UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Agent Asistent Inteligent
Prelucrarea și generarea vorbirii

Matei Mihalea

Coordonator științific:

Conf. dr. ing. Andrei Olaru

BUCUREȘTI

2018

CUPRINS

SINOPSIS

Lucrarea propune un sistem care îi permite utilizatorului să interacționeze cu aplicația, astfel el fiind împărțit în două părți: procesarea vorbirii, în urma căreia se va obține cererea utilizatorului în format text, respectiv partea de generare a vorbirii pornind de la text. În cadrul acestei lucrări accentul este pus pe procesarea vorbirii și pe găsirea de modalități de a reduce erorile provenite din urma acestui proces.

ABSTRACT

The paper proposes a system that allows the user to interact with the application, so it consists of two parts: the processing of speech, which will result in the user's request in text format, respectively the speech generation part starting from the text. Inside this paper the emphasis is on speech processing and finding ways to reduce the errors resulting from this process.

MULȚUMIRI

(opțional) Aș vrea să îi mulțumesc lui Andrei Olaru pentru ajutorul și sfaturile acordate în cadrul proiectului si realizării lucrării de diplomă.

1 INTRODUCERE

1.1 Context

Domeniul asistenților inteligenți este unul foarte vast, lucru care se datorează atât multitudinii de taskuri pe care diverse astfel de aplicații sunt concepute pentru a le realiza(spre exemplu asistentul celor de la Amazon: Alexa a fost proiectat astfel încât să poată fi extins cu până la 15000 de eventuale noi aptitudini [8]), cât și avansarea rapidă a tehnologiei, spre exemplu odată cu avansarea și creșterea rapidă în popularitate a dispozitivelor din sfera Internet of Things(IoT) [6], vor apărea în consecință noi oportunități și pe piața asistenților inteligenți, care ar putea prezenta un control unitar asupra tuturor dispozitivelor IoT de care dispune un utilizator.
Cu toate acestea scopul principal al oricărui tip de asistent inteligent este cel de a ajuta omul în orice fel de acțiuni acesta dorește să întreprindă, iar acesta în sine este un motiv suficient pentru care dezvoltarea acestora nu se va opri, iar funcționalitățile asistenților inteligenți vor continua să se extindă odată cu progresul tehnologic și descoperirea de noi modalități de utilizare.
Proiectul Intelligent Assistant Agent propune o implementare offline și open-source care poate fi o soluție pentru garantarea securității și integrității datelor utilizatorilor, salvarea acestora pe un server centralizat fiind o soluție pe care mulți utilizatori nu o găsesc sigură și mai mult de atât, poate fi chiar invazivă în unele cazuri. Este de asemenea important de remarcat că implementarea este bazată pe un graf de cunoștiințe, lucru care poate constitui o soluție pentru o problemă actuală chiar și pentru aplicațiile de asistenți inteligenți de ultimă generație, și anume înțelegerea contextului dintr-o conversație.

1.2 Problema

În cadrul proiectului, accentul este pus pe dialogul dintre utilizator și aplicație și nu neapărat pe utilizarea informației pentru realizarea anumitor task-uri, lucru care ar fi mai ușor de

implementat odată ce ne asigurăm că asistentul **înțelege** orice fel de comandă primită de la utilizator.
Principala funcționalitate a asistentului este deci cea de căutare și furnizare de informații sub formă de limbaj natural, păstrarea impresiei de conversație cu o persoană reală fiind de asemenea o componentă importantă în cadrul dezvoltării proiectului, motiv pentru care chiar și în cazurile în care nu este găsit răspunsul sunt definite replici implicite cu scopul de menținere a fluxului conversației.

1.3 Obiective

Principalul obiectiv impus de la începutul dezvoltării a fost ca interacțiunea cu asistenul inteligent să fie realizată exclusiv folosind vocea (în limbaj natural sau o forma simplificată de limbaj natural), iar acesta să fie capabil să întoarcă un răspuns, de asemenea în limbaj natural. Soluția propusă pentru rezliazarea acestui lucru este complet locală și are toate informațiile necesare generării răspunsurilor salvate sub formă de grafuri de context, această abordare având obiectivul de a elimina eventualele riscuri apărute odată cu utilizarea serviciilor de cloud.
Pentru a asigura performanța globală a aplicației, chiar presupunând că modulul de generare al răspunsului de la text funcționează perfect, încă vom avea de rezolvat problema obținerii acelei comenzi în format text, prin folosirea unui modul de recunoaștere a vorbirii. Din această cauză, performanța componentei de speech-recognition va avea un impact puternic asupra funcționării întregii aplicații, iar găsirea unei soluții eficiente și precise este esențială pentru rezolvarea proiectului.

1.4 Soluția propusă

Din punctul de vedere al interacțiunii utilizator-aplicație, am folosit motoare de recunoaștere vocală open-source (mai exact Sphinx4 [10] și Mozilla DeepSpeech [2]) pentru obținerea inputului utilizatorului în format text, urmată de un modul de corectare a erorilor care asigură faptul că o comandă corectă va ajunge la intrarea modulului de generare a răspunsului, componentă care pe baza grafului de cunoștiințe va forma replica asistentului sub formă de text, aceasta ajungând la sfârșit înapoi la utilizator în format vocal după utilizarea componentei de genereare a vorbirii din text (text-to-speech).

1.5 Rezultatele obținute

Rezultatele proiectului au fost promițătoare pentru scenariile de test asupra cărora s-au realizat măsurătorile, în cazul recunoașterii vorbirii obținând o precizie de identificare de peste 75% . Rezultatele obținute sunt relevante în domeniul asistenților inteligenți deoarece marea majoritate a aplicațiilor moderne de acest tip nu sunt complet locale și open-source, iar testele noastre ne permit să observăm limitările unei astfel de abordări comparativ cu o soluție în cloud spre exemplu. O analiză detaliată a rezultatelor obținute este disponibilă în cadrul capitolului 6

1.6 Structura lucrării

În continuare, structura lucrării este următoarea: În capitolul 2 se prezintă motivația alegerii de implementare locală și open-source a proiectului, capitolul 3 conține o prezentare a soluțiilor folosite de asistenți inteligenți de ultimă generație cu privire la recounașterea vorbirii, capitolul 4 propune o paralelă între soluțiile de cloud și soluțiile locale, urmată de diverse modalități de corectare a erorilor ce pot apărea la interpretarea comenzilor vocale. În cadrul capitolului 5 conține implementarea proiectului, accentul fiind pus pe recunoașterea vocii și modalități de reducere a erorilor de interpretare.Capitolul 6 conține o testare amănunțită a influenței zgomotului asupra performanței aplicației și direcții de dezvoltare ulterioară, pentru ca în final sa avem concluziile asupra proiectului în capitolul 7.

2 ANALIZA CERINȚELOR / MOTIVAȚIE

Proiectul "Asistent Agent Inteligent" poate fi structurată în două mari părți componente:

- Procesarea și generarea vorbirii
- Generarea de răspunsuri folosind grafuri de context

În cadrul lucrării de licență m-am ocupat de prima parte, și anume procesarea și generarea vorbirii. După cum se observă încă din nume, acest lucru a presupus realizarea unui modul de interpretare a vocii și obținerea echivalentului în format text și un modul de generare a vocii pornind de la text. În plus am implementat și un modul de corectare a eventualelor erori provenite din recunoașterea vocii, cât și o interfață grafică pentru facilitarea utilizării asistenului inteligent.

După cum am precizat în secțiunea 1.3, principalul obiectiv încă din fazele incipiente ale proiectului au fost implementarea **locală** și **open-source** a acestuia. Motivația acestei alegeri este prezentată în continuare prin formularea de argumente pentru susținerea celor două obiective.

