UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Sistem Lingvistic Inteligent pentru Gramatica Limbii Române

Ioan-Florin-Cătălin NIȚU

Coordonator științific:

Conf. Dr. Ing. Traian-Eugen REBEDEA

BUCUREȘTI

2020

UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL DE CALCULATOARE

PROIECT DE DIPLOMĂ

Sistem Lingvistic Inteligent pentru Gramatica Limbii Române

Ioan-Florin-Cătălin NIȚU

Coordonator științific:

Conf. Dr. Ing. Traian-Eugen REBEDEA

BUCUREȘTI

2020

Andreei, lui Vali, surorii mele, părinților, familiei, profesorilor, colegilor și prietenilor care m-au ajutat să depășesc momentele întunecate ale vieții, căci fără sprijinul vostru în acești patru ani (și nu numai) nu aș fi reușit să ajung astăzi aici. Am învățat atât de multe de la voi!

I Sinopsis

Domeniul prelucrării limbajului natural nu este la fel de puternic dezvoltat pentru limba română așa cum este pentru altele, cum este limba engleză. Faptul de a scrie corect texte a fost mereu o necesitate, iar dezvoltarea unor instrumente care să fie de folos în această nevoie este critică. Sistemul de corectare propus primește o propoziție cu greșeli gramaticale și o corectează, folosind tehnologii de ultimă oră pentru a realiza această operație cum sunt modelele neurale bazate pe atenție ca Transformatoarele cu Codificator-Decodificator. Acestea sunt o piatră de temelie în dezvoltarea de instrumente inteligente pentru prelucrări – traducerea, rezumarea sau corectarea – de texte și sunt fundația pentru proiectul de față. Lucrarea folosește RONACC, primul corpus pentru corecții gramaticale în română pentru modelarea, antrenarea, testarea și validarea proiectului. Folosind un set de date foarte mare cu peste un milion de exemple de învățare a fost obținut un scor BLEU mediu de 45.29 de puncte, într-un timp de antrenare destul de scurt (numai două ore pentru cinci epoci) executat pe mai multe GPU-uri. Totuși, chiar și un set de date redus, de numai cincizeci de mii de exemple cu un număr de o sută de epoci obține un scor BLEU mediu de 33.29 de puncte în trei ore. **Cuvinte cheie**: limba română, gramatică, transformatoare, atenție, codificare poziționlă.

II Abstract

The field of natural language processing is not as strongly developed for the Romanian language as it is for others, as is the English language. Writing texts correctly has always been a necessity, and the development of tools that will be useful in this need is critical. The proposed correction system receives a sentence with grammatical errors and corrects it, using state-of-the-art technologies to perform this operation such as attention-based neural models like Encoder-Decoder Transformers. These are a cornerstone in the development of intelligent tools for processing – translating, summarizing, or proofreading – texts and are the foundation for this project. The paper uses RONACC, the first corpus for grammatical corrections in Romanian for modeling, training, testing, and validating the project. Using a very large data set with over a million learning examples, an average BLUE score of 45.29 points was obtained, in a rather short training time (only two hours for five epochs) executed on several GPUs. However, even a small data set of only fifty thousand examples with as many as one hundred epochs achieves an average BLUE score of 33.29 points in three hours. **Keywords**: Romanian language, grammar, transformers, attention, positional encoding.

III Mulțumiri

Le sunt recunoscător lui Traian-Eugen Rebedea și lui Teodor-Mihai Coteț pentru timpul, sfaturile, inspirația și comentariile oferite în realizarea acestui proiect. Îi mulțumesc lui Alexandru Meterez pentru ajutorul, expertiza, explicațiile și răbdarea oferite în studiul Învățării Automate (Machine Learning).

IV Cuprins

I Sinopsis iv
II Abstract iv
III Mulțumiri iv
IV Cuprins v
V Lista de Tabele viii
VI Lista de Figuri viii
Capitolul 1 Introducere 2
1.1 Context 2
1.2 Problema 3
1.3 Obiective 3
1.4 Structura Lucrării 3
Capitolul 2 Analiza și Specificarea Cerințelor 4
2.1 Descrierea Categoriilor de Utilizatori 5
2.2 Cerințe de Sistem 5
2.3 Cerințe Funcționale 5
2.4 Cerințe Nefuncționale 5
2.5 Modele ale Sistemului 5
2.5.1 Actorii și Cazurile de Utilizare 6
2.5.2 Descrierea Cazurilor de Utilizare ale Sistemului 6
Capitolul 3 Abordări Existente 7
3.1 Produse Comerciale 7
3.2 Metode Existente 8
3.3 Tehnologii Utilizate 11
Capitolul 4 Soluția Propusă 12
4.1 Arhitectura 12
4.2 Atenția 12
4.2.1 Atenția Produsului Scalar Scalat 12
4.2.2 Atenție cu mai Multe Capete 13
4.3 Straturi de Codare 15
4.3.1 Rețea de Transmitere Înainte Point Wise 15

4.3.2 Strat de Codificare 15
4.3.3 Strat de Decodificare 16
4.4 Codare 16
4.4.1 Codificare Poziţională 16
4.4.2 Codificator 17
4.4.3 Decodificator 18
4.5 Transformator 18
4.5.1 Hyperparametri 19
4.6 Optimizator 20
4.7 Pierderea 20
4.8 Antrenare 20
4.9 Evaluare 20
4.10 Diagrama UML 21
Capitolul 5 Detalii de Implementare 22
5.1 Configurarea Pipeline-ului de Intrare 22
5.1.1 Setul de Date 24
5.1.2 Mascarea 25
5.2 Atenţia 25
5.2.1 Atenţia Produsului Scalar Scalat 25
5.2.2 Atenţie cu mai Multe Capete 25
5.3 Straturi de Codare 26
5.3.1 Reţea de Transmitere Înainte Point Wise 26
5.3.2 Strat de Codificare 26
5.3.3 Strat de Decodificare 26
5.4 Codare 27
5.4.1 Codificare Poziţională 27
5.4.2 Codificator 28
5.4.3 Decodificator 28
5.5 Transformator 28
5.5.1 Hyperparametri 28
5.6 Optimizator 28
5.7 Pierderea şi Metrici 29

5.8 Antrenare și Puncte de Control 29
5.8.1 Puncte de Control 31
5.8.2 Antrenare 31
5.9 Evaluare 31
5.9.1 Corectare 31
5.9.2 Testare 31
Capitolul 6 Evaluarea Rezultatelor 32
6.1 Testare 32
6.2 Evaluare 33
6.2.1 Evaluare Calitativă 35
Capitolul 7 Concluzii 36
7.1 Dezvoltări Ulterioare 36
Bibliografie 37
Anexa A Ponderile Atenției ..... a
Anexa B Codul Sursă c

V Lista de Tabele

Tabelul 1 Analiză SWOT 4
Tabelul 2 Comparație traduceri (EN-DE, EN-FR) între Transformator și alte modele state-of-the-art 10
Tabelul 3 Comparație analiza constituenței limbii engleze între Transformator și alte modele state-of-the-art 11
Tabelul 4 Exemplu Tokenizare pentru Transformator 23
Tabelul 5 Exemplu Set de Date 24
Tabelul 6 Set de Date Modificat 24
Tabelul 7 Rezultate antrenare 34

VI Lista de Figuri

Figura 1 Atenția Produsului Scalar Scalat 13
Figura 2 Atenție cu mai Multe Capete 14
Figura 3 Atenție cu mai Multe Capete cu Împărțirea Matricelor 14
Figura 4 Rețea de Transmitere Înainte Point Wise 15
Figura 5 Strat de Codificare 15
Figura 6 Strat de Decodificare 16
Figura 7 Codificator 17
Figura 8 Decodificator 18
Figura 9 Transformator 19
Figura 10 Diagramă UML 21
Figura 11 Codificare Poziționlă 27
Figura 12 Variația Ratei de Învățare 29
Figura 13 Acuratețea pentru setul de date mic 33
Figura 14 Pierderea pentru setul de date mic 33
Figura 15 Acuratețea pentru setul de date mediu 34
Figura 16 Pierderea pentru setul de date mediu 34
Figura 17 Exemplu 1: Atenție ..... a
Figura 18 Exemplu 2: Atenție ..... b
Figura 19 Exemplu 3: Atenție ..... b

Nu spun că alte limbi, alte vorbiri nu ar fi minunate și frumoase. Dar atît de proprie, atît de familiară, atît de intimă îmi este limba în care m-am născut, încît nu o pot considera altfel decît iarbă. Noi, de fapt, avem două părți coincidente, odată este patrie de pămînt și de piatră și încă odată este numele patriei de pămînt și de piatră. Numele patriei este tot patrie.

O patrie fără de nume nu este o patrie. Limba română este patria mea. De aceea, pentru mine, muntele munte se zice, de aceea, pentru mine iarba iarbă se spune, de aceea, pentru mine izvorul izvorăște, de aceea, pentru mine viața se trăiește.

– Nichita Stănescu

Capitolul 1 Introducere

Niciodată să nu vorbești incorect; nu numai că insulți gramatica, dar faci să sufere și sufletele.
– Socrate

Gramatica (din greacă antică γραμματική) reprezintă în lingvistică setul de reguli structurale care stabilește într-un limbaj natural raporturile pentru compoziția clauzelor, frazelor și cuvintelor; dar vizează și studiul unor astfel de reguli, incluzând fonologia, morfologia și sintaxa, dar și fonetica, semantica și pragmatica.[1]

Încă din Antichitate, așa cum este descris în dialogurile lui Platon, gramatica era o parte esențială a educației (alături de logică și retorică) ca parte a trivium-ului.[2] În epoca contemporană gramatica nu lipsește din educația tinerilor, încă din primii ani de școală aceștia sunt introduși în regulile gramaticale ale limbii vorbite în țara în care trăiesc, limba română nefăcând excepție de la această cutumă.

Limba română este limba oficială a României (implicit și una dintre limbile oficiale ale Uniunii Europene) și a Republicii Moldova (referită și ca limba moldovenească). Româna face parte din sub-ramura limbilor romanice orientale a limbilor neo-latine – grup lingvistic ce s-a dezvoltat din dialecte ale latinei vulgare, separat de limbile romanice occidentale între secolele V și VIII – ca parte a limbilor romanice balcanice: pe lângă limba română (daco-româna), fiind și aromâna (macedo-româna), istro-româna și megleno-româna. Limba română este împărțită în mai multe sub-dialecte (accente sau graiuri) grupate ca nordice (moldovenesc, ardelenesc, bănățean) și sudice (muntenesc, dician – dobrogean).[3]

1.1 Context

În România rata analfabetismului funcțional este de 39% în rândul elevilor[4], iar greșelile gramaticale se regăsesc frecvent în toate categoriile populației[5]. Pentru limba engleză au apărut diverse programe care ajută utilizatorii să scrie corect, inteligibil și clar, astfel încât mesajul transmis să fie bogat în informație, succint, dar și ușor de parcurs. Limba română nu are un astfel de program computerizat care să poată ajuta un utilizator să redacteze documente corect. De asemenea, societatea se mișcă într-un ritm din ce în ce mai alert, iar oamenii au nevoie de soluții rapide și eficiente pentru problemele lor.

[1] Wikipedia, „Grammar," [Interactiv]. Disponibil: https://en.wikipedia.org/wiki/Grammar. [Accesat: 9 Octombrie 2019].
[2] Wikipedia, „Trivium," [Interactiv]. Disponibil: https://en.wikipedia.org/wiki/Trivium. [Accesat: 9 Octombrie 2019].
[3] Wikipedia, „Romanian language," [Interactiv]. Disponibil: https://en.wikipedia.org/wiki/Romanian_language. [Accesat: 9 Octombrie 2019].
[4] Digi24, „39% dintre elevii români sunt analfabeți funcțional. Cum îi va afecta pe viitor," 24 Ianuarie 2019. [Interactiv]. Disponibil: https://www.digi24.ro/stiri/actualitate/educatie/39-dintre-elevii-romani-sunt-analfabeti-functional-cum-ii-va-afecta-pe-viitor-1070140. [Accesat: 9 Octombrie 2019].
[5] R. Paraschivescu, Intervievat *Cei care vorbesc corect vor forma un soi de trib, de sectă, care se va refugia undeva în păduri*, [Interviu]. 26 Iulie 2018. Disponibil: http://www.contributors.ro/cultura/interviu-radu-paraschivescu-"cei-care-vorbesc-corect-vor-forma-un-soi-de-trib-de-secta-care-se-va-refugia-undeva-in-paduri".

1.2 Problema

Inteligența artificială a ajutat foarte mult în rezolvarea unor probleme sau în crearea de tehnologii care să ușureze munca oamenilor. Aplicația propusă are la bază inteligența artificială pentru a putea corecta și propune sugestii de modificare asupra unui text. În ansamblu, aplicația oferă un câmp de text în care utilizatorul își poate adăuga conținutul (desigur, pe viitor funcționalitățile vor fi extinse, astfel încât utilizatorul să poată lucra direct din aplicațiile sale preferate), iar programul va verifica eventualele greșeli gramaticale, lexicale, de punctuație, de stil sau chiar de ton al limbajului, pentru a asigura crearea unor texte profesioniste, corecte și coerente. Greșelile vor putea fi scoase în evidență printr-un chenar roșu, iar eventualele avertismente (lucruri care nu sunt greșite în sinea lor, dar pot fi îmbunătățite) vor fi evidențiate printr-un chenar galben. Programul va oferi posibilitatea de corectare a greșelilor gramaticale sau de scriere, îmbunătățirea vocabularului sau îmbunătățirea lizibilității.

1.3 Obiective

Îmbunătățirea limbajului românilor, dar și ajutarea cetățenilor a căror limbă maternă nu este româna trebuie să fie o prioritate. Folosirea unui mod modern, precum un program pe calculator (care să poată fi folosit atât pe calculatoare personale, cât și pe telefoane mobile, dar și integrat cu alte aplicații deja existente – editoare și procesoare de text, programe de e-mail, navigatoare web etc.) poate fi folosit atât în scop didactic de către profesori, cât și în scop profesional – în industrie sau academie – pentru perfecționarea limbajului din documente sau eventual chiar și detectarea plagiatului.

1.4 Structura Lucrării

În Învățarea Automată, un model este o reprezentare (matematică) a unui proces cu aplicații în viața reală, obținut prin aplicarea unui algoritm asupra unui set de date. În cadrul lucrării trebuie făcută distincția între modelul de învățare automată și alte modele (cum este de exemplu modelul software).

