UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE LICENŢĂ

Dezvoltarea unui chatbot cu recunoaștere vocală pe Android

Conducător ştiinţific
Lect. Dr. Lazăr Ioan

Absolvent
Băciulescu Raul-Ovidiu

2023

ABSTRACT

This paper presents the development of a chatbot application for Android devices that combines memory-based functionality, PDF-based conversation capabilities, and voice messaging, aiming to enrich the user experience through personalized and adaptable interactions. In this context, my contribution encompasses training the RobinASR project to enhance speech recognition for the Romanian language. Furthermore, I have developed an Android application that leverages the existing popularity of chatbots for browsers.

Cuprins

Capitolul 1

Introducere

1.1 Motivație

Factorul care a stat la baza deciziei de a dezvoltă această aplicație este reprezentat de faptul că ne aflăm la începutul unei noi ere în educația noastră pe internet, odată cu apariția chatbotilor. Acestea pot furniza răspunsuri la întrebări din diverse domenii și sunt adesea considerate un înlocuitor perfect pentru motoarele de căutare tradiționale. Am optat pentru dezvoltarea unui chatbot pentru telefoane mobile pentru a facilita învățarea și adaptarea utilizatorilor la această evoluție tehnologică.

Am luat decizia de a antrena un model de recunoaștere vocală pe limba română având în vedere pasiunea mea pentru acest domeniu fascinant. Sunt intrigat de procesul prin care un mesaj audio este transformat în text și doresc să înțeleg în profunzime funcționarea sistemelor de recunoaștere vocală. În plus, conștientizez importanța îmbunătățirii modelelor de speech recognition pentru limba română, deoarece acestea pot avea un impact semnificativ în îmbunătățirea experienței utilizatorilor de telefoane mobile și în facilitarea comunicării eficiente în limba maternă. Prin antrenarea unui model de recunoaștere vocală, îmi propun să contribui la avansarea tehnologiei în acest domeniu și să aduc beneficii comunității utilizatorilor de limba română.

1.2 Introducere în lumea roboților de chat

În ultimele decenii tehnologia a evoluat rapid, oferindu-ne noi instrumente pentru a ne ajuta viața de zi cu zi. Inteligență artificială a cunoscut un avans semnificativ, oferind oportunități nelimitate de comunicare și cunoaștere. Un instrument apărut în ultimii ani, despre care marea majoritate a lumii nu este familiară cu el, este ChatGPT.

ChatGPT este un model de limbaj dezvoltat de OpenAI. Este bazat pe arhitec-

tura GPT-3.5, care reprezintă versiunea a treia a Generative Pre-trained Transformer(GPT).

ChatGPT este conceput pentru a răspunde la întrebări și a purta conversații cu utilizatori umani, oferind răspunsuri relevante și coerente. Modelul a fost antrenat pentru a înțelege contextul și a produce texte care să se potrivească cu întrebările și solicitările utilizatorilor. Consider că un chatbot bine antrenat poate deveni o alternativă valoroasă la motoarele de căutare clasice, contribuind astfel la economisirea timpului prețios pe care îl investim în căutarea de informații noi. Capacitatea avansată a unui chatbot de a procesa limbajul natural și de a învăța automat îl poate face chiar mai puternic decât un motor de căutare tradițional.

Diferența majoră constă în faptul că, în timp ce un motor de căutare furnizează o listă de site-uri web relevante în funcție de termenii de căutare introduși, un chatbot poate analiza și înțelege întrebarea formulată și poate oferi un răspuns precis și personalizat, ținând cont de context. Acest lucru se datorează abilităților sale de procesare a limbajului natural și de adaptare la nevoile specifice ale utilizatorului.

1.3 Introducere în recunoașterea vocală

Recunoașterea vocală a devenit o componentă esențială a tehnologiilor de prelucrare a limbajului natural și a interacțiunii om-calculator. Această tehnologie a avut un impact semnificativ într-o varietate de domenii, inclusiv în aplicații de asistență vocală, transcrierea automată, comenzi vocale în mașini sau dispozitive inteligente și multe altele.

În următoarele capitole, voi expune cercetarea realizată până în prezent în acest domeniu, utilizând câteva dintre cele mai semnificative lucrări și voi efectua antrenarea proiectului RobinASR pentru a realiza recunoașterea vocii.

1.4 Descrierea pe scurt a aplicației

Aplicația dezolvatată se numește ”Gepeto”, este o aplicație Android și are ca scop comunicarea cu un chatbot. Funcțiile de bază sunt: trimiterea de mesaj într-un chat care are memorie, trimiterea de mesaje audio, dar și trimiterea de mesaje bazate pe fișiere PDF. Mai precis, chatbot-ul poate răspunde întrebărilor bazându-se pe conținutul PDF-ului primit.

Capitolul 2

Prezentarea caracteristicilor unice ale limbii române

2.1 Variația dialectală și regională

Limba română prezintă variații dialectale și regionale semnificative în ceea ce privește pronunția, gramatica și lexicul. Aceste diferențe pot fi observate între regiunile țării sau a comunităților lingvistice din România. De exemplu, unele cuvinte sau expresii pot fi folosite într-un anumit mod într-o regiune și în alt fel în altă regiune. Aceste diferențe pot face dificilă recunoașterea vocală, deoarece sistemul trebuie să fie capabil să identifice și să se adapteze la aceste variații pentru a obține o acuratețe ridicată în recunoașterea vocală a vorbitorului. De exemplu, cuvântul ”birou” poate fi pronunțat ”biro” sau ”biurou” în funcție de regiunea în care este vorbită limba. Astfel, sistemul de recunoaștere vocală trebuie să fie capabil să recunoască aceste variații și să le interpreteze corect.

2.2 Cuvinte compuse și cuvinte cu sens dublu

Limba română are o mare varietate de cuvinte compuse și cuvinte cu sens dublu, care pot crea confuzie în procesul de recunoaștere vocală. De exemplu, cuvântul ”cașcaval” poate fi interpretat în două moduri diferite, fie că un singur cuvânt, fie ca două cuvinte separate - ”caș” și ”caval”. De asemenea, cuvintele compuse, cum ar fi ”autobuz” sau ”alb-negru”, pot fi dificil de interpretat dacă sistemul de recunoaștere vocală nu este capabil să recunoască componentele cuvintelor.

Capitolul 3

Rezumat al cercetărilor existente privind recunoașterea vocală pentru limba română

3.1 Evoluția arhitecturii aplicațiilor de speech recognition

Învățarea profundă(Deep learning) este o tehnică de inteligență artificială care a devenit extrem de populară în ultimii ani, datorită capacității sale de a aborda probleme complexe în domenii precum recunoașterea imaginilor, recunoașterea vocală, traducerea automată și multe altele. Învățarea profundă a luat amploare în mare parte datorită avansurilor în puterea de calcul și disponibilitatea hardware-ului. Această evoluție a permis cercetătorilor și dezvoltatorilor să creeze și să antreneze modele de învățare profundă cu mai multe straturi, care pot fi utilizate într-o varietate de aplicații, de la recunoașterea imaginilor la traducerea automată a limbilor.

Rețelele neuronale artificiale sunt inspirate din biologia creierului uman[23]. În mod similar rețeaua neuronală artificială este compusă din unități de procesare numite neuroni, care sunt conectați prin sinapse. Conducerea impulsurilor nervoase urmează principiul tot sau nimic (all or none) . Dacă un neuron răspunde, trebuie să răspundă complet. Neuronul artificial cea mai mică și importantă parte a sistemului. Acesta este de fapt o funcție de forma $h(x) = \sigma(w^T x + b)$. Vectorul w e cunoscut că „weights", scalarul sau termenul liber, b este cunoscut că „bias". Tuplul dintre w și b se numește „inference" și îl voi notă cu θ. Funcția sigma este o funcție neliniară care este folosită că funcție de activare a neuronilor în rețelele neuronale artificiale. Această funcție de activare simulează comportamentul natural al neuronilor, care urmează principiul tot sau nimic (all or none)[23]. Pentru a determina parametrii θ avem nevoie să antrenăm rețeaua neuronală artificială. În procesul de antrenare a

unui model, avem nevoie de un set de date de test compus din intrarea x și ieșirea y corespunzătoare. După ce modelul face o predicție y, dorim să evaluăm cât de precisă este aceasta în comparație cu y-ul real, iar pentru aceasta utilizăm o funcție de pierdere (loss). Funcția de pierdere măsoară diferența dintre y-ul prezis și y-ul real, iar pe parcursul antrenării modelului, aceasta ne ajută să monitorizăm performanța acestuia. Pentru a actualiza constant ponderile din rețea, folosim procesul de back-propagation. Acest proces necesită calculul gradientului funcției de pierdere față de fiecare pondere din rețea, incepand de la ultimul strat al retelei neuronale. Ajustarea ponderilor se face în funcție de rata de învățare și de gradientul funcției de pierdere în raport cu fiecare pondere din rețea.

Înainte de apariția deep learning-ului, tehnici precum Hidden Markov Models (HMMs) erau populare în industria de inteligență artificială. Modelul Markov Ascuns (HMM) este un model statistic propus pentru prima dată de Baum L.E. (Baum și Petrie, 1966) și folosește un proces Markov care conține parametri ascunși și necunoscuți. În acest model, parametrii observați sunt utilizați pentru a identifica parametrii ascunși, care sunt apoi utilizați pentru analiză ulterioară. Printr-un exemplu în speech recognition putem presupune că, cuvintele rostite sunt observate, iar modelul HMM este utilizat pentru a deduce care sunt sunetele individuale care compun cuvintele respective. Aceste sunete sunt considerate că fiind parametrii ascunși și sunt utilizați pentru a identifica cuvintele individuale rostite. Cu toate acestea, HMM-urile aveau limitări în ceea ce privește capacitatea lor de a înțelege semnificația contextului și de a procesa informații complexe.

O lucrare de referinta pentru tehnica HMM este cea a lui Corneliu Octavian Dumitru si Inge Gavat din 2006. Lucrarea pune accentul pe extragerea caracteristicilor audio cu tehnici precum PLP(Perceptual Linear Prediction), LPC(Linear predictive coding) si MFCC(Mel-frequency cepstral coefficients). Testele au fost facute pe diferite seturi de date, impartite pe seturi de date in care vorbitorii sunt de gen masculin si separat de gen feminin. Pentru extragerea caracteristicilor LPC, ratele de recunoaștere a cuvintelor obținute sunt scăzute: 63,55% antrenament și testare cu FS (Dumitru et al. [12]). Pentru extragerea caracteristicilor PLP, rezultatele obținute sunt foarte promițătoare: rate de recunoaștere a cuvintelor de aproximativ 75,78% antrenament MS(male speakers) și FS(female speakers) și testare MS (Dumitru et al. [12]). Pentru extragerea caracteristicilor MFC(Mel-frequency cepstral), au obținut cele mai bune rezultate, cum era de așteptat, având în vedere că MFCC sunt caracteristici standard în recunoașterea vocală: 90,41% antrenament MS și testare cu MS (Dumitru et al. [12]).

