Speech to text
cenzurare cuvinte

Autor: Stoian Mihai, 331AB

An: 2024

Cuprins

1. Introducere

O dată cu creșterea explozivă a conținutului audio online și a streaming-ului există o cerere tot mai mare în industria media de a modera conținutul eficient, în special în timp real. Astfel proiectul meu are că scop realizarea unui program care să fie capabil să identifice cuvintele țintă din fișierele audio.

Prima idee care a dictat domeniul în care ar putea să se aplice proiectul dezvoltat a fost strânsă legătură dintre televiziune și noțiunile învățate la materia de Aplicații Mutlimedia. Mai departe, am fost influențat de momentele când la televiziune au fost difuzate cuvinte licențioase. Acest fapt nu doar a atras amenzi asupra posturilor, dar a și expus un limbaj neadecvat telespectatorilor mai mici.

Conform [2] expunerea la un limbaj licențios poate avea consecințe grave asupra copiilor și adolescenților, deoarece expunerea în mod repetat duce la normalizarea și însușirea acestui tip de limbaj, ce poate influența negativ modul în care aceștia comuinca și interacționează.

Dată fiind natură domeniului și faptul că nu se permit întârzieri mari în ceea ce înseamnă transmiterea datelor, timpul de procesare al fișierului trebuie să fie relativ scăzut. Nu trebuie chiar să fie chiar mai mică de 50ms, asemenea răspunsului vocal în realitate virtuală [3], deoarece implementarea presupune o zona tampon în care să proceseze semnalul audio și mai apoi să fie trimis către telespectatori.

Ba chiar mai mult de atât proiectul poate fi aplicat nu doar în cadrul eminsiunilor de televiziune dar și în domeniul streaming-ului online, unde există o nevoie mare de filtrare și moderare a conținutului publicat.

Obiectivele proiectului sunt de a realiza un program care să îndeplinească această funcție și în același timp să evaluăm cum evoluează dimensiunea spațiului tampon necesar programului propus (cât de mult durează această procesare de fișier audio dacă avem mai multe sau mai puține cuvinte țintă, fișiere de dimensiuni mai mari) și propunerea unei metode de paralelizare a procesului pentru a micșora timpul de prelucrare și a aduce programul cât mai aproape de idealul de moderare a conținutului în timp real.

2. Suport tehnic

Proiectul este realizat în Python și utilizează mai multe biblioteci, cum ar fi :

- PyQt5 : pentru realizarea interfeței grafice
- speech_recognition [16], soundfile[15], sounddevice [14], pydub [13], wave [12]: pentru prelucrarea fișierelor audio și pentru determinarea textului din fișierul audio introdus

Modulul de speech_recognition a fost utilizat că fiind ceva black box, dar majoritatea algoritmilor de recunoaștere vocală se bazează pe un model de machine learning numit Hidden Markov Model, descris pe larg în [4] , mai în detaliu în articolul [1].

Partea dificilă și mai intersanta a fost identificarea timestampurilor, unde inițial propusesem o soluție care funcționa doar dacă debitul de cuvinte era constant pe parcursul fișierului audio. Până am descoperit acest cod[11], pe care l-am modificat și o să descriu exact cum.

```
for idx, word in enumerate(text_words):
    if word == target_word.lower():
        timestamp = idx * segment_duration_seconds / len(text_words)
        segment_timestamps.append(timestamp)
        segment_word_count += 1
```

Figure 1 Secvență determinare timestamp(soluție inițială)

A două soluție și cea pe care am rămas se bazează pe corelația dintre semnale. Asta presupune să avem cuvintele țintă deja înregistrare. După ce calculăm corelația dintre semnale o să obținem un grafic acesemanator :