În Capitolul 1 (Introducere) este prezentată tema proiectului, contextul, problema și obiectivele acestuia. În continuare, Capitolul 2 (Analiza și Specificarea Cerințelor) prezintă motivația realizării proiectului și cerințele de dezvoltare. Cercetarea asupra abordărilor existente este făcută în Capitolul 3 (Abordări Existente) în care a fost făcut un studiu de piață, o analiză a metodelor existente și a state-of-the-art-ului dar și prezentarea tehnologiei utilizate. Arhitectura este descrisă în Capitolul 4 (Soluția Propusă), iar realizarea acesteia este descrisă în Capitolul 5 (Detalii de Implementare); cele două capitole sunt organizate în subcapitole cu același nume, în primul fiind descrise detaliile teoretice, iar în al doilea modalitățile de realizare practice. Capitolul 6 (Evaluarea Rezultatelor) expune detaliile de testare și evaluare a proiectului și rezultatele obținute, iar Capitolul 7 (Concluzii) prezintă stadiul curent al dezvoltării și conclude cu posibilele dezvoltări ulterioare.

Capitolul 2 Analiza și Specificarea Cerințelor

Scrisul corect e pâinea profesorilor de limba română.
– Camil Petrescu

Pentru o mai bună analiză și planificare a strategiei de dezvoltare, a fost utilizată analiza SWOT din Tabelul 1 Analiză SWOT pentru a determina punctele forte, punctele slabe, oportunitățile și amenințările legate de dezvoltarea proiectului.

Analiza SWOT	*Benefice* *atingerii obiectivelor*	*Pun în pericol* *atingerea obiectivelor*
Sursă internă *(mediul intern)*	<u>Puncte tari</u> - poate fi folosit în scop autodidact; - îmbunătățește vorbirea în toate aspectele ei, deci se poate adresa unei categorii largi de persoane.	<u>Puncte slabe</u> - procesul de creare a seturilor de date pentru program este foarte laborios; - timpul de antrenare al modelului este direct proporțional cu dimensiunea setului de date pentru antrenare.
Sursă externă *(mediul extern)*	<u>Oportunități</u> - lipsa unor astfel de programe în limba română; - nevoia îmbunătățirii vorbirii.	<u>Amenințări</u> - reticența utilizatorilor în a folosi unelte noi (preferarea metodelor clasice: întrebarea altor oameni sau căutarea pe internet); - lipsa de timp a oamenilor; - oameni care se ocupă cu tehnoredactare și prelucrare de texte reprezintă o posibilă concurență.

Tabelul 1 Analiză SWOT

În urma analizei au fost determinate cerințele pe care trebuie să le îndeplinească proiectul. Acesta trebuie să fie simplu și ușor de folosit, utilizatorul neavând nevoie de o pregătire tehnică pentru a înțelege cum funcționează și cum poate fi utilizată aplicația. Pentru acesta este de ajuns să poată introduce un text, cu eventuale erori, iar rezultatul primit să fie un alt text în care erorile să fie corectate.

2.1 Descrierea Categoriilor de Utilizatori

Utilizatorii aplicației sunt persoanele care vor sa își asigure corectitudinea gramaticală a textelor scrise de aceștia și nu numai. Spectrul larg cuprinde întreaga populație vorbitoare de limba română. Iar publicul țintă specific este cel al oamenilor care doresc să își îmbunătățească abilitățile de scriere, cum ar fi:

- Elevi și studenți care doresc să își îmbunătățească lucrările în vederea pregătirii academice;
- Profesori și studenți care doresc să scrie lucrări științifice pentru publicații de specialitate;
- Autori, jurnaliști sau ziariști care lucrează în redacție;
- Profesori și tehnoredactori de care nu va mai fi nevoie pentru corectarea textelor atunci când aplicația va fi pusă în funcțiune;
- Oameni care doresc să își îmbunătățească abilitățile de scriere;
- Oameni care în cadrul slujbei trebuie să alcătuiască texte sau rapoarte (de exemplu: avocați, polițiști, juriști, economiști).

2.2 Cerințe de Sistem

Pentru executarea codului sursă (ce corespunde încărcării setului de date și antrenării modelului) este nevoie de un computer ce are instalată o versiune de Python[6] mai nouă de 3.5 și o listă de frameworkuri, biblioteci și pachete: tensorflow[7], numpy[8] și matplotlib[9].

2.3 Cerințe Funcționale

Aplicația trebuie să le permită utilizatorilor să introducă texte (de la tastatură sau din fișiere) pentru a fi corectate. Aplicația trebuie să le permită utilizatorilor să poată salva textele respective.

2.4 Cerințe Nefuncționale

Aplicație trebuie să poată fi executată și fără a antrena modelul, dacă acesta a fost deja antrenat. Aplicația trebuie să permită antrenarea modelului folosind noi date.

2.5 Modele ale Sistemului

Modelul sistemului are la bază schimbul de informații dintre aplicație și utilizatori: utilizatorii antrenează modelul și oferă sistemului o intrare (o propoziție ce trebuie corectată), iar aplicația oferă ieșirea sistemului (propoziția corectată) folosind modelul antrenat.

[6] Python. Disponibil: https://www.python.org. [Accesat: 3 Iunie 2020].
[7] TensorFlow. Disponibil: https://www.tensorflow.org. [Accesat: 3 Iunie 2020].
[8] NumPy. Disponibil: https://numpy.org. [Accesat: 3 Iunie 2020].
[9] Matplotlib. Disponibil: https://matplotlib.org. [Accesat: 3 Iunie 2020].

2.5.1 Actorii și Cazurile de Utilizare

Actorii: utilizatorii aplicației (cei ce doresc corectarea textelor).

Cazurile de Utilizare:

- Antrenarea modelului;
- Corectarea unei propoziții.

2.5.2 Descrierea Cazurilor de Utilizare ale Sistemului

Cazurile de utilizare prezentate sunt descrise în cele ce urmează:

1. Antrenarea modelului:
    a. Precondiție: Aplicația este pornită.
    b. Flux de bază:
        i. Se încarcă seturile de date (pentru a fi prelucrate);
        ii. Se inițializează modelul;
        iii. Se antrenează modelul.
    c. Alternative:
        i. Seturile de date nu există sau sunt invalide:
            ❖ Eroare de execuție.
        ii. Parametrii modelului sunt incorecți:
            ❖ Eroare de execuție.
        iii. Modelul nu a fost inițializat:
            ❖ Eroare de execuție.
    d. Postcondiție:
        i. Modelul este antrenat iar utilizatorul poate corecta propoziții.
2. Corectarea unei propoziții:
    a. Precondiție: Modelul este antrenat.
    b. Flux de bază:
        i. Utilizatorul introduce o propoziție pentru a fi corectată;
        ii. Utilizatorul poate introduce și varianta corectă a propoziției pentru afișare și comparare.
    c. Alternative:
        i. Utilizatorul nu a introdus o propoziție validă:
            ❖ Este ignorată corectarea propoziției.
        ii. Utilizatorul nu a introdus propoziția corectă ca referință:
            ❖ Este ignorată afișarea acesteia.
    d. Postcondiție:
        i. Sistemul afișează propoziția corectată.

Capitolul 3 Abordări Existente

Destinul unui popor depinde de starea gramaticii sale. Nu există mare națiune fără proprietatea limbii.
– Fernando Pessoa

Problematica corectării textelor și a limbajului natural în general este des întâlnită în știința calculatoarelor, inteligența artificială și învățarea automată. Studiul prelucrării limbajului natural a adus tehnologii, modele și produse care să poată aborda într-un mod competitiv această problemă. Soluțiile sunt de obicei aproximative, dar destul de relevante. Majoritatea problemelor de manipulare a textelor și a secvențelor (traducere, corectare, rezumare) se pot modela în același fel și tehnologiile pot fi adaptate de la caz la caz.

3.1 Produse Comerciale

Dacă pentru limba engleză există o multitudine de produse, cum se va putea urmări în continuare, pentru limba română proiectele de acest fel sunt în număr redus și de obicei nu sunt la fel de competitive ca cele pentru limba engleză (sau pentru ale limbi străine unde a existat o investiție considerabilă în cercetare). De altfel, particularitățile limbii române, comparativ cu limba engleză, au făcut ca dezvoltarea de programe inteligente pentru corectarea textelor să nu fie la fel de puternică.

Microsoft Office Spell Checker[10] este unul dintre cele mai folosite corectoare pentru texte. Popularitatea acestuia constă în faptul că toate aplicațiile din suita Office folosesc această caracteristică ce este disponibilă în mai multe limbi (limba engleză și chiar și limba română). Din experiență, produsul e bun pentru corectarea micilor greșeli de scriere, cel puțin în limba română nedepistând toate greșelile sau problemele. Un alt dezavantaj este că nu oferă soluții de înțelegere și îmbunătățire a textelor.

Grammarly[11] este o companie americană internațională care dezvoltă un instrument de scriere foarte popular pentru limba engleză cu ajutorul inteligenței artificiale și a procesării limbajului natural. Produsul oferit oferă servicii de verificare gramaticală, verificare ortografică, adaugă sugestii pentru claritatea și concizia textului, îmbunătățirea vocabularului, a stilului și a tonului, toate acestea prin folosirea algoritmilor de învățare automată și învățare adâncă. Dezavantajul principal este faptul că soluția oferită este doar pentru limba engleză.

Qordoba[12] este o altă companie americană de inteligență artificială ale cărei tehnologii se axează pe partea de software ca serviciu. Principala platformă dezvoltată este folosită în informații de conținut, ce include un asistent inteligent alături de alte instrumente pentru calitatea conținutului. Asistentul este util pentru a uniformiza stilul, terminologia și tonul documentelor redactate într-o companie, astfel o organizație poate obține consecvență pe

[10] Microsoft Office. Disponibil: https://www.office.com. [Accesat: 20 Iunie 2020].
[11] Grammarly. Disponibil: https://www.grammarly.com. [Accesat: 20 Iunie 2020].
[12] Qordoba. Disponibil: https://qordoba.com. [Accesat: 20 Iunie 2020].

toate planurile de conținut: marketing, produse, politici juridice și de resurse umane, documentație tehnică, comunicare, baze de cunoștințe pentru clienți și multe altele. Dezavantajul soluției provine din faptul că este oferită pe partea de business și nu pentru utilizatorii obișnuiți.

Google Translate[13] este un serviciu gratuit de traduceri automate multi-lingvistic bazat pe statistică și rețele neurale folosit pentru traducerea de texte și pagini web. Serviciul este capabil să corecteze texte primite cu mici greșeli atunci când nu reușește să înțeleagă contextul sau anumite cuvinte. Totuși, la scară largă este specializat în traduceri și nu în corecții gramaticale.

DEX Online[14] este o platformă web organizată ca o colecție de dicționare electronice. Platforma oferă definiții pentru cuvintele din limba română, alături de declinări, conjugări, sinteză, etimologie și chiar exemple de utilizare în diferite contexte. Problematica acestei platforme este că nu oferă niciun suport pentru corectarea automată a textelor, totuși de departe fiind unul dintre puținele proiecte care au avut în vedere dezvoltarea unui produs gratuit (sau a unui produs în general) pentru limba română.

Piața de produse și servicii software pentru scrierea și modelarea textelor este performantă și competitivă în spațiul limbii engleze (datorită investițiilor, dar probabil și din cauza faptului că limba engleză este o limbă destul de simplă în reguli, motiv pentru care este și foarte populară). Totuși, pentru limba română nu există produse similare celor pentru limba engleză care să permită corectarea și îmbunătățirea textelor. Chiar dacă limba română este mult mai complexă în regulile sale gramaticale, nu înseamnă că nu ar trebui investit și cercetat în acest domeniu. Pentru corectarea cuvintelor cu mici greșeli există programe și module care permit verificarea acestora, dar doar la nivel de cuvânt, fără a verifica gramatica sau contextul.

3.2 Metode Existente

Tehnologiile de ultimă generație din domeniul traducerilor automate pot fi folosite și în alte domenii adiacente prelucrării limbajului natural, cum sunt generarea de rezumate pentru lucrări sau corectarea textelor. În continuare sunt prezentate tehnologiile dezvoltate de-a lungul timpului în domeniul prelucrării limbajului natural, atât pentru traduceri dar și pentru corectări. Majoritatea se bazează pe tehnici de învățare automată (în special rețele neuronale și rețele neuronale adânci) și inteligență artificială: învățare secvență la secvență, învățare cu rețele neuronale recurente, codificatoare și decodificatoare, învățare cu rețele neuronale convoluționale și învățarea folosind atenția, probabil cea mai recentă și puternică evoluție în domeniu. Există și alte metode alternative care nu se folosesc de rețele neuronale, pentru care efortul de colectare a datelor statistice și modelare este foarte mare (acest lucru nu înseamnă că nu pot să fie cel puțin la fel de performante ca alte metode).

[13] Google Traducere. Disponibil: https://translate.google.ro. [Accesat: 20 Iunie 2020].
[14] Dex Online. Disponibil: https://dexonline.ro. [Accesat: 20 Iunie 2020].

Rețelele Neuronale Adânci sunt modele puternice care au obținut performanțe excelente în sarcinile dificile de învățare. Deși rețelele neuronale adânci funcționează bine ori de câte ori sunt disponibile seturi etichetate de antrenare mari, acestea nu pot fi utilizate pentru a mapa secvențe la secvențe. Se poate utiliza o abordare generală de la un capăt la altul de învățare a secvențelor care face presupuneri minime legate de structura secvenței. Metoda folosește o memorie de lungă durată pe termen scurt multistrat pentru a mapa secvența de intrare la un vector de dimensiune fixă, apoi o altă memorie adâncă pentru a decoda secvența țintă din vector. De asemenea, memoria poate învăța reprezentări ale frazelor și propozițiilor care sunt dependente de ordinea cuvintelor și sunt relativ invariabile la vocea activă și pasivă [1].

Un alt model de rețea neuronală propus, numit Rețea Neuronală Adâncă Codificator-Decodificator, constă în două rețele neuronale recurente. O rețea codifică o secvență de simboluri într-o reprezentare vectorială de lungime fixă, iar cealaltă decodifică reprezentarea într-o altă secvență de simboluri. Codificatorul și decodificatorul modelului propus sunt antrenate în comun pentru a maximiza probabilitatea condițională a unei secvențe țintă date cu o secvență sursă. Performanța unui sistem de traducere automată statistică se dovedește a fi îmbunătățită empiric folosind probabilitățile condiționale ale perechilor de fraze calculate de Rețeaua Neuronală Adâncă Codificator-Decodificator. Acest model învață o reprezentare semantică și sintactică semnificativă a frazelor lingvistice [2].

Traducerea automată neurală este o abordare recent propusă pentru traducerea automată. Spre deosebire de tradiționala traducere automată statistică, traducerea automată neurală are ca scop construirea unei rețele neuronale unice care poate fi reglată în comun pentru a maximiza performanța de traducere. Modelele propuse recent pentru traducerea automată neurală aparțin adesea unei familii de codificatoare-decodificatoare și constau dintr-un codificator care codifică o propoziție sursă într-un vector de lungime fixă și un decodificator care generează o traducere. Utilizarea unui vector cu lungime fixă poate fi un blocaj în îmbunătățirea performanței acestei arhitecturi de bază a decodificatorului și extinderea acesteia ar permite unui model să caute automat părți ale unui propoziție sursă care sunt relevante pentru a prezice un cuvânt țintă, fără a fi necesar să formeze aceste părți ca un segment în mod explicit. Această nouă abordare obține o performanță de traducere comparabilă cu sistemele existente de traducere de ultimă oră bazate pe fraze [3].