În anul 2018, a fost inițiat proiectul ROBIN, având o durată de trei ani și fiind coordonat de către Universitatea Politehnică din București, Facultatea de Informatică. Consorțiul ROBIN cuprinde o echipă de experți în prelucrarea limbaju-

lui natural din cadrul Institutului de Cercetare pentru Inteligență Artificială "Mihai Drăgănescu" al Academiei Române (RÂCĂI) și Universitatea Politehnică din București. De asemenea, sunt implicați specialiști în domeniul roboticii de la Universitatea Politehnică din București și Institutul de Matematică al Academiei Române, experți recunoscuți în tehnologia Internet of Things (IoT) de la Universitatea din București și Universitatea Tehnică din Cluj-Napoca, precum și specialiști în proiectarea vehiculelor autonome de la Universitatea "Dunărea de Jos" din Galați[21].

Proiectul se concentrează pe implementarea mai multor tipuri de roboți, fiecare având un scop specific. Acestea includ roboți specializați pentru asistența persoanelor cu nevoi speciale, roboți concepuți pentru interacțiunea cu clienții și roboți dezvoltați pentru vehicule. Componentele fundamentale ale acestor nevoi sunt considerate subcomponente esențiale ale proiectului[21]:

- ROBIN-Social (aimas.cs.pub.ro/robin/en/robin-social/)
- ROBIN-Car (aimas.cs.pub.ro/robin/en/robin-car/)
- ROBIN-Dialog (aimas.cs.pub.ro/robin/en/robin-dialog/)
- ROBIN-Context (aimas.cs.pub.ro/robin/en/robin-context/)
- ROBIN-Cloud (aimas.cs.pub.ro/robin/en/robin-cloud/)

Experimentul realizat în cadrul acestei lucrări de licență se bazează pe un studiu efectuat de Tufiș și colaboratorii săi [4]. În lucrarea lor, este prezentată o rețea neuronală care utilizează, de asemenea, un model de limbaj statistic pentru a realiza recunoașterea vorbirii.

Scopul acestui proiect este de a facilita interacțiunea oamenilor cu roboții prin intermediul recunoașterii vocale în limba română.(am reprodus din introducerea lor). Proiectul a fost antrenat 230 de ore și a reușit o eroare per cuvânt de 9.91% și eroare per caracter de 2.81%, cu cele mai celebre seturi de date din ultimii ani CoRoLa, OSCAR (Tufis et al. [4]).

Lucrarea expune de asemenea extragerea feature-urilor din fișierele audio, acesteaua au fost împărțite în ferestre de 20 ms și a calculat coeficienții cepstrali de frecvență Mel (MFCC) (Tufiș et al. [4]).

În domeniul recunoașterii vocale, MFC (Mel-frequency cepstral) este cea mai întâlnită opțiune pentru extragerea caracteristicilor, este mai adaptată la auzul uman. Cu toate acestea, alte tehnici precum PLP (Perceptual Linear Prediction) și LPC (Linear Predictive Coding) sunt, de asemenea, utilizate în recunoașterea vocală pentru a extrage caracteristici spectrale și temporale ale semnalului vocal.

Un punct forțe al MFCC-urilor este partea de preprocesare numită preemphasis. Preemphasis crește cantitatea de energie în frecvențele înalte. Pentru segmentele

vocale precum vocalele, există mai multă energie la frecvențele inferioare decât la frecvențele mai înalte.

Rețeaua neuronală este compusă din 2 straturi convoluționale, cu 32 de filtre: primul cu o dimensiune a kernelului de (41, 11) și un stride de (2, 2), iar al doilea cu o dimensiune a kernelului de (21, 11) și un stride de (2, 1) (Tufiș et al. [4]).

Chiar dacă modelul a fost antrenat pe multe ore dacă ne raportăm la câte seturi de date în limba română există până în prezent, acesta lasă loc de îmbunătări, astfel apare lucrarea de Tufis et al. [3], de aceeași autori. Aceasta lucrare se bazează în principal pe reducerea timpului în care modelul reușește să facă conversia audio text. Folosind modelul ASR și modelul lingvistic combinat(cu parametrii optimi), rezultatul general a fost de 9,91% WER pe setul de testare, îmbunătățind WER al sistemului ASR brut cu 5,66%. (Tufis et al. [3])

Sistemul de recunoaștere automată a vorbirii (ASR) pentru limba română, stă la baza arhitecturii DeepSpeech2. Pentru calculul caracteristicilor audio (feature-urilor), a fost realizat un proces de prelucrare a datelor, în care coeficienții cepstrali de frecvență Mel (MFCC) au fost calculați pe ferestre audio de 20 ms. Spectrograma rezultată a fost utilizată ca intrare într-un model de rețea neuronală profundă. Modelul folosit a constat în 8 straturi neuronale, incluzând 2 straturi convoluționale 2D, 4 straturi BiLSTM(Bidirectional Long Short-Term Memory) cu memorie bidirecțională pe termen scurt, 1 strat de convoluție de tip lookahead și un strat final complet conectat (FC) (Tufis et al. [3]).

În scopul optimizării instruirii modelului, a fost aplicată normalizarea pe batch-uri(BN) după fiecare strat, cu excepția ultimului strat. De asemenea, s-a folosit funcția de activare HardTanh înainte de fiecare strat de normalizare pe batch-uri (BN), ca o alternativă eficientă din punct de vedere computațional la funcția tanh.

Proiectul abordează metode de îmbunătățire a vitezei și de perfecționare a traducerilor, luând în considerare particularitățile unice ale limbii române care pot reprezenta obstacole pentru modelul de recunoaștere vocală. Sistemul dispune de două endpoint-uri REST, unul /transcribe, responsabil de convertirea audio-ului în text, iar celălalt /correct, primește că input text-ul și îl corectează conform caracteristicilor limbii române.

Georgescu et al. [13] au investigat utilizarea rețelelor neuronale în sistemele de recunoaștere automată a vorbirii (ASR) în limba română, folosind platforma de dezvoltare Kaldi3. În studiul lor, ei au evaluat performanța modelului lor pe două corpusuri: RSC-eval și SSC-eval, obținând rezultate ale Ratei de Erori a Cuvintelor (WER) de 2,79%, respectiv 16,63%. Deși cele două corpusuri reprezintă tipuri diferite de vorbire (citită și spontană), pentru implementarea noastră ne dorim un sistem ASR general, care să funcționeze independent de tipul de vorbire. Prin urmare, în scopul comparației, am luat în considerare media WER a ambelor corpusuri

de evaluare, rezultând o valoare medie a WER de 9,71% (Tufis et al. [3]).

O altă abordare de acest gen este a lui Tufiș et al. [21] în care este folosit toolkit-ul Kaldi. Proiectul este concentrat pe dezolvatarea robotului Pepper ca fiind un partener pentru dialogurile cu oamenii, in diverse „microworlds". Un „microworld" sau o micro-lume trebuie înțeleasă ca reprezentând o zonă limitată, din care fac parte o anumite categorie de cuvinte cheie, avand o anumita tematica. Astfel, dialogurile om-robot sunt dialoguri situate (bazate pe o micro-lume) (Tufiș et a. [21]).

Proiectul are 4 componente: un modul pentru recunoasterea vocii(ASR), o unitate de procesare a limbajului natural(NLG), un dialog manager(DM) si un sistem de convertire a textului in audio. Sistemul ASR a obtinut o eroare modesta per cuvânt de 25%. Eroarea este într-adevăr mare, poate fi datorată setului de date de antrenament și de natură să, dacă este citit sau vorbit spontan(Tufiș et a. [21]).

3.2 Descrierea instrumentelor disponibile pentru antrenarea modelelor de recunoaștere vocală în limba română

O îmbunătățire importantă cu care pot să-mi pun amprenta acestui proiect este antrenarea cu ajutorul setului de date Common Voice. Common Voice este o platforma creată de Mozilla, prin care se creaza seturi de date vocale disponibil public, alimentat de vocile colaboratorilor voluntari din întreaga lume. Platforma permite utilizatorilor să doneze înregistrări cu vocea lor și să valideze alte înregistrări. Persoanele care doresc să creeze aplicații vocale pot folosi setul de date pentru a antrena modele de învățare automată.[22]. Common Voice deține seturi de înregistrări pentru limba română in jur de 295 de ore inregistrate si 132 de ore validate.

Modelul ASR al proiectului ROBIN a fost antrenat cu cele mai celebre dată seturi de la aceea vreme, printre care si CoRoLa. Scopul creării acestui corpus este de a oferi o imagine obiectivă a limbii române actuale, atât în formă scrisă, cât și în cea vorbită, prin includerea unor texte diverse, care cuprind perioada de la 1989 până în prezent. [9]. Corpusul în prezent este în jur de 300 de ore de înregistrări, înregistrări profesionale din diverse surse, posturi de radio, studiouri de înregistrare. Printre celelalte corpusuri de date amintim: OSCAR, Romanian Digits (RoDigits), Romanian Common Voice (RSV), Romanian Speech Synthesis (RSS), Read Speech Corpus (RSC) (Tufiș et a. [4]).

Spre deosebire de celelalte OSCAR este un corpus uriaș multilingv cu sursă deschisă care a fost obținut prin filtrarea de Common Crawl și prin gruparea textului rezultat după limbă. Versiunea în limba română conține aproximativ 11 GB de propoziții amestecate deduplicate. (Tufiș et a. [4])

Romanian Speech Synthesis (RSS), creat pentru sinteză vocii și conține 4 ore de

înregistrări audio realizate de o singură vorbitoare de sex feminin, folosind mai multe microfoane. Vorbitoarea a citit 4000 de propoziții extrase din române, articole de ziare selectate pentru acoperirea di-foanelor și povești. Corpusul RSS a fost extins și cu peste 1700 de exprimări ale altor două vorbitoare de sex feminin, ajungând astfel la o durată totală de 5,5 ore de înregistrări audio. (Tufiș et a. [4])

Corpusul RoDigits conține 37,5 ore de înregistrări audio cu cifre vorbite de la 154 de vorbitori, cu vârste cuprinse între 20 și 45 de ani. Fiecare vorbitor a înregistrat 100 de clipuri cu 12 cifre românești generate aleatoriu, iar după validarea semi-automată, corpusul final a inclus 15.389 de fișiere audio. (Tufiș et al. [4])

Read Speech Corpus(RSC) este un set de date ce conține vorbire citită, colectat de către Laboratorul de Cercetare a Vorbirii și Dialogului. Aceste înregistrări au fost realizate în diverse condiții, folosind diferite tipuri de microfoane și sisteme de înregistrare audio, prin intermediul unei aplicații online dezvoltate de același grup de cercetare. Participanții la înregistrări au fost în principal studenți și cadre didactice din cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației de la Universitatea "Politehnică" din București. Corpusul este constituit dintr-o colecție de 136.120 de înregistrări audio, obținute de la 164 de vorbitori nativi ai limbii române Georgescu et al. [12]. Fiecare înregistrare conține fraze și cuvinte selectate din literatură, articole de știri online și vocabular specific limbii române[**?**].

Scopul acestui corpus este de a aduce o contribuție semnificativă în domeniul recunoașterii vocale în limba română, prin furnizarea unui set de date, având în vedere că multe dintre seturile de date existente în România nu sunt disponibile publicului larg.