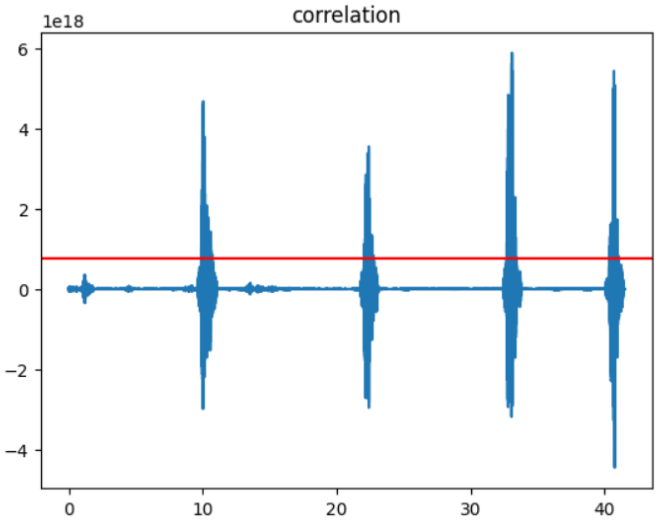

Figure 2 Exemplu grafic de corelație între semnale

În codul inițial acel threshold trebuia setat manual în funcție de ce vedea un utilizator pe ecran, eu am propus o soluție care modifică threshholdul corespunzător. Modificările pe care le-am adus au fost:

- am introdus în funcție ca și parametru numărul de cuvinte care trebuie să fie găsite în segment (deja calculăm în altă funcție de câte ori apar)
- am introdus o bulcă while cu condiția True cu o condiție de oprire și 2 condiții de reiterare a while-ului care modifică threshholdul în funcție de câte cuvinte am găsit comparativ cu câte trebuia:

```
while(True):
    timestamps=[]
    peaks = signal.find_peaks(
        z,
        height = height - step,
        distance=50000
    )
    peaks_idxs = peaks[0]
    for i, peak_idx in enumerate(peaks_idxs):
        start = (peak_idx-snippet.size/2) / rate
        center = (peak_idx) / rate
        end   = (peak_idx+snippet.size/2) / rate
        # timestampurile o sa fie in secunde
        # aici am incercat sa modific sa fie putin mai devreme fie putin mai tarziu timestamp-ul
        # in principiu cele mai bune rezultate le am obtinut lasand forma de mai jos
        timestamps.append(round(center,2))
    if(len(timestamps) == timestamp_count):
        print("A reusit sa indentifice !")
        break
    elif(len(timestamps) < timestamp_count):
        height = initial_height
        step = step + (step * 0.5) / 100
        print("Caz nefericit, reincepem cautarea si marim pasul cu 0.5%")
    else :
        print("Caz si mai nefericit ca avem mai multe cuvinte suspecte :  scadem pasul cu 0.5%")
        height = initial_height
        step = step - (step * 0.5) / 100
    count_iter = count_iter + 1
    print(f"Count iter : {count_iter}")
    if(count_iter == 25000):
        print("Nu am putut identifica timestampurile !")
```

Figure 3 Secvență modificată algoritm identificare timestamp

Un lucru similar cu ceea ce am utilizat eu este Whisper un sistem ASR (Automatic Speech Recognition) dezvoltat de cei de la OpenAI, conform acestora în [5], sistemul este antrenat pe 680,800 de ore de date din diferite limbi, dar chiar și limbi diferite în același timp.

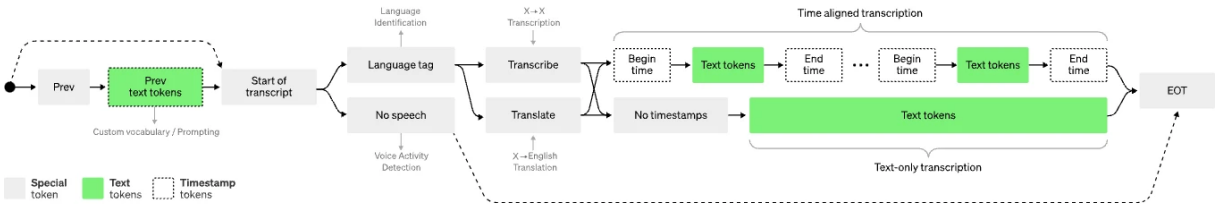

Figure 4 [4] Whisper Arhitecture

Înainte de a defini ceea ce am urmărit eu prin posibilitatea de a procesa în paralel date din cadrul programului, o să definim ceea ce înseamnă procesarea paralelă în sine.