O altă arhitectură este cea bazată în întregime pe rețele neuronale convoluționale, în comparație cu modelele recurente, în care calculele pentru toate elementele pot fi complet paralelizate în timpul antrenării, iar optimizarea este mai ușoară, deoarece numărul de neliniarități este fixat și independent de lungimea de intrare. Utilizarea de unități liniare închise ușurează propagarea gradienților și fiecare strat de decodare este echipat cu un modul de atenție separat [4].

Cele mai performante modele conectează codificatorul și decodificatorul printr-un mecanism de atenție. Transformatorul este o arhitectură de rețea simplă, bazată exclusiv pe mecanisme de atenție, care se descotorosește complet de recurențe și de convoluții. Experimentele

efectuate pe două sarcini de traducere automată arată că aceste modele sunt de calitate superioară, fiind paralelizabile și necesitând un timp semnificativ mai mic de antrenare. Transformatorul generalizează foarte bine la alte sarcini, putând fi aplicat cu succes pentru analiza textelor și producerea de texte noi [5].

În Tabelul 2 Comparație traduceri (EN-DE, EN-FR) între Transformator și alte modele state-of-the-art [6] se pot vedea rezultatele obținute de Transformator în traduceri din limba engleză (EN) în limba germană (DE) sau în limba franceză (FR).

Model	**BLEU**		**Cost Antrenare (FLOPs)**	
	EN-DE	**EN-FR**	**EN-DE**	**EN-FR**
ByteNet	23.75			
Deep-Alt + PosUnk		39.2		$1.0 \cdot 10^{20}$
GNMT + RL	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnkEnsemble		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformator (de bază)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformator (mare)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Tabelul 2 Comparație traduceri (EN-DE, EN-FR) între Transformator și alte modele state-of-the-art

În Tabelul 3 Comparație analiza constituenței limbii engleze între Transformator și alte modele state-of-the-art [6] se pot observa rezultatele obținute de Transformator și modul în care generalizează și pentru alte sarcini, nu doar pentru traduceri.

Interpretor	**Antrenare**	**WSJ 23 F1**
Vinyals & Kaiser et al. (2014)	doar WSJ, caracteristic	88.3
Petrov et. al. (2006)	doar WSJ, caracteristic	90.4
Zhu et. al. (2013)	doar WSJ, caracteristic	90.4
Dyer et. al. (2016)	doar WSJ, caracteristic	91.7
Transformator (4 straturi)	**doar WSJ, caracteristic**	**91.3**

Zhu et. al (2013)	semi-supervizat	91.3
Huang & Harper (2009)	semi-supervizat	91.3
McClosky et. al (2009)	semi-supervizat	92.1
Vinyals & Kaiser et. al. (2014)	semi-supervizat	92.1
Transformator (4 straturi)	**semi-supervizat**	**92.7**
Luong et. al (2015)	multi-task	93.0
Dyer et. al. (2016)	generativ	93.3

Tabelul 3 Comparație analiza constituenței limbii engleze între Transformator și alte modele state-of-the-art

Un tip alternativ de corector ortografic folosește numai informații statistice, cum ar fi n-grame, pentru a recunoaște erorile în loc de cuvintele ortografiate corect. Această abordare necesită de obicei mult efort pentru a obține informații statistice suficiente. Printre avantajele cheie se numără necesitatea unui spațiu de stocare mai mic și capacitatea de a corecta erorile în cuvinte care nu sunt incluse într-un dicționar.[15]

3.3 Tehnologii Utilizate

Lucrarea de față folosește Python și TensorFlow pentru a implementa o soluție bazată pe Transformatoare propusă în articolul realizat de Vaswani et. al. „Attention Is All You Need" [6].

Rețelele neuronale recurente și memoria de lungă durată pe termen scurt au fost stabilite ca abordări de ultimă generație în modelarea secvențelor și problemelor de raționament, cum ar fi modelarea limbajului și traducerea automată [2] [3]. De atunci, numeroase eforturi au continuat să împingă limitele modelelor de limbaj recurente și arhitecturilor de codificatoare-decodificatoare [7].

Mecanismele de atenție au devenit o parte integrată a modelelor de secvență convingătoare și a modelelor de raționament în diferite sarcini, ceea ce permite modelarea dependențelor fără a ține cont de distanța lor în secvențele de intrare sau de ieșire [3]. În majoritatea cazurilor astfel de mecanisme de atenție sunt utilizate împreună cu o rețea recurentă. Transformatoarele sunt introduse ca o arhitectură de model care evită recurențele și se bazează pe un mecanism de atenție pentru a atrage dependențe globale între intrare și ieșire. Transformatorul permite o paralelizare semnificativ mai mare și poate atinge un nou stadiu al tehnologiei în ceea ce privește calitatea traducerii sau corectării textelor după ce a fost antrenat [5].

[15] M. Kantrowitz și S. Baluja, „Method for rule-based correction of spelling and grammar errors". SUA Brevet US6618697B1, 9 Septembrie 2003.

Capitolul 4 Soluția Propusă

Ce poate fi mai trist decât atât? Am ajuns să ne mirăm de cei capabili să respecte limba română.
– Andrei Ş.L. Evelin

Transformatoarele sunt modele incredibile ce au la bază o structură de tipul codificator-decodificator [1] [2] [3] [4] luând cu asalt lumea limbajului natural. Au împins barierele tehnologiilor de ultimă oră și au doborât recorduri în lumea NLP. Și-au găsit locul în diferite aplicații, de la traduceri și agenți conversaționali până la îmbunătățirea motoarelor de căutare și sunt ultimul răcnet în învățarea adâncă.

În continuare sunt prezentate aspectele teoretice ale acestora și cum sunt folosite în prelucrarea limbajului natural. Soluția propusă folosește transformatoarele pentru a prelucra limbajul natural și pentru a corecta textele scrise în limba română.

4.1 Arhitectura

Modelele de raționament neuronale competitive au o structură de tipul codificator-decodificator [1] [2] [3] [4]. Codificatorul transformă intrarea simbolică $x = (x_1, \dots, x_n)$ într-o secvență a reprezentărilor continue $z = (z_1, \dots, z_l)$. Având z, decodificatorul generează ieșirea simbolică $y = (y_1, \dots, y_m)$, câte un element la un moment dat. Modelul este auto-regresiv [8], folosind elementele generate anterior ca intrare adițională pentru generarea următorului element. Transformatorul are la bază această arhitectură folosind atenția și straturi complet conectate point wise pentru codificator și decodificator [6].

4.2 Atenția

Atenția este un mecanism prin care transformatoarele pot avea o memorie teoretică infinită și astfel se pot concentra și asupra unor cuvinte generate cu mult anterior [6].

4.2.1 Atenția Produsului Scalar Scalat

Pentru a obține atenția, intrarea se distinge în trei straturi pentru a crea vectorii de interogare, cheie și valoare [9].

> *The key/value/query concepts come from retrieval systems. For example, when you type a query to search for some video on YouTube, the search engine will map your **query** against a set of **keys** (video title, description etc.) associated with candidate videos in the database, then present you the best matched videos (**values**).*[16]

Calculul atenției este realizat după modelul din Figura 1 Atenția Produsului Scalar Scalat [5], unde interogările de dimensiune d_k sunt grupate într-o matrice Q cu l_k linii, cheile de

[16] dontloo, „What exactly are keys, queries, and values in attention mechanisms?,” [Răspuns] Stats StackExchange, 29 August 2019. [Interactiv]. Disponibil: https://stats.stackexchange.com/a/424127. [Accesat 19 Iunie 2020].

dimensiune d_k sunt grupate într-o matrice K cu l_k linii și valorile de dimensiune d_v sunt grupate într-o matrice V cu l_k [6].

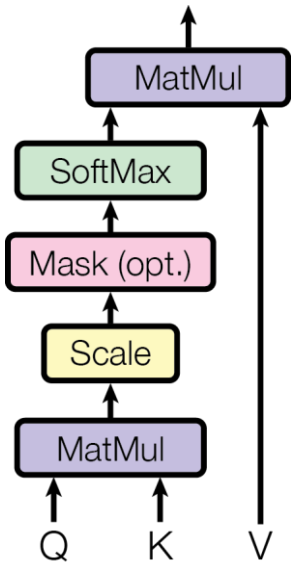

Figura 1 Atenția Produsului Scalar Scalat

Calculele sunt realizate simultan sub formă matriceală, chiar dacă acestea au fost definite inițial pentru vectori, astfel:

1. Produsul scalar dintre interogări și chei: produce o matrice de scor ce stabilește cât de relevante sunt celelalte cuvinte, relativ la un cuvânt (cu cât este scorul mai mare, cu atât anumite cuvinte sunt mai relevante), iar în modul acesta interogările și cheile sunt mapate;
2. Scalarea scorului: permite gradienți mai stabili;
3. Mascare: adunarea unei matrice de aceeași dimensiune unde pe pozițiile ce trebuie eliminate se află $-\infty$ iar pe restul pozițiilor 0; rezultatul mascării se observă la pasul următor, unde la aplicarea funcției softmax valorile infinite devin zerouri;
4. Softmax: se obțin ponderile atenției ce oferă probabilități proporționale cu scorurile; modelul va fi mai încrezător asupra cuvintelor ce trebuie să participe la predicție;
5. Înmulțirea cu valorile: ponderile sunt înmulțite cu valorile, iar scorurile mai mari vor păstra cuvintele pe care modelul le învață ca fiind mai importante [9].

4.2.2 Atenție cu mai Multe Capete

Atenția cu mai multe capete presupune proiectarea liniară a interogărilor, cheilor și valorilor cu diferite proiecții liniare învățate. Fiecare proiecție liniară va fi intrare pentru calculul atenției, iar calculele se pot efectua în paralel, rezultatele fiind concatenate la final, ca în Figura 2 Atenție cu mai Multe Capete [5] [6]. Relațiile de calcul sunt date de:

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

$$unde\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Parametrii de proiecție sunt $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ și $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$ [6].

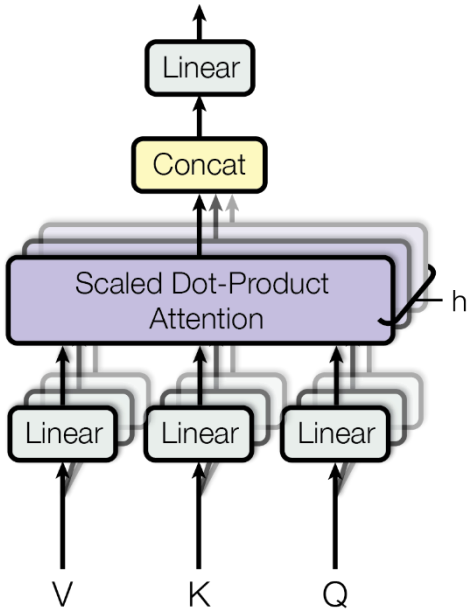

Figura 2 Atenție cu mai Multe Capete

O alternativă este ca după învățare matricele să fie împărțite în h sub-matrice ca în Figura 3 Atenție cu mai Multe Capete cu Împărțirea Matricelor [5], procesul urmând să se desfășoare la fel.

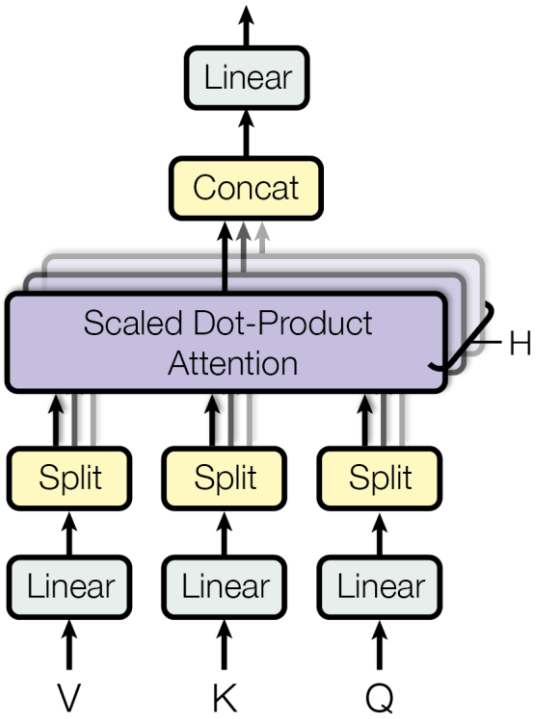

Figura 3 Atenție cu mai Multe Capete cu Împărțirea Matricelor

4.3 Straturi de Codare

Fiecare sub-strat din straturile de codare are o conexiune reziduală [10] (ieșirea este adunată cu intrarea) care este apoi normalizată [11]. Conexiunile reziduale ajută modelul la antrenare, gradienții putând trece direct prin rețea, în timp ce normalizarea este utilizată pentru a stabiliza rețeaua, ceea ce implică o reducere semnificativă a timpului de antrenare [9].

4.3.1 Rețea de Transmitere Înainte Point Wise

Rețeaua de transmitere înainte point wise este folosită pentru a proiecta atenția ieșirii, ceea ce îi poate da o reprezentare mai bogată [9]. Structura rețelei este dată în Figura 4 Rețea de Transmitere Înainte Point Wise [5].

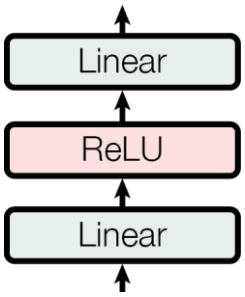

Figura 4 Rețea de Transmitere Înainte Point Wise

Ieșirea rețelei este dată de $FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$ și poare fi privită ca rezultatul a două convoluții cu dimensiunea nucleului de 1 [6].

4.3.2 Strat de Codificare

Stratul de codificare este compus dintr-un sub-strat de calcul a atenției cu mai multe capete și o rețea de transmitere înainte point wise, așa cum este prezentat în Figura 5 Strat de Codificare [5]. În sub-stratul de atenție toate interogările, cheile și valorile provin din același loc (stratul anterior din codificator) [6].

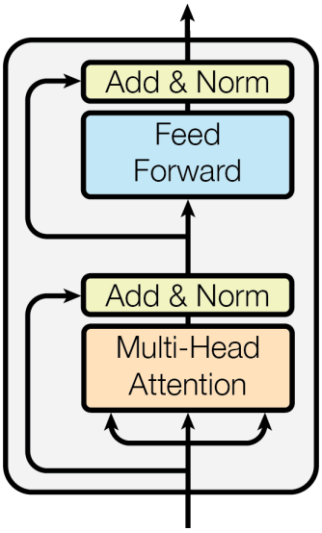

Figura 5 Strat de Codificare

4.3.3 Strat de Decodificare

Stratul de decodificare este compus din două sub-straturi de calcul a atenției cu mai multe capete și o rețea de transmitere înainte point wise, așa cum este prezentat în Figura 6 Strat de Decodificare [5]. Straturile de atenție permit fiecărei poziții să participe la toate pozițiile din decoder până la poziția aceasta. Pentru a păstra proprietatea de auto-regresie [8], pozițiile dincolo de cea curentă trebuie mascate (setate la $-\infty$) [6].

