Vrednovanje jezičnog alata za transkripciju govora: ParlaSpeech-HR ASR model

Porupski, Ivan

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Humanities and Social Sciences / Sveučilište u Zagrebu, Filozofski fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:131:019095

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-10-04**

Sveučilište u Zagrebu Filozofski fakultet University of Zagreb Faculty of Humanities and Social Sciences

Repository / Repozitorij:

ODRAZ - open repository of the University of Zagreb Faculty of Humanities and Social Sciences

zir.nsk.hr

dabar DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

Sveučilište u Zagrebu
Filozofski fakultet
Odsjek za lingvistiku

Ivan Porupski

VREDNOVANJE JEZIČNOG ALATA ZA TRANSKRIPCIJU GOVORA: PARLASPEECH-HR ASR MODEL

Diplomski rad

Zagreb, rujan 2024.

Sveučilište u Zagrebu
Filozofski fakultet
Odsjek za lingvistiku

Ivan Porupski

VREDNOVANJE JEZIČNOG ALATA ZA TRANSKRIPCIJU GOVORA: PARLASPEECH-HR ASR MODEL

Diplomski rad

Mentor: dr. sc. Božo Bekavac, docent

Zagreb, rujan 2024.

PODACI O AUTORU

Ime i prezime: Ivan Porupski

Naziv oba studija: Računalna lingvistika, znanstveno usmjerenje fonetike

PODACI O RADU

Naslov rada na hrvatskome jeziku:

Vrednovanje jezičnog alata za transkripciju govora: ParlaSpeech-hr ASR model

Naslov rada na engleskome jeziku:

Evaluation of a language tool for speech transcription: ParlaSpeech-hr ASR model

Datum predaje rada: 28. kolovoza 2024.

IZJAVA O AUTORSTVU DIPLOMSKOGA RADA

Ovim potvrđujem da sam osobno napisao diplomski rad pod naslovom

Vrednovanje jezičnog alata za transkripciju govora: ParlaSpeech-hr ASR model

i da sam njegov autor.

Svi dijelovi rada, podaci ili ideje koje su u radu citirane ili se temelje na drugim izvorima (mrežni izvori, udžbenici, knjige, znanstveni, stručni članci i sl.) u radu su jasno označeni kao takvi te su navedeni u popisu literature.

Ime i prezime studenta:
Ivan Porupski

Zagreb, rujan 2024.

Zahvala

Htio bih se zahvaliti svojoj obitelji, pogotovo majci i sestri, svim prijateljima te nastavnom i nenastavnom osoblju koji su mi pružali konstantnu podršku kroz moj studij.

Sadržaj

0. Uvod i cilj rada 3
1. Teorijska podloga 4
1.1. Umjetna inteligencija 4
1.2. Razvoj ASR 6
1.3. Strojno učenje (ASR) 8
1.4. Arhitektura neuronskih mreža 10
1.5. Akustički model (wav2vec 2.0) 11
1.6. Jezični model (kenLM) 15
1.7. Metode vrednovanja ASR modela 16
1.7.1. WER, CER 19
1.7.2. Mjera F1 20
2. ParlaSpeech-HR korpus i model 21
2.1. ParlaSpeech-HR korpus 21
2.1.1. Analiza tekstovnog korpusa 21
2.1.2. Analiza govornog korpusa 23
2.2. ParlaSpeech-HR ASR model 24
3. Vrednovanje ASR modela 25
3.1. Rezultati WER i CER za oba modela 27
3.2. Mjera F1 za oba modela 27
4. Zaključak 29
5. Literatura 30
Sažetak 34
Abstract 35
Priloženo 36
Priloženo A – kôd za korištenje ParlaSpeechHR modela 36
Priloženo B – kôd za izradu kenLM .arpa i .bin modela, te wav2vec2WithLM. 37

Izrada ARPA modela iz .txt 37

Izrada ARPA dekodera 37

Izrada BIN modela iz ARPA: 39

Izrada BIN dekodera 39

Definiranje Wav2Vec2ProcessorWithLM_UTF8 41

Izrada kenLM modela (iz ARPA ili BIN) 42

Priloženo C – Arhitektura akustičnog (wav2vec 2.0) modela 46

0. Uvod i cilj rada

Ideje o stvaranju umjetnog života prisutne su u mnogim mitovima i legendama diljem svijeta već kroz tisućljeća. Jedan takav mit je mit o Pigmalionu, kiparu iz drevne Grčke koji je stvorio prekrasni kip kiparice od bjelokosti. Pigmalion se zaljubljuje u svoje djelo, a boginja Venera, na njegovu molbu, oživljava kip u ženu[1]. Iako je malo vjerojatno pretvoriti kip od bjelokosti u živu osobu, svakako je danas moguće napraviti stroj, računalni sustav ili program koji može oponašati pravu osobu do visoke razine uvjerljivosti. Prvi korak ka uvjeravanju je Turingov test. Turingov test, koji je predložio Alan Turing, smišljen je kao misaoni eksperiment koji zaobilazi filozofsku nejasnoću pitanja "Može li stroj misliti?" pri čemu računalo prolazi test ako ljudski ispitivač, nakon postavljanja pisanih pitanja, ne može razlikovati dolaze li pisani odgovori od osobe ili računala. (Russell i Norvig 2020:20). Mogli bismo reći da je Turingov test zamišljen da ispita nešto što bismo nazvali generalnom umjetnom inteligencijom (eng. AGI, *artificial general intelligence*) – vrsta umjetne inteligencije koja je sposobna raditi sve što i čovjek. Takva generalna umjetna inteligencija sastoji se od mnoštva manjih dijelova, odnosno sposobnosti kao što su računalni vid, razumijevanje jezika, razmišljanje, planiranje, navigacija, itd. (Domingos 2015:36). Ovaj rad fokusirat će se specifično na jednu takvu sposobnost, a to je automatsko prepoznavanje govora (eng. ASR, *automatic speech recognition* u daljem tekstu ASR). Cilj ASR je pronaći najvjerojatniji niz riječi, s obzirom na dobiveni zvuk, odnosno niz glasova te rekonstrukcija nizova bitova prenesenih preko šumnog kanala (Russell i Norvig 2020:484). Drugim riječima, olakšati komunikaciju između računala i ljudskih korisnika na dva načina: dopuštajući računalima da razumiju izgovorene naredbe i prepisivanjem teksta iz govornih izvora. Turingov bi se test mogao primijeniti na ASR – je li stroj transkribirao kao što bi čovjek? Je li stroj "shvatio" što mu je bilo rečeno te adekvatno reagirao? Probleme svakako nalazimo u raznovrsnosti jezika, govornika, narječja, slenga i dr. Ako bi neki ASR sustav mogao odlično transkribirati jedan jezik, ne možemo garantirati da će jednako dobro transkribirati neki drugi jezik ili govor nekog drugog govornika. Neiznenađujuće, češći su ASR sustavi za veće jezike, kao što je engleski, dok su alati za hrvatski jezik manje reprezentirani. Dok komercijalni alati možda nude i pokriće hrvatskog jezika, postoji li besplatna (eng. *open-source*) alternativa? Zahvaljujući grupi istraživača, sada postoji ParlaSpeech-HR – javno dostupna ASR baza podataka i ASR model (Ljubešić i sur. 2022). U svojem radu navode izuzetno dobru učikovitost modela. Nakon što se u kontekstu

[1] Ovidijeve Metamorfoze (2011:204).

ovoga rada odradila analiza v1.0 verzije korpusa, Ljubešić i sur. (2024) objavili su novi ParlaSpeech-HR v2.0 korpus[2], koji se razlikuje se po većem opsegu (ParlaMint[3] 4.0 umjesto ParlaMint 2.1). Ovaj rad provjerit će navedene tvrdnje vrednovanjem ParlaSpeech-HR modela koristeći uzorak iz ParlaSpeech-HR v1.0 korpusa.

1. Teorijska podloga

1.1. Umjetna inteligencija

Russel i Norvig (2021:19) su primjetili kako se pojmovi „umjetna inteligencija“ (eng. *artificial intelligence*, dalje u radu AI) i „strojno učenje“ (eng. *machine learning*, dalje u radu ML) često zamjenjuju u javnim diskursima. Naime, strojno učenje je grana umjetne inteligencije koja proučava sposobnost unapređenja učinka kroz iskustvo. U drugim riječima, ako određenom računalnom modelu damo neku informaciju, s obzirom na tu informaciju dobivamo predviđanje modela, koje onda može biti točno ili netočno, precizno ili neprecizno. Ako tom računalnom modelu damo povratnu informaciju na dobiveno predviđanje, model je stekao „iskustvo“ kako bi točno predviđanje trebalo izgledat za danu informaciju.

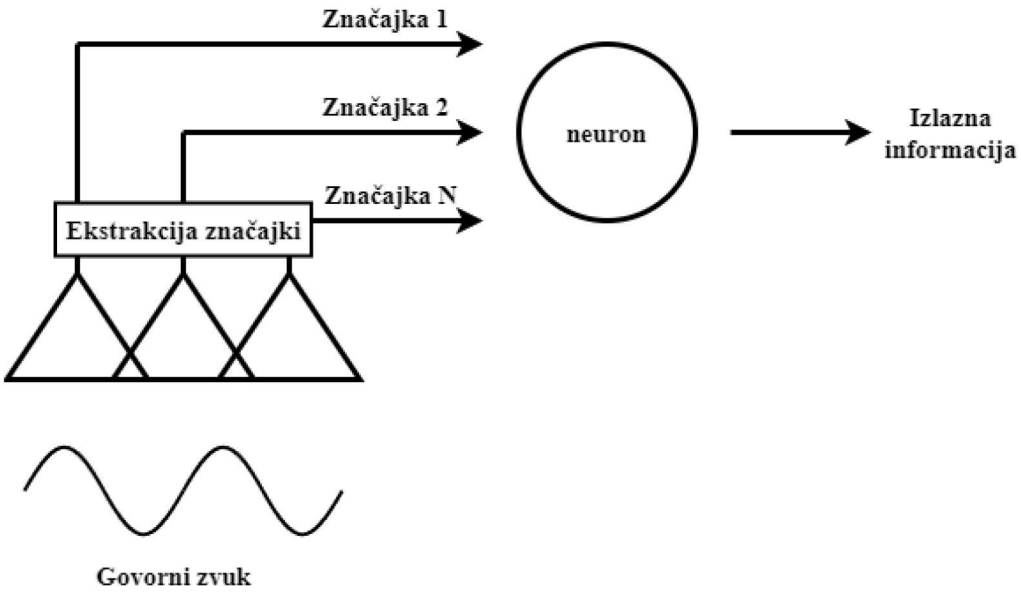
Široko gledano, AI se bavi izgradnjom, ali i razumijevanjem inteligentnih entiteta – strojeva koji mogu reagirati učikovito i sigurno na široku lepezu mogućih događaja (Russel i Norvig 2021:19). U kontekstu ovoga rada, pažnja će biti usmjerena na djelomičnu izgradnju te vrednovanje, ali ne i na opširno razumijevanje AI jezičnog alata, specifičnog za automatsko prepoznavanje govora, odnosno ASR. ASR prihvaća akustički valni oblik (govor) kao ulaz i proizvodi niz riječi (ili znakova) kao izlaz (Jurafsky i Martin 2009:318; Goyal i sur. 2018:19).

ASR spada pod *obradu prirodnog jezika* (eng. *natural language processing*, dalje u radu NLP). Goyal i sur. (2018:16) definiraju NLP kao sposobnost računala ili sistema da istinski razumije ljudski jezik i obradi ga na isti način kao što bi čovjek.

Općenito gledano, umjetne inteligencije temelje se na neuronskim mrežama (eng. *neural net*). Neuronsku mrežu čini zbir osnovnih elemenata, *umjetnih neurona* ili *perceptrona*,

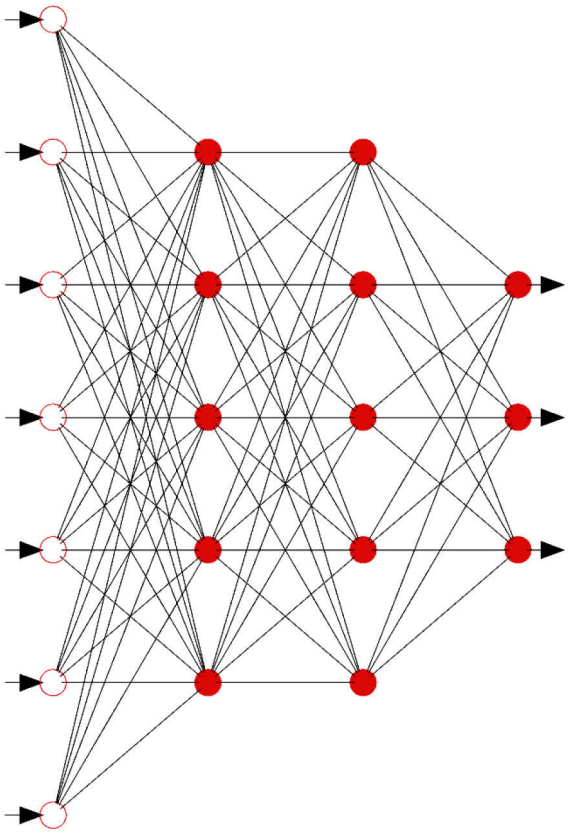
[2] https://www.clarin.si/repository/xmlui/handle/11356/1914
[3] https://www.clarin.eu/parlamint

koji je prvi razvio Frank Rosenblatt 1950-ih godina. Oni primaju više binarnih (0 ili 1) ulaza, *x1, x2, ..., xN*, i proizvode jedan binarni izlaz ako je zbroj veći od aktivacijskog potencijala (Goyal i sur. 2018:40). Drugim riječima, jedan neuron prima više ulaznih signala. Ako su ulazni signali odgovarajući, neuron se aktivira i šalje jedan binarni izlaz. Naravno, moguće je da neuron ima više različitih izlaza. Ilustracija pojednostavljenog principa prikazana je na Slici 1.