Implementarea open-source oferă:

- posibilitateaa dezvoltării rapide
- debugging, utilizatorii pot raporta erori în software sau chiar propune soluții la astfel de probleme
- posibilitatea de personalizare, fiecare client având posibilitatea să extindă funcționalitățile deja existente ale asistentului, și chiar să adauge utilități și funcționalități noi

Implementarea locală oferă:

- securitate sporită
- asigură confidențialitatea datelor
- asigură integritatea datelor

Marele avantaj al unei implemntări open-source a unui asistent inteligent este posibilitatea personalizării. Întrucât asistentul nu are definit precis un domeniu de lucru, utilizatorii pot veni cu propriile idei și, prin adăugarea sau schimbare de date din baza de cunoștiințe a aplicației sau prin adăugarea de funcționalități la structura deja definită, pot crea un "nou" asistent specializat în orice domeniu. Din moment ce raza de aplicabilitate a asistențiilor inteligenți se întinde pe nenumărate domenii, precum:

- traduceri
- găsirea de informație
- interacțiune în limbaj natural
- răspunsul la întrebări
- recomandări
- trimitere și preluare de mesaje
- setarea de alarme, intrări în calendar, întâlniri,

și multe altele, idea realizării unei astfel de tehnologii open-source poate da naștere la multiple dezvoltări ulterioare și numeroase noi funcționalități.

Întrucât marea majoritate a asistenților inteligenți de ultimă generație folosesc servicii de cloud [9] [1] ale companiilor de către care sunt dezvoltate (de ex: Alexa folosește serviciile cloud de la Amazon, Siri folosește cloud-ul de la Apple sau Cortana folosește cloud-ul Microsoft), problemele securității și în special confidențialității datelor au devenit în ultimii ani un subiect de discuție și controverse, în jurul acestora existând un mare semn de întrebare, iar o cale către rezolvarea la nivel global a acestei probleme nu a fost încă găsită.

După cum este menționat în cadrul articolului [14], prima problemă cu care se confruntă utilizatorii de cloud este aceea că trebuie să încredințeze securitatea și integritatea datelor unei părți terțe, însă chiar și dacă presupunem că putem avea încredere în furnizorul serviciilor de cloud, există posibilitatea ca căutări guvernamentale să fie realizate pe datele tale, lucru facilitat de o excepție în legislație care specifică faptul că o persoană nu se poate aștepta la confidențialitate atunci când informația este dezvăluită unei părți terțe. Acest fapt ridică mari semne de întrebare pentru o mulțime de domenii, precum cel al organizațiilor de asistență medicală, care au salvate datele confidențiale ale pacienților în cloud. Așa cum pacienții ar putea manifesta neîncredere că datele lor vor fi păstrate în siguranță, așa ar putea și utilizatorii

de asistenți inteligenți să pună la îndoială cine are acces la datele lor personale.

Luând în considerare toate argumentele menționate mai sus, alegerea dezvoltării unui asistent inteligent open-source și local puternic legată de nevoia de securitate a utilizatorului, însă și de posibilitatea de dezvoltare a unui asistent bazat pe nevoile proprii ale clientului.

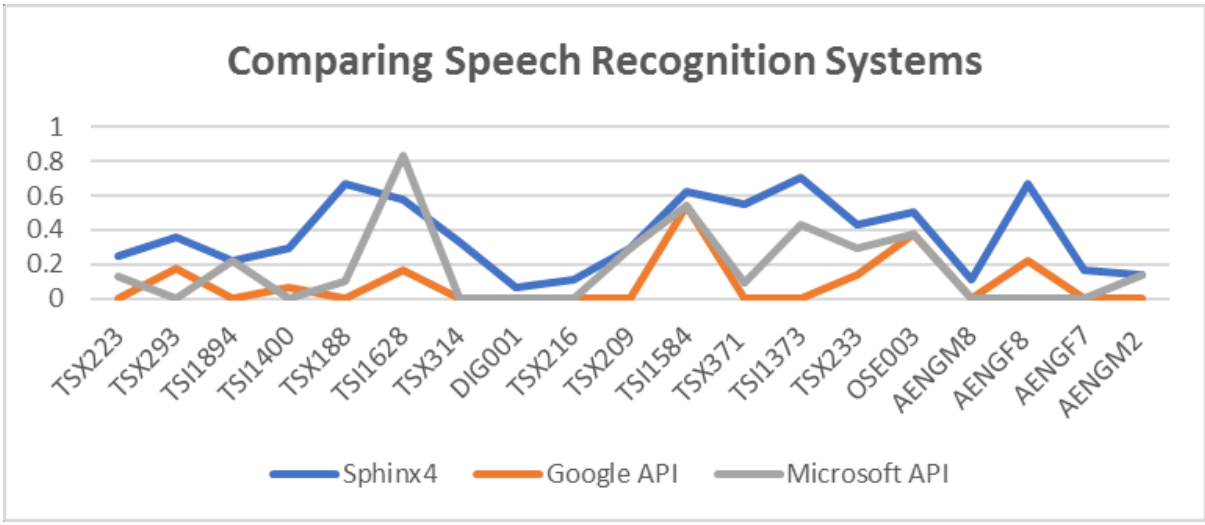
3 STUDIU DE PIAȚĂ / METODE EXISTENTE

Din punctul de vedere al tehnologiilor de ultimă oră ("State of the Art") din domeniul asistenților inteligenți, pentru realizarea funcției de recunoaștere vocală(componenta centrală a proiectului de licență), toate marile companii dezvoltatoare folosesc servicii de cloud, iar din moment ce restricția implementării locale nu ne permite să facem același lucru, a fost nevoie de o îndelungată periodă de studiu al domeniului motoarelor de recunoaștere vocală locale. Printre soluțiile cercetate și testate se numără: Sphinx4 [10], Mozilla Deepspeech[2], Kaldi[15], Julius [11]. La începutul studiului majoritatea articolelor și lucrărilor [4] din domeniu indicau Sphinx4 ca fiind cel mai performant dintre aceste sisteme, însă pe parcurs am găsit referințe mai recente [5] [13] care susțin că noul motor propus de Mozilla(Deepspeech) va ajunge în scurt timp să depășească orice altă metodă de recunoștere vocală open-source, dacă nu a făcut-o deja.

Deși abordarea în cloud nu a fost dorită în cadrul acestui proiect, am continuat studiul și testarea Google Speech Recognition API, în paralel cu cea a sistemelor open-source, pentru a putea realiza o analiză comparativă. Deși există deja studii și articole care fac această paralelă [12], am considerat că testarea tuturor soluțiilor "hands-on" poate produce o imagine mai clară a diferențelor, cu atât mai mult că această abordare permite observare performanțelor pe testele create de tine, în circumstanțele de utilizare(zgomot) alese de noi.

Diferența preciziei de recunoaștere a vorbirii între Google Speech Recognition API și motoare open-source (Sphinx4 spre exemplu) este descrisă ca fiind una semnificativă în documente și studii realizate pe această temă [3], prin rezultate precum cele din figurile 1 și 2.