În stratul de atenție care leagă codificatorul de decodificator (al doilea strat – cel din mijloc) interogările provin din stratul decodificator anterior, în timp ce cheile și valorile provin de le ieșirea codificatorului. Acest lucru permite fiecărei poziții din decodificator să participe la fiecare poziție din intrare, asemănător cu alte mecanisme de atenție în modele secvență-la-secvență, cum sunt [3] [4] [6] [7].

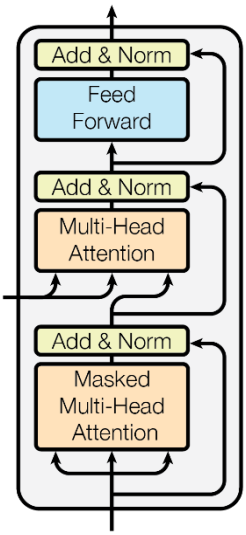

Figura 6 Strat de Decodificare

4.4 Codare

Pentru convertirea tokenilor de intrare și de ieșire în vectori de dimensiunea d_{model} se folosește încorporarea învățată. După decodificare, ieșirea este convertită în probabilități folosind funcția softmax și o transformare liniară învățată. În model, matricea de ponderi este împărțită între cele două straturi de încorporare și transformarea liniară pre-softmax. În straturile de încorporare, greutățile sunt înmulțite cu $\sqrt{d_{model}}$.

4.4.1 Codificare Poziționlă

Transformatorul nu folosește recurențe și are nevoie de informații despre pozițiile din intrarea încorporată [9]. Pentru acest lucru se folosește codificarea poziționlă ce are aceeași dimensiune d_{model} pentru a putea fi adunată cu încorporarea:

$$PE(pos, 2i + j) = \begin{cases} \sin \dfrac{pos}{1000^{\frac{2i}{d_{model}}}}, & j = 0 \\ \cos \dfrac{pos}{1000^{\frac{2i}{d_{model}}}}, & j = 1 \end{cases}$$

unde pos este poziția și i este dimensiunea. Fiecare dimensiune a codificării poziționale corespunde unei sinusoide, lungimile de undă formând o progresie geometrică $2k\pi$ [6].

Aceste funții, ipotetic, permit modelului să învețe cu ușurință valorile, deoarece pentru orice $k, PE(pos + k, \cdot)$ poate fi reprezentat ca o funcție liniară a $PE(pos, \cdot)$. Funcțiile sinusoidale permit modelului să extrapoleze pentru lungimi mai mari ale secvenței decât cele prezente la antrenare [6]. Există și alte posibilități de codificare poziționlă, cum sunt cele învățate [4].

4.4.2 Codificator

Arhitectura codificatorul este prezentată în Figura 7 Codificator [5] și este formată din:

- Încorporare de intrare;
- Codificare poziționlă;
- Straturi de codificare [6] [9].

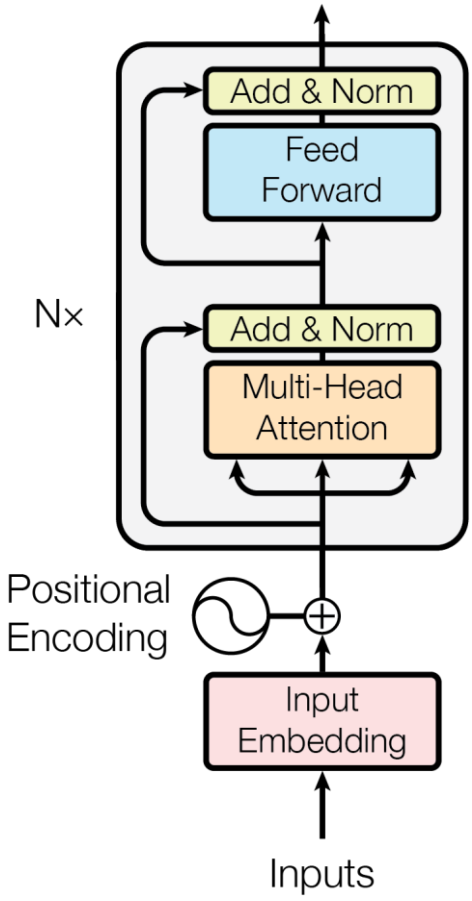

Figura 7 Codificator

4.4.3 Decodificator

Arhitectura decodificatorul este prezentată în Figura 8 Decodificator [5] și este formată din:

- Încorporare de ieșire;
- Codificare pozițională;
- Straturi de decodificare [6] [9].

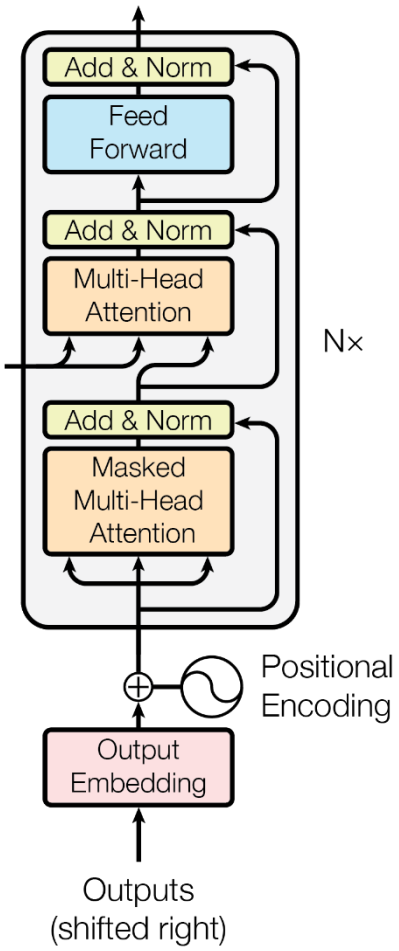

Figura 8 Decodificator

4.5 Transformator

Transformatorul este compus din codificator, decodificator și un strat final liniar de dimensiunea vocabularului țintă, ca în Figura 9 Transformator [5].

Codificatorul mapează toate secvențele de intrare în reprezentări continue abstracte, care conțin informațiile învățate despre întreaga secvență. Decodorul este auto-regresiv, începe cu un token de start și primește ieșirile anterioare ca intrări alături de ieșirile codificatorului care conțin informații despre atenția intrării. Ieșirea decodificatorului trece printr-un strat final liniar cu rol de clasificator (numărul de clase este egal cu dimensiunea vocabularului), iar cuvântul prezis este dat de:

$$predicted_{id} = \underset{id \in \{0 \dots vocab_{size}\}}{\operatorname{argmax}} softmax(id)$$

Ieșirea obținută este adăugată la intrările decodificatorului și se continuă procedeul până când se ajunge la tokenul de final [9].

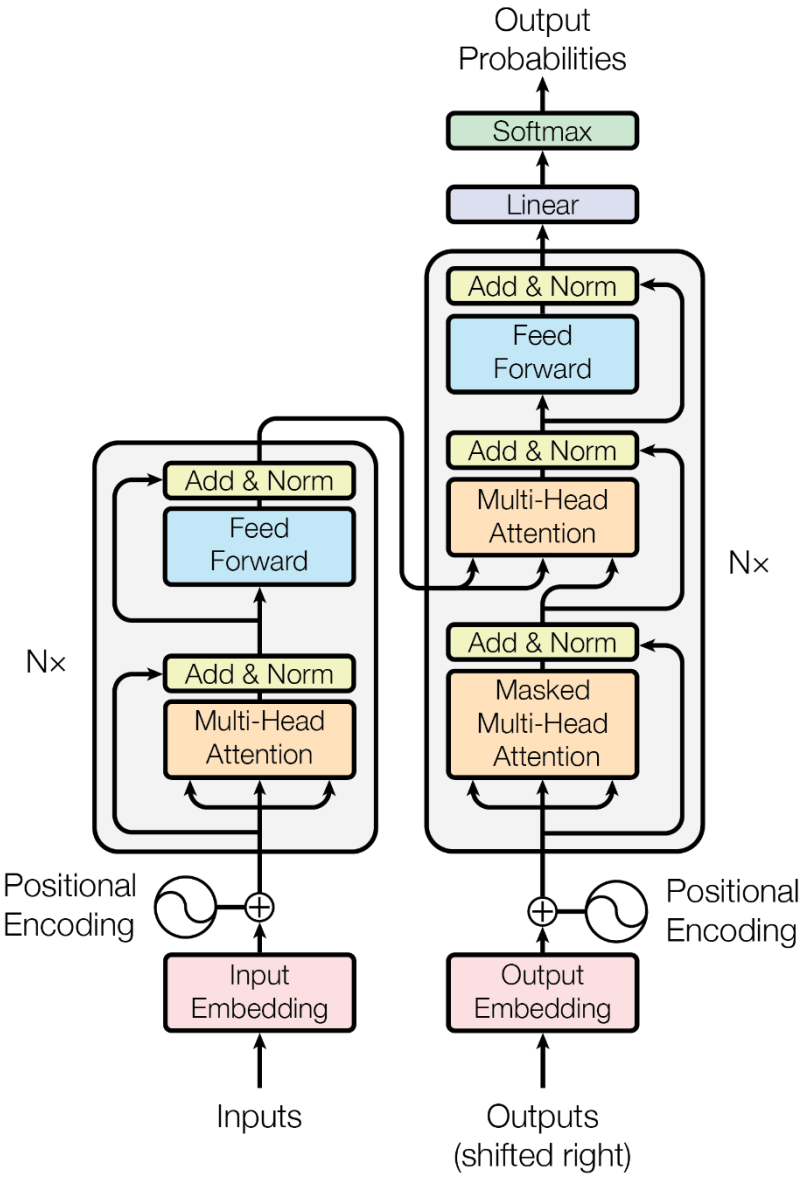

Figura 9 Transformator

4.5.1 Hyperparametri

Modelul folosește următorii hyperparametri:

- n: numărul de straturi de codare;
- d_{model}: dimensiunea vectorilor de tokeni pentru intrare/ieșire;
- dff: dimensiunea straturilor interne pentru rețeaua de transmitere înainte;
- h: numărul de capete pentru calculul atenției cu mai multe capete.

4.6 Optimizator

Optimizatorul Adam [12] cu parametrii $\beta_1 = 0.9$, $\beta_2 = 0.98$ și $\epsilon = 10^{-9}$ este folosit alături de o rată a învățării variabilă de-a lungul antrenării:

$$l_{rate} = d_{model}^{-0.5} \cdot \min\left(step_{num}^{-0.5}, step_{num} \cdot warmup_{steps}^{-1.5}\right)$$

Rata învățării este reprezentată ca o creșterea liniară corespunzătoare primilor pași de antrenare până la $warmup_{steps} = 4000$ și apoi o scădere proporțională cu rădăcina pătrată inversă a pasului [6].

4.7 Pierderea

Pierderea este metrica optimizată la antrenare pentru a obține un model competitiv. Pierderea este calculată între două distribuții de probabilitate (cea prezisă și cea reală). Pentru a compara cele două distribuții facem diferența lor, calculăm Entropia Încrucișată sau divergența Kullback-Leibler [13]. În implementare a fost folosită Entropia Încrucișată.

4.8 Antrenare

Pentru antrenare se pot folosi două metode de regularizare: [6]

1. Renunțare Reziduală: $P_{drop_{out}} = 0.1$ aplicarea renunțării la ieșirea fiecărui sub-strat, înainte de a fi trecut ca input pentru un alt sub-strat sau normalizat; aplicarea renunțării la suma dintre încorporări și codificări poziționale atât la codificator cât și la decodificator [14].
2. Netezirea Etichetelor[17]: $\varepsilon_{ls} = 0.1$ modelul învață să fie mai nesigur, dar îmbunătățește acuratețea și scorul BLEU [15].

4.9 Evaluare

Alte metrici folosite în antrenare și evaluare, pe lângă pierdere (Entropia Încrucișată) [13], sunt:

- acuratețea;
- scorul BLEU.

BLEU (bilingual evaluation understudy – evaluarea aprecierii bilingve) este o metodă de evaluare a calității textelor traduse automat dintr-o limbă naturală în alta. Calitatea textelor este corespondența dintre rezultatul tradus automat și rezultatul traducerii unui om. Scorurile sunt calculate asupra textelor segmentate individual (de obicei în propoziții) prin compararea lor cu un set de o foarte bună calitate de traduceri. Scorul BLEU este mereu între 0 si 1, interpretat ca asemănarea dintre textul candidat și textele de referință, unde valorile mari (apropriate de 1) reprezintă texte asemănătoare [15].

[17] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens și Z. Wojna, „Rethinking the Inception Architecture for Computer Vision,” 2 Decembrie 2015. [Interactiv]. Disponibil: https://arxiv.org/abs/1512.00567. [Accesat 19 Iunie 2020].

4.10 Diagrama UML

În Figura 10 Diagramă UML este prezentată diagrama de clase pentru implementarea Transformatorului, Codificatorului, Decodificatorului, Substraturilor și a ratei de învățare.

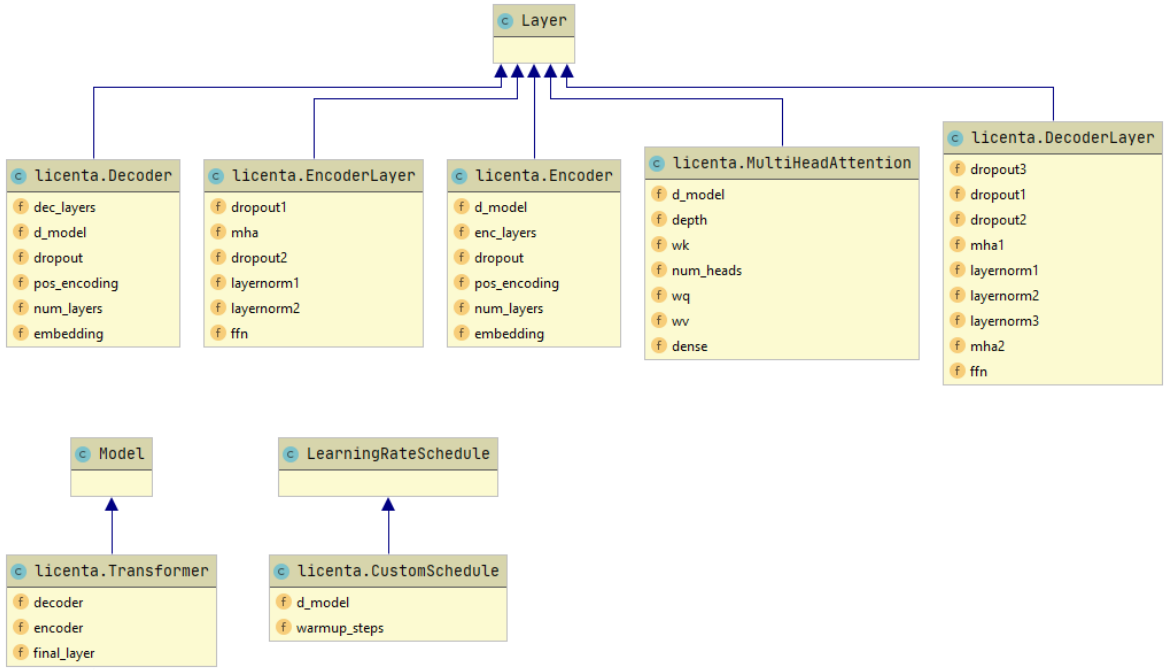

Figura 10 Diagramă UML

În diagrama de clase este prezentată descrierea sistemului și a claselor ce definesc componentele acestuia (atributele acestora și relațiile dintre ele). Diagrama ajută la realizarea și înțelegerea structurii modelului implementat.