Slika 1. Shematski prikaz funkcije umjetnog neurona. Iz govornog zvuka vade se značajke. Prema N broju značajki, neuron donosi odluku koju izlaznu informaciju prosljeđuje dalje.

Neuronska mreža tipično se sastoji od ulaznog sloja (eng. *input layer*), skrivenih slojeva (eng. *hidden layers*) i izlaznog sloja (eng. *output layer*), pri čemu se najčešće veliki broj neurona u ulaznom sloju smanjuje na manji broj u izlaznom sloju. (Goyal i sur. 2018:44). Ilustracija pojednostavljene neuronske mreže je prikazan na Slici 2. Više o arhitekturi neuronskih mreža biti će u poglavlju 1.4.



Slika 2. Prikaz jednostavne duboke neuronske mreže (Praat[4], 7-5-5-3 FFNet). Ulazni sloj sastoji se od 7 čvorova (neurona), dva skrivena sloja po 5 čvorova, te izlazni sloj s tri kategorije u koje se kategorizira. Crte pokazuju međusobnu povezanost između pojedinih neurona.

1.2. Razvoj ASR

Razvoj automatskog prepoznavanja govora, odnosno ASR, obuhvaća ključne tehnološke prekretnice, od uvodnih statističkih metoda poput skrivenih Markovljevih modela do suvremenih naprednih tehnika dubokog učenja.

Ruski matematičar Andrej Markov je 1913. godine objavio rad u kojem je primijenio teoriju vjerojatnosti na analizu tekstova, posebno poezije. U tom radu, Markov je modelirao klasično djelo ruske književnosti, Puškinova "Evgenija Onjegina", koristeći ono što danas nazivamo *Markovljevim lancem* (engl. *Markov chain*). Umjesto da pretpostavi da je svako slovo generirano nasumično i neovisno o ostalima, uveo je minimum sekvencijalne strukture:

[4] Boersma i Weenink (2024)

pretpostavio je da vjerojatnost pojavljivanja svakog slova ovisi o slovu koje mu neposredno prethodi (Basharin i sur. 2004; Domingos 2015).

Skriveni Markovljevi modeli (eng. *Hidden Markov Models*, dalje u radu HMM) su Markovljevi lanci u kojima se stanja ne mogu izravno opažati, već se moraju indirektno zaključiti putem opažanja. Na primjer, u automatskom prepoznavanju govora (ASR), skrivena stanja mogu predstavljati foneme (glasovne jedinice) ili riječi, dok opažanja predstavljaju zvučne signale. HMM povezuje svako opažanje sa skrivenim stanjem, gdje jedan zvuk može odgovarati određenom fonemu, a niz zvukova može odgovarati riječima.

Jelinek (1976) te Rabiner i Juang (1986) predstavili su HMM kao osnovu za *kontinuirano prepoznavanje govora*, čime su postavili temelje za daljnji razvoj u području.

Prijelaz na *duboke neuronske mreže* (eng. *deep neural networks*, DNN) za akustičko modeliranje, kao što su opisali Hinton i sur. (2012), označio je značajan napredak u točnosti i učinkovitosti ASR sustava.

Amodei i sur. (2016) unaprijedili su modeliranje govora uvođenjem *Deep Speech 2* modela, koji omogućuje direktno mapiranje govornog ulaza na tekstualni izlaz, bez potrebe za eksplicitnim izvlačenjem značajki ili modeliranjem. Ovaj koncept uvelike je pojednostavio izradu jezičnih alata, koji su istovremeno *robustni*, odnosno mogu rukovati sa šumnim snimkama, različitim naglascima te raznim jezicima. Do ovog trenutka, modele se učilo takozvanim *nadziranim učenjem* (eng. *supervised learning*). Nadzirano učenje podrazumijeva set podataka (x, y) pri čemu su x ulazni podaci (npr. govorni segment), a y izlazni podaci (transkript govornog segmenta). Model mijenja vrijednosti svojih čvorova sve dok zadani ulazni podaci x ne rezultiraju zadanim izlaznim podacima y. S druge strane, ukoliko nemamo izlazne podatke y, to bi značilo da se radi *nenadzirano uče*nje (eng. *unsupervised learning*) (Russell i Norvig 2021; Šnajder 2021).

Schneider i sur. (2019) uvode prvu primjenu nenadziranog učenja ASR modela, *wav2vec*. Wav2vec je treniran na velikoj količini neoznačenih (eng. *unlabeled*) govornih snimki, tj. govornih snimki koje nisu popraćene tekstnom transkripcijom. Model nadmašuje Deep Speech 2 u transkripciji, koristeći dva reda veličine manje anotiranih ili označenih govornih snimki.

Baevski i sur. (2020), nastavno na prethodni rad, razvijaju *wav2vec 2.0* (dalje u radu wav2vec2). Koristeći potpuno označene snimke, model postiže odličnu učinkovitost. Ako se iskoristi svega 10 minuta označenih snimki i 53 tisuće sati neoznačenih snimki govora, moguće je dobiti vrlo dobru učinkovitost.

Takvi napredni modeli omogućuju ASR sustavima da postignu iznime rezultate u složenim i promjenjivim uvjetima govora, dok se smanjuje potreba za količinom podataka i ručnim označavanjem. Ujedno, oni olakšavaju integraciju ASR tehnologije u različite uređaje ili aplikacije, od virtualnih asistenata do sustava za transkripciju i automatsko prevođenje.

1.3. Strojno učenje (ASR)

Kako bi se mogla napraviti adekvatna provjera nekog modela, prvo je proces potrebno razložiti na korake. Općenito gledano, primjena algoritma strojnog učenja sastoji se od ovih koraka: priprema podataka, odabir modela, ekstrakcija značajki, učenje modela te vrednovanje modela (Šnajder 2021).

A) Priprema podataka ovisi o željenom cilju. Ako se želi transkribirati govor, model treba učiti na audio snimkama govora popraćene transkriptom. Jurafsky i Martin (2009:264) kratko se dotiču mogućih govornih resursa, odnosno baze podataka govora i njezine transkripcije u svrhe obrade prirodnog jezika. Spominju takozvani TIMIT korpus koji sadrži 6300 rečenica na engleskom jeziku, pri čemu je svaka rečenica u korpusu bila je fonetski označena ručno, sekvencija fonema automatski je usklađena s valnim zapisom rečenice, a zatim su automatske granice fonema ručno ispravljene. Ovakva vrsta korpusa izuzetno je poželjna jer jako dobro opisuje dinamičke elemente govora, dok je s druge strane izimno zahtjevno izradit takav korpus. Ako se želi „čitati s lica“, model treba naučiti video snimkama artikulacije govora popraćene transkriptom. Osim isključivo zvuka govora, uz naprednije tehnike, mogu se koristiti i artikulacijski korpus (npr. video snimka govornika te pokret njihovih usana, čeljusti, ruka, i sl.) (Jurafsky i Martin 2009:266).

B) Izbor modela svakim danom je sve veći. No, prije nego što se odabere model, mora se odlučiti unutar kojeg okvira (eng. *framework*) će se raditi s modelom. Naime, postoje već mnoštvo dostupnih okvira, a među najpopularnijima su *TensorFlow* (Abadi i sur. 2015, Google)

i *PyTorch* (Paszke i sur. 2019, Facebook AI). Za njih, a i mnoge druge, dostupna je dokumentacija te razni vodiči ili tečajevi[5]. Iako je wav2vec2 rađen uz pomoć PyTorch-a, svakako je moguće napraviti repliku wav2vec2 arhitekture i u drugim okvirima. Jedan takav okvir je *Open Neural Network Exchange* (dalje u radu ONNX) (Bai i sur. 2019). ONNX povezuje alate za razvoj modela s uređajima na kojima se želi koristiti model.

C) *Ekstrakcija značajki* – na ulaz algoritma strojnog učenja dovode se podaci u obliku skupa primjera. Svaki je primjer opisan kao vektor značajki (eng. feature *vector*), odnosno ključnih karakteristika koje su indikativne za identifikaciju (klasifikaciju) sličnih, budućih primjera. Šnajder (2021) nadodaje kako je u ovom koraku potrebno osmisliti na koji način prikazati primjer kao skup značajki te implementirati postupke ekstrakcije značajki te da je u većini slučajeva ovo najkreativniji korak.

Jedan takav primjer ekstrakcije značajki u govoru su *koeficijenti mel-frekvencijskog kepstra* (eng. *mel frequency cepstral coefficients*, dalje u radu MFCC). MFCC je rezultat primjene diskretne Fourierove transformacije (eng. *discrete Fourier transform*, DFT). Transformacijom se određuje količina energije u svim dijelovima spektra govora. Kako ljudsko uho nije jednako osjetljivo prema svim frekvencijama, dobiveni transformat se provlači kroz banku mel filtera (eng. *mel filter bank*), kako bi bolje reprezentiralo ljudski sluh (Jurafsky i Martin 2009:326). Drugim riječima, vrsta podataka, arhitektura modela i ekstrakcija značajki ograničeni su samo ljudskom maštom.

D) *Učenje modela* uključuje nekoliko ključnih koraka koji omogućuju modelu da nauči iz podataka i prilagodi svoje parametre za optimalne rezultate. Prvi korak je odabir arhitekture modela. Drugi korak je odabir metode optimizacije, kao što je stohastički gradijentni spust (eng. *stochastic gradient descent*, SGD), koji se koristi za minimiziranje funkcije gubitka prilagođavanjem pondera modela na temelju gradijenata izračunatih za svaki primjer u skupu podataka. *Backpropagation* je tehnika koja omogućava izračunavanje tih gradijenata i njihovo korištenje za ažuriranje pondera mreže kroz sve slojeve. U današnje vrijeme, učenje modela može biti izuzetno zahtjevno s obzirom na složenost modela i veličinu podataka, stoga se često koristi sklopovlje (eng. *hardware*) poput grafičke procesorne jedinice (eng. *graphical*

[5] Web stranica za početnike Learn PyTorch for Deep Learning (https://www.learnpytorch.io/) i YouTube video Daniela Bourkea (https://youtu.be/Z_ikDlimN6A, pristupljeno 20. srpnja 2024.)

processing unit, dalje u radu GPU) za ubrzavanje procesa obuke u odnosu na centralnu procesorsku jedinicu (eng. *central processing unit*, dalje u radu CPU), koji je znatno sporiji za ove vrste operacija. Međutim, dok CPU može obrađivati sve vrste operacija potrebne strojnom učenju, upotreba GPU-a omogućuje paralelnu obradu velikih količina podataka i bržu konvergenciju modela. Ovaj rad drži se ograničenja na isključivo CPU resurs.

E) *Vrednovanje* se izvodi nakon što je model naučen kako bi se procijenila njegova učinkovitost i preciznost. Idealna mjera vrednovanja trebala bi biti direktna, objektivna, automatski izračunata i jasno interpretabilna u odnosu na krajnju korisnost aplikaciji (McCowan i sur. 2004). Jedna od najčešće korištenih metrika u vrednovanju modela za automatsko prepoznavanje govora je *Word Error Rate* (WER), koji mjeri postotak pogrešaka u transkripciji u odnosu na referentni tekst. *Character Error Rate* (CER) sličan je WER-u, ali mjeri pogreške na razini znakova. *Mjera F1* je metrika koja kombinira preciznost i odziv, pružajući ravnotežu između tih dviju mjerenja (McCowan i sur. 2004; Derczynski 2016; Šnajder 2021). Ove metrike, među ostalima, pomažu u identifikaciji slabih točaka modela i usmjeravanju daljnjih poboljšanja. Vrednovanja koja se koriste u ovome radu više su razrađena u poglavlju 1.7.

