

Govorna interakcija u stvarnom vremenu s robotom TIOSS koristeći umjetnu inteligenciju

Iličić, Klara

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: <https://urn.nsk.hr/urn:nbn:hr:168:289046>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-04-01**



Repository / Repozitorij:

[FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repository](#)



SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 510

**GOVORNA INTERAKCIJA U STVARNOM VREMENU S
ROBOTOM TIOSS KORISTEĆI UMJETNU INTELIGENCIJU**

Klara Ilić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 510

**GOVORNA INTERAKCIJA U STVARNOM VREMENU S
ROBOTOM TIOSS KORISTEĆI UMJETNU INTELIGENCIJU**

Klara Iličić

Zagreb, lipanj 2024.

DIPLOMSKI ZADATAK br. 510

Pristupnica: **Klara Ilić (0036523700)**

Studij: Računarstvo

Profil: Znanost o podacima

Mentorica: izv. prof. dr. sc. Ana Sović Kržić

Zadatak: **Govorna interakcija u stvarnom vremenu s robotom TIOSS koristeći umjetnu inteligenciju**

Opis zadatka:

Robot TIOSS je prvi hrvatski humanoidni robot, težak preko 150 kg i visok preko dva metra. U svojem originalnom obliku ovaj robot je putem radio veze mogao komunicirati s ljudima. Cilj diplomskog rada je omogućiti komunikaciju ljudi s robotom putem umjetne inteligencije. Istražiti i implementirati cijeli proces koji uključuje prepoznavanje govora, pretvaranje govora u tekst, korištenje modela umjetne inteligencije za dobivanje odgovora na postavljene pisane upite te sinteza govora na temelju dobivenog odgovora. Prilagoditi izabrane modele umjetne inteligencije kako bi se mogli koristiti u stvarnom vremenu na Raspberry Pi mikroračunalu.

Rok za predaju rada: 28. lipnja 2024.

Sadržaj

Uvod	1
1. Robot TIOSS	3
1.1. Opis robota TIOSS	3
1.2. Trenutno stanje	5
2. Opis projektnog zadatka	7
2.1. Postojeća rješenja i primjeri	7
3. Pregled teoretske pozadine i tehnologija	9
3.1. Detekcija glasovne aktivnosti	9
3.1.1. Govornik u tihom okruženju	10
3.1.2. Govornih u bučnom okruženju	11
3.1.3. Klasifikatori	12
3.2. Pretvaranje govora u tekst	13
3.2.1. Dijelovi automatskog prepoznavanja govora	13
3.2.2. Tipovi automatskog prepoznavanja govora	14
3.2.3. Modeli za prepoznavanje govora velikog vokabulara (LVCSR)	15
3.2.4. Vrste pogrešaka i evaluacija	17
3.2.5. Google primjer	18
3.3. Veliki jezični modeli	20
3.3.1. GPT-3	22
3.4. Sinteza govora	22
3.4.1. Konkantenativna sinteza govora	23
3.4.2. Statistička parametarska sinteza govora	25
4. Dizajn sustava	28
4.1. Hardverske komponente sustava	28
4.1.1. Raspberry Pi 5	29

4.2.	Softverske komponente	30
5.	Implementacija i testiranje.....	32
5.1.	Postavljanje Raspberry Pi ja i radne okoline	32
5.2.	Prepoznavanje zvučnika i mikrofona	33
5.2.1.	ALSA.....	35
5.3.	Prepoznavanje govora.....	35
5.3.1.	<i>Speech Recognition</i> biblioteka	35
5.3.2.	ALSA konfiguracija	38
5.4.	Generiranje odgovora pomoću velikog jezičnog modela	40
5.4.1.	Isprobavanje modela.....	40
5.4.2.	GPT-3.5	43
5.4.3.	Prilagodna i učenje modela.....	44
5.4.4.	Nedostatci i prednosti uporabe GPT-3.5 modela	48
5.5.	Pretvorba teksta u zvuk.....	49
5.6.	Testiranje	50
6.	Budući rad i nadogradnje.....	52
	Zaključak	53
	Literatura	54
	Sažetak.....	58
	Summary.....	59

Uvod

Na Zavodu za elektroničke sustave i obradbu informacija Fakulteta elektrotehnike i računarstva u Zagrebu već se dvije godine radi na renovaciji robota TIOSS. TIOSS je prvi robot u Hrvatskoj, a nastao je na zavodima Fakulteta elektrotehnike i računarstva (tadašnjeg Elektrotehničkog fakulteta) šezdesetih godina 20. stoljeća.