Raportându-ne la lucrarea lui Tufiș et al. [4] Common Voice avea în jur de 7 ore de audio transcris înregistrat de 79 de difuzoare în limba română, comparativ cu cele 200 de ore pe care le deține în prezent.

Cele mai importante seturi de date în limba română 3.1

Tabela 3.1: Dataseturi România (Georgescu, et al.[14])

Nume	**Tipul vorbirii**	**Domeniu**	**Ore**	**Disponibilitate**
RASC	Read	Wikipedia Articles	4.8	Public
RoDigits	Read Spoken	Digits	38.0	Public
RO-GRID	Read	General	6.6	Public
IIT	Read	Literature	0.8	Non-Public
N/A	Read	Eurom-1 Adapted Transla-tions	10.0	Non-Public
N/A	Spont.	Internet, TV	4.0	Non-Public
RSS	Spont.	Internet, TV	4.0	Public
SWARA	Read	Newspapers	21.0	Public
MaSS	Read	Bible	23.1	Public
N/A	Spont.	Broadcast News	31.0	Non-Public
N/A	Spont.	Banking	40.0	Non-Public
SSC-train1	Spont.	Radio and TV	27.5	Non-Public
SSC-train2	Spont.	Radio and TV	103.0	Non-Public
SSC-train3	Spont.	Radio and TV	49.5	Non-Public
SSC-train4	Spont.	Radio and TV	280.0	Non-Public
CoRoLa	N/A	Various Sources	152.0	Cvasi-Public
Common Voice	N/A	Various Sources	295.0	Public

3.3 Descriere a chatbotilor relevanti in prezent

În domeniul modelelor de limbaj natural, există câteva modele extrem de semnificative dezvoltate de companii precum OpenAI și Google. Printre acestea se numără GPT de la OpenAI, Bert de la Google, LaMDA de la Google, PaLM de la Google și

LLaMA de la Meta AI.

Deși Google a dezvoltat mai multe variante și implementări ale acestor modele, în opinia mea, cel mai relevant model este GPT de la OpenAI. În prezent, versiunea stabilă și actuală a acestui model este GPT-3.5. Este disponibilă de asemenea varianta de GPT-4, doar pentru utilizatorii care au cont premium.

Modelul GPT-4 este caracterizat de un timp de răspuns mai rapid la întrebări și vine cu o caracteristică nouă și importantă, respectiv capacitatea de a se conecta la pluginuri. Unul dintre exemplele notabile de utilizare a acestor pluginuri este integrarea cu un plugin care efectuează căutări pe internet și oferă răspunsuri bazate pe rezultatele obținute. De asemenea, GPT-4 poate fi utilizat împreună cu pluginul Wolfram Alpha, transformând astfel modelul de limbaj într-un calculator avansat.

Modelul GPT-3.5 reprezintă o versiune avansată a modelului GPT-3, fiind succesorul acestuia. Ambele serii de modele aduc îmbunătățiri și funcționalități remarcabile. În ceea ce privește GPT-3.5, amintim urmatoarele modele: code-davinci-002, text-davinci-002, text-davinci-003 și gpt-3.5-turbo. Pe de altă parte, GPT-3 include versiunile davinci și text-davinci-001 conform lui Junjie Ye et al.[24].

- davinci: baza modelelor din seria GPT-3, care poate înțelege și genera limbaj natural cu o calitate superioară.
- text-davinci-001: un model InstructGPT(genereaza instrucțiuni și răspunsuri într-o varietate de domenii și contexte. Acest model este antrenat pe o vastă colecție de texte care cuprind instrucțiuni și informații practice provenite din diverse surse, inclusiv manuale, tutoriale și ghiduri.) bazat pe davinci (Junjie Ye et al.[24]).
- code-davinci-002: cel mai performant model Codex, care este un descendent al GPT-3 și care stă la baza modelului modelele din seria GPT-3.5, cu date de instruire care conțin atât limbaj natural, cât și miliarde de linii de cod sursă din surse disponibile public, inclusiv coduri din depozitele publice GitHub. (Junjie Ye et al.[24]).
- text-davinci-002: un model InstructGPT bazat pe code-davinci-002. (Junjie Ye et al. [24]).
- text-davinci-003: o versiune îmbunătățită a text-davinci-002, dar antrenată cu metoda Proximal Policy. (Junjie Ye et al.[24]).
- gpt-3.5-turbo: cel mai capabil model GPT-3.5 și optimizat pentru chat la 1/10 din costul textdavinci-003. (Junjie Ye et al.[24]).

În ceea ce privește celelalte modele de limbaj disponibile, o abordare captivantă este Alpaca. Alpaca reprezintă un model avansat de limbaj natural, dezvoltat de la

baza LLaMa, optimizat pentru performanță. Ceea ce îl face distinctiv de Alpaca este dimensiunea sa redusă și eficiența sa remarcabilă. Cu doar 7 miliarde de parametri, Alpaca poate fi comparat cu modelul text-davinci-003 de la OpenAI. Un avantaj semnificativ al acestui model este că poate fi utilizat local, fără a necesita o conexiune la internet[20].

Pentru a utiliza Alpaca local, este necesar un calculator decent, chiar și fără o placă grafică, deși timpul de răspuns al modelului poate fi mai lent. De asemenea, este necesară o capacitate de stocare liberă de 8 GB.

Cu toate acestea, în proiectul meu, am ales să nu utilizez Alpaca, deși este o opțiune convenabilă pentru momentele in care nu aplicatia nu este conectata la internet. Motivul constă în faptul că acest model nu este deloc fiabil pe dispozitivele mobile datorită modului de instalare și cerințelor de putere hardware pe care le necesită.

Capitolul 4

Tehnologii folosite în dezolvarea aplicației

4.1 Flask

Flask este un framework pentru dezvoltarea aplicațiilor web în Python, dezvoltat pentru a fi simplu de învățat și utilizat. Acesta este conceput pentru a oferi un set de funcționalități de bază, în rest fiind personalizabil. Printre funcționalitățile de bază amintim: rutarea și gestionare cererilor (facilitează definirea rutelor URL și gestionarea cererilor HTTP asociate acestora), integrarea cu baza de date (nu impune o bază de date, dar oferă suport pentru acestea). Fiind un microframework, dar acest lucru nu înseamnă că întreaga aplicație trebuie să fie inclusă într-un singur fișier Python.

Micro înseamnă că framework-ul Flask este simplu, dar extensibil. Poți lua toate deciziile: ce bază de date să folosești, dacă dorești un ORM, Flask nu decide pentru tine[6].

Flask este unul dintre cele mai populare framework-uri web, ceea ce înseamnă că este actualizat și modern. Poți extinde ușor funcționalitatea sa. Poți scala aplicația pentru aplicații complexe. Am optat pentru utilizarea framework-ului Flask în locul Django, deoarece acesta este un framework independent, creat special pentru sisteme cu o complexitate mai redusă. În cazul meu, am ales Flask pentru a dezvolta funcționalitatea de recunoaștere vocală și pentru a comunica cu chatbotul utilizând biblioteca LangChain.

4.2 LangChain

LangChain reprezintă o bibliotecă care facilitează lucrul cu un model lingvistic extins(large language model). Un large language model e un model care are rolul de a

prezice ce caracter/cuvant apare dupa o anumita secvență de a evalua probabilitatea apariției a unui șir de cuvinte sau caractere într-un anumit limbaj.

Principiile fundamentale pe care se bazează LangChain, "Data-aware" și "Agentic"[16], permit integrarea large language modelului cu diverse surse de date, cum ar fi motoarele de căutare, fișiere externe utilizate ca intrare sau servicii de stocare, cum ar fi Google Drive. Astfel, LangChain facilitează colaborarea cu aceste surse de date externe, extinzând astfel funcționalitatea acestuia în cadrul aplicațiilor. Unul dintre elementele de baza ale acestei librarii este interfața standard pentru modele. Pentru a crea o instanță a unui model, putem seta modelul folosit, parametrul temperature, care seteaza cat de unic sa fie raspunsul generat, dar si numarul maxim de tokenuri folosite pentru a genera răspunsul.

```
llm = OpenAI(
    model_name='text-davinci-003',
    temperature=0,
    max_tokens=512
)
```

Modelul folosit de mine "text-davinci-003" poate stoca pana la 4097 de tokenuri. Acesta poate îndeplini orice sarcină lingvistică cu o calitate mai bună, cu o producție mai lungă și cu o respectare consecventă a instrucțiunilor decât modelele Curie, Babbage sau Ada. De asemenea, suportă inserarea de completări în text[18].

Memoria pe care Langchain o cuprinde este o caracteristică esențială in construcția unui chatbot. LangChain oferă mai multe tipuri de memorie la dispoziție, printre care se află: Conversation Buffer Memory, Conversation Summary Memory, Conversation Summary Buffer Memory, Conversation Entity Memory, Conversation Knowledge Graph Memory. Fiecare tip de memorie are avantaje si dezavantaje, pe care le voi prezenta în urmatoarea secțiune.

4.3 Comparație între memoriile oferite de LangChain

Conversation Buffer Memory, permite stocarea tututor mesajelor, dintr-o conversație și apoi extragerea mesajele într-o variabilă. Conversation Summary Memory, creează un rezumat al conversației de-a lungul timpului. Acest lucru poate fi util pentru a condensa informațiile din conversație. Conversation Buffer Window Memory păstrează o listă a ultimelor k interacțiuni(k fiind un parametru setat la inițializare). Acest lucru poate fi util pentru a păstra o fereastră glisantă a celor mai recente interacțiuni, astfel încât memoria să nu devină prea mare.

Conversation Summary Buffer Memory este similar cu cel anterior, doar ca nu mai există un k care să decidă câte din ultimele mesaje sunt salvate, avem un para-

metru ”max_token_limit”, iar când lista de mesaje depășește acestă limită, se va face un rezumat al ultimelor mesaje.

Entity Memory extrage cu ajutorul large language model-ului caracteristici cheie din mesaje și le salveaza. De exemplu din mesajul “Deven & Sam lucrează la un proiect”, entitatea “Sam” va avea asociată acțiunea “lucrează la un proiect cu Deven”

Conversation Knowledge Graph Memory utilizează un grafic de cunoștințe pentru a recrea memoria. De exemplu pentru mesajul “Culoarea preferata a lui Sam este rosu”, graful va arata in felul urmator:“subject”=”Sam”, “predicate”=”culoarea preferata”, “object”=”red”.