1.4. Arhitektura neuronskih mreža

Mnogo truda u istraživanju dubokog učenja uloženo je u pronalaženje mrežnih arhitektura koje dobro generaliziraju. Doista, za svaku pojedinu vrstu podataka – slike, govor, tekst, video itd. – dobar dio napretka u izvedbi došao je istraživanjem različitih vrsta mrežnih arhitektura i variranja broja slojeva, njihove povezanosti te vrste čvorova u svakom sloju (Russell i Norvig 2021:819). Neuronske mreže mogu se podijeliti u razne vrste, no u ovom radu spomenuti će se feedforward mreža, rekurentna neuronska mreža te konvolucijska neuronska mreža.

Feedforward mreža (eng. *feedforward network*, dalje u radu FFN) najjednostavniji je tip mreže koji ima veze samo u jednom smjeru, odnosno čini usmjereni aciklički graf s označenim ulaznim (eng. *input*) i izlaznim (eng. *output*) čvorovima. Svaki čvor izračunava funkciju svojih ulaza i prosljeđuje rezultat svojim slijednicima u mreži. Informacije teku kroz

mrežu od ulaznih čvorova do izlaznih čvorova, a ne postoje petlje (Russell i Norvig 2021:802). Primjer takve mreže već je bio prikazan Slikom 2.

Rekurentne neuronske mreže (eng. *reccurent neural network*, dalje u radu RNN) razlikuju se od FFN po tome što dopuštaju cikluse u grafu računanja. Svaki ciklus u RNN-u ima kašnjenje, što omogućava jedinicama da kao ulaz koriste vrijednosti koje su prethodno izračunate iz vlastitog izlaza. Ova značajka omogućava RNN-ima da imaju unutarnje stanje ili memoriju, pri čemu unos primljen u ranijim vremenskim koracima utječe na reakciju mreže na trenutni ulaz (Russell i Norvig 2021:823).

Konvolucijske neuronske mreže (eng. *convolutional neural network*, dalje u radu CNN) su iznimno uspješni klasifikatori slika. Uz dovoljno veliku bazu podataka i dovoljno domišljatom obukom, CNN proizvodi vrlo uspješne klasifikacijske sustave, značajno bolje nego što je itko uspio proizvesti drugim metodama (Russell i Norvig 2021:1003).

RNN i CNN koriste se za različite vrste podataka zbog njihovih specifičnih arhitektura, odnosno slojeva. RNN-ovi su dizajnirani za rad sa sekvencijalnim podacima, poput teksta i govora, jer imaju povratne veze koje omogućuju pamćenje prethodnih informacija i učenje iz vremenskih serija. Ova struktura čini ih pogodnim za zadatke kao što su generiranje teksta i analiza vremenskih serija. S druge strane, CNN-ovi su optimizirani za rad s prostornim podacima kao što su slike i video, koristeći konvolucijske slojeve za automatsko otkrivanje i učenje lokalnih značajki poput rubova i tekstura. Ova arhitektura omogućava učinkovitiju ekstrakciju značajki i smanjenje broja parametara, što ih čini idealnim za zadatke poput prepoznavanja slika i segmentacije slika. Stoga, izbor između RNN-a i CNN-a ovisi o prirodi podataka i specifičnim zahtjevima analize (Russel i Norvig 2021:819)

1.5. Akustički model (wav2vec 2.0)

Akustički model u wav2vec 2.0 je model dubokog učenja dizajniran za samonadzirano učenje reprezentacija govora. U strojnom učenju, samonadzirano učenje (eng. *self-supervised learning*) se pojavilo kao paradigma za učenje općih podataka reprezentacije iz neoznačenih primjera i za fino podešavanje (eng. *fine-tuning*) modela na označenim podacima. Samonadzirano učenje koristi veliku zbirku neoznačenih govornih snimki. Imajući na umu da

ništa nije označeno, sâm model je prisiljen uočiti obrasce u bazi podataka. Model to radi uz pomoć kodirnika (eng. *encoder*), odnosno više-slojne CNN mreže (Baverski i sur. 2020). Nakon što je model naučen na velikoj količini neoznačenih podataka, fino podešavanje se izvodi uz manju količinu označenih podataka i uz *Connectionist Temporal Classification* (eng, dalje u radu CTC) funkciju gubitka (eng. *loss function*) (Baverski i sur. 2020).

Connectionist Temporal Classification (dalje u radu CTC) je tehnika koja omogućava modelima dubokog učenja, poput RNN, obradu i označavanje sekvenci podataka koje nemaju unaprijed definirane oznake ili segmentaciju. Ova metoda je posebno korisna u zadacima prepoznavanja sekvenci poput prepoznavanja govora, prepoznavanja rukopisnih znakova i drugih aplikacija gdje ulazni podaci nisu segmentirani u unaprijed definirane dijelove. Drugim riječima, model može izravno obraditi sirove, neoznačene sekvence (snimke govora) i predviđati vremenski položaj dijelova sekvenci (Graves i sur. 2006). CTC također omogućava automatskom sustavu prepoznavanja govora (ASR) da generira transkripciju iz govorne snimke, čak i kada nema jasnih granica između riječi ili glasnika. Kada se snimci govora pridruži ručno dobivena transkripcija, CTC može učinkovito sravniti (eng. *force align*) tu transkripciju sa snimkom govora. To znači da CTC određuje koji dijelovi transkripcije odgovaraju određenim dijelovima govorne snimke, čak i ako u snimci postoje šumovi, tišine ili nejasnoće. Na taj način, CTC model može naučiti prepoznavati govor iz neoznačenih podataka, dok istovremeno koristi ručno dobivene transkripcije za dodatno fino podešavanje svojih predviđanja. Ova fleksibilnost omogućava bolju prilagodbu modela na različite govorne uzorke i dijalekte te povećava točnost transkripcije, čineći CTC izuzetno povoljnom tehnikom za razvoj naprednih ASR sustava.

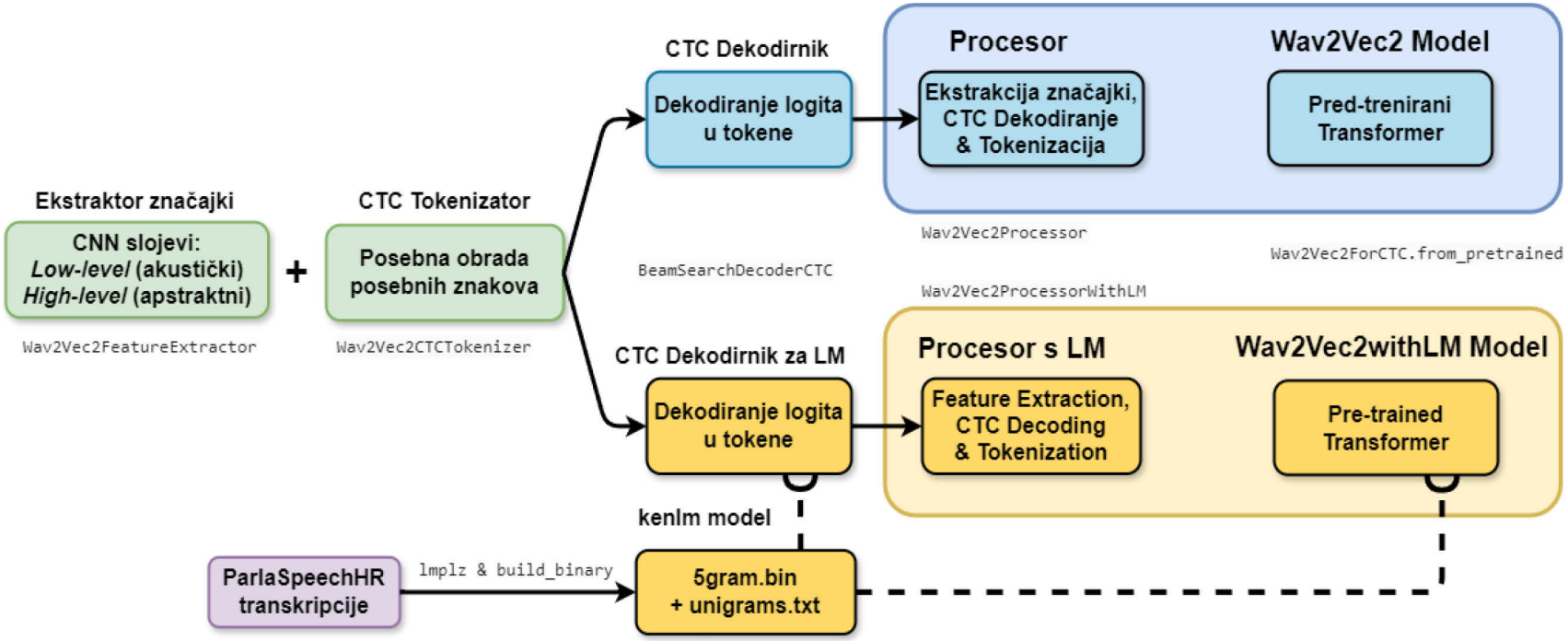
Nakon kodirnika za karakteristike, wav2vec2 koristi takozvanu *transformer-based context network*. Ova transformerska mreža dizajnirana je za modeliranje dugoročnih ovisnosti u govornom signalu. Uzima kontinuirane reprezentacije iz kodirnika za karakteristike i proizvodi kontekstualizirane reprezentacije. Ovo je ključno za povezivanje složenih ovisnosti između različitih dijelova govornog signala. Dekodirnik (eng. *decoder*) zatim uzima dobivene reprezentacije, odnosno vjerojatnosti te ih pretvara u oznake (eng. *labels*), odnosno slova.

Wav2vec2 zatim koristi *kontrastivno učenje* (eng. *contrastive learning*) za obuku modela. Kontrastivno učenje je u osnovi tehnika koja naglašava izdvajanje značajnih obrazaca iz podataka sučeljavanjem pozitivnih (sličnih) i negativnih (različitih) parova instancija.

Djeluje pod pretpostavkom da bi sve instancije koje pokazuju sličnost trebale biti blisko poredane u naučenom prostoru uležišta (eng. *embedding space*), dok bi one koje su različite trebale biti udaljenije (Chen i sur. 2020). Kako bi poboljšao učinkovitost i smanjio računalno opterećenje, wav2vec2 koristi *kvantizaciju* (eng. *quantization*). Model pretvara kontinuirane reprezentacije u diskretne reprezentacije, odnosno kao fiksni skup kodnih vektora – smanjujući veličinu modela, a time i potrošnju energije. Ova kvantizacija pomaže u smanjenju složenosti modela i čini ga učinkovitijim za naknadne zadatke (Russell i Novig 2021:896).

Izrada ovakvog modela izuzetno je pojednostavljena već spomenutim okvirima, dok je pokretanje pred-treniranog modela još lakše. Tako recimo pokretanje pred-treniranog wav2vec2 (*Large Slavic*) ParlaSpeech HR modela može se postići uz pomoć PyTorch okvira i nekoliko linija Python kôda[6] (vidi Priloženo A). Infrastruktura akustičkog (i akustičkog + lingvističkog) modela unutar PyTorch okvira prikazana je na Slici 3.

[6] https://huggingface.co/classla/wav2vec2-large-slavic-parlaspeech-hr



Slika 3. Infrastruktura wav2vec2 modela u PyTorch okviru, zajedno s integracijom kenLM modela.

N-gram modeli su probabilistički modeli koji predviđaju sljedeću riječ na temelju prethodnih N − 1 riječi. Takvi statistički modeli nizova riječi nazivaju se također jezični modeli (eng. *language model*). Izračunavanje vjerojatnosti sljedeće riječi pokazat će se blisko povezano s izračunavanjem vjerojatnosti niza riječi (Jurafsky i Martin 2009:93).

Jezični model je vjerojatnosna distribucija koja opisuje vjerojatnost bilo kojeg niza riječi u jeziku. Ovaj model omogućava procjenu koliko je određeni niz riječi, poput „Ne pada kruška daleko od drveta." vjerojatan kao gramatički ispravan izraz na hrvatskom jeziku, dok izraz poput „Drveta daleko pada ne od kruška." pokazuje vrlo malu vjerojatnost, ali nije nemoguć.

Jezični modeli pomažu u predviđanju koje riječi će se vjerojatno pojaviti sljedeće u tekstu, što omogućava predlaganje dovršetaka za e-poštu ili tekstovne poruke, kao i izračunavanja promjena u tekstu koje bi povećale njegovu uspješnost (npr. novinski članak, prijava za posao, itd.), u obliku predloženih ispravka pravopisa ili gramatike. Uz pomoć jezičnih modela možemo također odrediti najvjerojatniji prijevod rečenice ili najvjerojatniji odgovor na pitanje temeljen na primjerima pitanja i odgovora korištenim kao podaci za obuku. Jezični modeli su stoga ključni za širok spektar zadataka obrade prirodnog jezika i služe kao zajednički osnovni pokazatelj (eng. *benchmark*) za mjerenje napretka u razumijevanju jezika. Budući da su prirodni jezici složeni, svaki jezični model predstavlja svega približan prikaz stvarnosti (Russel i Norvig 2021:874).