TIOSS je bio prava senzacija. Mnogo je putovao i bio eksponat na izložbama i događajima diljem države. Zbog stalih izbivanja, nemogućnosti kvalitetnog skladištenja i nadzora, s vremenom je počeo gubiti svoje dijelove i funkcionalnosti. Na kraju je i sam nestao. Kada je pronađen u otpadu kao „staro željezo”, rodila se želja za njegovom renovacijom i vraćanjem u prvotno stanje.

Prvo je bilo potrebno istražiti originalne funkcije koje je robot imao, vidjeti koji su dijelovi još u njemu i koji rade. Prvotni je cilj bio vratiti funkciju kretanja. Prvih su godinu dana kolege bivši studenti, uz pomoć profesora i osoblja sa Zavoda, radili na restauraciji motora glave i nogu. Osim vraćanja u prvotno stanje, odlučeno je kako neće isključivo pokušavati osposobiti funkcionalnosti i dijelove originalnog robota, nego će ga modernizirati. Naime, originalni robot bio je upravljan relejima. Proučavanjem raznih opcija došlo je do ideje kako bi se upravljačke opcije mogle modernizirati. Kako bi se omogućila modernizacija i kasnije dodavanje funkcionalnosti „modernijih” robota, ugrađen je Raspberry Pi 4 Model B [1].

Ugradnja Raspberry Pija pruža razne mogućnosti za nadogradnju. Jedna od njih je i česta funkcionalnost humanoidnih robota, mogućnost razgovora. Virtualni asistenti i *chatbotovi* postali su neizostavan dio pametnih telefona i korisničkih aplikacija. Danas korisnici pomoću virtualnih asistenata mogu upravljati kućanskim uređajima, kalendarima, elektroničkom poštom i raznim drugim alatima. Nedavna je pojava velikih jezičnih modela (LLM), koji imaju iznimne sposobnosti sinteze odgovora na zadane upite, donijela još veći interes za područje virtualnih asistenata.

S obzirom da je TIOSS aluminijski dvometraš čije kretanje treba strogo nadzirati iz sigurnosnih razloga, došlo se do zaključka kako mu treba dodati funkcionalnosti koje će publici biti zanimljive i dok se on ne kreće. Implementacija virtualnog asistenta, to jest, mogućnosti razgovora, pokazala se kao dobar izbor za novu funkcionalnost. S obzirom da

je originalni robot imao zvučnik radioprijemnika koji je prenosio ljudski glas, ovaj se zadatak činio, ustvari, kao modernizacija početne verzije.

Ugradnja Raspberry Pi ja i napredak u području prepoznavanja i sinteze odgovora i govora, omogućuju nam implementaciju kvalitetnog rješenja koje će nam omogućiti *real-time* razgovor. Ideja je da se model za sintezu odgovora nauči o povijesti robota, njegovim tvorcima, dijelovima i idejama koje stoje iza njegovog nastanka, ali i o ljudima koji trenutno rade na njemu, novim dijelovima i novim funkcionalnostima.

1. Robot TIOSS

Ideje stvaranja mehaničkih naprava koje bi služile za obavljanje fizičkog posla postojale su od drevnih vremena. O automatiziranim napravama u ljudskom ili životinjskom obliku može se čitati u mitovima i pričama iz staroga Egipta i antičke Grčke. Kroz stoljeća, potaknuto razvojem tehnologije i industrijskim revolucijama, zanimanje za robotima i automatizacijom raznih procesa je raslo. Osim mogućnosti ubrzanja proizvodnje, rastao je interes za proizvodnjom robota koji bi oponašali ljude. Znanstvenici i inženjeri nastojali su proizvesti robota koji bi mogao govoriti i hodati.