Conform [9] , multiprocesarea este definită : “the division of a process into different parts, which are performed at the same time by different processors în a computer” . Pe lângă cele menționate anterior în dicționarul Oxford mai putem să și menționăm că procesarea în paralel maximizează utilizarea resurselor și aceelereaza timpul de calcul în sistemele informatice.

Utilizând informațiile din [10], există mai multe tipuri de paralelizare, dar sunt bazate pe una din cele 2 tehnici :

- SIMD (Single Instruction Multiple Data)
- MIMD (Multiple Instruction Multiple Data)

Cel care ar fi mai potrivit pentru programul meu ar fi bazat pe SIMD, deoarece doresc să aplic același set de instrucțiuni pe diferite date (segmentele audio din fișierul meu). Pentru paralelizarea procesului am putea trimie pe fiecare procesor un segment și un segment de tranziție pentru a accelera procedura de identificare. În domeniul transmisiunilor se dorește o întârziere cât mai mică între emisie și utilizator, deoarece dăunează calității serviciului oferit, deci un timp cât mai mic de procesare ar asigura că nu se întâmplă așa ceva.

3. Prezentarea tehnica

Primul lucru pe care am putea să-l facem pentru a accelera procesul este de a împărți fișierul în secvențe, care mai departe o să fie referite că segmente (segments). Acest lucru ridică o problema. Să presupunem că avem un fișier de un minut. Dorim să îl împărțim în segmente de câte 30 de secunde, pe care să le procesăm în paralel. Astfel algoritmul nu va mai identifica un cuvânt țintă, dacă acesta se află fix la limita dintre 2 segmente. Problema este redactată grafic și în ilustrația de mai jos:

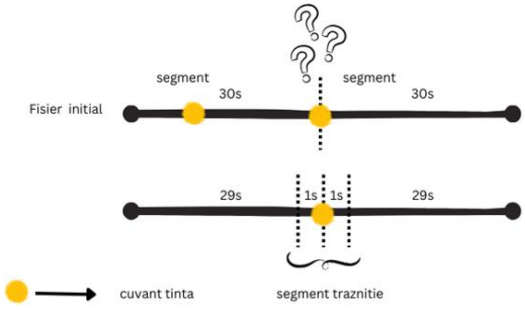

Figure 5 Creearea de segmente de tranzitie

La finalul secțiunii de Prezentare tehnică se află și o diagramă care descrie vizual ceea ce este descris mai jos. Căsuțele dreptunghiulare reprezeinta obiecte, în cazul nostru fișiere, iar căsuțele cu colțuri rotunjite reprezeinta funcții apelate / acțiuni.

Putem soluționa această problema prin creearea unui segment de tranziție (transition segment), care să fie practic ultimele secunde din primul segment alăturate primelor câteva secunde din segmentul 2.

Programul nostru dispune și de o interfață grafică care permite captarea de semnale audio, dar o să presupunem că avem deja un fișier audio cu un identificator unic (dată și ora la care a fost salvat), pe care urmează să îl prelucrăm.

Fișierul va fi împărțit în segmente de 30 de secunde (care vor fi salvate în folderul batches), pe baza acestora se vor crea și segmentele de tranziție(care vor fi salvate în folderul T_Batches). Se va aplică asupra ambelor tipuri de segmente funcția count_word_and_timestamps și vor rezultă 2 liste pentru fiecare tip de segment. Lista timestamps care reține momentele de timp la care a apărut unul din cuvintele țintă și lista word_count care reține de câte a apărut un cuvânt țintă în fiecare din fișierele prelucrate.