Jedna takva aproksimacija stvarnosti jest kenLM[8] jezični model (Heatfield 2011). KenLM model koristi modificirano Kneser-Ney zaglađivanje (eng. *smoothing*) kako bi postigao veću preciznost, skalabilnost sa sve većim bazama podataka te optimizirao sâm izračun (Heatfield i sur. 2013). Ključna ideja Kneser-Ney zaglađivanja jest da se umjesto jednostavnog brojanja frekvencija pojavljivanja n-grama, uzme u obzir i broj različitih konteksta u kojima se određena riječ pojavljuje. Ovo je važno jer neki n-grami mogu imati nisku frekvenciju, ali njihovi elementi, odnosno riječi, mogu se pojavljivati u mnogim različitim kontekstima. Chen i Goodman (1999) napravili su empirijsko istraživanje u kojem su radili poredbu najčešće korištenih algoritama za zaglađivanje prilikom izrade n-gramskih

[8] https://kheafield.com/code/kenlm/ (Pristupljeno 18. srpnja 2024.)

modela. Autori uvode metodologiju za vrednovanje takvih algoritama te dolaze do zaključka kako varijacija Kneser-Ney zaglađivanja dosljedno nadmašuje ostale algoritme uključene u istraživanju.

Primjer integracije kenLM jezičnog modela zajedno s osnovnim wav2vec2 akustičnim modelom opisan je na Hugging Face blogu (von Platen 2022). Autor demonstrira, korak po korak, kako pokrenuti prethodno istrenirani wav2vec2 model, zatim kako izraditi kenLM model i spojiti ga s akustičnim modelom, te poredba transkripcije svakog. Grafički prikaz dijelova potrebnih za izradu oba modela prikazan je u Slici 3., dok Slika 4. prikazuje shemu vrednovanja istih. Koraci i kôd za izradu modela priloženi su na kraju (Priloženo B).

1.7. Metode vrednovanja ASR modela

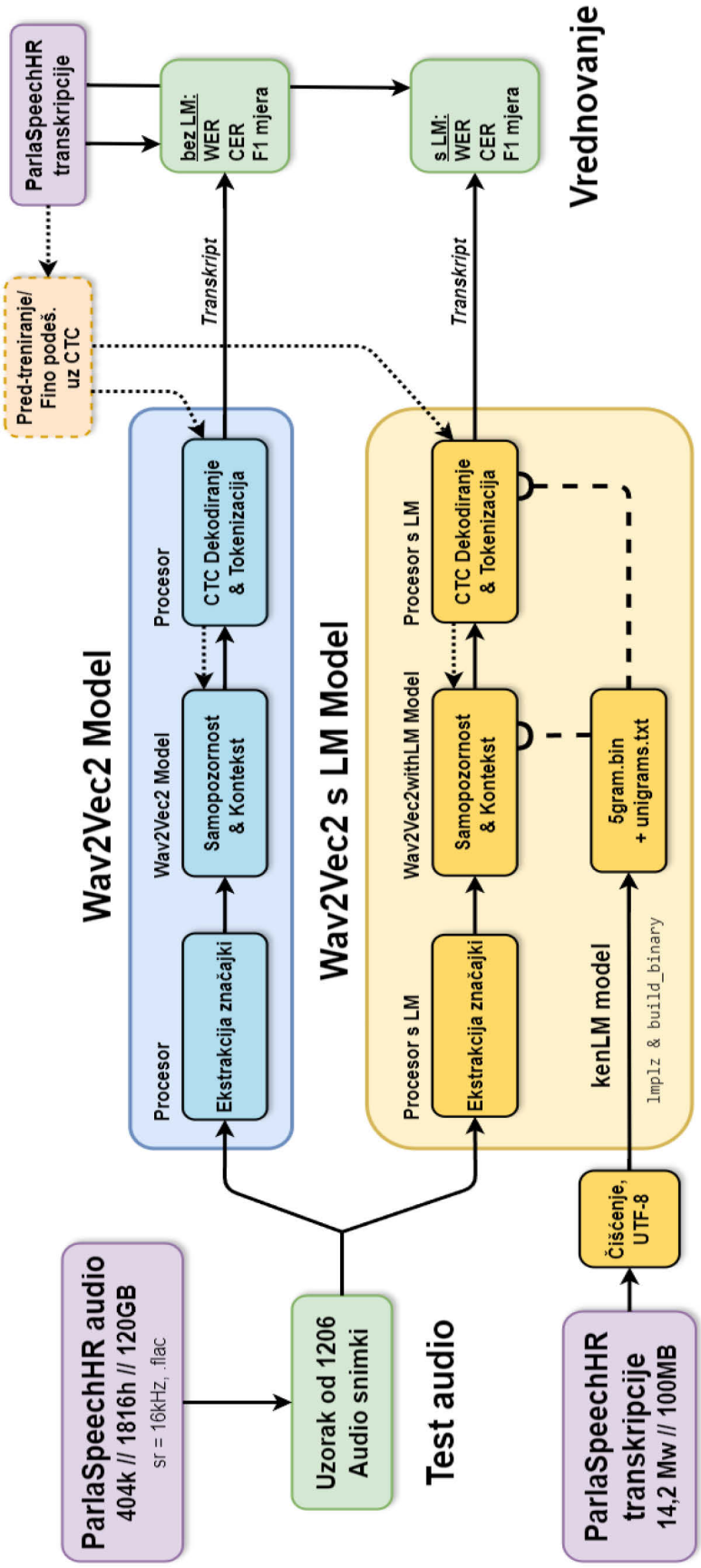
Vrednovanje ASR modela često se provodi pomoću linearnih metrika (Slika 4.). Željeni rezultat je tekstni transkript govora koji bi trebao imati visoku razinu točnosti u odnosu na stvarni zvuk govora koji se daje modelu. Iako je WER (*Word Error Rate*) široko korištena mjera za ocjenu točnosti ASR modela, McCowan i sur. (2004) ističu brojne probleme povezane s ovom metrikom, kao što su osjetljivost na različite vrste pogrešaka i otežana interpretacija rezultata u specifičnim kontekstima. Kao alternativu, predlažu F1 mjeru koja integrira preciznost i odziv, pružajući tako sveobuhvatniju procjenu performansi modela.

S druge strane, stranica Hugging Face (*Audio Course*, 5. poglavlje) opisuje WER i CER (*Character Error Rate*) kao primarne metrike za vrednovanje ASR modela. Ove metrike omogućuju procjenu preciznosti u prepoznavanju riječi i znakova, što je ključno za razumijevanje učinkovitosti modela u stvarnim situacijama. Međutim, ovisnost o samo tim metrikama može ograničiti sveobuhvatan uvid u performanse, posebno u slučajevima kada se model suočava s neurednim ili složenim govorom.

Ulasik i sur. (2020) predstavili su CEASR korpus kao standardizirani skup podataka za vrednovanje ASR modela, pružajući prvotno prikupljene i objavljene rezultate više ASR modela vrednovanih na istom korpusu. Ova standardizacija omogućava usporedbu različitih modela u istim uvjetima i minimizira probleme uzrokovane relativnom prirodom WER-a i

CER-a. Ipak, CEASR korpus sastoji se od engleskih i njemačkih govornih korpusa, što ga čini neprimjerenim za vrednovanjem hrvatskog ASR modela.

Magalhães i sur. (2022) razvili su dodatne pristupe za vrednovanje ASR modela koristeći WER zajedno s BLEU, METEOR i kosinusna sličnost (eng. *cosine similarity*), primijenjene na dvije javno dostupne baze govornih podataka. Njihova metoda za izračunavanje WER-a razlikuje se od standardne, jer uključuje sumu umetnutih, zamijenjenih i izostavljenih riječi podijeljenu s ukupnim brojem podudarenih, zamijenjenih i izostavljenih riječi. Ova detaljna analiza omogućava sveobuhvatniju evaluaciju i usporedbu šest aktualnih ASR modela, uključujući wav2vec2, za koji su naveli WER vrijednost od 10,80 %. Ovakve procjene doprinose boljem razumijevanju prednosti i ograničenja različitih ASR modela, ali istovremeno naglašavaju potrebu za dodatnim istraživanjem specifičnih prilagodbi za različite jezike i primjene.



Slika 4. Shematski prikaz vrednovanja dva wav2vec2 modela.

1.7.1. WER, CER

Word Error Rate (WER) je mjera koja se često koristi za evaluaciju sustava prepoznavanja govora, a temelji se na Levenshteinovoj udaljenosti (eng. *Levenshtein distance*), koja mjeri minimalan broj umetanja, brisanja i zamjena potrebnih za pretvaranje jednog niza u drugi. WER se definira kao odnos između ukupnog broja grešaka (zamjene, brisanja i umetanja) i broja riječi u referentnoj transkripciji. Dok je WER direktna i objektivna mjera performansi, ona ima nekoliko praktičnih nedostataka. Normalizacija prema duljini referentnog niza može otežati interpretaciju – jer WER može premašiti vrijednost 1, odnosno 100 %, što znači da može biti negativan, a samim tim i teško usporediv. Osim toga, WER ne omogućuje detaljnu analizu performansi prema važnosti riječi, jer se udaljenost između nizova teško razlaže na mjere po pojedinim riječima. Alternativne mjere, poput *Word Recognition Rate* (WRR) i *Word Correct Rate* (WCR), ne pate od ovih problema jer nude jasniju interpretaciju, iako ne uzimaju u obzir sve vrste grešaka kao što su umetanja. Ove karakteristike znače da WER možda nije uvijek prikladan za analizu u različitim aplikacijama i kontekstima istraživanja, gdje su potrebni precizniji ili dodatni okviri za evaluaciju (McCowan i sur. 2004).

Wang i sur. (2003) dolaze do zaključka kako WER nije uvijek najbolja mjera točnosti ASR sustava. Smatraju kako bi jezični model trebao biti optimiziran za razumijevanje, a ne samo prepoznavanje govora. Neovisno o manama WER mjere, u ovome radu primjenit će se WER mjera iz razloga što omogućava lakšu poredbu s drugim rezultatima.

Character Error Rate (CER) je mjera vrlo slična WER-u, no umjesto broja riječi, gledaju se brojevi znakova (slova). Korištene jednadžbe za izračun WER-a i CER-a glase:

$$WER = \frac{S + A + D}{N_{riječi}}$$

$$CER = \frac{S + A + D}{N_{slova}}$$

N = ukupan broj pojavnica (riječi ili slova)
S = broj zamjenjenih pojavnica
A = broj dodanih pojavnica
D = broj izostavljenih pojavnica

1.7.2. Mjera F1

Mjera F1 (eng. *F1 score*) je mjera koja se koristi za procjenu učinkovitosti modela u prepoznavanju pozitivnih instancija (eng. *true positives*, Tp), balansirajući između preciznosti i odziva. *Preciznost* (eng. *precision*, P) se odnosi na to koliko su točne predviđene pozitivne instancije, tj. koliko od njih stvarno pripada pozitivnim instancijama. S druge strane, *odziv* (eng. *recall*, R) mjeri sposobnost modela da ispravno identificira sve pozitivne instancije, pokazujući koliko sveobuhvatno model prepoznaje sve relevantne slučajeve. Mjera F1 predstavlja harmonijsku sredinu između preciznosti i odziva, pružajući jedinstvenu metriku koja pomaže u ravnoteži između ove dvije komponente i omogućuje cjelovitu procjenu performansi modela (Derczynski 2016; Šnajder 2021).

Pravi pozitivni:

$$T_P = N - (S + D)$$

Preciznost:

$$P = \frac{T_p}{T_p + A}$$

Odziv:

$$R = \frac{T_p}{N}$$

Mjera F1:

$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

N = ukupan broj pojavnica
S = broj zamjenjenih pojavnica
A = broj dodanih pojavnica
D = broj izostavljenih pojavnica

2. ParlaSpeech-HR korpus i model

Kako bismo temeljito procijenili učinkovitost automatskog sustava za prepoznavanje govora (ASR), važno je obratiti pažnju na tri ključne komponente: sâm ASR model, korpus podataka na kojem je model treniran, te rezultate koje model generira.

ParlaSpeech-HR predstavlja prvi veliki i javno dostupni hrvatski korpus podataka namijenjen za ASR. Ovaj korpus sadrži ukupno 1 816 sati snimljenog govora, koji su prikupljeni iz različitih izvora, uključujući parlamentarne transkripte i snimke iz ParlaMint korpusa. ParlaMint korpus sadrži službene transkripte govora iz parlamentarnih sjednica i drugih relevantnih političkih događaja, što osigurava raznolike i visoko kvalitetne podatke. (Ljubešić i sur. 2022).