Capitolul 5 Detalii de Implementare

If you want to make an apple pie from scratch, you must first create the universe.
– Carl Sagan, Cosmos

Implementarea soluției a fost făcută într-un Jupyter Notebook[18], folosind Google Colab[19] pentru dezvoltare. Codul a fost scris în Python3 folosind frameworkul TensorFlow [16] pentru dezvoltarea și antrenarea modelului și TensorFlow Datasets [17] pentru manipularea seturilor de date. Pentru calcule și operații matematice a fost folosită și biblioteca NumPy [18], pe lângă operațiile puse la dispoziție de TensorFlow, iar pentru reprezentările grafice a fost folosită biblioteca Matplotlib.

5.1 Configurarea Pipeline-ului de Intrare

Pentru rezolvarea problemei de corectare a propozițiilor este necesară configurarea modulului în care datele sunt citite, dar și a modulului în care datele sunt manipulate și interacționează între ele.

Setul de date este alcătuit din mai multe fișiere TSV Tab-Separated Values (nu CSV Comma-Separated Values), fiecare linie din fișier conținând câte o pereche de forma *(propoziție corectă, propoziție greșită)*, unde *propoziția corectă* reprezintă referința sau rezultatul așteptat în urma corectării *propoziției greșite*. Seturile de date, deși despărțite în mai multe fișiere, sunt concatenate în timpul rulării și se organizează în trei categorii:

- Seturi de date de antrenare/învățare;
- Seturi de date de validare;
- Seturi de date de testare.

Încărcarea unui set de date se face folosind tf.data.TextLineDataset, apoi asupra fiecărei linii se aplică o operație de map pentru a împărți linia cu perechea în cele două propoziții, folosind tf.data.TextLineDataset cu separatorul '\t'. La pasul de încărcare se folosește și modulul os[20] din Python pentru a găsi calea către fișierul cu setul de date. După încărcarea seturilor de date sunt folosite două tokenizere (unul pentru propozițiile greșite, iar celălalt pentru propozițiile corecte) folosind tfds.features.text.SubwordTextEncoder. Codificatoarele de text tfds.features.text.TextEncoder de acest tip sunt construite dintr-un corpus de propoziții din care vor fi generate sub-cuvinte folosind metoda tfds.features.text.SubwordTextEncoder.build_from_corpus. Rezultatul obținut este o mapare a cuvintelor identificate în text la numere întregi. Aceste numere pot fi folosite pentru a codifica cuvintele în numere, respectiv pentru a decodifica numerele în cuvinte.

[18] Jupyter Notebook. Disponibil: https://jupyter.org. [Accesat: 1 Octombrie 2019].
[19] Google Colab. Disponibil: https://colab.research.google.com. [Accesat: 24 Februarie 2020].
[20] Miscellaneous operating system interfaces. Disponibil: https://docs.python.org/3/library/os.html. [Accesat: 1 Iunie 2020].

Pentru a putea antrena, valida și testa modelul este necesar ca toate seturile de date să fie codificate folosind cele două tokenizere (pentru a codifica atât propozițiile greșite cât și propozițiile corecte). În plus fiecare propoziție va avea două tokenuri speciale, un token de start (*sos*: Start Of String – Început De Șir) și unul de sfârșit (*eos*: End Of String – Final De Șir). Fiecare codificator este compus dintr-un vocabular cu o dimensiune specificată la construcție. Cum fiecare cuvânt are corespondent un identificator numeric din vocabular, mai mic ca dimensiunea vocabularului, valorile atribuite pentru cei doi tokeni speciali trebuie alese astfel încât să fie mai mari sau egale cu dimensiunea vocabularului.

Cea mai importantă este cea surprinsă asupra lunii Noiembrie.	
Token	**Sub-cuvânt**
194	Ce
3	a
19	mai
1638	important
7	ă
28	este
362	cea
613	sur
616	prin
10	să
983	asupra
1036	luni
8	i
1120	No
124	ie
863	mb
614	rie
1827	.

Tabelul 4 Exemplu Tokenizare pentru Transformator

Cuvintele pentru care nu exista o asociere în vocabular au fost despărțite în sub-cuvinte pentru care exista o asociere, sau au fost create asocieri noi acolo unde nu existau.

Vocabularul a fost construit pe baza seturilor de date. În Tabelul 4 Exemplu Tokenizare este exemplificat procedeul de tokenizare.

5.1.1 Setul de Date

Setul de date folosit pentru antrenarea, validarea și testarea modelului este corpusul RONACC[21]. Setul de date inițial era compus din perechi de linii, ca în Tabelul 5 Exemplu Set de Date, unde prima linie era cea corectă, iar a doua cea greșită, dar a fost preprocesat astfel încât perechile să se găsească pe aceeași linie (facilita o încărcare a datelor mai ușor de realizat) despărțite folosind caracterul tab ('\t' pentru fișiere TSV – se putea folosi și caracterul ',' pentru fișiere CSV dar cum este un semn de punctuație ar fi trebuit evitat în text) ca în Tabelul 6 Set de Date Modificat. La citirea din fișier datele sunt interpretate folosind separatorul '\t' și se creează seturile de date corespunzătoare, de forma *(propoziție greșită, propoziție corectă)*. Greșelile au fost evidențiate, alături de variantele corecte prin îngroșare.

Propoziție Corectă	**Propoziție Greșită**
Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **improvizate**.	Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **inprovizate**.
Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținătorii** căreia spun că este folosită din 1383, și Stella Artois din 1366	Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținători** căreia spun că este folosită din 1383, și Stella Artois din 1366.
Cea mai importantă este **cea** surprinsă asupra **lunii** Noiembrie.	Cea mai importantă este **ceea** surprinsă asupra **luni** Noiembrie.

Tabelul 5 Exemplu Set de Date

Pentru conversie a fost folosit utilitarul paste[22]:

```
paste -s -d "\t\n" original_dataset > dataset
```

Propoziții Concatenate (tab-urile au fost subliniate)
Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **improvizate**._____Acoperișul din șindrilă nu se mai păstrează, fiind înlocuită cu țiglă în 1936, fapt ce a necesitat sprijinirea acoperișului cu structuri **inprovizate**.
Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținătorii** căreia spun că este folosită din 1383, și Stella Artois din 1366. Alte mărci comerciale utilizate vreme îndelungată sunt Löwenbräu, **deținători** căreia spun că este folosită din 1383, și Stella Artois din 1366.
Cea mai importantă este **cea** surprinsă asupra **lunii** Noiembrie._____Cea mai importantă este **ceea** surprinsă asupra **luni** Noiembrie.

Tabelul 6 Set de Date Modificat

[21] Romanian National Audiovisual Council Corpus. Disponibil: https://nextcloud.readerbench.com/index.php/s/9pwymesT5sycxoM. [Accesat: 10 Iunie 2020].
[22] Paste. Disponibil: https://www.freebsd.org/cgi/man.cgi?query=paste. [Accesat: 10 Iunie 2020].

5.1.2 Mascarea

Măștile folosite sunt cele pentru umplutură și cele pentru privire înainte.

Măștile pentru umplutură asigură că modelul nu va trata umplutura ca intrare, indicând locurile unde se află valori de zero. Pentru a crea masca se folosește tf.math.equal pentru a marca pozițiile egale cu zero.

Măștile cu privire înainte sunt utilizate pentru a indica ce tokeni nu trebuie utilizați. De exemplu, pentru a prezice cuvântul i, doar cuvintele $1, 2, \dots, i-1$ vor fi utilizate, astfel nu îi este permis modelului să vadă ieșirea așteptată. Pentru acest lucru se folosește tf.linalg.band_part pentru a crea o matrice inferior triunghiulară de dimensiune $i \times i$ ce este apoi complementată la o matrice superior triunghiulară fără diagonala principală.

Funcția de creare a măștilor se ocupă cu măștile pentru codificare (mască de umplutură) și decodificare (mască de umplutură) folosită în al doilea bloc de atenție al decodorului pentru a masca ieșirile codificatorului, dar și o mască combinată (mască de umplutură și mască de privire înainte) folosită în primul bloc de atenție al decodorului utilizată atât pentru umplutură cât și pentru a masca tokenii viitori din intrare primiți de decoder. Primele două măști sunt create relativ la propoziția greșită, iar cea combinată relativ la propoziția corectă.

5.2 Atenția

Calculul atenției presupune o funcție ce primește o interogare și un set de perechi cheie – valoare și returnează o mapare ca ieșire [6].

5.2.1 Atenția Produsului Scalar Scalat

Calculul atenției asupra unui set (pentru calcul simultan) de interogări de dimensiune d_k grupate într-o matrice Q cu l_k linii și a unor chei de dimensiune d_k grupate într-o matrice K cu l_k linii și a unor valori de dimensiune d_v grupate într-o matrice V cu l_k linii [6] este realizat astfel:

1. Matricele Q și K^T sunt înmulțite folosind tf.linalg.matmul;
2. Rezultatul înmulțirii este scalat cu $\sqrt{d_k}$ folosind tf.math.sqrt;
3. Dacă o mască a fost trimisă ca parametru, aceasta este înmulțită cu o valoare apropriată de $-\infty$ $(-1e9)$ și apoi este adunată la rezultat;
4. Se calculează ponderile atenției: $softmax(\frac{QK^T}{\sqrt{d_k}})$ folosind tf.nn.softmax;
5. Atenția este dată de rezultatul ponderilor înmulțit cu matricea V folosind tf.linalg.matmul [9].

5.2.2 Atenție cu mai Multe Capete

Pentru a calcula atenția cu mai multe capete, interogarea, cheia și valoarea trebuie împărțite în num_{heads} vectori [9]. Folosind tf.keras.layers.Dense se calculează ponderile pentru fiecare în parte, apoi acestea sunt împărțite în mai multe capete. Având cele trei matrice împărțite

pentru mai multe capete, se calculează pentru fiecare capăt atenția și se concatenează rezultatele folosind tf.reshape (datele sunt ținute într-o structură multi-dimensională). În final, pentru a obține ieșirea, atenția este trecută printr-un strat de tipul tf.keras.layers.Dense.

5.3 Straturi de Codare

Straturile de codare sunt alcătuite folosind straturi de atenție cu mai multe capete, adunare și normalizare și rețele de transmitere înainte.

5.3.1 Rețea de Transmitere Înainte Point Wise

Rețeaua de Transmitere Înainte este implementată folosind tf.keras.Sequential cu două straturi liniare tf.keras.layers.Dense cu un strat de activare unitate liniară rectificată (ReLU) între ele. Dimensiunea stratului intern este dff, iar intrarea și ieșirea au dimensiunea d_{model}.

5.3.2 Strat de Codificare

Fiecare strat de codificare este format din sub-straturi de:

- Atenție cu mai multe capete (cu mască de umplutură);
- Rețele de transmitere înainte point wise [19].

Fiecare dintre aceste straturi are o legătură reziduală [10] în jurul ei urmată de o normalizare [11] a stratului de tipul tf.keras.layers.LayerNormalization. Conexiunile reziduale ajută la evitarea problemei gradientului dispărut în rețelele adânci. Rezultatul fiecărui strat secundar este $LayerNorm(x + Sublayer(x))$ [19].

5.3.3 Strat de Decodificare

Fiecare strat de decodificare este format din sub-straturi de:

- Atenție mascată cu mai multe capete (cu mască de privit înainte și mască de umplutură);
- Atenție cu mai multe capete (cu mască de umplutură). V (valoare) și K (cheia) primesc ieșirea codificatorului ca intrări. Q (interogare) primește ieșirea din sub-stratul de atenție mascată cu mai multe capete;
- Rețele de transmitere înainte point wise [19].

Fiecare dintre aceste straturi are o legătură reziduală [10] în jurul ei urmată de o normalizare [11] a stratului de tipul tf.keras.layers.LayerNormalization. Conexiunile reziduale ajută la evitarea problemei gradientului dispărut în rețelele adânci. Rezultatul fiecărui strat secundar este $LayerNorm(x + Sublayer(x))$ [19].

Deoarece Q primește ieșirea din primul bloc de atenție al decodificatorului, iar K primește ieșirea codificatorului, ponderile de atenție reprezintă importanța acordată intrării decodificatorului pe baza ieșirii codificatorului. Cu alte cuvinte, decodificatorul prezice următorul cuvânt, uitându-se la ieșirea codificatorului și realizând auto-regresie [8] la propria sa ieșire [19].

5.4 Codare

Codarea este realizată prin încorporarea intrării/ieșirii, codificarea poziționlă a acesteia și un număr de straturi de codare. Același număr de straturi este folosit atât pentru codificare cât și pentru decodificare.

5.4.1 Codificare Poziționlă

Codificarea poziționlă este implementată folosind np.sin și np.cos pentru calculul funcțiilor trigonometrice. Calculele sunt făcute vectorial: se calculează unghiul folosind np.power pentru ridicarea la putere, apoi se creează doi noi vectori – pentru pozițiile pare se calculează sinusul, iar pentru pozițiile impare se calculează cosinusul. Pentru fiecare poziție se creează un vector de valori corespunzătoare indicilor cuvintelor (dimensiunea vocabularului) ca în Figura 11 Codificare Poziționlă.