Avantaje și dezavantaje pentru Conversation Buffer Memory:

- O abordare simplă și inuitivă
- Stochează maximul de informație
- Stochează toate tokenurile, dar există limitări are llm-ului în ceea ce privește numărul de tokenuri, generează costuri mai mari
- Timp îndelungat pentru requesturi

Avantaje și dezavantaje pentru Conversation Summary Memory:

- Necesită mai puține tokenuri pentru conversații lungi
- Conversațiile pot fi mai lungi
- Pentru conversațiile scurte reține prea puține informații

Avantaje și dezavantaje pentru Conversation Summary Buffer Memory:

- Componenta care se ocupă de rezumat își amintește interacțiunile la distanță
- Reține maximul despre conversațiile cele mai recente
- Conversațiile pot fi mai lungi
- Poate fi prea complex, deși calitatea lui depinde de o mulțime de factori (calitatea rezumatului, numărul de tokenuri sau k care uneori e setat prea mic, altădată prea mare)
- Foloseste multe tokenuri la conversațiile scurte și pentru realizarea rezumatului

Pentru a putea reprezenta cât mai bine diferențele dintre tipurile de memorii LangChain am preluat acest grafic[7]:

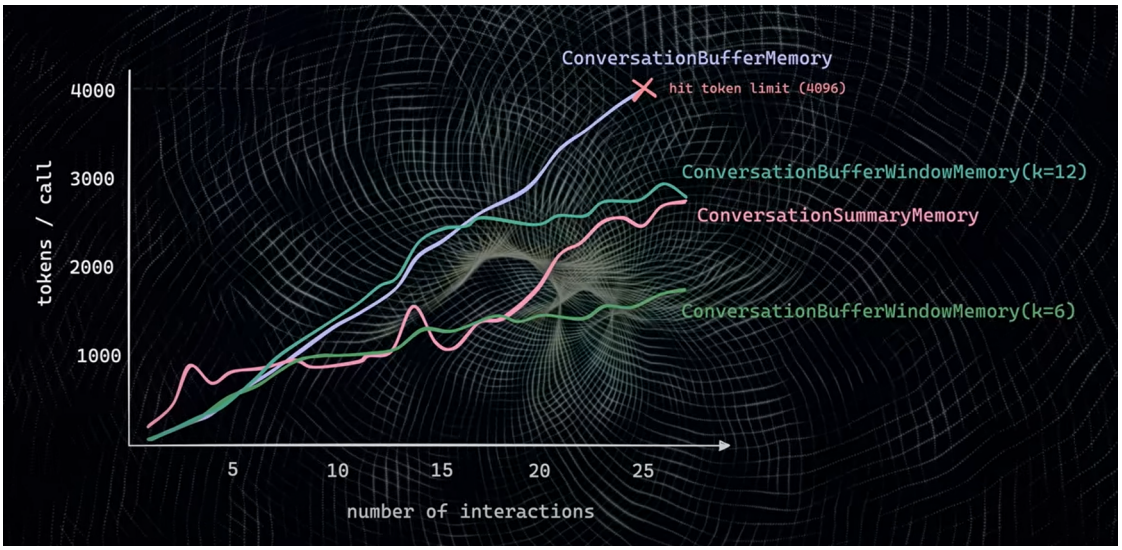


Figura 4.1: Grafic în funcție de numărul de mesaje trimise memoriei și numărul de tokenuri consumate

4.4 Kotlin și Jetpack Compose

Kotlin este un limbaj de programare static, orientat pe obiecte, dezvoltat inițial de JetBrains (o companie de software) și lansat în 2011. Acesta s-a inspirat din mai multe limbaje de programare, inclusiv (dar nu numai) Java, Scala, C# și Groovy. Una dintre ideile principale din spatele Kotlin este aceea de a fi pragmatic, adică de a fi un limbaj de programare util pentru dezvoltarea de zi cu zi, care ajută utilizatorii să își facă treaba prin intermediul funcțiilor și instrumentelor sale. Astfel, o mulțime de decizii de proiectare au fost și încă sunt influențate de cât de benefice sunt aceste decizii pentru utilizatorii Kotlin[17]. În ultimii ani, acest limbaj a castigat o popularitate semnificativă pentru dezvoltarea de aplicații pe dispozitive mobile cu Android. Kotlin a câștigat popularitate în rândul dezvoltatorilor Android datorită compatibilității sale cu Java, ceea ce a permis adoptarea ușoară a limbajului în ecosistemul existent. Jetpack Compose este un toolkit modern care funcționează împreuna cu Kotlin pentru crearea interfeței cu utilizatorul[10]. Acesta simplifică și accelerează dezvoltarea interfețelor utilizator pe Android, dând viață aplicațiilor cu mai puțin cod, instrumente puternice și API-uri intuitive[11]. Modul ușor de a folosi Jetpack Compose în proiecte, a constituit un motiv solid pentru folosirea acestuia. Astfel, putem crea functii adnotate cu cuvantul cheie "@Composable", iar acestea vor fi afișate pe ecran.

Listing 4.1: Simple code listing.

```
@Composable
fun MyScreen() {
```

```
    MaterialTheme {
        Column {
            Text(text = "Salut, Jetpack Compose!")
            Button(onClick = { /* Ac iunea dvs. */ }) {
                Text(text = "Apas -m !")
            }
        }
    }
}
```

În mod tradițional, dezvoltarea interfețelor utilizator pentru aplicațiile Android presupunea lucrul cu fișiere XML și gestionarea componentelor UI în mod programatic. Această abordare a dus adesea la un cod complex și redundant, făcând din dezvoltarea interfețelor pentru utilizator o sarcină consumatoare de timp și predispusă la erori.

4.5 Dagger Hilt

Dependency injection este un principiu semnificativ în dezvoltarea aplicațiilor. Aceasta ajută la decuplarea componentelor, promovează reutilizarea și facilitează dezvoltarea modulară. Dagger-Hilt este una din librăriile populare pentru gestionarea dependențelor în aplicațiile Android. Dagger Hilt simplifică procesul de injectare a dependențelor prin reducerea codului și prin furnizarea unei abordări standardizate a gestionării dependențelor. Acesta este construit peste framework-ul Dagger, cunoscut pentru eficiența și performanța sa. Cu Hilt, dezvoltatorii pot profita de avantajele Dagger, bucurându-se în același timp de o experiență mai raționalizată și mai intuitivă, special concepută pentru dezvoltarea Android. Un exemplu de utilizare, pentru ViewModel:

Listing 4.2: Simple code listing.

```
@HiltViewModel
class ChatViewModel @Inject constructor(
    private val messageRepository: MessageRepository,
    savedStateHandle: SavedStateHandle
) : ViewModel() { /* Codul din ViewModel */ }
```

4.6 Java si Spring Boot

Spring Boot este un framework Java open-source care a revoluționat modul în care dezvoltăm și implementăm aplicații Java. Spring Boot se concentrează pe simplificarea dezvoltării, configurării și implementării aplicațiilor Java, oferind un cadru robust și eficient pentru construirea de aplicații de înaltă calitate. Dezvoltatorii din întreaga lume își încep călătoria de codare învățând Java. Flexibil și ușor de utilizat, Java este preferatul dezvoltatorilor pentru o varietate de aplicații - totul, de la aplicații de social media, web și jocuri până la aplicații de rețea și aplicații de întreprindere[5]. În acest proiect, modulul Spring Boot funcționează ca un gateway, prin intermediul căruia requesturile sunt direcționate în mod autorizat către modulul Flask, reprezentând componenta de inteligență artificială. Răspunsul este apoi prelucrat de către Spring Boot, care de asemenea se ocupă de salvarea în baza de date a entităților și procesarea acestora.

4.7 Pinecone

Embeddings sunt reprezentări numerice generate de modelele de inteligență artificială, care capturează caracteristicile și relațiile complexe ale datelor. Deoarece aceste embeddings pot avea o dimensiune mare și sunt dificil de gestionat, avem nevoie de o soluție specializată pentru stocarea și interogarea lor[19].

Pinecone este o bază de date vectorială specializată, concepută pentru a gestiona eficient acest tip de date. O bază de date vectorială, cum este Pinecone, oferă funcționalități optimizate pentru stocarea și interogarea încorporărilor vectoriale. Aceasta combină avantajele unei baze de date tradiționale, cum ar fi scalabilitatea și capacitățile de interogare complexe, cu specializarea în gestionarea încorporărilor vectoriale.

Prin utilizarea Pinecone și altor baze de date vectoriale, putem beneficia de o performanță superioară în stocarea și căutarea eficientă a acestor embeddings, facilitând dezvoltarea de aplicații AI și îmbunătățind rezultatele și experiențele utilizatorilor.

4.8 Librăria SpeechRecognition

În cadrul aplicației mele, am optat pentru utilizarea bibliotecii SpeechRecognition[25] pentru a converti înregistrările audio în text. Această bibliotecă utilizează API-ul Google Cloud pentru transformarea audio-ului în text.

Am luat această decizia de a folosi în proiectul meu această librărie în locul proiectului RobinASR, pe care îl și antrenez, fiindcă funcționalitatea oferită de Spee-

chRecognition furnizează transcrieri mai precise și într-un timp mai scurt. De asemenea, am avut în vedere faptul că am nevoie să transform în text înregistrări audio scurte, cu o durată de până la 10 secunde, pentru a le utiliza în mesaje. Astfel, timpul necesar pentru transformarea acestor înregistrări este extrem de important.

Prin alegerea SpeechRecognition, am obținut rezultatele dorite într-un mod eficient și rapid, asigurându-ne că transformarea în text a înregistrărilor se realizează într-un timp optim, în concordanță cu necesitățile aplicației.

Capitolul 5

Dezoltarea aplicației

5.1 Introducere

Aplicația are o arhitectură client-server, în care clientul este o aplicație Android, iar serverul este compus din două module: un modul Spring Boot și un modul Flask. Pentru a implementa funcționalitatea de inteligență artificială în aplicație, care poate fi accesată doar cu ajutorul librăriilor Python, a fost necesară împărțirea codului între cele două module. În urmatoarele secțiuni voi prezenta câteva diagrame importante pentru a înțelege fluxul aplicației, cele cele doua module backend și aplicația Android.

5.2 Cazuri de utilizare

Diagrama cazurilor de utilizare prezintă funcționalitățile principale ale aplicației și modul în care utilizatorii interacționează cu sistemul. Această diagramă oferă o imagine de ansamblu a interacțiunilor și scenariilor esențiale pe care le oferă aplicația.

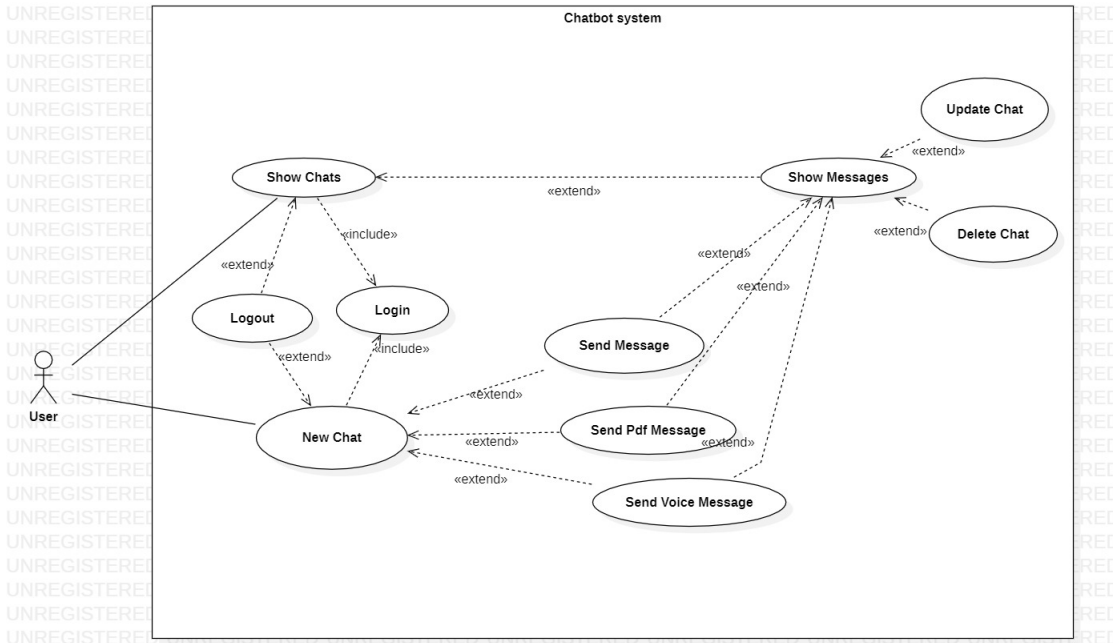


Figura 5.1: Cazuri de utilizare

Tabela 5.1: Descriere caz de utilizare Login

Nume	Login		
Actori	Utilizatorul		
Descriere	Utilizatorul primește acces la aplicație		
Pre-condiții			
Post-condiții	Utilizatorul primește acces la aplicație		
Flux	Nr.	Acțiune utilizator	Răspunsul sistemului
	1	Utilizatorul intră în aplicație	
	2		Încărcarea interfeței de login
	3	Utilizatorul introduce email-ul și parola și apasă butonul ”Login”	
	4		Serverul verifică datele introduse și deschide fereastra principală a aplicatiei, adică o nouă conversație.