Ljubešić i sur. (2022) navode sadržaj korpusa kao 403 925 unosa, pri čemu svaki sadrži (1) put do *.wav* datoteke, koja također predstavlja ID unosa. Daljnje informacije uključuju: (2) naziv izvorne YouTube datoteke, (3) početak (u milisekundama) unosa u izvornoj snimci, (4) kraj (u milisekundama) unosa u izvornoj snimci, (5) popis riječi iz ručnog transkripta, (6) lokalni vremenski pomaci za svaku riječ u ručnom transkriptu, (7) popis riječi u normaliziranom transkriptu, (8) lokalni vremenski odmaci za svaku riječ u normaliziranom transkriptu, i (9) ručno ispravljene normalizirane riječi, dostupne samo za 484 unosa, korištenih u analizi u odjeljku 4, te (10) informacije o govorniku, ukoliko je u snimci samo jedan govornik.

2.1. ParlaSpeech-HR korpus

2.1.1. Analiza tekstovnog korpusa

U svrhe vrednovanja tekstovne transkripcije, potrebno je analizirati sastav tekstovnog dijela korpusa. Analiza je bila obavljena pomoću Python alata u Jupyter Lab radnoj okolini[9]. Prvo, svi tekstni segmenti spojeni su u jedan veliki tekstni blok. Cijeli tekst prebačen je u mala slova, a zatim se blok teksta pročistio[10], uklanjajući sve simbole izuzev 30 glasnika hrvatskog

[9] https://github.com/porupski/diplomski (jupyter notebooks/NLTK_SyllableTokenizer_HR_custom_v6.0.ipynb)
[10] U ovome radu, pojmovi fonema i glasnika odnose se na isti pojam iz razloga što se analizirao tekstovni korpus koji nije uzimao u obzir varijacije svih glasnika, već je transkripcija ograničena na hrvatske foneme.

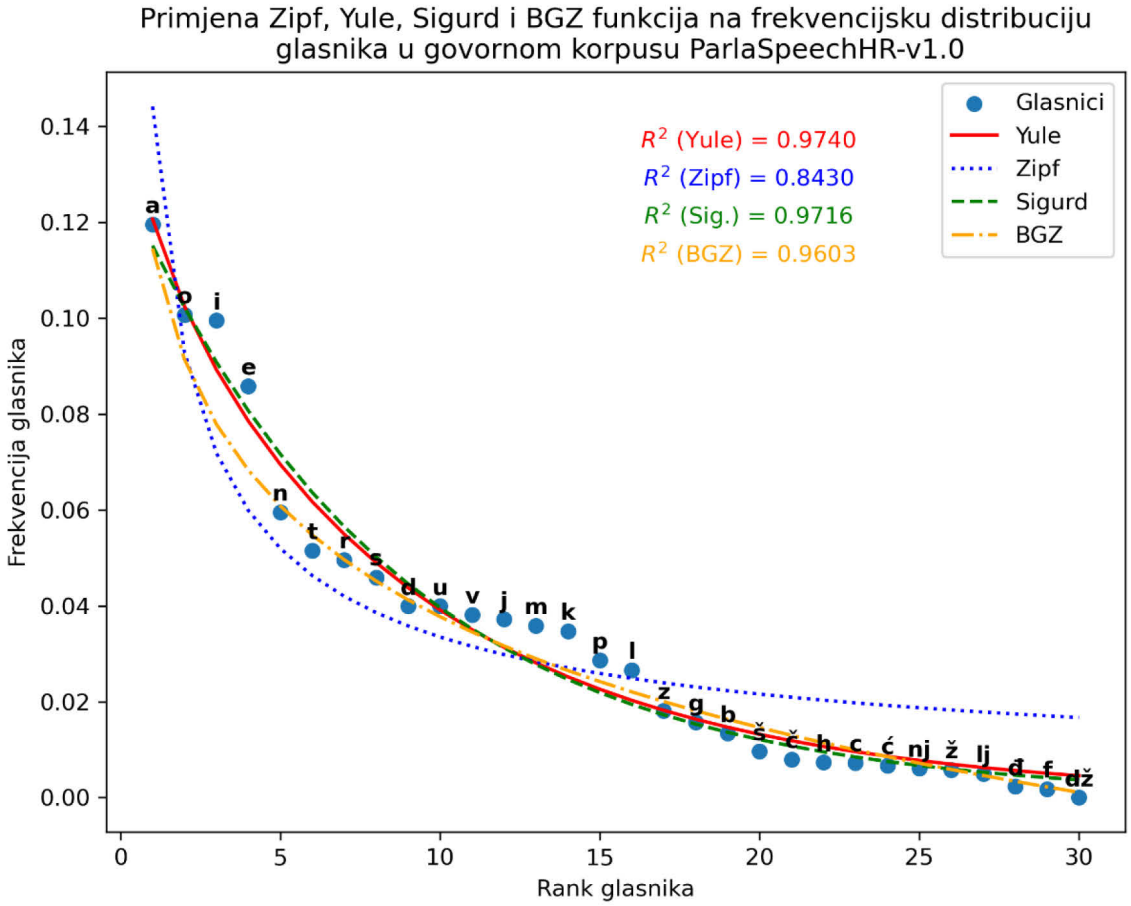
jezika. Takav pročišćeni blok teksta se zatim opojavničio (eng. *tokenize*) koristeći se javno dostupnim NLTK-om[11] (Natural Language Toolkit, funkcijom *word_tokenizer*). Pojavnice su bile opojavničene u slogove koristeći NLTK-ovu funkciju SyllabeTokenizer za koju se pripremila vlastita hijerarhija sonornosti prema Škariću (2007;88).

Ovime je ustanovljen opseg korpusa veličine 14,25 milijuna riječi, odnosno 31,67 milijuna slogova ili 70,8 milijuna glasnika. Od toga, utvrđeno je 173 225 jedinstvenih pojavnica, odnosno 9 207 jedinstvenih slogova, dok je broj glasnika, jasno, 30.

Čestotna razdioba glasnika u korpusu prikazana je na Slici 5. Kako bi se opisala distribucija riječi, ali i glasnika, često se primjenuje takozvana *Zipfova funkcija*. Osim Zipf-ove, često se spominje i *Yuleova distribucija*, dok Tambovtsev i Martindale (2007) također primjenjuju i Sigurdovu i BGZ-ovu funkciju na čak 95 jezika, ali ne uključujući hrvatski jezik. Sve četiri funkcije primijenjene su na distribuciju glasnika u korpusu ParlaSpeech-HR te je njihov koeficijent korelacije prikazan u gornjem desnom kutu. Yuleova distribucija najbolje opisuje distribuciju ($R^2 = 0{,}9740$).

Analiza čestotne razdiobe glasnika pruža nam izvanredni uvid u sastav samog korpusa, tekstovnog i glasovnog, iako glasovnog u nešto manjoj mjeri zbog nepouzdanosti transkripcije. Čestotna razdioba nam pokazuje kako se neki glasnici kao što su /đ/, /f/[12] i /dž/ javljaju izuzetno rijetko. Ova informacija može biti od koristi u slučaju kada neki ASR model nema visoku sigurnost za neki glasnik u bučnom okruženju. Na primjer, glasnici /h/ i /f/ vrlo su akustički slični (Bakran 1996), ali glasnik /h/ je preko 4 puta vjerojatni.

[11] https://www.nltk.org/index.html
[12] Damjanović (2003:71) ističe kako se glasnik /f/ pojavljuje samo u riječima stranog podrijetla, što objašnjava njegovu nisku čestoću.



Slika 5. Frekvencijska (čestotna) razdioba glasnika u korpusu ParlaSpeech-HR-v1.0.

2.1.2. Analiza govornog korpusa

Snimke su pohranjene u *.flac* formatu i uzorkovane frekvencijom (eng. *sample rate*) od 16 kHz. Nažalost, podrobna analiza govornog korpusa moguća je trenutno samo ručnim preslušavanjem svih snimaka. Čak i sami autori navode kako su koristili mnoštvo već dostupnih alata kako bi razvili prototip ASR modela, koji bi nadalje transkribirao još govornog materijala na kojem je ultimativno bio treniran završni model. Ovakav način slabo nadziranog strojnog učenja (eng. *weak supervision machine learning*) izuzetno je ekonomski i vremenski povoljan, iako kvaliteta nastalog korpusa ili modela može biti slabija od potpuno nadgledanog učenja zbog mogućih propusta.

Preslušavanjem nasumično odabranih snimki govornog materijala, kvaliteta govora govornika je uglavnom dobra, govornika se jasno i razabirljivo čuje, dok, ovisno o snimci, postoji žamor, udaranje i drugi razni zvukovi.

2.2. ParlaSpeech-HR ASR model

Arhitektura modela wav2vec2 namijenjena je automatskom prepoznavanju govora. Sastoji se od ekstraktora značajki (eng. *feature extractor*) i kodirnika temeljenog na transformerima (eng. *transformer-based encoder*). Ekstraktor značajki, izgrađen od konvolucijskih slojeva (CNN), obrađuje sirovi zvuk kako bi izvukao hijerarhijske značajke, koje se zatim projiciraju u višu dimenziju. Enkoder, koji se sastoji od 24 transformacijska sloja, usavršava te značajke kroz mehanizme samopozornosti (eng. *self-attention*) i FFN. Koristi pozicijske konvolucije i normalizaciju slojeva kako bi uhvatio kontekstualne informacije. Na kraju, linearni sloj mapira kodirane značajke na izlazni prostor, gdje model predviđa niz pojavnica ili znakova.

Ljubešić i sur. (2022) odlučili su se za XLS-R model, temeljen wav2vec2, odnosno na transformerskoj arhitekturi za više jezika, pri čemu je model već bio treniran na sirovim hrvatskim snimkama govora. Ovaj model pokazao se vrlo uspješnim u pružanju točnih rezultata transkripcije čak i uz ograničenu količinu dostupnih podataka. Korištenje transformerske arhitekture omogućilo je modelu da efikasno generalizira nepoznate govorne uzorke zahvaljujući svom kapacitetu da nauči složene jezične obrasce.

Autori su razvili nekoliko verzija modela kako bi istražili učinak različitih veličina i kvaliteta skupa podataka na učinkovitost prepoznavanja govora. Početni XLS-R model treniran je na 110 sati hrvatskog govora, dok je naprednija verzija trenirana na 300 sati govora, čime je omogućeno dublje učenje jezičnih struktura i akustičnih svojstava jedinstvenih za hrvatski jezik. Sljedeći korak bio je razvoj Slavic modela, također treniranog na 300 sati, ali s naglaskom na unaprjeđenje modela za slavenske jezike u cjelini, što omogućuje bolju usporedbu i prijenos znanja između srodnih jezika. Dodatno, finalni model Slavic + 5-gram jezični model (kenLM) integrira jezični model baziran na n-gramima koji značajno poboljšava učinkovitost u kontekstu transkripcije specifičnih jezičnih struktura.

Rezultati testiranja pokazuju različite vrijednosti za *Word Error Rate* (WER) i *Character Error Rate* (CER) za svaki od modela. XLS-R model treniran na 110 sati govora imao je najlošije performanse (WER = 10,57 %, CER = 3,23 %), što ukazuje na ograničenja modela pri korištenju manjeg skupa podataka. S druge strane, kombinacija Slavic modela i 5-gramskog jezičnog modela rezultirala je najboljim performansama (WER = 4,30 %, CER = 1,88 %) što naglašava važnost jezičnog modeliranja u poboljšanju točnosti prepoznavanja govora.

Autori pružaju poveznice na tri javno dostupna modela, omogućujući daljnje istraživanje i prilagodbu modela u različitim kontekstima upotrebe. Međutim, ističu potrebu za dodatnim istraživanjima koja bi dublje istražila povezanost dobivenih rezultata s metodama treniranja i testiranja, kao i s lingvističkim materijalima korištenim u kenLM modelu. Posebno se naglašava potreba za istraživanjem utjecaja dijalekata, akcenata i stilova govora na performanse modela, što bi moglo pomoći u daljnjem unaprjeđenju ASR tehnologije za hrvatski i druge slavenske jezike. Nadalje, autori predlažu istraživanje utjecaja različitih pristupa finog podešavanja modela kako bi se dodatno smanjile stope grešaka, posebno u uvjetima sa šumovima i varijacijama u govoru.