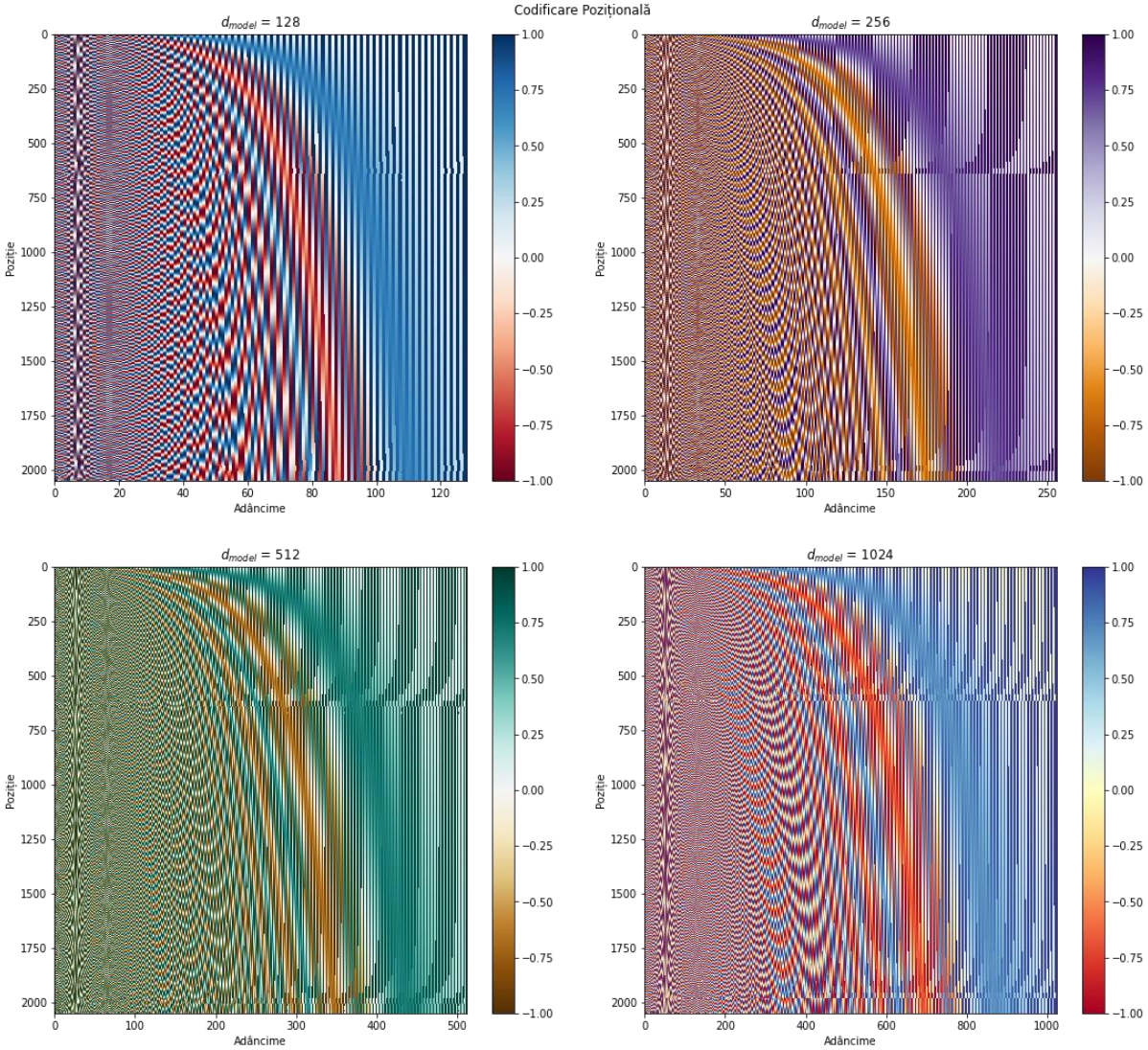

Figura 11 Codificare Poziționlă

5.4.2 Codificator

Codificatorul este format din:

- Încorporare de intrare;
- Codificare poziţională;
- Straturi de codificare.

Intrarea este pusă printr-o încorporare tf.keras.layers.Embedding care este însumată cu codificarea poziţională. Rezultatul acestei însumări este intrarea în straturile codificatorului. Ieşirea codificatorului este intrarea către decodificator [19].

5.4.3 Decodificator

Decodificatorul este format din:

- Încorporare de ieşire;
- Codificare poziţională;
- Straturi de codificare.

Ţinta este pusă printr-o încorporare tf.keras.layers.Embedding care este însumată cu codificarea poziţională. Rezultatul acestei însumări este intrarea în straturile decodificatorului. Ieşirea decodificatorului este intrarea în stratul liniar final [19].

5.5 Transformator

Transformatorul este alcătuit din codificator, decodificator şi un strat final liniar de dimensiunea vocabularului ţintă, implementat cu tf.keras.layers.Dense.

5.5.1 Hyperparametri

Modelul foloseşte următorii hyperparametri:

- num_{layers}: numărul de straturi de codare;
- d_{model}: dimensiunea vectorilor de tokeni pentru intrare/ieşire;
- dff: dimensiunea straturilor interne pentru reţeaua de transmitere înainte;
- num_{heads}: numărul de capete pentru calculul atenţiei cu mai multe capete.

5.6 Optimizator

Adam [12] este optimizatorul folosit, disponibil ca tf.keras.optimizers.Adam. Rata de învăţare este implementată cu tf.keras.optimizers.schedules.LearningRateSchedule:

1. Se calculează $\frac{1}{\sqrt{step}}$ folosind tf.math.rsqrt;
2. Sa calculează $step \cdot {warmup_{steps}}^{-1.5}$ folosind operaţiile din Python;
3. Se alege minimul celor două valori folosind tf.math.minimum;
4. Rezultatul este înmulţit cu $\frac{1}{\sqrt{d_{model}}}$ calculat cu tf.math.rsqrt.

Această rată de învățare este dinamică, crescând la început, iar apoi diminuându-se pe măsură ce învață mai mult după pașii de încălzire și depinde de dimensiunea modelului, așa cum se observă în Figura 12 Variația Ratei de Învățare.

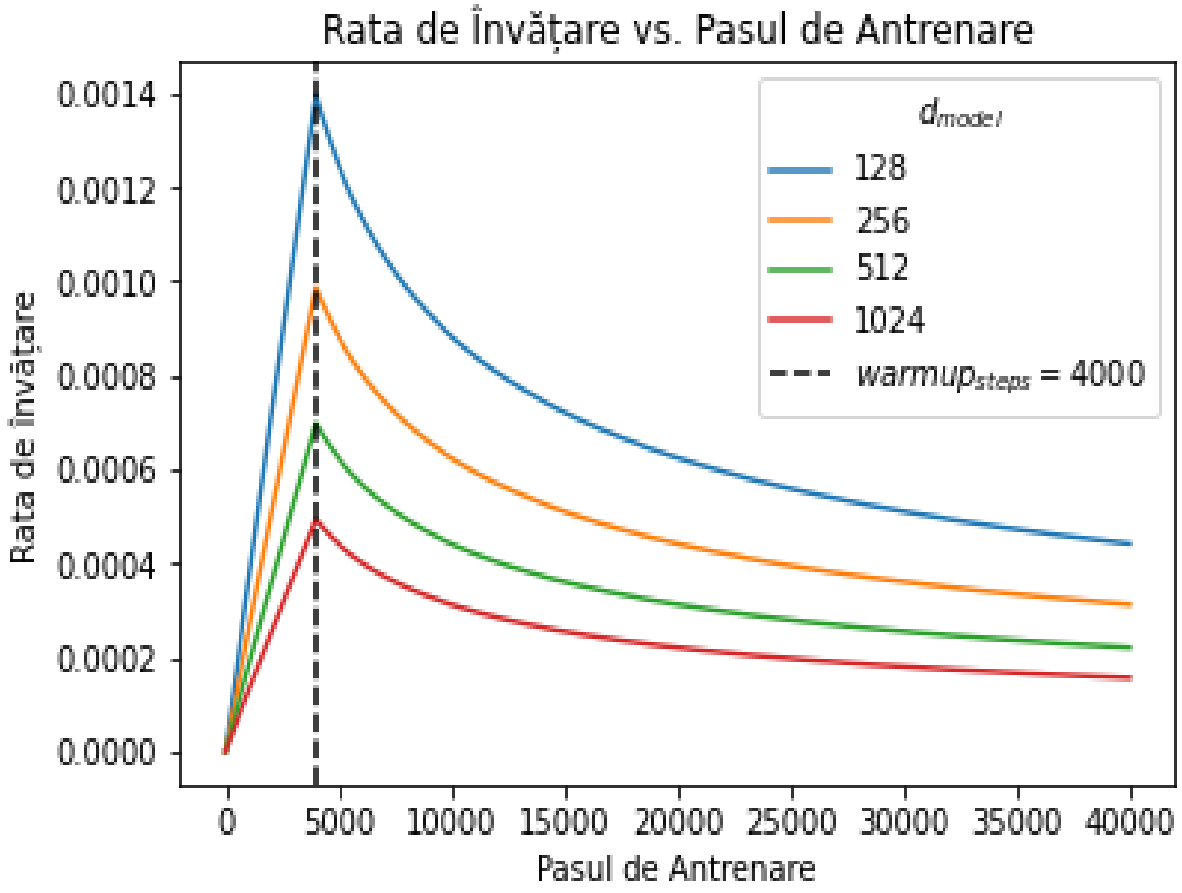

Figura 12 Variația Ratei de Învățare

5.7 Pierderea și Metrici

Pierderea este implementată folosind tf.keras.losses.SparseCategoricalCrossentropy și ignoră, folosind o mască, valorile de umplutură.

La antrenare (respectiv evaluare) se mai folosesc alte două metrici: o pierdere implementată cu tf.keras.metrics.Mean și o acuratețe folosind tf.keras.metrics.SparseCategoricalAccuracy.

5.8 Antrenare și Puncte de Control

Antrenarea este realizată în mai multe epoci. Din când în când, după unele epoci se face un punct de control pentru a salva antrenarea făcută.

Pasul de antrenare este o funcție de tipul tf.function. Parametrii funcției sunt propoziția greșită și propoziția corectă. Funcția @tf.funciton trace-compilează train_step într-un graf TF pentru o execuție mai rapidă. Funcția este specializată în forma precisă a tensorilor

argumentului. Pentru a evita retracing din cauza lungimilor secvenței variabile sau a dimensiunilor variabile ale lotului (ultimul lot este mai mic), se utilizează input_signature pentru a specifica forme mai generice [19].

Având în vedere faptul că train_step este executată frecvent, în loc ca procesul de compilare sa fie executat la cerere, la începutul fiecărei execuții a funcției (JIT – just-in-time), se introduce o optimizare care constă în precompilarea funcției înaintea execuției, în vederea accelerării rulării (tracing just-in-time compilation), aceasta bazându-se pe observațiile interpretorului în legătură cu funcționalitățile executate cel mai frecvent.[23]

```
train_step_signature = [
                        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
                        tf.TensorSpec(shape=(None, None), dtype=tf.int64),
                       ]

@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
    tar_inp = tar[:, :-1]
    tar_real = tar[:, 1:]
    # train step code
```

Transformatorul este un model auto-regresiv [8]: face predicții câte o parte la un moment dat și își folosește ieșirea de până acum pentru a decide ce să facă în continuare [19].

În timpul antrenării, acest exemplu folosește *forțarea-profesorului*. *Forțarea-profesorului* trece adevărata ieșire la următoarea etapă, indiferent de ceea ce modelul prezice la pasul curent [19]. Pentru a realiza acest lucru, din propoziția corectă se creează două propoziții:

- Prima propoziție (propoziția corectă de intrare) nu are ultimul cuvânt;
- A doua propoziție (propoziția corectă de referință) nu are primul cuvânt; astfel pentru fiecare indice din prima propoziției, in a doua propoziție se va găsi cuvântul ce trebuie prezis.

Pe măsură ce transformatorul prezice fiecare cuvânt, atenția îi permite să privească cuvintele anterioare din secvența de intrare pentru a prezice mai bine următorul cuvânt [19].

Sunt create apoi măștile folosind propoziția greșită și propoziția corectă de intrare, se obțin predicțiile folosind transformatorul și se calculează pierderea dintre propoziția corectă de referință și predicție. Se calculează gradienții folosind tf.GradientTape și apoi se aplică asupra optimizatorului folosind tf.keras.optimizers.Optimizer.apply_gradients. Se salvează pierderea de antrenare și acuratețea dintre propoziția corectă de referință și predicție.

Pentru evitarea overfitingului [14] se folosește renunțarea cu tf.keras.layers.Dropout.

[23] Wikipedia, „Tracing just-in-time compilation,” [Interactiv]. Disponibil: https://en.wikipedia.org/wiki/Tracing_just-in-time_compilation. [Accesat: 20 Iunie 2020].

5.8.1 Puncte de Control

Punctele de control sunt niște fișiere în care sunt salvate rezultatele antrenării. Punctele de control sunt create folosind tf.train.Checkpoint, iar pentru managementul acestora se folosește un manager creat cu tf.train.CheckpointManager ce permite restaurarea și salvarea unui punct de control, dar și limitarea numărului de puncte de control salvate.

5.8.2 Antrenare

Pentru fiecare epocă sunt resetate stările pierderii și acurateței de antrenare. În fiecare epocă se ia pe rând câte un lot din setul de antrenare și se face un pas de antrenare. După un anumit număr de loturi se afișează statistici despre pierdere și acurateță. La finalul unei epoci sunt afișate pierderea, acuratețea și timpul de antrenare (pentru acesta este folosit modulul time[24]).

5.9 Evaluare

Evaluarea este realizată într-o singură epocă, se ia pe rând câte un lot din setul de testare și se face un pas de evaluare. După un anumit număr de loturi se afișează statistici despre pierdere și acurateță. La finalul epocii sunt afișate pierderea, acuratețea și timpul de evaluare.

Pasul de evaluare este asemănător cu pasul de antrenare, doar că nu se mai calculează și nici nu se mai aplică gradienții.

5.9.1 Corectare

Procesul de corectare al unei propoziții presupune codificarea acesteia folosind tokenizerul pentru propoziții greșite și adăugarea celor doi tokeni speciali (cel de start și cel de stop). Folosind această intrare se dorește construirea ieșirii folosind transformatorul. Pentru acest lucru se inițializează ieșirea cu tokenul special de stop pentru tokenizerul pentru propoziții corecte. La fiecare pas sunt create măștile, se obțin predicțiile și ponderile pentru atenție folosind transformatorul, se alege ultimul cuvânt din predicții (se oprește iterarea dacă este cuvântul de final) și se concatenează la ieșire. Dacă există o limitare în lungime a ieșirii se oprește iterarea; sunt returnate ieșirile și ponderile finale. Ponderile sunt folosite pentru a reprezenta grafic distribuția atenției pe diferite nivele și blocuri. Ieșirile sunt decodificate folosind tokenizerul pentru propoziții corecte pentru a obține predicția – propoziția corectată. Eventual se poate afișa și propoziția de referință dacă există și este trimisă ca parametru.

5.9.2 Testare

Pentru testare au fost încercate câteva propoziții cu greșeli pentru a vedea predicțiile modelului legate de variantele corecte folosind funcția de corectare, dar și pentru a urmări (opțional) atenția în diferite straturi ale modelului.

[24] Time. Disponibil: https://docs.python.org/3/library/time.html. [Accesat: 2 Iunie 2020].