Figura 1: Comparație rata de eroare Sphinx4 - Google API - Microsoft API



File	Sphinx4		Google API		Microsoft API	
	WA	WER	WA	WER	WA	WER
TSX223	0.88	0.25	1.0	0.0	0.88	0.13
TSX293	0.64	0.36	0.82	0.18	1.0	0.0
TSI1894	0.78	0.22	1.0	0.0	0.78	0.22
TSI1400	0.79	0.29	0.93	0.07	1.0	0.0
TSX188	0.33	0.67	1.0	0.0	0.0	0.1
TSI1628	0.42	0.58	0.83	0.17	0.17	0.83
TSX314	0.67	0.33	1.0	0.0	1.0	0.0
DIG001	0.93	0.07	1.0	0.0	1.0	0.0
TSX216	0.89	0.11	1.0	0.0	1.0	0.0
TSX209	0.71	0.29	1.0	0.0	0.71	0.29
TSI1584	0.38	0.62	0.46	0.54	0.46	0.54
TSX371	0.45	0.55	1.0	0.0	0.91	0.09
TSI1373	0.29	0.71	1.0	0.0	0.57	0.43
TSX233	0.71	0.43	0.71	0.14	0.71	0.29
OSE003	0.63	0.5	0.63	0.38	0.63	0.38
AENGM8	0.89	0.11	1.0	0.0	1.0	0.0
AENGF8	0.33	0.67	0.78	0.22	1.0	0.0
AENGF7	0.83	0.17	1.0	0.0	1.0	0.0
AENGM2	0.86	0.14	1.0	0.0	0.86	0.14
Mean	WER: 0.37		WER: 0.09		WER:0.18	

Figura 2: Comparație rata de eroare și precizia Sphinx4 - Google API - Microsoft API

Deși diferența preciziei este una mare între Sphinx4 și Google (error rate 37% comparativ cu 9%), aceste rezultate confirmă însă faptul că efectuarea propriilor teste pe motorul de recunoaștere vocală în cloud a fost utilă, întrucât după cum se observă din tabelele 1 și 2, Google Speech Recognition API obține o precizie aproape perfectă (99%), atât în condițiile de testare cu zgomot puternic, cât și cu zgomot redus.

Ipostazele de test proprii au dezvăluit de asemenea și faptul că Sphinx4 este foarte sensibil la zgomote, de fundal, soluția funcționând foarte bine în condiții idele (fără zgomot) și din ce în ce mai prost pe măsură ce intensitatea zgomotului crește, fapt care nu era menționat în cadrul studiului discutat mai sus.

4 SOLUȚIA PROPUSĂ

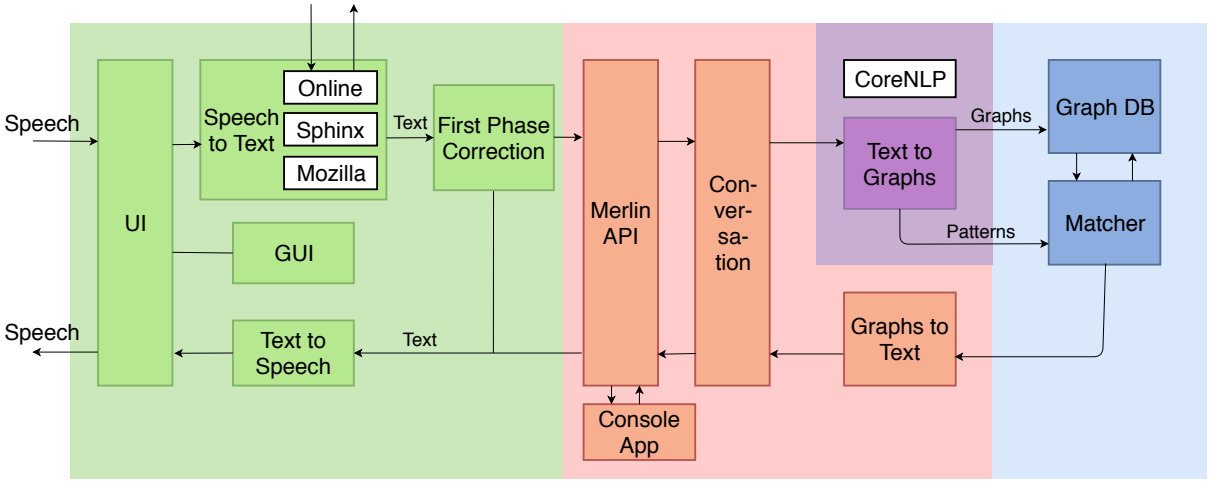


Figura 3: Arhitectura proiectului

În figura 3 este prezentată o schema generală a proiectului. În cadrul proiectului m-am ocupat în mare parte de interacțiunea dintre utilizator și aplicație, în particular înregistrarea fișierului audio care conține input-ul utilizatorului, sub forma unei propoziții în limbaj natural, obținerea afirmației utilizatorului în format text, folosind un instrument de recunoaștere vocală, precum și generarea vorbirii plecând de la text, folosită pentru a obține răspunsul la cererea utilizatorului de asemenea în format vocal.

În continuare sunt prezentate pe scurt clasele folosite pentru atingerea obiectivelor menționate mai sus, și anume:

- class JavaSoundRecorder - clasa care realizează înregistrarea unui fișier audio, cu un format specificat în cadrul metodei `getAudioFormat()`, și folosește funcțiile `start()` și `finish()` pentru a începe, respectiv termina înregistrarea fișierului.
- class SpeechUtils - clasa utilizată pentru realizarea componentei de text-to-speech a aplicației. În cadrul funcției `init(String voiceName)` este transmisă ca parametru vocea care va fi folosită, iar în cadrul metoda `doSpeak(String speakText)` se va produce output-ul sonor aferent șirului de caractere primit ca parametru. Funcția `terminate()` realizează dealocarea structurilor de date alocate în init.

- public class SpeechToText - clasa principală, folosește cele două clase menționate mai sus și un motor de recunoaștere a vorbirii (Sphinx4/ Mozilla DeepSpeech) pentru a obține comenzile utilizatorului sub formă de text, iar răspunsul întors de aplicație sub formă de voce.

4.1 Soluții offline

Întrucât unul din obiectivele proiectului era acela de a realiza o aplicație open-source și locală, am căutat în primul rând o implementare a unui motor de recunoaștere vocală care să fie de asemenea locală și open-source. O implementare offline presupune salvarea tuturor datelor și informațiilor din cadrul aplicației local, acest lucru conducând la eliminarea pericolelor de securitate introduse de soluțiile de cloud. După cum am specificat si in secțiunea 2, pericolele introduse de salvarea datelor personale pe un server central constituie subiectul unei largi dezbateri la nivel mondial, existând numeroase exemple de situații in care companii folosesc datele personale ale clienților pentru data-mining. Acest lucru cauzează lipsa de încredere a cliențiilor în astfel de soluții, și de aici motivația noastră de a găsi o alternativa pentru rezolvarea acestei probleme. O astfel de soluție este însa însoțită si de o serie de dezavantaje, cele mai notabile fiind memoria adițională ocupată de structuri precum alfabetul limbii în care se realizează traducere sau modelul limbii, și după cum am descoperit pe parcursul implementarii timpul (și calitatea) de procesare a fișierelor audio este inferior soluțiilor de cloud.

4.2 Soluții cloud

În cadrul proiectului, am testat și soluții cloud pentru recunoașterea vorbirii, în particular este vorba despre Google Speech Recognition API. Motivul pentru care am făcut acest lucru a fost în principal pentru a efectua o evaluare comparativă a software-ului local, open-source față de engine-ul online al celor de la Google, însă și pentru a avea o opțiune de rezervă în cazul în care soluțiile locale nu atingeau o precizie a detecției necesară pentru buna funcționare a aplicației. Era evident încă de înainte de a începe testarea că soluțiile de cloud vor avea o performanță superioară, atât din punct de vedere al vitezei de execuție, cât și din punct de vedere al ratei erorilor, însă implemtarea curentă a aplicației reușește reducerea erorilor prin

tehnici precum eliminarea zgomotului de fundal din fișiere audio sau împărțirea pe cuvinte a fișierului. Despre aceste tehinci se vorbește în detaliu în continuare și de asemenea în capitolul 5. De asemenea o comparație mai detaliată a performanțelor este realizată în capitolul 6.