Pe baza listei word_count se decide dacă se mai cenzurează fișierul corespunzător poziției din lista sau nu (cenzură se realizează cu un sunet de 3 secunde luat de aici [6]). În cazul în care nu acesta este pur și simplu mutat în folderul Censored, alfel mai întâi o să se aplice funcția de cenzurare și apoi o să fie salvat aferent în folderul Censored. Am ales în final un sunet de 3 secunde, deoarece prin încercări cu suntete de 1 secundă, 2 secunde, nu reușeau să acopere mereu cuvântul.

După ce am ajuns în folderul în care avem toate fișierele cenzurate se pune problema cum o să reconstruim noi fișierul audio. Dacă avem un segment de tranziție în care s-a spus un cuvânt țintă, segmentul de dinaintea acestuia și cel de după vor trebui să fie retușate prin eliminarea secvenței care a fost utilizată la construirea segmentului de tranziție. Dacă nu s-a găsit cuvânt țintă în segmentul de tranziție acesta este ignorat și se vor alipii cele 2 segmente care au fost deja cenzurate.

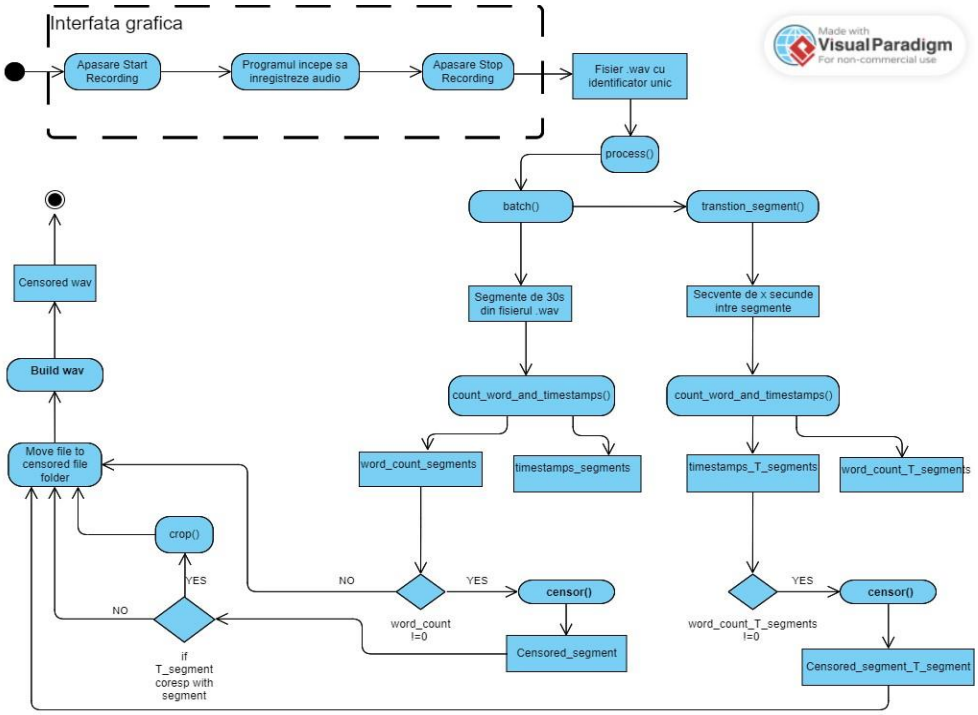

Figure 6 Diagrama descriere functionalitate program

4. Prezentare mod de utilizare

După cum am menționat și anterior programul dispune de o interfață grafică pentru a face proiectul mai interactiv și pentru a înlătura diferite situații care credeam că o să apară, Spre exemplu din experiență cu speech to text (caption-urile autogenerate de youtube) am presupus că o să fie destul de greu să recunoască cuvintele cu diacritice, dar m-am înșelat și am fost suprins de cât de bine recunoaște cuvintele cu diacritice biblioteca speech_recognizer.