Capitolul 6 Evaluarea Rezultatelor

Insanity is doing the same thing over and over again, but expecting different results.
– Rita Mae Brown, Sudden Death

Fiind un model statistic, testarea corectitudinii sistemului trebuie distinsă de evaluarea corectitudinii acestuia. Astfel sistemul poate funcționa corect în parametrii de execuție, dar totuși poate produce un rezultat care să nu fie considerat corect. Acest lucru este datorat naturii problemelor de inteligență artificială unde se încearcă aproximarea unei soluții, ceea ce înseamnă că soluția obținută nu este optimă sau neapărat corectă. Partea de evaluare investighează natura rezultatelor obținute de sistem.

6.1 Testare

Pentru testarea funcționalității sistemului și a corectitudinii s-au folosit teste unitare și de integritate.

Interacțiunea principală a utilizatorilor cu sistemul este realizată prin intermediul funcției de corectare a unei propoziții, care primește ca parametru o propoziție greșită și opțional mai poate primi ca parametrii varianta corectă ca referință și numele unui strat pentru a afișa ponderile atenției. Pentru a asigura sistemul asupra datelor de intrare au fost făcute teste de tip (sistemul trebuie să primească ca intrare o propoziție greșită și eventual una corectă care să fie șiruri de caractere și nu altceva). Pentru acest lucru s-au introdus operații de assert de tipul `assert isinstance(sentence, str) == True`. În continuare au fost folosite teste unitare pentru a asigura corectitudinea execuției sistemului cu diferite intrări și asigurarea că sistemul va primi de la utilizator date de intrare corecte.

Pentru a asigura integritatea sistemului au fost realizate teste de integritate asupra operațiilor efectuate. De exemplu, pentru a asigura integritatea calculelor atenției cu mai multe capete, atunci când matricele sunt împărțite la mai multe capete, dimensiunea modelului (corespunzătoare intrării) trebuie să fie divizibilă cu numărul de capete utilizate de sistem. Acest lucru este asigurat folosind `assert d_model % self.num_heads == 0`.

Funcționalitățile testate ale sistemului au fost implementate corect și complet. Acesta este antrenat și evaluat pe un set de date (setul de date trebuie să fie formatat corespunzător formatului TSV) și produce rezultate concrete. Chiar dacă o propoziție corectată nu este corectă, sistemul totuși poate funcționa corect, iar problema să fie una de acuratețe/precizie a rezultatelor prezise. Un exemplu de rulare corectă în care rezultatul nu este cel așteptat:

```
correct("Cea mai importantă este ceea surprinsă asupra luni Noiembrie.",
        real_sentence="Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.")
Input: Cea mai importantă este ceea surprinsă asupra luni Noiembrie.
Predicted correction: Cel mai importantă este cea surprinsă asupra lunii
Noiembrie.
Real correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
BLEU score: 0.8633
```

Această situație este specifică problemelor de învățare automată și trebuie tratată din punct de vedere al evaluării sistemului, nu al testării corectitudinii.

Pentru a evita posibile erori și defecte în cadrul codului sursă au fost respectate convenții de stil de cod, cum sunt alinierea, linii mai scurte de 80 de caractere, modulare, funcții și metode cu rol clar și denumiri de variabile pline de sens.

6.2 Evaluare

Modelul a fost evaluat pe mai multe seturi de date de antrenare și un set de test cu 1519 perechi de propoziții: mic: 7082 de perechi de propoziții; mediu: 7082 + 50000 de perechi de propoziții; mare: 7082 + 1000000 de perechi de propoziții.

La final rezultatele au fost validate folosind un set de validare de 1518 perechi de propoziții. Modelul a fost antrenat timp de 100 de epoci pe setul de date mic și mediu, la fiecare epocă fiind testat și măsurându-se acuratețea (Figura 13 Acuratețea pentru setul de date mic și Figura 15 Acuratețea pentru setul de date mediu) și pierderea (Figura 14 Pierderea pentru setul de date mic și Figura 16 Pierderea pentru setul de date mediu). Din grafice se poate observa cum aceste valori converg, iar în plus pierderea tinde spre zero.

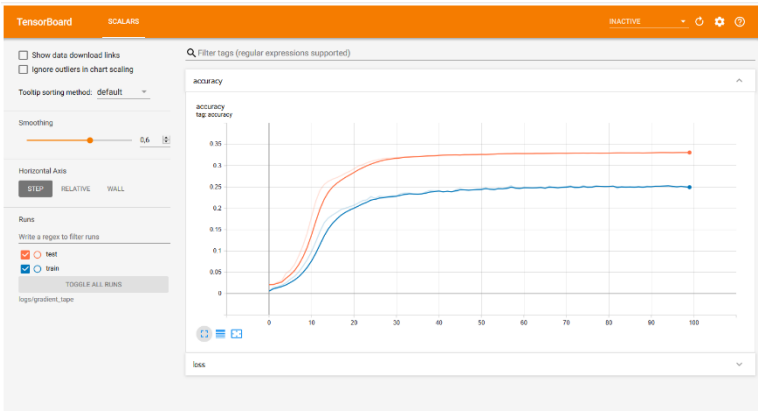

Figura 13 Acuratețea pentru setul de date mic

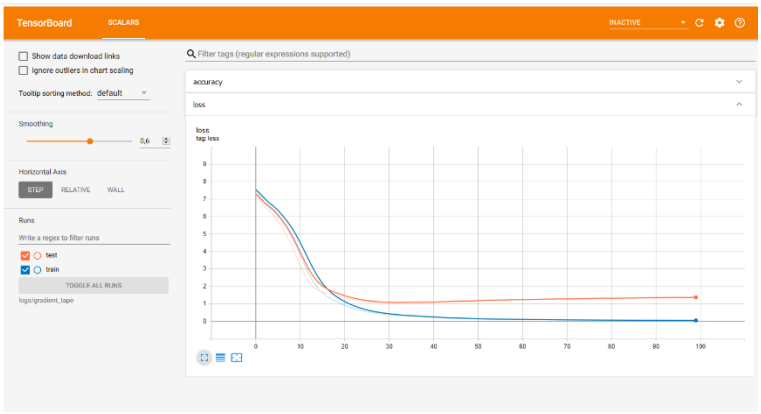

Figura 14 Pierderea pentru setul de date mic

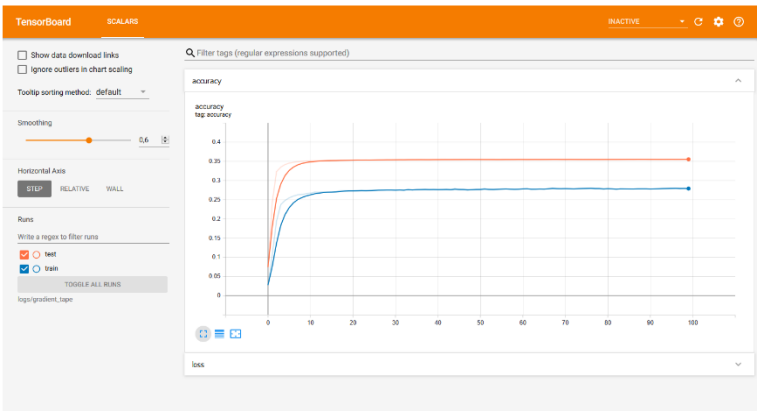

Figura 15 Acuratețea pentru setul de date mediu

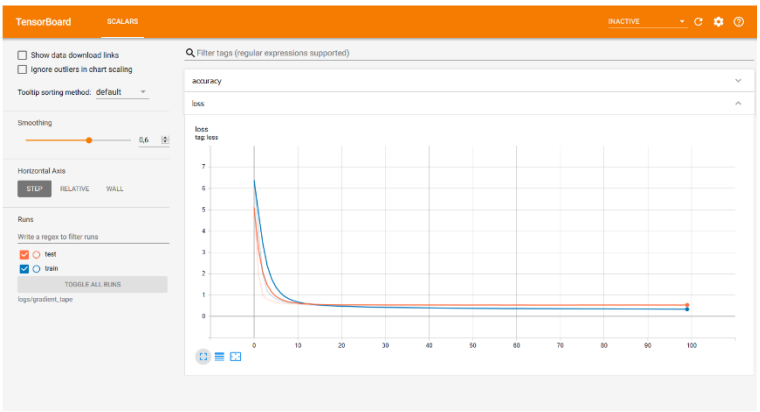

Figura 16 Pierderea pentru setul de date mediu

Modelul propus în [5] obține un scor BLEU de 27.3, respectiv 38.1 pentru traducerea din engleză în germană, respectiv franceză pentru Transformatorul cu model de bază și un scor BLEU de 28.4, respectiv 41.8 pentru Transformatorul mare, în timp ce modelul implementat în lucrarea de față obține un scor BLEU competitiv de 20.56 pentru corectarea de texte în limba română având un set de antrenare modest și un scor de 33.29 având un set de antrenare de dimensiune mai mare, așa cum se poate vedea în Tabelul 7 Rezultate antrenare, ceea ce reprezintă o îmbunătățire substanțială. Datele nu pot fi comparate direct, deoarece articolul [5] folosea alte seturi de date, opera asupra altor probleme și cu alte limbi. Totuși, trebuie avut în vedere că domeniul este destul de limitat la ora actuală pentru corectarea textelor în limba română pentru a face comparații asupra altor sisteme. În tabel se poate observa cum un set de date de antrenare mai mare poate influența substanțial scorul (45.29).

Setul de date	Epoci	Scorul BLEU	Timp de antrenare
Mic	100	20.56	9.66 secunde / epocă
Mediu	100	33.29	99 secunde / epocă
Mare	5	**45.29**	1032 secunde / epocă

Tabelul 7 Rezultate antrenare

Calculul scorului BLEU [15] a fost implementat folosind nltk.translate.bleu_score.sentence_bleu și nltk.translate.bleu_score.SmoothingFunction cu method4 pentru cazul propozițiilor scurte [20]. O altă metrică importantă este atenția pe care cuvintele o acordă altor cuvinte, așa cum se poate vedea în exemplele din Anexa A (Ponderile Atenției).

6.2.1 Evaluare Calitativă

În Anexa A (Ponderile Atenției), sunt prezentate câteva exemple de rulare a modelului pe diferite propoziții de intrare. Dacă rezultatele cantitative obținute au fost în medie satisfăcătoare, din punct de vedere al scorului BLEU, de exemplu, pentru care au fost observate îmbunătățiri, calitativ lucrurile stau puțin diferit din cauza particularităților fiecărui tip de greșeală gramaticală.

Un exemplu pe care modelul îl poate corecta foarte ușor este cazul **ceea – cea**, cum este cazul propoziției *„Cea mai importantă este ceea surprinsă asupra luni Noiembrie.”* pentru care sistemul produce predicția *„Cea mai importantă este cea surprinsă asupra lunii Noiembrie.*” care este și varianta corectă așteptată la ieșire. În schimb, cazul **n-am – nam (lipsa cratimei)** nu este la fel de bine învățat de către model, de exemplu *„„Nici odată nam văzut cartea așa” a mărturisit el.*” pentru care modelul nu ajunge la varianta corectă *„„Niciodată n-am văzut cartea așa”, a mărturisit el.”* fiind un caz mai complicat deoarece o parte lexicală este scrisă greșit prin alipire (nam în loc de n-am) iar alta este scrisă greșit prin despărțire (nici odată în loc de niciodată). Problema corectării provine și din cauza intersecției celor două cazuri.

Un alt exemplu pentru care modelul nu generalizează la fel de bine este cel al cuvintelor pe care nu le cunoaște (nu apar în dicționarul de ieșire) și nu știe cum să facă asocierile. De exemplu, în cazul cuvintelor scrise greșit, dacă acestea nu au fost învățate nu vor putea fi corectate. Pentru propoziția *„În prezent, satul are 3.897 locuitori, prepoderent ucraineni.*” Modelul nu va produce ieșirea așteptată *„În prezent, satul are 3.897 locuitori, preponderent ucraineni.*”, deoarece modelul nu cunoaște asocierea **prepoderent – preponderent** și nu știe că există o greșeală de acest tip. Pentru acest exemplu scorul BLEU este înșelător, undeva la 70%, ceea ce poate părea a fi un rezultat destul de echitabil.

Astfel de cazuri trebuie avute în vedere atunci când se dorește evaluarea modelului. Posibile soluții în astfel de probleme sunt mărirea numărului de epoci de antrenare și folosirea unor seturi de antrenare (cu dezavantajul timpului mărit de antrenare) cu un număr extins de exemple de învățare care să acopere diferite cazuri și greșeli frecvent întâlnite în limbaj.

Totuși, este de remarcat, așa cum se poate observa în Anexa A (Ponderile Atenției), că pentru o propoziție care este scrisă corect cel mai probabil modelul nu va încerca să o modifice, deoarece cuvintele sunt regăsite în dicționar, iar legăturile dintre acestea produc ponderi mari ale atenției.

Capitolul 7 Concluzii

Finis coronat opus. (Sfârșitul încununează opera.)
– Ovidiu, Heroides

Rezultatul acestui proiect este un corector de texte cu o funcționalitate redusă. În realizarea acestuia s-a ținut cont atât de nevoile utilizatorilor cât și de progresul științific și tehnic din domeniu, dar și de tehnologiile disponibile pentru implementarea Transformatorului.

Comparativ cu modelul propus în [5] care a fost antrenat douăsprezece ore pe opt GPU-uri P100, modelul implementat a fost antrenat, pe setul mediu aproape trei ore, iar pe setul mare o oră și jumătate (doar cinci epoci). A fost folosit un singur super set de date pentru modelare și nu au putut fi realizate comparații cu alte seturi de date și în plus, din păcate, dezvoltarea în domeniu nu a făcut posibilă compararea modelului implementat cu alte modele. În continuare implementarea este un punct de pornire în dezvoltarea domeniului și poate fi folosit ca un punct de pornire în dezvoltarea și modelarea unor tehnologii performante și competitive pentru corectarea textelor în limba română. Generalizarea modelului nu este cea așteptată, dar sunt cazuri în care acesta a reușit să prezică cu exactitate o propoziție corectă pornind de la o propoziție ce avea greșeli gramaticale (a se vedea Anexa A Ponderile Atenției).

7.1 Dezvoltări Ulterioare

Dezvoltarea și îmbunătățirea modelului (sau cel puțin al domeniului corectării textelor în limba română) este impetuos necesară, iar acest lucru poate fi realizat folosind mai multe date de antrenare, un număr mai mare de epoci de antrenare sau un set extins de date de validare. Cum tehnologia evoluează, trebuie făcută actualizarea și îmbunătățirea modelului la cele mai noi tehnologii de ultimă oră. Un exemplu de astfel de tehnologie este BERT (Reprezentări Bidirecționale ale Codificatorului de la Transformatoare) [21].

Pe lângă îmbunătățirea modelului, dezvoltările ulterioare ale aplicației impun folosirea unei interfețe pentru interacțiunea dintre utilizator și program. Aplicația trebuie să poată fi folosită din cadrul diferitelor sisteme de operare, cum sunt Android[25], iOS[26], macOS[27], Linux[28] sau Microsoft Windows[29]. Se poate dezvolta și o versiune accesibilă din navigatoare web ca Mozilla Firefox[30] sau Google Chrome[31]. Un alt pas important este integrarea software-ului cu alte aplicații de procesare de texte cum sunt cele din suita Microsoft Office[32].