Do pravog zamaha u istraživanju i proizvodnji robota došlo je početkom 20. stoljeća. Na Svjetskoj izložbi u New Yorku 1939. godine, Westinghouse Electric Corporation predstavio je humanoidnog robota imena Elekto koji je mogao hodati, pričati i pušiti cigarete [2]. Petnaest godina nakon Elektra, 1954. George Devol patentirao je prvog industrijskog robota imena Unimate. Zadaća mu je bila izvoditi zadatke poput zavarivanja i rukovanja teškim materijalima, što je unaprijedilo industrijsku proizvodnju, ali i pobudilo još veći interes za istraživanjem i razvojem robotike [3].

No, nije se robotika razvijala samo u Americi. San o stvaranju autonomnog robota koji samostalno misli i funkcionira imao je 1950-ih Branimir Makanec, tada student Elektrotehničkog fakulteta. Godine 1961., pri završetku studija, zajedno s nekoliko asistenata s raznih zavoda, osnovao je sekciju *Grupa kibernetičara*. U okviru sekcije, prema nacrtu i viziji Branimira Makaneca, napravljen je TIOSS, prvi robot u Hrvatskoj [4].

1.1. Opis robota TIOSS

Teledirigirani izvršni organ samoorganizirajućeg sustava, kraće nazvan TIOSS (Slika 1.1), nastao je iz želje da se uveleća jedna proslava zagrebačkog Elektrotehničkog fakulteta na kojoj su studenti i profesori predstavljali svoje projekte. Za projekt su si dali optimističnih dva mjeseca. Na kraju je trebalo malo više od godinu dana da se TIOSS dovrši.

Dijelove za izradu robota članovi sekcije sakupljali su na vojnim smetlištima. Uglavnom se radilo o dijelovima starih američkih aviona iz Drugoga svjetskog rata. Iako je TIOSS visok oko 2 metra i ima gotovo 200 kilograma, bio je nedovoljno velik kako bi se u njega ugradilo računalo toga vremena. Zato se njime upravljalo putem prijenosnog radio-prijemnika i



Slika 1. 1 Robot TIOSS

pomoću bežičnog daljinskog upravljača. Kretanje je bilo omogućeno pomoću motora pronađenih na avionima i kotača koji su bili na dnu stopala. Mekanec ističe kako su upravo noge, u kojima je bila koncentrirana gotovo sva težina, bile najbitniji dio. U noge su ugradili 12 voltni akumulator. U lijevoj se nozi nalazio 220 voltni punjač akumulatora, a u desnoj relejski sklopovi. Robot je imao i pokretne ruke kojima je mogao hvatati i ispuštati predmete pritom znajući ima li nešto u ruci ili ne. Od opreme, TIOSS je u ustima imao zvučnik

radioprijemnika koji je prenosio ljudski glas. Robot je imao dva svjetlosna senzora koji su bili smješteni u očima. Svjetlosni su senzori bili spojeni na elektronski sklop koji je vrlo precizno određivao prima li više svjetlosti lijevi ili desni senzor. TIOSS je bio programiran tako da okreće glavu na stranu odakle mu je dolazilo više svjetlosti.

TIOSS je prvi puta javnosti predstavljen 1965. godine na Zagrebačkom velesajmu. Osim što je ljude zainteresirao svojim moćnim izgledom, pažnju je privukao i svojim, do onda, neviđenim mogućnostima. Uloga mu je bila dijeliti reklamne brošure. Kada bi netko uzeo brošuru iz njegove ruke, on bi uzeo drugu brošuru i nudio idućem posjetitelju govoreći „Izvolite” [5].

TIOSS se često selio po raznim ustanovama i izložbama. S vremenom su mu pokradeni dijelovi. Nestali su akumulatori, relejski sklopovi i motori. Nakon godina putovanja, TIOSS je nestao. Njegov autor, Branimir Makanec, nije mislio kako će se ponovno sresti. No, i to se dogodilo šezdesetak godina nakon njegove izrade. Prije nekoliko godina robot je došao u Klub mladih tehničara Dubrave kao „staro željezo”. Znajući o čemu se radi, članovi su ga pokušavali renovirati, no to im nije išlo za rukom dugo vremena jer su nedostajali glavni dijelovi [6].