Inițial (înainte de a apasă butonul de start)interfață grafică dispune de :

- Un buton start, un buton de stop
- Un timer pentru a indică timpul și verifică mai ușor dacă segmentarea a fost realizată corect, am adaptat un cod pentru timer de pe GeeksForGeeks [7]

După apăsarea butonului de stop recording interfață grafică dispune de :

- Un plot al semnalului audio captat, codul a fost adaptat de aici [8]
- căsuța de text care continele mesajele trimise de cod, în cazul care nu au fost identificate cuvinte deloc în segmentele prelucrate
- în plus se resetează timerul, și se începe procesarea fișierului audio, rezulatul se găsește în folderul Censored, sub denumirea censored_file_ + identificatorul unic al fișierului audio

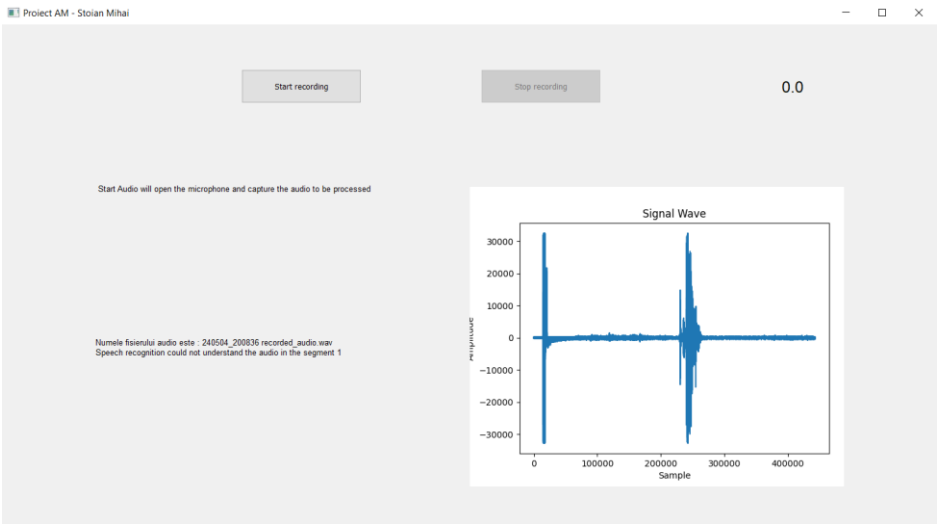

Figure 7 Interfata Grafica, dupa procesarea fisierului

5. Concluzii

Evolutie timp procesare

Pentru 20 de secunde am rulat urmatoarele experimente :

Aparitii cuvint tinta	Numar cuvinte fisier audio	Timp procesare	Aparitii omise/ cenzurare deplasata
1	10	2.57s	nu
5	10	3.54s	nu
10	10	3.92s	nu

Pentru 50 de secunde am rulat urmatoarele experimente :

Aparitii cuvint tinta	Numar cuvinte fisier audio	Timp procesare	Aparitii omise / cenzurare deplasata
20	40	19.18s	2 – aparitii omise
30	40	16.73s	2 – in plus

Experimentele de mai sus au fost realizate cu varianta a două de identificare a timpilor, și nu doar că este mai precis, ci și mult mai rapid (spre exemplu prin prima metodă un fișier de 50 de secunde îl cenzură în mai mult de 30 de secunde)

Am încercat să testez situații similare cu viață reală : dacă există o scăpare ce necesită censurare fie e ceva izolat, un cuvânt maxim 2 consecutive, fie e ceva aproape continuu care trebuie acoperit. În situațiile studiate algoritmul se comportă adecvat, dar pentru fișierele de 50 de secunde au existat erori, pentru că algoritmul identifica alte cuvinte că fiind cele țintă și nu cele țintă, deci există loc de imbunatiri la asupra algoritmului de identificare.