4.3 Împărțirea fișierului în cuvinte

Întrucât Sphinx4 oferă posibilitatea întoarcerii celor mai bune n variante ale unei traduceri, unde n este un întreg primit ca parametru, am decis o abordare alternativă față de traducerea fișierului audio complet aferent unei comenzii vocale ale utilizatorului, această abordare fiind împărțirea pe cuvinte a fișierului, astfel dacă o comandă are spre exemplu 7 cuvinte, se vor crea 7 noi fișiere audio, fiecare conținând câte un cuvânt și fiind tradus independent pentru a obține cele mai bune potriviri pentru fiecare cuvânt în parte. Această abordare are meritul de a permite utilizatorului alegerea fiecărui cuvânt pentru a forma în cele din urmă propoziția completă. Astfel, în funcție de aplicație, se pot introduce diverse "formule" pentru găsirea cuvântului potrivi(spre exemplu o combinație dintre încrederea întoarsă de motorul de recunoaștere vocală și spre exemplu diferența față de cea mai apropiată frază dintr-un set de fraze predefinit sau o listă de șabloane care ar avea ca scop asigurarea corectitudinii gramaticale a propoziției).

4.4 Eliminarea zgomotului de fundal

Când am testat prima pentru dată soluții open-source de recunoaștere a vocii, rezultatele erau atât de slabe încât am crezut că vom fi nevoiți să utilizăm serviciile de cloud ale celor de la Google, care părea să funcționeze perfect. Am descoperit mai târziu că acest comportament al proiectelor open-source era cauzat de zgomotul de fundal, situație care cel mai probabil era tratată în cazul implementării API-ului din cloud. Modul prin care am realizat că zgomotul de fundal era cauza pentru proasta funcționare a sistemului a fost înregistrarea de fișiere fără astfel de zgomot prezent (prin captura sunetului direct de pe placa de sunet a laptop-ului, altfel spus, captura sunetului produs de laptop). Secțiunea 5.3.3 din capitolul următor menționează mai multe detalii referitoare la acest proces și modul în care a fost implementat. Astfel am fost nevoit să găsesc o modalitatea de a elimina sunetul de background dintr-un fișier audio întrucât orice fel de înregistrare audio realizată folosind un microfon va conține în

mod inevitabil zgomot. Prima și cea mai populară soluție pe care am găsit-o a fost utilizarea unui software de prelucrare audio în interfața grafică (GUI), însă această metodă presupunea realizarea manuală a procesului pentru fiecare fișier pentru care se dorea eliminarea zgomotului din background. Am căutat deci o soluție alternativă și am găsit o modalitate de a face același lucru folosind sox, un utilitar de modelare audio din linia de comandă. Mai multe detalii se găsesc în secțiunea 5.3.2.

4.5 Corectarea erorilor

Pentru a asigura corectitudinea traducerii și acțiunea aplicației va fi luată în conformitate cu cerințele utilizatorului, acestuia i se oferă posibilitatea de a corecta traduceri greșite, prin comenzi vocale de genul "not word1, word2", acest lucru cauzând schimbarea cuvântului "word1" cu "word2" pentru traducerea curentă. Acest proces se repetă până când comanda nu va mai conține nicio eroare.

4.6 Text to speech

Pentru orice aplicație de asistent inteligent și orice aplicație software în general, este esențială simplitatea utilizării. Acest lucru conduce în cazul asistenților inteligenți la realizarea comunicarii cu aplicația pe cât de mult posibil, sau chiar exclusiv pe cale vocală. Având în vedere aceste considerente, un modul de text-to-speech este o utilitate de baza pentru orice asistent inteligent produs la scară largă. Pe această temă, nu este importantă numai înțelegerea vorbirii produse de aplicație, ci și vocea capacitatea vocii produse de a părea umană, aspect foarte important deoarece utlizatorii pot găsi o voce care nu îndeplinește o astfel de restricție iritantă, sau se vor plictisi mai repede de utilizarea aplicație. După ce am testat mai multe tool-uri offline de generare a vorbirii, cel pe care l-am găsit a fi cel mai apropiat de vorbirea umană a fost pico2wave, un scurt ghid de utilizare putând fi găsit în cadrul secțiunii 5.4.

5 DETALII DE IMPLEMENTARE

5.1 Înregistrarea fișierului audio

După cum am menționat și în capitolul 4, înregistrarea unui fișier audio este realizată în cadrul clasei JavaSoundRecorder, aceasta având următoarele câmpuri:

- `static final long RECORD_TIME` - durata înregistrării, exprimata în milisecunde (pentru o înregistrare de 10 secunde va avea valoarea 10000).
- `File wavFile` - fișierul care va fi scris (exemplu de instanțiere: File wavFile = new File("AudioFiles/audio.wav");)
- `AudioFileFormat.Type fileType` - definește tipul fișierului înregistrat, în cadrul proiectului am folosit numai fișiere de tip wav, câmpul fiind deci instanțiat cu valoarea AudioFileFormat.Type.WAVE
- `TargetDataLine line` - linia de la care sunt captate datele audio

Formatul fișierului înregistrat este definit în cadrul funcției getAudioFormat, Este foarte important ca formatul fișierelor să rămână același în cadrul aplicației deoarece și sistemele de recunoaștere a vocii au astfel de setări realizate, iar încălcarea acestora va cauza o traducere greșită.

5.1.1 Înregistrarea folosind microfon

5.2 Recunoașterea vocii

Întrucât atât Sphinx4 cât, și Mozilla DeepSpeech au oferit în urma testelor performanțe comparabile, am decis ca în cadrul aplicației să lăsăm la latitudinea utilizatorului motorul de recunoaștere vocală utilizat. Adăugând și posibilitatea împărțirii pe cuvinte a inputului despre

care am vorbit în secțiunea anterioară, vom avea deci 4 moduri de funcționare, și anume: [Sphinx-1/DeepSpeech-2/Sphinx split-3/DeepSpeech split-4]

5.2.1 Sphinx

Sphinx4 este o bibliotecă de recunoaștere vocală de ultimă generație și independent de vorbitor scris exclusiv în Java și prevede un API intuitiv care convertește înregistrări ale vorbirii în text folosind modelele acustice CMUSphinx. Proiectul a rezultat dintr-o colaborare între grupul Sphinx al universității Carnegie Mellon, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL), and Hewlett Packard (HP), cu contribuții de la Universitatea din California de la Santa Cruz (UCSC) și Massachusetts Institute of Technology (MIT), fiind considerat unul dintre cele mai avansate și performante sisteme open-source de recunoaștere a vocii, alături de Mozilla DeepSpeech.
Înainte de a începe folosirea bibliotecii, este necesară setarea de către utilizator a 4 atribute:

- Modelul acustic
- Dicționarul
- Gramatica/ Modelul limbii
- Sursa vorbirii

În afară de sursa vorbirii, toate celelalte atribute vor fi inițializate ca parametrii pentru un obiect de tipul `Configuration`. Legarea sursei depinde de modul obiectul recognizer care realizează identificarea efectivă a input-ului și este în majoritatea cazurilor pasată ca argument al unei metode. După ce configurația a fost aleasă, se poate crea un obiect recognizer, care poate avea următoarele tipuri:

1. LiveSpeechRecognizer - folosește un microfon pe post de sursă a sunetului
2. StreamSpeechRecognizer - folosește un InputStream pe post de sursă a sunetului (poate fi un fișier, un socket sau un byte array)

Chiar dacă folosirea unui LiveSpeechRecognizer nu introduce overhead-ul adițional cauzat de lucrul cu fișiere am ales folosirea celei de-a doua variante (StreamSpeechRecognizer) în cadrul dezvoltării proiectului din următoarele motive:

1. Persistența înregistrării - Permite reutilizarea ulterioară a înregistrărilor
2. Statistici - Ușurintă în construirea de statistici și evaluări pe fișierele deja existente, nemaifiind nevoie de o înregistrare nouă pentru fiecare test
3. Debugging - Se pot testa funcții cu diferiți parametrii pe același fișier și observa pentru ce valoare funcționează cel mai bine (de exemplu funcția de eliminare a zgomotului de fundal, care este prezentată în detaliu în secțiunea 5.3.3)