Na kraju je Branimir Makanec odlučio robota darovati predsjednici Hrvatskog robotičarskog saveza, izv. prof. dr. sc. Ani Sović Kržić. TIOSS se sada nalazi u laboratoriju Zavoda za elektroničke sustave i obradbu informacija te se aktivno radi na njegovoj renovaciji.

1.2. Trenutno stanje

Kao što je ranije spomenuto, pri početku rada na ovom diplomskom radu, u TIOSS-u je već bio ugrađen Raspberry Pi 4. Preko 3.5 mm audio jacka na Raspberry Pi priključeni su zvučnici koji služe reprodukciji zvuka te se nalaze unutar tijela robota.

Što se softvera tiče, implementirana je samo produkcija zvuka. Točnije, u programskom kôdu korištena je kamera. Računalnim vidom implementirana je detekcija velike crvene površine. Kada bi crvena površina bila detektirana, pokrenula bi se funkcija za pokretanje zvučnoga zapisa. Tada bi se pomoću biblioteke *pygame* na zvučnike reproducirao unaprijed snimljeni zapis *mp3* formata. TIOSS bi tada „rekao” kako je on umjetno inteligentan [7].

S obzirom da je to bila jedina fraza koju je TIOSS govorio mjesecima, ideja ovoga rada zaista je bila učiniti ga, barem prividno, umjetno inteligentnim i omogućiti raznovrsnu glasovnu komunikaciju.

2. Opis projektnog zadatka

Kao što je ranije rečeno, ideja je bila omogućiti govornu interakciju s robotom TIOSS. Zamisao je TIOSS-a izlagati na događanjima poput Dana otvorenih vrata FER-a. U pregledu već postojećih rješenja i sličnih primjera (o kojima će više riječi biti u potpoglavlju 2.1.), viđena su samo rješenja na engleskome jeziku. Vrlo nam je bitno omogućiti razgovor na hrvatskome jeziku kako bi s robotom mogli razgovarati svi posjetitelji, bez obzira na dob i poznavanje engleskoga jezika.

Za početak, bitno je biti dobro upoznat s hardverskom komponentom koja služi kao „mozak” robota. U robotu se već nalazi Raspberry Pi 4 pa je zadatak bio usmjeren na implementiranje sustava koji će zadovoljavajuće raditi na tome jednodijelnom računalu. S obzirom kako su zvučnici već bili u robotu, jedino je još preostalo na Raspberry Pij povezati mikrofona koji će služiti kao ulazna komponenta.

Programsko rješenje mora podržati glasovni *input* koji će dolaziti s mikrofona, procesuirati ga, razumjeti što je rečeno, prebaciti ga u tekst. Nadalje, potrebno je pomoću modela strojnog ili dubokog učenja generirati smisleni odgovor na glasovni *input*. Potom je potrebno tekstualni zapis pretvoriti u zvuk koji će se reproducirati preko već postojećih zvučnika. Vrlo je važno da se cijeli proces odvija bez većih zastoja i čekanja kako bi mogao teći normalan razgovor, poput onoga s ljudima. Brzo generiranje odgovora dovest će do pozitivnog korisničkoga iskustva.

Modeli koji se koriste za prepoznavanje govora, generiranje odgovora te sintezu zvuka iznimno su zahtjevni za memoriju i procesor pa je stoga, prije početka rada na programskom rješenju, bilo važno detaljno proučiti mogućnosti Raspberry Pija, ali i razne vrste postojećih mogućih rješenja te pronaći optimalno rješenje za naš slučaj.

2.1. Postojeća rješenja i primjeri

TIOSS je nastao prije više od šezdeset godina te ga je teško uspoređivati s današnjim tehnički naprednijim rješenjima. Izgledom ga možemo usporediti s, već spomenutim, robotom Elektro. No, uzmemo li u obzir funkcionalnosti koje se trenutno razvijaju i koje se žele razviti, možemo ga usporediti s nekim već viđenim primjerima.