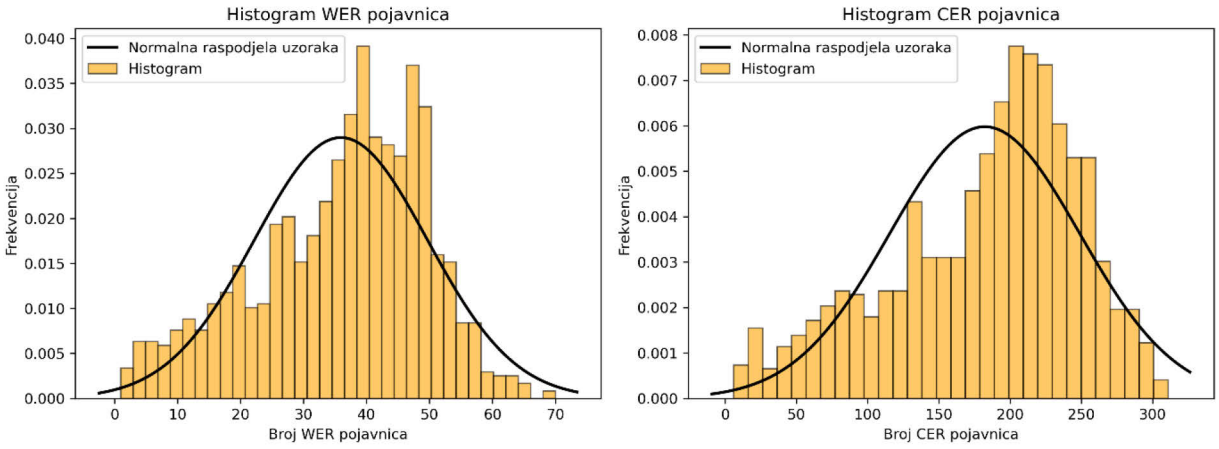
3. Vrednovanje ASR modela

Zbog tehničkih, odnosno vremenskih, ograničenja vezanih uz upotrebu isključivo CPU resursa, vrednovanje modela nije moglo biti provedeno na cijelom skupu podataka, već je provedeno na uzorku od nasumično odabranih 1 206 snimaka govora iz cijelog korpusa. Ovaj uzorak je odabran kako bi vjerodostojno prikazao cijeli korpus, uz zadržavanje statističke snage potrebne za precizno vrednovanje modela. Statističkom procjenom utvrđeno je da odabrani uzorak može pouzdano utvrditi razliku (Tablica 1.) u performansama akustičkog i jezičnog modela s visokim stupnjem pouzdanosti od 99 % (WER: t-vrijednost = 2,806, p-vrijednost = 0,005; CER: t-vrijednost = 2,998, p-vrijednost = 0,003). Ovi rezultati sugeriraju da su razlike u performansama modela statistički značajne i ne proizlaze iz slučajne varijacije u podacima.

	Broj pojavnica			
	WER(populacija)	WER(uzorak)	CER(populacija)	CER(uzorak)
prosjek	36,06	35,94	182,90	182,18
stand. dev.	13,47	13,76	65,91	66,70

Tablica 1. Prikaz prosjeka WER i CER pojavnica uz standardnu devijaciju za populaciju (korpus) i za uzorak.

Tablica 1. prikazuje statističku obradu podataka za populaciju (cijeli korpus) i uzorak, gdje su prikazane prosječne vrijednosti i standardne devijacije za WER i CER pojavnice. Na temelju ovih rezultata, prosječna snimka iz korpusa sadrži 36,06 ± 13,47 riječi (WER pojavnica), odnosno 182,90 ± 65,91 znakova (glasnika, CER pojavnica). Ovo ukazuje na to da se model relativno dosljedno ponaša i na uzorku i na cijelom korpusu, što dodatno potvrđuje validnost uzorka kao reprezentativnog. Iako, bitno je napomenuti kako se nasumičan odabir uzorka uzimao iz cjelokupnog korpusa, umjesto iz seta namijenjenog za testiranje, odnosno uzorak gotovo sigurno sadrži većinom snimke kojima model je bio treniran. Slika 6. prikazuje raspodjele pojavnica, odnosno histograme za WER i CER pojavnice.



Slika 6. Histogram za WER i CER uzorke korpusa ParlaSpeechHR-v1.0.

3.1. Rezultati WER i CER za oba modela

Za svaku pojavnicu napravljena je transkripcija pomoću modela wav2vec2 bez jezičnog modela (non-LM) i wav2vec2 u kombinaciji s kenLM jezičnim modelom (LM). Obje transkripcije uspoređene su s originalnom transkripcijom iz korpusa, te su prebrojane sve pogrešno transkribirane pojavnice, odnosno riječi i znakovi. Nakon analize 1.206 snimki, LM model pokazao se boljim u transkripciji govora od non-LM modela (Tablica 2., vidi Sliku 7. za WER i Sliku 8. za CER). Autori temeljnog istraživanja (Ljubešić i sur. 2022:115) navode vlastite WER i CER vrijednosti (Tablica 2.) za non-LM i LM modele, a te vrijednosti se razlikuju od dobivenih rezultata u ovoj analizi. S obzirom na to da su WER i CER relativne mjere, može se zaključiti da su rezultati usporedivi: omjer WER-a između non-LM i LM modela u ovoj analizi iznosi 1,20, dok je u originalnom istraživanju 1,58. Slično tome, omjer CER-a za non-LM i LM modele iznosi 1,19 u ovoj analizi, dok je u originalnom radu 1,18.

	Ljubešić i sur. (2022)		Dobiveni rezultati	
	non-LM	LM	non-LM	LM
WER	6,79 %	4,30 %	9,5 %	7,9 %
CER	2,22 %	1,88 %	5,6 %	4,7 %

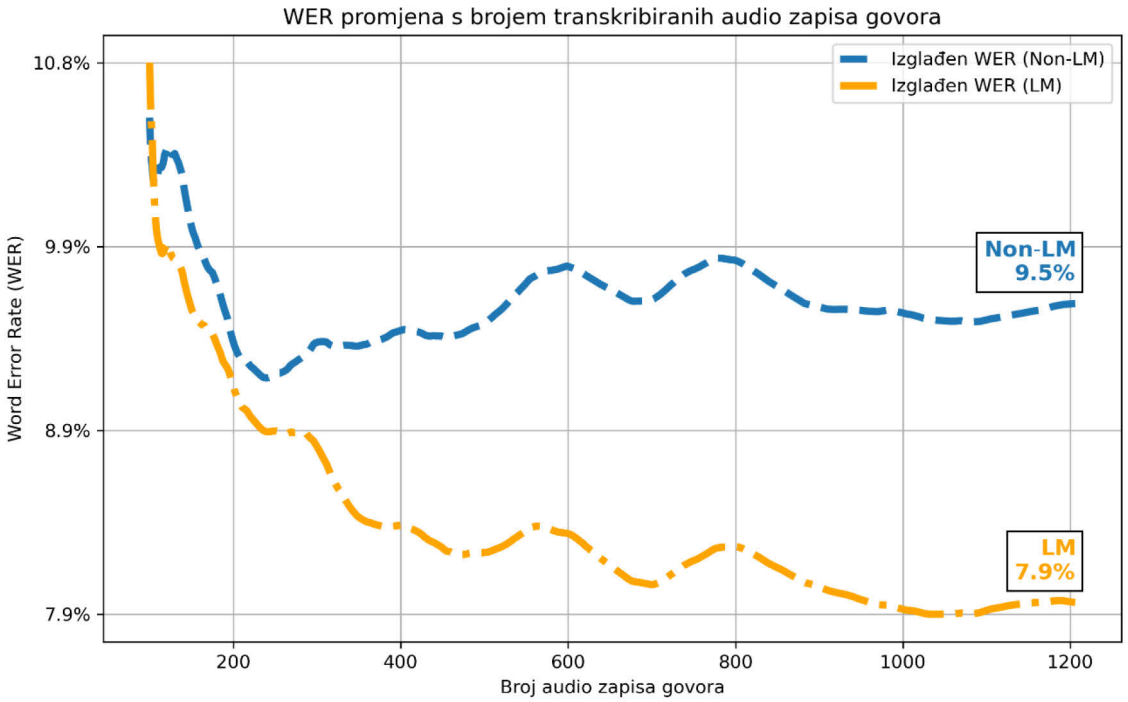
Tablica 2. Usporedba izvornih WER i CER rezultata s dobivenim, za akustični (non-LM) i akustični + jezični (LM) model.

3.2. Mjera F1 za oba modela

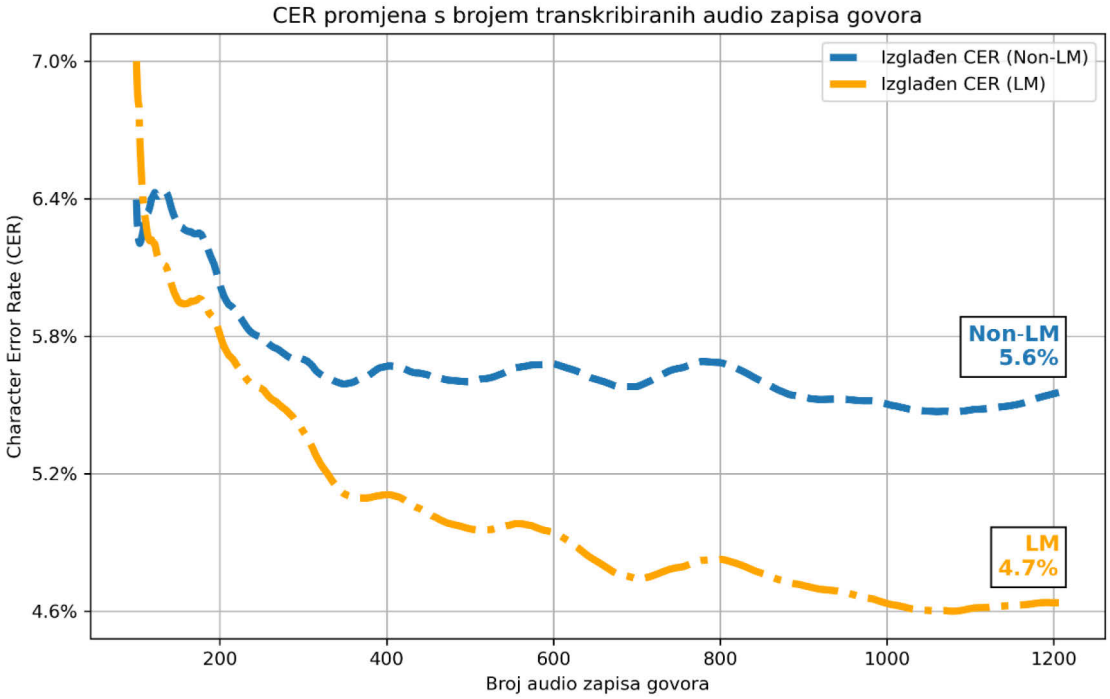
Izračun mjere F1 radio se koristeći formule spomenute u poglavlju 1.7. Analizom preciznosti i odziva oba modela, dobile su se mjere F1 za WER i CER pojavnice (Tablica 3.).

	Non-LM	LM	Razlika
prosjek F1 WER	0,9523	0,9606	+ 0,87 %
prosjek F1 CER	0,9700	0,9751	+ 0,53 %

Tablica 3. Rezultati mjere F1 vrednovanja za WER i CER te razlika u performansi LM modela nad non-LM.



Slika 7. WER rezultati poredbe dva modela na uzorku korpusa. Krivulja je izglađena (eng. *smoothed*) pomoću pokretnog prosjeka (eng. *moving average*) s prozorom veličine N = 100 uzorka.



Slika 8. CER rezultati poredbe dva modela na uzorku korpusa. Krivulja je izglađena pomoću pokretnog prosjeka s prozorom veličine N = 100 uzorka.

4. Zaključak

Rezultati ovoga rada podupiru zaključak Ljubešića i sur. (2022) da wav2vec2 ASR model s jezičnim modelom, poput kenLM, transkribira govorni zvuk značajno bolje nego model bez jezičnog modela. Akustički model u wav2vec2 koristi samonadzirano učenje, što mu omogućava da nauči reprezentacije govora iz velike količine neoznačenih podataka. Ova tehnika je korisna jer omogućava modelu da prepozna obrasce i strukture u govoru bez potrebe za ručnim označavanjem podataka, dok se fino podešavanje obavlja na manjoj količini označenih podataka koristeći funkciju gubitka poput *Connectionist Temporal Classification* (CTC). CTC omogućava modelima dubokog učenja, kao što je wav2vec2, točno označavaje sekvence govora, čak i kada ulazni podaci nisu segmentirani.

Jezični model kenLM, koji koristi Kneser-Ney zaglađivanje za bolju preciznost i skalabilnost, doprinosi ovom poboljšanju jer može efikasno predviđati najvjerojatniji niz riječi na temelju statističke distribucije jezičnih podataka. Ovo je ključno za točnije transkribiranje govora, jer omogućava modelu predviđanje riječi na temelju konteksta i strukture jezika. Analiza pokazuje da je kenLM model značajno smanjio *Word Error Rate* (WER) i *Character Error Rate* (CER) u usporedbi s modelom bez jezičnog modela, s poboljšanjem od 2,4 % za WER i 0,9 % za CER. Također, prosječne F1 mjere za WER i CER bile su veće za modele s jezičnim modelom (F1 WER: +0,87 %, F1 CER: +0,53 %).