Tabela 5.2: Descriere caz de utilizare Send Message

Nume	Send Message		
Actori	Utilizatorul		
Descriere	Utilizatorul trimite un mesaj chatbotului		
Pre-condiții	Utilizatorul trebuie să fie logat în aplicație		
Post-condiții	Utilizatorul primește răspuns la mesajul trimis		
Flux	Nr.	Acțiune utilizator	Răspunsul sistemului
	1	Navigarea către o conversație existentă sau o conversație nouă	
	2		Încarcarea mesajelor existente conversației
	3	Utilizatorul introduce un mesaj și apasă pe butonul "Send"	
	4		Serverul primește mesajul, îl procesează și îi trimite un răspuns utilizatorului

Tabela 5.3: Descriere caz de utilizare send PDF message

Nume	Send PDF Message		
Actori	User		
Descriere	Utilizatorul trimite un mesaj cu un fișier PDF chatbotului		
Pre-condiții	Utilizatorul trebuie să fie logat în aplicație		
Post-condiții	Utilizatorul primește răspuns la mesajul trimis		
Flux	Nr.	Acțiune utilizator	Răspunsul sistemului
	1	Navigarea către o conversație existentă marcat cu simbolul de PDF	
	2		Încărcarea mesajelor existente conversației
	3	Utilizatorul introduce un mesaj si apasă pe butonul "Send"	
	4		Serverul primește mesajul, îl procesează și îi trimite un răspuns utilizatorului
	Nr.	Acțiune utilizator	Răspunsul sistemului
Flux alternativ			

	1	Navigarea catre o conversație nouă și apăsarea butonului PDF pentru a selecta un fișier din file explorer	
	2		Sistemul stocheaza fișierul selectat
	3	Utilizatorul introduce un mesaj și apasă butonul "Send"	
	4		Serverul primește mesajul, îl procesează și îi trimite un răspuns utilizatorului

Tabela 5.4: Descriere caz de utilizare Send Voice Message

Nume	Send Voice Message		
Actori	Utilizatorul		
Descriere	Utilizatorul trimite un mesaj vocal chatbotului		
Pre-condiții	Utilizatorul trebuie să fie logat în aplicație		
Post-condiții	Utilizatorul primește un text corespunzator cu ce a inregistrat		
Flux	Nr.	Acțiune utilizator	Răspunsul sistemului
	1	Navigarea către o conversație existentă sau o conversație nouă	
	2		Încărcarea mesajelor existente conversației
	3	Utilizatorul ține apăsat butonul "Microphone" în timp ce rostește mesajul	
	4		Serverul primește mesajul, îl transformă în text și îl trimite înapoi clientului

5.3 Aplicația Spring Boot

Endpointurile puse la dispoziție sunt cele responsabile de autentificare, de procesarea mesajelor , a conversațiilor și de recunoașterea vocii. Mai exact clasa MessageController, un controller REST gestionează cererile legate de message din aplicație:

metodele de saveMessage și de saveMessageWithPdf, care primește ca parametru un MultipartFile, reprezentând PDF-ul la baza căruia este conversația.

Listing 5.1: Exemplu de controller

```
@RestController
@RequestMapping("/messages")
@RequiredArgsConstructor
public class MessageController {
    private final MessageService messageService;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public MessageResponse saveMessage(@RequestBody
       MessageRequest messageRequest) {
        return messageService.save(messageRequest);
    }

    @GetMapping("/{chatId}/{page}")
    public List<MessageResponse> getMessages(@PathVariable
       Integer chatId, @PathVariable Integer page) {
        return messageService.getMessagesByChat(chatId, page)
           ;
    }

    @PostMapping("/pdf")
    public MessageResponse saveMessageWithPdf(@RequestPart
       MultipartFile file, @RequestPart String text) {
        return messageService.saveMessageWithPdf(new
           MessagePdfRequest(file, text));
    }
}
```

Componenta de servicii a aplicatiei este în mare parte controlată de clasa MessageServiceImpl, care se ocupă de crearea de mesaje și conversații, apelul la repository, dar și la clasa de PythonServiceImpl. Aceasta se ocupă de apelul la modulul Flask al aplicației.

Modalitatea de comunicare între module este prin apeluri HTTP, cu ajutorul clasei RestTemplate. Aceasta este un client HTTP sincron utilizat pentru a trimite cereri HTTP către servicii web externe și a primi răspunsuri de la acestea.

Listing 5.2: Exemplu de apel HTTP cu RestTemplate

```
ResponseEntity<MessageResponse> response = restTemplate.
   postForEntity(
        pdfMessageUrl,
        requestEntity,
```

```
        MessageResponse.class
);
```

În ceea ce privește partea de repository, utilizez pachetul Spring JPA și interfața JpaRepository pentru a defini interfețele asociate obiectelor din aplicație. În exemplul dat, am definit o interfață care utilizează paginare pentru a prelua mesajele în funcție de conversația din care fac parte. Pentru a realiza acest lucru, am implementat interfața PagingAndSortingRepository.

Listing 5.3: Exemplu de repository

```
public interface MessageRepository extends JpaRepository<
   Message, Integer>, PagingAndSortingRepository<Message,
   Integer> {
    List<Message> findByChatId(Integer chatId);

    Page<Message> findAllByChatIdOrderByIdDesc(Integer chatId
       , Pageable pageable);

    void deleteByChatId(Integer chatId);
}
```

Pentru gestionarea bazei de date, folosesc PostgreSQL și Flyway. PostgreSQL este un sistem de gestiune a bazelor de date relaționale puternic și fiabil. Flyway este o unealtă specializată în gestionarea migrațiilor bazelor de date. În esență, migrațiile sunt schimbările structurale pe care dorim să le aplicăm într-o bază de date pe măsură ce dezvoltăm aplicația.

În timpul dezvoltării, este posibil să adăugăm sau să modificăm tabele, coloane, indici sau alte aspecte ale bazei de date. Flyway ne ajută să gestionăm aceste modificări prin intermediul fișierelor de migrație.

Un fișier de migrație conține instrucțiuni SQL care descriu schimbările pe care dorim să le aplicăm în baza de date. Flyway urmărește aceste fișiere și le aplică în ordine pe măsură ce versiunea aplicației noastre evoluează. De asemenea, Flyway ține evidența migrațiilor care au fost deja aplicate în baza de date, astfel încât să evităm duplicarea modificărilor.

Pentru securitatea aplicatiei am utilizat librăria Spring Security. Astfel, fiecare cerere HTTP, va fi interceptată de filtrul JwtAuthenticationFilter. Acest filtru are rolul de a verifica prezența antetului de autorizare (authorization header). Dacă antetul este prezent, JwtAuthenticationFilter extrage și validează token-ul JWT (JSON Web Token), obținând astfel informațiile despre utilizator și drepturile acestuia. Utilizând aceste informații, filtrul încarcă contextul de securitate (security context) cu numele de utilizator al acestuia.

Prin actualizarea corectă a contextului de securitate, sunt în măsură să autorizez cererile primite în funcție de configurarea definită în clasa SecurityConfig. Această clasă utilizează funcțiile requestMatchers(), authenticated() și permitAll() pentru a specifica regulile de acces în funcție de necesitățile aplicației. Astfel, fiecare cerere va fi verificată pentru a se asigura că utilizatorii autentificați au permisiunile corespunzătoare pentru a accesa resursele respective, în timp ce cererile neautentificate vor avea acces liber, conform setărilor de securitate stabilite.

Pentru a realiza conexiunea între utilizatorii din baza de date și biblioteca Spring Security, am utilizat două componente importante: AuthenticationManager și AuthenticationProvider. Prin intermediul bean-urilor de AuthenticationManager și AuthenticationProvider, am configurat sistemul pentru a valida autentificările și pentru a asigura coerența și consistența procesului de autentificare.

Listing 5.4: extragerea headerului Authorization și încărcarea SecurityContext

```
@Override
protected void doFilterInternal(@NonNull
   HttpServletRequest request, @NonNull
   HttpServletResponse response, @NonNull FilterChain
   filterChain) throws ServletException, IOException {
    final String authHeader = request.getHeader("
       Authorization");
    final String jwt;
    final String userEmail;

    if (authHeader == null || !authHeader.startsWith("
       Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    jwt = authHeader.substring(7);
    userEmail = jwtServiceImpl.extractUsername(jwt);

    if (userEmail != null && SecurityContextHolder.
       getContext().getAuthentication() == null) {
        UserDetails userDetails = userDetailsService.
           loadUserByUsername(userEmail);
        if (jwtServiceImpl.isTokenValid(jwt, userDetails)
           ) {
```

```
                UsernamePasswordAuthenticationToken
                    authenticationToken = new
                    UsernamePasswordAuthenticationToken(
                            userDetails,
                            null,
                            userDetails.getAuthorities()
                );

                authenticationToken.setDetails(new
                    WebAuthenticationDetailsSource().
                    buildDetails(request));
                SecurityContextHolder.getContext().
                    setAuthentication(authenticationToken);
            }
        }
        filterChain.doFilter(request, response);
    }
```

5.4 Aplicatia Flask

Modulul Flask oferă trei endpointuri principale pentru funcționalitatea de inteligență artificială: generarea de mesaje obișnuite, generarea de mesaje pe baza unui fișier PDF și conversia fișierelor audio în text.

Primul endpoint, primește ca date de intrare o listă formată din mesajele dintre utilizator și bot corespunzătoare unei conversații, pentru a putea fi încarcăte în memoria botului, cu ajutorul librăriei LangChain, descrise în capitolele anterioare. Clasa ConversationChain, este clasa care controlează conversația, ei îi asociem modelul de limbaj și tipul de memorie pe care dorim să-l utilizăm. Am folosit pentru memorie tipul de memorie „Conversation Summary Buffer Memory”, pe care l-am considerat cel mai avantajos din punct de vedere al timpului, al costului și al preciziei.