Pentru a ne asigura că tot fișierul a fost citit, se va apela funcția `getResult` până când se ajunge la capătul fișierului. Acest lucru se poate realiza ușor prin utilizarea unei bucle while:

```
while ((result = recognizer.getResult()) != null) {
    System.out.println(result.getHypothesis());
}
```

Pe lângă metoda getHyphotesis, care întoarce traducerea completă a segmentului audio, în cadrul proiectului am mai folosit și metodele:

- `getNBest` - primește ca argument un număr întreg n, și returnează cele mai bune n interpretări ale segmentului audio ce se dorește a fi recunoscut
- `getWords` - întoarce o listă a tuturor cuvintelor recunoscute, iar pentru fiecare cuvânt fiind atașată încrederea că rezultatul returnat de program este cel corectă, și foarte important conține de asemenea și marcajele temporale pentru începutul și sfârșitul cuvântului, care este unul din cele două modalități de împărțire a cuvântului în cuvinte (mai multe despre implementare în secțiunea 5.3.2)

5.2.2 Deepspeech

Spre deosebire de Sphinx, implemntarea sistemului Mozilla DeepSpeech este realizată în pythpon un model antrenat folosind tehnici de învățare automată, acesta fiind bazat pe articolul de cercetare "Baidu's Deep Speech" [2]. Există două modalități de utilizare a Mozilla DeepSpeech, și anume folosind:

- pachetul din python
- clientul din linia de comandă

Ambele metode sunt foarte ușor de folosit însă utilizarea pachetului din python are avantajul că nu necesită salvarea de fișiere local. Un exemplu de utilizare (dorim să traducem fișierul audio_input.wav):

```
//folosind clientul din linia de comanda
./deepspeech models/output_graph.pbmm models/alphabet.txt models/lm.binary
models/trie audio_input.wav
```

Este de remarcat că atât în cazul pachetului de python, cât și în cel al clientului din linia de comandă este necesară descărcarea modelului pre-antrenat disponibil în pentru a fi folosit, sau chiar antrenarea și ulterior folosirea unui model propriu.

Pentru a integra utilitarul de la Mozilla în cadrul proiectului este necesară rularea comenzii din terminal în cadrul codului scris în Java. Pentru a rula o comandă de terminal din Java, se poate folosi următoarea construcție:

```
try {
        Process p = Runtime.getRuntime().exec(command);
} catch (IOException e) {
        e.printStackTrace();
}
```

unde command este comanda pe care în mod normal am executa-o din terminal (de exemplu ls -l).

5.2.3 Împărțirea pe cuvinte

După cum am menționat și în cadrul secțiunii 4.3, funcționalitatea de împărțire a fișierului audio în cuvinte a fost adăugată din două motive principale:

1. traducerei multiple pentru cuvinte și posibilitatea alegerii
2. comparație cu abordarea folosind fișierul complet

De asemenea am găsit două modalități de implementare, acestea fiind prezentate puțin mai detaliat în continuare:

- folosind `marcajele de timp` întoarse de traducerea fișierului complet folosind Sphinx4

- introdce un overhead temporal destul de semnificativ deoarece practic face interpretarea aceluiași input de două ori

- folosind `pydub` [7], o bibliotecă de manipulare de conținut audio, disponibilă in python
  Din păcate, această metodă prezintă la rândul ei o serie de dezavantaje, prinre care se numără inconsistența de la un test la altul, acest lucru rezultând din faptul că utilizatorul trebuie să seteze doi parametrii pentru folosirea bibliotecii, și anume intensitatea maximă a zgomotului de fundal și distanța minimă de pauză în vorbire pentru a fi considerată delimitator între două cuvinte. Astfel pentru două fișiere diferite intensitatea zgomotului de fundal este diferită, iar aceleași setări nu vor funcționa pentru ambele. Chiar dacă am reușit să înlăturăm problema zgomotului de fundal prin rularea tuturor testelor în același mediu, rămâne problema vorbirii propriu-zise, mai exact în funcție de setările făcute mai multe cuvinte rostite fără o pauză suficient de mare între ele vor fi percepute de către program ca un singur cuvânt, sau invers un cuvânt rostit prea rar poate rezulta în mai multe cuvinte identificate.

  În continuare, în figura 4 avem secțiunea de cod utilizată pentru împărțirea unui fișier în cuvinte folosind pydub:

```
from pydub import AudioSegment
from pydub.silence import split_on_silence
import sys

sound_file = AudioSegment.from_wav(sys.argv[1])
words = split_on_silence(sound_file,
    # must be silent for at least half a second
    min_silence_len=300,

    # consider it silent if quieter than -16 dBFS
    silence_thresh=-20
)

for i, chunk in enumerate(words):

    out_file = "SplitAudio/word{0}.wav".format(i)
    #print "exporting", out_file
    chunk.export(out_file, format="wav")
```

Figura 4: Împărțirea unui fișier pe cuvinte

5.3 Debugging/Dificultăți

Întrucât proiectul a conținut o parte destul de semnificativă de debugging și îmbunătățiri aduse aduse implementării deja existente, am decis ca în cele ce urmează să prezint soluțiile care au îmbunătățit considerabil performanța sistemului.

5.3.1 Microfon

La început am folosit pentru a înregistra sunetul microfonul intern al laptop-ului pe care lucram și am observat că interpretările/traducerile tuturor sistemelor de recunoaștere open-source pe care le-am folosit întorceau rezultate nesatisfăcătoare, de multe ori chiar foarte depărtate de input-ul primit. Cu toate acestea, serviciile de cloud (Google Speech Recognition API) funcționau foarte bine pe aceste fișiere , returnând o traducere aproape perfectă pentru orice input primit. Din această cauză pentru mult timp am crezut că problema este cauzată de calitatea implemntării sistemelor open-source.
Eventual am ajuns însă la concluzia că problema nu provenea de la motoarele de speech recognition, ci de la modul în care se făcea înregistrearea sunetului. Acest lucru a devenit evident după ce am efectuat înregistrări audio direct de pe placa audio a laptop-ului (în principiu, sunetul înregistrat este **exact** sunetul produs de laptop - mai multe detalii despre modul în care am realizat astfel de înregisrări sunt prezentate în subcapitolul următor) și rezultatele obținute folosind această tehnică au fost surprinzător la acea vreme, extrem de bune. Am realizat astfel că o înregistrare de calitate superioară produce rezultate mai bune decât una de calitate slabă. Utilizarea unui microfon mai performant a avut un impact important și alături de alte tehnici precum eliminarea zgomotului de fundal au condus la o precizie cu mult superioară versiunii anterioare.

5.3.2 Înregistrarea de pe placa de sunet

Pentru executarea operației de înregistrare a sunetului de pe placa de sunet am urmat pașii următori:

1. detectează dispozitivul de pe care se va face captura: Comanda

`pacmd list | grep "monitor"`

va întoarce o listă a tuturor dispozitivelor de acest tip, din care va trebui ales cel aferent plăcii de sunet, spre exemplu alsa_output.pci-0000_00_1f.3.analog-stereo.monitor

2. înregistrează sunetul produs de dispozitivul selectat

   `timeout 10 parec -d alsa_output.pci-0000_00_1f.3.analog-stereo.monitor`

   `-file-format=wav AudioFiles/aux.wav`

   Exemplul de mai sus va înregistra fișierul audio AudioFiles/aux.wav. Folosind comanda timeout se poate opri înregistrarea după un număr de secunde dorit, altfel înregistrarea va continua până când procesul este terminat (omorât).