Osim što je ParlaSpeechHR prvi javno dostupni model za transkripciju hrvatskog jezika, također je vrlo točan, s točnošću između 90 i 95 %, čineći ga izuzetno korisnim jezičnim alatom za hrvatski jezik. Kombinacija akustičkog modela wav2vec2 i kenLM jezičnog modela pokazuje da je moguće postići višu točnost u automatskoj transkripciji govora, što otvara nove mogućnosti za primjene u raznim domenama koje zahtijevaju obradu govornog jezika na hrvatskom jeziku.

5. Literatura

1) Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Preuzeto s https://www.tensorflow.org/

2) Amodei, D., Ananthanarayanan, S., Anubhai, R., ... & McGrew, B. (2016). Deep Speech 2: End-to-End Speech Recognition in English and Mandarin. Proceedings of the International Conference on Machine Learning (ICML), 173-182.

3) Bai, J., Lu, F., Zhang, K., Wang, L., Chen, G., Yang, T., ... & Zhou, M. (2019). ONNX: Open Neural Network Exchange, GitHub (online), https://github.com/onnx/onnx (Pristupljeno 20. srpnja 2024.).

4) Bakran, J. (1996). Zvučna slika hrvatskoga govora. Ibis grafika.

5) Basharin, G. P., Langville, A. N., & Naumov, V. A. (2004). The life and work of A.A. Markov. Linear Algebra and Its Applications, 386, 3–26. doi:10.1016/j.laa.2003.12.041

6) Boersma, P., & Weenink, D. (2024). Praat: Doing phonetics by computer (Version 6.4.12) [Computer software]. http://www.praat.org/ (Pristupljeno: 2. svibnja 2024.).

7) Chen, S. F., & Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language, 13*(4), 359-394. https://doi.org/10.1006/csla.1999.0128

8) Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In International conference on machine learning (pp. 1597-1607). PMLR.

9) Damjanović, S. (2003). Staroslavenski jezik. Hrvatska sveučilišna naklada.

10) Derczynski, L. (2016). Complementarity, F-score, and NLP evaluation. In Proceedings of the Tenth International Conference on Language Resources and Evaluation

(LREC'16) (pp. 261–266). European Language Resources Association (ELRA). https://aclanthology.org/L16-1040 (Pristupljeno 28. srpnja 2024.).

11) Domingos, P. (2015). The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books.

12) Graves, A., Fernández, S., Gómez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. Proceedings of the 23rd International Conference on Machine Learning (ICML 2006), 369-376.

13) Goyal, P., Pandey, S., & Jain, K. (2018). Deep learning for natural language processing. Apress.

14) Heafield, K. (2011, July 30-31). KenLM: Faster and smaller language model queries. In Proceedings of the Sixth Workshop on Statistical Machine Translation (pp. 187-197). Edinburgh, Scotland, United Kingdom.

15) Heafield, K., Pouzyrevsky, I., Clark, J. H., & Koehn, P. (2013, August 4-9). Scalable modified Kneser-Ney language model estimation. In Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) (pp. 690-696). Sofia, Bulgaria.

16) Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. r., Jaitly, N., ... & Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. IEEE Signal Processing Magazine, 29(6), 82-97.

17) Jurafsky, D., & Martin, J. H. (2009). Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition (2nd ed.). Pearson.

18) Jelinek, F. (1976). Continuous Speech Recognition by Statistical Methods. Proceedings of the IEEE, 64(4), 532-556.

19) Ljubešić, N., Koržinek, D., Rupnik, P., & Jazbec, I. (2022). ParlaSpeech-HR - a Freely Available ASR Dataset for Croatian Bootstrapped from the ParlaMint Corpus. PARLACLARIN. http://hdl.handle.net/11356/1494

20) Ljubešić, N.; Koržinek, D. & Rupnik, P. (2024). Parliamentary spoken corpus of Croatian ParlaSpeech-HR 2.0, Slovenian language resource repository CLARIN.SI, ISSN 2820-4042, http://hdl.handle.net/11356/1914

21) Magalhães, R. P., Vasconcelos, D. J. R., Fernandes, G. S., Cruz, L. A., Sampaio, M. X., Macêdo, J. A. F., & da Silva, T. L. C. (2022). Evaluation of Automatic Speech Recognition Approaches. Journal of Information and Data Management, 13(3), 366–377.

22) Maretić, T. (Prevodilac i komentator). (2011). Metamorfoze (Ovidije). Bulaja. https://www.hrlektire.com/wp-content/uploads/2018/09/ovidije_metamorfoze.pdf (Pristupljeno: 28.7.2024.).

23) McCowan, I., Moore, D., Dines, J., Gática-Pérez, D., Flynn, M., Wellner, P.D., & Bourlard, H. (2004). On the Use of Information Retrieval Measures for Speech Recognition Evaluation.

24) Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. U 33. konferenciji za neuroničko informacijsko procesiranje (NeurIPS 2019). Preuzeto s https://pytorch.org/

25) Rabiner, L. R., & Juang, B. H. (1986). An Introduction to Hidden Markov Models. IEEE ASSP Magazine, 3(1), 4-16.

26) Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

27) Schneider, S., Baevski, A., Collobert, R., & Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. arXiv preprint arXiv:1904.05862.

28) Škarić, I. (2007). "Fonetika hrvatskoga književnoga jezika." U Glasovi i oblici hrvatskoga književnoga jezika, str. 17. Zagreb: Nakladni zavod Globus.

29) Šnajder, J. (2021). Strojno učenje, 02 Osnovni koncepti. TakeLab. https://strojnoucenje.takelab.fer.hr/ (Pristupljeno 10.7.2024.).

30) Tambovtsev, Y., & Martindale, C. (2007). Phoneme frequencies follow a Yule distribution. SKASE Journal of Theoretical Linguistics, 4(2), 1-11. http://www.skase.sk/Volumes/JTL09/ (Pristupljeno 19.12.2023.).

31) Ulasik, M. A., Hürlimann, M., Germann, F., Gedik, E., Benites, F., & Cieliebak, M. (2020). CEASR: A corpus for evaluating automatic speech recognition. In Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020) (pp. 6477–6485). European Language Resources Association (ELRA).

32) von Platen, P. (2022). Wav2Vec2 with n-gram models. Hugging Face. https://huggingface.co/blog/wav2vec2-with-ngram (Pristupljeno 28. srpnja 2024.).

33) Wang, Y.-Y., Acero, A., & Chelba, C. (2003). Is word error rate a good indicator for spoken language understanding accuracy? Speech Technology Group, Microsoft Research.

Vrednovanje jezičnog alata za transkripciju govora: ParlaSpeech-HR ASR model

Sažetak

Ovaj rad fokusira se na vrednovanje jezičnog alata za transkripciju govora koristeći ParlaSpeech-HR ASR model i korpus. ParlaSpeech-HR korpus sadrži 404 tisuće snimki govora iz hrvatskog parlamenta, zajedno s njihovim transkriptima. U radu je korišten *wav2vec2* akustični model, dok je kenLM (5-gramski) jezični model izrađen na temelju tekstualnog dijela korpusa. Spojeni su akustični i jezični model kako bi se stvorilo cjelovito ASR (Automatic Speech Recognition) rješenje. Vrednovanje je provedeno na uzorku od 1206 snimki iz govornog korpusa, koristeći metrike WER (Word Error Rate), CER (Character Error Rate) i F1 mjeru. Rezultati su pokazali da kombinirani akustični i jezični model postiže bolje performanse u usporedbi s modelom koji koristi samo akustičke podatke.

Ključne riječi: Vrednovanje ASR modela, ParlaSpeech-HR, jezični alat, hrvatski jezik

Evaluation of a Language Tool for Speech Transcription: ParlaSpeech-HR ASR Model

Abstract

This paper focuses on the evaluation of a language tool for speech transcription using the ParlaSpeech-HR ASR model and corpus. The ParlaSpeech-HR corpus contains 404 thousand speech recordings from the Croatian parliament, along with their transcripts. The study utilizes the *wav2vec2* acoustic model, while a kenLM (5-gram) language model was developed based on the textual part of the corpus. The acoustic and language models were combined to create a comprehensive ASR (Automatic Speech Recognition) solution. Evaluation was performed on a sample of 1206 recordings from the speech corpus, using WER (Word Error Rate), CER (Character Error Rate), and F1 metrics. The results indicate that the combined acoustic and language model achieves better performance compared to the model that uses only acoustic data.

Keywords: ASR model evaluation, ParlaSpeech-HR, language tool, Croatian language

Priloženo

Priloženo A – kôd za korištenje ParlaSpeechHR modela

```
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
import soundfile as sf
import torch
import os

# odabir uređaja (CPU ili GPU)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Stvaranje procesora i modela
processor = Wav2Vec2Processor.from_pretrained("classla/wav2vec2-large-slavic-parlaspeech-hr")
model = Wav2Vec2ForCTC.from_pretrained("classla/wav2vec2-large-slavic-parlaspeech-hr")

# Otvaranje audio datoteke
speech, sample_rate = sf.read("audio_snimka.wav")

# Predprocesiranje snimke i priprema za model
input_values = processor(speech, sampling_rate=sample_rate, return_tensors="pt").input_values.to(device)

# Uklanjanje audio datoteke
os.system("rm audio_snimka.wav ")

# Vraćanje logita
logits = model.to(device)(input_values).logits

# Predviđanje
predicted_ids = torch.argmax(logits, dim=-1)
transcription = processor.decode(predicted_ids[0]).lower()

# transkripcija govora je ispisana
```

Priloženo B – kôd za izradu kenLM .arpa i .bin modela, te wav2vec2WithLM.

Napomena: potrebno je imati *kenlm_lmplz_to_arpa.txt* datoteku u kojoj nema simbola izvan *wav2vec2*-ovog vokabulara (vocab) i u svakom redu je po jedna rečenice. Lmplz automatski postavlja oznake za početak i kraj rečenica (<s>, </s>). Imajući na umu kako je kenLM model enkodiran UTF-8 standardom, a Windows operacijski sustav automatski postavlja UTF-16, potrebno je ili koristit Linux operacijski sustav ili napraviti *wrapper* funkcije (sufiks _UTF8). Alternativno, standard UTF-8 može se na silu namjestit promjenom kôda u samoj pyctcdecode skripti.

```
from transformers import AutoProcessor

load_lm_proc = r"D:\ASR\ParlaspeechHR\wav2vec2-large-slavic-hr"

processor1 = AutoProcessor.from_pretrained(load_lm_proc)
print("loaded proc")

vocab_dict = processor1.tokenizer.get_vocab()
sorted_vocab_dict = {k.lower(): v for k, v in sorted(vocab_dict.items(), key=lambda item: item[1])}
print("got vocab")

print(f"\nvocab: \n{sorted_vocab_dict.keys()}")
```

 # Izrada ARPA modela iz .txt

```
D:/kenlm/bin/Debug/lmplz -o 5 -S 80% --skip_symbols < D:\ParlaSpeech-
HR.v1.0\kenlm_lmplz_to_arpa.txt > D:\ParlaSpeech-HR.v1.0\5gram.arpa
```

 # Izrada ARPA dekodera

```
import codecs
from pyctcdecode import build_ctcdecoder
from pyctcdecode.language_model import load_unigram_set_from_arpa as
original_load_unigram_set_from_arpa

# Force utf-8 encoding with this wrapper:
def load_unigram_set_from_arpa_utf8(arpa_path):
```

```
    with codecs.open(arpa_path, 'r', encoding='utf-8') as f:
        start_1_gram = False
        unigrams = set()
        for line in f:
            line = line.strip()
            if line == "\\1-grams:":
                start_1_gram = True
            elif line == "":
                start_1_gram = False
            elif start_1_gram:
                parts = line.split()
                if len(parts) >= 2:
                    unigrams.add(parts[1])
        return unigrams

# Use the modified function for loading unigrams
def build_ctcdecoder_utf8(labels, kenlm_model_path, unigrams=None, alpha=0.5, beta=1.0,
unk_score_offset=-10.0, lm_score_boundary=True):
    if unigrams is None and kenlm_model_path is not None:
        if kenlm_model_path.endswith(".arpa"):
            unigrams = load_unigram_set_from_arpa_utf8(kenlm_model_path)
        else:
            logger.warning(
                "Unigrams not provided and cannot be automatically determined from LM file (only "
                "arpa format). Decoding accuracy might be reduced."
            )
    return build_ctcdecoder(labels, kenlm_model_path, unigrams, alpha, beta, unk_score_offset,
lm_score_boundary)

# Define paths
darpa = r"D:\ParlaSpeech-HR.v1.0\5gram.arpa"

DEFAULT_ALPHA = 0.5
DEFAULT_BETA = 0.5
DEFAULT_UNK_LOGP_OFFSET = -1.0
DEFAULT_SCORE_LM_BOUNDARY = True

# Build the decoder with labels, kenlm_model_path, and unigrams ## SUPER IMPORTANT TO USE
WRAPPER FUNCTION ON WINDOWS
decoder_arpa = build_ctcdecoder_utf8(
```

```
                labels=list(sorted_vocab_dict.keys()),
                kenlm_model_path=darpa,
                alpha=DEFAULT_ALPHA,
                beta=DEFAULT_BETA,
                unk_score_offset=DEFAULT_UNK_LOGP_OFFSET,
                lm_score_boundary=DEFAULT_SCORE_LM_BOUNDARY,
    )
```

