialul american a subliniat că administrația Biden a ajuns la această concluzie după ce a analizat comportamentul Chinei din ultimele două luni. Douăzeci de avioane militare americane au pătruns vineri în zona aeriană din jurul Taiwanului.”

Rezumat generat de modelul antrenat doar in mod supervizat cu parametrul encoder_no_repeat_ngram_size setat la 10:

„Administrația Joseph Biden este într-o oarecare îngrijorare că Beijingul ar putea prelua controlul asupra Taiwanului, în condițiile în care președintele Xi Jinping pare din ce în ce mai dispus să își asume riscuri pentru a-și lăsa o moștenire politică importantă, afirmă

un înalt oficial american.”

Rezumat generat de modelul antrenat cu DPO cu parametrul encoder_no_repeat_ngram_size setat la 5:

„Administrația Joe Biden e îngrijorată de posibilitatea unei invazii militare a Chinei asupra Taiwanului. Un oficial american a declarat că Beijingul este pregătit să asume riscuri în acest sens. Zborurile au avut loc vineri, la o zi de când Washingtonul și Taiwanul și-au intensificat cooperarea în domeniul securității aeriene.”

Rezumat generat de modelul antrenat doar in mod supervizat cu parametrul encoder_no_repeat_ngram_size setat la 10:

„Administrația Joseph Bid este îngrijorarea că Beijingul ar încerca să recupereze controlul asupra Taiwanului. Xi Jinping pare mai dispus să se pregătească pentru al treilea mandat..Zborurile militare din jurul Taiwanului au avut loc vineri, la o zi de când Washingtonul și Taiwanul s-au stabilit să intensifice colaborarea.”

În toate cazurile textul a fost generat cu Beam Search cu 4 beam-uri și early stopping, cu un număr minim de token-uri ce trebuie generate de 35 iar numărul maxim fiind de 250. Totodată, modelul nu are voie să repete o secvență generată de acesta lungă de cel puțin 3 token-uri decât o data. Pe parcursul generării rezumatelor am păstrat pe cât posibil determinismul operației. Articolul de știri este preluat de la publicația alephnews.ro