Listing 5.5: Inițializarea memoriei

```
memory = ConversationSummaryBufferMemory(
    llm=OpenAI(),
    max_token_limit=2500
)
```

ConversationChain este obiectul căruia i-am asociat modelul de limbaj(LLM-ul) și memoria încărcată anterior. Am lăsat parametrul de verbose cu valoarea True,

pentru a putea vedea afișările cu prompturile pe care le primește modelul.

Listing 5.6: Inițializarea conversației

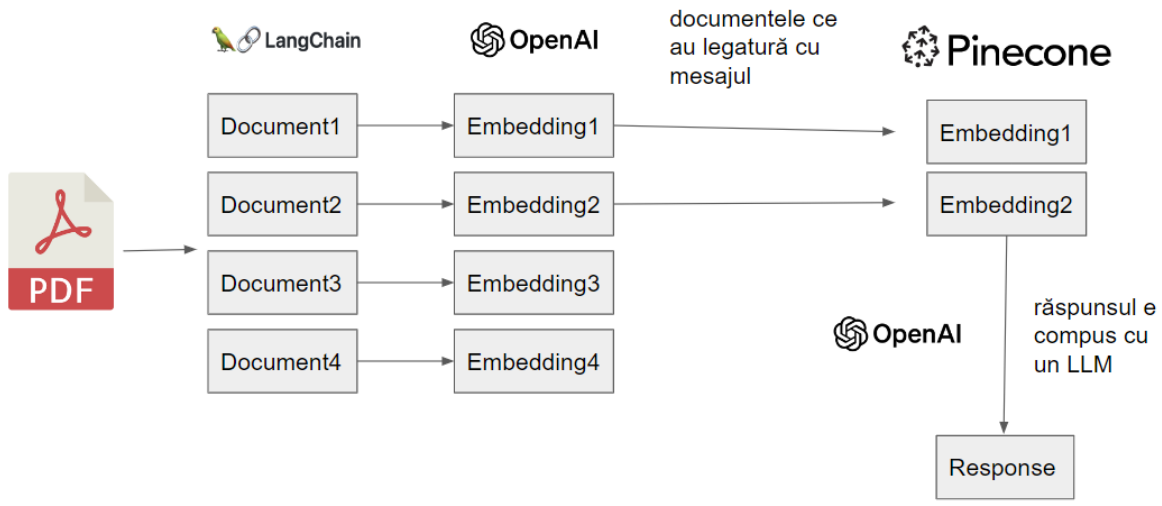
```
summary_chain = ConversationChain(
    llm=OpenAI(model="text-davinci-003"),
    verbose=True
)
summary_chain.memory = memory
```

Endpoint-ul următor are responsabilitatea de a genera mesaje bazate pe un fișier PDF. Pentru a realiza conexiunea între modelul de limbaj larg(LLM) și fișierul PDF, am utilizat Pinecone. Fluxul(Flow-ul) acestei funcționalități este reprezentat în figura de mai jos.

Pentru a realiza acest lucru, am folosit obiectul RecursiveCharacterTextSplitter din Langchain pentru a împărți fișierul PDF în obiecte de tip Document, fiecare având o dimensiune de 1000 de caractere (chunk-uri). Ulterior, am utilizat un obiect de tip embedding de la OpenAI pentru a converti aceste documente în embeddings, adică într-un vector de numere. Aceste embeddings au fost încărcate într-un index Pinecone.

Mesajul va fi reprezentat de asemenea intr-un embedding, iar Pinecone va identifica legătura dintre documentele stocate în index și mesajul respectiv. După identificarea embeddings-urilor asociate cu mesajul, acestea vor fi utilizate în prompt-ul modelului de limbaj, permițându-i acestuia să furnizeze un răspuns corespunzător la întrebări.

Figura 5.2: Fluxul mesajelor bazate pe PDF



Ultimul endpoint se ocupă de conversia mesajelor audio în text. Pentru această funcționalitate, am optat pentru utilizarea bibliotecii SpeechRecognition, care folosește

API-ul Google Cloud. Pentru a putea realiza conversia audio, am transformat fișierul într-un obiect de tip AudioFile, pe care l-am folosit în funcția recognize.

```
import speech_recognition as sr

r = sr.Recognizer()
audio_file = sr.AudioFile(filename)
with audio_file as source:
    audio = r.record(source)
text = r.recognize_google(audio, language='ro-RO')
```

5.5 Aplicația Android

În cadrul proiectului, am aplicat arhitectura Model-View-ViewModel (MVVM) în dezvoltarea aplicației mele Android. MVVM este o abordare arhitecturală modernă care facilitează o separare clară a responsabilităților și o gestionare eficientă a codului, permițând îmbunătățirea experienței utilizatorului și ușurarea procesului de mentenanță. Modelul reprezintă datele și logica de business a aplicației, asigurând o gestionare eficientă a acestora. View-ul se concentrează pe afișarea informațiilor și interacțiunea cu utilizatorul, fără a conține logica de business. ViewModel-ul îndeplinește rolul de intermediar între Model și View, asigurând gestionarea logicii de afișare a datelor și facilitând comunicarea cu Modelul. Acesta furnizează View-ului metode și obiecte observabile(state-uri) pentru a actualiza interfața grafică în mod reactiv, în concordanță cu modificările apărute în Model. Aplicatia oferă trei ecrane principale: ecranul de autentificare, de inregistrare și ecranul principal, care cuprinde conversațiile și mesajele si un drawer.

Aplicatia dispune de două navigari: o navigare între ecranelele principale ale aplicației Login/Register și Home și încă o navigare pentru ecranul Home, cu ajutorul căreia se navighează de la o conversație la alta. Motivul pentru care am optat pentru cele două navigări este pentru ca atunci când navighez între conversații să se facă recompozitia doar la ecranul conversație, nu și la drawer. Astfel am reusit să separ componenta de drawer și ceea de conversație in două componente independente.

În cadrul arhitecturii MVVM, am implementat două ViewModele principale: *drawerViewModel* și *chatViewModel*, care corespund celor două componente de ecran. Partea de View a aplicației este reprezentată de funcțiile *@Composable*, cum ar fi *ChatScreen*5.5, *DrawerScreen*5.4 și *LoginScreen*5.3.

Pentru componenta de Model, am utilizat clase de date (data class) pentru *Chat* și *Message*. Clasele de repository sunt apelate de viewModele, iar aceste clase repo-

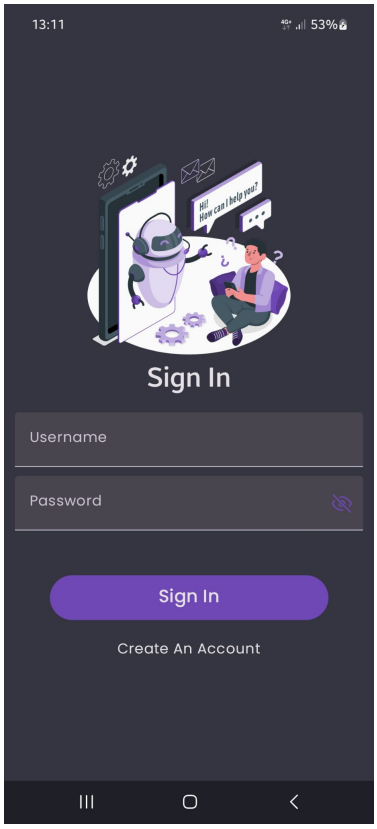


Figura 5.3: LoginScreen

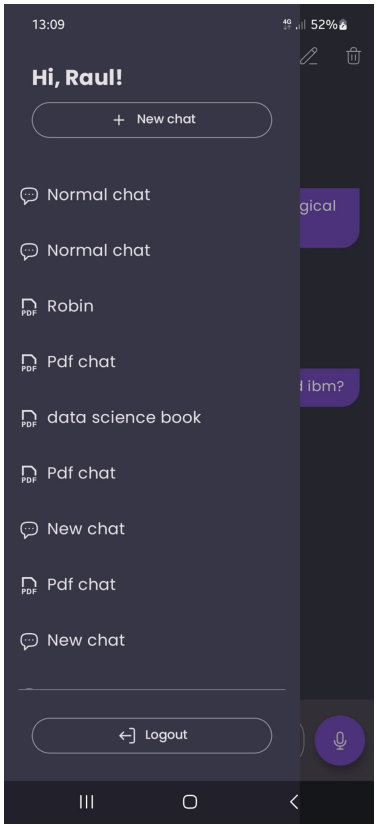


Figura 5.4: DrawerScreen

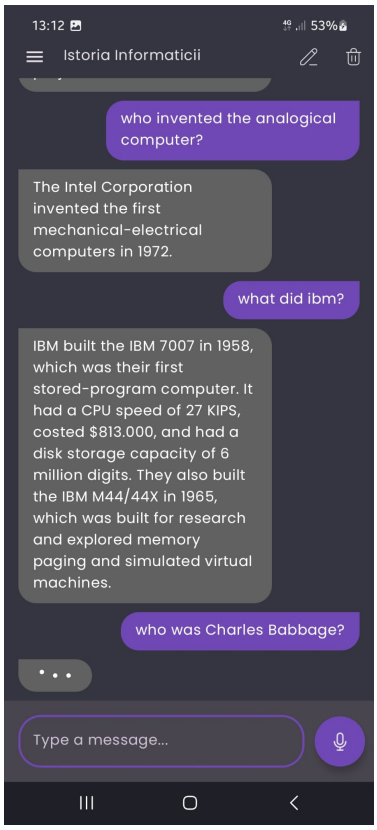


Figura 5.5: ChatScreen împreună cu componenta de LoadingMessageItem

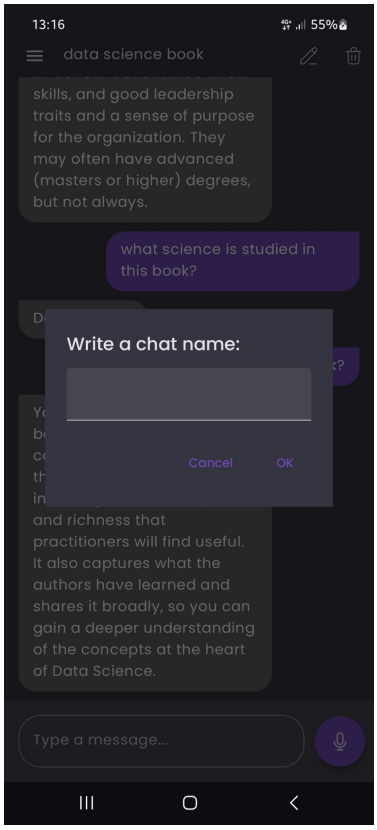


Figura 5.6: Editarea titlului conversației

sitory apelează clasele de data source, care la rândul lor fac apel la interfețele de API create cu ajutorul bibliotecii Retrofit.