Izrada BIN modela iz ARPA:

D:\ASR\kenlm\bin\Debug\build_binary D:\ParlaSpeech-HR.v1.0\5gram.arpa D:\ParlaSpeech-HR.v1.0\5gram.bin

Izrada BIN dekodera

```
import codecs
from pyctcdecode import build_ctcdecoder
from pyctcdecode.language_model import load_unigram_set_from_arpa as original_load_unigram_set_from_arpa
from typing import List, Optional, Collection
import logging

# Configure logging (set level and handlers as needed)
logging.basicConfig(level=logging.INFO)  # Adjust level as needed
logger = logging.getLogger(__name__)  # Create logger instance for your module

# Function to load unigrams from a text file
def load_unigrams_from_txt(txt_file_path: str) -> List[str]:
    with codecs.open(txt_file_path, 'r', encoding='utf-8') as f:
        unigrams = [line.strip() for line in f if line.strip()]
    return unigrams

# Modified build_ctcdecoder_utf8 function
def build_ctcdecoder_utf8(labels: List[str], kenlm_model_path: Optional[str] = None,
                          unigrams: Optional[Collection[str]] = None,
                          unigrams_txt_file: Optional[str] = None,
                          alpha: float = 0.5, beta: float = 1.0,
                          unk_score_offset: float = -10.0,
                          lm_score_boundary: bool = True):
```

```
    # Load unigrams from file if not provided
    if unigrams is None and unigrams_txt_file is not None:
        unigrams = load_unigrams_from_txt(unigrams_txt_file)

    # Load unigrams from ARPA if kenlm_model_path is provided and unigrams are still None
    if unigrams is None and kenlm_model_path is not None:
        if kenlm_model_path.endswith(".arpa"):
            unigrams = load_unigram_set_from_arpa_utf8(kenlm_model_path)
        else:
            logger.warning(
                "Unigrams not provided and cannot be automatically determined from LM file (only "
                "arpa format). Decoding accuracy might be reduced."
            )

    return build_ctcdecoder(labels, kenlm_model_path, unigrams, alpha, beta, unk_score_offset,
lm_score_boundary)


# Define paths
dbin = r"D:\ParlaSpeech-HR.v1.0\5gram.bin"

DEFAULT_ALPHA = 0.5
DEFAULT_BETA = 1.0
DEFAULT_UNK_LOGP_OFFSET = -10.0
DEFAULT_SCORE_LM_BOUNDARY = True

unigrams_txt_file = r"D:\ParlaSpeech-HR.v1.0\unigrams.txt"  # Specify the path to save unigrams
unigrams = load_unigrams_from_txt(unigrams_txt_file)

# Build the decoder with labels, kenlm_model_path, and unigrams
decoder_bin = build_ctcdecoder_utf8(
                        labels=list(sorted_vocab_dict.keys()),
                        kenlm_model_path=dbin,
                        unigrams=unigrams,
                        alpha=DEFAULT_ALPHA,
```

```
                    beta=DEFAULT_BETA,
                    unk_score_offset=DEFAULT_UNK_LOGP_OFFSET,
                    lm_score_boundary=DEFAULT_SCORE_LM_BOUNDARY)

    print(f"\ndecoder_bin of the lm type is here\n")
```

Definiranje Wav2Vec2ProcessorWithLM_UTF8

```
    from transformers import PreTrainedTokenizer, SequenceFeatureExtractor, ProcessorMixin,
    requires_backends, AutoFeatureExtractor, Wav2Vec2CTCTokenizer, Wav2Vec2ProcessorWithLM
    from torch import nn
    import os

    class Wav2Vec2ProcessorWithLM_UTF8(Wav2Vec2ProcessorWithLM, nn.Module):

      feature_extractor_class = "AutoFeatureExtractor"
      tokenizer_class = "Wav2Vec2CTCTokenizer"

      @classmethod
      def from_pretrained_utf8(cls, pretrained_model_name_or_path, **kwargs):
        requires_backends(cls, "pyctcdecode")
        from pyctcdecode import BeamSearchDecoderCTC

        feature_extractor, tokenizer, decoder = super().from_pretrained(pretrained_model_name_or_path,
    **kwargs)

        if os.path.isdir(pretrained_model_name_or_path) or
    os.path.isfile(pretrained_model_name_or_path):
          unigram_encoding = kwargs.get("unigram_encoding", "utf-8")
          decoder = BeamSearchDecoderCTC.load_from_dir(pretrained_model_name_or_path,
    unigram_encoding)
        else:
          kwargs.pop("_from_auto", None)
          kwargs.pop("trust_remote_code", None)

          language_model_filenames =
    os.path.join(BeamSearchDecoderCTC._LANGUAGE_MODEL_SERIALIZED_DIRECTORY, "*")
```

```
            alphabet_filename = BeamSearchDecoderCTC._ALPHABET_SERIALIZED_FILENAME
            allow_patterns = [language_model_filenames, alphabet_filename]

            decoder = BeamSearchDecoderCTC.load_from_hf_hub(
                pretrained_model_name_or_path, allow_patterns=allow_patterns, **kwargs
            )

        # set language model attributes
        for attribute in ["alpha", "beta", "unk_score_offset", "score_boundary"]:
            value = kwargs.pop(attribute, None)
            if value is not None:
                cls._set_language_model_attribute(decoder, attribute, value)

        missing_decoder_tokens = cls.get_missing_alphabet_tokens(decoder, tokenizer)
        if len(missing_decoder_tokens) > 0:
            raise ValueError(
                f"The tokens {missing_decoder_tokens} are defined in the tokenizer's "
                "vocabulary, but not in the decoder's alphabet. "
                f"Make sure to include {missing_decoder_tokens} in the decoder's alphabet."
            )

        return cls(feature_extractor=feature_extractor, tokenizer=tokenizer, decoder=decoder)

    @staticmethod
    def _get_arguments_from_pretrained(pretrained_model_name_or_path, **kwargs):
        feature_extractor = SequenceFeatureExtractor.from_pretrained(pretrained_model_name_or_path,
**kwargs)
        tokenizer = PreTrainedTokenizer.from_pretrained(pretrained_model_name_or_path, **kwargs)
        return feature_extractor, tokenizer
```

Izrada kenLM modela (iz ARPA ili BIN)

```
import os
import shutil
from transformers import Wav2Vec2ProcessorWithLM
```

```
class BeamSearchDecoderCTC_UTF8:
    def __init__(self, original_decoder):
        self._decoder = original_decoder

    def save_to_dir_utf8(self, filepath):
        """Save the decoder to a directory with UTF-8 encoding."""
        alphabet_path = os.path.join(filepath, self._decoder._ALPHABET_SERIALIZED_FILENAME)
        with open(alphabet_path, "w", encoding="utf-8") as fi:
            fi.write(self._decoder._alphabet.dumps())

        lm = self._decoder._language_model
        if lm is None:
            print("Decoder has no language model.")
        else:
            lm_wrapper = LanguageModel_UTF8(lm)
            lm_wrapper.save_to_dir_utf8(filepath)

# Create a UTF-8 wrapper for saving language model files
class LanguageModel_UTF8:
    def __init__(self, original_lm):
        self._lm = original_lm
        self._unigram_set = self._lm._unigram_set
        self._attrs = self._lm.serializable_attrs

    def save_to_dir_utf8(self, filepath):
        """Save the language model to a directory with UTF-8 encoding."""
        lm_dir = os.path.join(filepath, "language_model")
        os.makedirs(lm_dir, exist_ok=True)

        # Save unigrams
        unigrams_path = os.path.join(lm_dir, self._lm._UNIGRAMS_SERIALIZED_FILENAME)
        with open(unigrams_path, "w", encoding="utf-8") as fi:
            for unigram in sorted(self._unigram_set):
```

```
            fi.write(unigram + "\n")


        # Save attributes
        attrs_path = os.path.join(lm_dir, self._lm._ATTRS_SERIALIZED_FILENAME)
        with open(attrs_path, "w", encoding="utf-8") as fi:
            for attr_name, attr_value in self._attrs.items():
                fi.write(f"{attr_name}: {attr_value}\n")


        ###### CHANGE EXTENTION HERE
        # Save kenlm model
        kenlm_extentions = ".bin"
        kenlm_name = f"5gram{kenlm_extentions}"


        kenlm_path = os.path.join(lm_dir, kenlm_name)
        print(f"Copying kenlm model from {self._lm._kenlm_model.path} to {kenlm_path}. This may take some time.")
        shutil.copy2(self._lm._kenlm_model.path, kenlm_path)


# Create a subclass of Wav2Vec2ProcessorWithLM with UTF-8 encoding
class Wav2Vec2ProcessorWithLM_UTF8(Wav2Vec2ProcessorWithLM):
    def save_pretrained_utf8(self, save_directory):
        """
        Save a model and its configuration file to a directory, with UTF-8 encoding for language model files.


        Args:
            save_directory (str): Directory to save the processor.
        """
        if os.path.isfile(save_directory):
            logger.error(f"Provided path ({save_directory}) should be a directory, not a file")
            return
        os.makedirs(save_directory, exist_ok=True)


        # Save feature extractor and tokenizer
        self.feature_extractor.save_pretrained(save_directory)
```

```
        self.tokenizer.save_pretrained(save_directory)

        # Save the decoder using the UTF-8 wrapper method
        utf8_decoder = BeamSearchDecoderCTC_UTF8(self.decoder)
        utf8_decoder.save_to_dir_utf8(save_directory)

        print(f"\nProcessor with LM saved successfully to:\n    {save_directory}")

# Assuming decoder_arpa and processor1 are already built
# Create the processor with the existing decoder using our UTF-8 wrapper
processor_with_lm = Wav2Vec2ProcessorWithLM_UTF8(
                                feature_extractor=processor1.feature_extractor,
                                tokenizer=processor1.tokenizer,
                                decoder=decoder_bin #CHOOSE HERE decoder_arpa or decoder_bin
)

save_new_lm_path = r"D:\ASR\ParlaspeechHR\wav2vec2-large-slavic-hr-lm-custom"
processor_with_lm.save_pretrained_utf8(save_new_lm_path)
```

KenLM model je uspješno napravljen i prospremljen te je spreman za korištenje. Za više detalja: https://github.com/porupski/diplomski/ Jupyter Lab Notebook *LING-dipl_ParlaRUN_v5.2.ipynb*

Priloženo C – Arhitektura akustičnog (wav2vec 2.0) modela

```
Wav2Vec2ForCTC(
  (wav2vec2): Wav2Vec2Model(
    (feature_extractor): Wav2Vec2FeatureEncoder(
      (conv_layers): ModuleList(
        (0): Wav2Vec2LayerNormConvLayer(
          (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,))
          (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (activation): GELUActivation()
        )
        (1-4): 4 x Wav2Vec2LayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,))
          (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (activation): GELUActivation()
        )
        (5-6): 2 x Wav2Vec2LayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,))
          (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
          (activation): GELUActivation()
        )
      )
    )
    (feature_projection): Wav2Vec2FeatureProjection(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (projection): Linear(in_features=512, out_features=1024, bias=True)
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): Wav2Vec2EncoderStableLayerNorm(
      (pos_conv_embed): Wav2Vec2PositionalConvEmbedding(
        (conv): ParametrizedConv1d(
          1024, 1024, kernel_size=(128,), stride=(1,), padding=(64,), groups=16
          (parametrizations): ModuleDict(
            (weight): ParametrizationList(
```

```
            (0): _WeightNorm()
          )
        )
      )
      (padding): Wav2Vec2SamePadLayer()
      (activation): GELUActivation()
    )
    (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.0, inplace=False)
    (layers): ModuleList(
      (0-23): 24 x Wav2Vec2EncoderLayerStableLayerNorm(
        (attention): Wav2Vec2Attention(
          (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (out_proj): Linear(in_features=1024, out_features=1024, bias=True)
        )
        (dropout): Dropout(p=0.0, inplace=False)
        (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
        (feed_forward): Wav2Vec2FeedForward(
          (intermediate_dropout): Dropout(p=0.0, inplace=False)
          (intermediate_dense): Linear(in_features=1024, out_features=4096, bias=True)
          (intermediate_act_fn): GELUActivation()
          (output_dense): Linear(in_features=4096, out_features=1024, bias=True)
          (output_dropout): Dropout(p=0.0, inplace=False)
        )
        (final_layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
)
(dropout): Dropout(p=0.0, inplace=False)
(lm_head): Linear(in_features=1024, out_features=50, bias=True)
)
```