Există și o anomalie în timpul de execuție, dar acesta influențat în primul rând de cât ușor au fost identificate cuvintele și cât de diferite au fost. Pentru cuvinte similare trebuie să se execute mai multe iterații, deci un mai timp mare de prelucrare.

Că și viitoare direcție de dezvolare aș consideră antrenarea unui model de machine learning, care să aibă că și set de date diferite metode de pronunțare ale cuvintelor țintă și să facă identificarea pe baza această, pentru că în realitate este improbabil să avem acel sample cu cuvântul țintă care să ne ajute să identificăm apoi cuvântul într-un fișier audio.

Limitari

În prima versiune de rezolvare a identificării de timestampuri algoritmul avea o limitare care făcea impractica aplicați , deoarece debitul de cuvinte trebuia să fie constant pe parcursul segmentului audio, lucru total nerealist.

În a două versiune, pentru anumite cazuri fișierul nu poate identifica timestampul (dacă algoritmul execută acel număr de iterații și nu are success o să se încheie procesarea).

O altă limitare ar fi faptul că programul trebuie să aibă cuvântul țintă deja pronunțat de persoană respectivă pentru a putea face identificarea, cum am menționat și mai sus, dar există posibilități prin care am putea să evităm să folosim neapărat cuvântul pre-înregistrat.

Mai este necesara multiprocesarea ?

Am obținut un timp bun de procesare, și atâta timp cât putem să procesa un fișier audio de 50 în care din 40 de cuvinte 30 trebuie cenzurate, în aproximativ 20 de secunde, pare că nu ar mai fi nevoie. Dar dacă totuși am avea posibilitatea și am putea să procesăm fișierele audio și mai repede am rezolva și probleme de omisiuni de cenzură. Dacă ar există un operator uman care să asculte transmisia audio după ce a fost trecută prin program și ar vedea că a existat o omisiune, acesta să intervină și cu ajutorul altui program să cenzureze și partea aceea. Asta ar duce la o un delay mai mare între emisie și spectatori, dar ar rezolva imperfecțiunile din program. Deși nu este perfect consider că programul propus reprezintă încă o măsură de siguranță care ar reuși să prindă o bună parte din cuvintele licențioase rostite pe streamurile audio atât din televiziune, cât și pe alte platforme.

6. Referinte bibliografice

1. Lawrence R Rabiner (1989). A tutorial on hidden Markov models and selected applications in speech recognition - Proceedings of the IEEE 1989

2. Douglas A. Gentile (2003). The Impact of Media Violence on Children and Adolescents: Opportunities for Clinical Interventions - Child and Adolescent Psychiatric Clinics of North America

3. Curs 2 Aplicatii Multimedia

4. Hannes van Lier : videoclip explicativ Hidden Markov Model

5. OpenAI : Whisper an ASR system

6. BeepSound Website : https://www.soundjay.com/censor-beep-sound-effect.html

7. GeeksForGeeks – Timer : https://www.geeksforgeeks.org/pyqt5-digital-stopwatch/

8. Wavelength Plot : https://stackoverflow.com/questions/18625085/how-to-plot-a-wav-file

9. Oxford Dictionary : https://www.oxfordlearnersdictionaries.com/

10. TechTarget : https://www.techtarget.com/searchdatacenter/definition/parallel-processing

11. StackOverflow : https://stackoverflow.com/questions/67463919/how-to-find-timestamps-of-a-specific-sound-in-wav-file

Biblioteci :

12. Wave: https://docs.python.org/3/library/wave.html

13. Pydub/AudioSegm : https://audiosegment.readthedocs.io/en/latest/audiosegment.html

14. Sounddevice : https://python-sounddevice.readthedocs.io/en/0.4.6/

15. Soundfile : https://python-soundfile.readthedocs.io/en/0.11.0/

16. Speech_recognition : *Zhang, A. (2017). Speech Recognition (Version 3.8) [Software]. Available from https://github.com/Uberi/speech_recognition#readme.*