Motivul pentru care am ales sa existe un pas în plus la apelurile API-urilor este pentru a putea face gestionarea erorilor mai ușor, cu ajutorul obiectului Result:

Listing 5.7: Simple code listing.

```
suspend fun login(loginRequest: LoginRequest): Result<
   TokenHolder> {
    return try {
        Result.success(authApi.login(loginRequest))
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

Ecranul de Login sau Register apare doar la prima interacțiune cu aplicația, fiindcă tokenul JWT primit în urma loginului este salvat în UserPreferences, prin librăria datastore. Astfel în momentul în care apare ecranul Splash al aplicației, este verificată existența tokenului în UserPreferences, iar dacă există utilizatorul va fi redirecționat direct în pagina principală a aplicației.

Ecranul Home contine: drawer-ul mentionat mai sus este un ModalNavigationDrawer care conține lista de conversații existente, butonul de „Logout” și butonul de „New Chat”.

Listing 5.8: Simple code listing.

```
ModalNavigationDrawer(
    drawerState = drawerState,
    drawerContent = {
        ModalDrawerSheet(modifier = Modifier.fillMaxWidth(.8f
           )) {
            DrawerScreen(
                viewModel = viewModel,
                onNewChatClick = onNewChatClick,
                onNewPdfChatClick = onNewPdfChatClick,
                onChatClick = onChatClick,
                onLogoutSuccessful = onLogoutSuccessful
            )
        }
    },
    content = content
)
```

De asemenea ecranul Home contine și ecranul conversațiilor, care este compus dintr-un Scaffold:

Listing 5.9: Simple code listing.

```
Scaffold(
    topBar = {
        AppBar(
            chatTitle = state.chatTitle,
            onMenuClick = onMenuClick,
            onDeleteClick = {
                viewModel.deleteChat()
            },
            onUpdateClick = { chatTitle ->
                viewModel.updateChat(chatTitle)
            },
            onNewPdfChatClick = onNewPdfChatClick,
        )
    },
    bottomBar = {
        MessageSection(
            messageState = viewModel.messageState,
            onSendMessage = { text ->
                viewModel.sendMessage(text)
            },
            onSendPdfMessage = { text, path ->
                viewModel.onSendPdfMessage(text, path)
            }
        )
    }
) {
    MessagesSection(
        modifier = Modifier.padding(
            bottom = it.calculateBottomPadding(),
            top = it.calculateTopPadding()
        ),
        state = state,
        messageState = messageState,
        loadMoreMessages = viewModel::loadMoreMessages
    )
```

```
}
```

Componenta *Scaffold* ajută la organizarea interfeței utilizatorului într-o structură coerentă. Este utilizat pentru a crea o structură pentru o aplicație de chat. Bara de sus *topBar* conține un *AppBar* personalizat, care afișează titlul conversației, butoanele de meniu și permite utilizatorului să efectueze acțiuni precum ștergerea conversației sau actualizarea titlului acesteia5.6. Pentru a lucra cu fereastra adițională pentru editarea titlului am folosit librăria Sheets-Compose-Dialogs[15]

Bara de jos *bottomBar* conține o secțiune de mesaje *MessageSection*, unde utilizatorul poate introduce și trimite mesaje text sau se poate inregistra pentru a trimite un mesaj audio.

În interiorul *Scaffold*, conținutul principal al ecranului este reprezentat de *MessagesSection*. Acesta afișează mesajele existente și oferă opțiunea de a încărca mai multe mesaje prin paginare, atunci când utlizatorul face scroll in sus. Folosește un *Modifier* pentru a ajusta spațiul între conținutul secțiunii și marginile ecranului.

Secțiunea *MessagesSection* este definită de codul de mai jos, care folosește *LazyColumn* pentru a afișa mesajele într-o listă.

Listing 5.10: Simple code listing.

```
LazyColumn(
    state = listState,
    modifier = modifier.padding(horizontal = 16.dp),
    reverseLayout = true
) {
    if (messageState.isLoading) {
        item { LoadingMessageItem() }
    }
    item {
        if (state.isLoading) {
            Row(
                modifier = Modifier
                    .fillMaxWidth()
                    .padding(8.dp),
                horizontalArrangement = Arrangement.Center
            ) {
                CircularProgressIndicator()
            }
        }
    }
    itemsIndexed(state.items) { index, message ->
```

```
            if (index >= state.items.size - 1 && !state.
               endReached && !state.isLoading) {
                loadMoreMessages()
            }
            MessageItem(
                messageText = message.text,
                type = message.type,
                isLast = index == state.items.lastIndex
            )
        }
    }
```

În acest cod, *LazyColumn* este utilizat pentru a crea o listă de elemente afișate vertical. El primește câteva parametri pentru a controla comportamentul și aspectul listei. Parametrul state este utilizat pentru a derula lista mesajelor la ultimul mesaj, atunci cand se trimite un mesaj sau se primeste un mesaj. În interiorul *LazyColumn*, se găsesc mai multe elemente (item și itemsIndexed) care definesc modul în care sunt afișate mesajele. Primul element este *LoadingMessageItem()*, care este afișat doar atunci când messageState.isLoading este true. Acesta este utilizat pentru a afișa un indicator de încărcare în timp ce se așteaptă răspunsul la un mesaj5.5.

CircularProgressIndicator este componenta care se afiseaza in timp ce se incarca mesajele de la server. Apoi, sunt afișate elemente individuale pentru fiecare mesaj în lista state.items. În interiorul acestei bucle, există o verificare pentru a încărca mai multe mesaje *loadMoreMessages()* atunci când se ajunge la sfârșitul listei și încă nu s-a atins limita *state.endReached* și nu se încarcă *state.isLoading*. Pentru fiecare mesaj, se afișează un MessageItem, care primește textul mesajului, tipul acestuia și indică dacă este ultimul mesaj din listă *isLast*. Aceasta permite afișarea corespunzătoare a stilurilor sau indicatorilor vizuali pentru ultimul mesaj.

Am utilizat biblioteca Wave Recorder pentru înregistrarea vocii. Aceasta este o bibliotecă scrisă în Kotlin, permite înregistrarea audio în diferite formate. Este ușor de utilizat: instantiem obiectul *WaveRecorder* cu locația unde dorim să salvăm înregistrarea audio și apelăm funcțiile startRecording/stopRecording[2].

Listing 5.11: Simple code listing.

```
val filePath:String = externalCacheDir?.absolutePath + "/
   audioFile.wav"

val waveRecorder = WaveRecorder(filePath)
waveRecorder.startRecording()
waveRecorder.stopRecording()
```

Pentru extragerea fisierelor PDF am folosit libraria Compose Multiplatform File Picker[1], personalizata pentru proiectele Jetpack Compose. Pentru a face vizibila componenta FilePicker setez parametrul *showFilePicker* pe true la apasarea unui buton. Functia *handleUri*, primeste un path de tipul „android file" salveaza fisierul in cache, pentru a putea fi trimis la backend.

Listing 5.12: Simple code listing.