3. convertește înregistrarea la formatul dorit folosind sox, pentru proiectul nostru comanda va fi:

   `sox AudioFiles/aux.wav -r 16k -b 16 -c 1 AudioFiles/audio.wav,`

   fișierul final pe care se va realiza recunoașterea fiind AudioFiles/audio.wav.

5.3.3 Eliminarea zgomotului de fundal

După cum am menționat și în secțiunea 4.4, eliminarea zgomotelor se poate face din linia de comandă folosind exclusiv utilitarul sox. Având la dispoziție fișerul inițial nealterat (cu zgomot) audio.wav, se vor folosi următoarele comenzi pentru a îndepărta zgomotul de fundal:

```
sox audio.wav -n trim 0 1.0 noiseprof speech.noise-profile
sox audio.wav audio_clean.wav noisered speech.noise-profile 0.4
```

Prima comandă creează un profil al zgomotului de fundal care va fi scris în fișierul speech.noise-profile. În cadrul comenzii se selectează o secțiune a fișierului original (de ex: 0 1.0 - prima secundă) care în mod ideal nu ar conține sunet, însă în realitate conține zgomot.

Cea de-a doua comandă realizează eliminarea efectivă a zgomotului din fișierul de intrare (audio.wav), salvând rezultatul într-un nou fișer (audio_clean.wav). Utilizatorului îi este lăsată la alegere cantitatea de zgomot pe care îl doreste eliminat (în exemplul de mai sus 0.4, în mod default 0.5). Acest argument ia valori între 0 și 1 (valori mai mari vor însemna riscul crescut de a elimina și porțiuni utile din fișierul audio pe lângă zgomotul de fundal, iar valori mai scăzute pot conduce la păstrarea zgomotului în fișierul rezultat, însă cu o intensitate mai scăzută). După teste pentru mai multe valori ale argumentului, am observat că folosirea de valori mai mici ale argumentului produce rezultate mai bune, valori mai mari cauzând pierderi

de date utile.

5.3.4 Corectarea erorilor

O vedere de ansamblu asupra aplicației conferă o valoare ridicată acestei componente de corectare a erorilor din traduceri, și anume privind de sus proiectul este împărțit în două mari părți (partea de recunoaștere a vorbirii și obținerea a unui răspuns bazat pe graful de cunoștiințe), aceste două părți fiind independente una de cealaltă, componenta care se ocupă de obținerea răspunsului la cererea utilizatorului primind ca input ieșirea componentei de recunoaștere a vorbirii. Presupunând deci o funcționare perfectă a componentei de generare a răspunsului, orice eroare pe partea de recunoaștere a vorbirii va cauza o funcționare deficitară a întregii aplicații.
De asemenea, mai este important de menționat că în cazul de față, chiar și cea mai mică eroare poate avea un impact esențial asupra aplicației, spre exemplu o modificare/ adăugare/ eliminare a unei singure litere dintr-un cuvânt poate schimba înțelesul unei întregi fraze, conducând astfel la o situații pentru care nu se poate găsi un răspuns. Din cauza acestor considerente, o componentă precum cea de corectare a erorilor a devenit una obligatorie în cadrul proiectului.
Implementarea este una destul de facilă, forțând execuția programului să rămână blocată într-o buclă până când interpretarea fișierului de intrare va fi cea corectă, deci toate erorile vor fi corectate. Acest lucru se realizează folosind trei pași simpli, prezentați în continuare:

1. Întreabă utilizatorul dacă varianta curentă este corectă: în caz afirmativ se va ieși din buclă și se va continua cu generarea răspunsului, în caz contrar se va continua cu pasul 2.
2. Utilizatorul înregistrează un nou fișier audio ce va avea formatul "not word1, word2", unde word1 este cuvântul greșit ce se dorește a fi înlocuit, iar word2 este cuvântul corect.
3. Se interpretează noul fișier audio, după care utilizatorul este interogat dacă rezultatul este cel corect: în caz afirmativ se va înlocui cuvântul "word2" cu "word2" în cadrul traducerii curente, și se va continua cu pasul 1, in caz contrar, se va repeta pasul 3.

5.4 Text to speech

Utilitarul pico2wave cu care am realizat funcția de generare a vorbirii este foarte ușor de folosit și, spre deosebire de majoritatea tool-urilor de text-to-speech din linia de comandă va înregistra un fișier temoral, care poate ulterior fi redat folosind comanda "aplay" spre exemplu. De asemenea fișierul poate fi șters după redare, precum în exemplul următor:

```
pico2wave -w /tmp/test.wav "Your text"
aplay /tmp/test.wav
rm /tmp/test.wav
```

5.5 GUI

Interfața grafică a aplicației este una extrem de simplistă, conținând două meniuri derulabile de tipul `JComboBox` utilizate pentru selectare modului de înregistrare a fișierului de intrare (Microphone / Laptop) și de asemenea pentru alegerea motorului de recunoaștere vocală folosit (Sphinx4, Mozilla, Google). Pe lângă acestea, interfața prezintă un buton pentru începerea, respectiv terminare înregistrării.

Stările prin care trece aplicația într-un ciclu de rulare sunt:

- în starea inițială, utilizatorul alege opțiunile dupa care apasă butonul "Start Recording!", figura 5

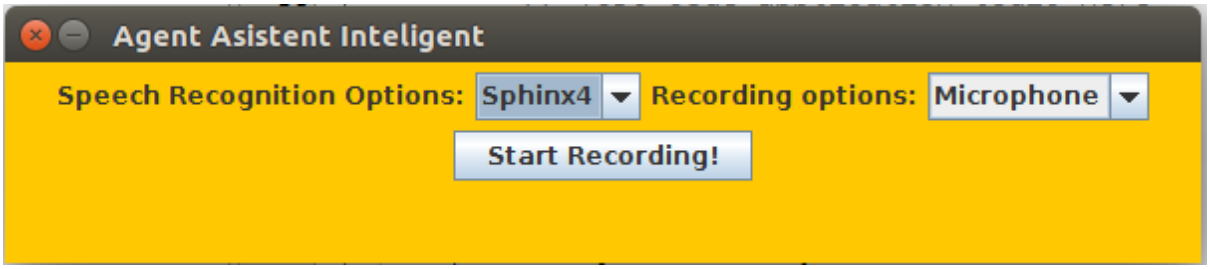


Figura 5: Interfața până la începerea înregistrării

- butonul își schimbă valoarea în "Stop Recording!"(figura6), activându-se modulul de înregistrare audio

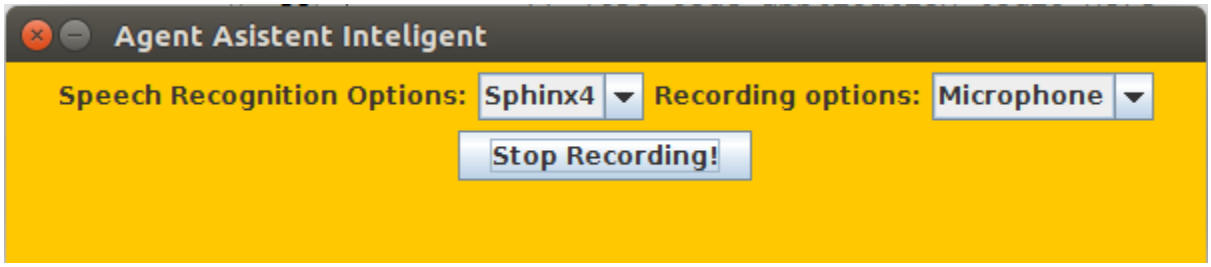


Figura 6: Interfața după începerea înregistrării

- după ce utilizatorul termină de vorbit, apasă butonul "Stop Recording!", terminând înregistrarea fișierului și semnalând startul pentru modulul de recunoaștere vocală.
- interfața se blochează până la primirea răspunsului(pe cale vocală), moment în care butonul își schimbă înapoi valoare în "Start Recording!", iar aplicația întorcându-se în starea inițială