Codul sură al implementării este disponibil la adresele din Anexa B (Codul Sursă).

[25] Android. Disponibil: https://www.android.com. [Accesat: 20 Iunie 2020].
[26] iOS. Disponibil: https://www.apple.com/ios. [Accesat: 20 Iunie 2020].
[27] macOS. Disponibil: https://www.apple.com/macos. [Accesat: 20 Iunie 2020].
[28] Linux. Disponibil: https://www.linuxfoundation.org. [Accesat: 20 Iunie 2020].
[29] Microsoft Windows. Disponibil: https://www.microsoft.com/windows. [Accesat: 20 Iunie 2020].
[30] Mozilla Firefox. Disponibil: https://www.mozilla.org/firefox/new. [Accesat: 21 Iunie 2020].
[31] Google Chrome. Disponibil: https://www.google.com/chrome. [Accesat: 21 Iunie 2020].
[32] Microsoft Office. Disponibil: https://www.microsoft.com/microsoft-365. [Accesat: 22 Iunie 2020].

Bibliografie

[1] I. Sutskever, O. Vinyals și Q. V. Le, „Sequence to Sequence Learning with Neural Networks,” 10 Septemrbie 2014. [Interactiv]. Disponibil: https://arxiv.org/abs/1409.3215. [Accesat 20 Iunie 2020].

[2] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk și Y. Bengio, „Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” 3 Iunie 2014. [Interactiv]. Disponibil: https://arxiv.org/abs/1406.1078. [Accesat 20 Iunie 2020].

[3] D. Bahdanau, K. Cho și Y. Bengio, „Neural Machine Translation by Jointly Learning to Align and Translate,” 1 Septembrie 2014. [Interactiv]. Disponibil: https://arxiv.org/abs/1409.0473. [Accesat 20 Iunie 2020].

[4] J. Gehring, M. Auli, D. Grangier, D. Yarats și Y. N. Dauphin, „Convolutional Sequence to Sequence Learning,” 8 Mai 2017. [Interactiv]. Disponibil: https://arxiv.org/abs/1705.03122. [Accesat 21 Iunie 2020].

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser și I. Polosukhin, „Attention Is All You Need,” 12 Iunie 2017. [Interactiv]. Disponibil: https://arxiv.org/abs/1706.03762. [Accesat 10 Iunie 2020].

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser și I. Polosukhin, „Attention Is All You Need,” în *Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA, 2017.

[7] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Ł. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes și J. Dean, „Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” 26 Septembrie 2016. [Interactiv]. Disponibil: https://arxiv.org/abs/1609.08144. [Accesat 20 Iunie 2020].

[8] A. Graves, „Generating Sequences With Recurrent Neural Networks,” 4 August 2013. [Interactiv]. Disponibil: https://arxiv.org/abs/1308.0850. [Accesat 20 Iunie 2020].

[9] M. Phi, „Illustrated Guide to Transformers: Step by Step Explanation,” 30 Aprilie 2020. [Interactiv]. Disponibil: https://www.michaelphi.com/illustrated-guide-to-transformers. [Accesat 19 Iunie 2020].

[10] K. He, X. Zhang, S. Ren și J. Sun, „Deep Residual Learning for Image Recognition,” 10 Decembrie 2015. [Interactiv]. Disponibil: https://arxiv.org/abs/1512.03385. [Accesat 21 Iunie 2020].

[11] J. Lei Ba, J. R. Kiros și E. G. Hinton, „Layer Normalization,” 21 Iulie 2016. [Interactiv]. Disponibil: https://arxiv.org/abs/1607.06450. [Accesat 21 Iunie 2020].

[12] D. P. Kingma și J. Ba, „Adam: A Method for Stochastic Optimization,” 22 Decembrie 2014. [Interactiv]. Disponibil: https://arxiv.org/abs/1512.03385. [Accesat 6 Iunie 2020].

[13] J. Alammar, „The Illustrated Transformer,” 27 Iunie 2018. [Interactiv]. Disponibil: http://jalammar.github.io/illustrated-transformer. [Accesat 20 Iunie 2020].

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever și R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* vol. I, nr. 15, pp. 1929-1958, Iunie 2014.

[15] K. Papineni, S. Roukos, T. Ward și W.-J. Zhu, „Bleu: a Method for Automatic Evaluation of Machine Translation,” în *Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA, 2002.

[16] Google Brain Team, „All symbols in TensorFlow 2,” TensorFlow, 9 Noiembrie 2015. [Interactiv]. Disponibil: https://www.tensorflow.org/api_docs/python. [Accesat 16 Iunie 2020].

[17] Google Brain Team, „All symbols in TensorFlow Datasets,” TensorFlow, 9 Noiembrie 2015. [Interactiv]. Disponibil: https://www.tensorflow.org/datasets/api_docs/python. [Accesat 16 Iunie 2020].

[18] NumPy, „NumPy Reference,” NumPy, 24 Mai 2020. [Interactiv]. Disponibil: https://numpy.org/doc/stable/reference. [Accesat 16 Iunie 2020].

[19] TensorFlow, „Transformer model for language understanding,” 22 August 2019. [Interactiv]. Disponibil: https://www.tensorflow.org/tutorials/text/transformer. [Accesat 10 Iunie 2020].

[20] Team NLTK, „NLTK documentation,” Natural Language Toolkit, 2001. [Interactiv]. Disponibil: https://www.nltk.org/. [Accesat 21 Iunie 2020].

[21] J. Devlin, M.-W. Chang, K. Lee și K. Toutanova, „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 11 Octombire 2018. [Interactiv]. Disponibil: https://arxiv.org/abs/1810.04805. [Accesat 23 Iunie 2020].

Anexa A Ponderile Atenției

Următoarele exemple au fost obținute în urma antrenării transformatorului pe setul de date mediu. Atenția a fost măsurată la ieșirea (al doilea substrat) din ultimul strat de decodare al decodorului. Exemplele prezentate sunt și cazuri corectate complet, propoziții parțial corectate, dar și propoziții cu erori sau care nu au putut fi corectate.

```
correct("Cea mai importantă este ceea surprinsă asupra luni Noiembrie.",
        real_sentence="Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.",
        plot="decoder_layer2_block2")
Input: Cea mai importantă este ceea surprinsă asupra luni Noiembrie.
Predicted correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
Real correction: Cea mai importantă este cea surprinsă asupra lunii
Noiembrie.
BLEU score: 1.0000
```

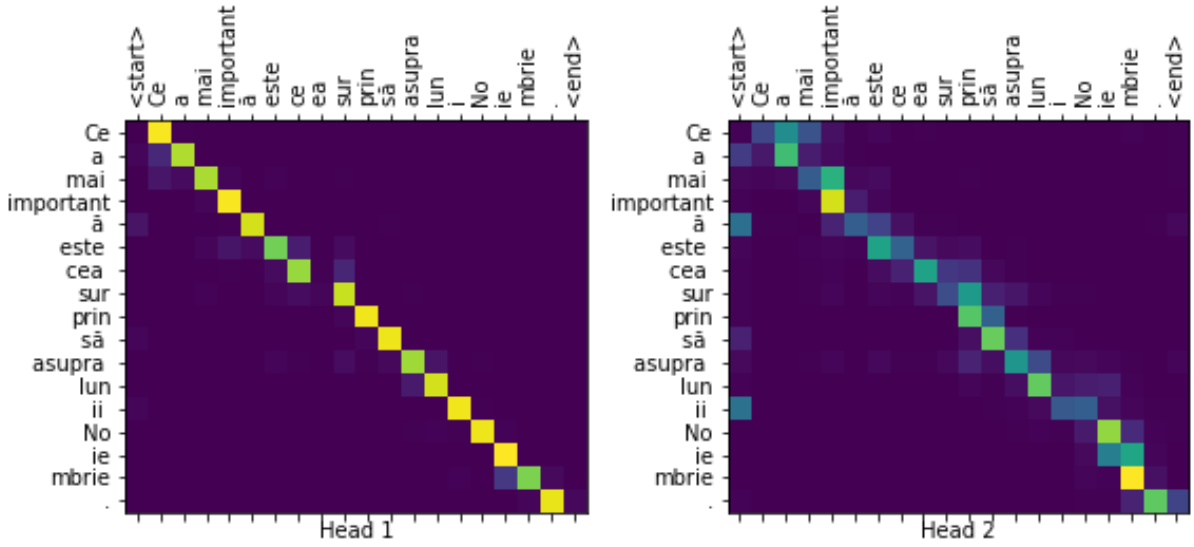

Figura 17 Exemplu 1: Atenție

```
correct("„Nici odată nam văzut cartea așa” a mărturisit el.",
        real_sentence="„Niciodată n-am văzut cartea așa”, a mărturisit
el.",
        plot="decoder_layer2_block2")
Input: „Nici odată nam văzut cartea așa” a mărturisit el.
Predicted correction: „Nici odată nam văzut cartea așa” a mărturisit el.
Real correction: „Niciodată n-am văzut cartea așa”, a mărturisit el.
BLEU score: 0.2741
```

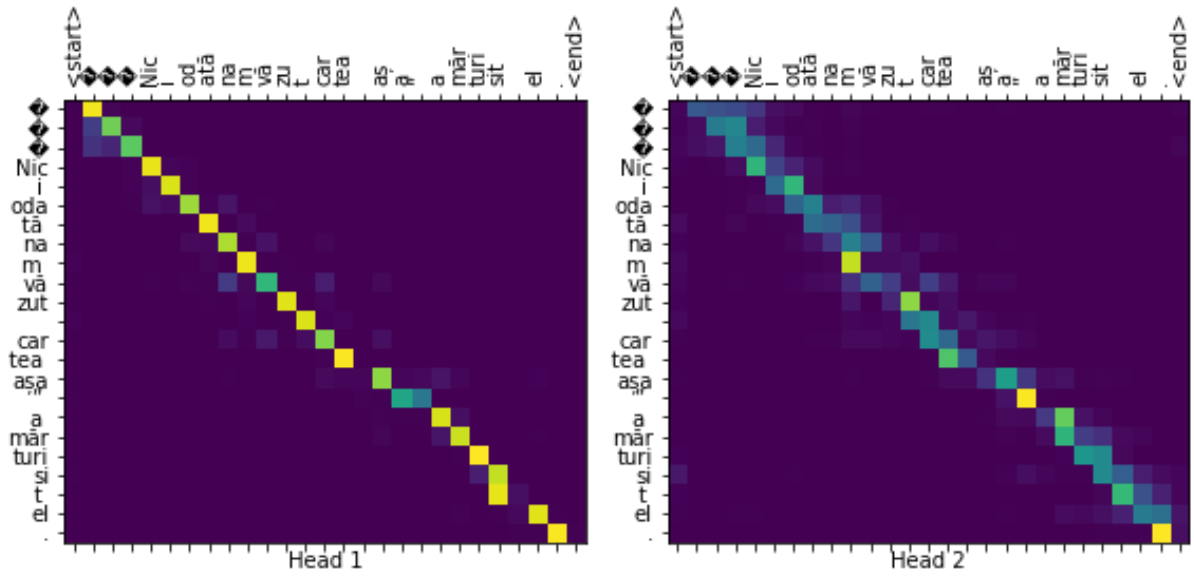

Figura 18 Exemplu 2: Atenție

```
correct("În prezent, satul are 3.897 locuitori, prepoderent ucraineni.",
        real_sentence="În prezent, satul are 3.897 locuitori, preponderent
ucraineni.",
        plot="decoder_layer2_block2")
Input: În prezent, satul are 3.897 locuitori, prepoderent ucraineni.
Predicted correction: În prezent, satul are 3.897 locuitori, prepoderent,
ucraineni.
Real correction: În prezent, satul are 3.897 locuitori, preponderent
ucraineni.
BLEU score: 0.7071
```

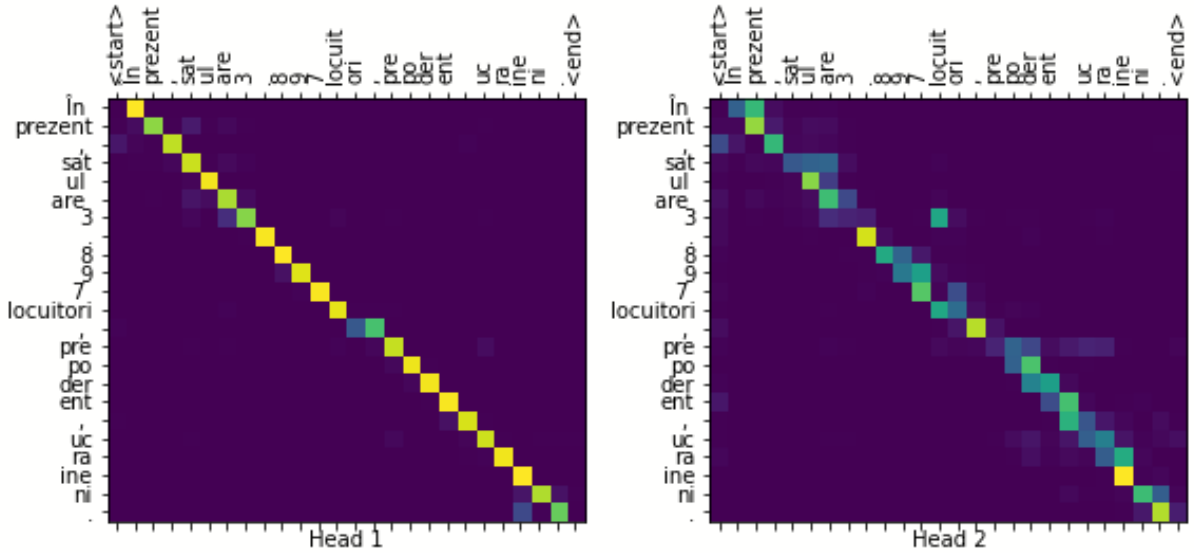

Figura 19 Exemplu 3: Atenție

Anexa B Codul Sursă

Codul sursă folosit în dezvoltarea proiectului, alături de alte documente, este disponibil la următoarele adrese:

- https://gitlab.cs.pub.ro/ioan_florin.nitu/l4-diploma-project
- https://github.com/nitu-catalin1998/diploma-project

Cele două repository-uri conțin aceleași date (duplicate pentru a avea o copie de rezervă). Folosirea lor a facilitat controlul versiunilor și urmărirea modificărilor în timpul dezvoltării software. Pe lângă controlul versiunilor surselor, în repository se află resurse importante pentru dezvoltarea proiectului: seturi de date, configurații ale modelului, grafice și figuri, referințe la articolele și tutorialele folosite în dezvoltarea proiectului. Codul este scris în Python și sunt disponibile două versiuni ce pot fi utilizate. Una dintre versiuni este un Jupyter Notebook ce poate fi executat folosind Google Colab, iar cealaltă versiune este codul Python simplu obținut prin conversia notebookului.