```
FilePicker(
    show = showFilePicker,
    fileExtensions = listOf("pdf")
) { path ->
    showFilePicker = false
    val myPath = uriPathFinder.handleUri(context, Uri.parse(
       path!!.path))
    onNewPdfChatClick(myPath!!)
    Toast.makeText(
        context,
        "Pdf file saved successfully!",
        Toast.LENGTH_SHORT
    ).show()
}
```

5.6 Deploy-ul aplicației

Am folosit Cloud Run pentru a face deploy componentelor de backend ale aplicației, adică modulele Spring Boot și Flask.

Cloud Run este o platformă de calcul gestionată care permite rularea containerelor direct pe infrastructura scalabilă a Google[8]

Am împachetat fiecare modul în imagini de Docker folosind câte un Dockerfile pentru fiecare modul.

Serviciul Cloud Run mi-a permis să rulez aplicațiile mele într-un mediu scalabil și complet gestionat, fără a fi nevoie să-mi fac griji cu privire la infrastructură. Scalabilitatea automată și facturarea pe bază de resurse utilizate au fost alte avantaje remarcabile ale utilizării Cloud Run.

De asemenea, pentru gestionarea și stocarea fișierelor PDF, am utilizat serviciul Cloud Storage. Cloud Storage este un serviciu oferit de Google Cloud care permite stocarea, accesul și gestionarea fișierelor într-un mod scalabil și sigur.

Capitolul 6

Antrenarea unui model de recunoaștere a vorbirii pentru limba română

Pentru a antrena proiectul RobinASR, am utilizat un set de date preluat de pe CommonVoice, mai exact setul Common Voice Delta Segment 13.0 din anul 2023. Acest set de date este format din 666 de înregistrări audio scurte, furnizate de 17 voci distincte. Aceste înregistrări audio, cu o durată medie de aproximativ 5 secunde fiecare, sunt specifice domeniului legislativ. În total, corpusul de date cuprinde aproximativ o oră de înregistrări audio, oferind o varietate semnificativă de conținut și surse vocale.

Împărțirea setului de date s-a realizat în proporții de 80% pentru antrenament, 10% pentru testare între epoci și încă 10% pentru evaluarea modelului final. Această împărțire asigură o reprezentare echilibrată a datelor în fiecare etapă a antrenamentului și permite o evaluare corectă a performanței modelului în final.

Inițial, am efectuat antrenarea modelului de inteligență artificială pe un procesor, care este considerat un mediu nefavorabil pentru acest proces. Acest lucru a implicat o durată de aproximativ 15 ore pentru a finaliza 30 de epoci. Din cauza numărului redus de epoci, modelul nu a reușit să fac modelul sa conveargă la o soluție precisă.

Ulterior, am încercat să rulez antrenarea pe o placă video GeForce RTX 4090. Această placa video puternică a permis efectuarea a 300 de epoci, fără un efort considerabil. Utilizând această configurație, am obținut o durată de antrenare semnificativ mai scurtă, iar modelul a avut ocazia să acumuleze mai multe cunoștințe și să învețe mai eficient. Pentru această antrenare, am folosit o *learning_rate* inițial de 0.002 și am aplicat un *learning_anneal* de 1.05. Datele au fost de asemenea împărțite în batch-uri de câte 50 de date.

În timpul antrenamentului, am urmărit progresul funcției de loss pentru a evalua

performanța modelului. Am utilizat funcția de loss CTC (Connectionist Temporal Classification) pentru antrenarea modelului de recunoaștere a vorbirii.

Funcția de loss CTC este o metodă specifică pentru problemele de recunoaștere a vorbirii, care abordează problema de aliniere între secvențele de intrare și etichetele corecte asociate acestora. Ea permite modelului să învețe să recunoască și să decodeze secvențe de vorbire fără a necesita o aliniere exactă între intrare și etichetă. Funcția de loss CTC ia în considerare toate posibilele alinieri între intrare și etichetă și măsoară discrepanța dintre acestea. Scopul principal al funcției de loss CTC este să determine probabilitățile corecte pentru fiecare etichetă în cadrul secvenței de intrare. Graficul funcției de loss atașat prezintă evoluția acestei funcții pe măsură ce modelul se antrenează. Pe măsură ce antrenamentul avansează, observăm că valoarea funcției de loss scade treptat. Acest lucru indică faptul că modelul își îmbunătățește performanța în recunoașterea vorbirii, deoarece valoarea lossului reprezintă o măsură a discrepanței dintre etichetele corecte și predicțiile făcute de model.

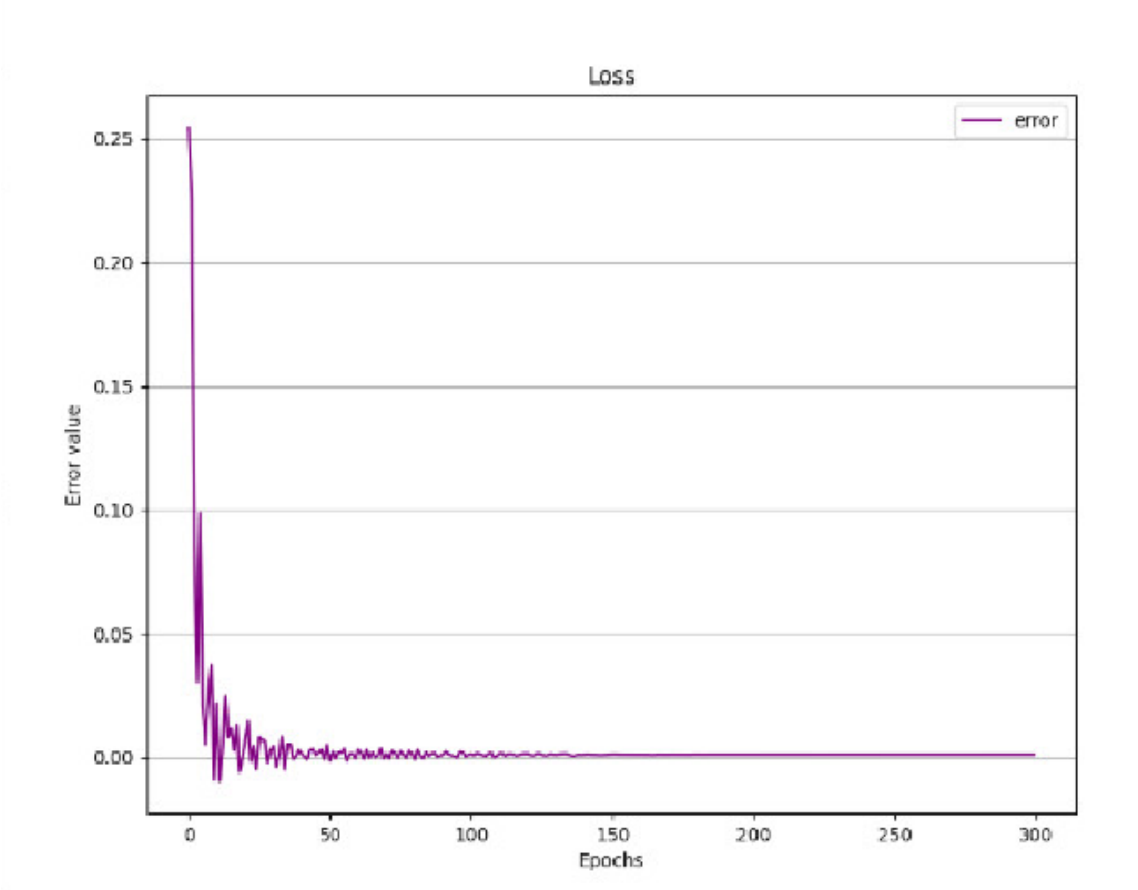


Figura 6.1: Funcția de pierdere

Analizând graficul, putem trage concluzii cu privire la eficacitatea antrenamentului și optimizarea modelului, în funcție de modul în care funcția de loss evoluează

pe parcursul epocilor. Scăderea loss-ului indică faptul că modelul se apropie de o stare în care este capabil să facă predicții din ce în ce mai precise în ceea ce privește recunoașterea vorbirii.

După evaluarea modelului utilizând date care nu au fost folosite în procesul de antrenare, am obținut o eroare de 40.86% per cuvânt și o eroare de 10.91% per caracter. Eroarea per cuvânt indică faptul că modelul face aproximativ 4 erori la fiecare 10 cuvinte recunoscute. În același timp, eroarea per caracter de 10.91% arată că modelul greșește aproximativ 1 din 10 caractere recunoscute.

Capitolul 7

Posibilități ulterioare de dezvoltare și concluzii

7.1 Posibilități ulterioare de dezvoltare

Utilitatea acestei aplicații poate fi pusă în valoare prin usurința de folosire și eficiența de care dă dovadă. Cu toate acestea, există o serie de direcții în care aplicația poate fi dezvoltată în viitor pentru a oferi funcționalități suplimentare și pentru a îmbunătăți experiența utilizatorilor. Aceste posibilități ulterioare de dezvoltare includ:

- Extinderea suportului pentru alte tipuri de fișiere: Pe lângă mesajele bazate pe fișiere PDF, aplicația ar putea fi extinsă pentru a permite trimiterea și primirea altor formate de fișiere, cum ar fi documente Word, prezentări PowerPoint sau fișiere Excel. Acest lucru ar oferi utilizatorilor o gamă mai largă de opțiuni și le-ar permite să partajeze diferite tipuri de conținut în conversațiile lor.

- Integrarea cu servicii de stocare în cloud: O altă funcționalitate valoroasă ar fi integrarea cu servicii populare de stocare în cloud, cum ar fi Google Drive, Dropbox sau OneDrive. Aceasta ar permite utilizatorilor să acceseze direct fișierele lor stocate în cloud și să le trimită în conversații sau să primească fișiere direct în aplicație, fără a fi necesară descărcarea și încărcarea manuală a acestora.

- Integrarea de plugin-uri cum ar fi Wolfram Alpha, prin care utilizatorii ar putea obține rezultate precise și detaliate pentru întrebări matematice sau plugin pentru cautarea pe internet, care ar permite utilizatorilor să efectueze căutări direct din aplicație și să primească rezultate relevante și actualizate.

7.2 Concluzii

Unele dintre cele mai bune idei se nasc din dorința de a simplifica, accelera, îmbunătăți sau consolida un proces existent. Totodată, ideile cu adevărat remarcabile sunt rezultatul nenumăratelor experimente eșuate și al prototipurilor care, prin performanța lor, alimentează pasiunea creatorului pentru inovație.

În cadrul acestui proiect, am dezvoltat un chatbot cu memorie, care are capacitatea de a înțelege și de a răspunde la conversații anterioare, adăugând astfel o dimensiune de continuitate și personalizare în interacțiunea cu utilizatorii. De asemenea, am integrat funcționalitatea de conversație bazată pe un fișier PDF, permițând utilizatorilor să comunice și să primească informații relevante în baza acestui format. În plus, am antrenat și implementat un model existent de inteligență artificială, care a contribuit la îmbunătățirea capacităților de recunoaștere vocală pentru limba română. În plus antrenarea modelului existent a contribuit în mod semnificativ la creșterea nivelului meu de cunoștințe și înțelegere în domeniul recunoașterii vocalele. Toate aceste eforturi și realizări în dezvoltarea aplicației au condus la îmbunătățirea experienței unor potențiali utilizatori și la posibilitatea de a explora și mai multe idei și inovații în viitor.

Bibliografie

[1] George Shalvashvili Adam Brown. Compose multiplatform file picker. https://github.com/Wavesonics/compose-multiplatform-file-picker. Online; accessed 15 May 2023.

[2] Andre-max Ahmadreza, Abhay Koradiya. Android wave recorder. https://github.com/squti/Android-Wave-Recorder. Online; accessed 15 May 2023.

[3] Andrei-Marius Avram, Vasile Păiş, and Dan Tufiş. Romanian speech recognition experiments from the robin project. *arXiv preprint arXiv:2111.12028*, 2021.

[4] Andrei-Marius Avram, Vasile Păiș, and Dan Tufiș. Towards a romanian end-to-end automatic speech recognition based on deepspeech2. In *Proceedings of the Romanian Academy, Series A*, pages 395–402, 2020.

[5] Microsoft Azure. What is java spring boot? https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-java-spring-boot/. Online; accessed 15 May 2023.

[6] Python Basics. What is flask python. https://pythonbasics.org/what-is-flask-python/. Online; accessed 15 May 2023.

[7] James Briggs. Chatbot memory for chat-gpt, davinci + other llms - langchain 4. https://www.youtube.com/watch?v=X05uK0TZozM&ab_channel=JamesBriggs. Online; accessed 15 May 2023.

[8] Google Cloud. What is cloud run. https://cloud.google.com/run/docs/overview/what-is-cloud-run. Online; accessed 15 May 2023.

[9] CoRoLa. Corola descriere generalĂ. https://corola.racai.ro/. Online; accessed 15 May 2023.

[10] Android Developers. Build better apps faster with jetpack compose. https://developer.android.com/jetpack/compose. Online; accessed 15 May 2023.

[11] Android Developers. Why adopt compose. `https://developer.android.com/jetpack/compose/why-adopt/#less-code`. Online; accessed 15 May 2023.

[12] Corneliu Octavian Dumitru and Inge Gavat. A comparative study of feature extraction methods applied to continuous speech recognition in romanian language. In *Proceedings ELMAR 2006*, pages 115–118. IEEE, 2006.

[13] Alexandru-Lucian Georgescu, Horia Cucu, and Corneliu Burileanu. Kaldi-based dnn architectures for speech recognition in romanian. In *2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD)*, pages 1–6. IEEE, 2019.

[14] Alexandru-Lucian Georgescu, Horia Cucu, Andi Buzo, and Corneliu Burileanu. Rsc: A romanian read speech corpus for automatic speech recognition. In *Proceedings of the 12th language resources and evaluation conference*, pages 6606–6612, 2020.

[15] Maximilian Keppeler. Sheets-compose-dialogs. `https://github.com/maxkeppeler/sheets-compose-dialogs`. Online; accessed 15 May 2023.

[16] LangChain. Welcome to langchain. `https://python.langchain.com/en/latest/index.html`. Online; accessed 15 May 2023.

[17] Mikhail Belyaev Marat Akhin. Kotlin language specification. `https://kotlinlang.org/spec/introduction.html`. Online; accessed 15 May 2023.

[18] OpenAI. Models. `https://platform.openai.com/docs/models/gpt-3-5`. Online; accessed 15 May 2023.

[19] Roie Schwaber-Cohen. What is a vector database? `https://www.pinecone.io/learn/vector-database/`. Online; accessed 15 May 2023.

[20] Arjun Sha. How to run a chatgpt-like llm on your pc offline. `https://beebom.com/how-run-chatgpt-like-language-model-pc-offline/`. Online; accessed 15 May 2023.

[21] Dan Tufis, Verginica Barbu Mititelu, Elena Irimia, Maria Mitrofan, Radu Ion, and George Cioroiu. Making pepper understand and respond in romanian. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, pages 682–688. IEEE, 2019.

[22] Common Voice. Why common voice? `https://commonvoice.mozilla.org/en/about`. Online; accessed 15 May 2023.

[23] Quan Wang. Speaker recognition — by award winning textbook author. `https://www.udemy.com/course/speaker-recognition/`. Online; accessed 15 May 2023.

[24] Junjie Ye, Xuanting Chen, Nuo Xu, Can Zu, Zekai Shao, Shichun Liu, Yuhan Cui, Zeyang Zhou, Chao Gong, Yang Shen, et al. A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. *arXiv preprint arXiv:2303.10420*, 2023.

[25] A. (2017) Zhang. Speech recognition (version 3.8) [software]. `https://github.com/Uberi/speech_recognition#readme`. Online; accessed 15 May 2023.