6 EVALUARE

În cadrul secțiunii curente, se vor prezenta comparativ performanțele aplicației obținute în urma utilizării soluțiilor locale/open-source (Spjinx4 și Mozilla DeepSpeech) cu cele obținute folosind motoare de recunoaștere vocală din cloud (Google Speech Recognition API). Comparația va fi realizată atât din punctul de vedere al performanțelor temporale (eficiența/timpul de execuție al fiecărei metode în parte), însă și mai important, din punctul de vedere al rezultatelor obținute, prin compararea preciziei de identificare a cuvintelor pentru fiecare tehincă. Metrica folosită pentru obținerea eficiențelor este timpul de execuție pentru fiecărui motor de speech-recognition exlusiv în bucla de identificare a fișierului audio, iar metrica utilizată pentru obținerea acestor preciziei de identificare este numărul de cuvinte identificate corect de împpărțit la numărul total de cuvinte.Pentru ca această metrică să aibă sens, trebuie să definim ce înțelegem prin cuvânt tradus greșit. Vom avea 3 cazuri în care considerăm că un cuvânt nu a fost interpretat corect (vom folosi un exemplu pentru o mai bună înțelegere, propziția corectă fiind "Recommend me **a** restaurant which serves italian food."):

1. cuvânt shimbat - cuvânt diferit față de cel din traducerea corectă a propoziției:
   Propoziția identificată: "Recommend me **the** restaurant which serves italian food.", cuvântul "a" schimbat cu "the".
2. cuvânt omis - cuvânt din traducerea corectă a propoziției care lipsește în interpretare:
   Propoziția identificată: "Recommend me restaurant which serves italian food.", lipsește cuvântul "a"
3. cuvânt adăugat - cuvânt în plus față de traducerea corectă a propoziției:
   Propoziția identificată: "Recommend me a **the** restaurant which serves italian food.", cuvântul "the" este în plus față de traducerea corectă.
   În această categorie am considerat și cazurile în care un cuvânt este identificat greșit ca fiind de fapt două cuvinte, spre exemplu: "Recommend me a rest rant which serves italian food.", situație care este considerată de asemenea a fi **o singură** eroare.

Acestea fiind metricile folosite în cadrul evaluării, în continuare sunt prezentate ipostazele în

care au fost realizate testele, detalii despre conținutul acestora și evident rezultatele obținute.

6.1 Testele

Testele conțin 18 fraze, însumate însemnând 150 de cuvinte. Frazele au fost alese dintr-un set de comenzi uzuale folosite în cazul asistenților inteligenți. Frazele au fost alese încât să conțină atât cuvinte foarte uzuale, precum și cuvinte puțin mai complicate, sustantive proprii, ora, chiar și cuvinte abreviate.

6.2 Testarea

Testarea a fost realizată folosind niveluri de zgomot diferite, și anume într-o cameră cu geamul închis(nivelul de zgomot va fi mai redus) si aceeași cameră cu geamul deschis (nivelul de zgomot va crește). Pentru fiecare din aceste două cazuri, înregistrarea fișierului audio va fi realizată folosind:

- un microfon, cu zgomot de fundal
- un microfon, cu zgomotul de fundal eliminat
- înregistrarea sunetului produs de laptop

Am testat de asemenea și influența argumentului care stabilește nivelul de cantitatea de zgomot eliminat din fișierul inițial. Deoarce acest test necesită un număr mai mare de rulări (pentru fiecare valoare a parametrului), am decis să folosesc doar 4 din cele 18 fraze de mai sus.

Pentru testare am folosit vocile puse la dispoziție de `https://www.naturalreaders.com/`.

6.3 Rezultate

În continuare, în tabelul 1 sunt prezentate rezultatele obținute pentru primul test menționat în secționea precedentă (nivelul de zgomot redus), în tabelul 2 se regăsesc rezultatele în vazul unui nivel de zgomot de fundal, ridicat, iar în tabelul 3 sunt rezultatele testului în care variem valoare parametrului pentru reducera zgomotului de fundal.

Concluziile trase în urma analizei rezultatelor sunt prezentate mai jos. La sfarșit se găsește și un grafic cu timpii de execuție pentru fiecare dintre cele 3 motoare de recunoaștere vocală în funcție de durata fișierului de intrare, urmat de observatii asupra rezultatelor.

Tabela 1: Microfon, zgomot redus (arg=0.3). Rezultat = număr cuvinte corecte / număr total de cuvinte(Procentaj %)

Tip test	Sphinx	Mozilla	Google
microfon, cu zgomot de fundal	116/150 (77.33%)	116/150(77.33%)	149/150(99.33%)
microfon, cu zgomotul de fundal eliminat	107/150 (71.33%)	112/150(74.66%)	148/150(98.66%)
înregistrare laptop	132/150 (88%)	120/150(80%)	149/150(99.33%)

Tabela 2: Microfon, zgomot puternic (arg=0). Rezultat = număr cuvinte corecte / număr total de cuvinte (Procentaj %)

Tip test	Sphinx	Mozilla
microfon, cu zgomot de fundal	88/150 (58.66%)	149/150(99.33%)
microfon, cu zgomotul de fundal eliminat	91/150 (60.66%)	149/150(99.33%)
înregistrare laptop	120/150 (80%)	149/150(99.33%)

Tabela 3: Variație parametru pentru reducearea zgomotului. Rezultat = număr cuvinte corecte / număr total de cuvinte (Procentaj %)

argument	Mozilla
0	20/39 (51.28%)
0.1	19/39 (48.72%)
0.2	15/39 (38.46%)
0.3	9/39 (23.07%)
0.4	1/39 (2.56%)

Concluzii evaluare:

- Sphinx4 este superior Mozilla DeepSpeech doar în cazul în care nu există zgomot de fundal (înregistrare sunet de pe laptop), după cum se observă rezultatele de pe coloana 3 a tabelului 1
- Sphinx4 este foarte sensibil la zgomotul, cu cât creștem intensitatea zgomotului de fundal, cu atât va funcționa mai slab
- În cazul unui zgomot de fundal puternic, precum un ventilator, în apropierea microfonului, Sphinx4 nu a mai găsit nicio potrivire pentru orice input testat, de aceea l-am exclus din tabelul 2
- eliminarea zgomotului de fundal nu îmbunătățește rezultatele în cazul în care acesta intensitatea zgomotului este scăzută, ci dimpotrivă s-au obținut rezultate mai slabe față de testele pe versiunea nefiltrată a fișierului de intrare. Acest lucru este cauzat de faptul că, odată cu eliminarea zgomotului de fundal, există riscul de a elimina și porțiuni utile ale fișierului audio, astfel scăzând și șansele unei traduceri mai bune.
- în caz contrar pentru testul 2, precizia identificării a crescut odată cu eliminarea zgomotului, deși nu cu mult, o privire mai detaliată arătând că există teste pentru care această tehnică dă rezultate mai bune, cât și teste pentru care rezultatele sunt mai slabe.
- s-a constatat că Mozilla funcționează mai bine pentru majoritatea cazurilor realiste(avem zgomot)
- altă observație interesantă fiind legată de modul în care se produc erorile în cazul mo-

torului de recunoaștere vocală de la Mozilla: majoritatea erorilor sunt cauzate de concatenarea cuvintelor de după un anumit moment din cadrul traducerii. Din câte am putut să observ acest comportament este cauzat de o eroare apărută în traducere care se propagă până la sfârșitul acesteia. Dacă am reuși cumva să eliminăm acest tip de eroare pentru Mozilla precizia detecției ar crește semnificativ, ajungând probabil în jurul valorii de 85% sau chiar mai sus. Tratarea acestui caz poate fi subiectul unei dezvoltări ulterioare a proiectului.

- din rezultatele din tabela 3 se observă că odată cu creșterea valorii parametrului, precizia detectării va scădea, lucru care era de așteptat ținând cont ca testul a fost efectuat într-un mediu cu zgomot redus, și cunoscând faptul că valori ridicate ale parametrului pot elimina informația utilă a fișierului.

În continuare, în figura 7 este prezentată evoluția timpilor de execuție pentru Sphinx, Mozilla și Google.

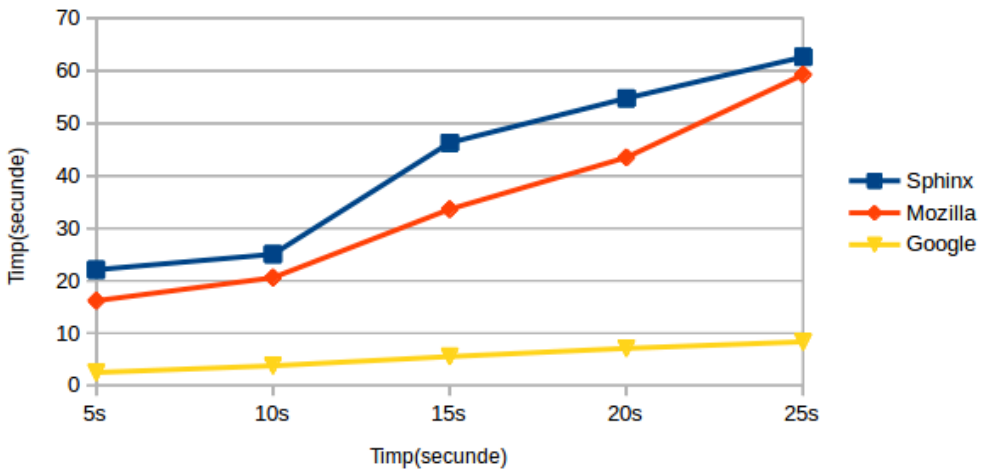


Figura 7: Timpii de execuție în funcție de dimensiunea fișierului de intrare

Se observă:

- Timpii obținuți la Google sunt mult superiori atât Sphinx cât și Mozilla
- Mozilla este puțin mai rapid decât Sphinx
- Din punctul de vedere al timpilor, Google face recunoașterea într-un timp de 2-3 ori mai scurt decât lungimea fișierului de intrare, pe când atât Sphinx cât și Mozilla realizează același lucru într-un interval de timp de până la 6-7 ori mai mare.

7 CONCLUZII

Lucrul la proiectul de licență a presupus, în primul rând o etapă îndelungată de cercetare a metodelor existente open-source de procesare și recunoaștere a vorbirii, urmată de o perioadă și mai lungă de testare a metodelor care am considerat că ar putea să ne ajute în vederea atingerii scopurilor finale ale proiectului, pentru ca în cele urmă să urmeze etapa de debugging / îmbunătățire a performanțelor.

Implementarea blocului de prelucrare și generare a vocii a presupus implementarea și dezvoltarea următoarelor module, enumerate în ordinea utilizării lor în cadrul aplicației:

- modul de GUI - interfața proiectului
- modul de înregistrare a fișierelor audio - are ca output comanda utilizatorului în format vocal
- modul de eliminare a zgomotului - primește fișierul nealterat și formează la ieșire un nou fișier cu nivelul de zgomot diminuat.
- modul de recunoaștere a vorbirii - componenta centrală a proiectului - prelucrează ieșirea modulului precedent(eliminarea zgomotului) și produce o comandă în format text
- modul de corectare a erori - se asigură că mesajul final trimis blocului de generare a răspunsului este corect.
- modul de generare a vorbirii - transformă răspunsul generat din format text în format vocal.

Utilizând toate modulele prezentate mai sus am atins obiectivele inițiale ale proiectului, și anume:

- comunicarea cu asistentul sub formă de limbaj natural în ambele direcții
- proiect open-source și local
- interfață grafică care facilitează utilizarea aplicației

Întrucât performanțele globale ale aplicației sunt strâns legate de funcționarea modulului de

recunoaștere a vorbirii, ultimul pas, cel de debugging a fost cu atât mai important, rezultatele evaluărilor confirmând încă o dată acest lucru prin dezvăluirea faptului că prin tehnici precum eliminarea zgomotului de fundal din fișierul de intrare pot îmbunătății performanța sistemului per total.

7.1 Dezvoltări ulterioare

Aceste situații asupra cărora au fost realizate testele sunt relevante pentru dezvoltarea ulterioară a aplicației deoarece utilizatorii nu ar trebui să stea să își pună problema dacă zgomotul este prea puternic atunci când doresc să folosească aplicația, aceasta fiind necesar să prezinte o performanță ridicat în orice ipostază de funcționare. Pentru realizarea acestui scop, am imaginat trei soluții, și anume:

- folosirea unui microphon mai performant care nu face înregistarea zgomotului de fundal
- folosirea de software specializat mai performant pentru eliminarea zgomotului, fără să alternăm înregistrarea vocii.
- găsirea undei modalități programatice de a realiza eliminarea erorilor în cazul sistemului Mozilla Deepspeech poate conduce la îmbunătățiri majore ale performanțelor.

BIBLIOGRAFIE

[1] Rohan Killedar Abhay Dekate, Chaitanya Kulkarni. Study of voice controlled personal assistant device. *International Journal of Computer Trends and Technology (IJCTT)*, 42, 12 2016.

[2] Jared Casper Awni Hannun, Carl Case. Deep speech: Scaling up end-to-end speech recognition. 2014.

[3] Gamal Bohouta and Veton Këpuska. Comparing Speech Recognition Systems (Microsoft Api, Google Api and CMU Sphinx). *Int. Journal of Engineering Research and Application*, 2248-9622:20–24, 03 2017.

[4] Rico Petrick Christian Gaida1, Patrick Lange. Comparing Open-Source Speech Recognition Toolkits.

[5] Doru Ciobanu. Mozilla Releases Open Source Speech Recognition Engine and Voice Dataset. `https://designmodo.com/mozilla-releases-speech-recognition-engine/`.

[6] Dave Evans. The internet of things how the next evolution of the internet is changing everything. April 2011.

[7] Theodoros Giannakopoulos. pyaudioanalysis: An open-source python library for audio signal analysis. *PloS one*, 10(12), 2015.

[8] Bret Kinsella. Amazon alexa skill count passes 15,000 in the u.s. `https://voicebot.ai/2017/07/02/amazon-alexa-skill-count-passes-15000-in-the-u-s/`, 2017.

[9] Eigenbrod L. Knote R., Janson A. The what and how of smart personal assistants: Principles and application domains for is reserach. *Multikonferenz Wirtschaftsinformatik (MKWI).*, 2018.

[10] Paul Lamere, Philip Kwok, and Walker. Design of the cmu sphinx-4 decoder., 01 2003.

[11] Tatsuya Lee, Akinobu; Kawahara. Recent development of open-source speech recognition engine julius. 10 2009.

[12] Rami Matarneh, Svitlana Maksymova, Vyacheslav Lyashenko, and Nataliya V. Belova. Speech recognition systems: A comparative review. *IOSR Journal of Computer Engineering*, 19:71–79, 10 2017.

[13] Reuben Morais. A Journey to <10% Word Error Rate. `https://hacks.mozilla.org/2017/11/a-journey-to-10-word-error-rate/`.

[14] Doug Pollack. Is your cloud data safe from government searches? `https://iapp.org/news/a/is-your-cloud-data-safe-from-government-searches/`.

[15] Daniel Povey, Arnab Ghoshal, and Boulianne. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.