Krenuvši od humanoidnih robota, prvi primjeri koji se nameću, poznati su roboti Pepper i NAO. Njihove su odlike humanoidni dizajn, velika pokretljivost, mogućnost detekcije emocija kod sugovornika te, između ostalog, govor. Primjenjuju se u razne svrhe, edukacijske, asistivne, rehabilitacijske, ali i zabavne [8, 9]. Kao što želimo postići i s TIOSS-om, koriste se na raznim sajmovima i izložbama. Na takvim se događanjima poglavito ističe njihova mogućnost glasovne komunikacije na više desetaka jezika.

Umjetna inteligencija postaje iznimno popularna u muzejima i znanstvenim centrima. Jedan od primjera je i robot Kopernik. Podloga za ideju ovoga rada, upravo je robot Nikole Kopernika u Znanstvenom centru Copernicus u Varšavi. Robot Kopernika napravljen je da likom izgleda poput Nikole Kopernika kakvim je prikazan na povijesnim slikama, a s posjetiteljima komunicira pomoću modela ChatGPT3. Zahvaljujući ChatGPT-u može razgovarati o raznim temama [10]. Osim Kopernika, mnogi su primjeri uporabe robota ili *chatbotova* u muzejima diljem svijeta. U muzeju Musée du quai Branly možete sresti robota Pepper u ulozi kritičara umjetnosti. Louvre pak u ponudi ima virtualnog asistenta Leonarda. Muzeji ove tehnologije koriste kako bi povećali angažman posjetitelja. Modeli koji se koriste u pozadini iskorišteni su kako bi posjetiteljima pružili interaktivno i personalizirano iskustvo odgovarajući na njihova pitanja u *real-timeu* [11].

Kako bi išta od navedenog bilo moguće implementirati, potrebno je bilo istražiti postojeće primjere sustava za prepoznavanje i sintezu govora te njihovo povezivanje s velikim jezičnim modelom. Neki od najpoznatijih komercijalnih glasovnih asistenata su Google Assistant i Amazon Alexa koji se široko koriste u pametnim kućama i pametnim uređajima [12]. Kada su u pitanju rješenja otvorenog kôda, ističe se Mycroft. Radi na raznim uređajima, uključujući računala, pametne telefone i Raspberry Pi. omogućava korisnicima stvaranje prilagodljivih glasovnih asistenata koji mogu obavljati razne zadatke poput pružanja informacija ili upravljanja pametnim kućama [13]. Još jedan poznati primjer je i Rhasspy, servis za glasovne asistente otvorenog kôda koji može raditi izvanmrežno [14].

3. Pregled teoretske pozadine i tehnologija

Umjetna inteligencija i robotika uvijek je pobuđivala veliki interes. Proizvesti sustav koji se ponaša i misli racionalno poput ljudi, znanstvenicima je cilj već stoljećima. Važna stavka u razvitku robotike je računalni vid i kontrola kretanja pomoću umjetne inteligencije. Takvi roboti bolje razumiju okolinu u kojoj se nalaze, brže se prilagođavaju i imaju veliku dozu autonomije. No, uz računalni vid, veliku pozornost privlači i obrada prirodnog jezika u robotici.

Obrada prirodnog jezika glavna je komponenta interakcije čovjeka i robota. Postoje razna područja na kojima je korištenje obrade prirodnog jezika ključno. Izdavanje glasovnih naredbi, navigacija, dobivanje informacija... Ranijih je godina interakcija bila složena jer su ljudi morali izdavati točne upute koje su se nalazile unutar skupa mogućih uputa kako bi ih roboti razumjeli. Potom se istraživanje preusmjerilo na neuronske mreže. Neuronske su mreže omogućile složenije povezivanje prepoznatog jezika i radnji, ali zahtijevale su velike količine točno označenih podataka za treniranje. Do revolucionarnog pomaka došlo je 2017. godine kada je izašao rad osmorice Googleovih znanstvenika naziva "Attention Is All You Need" [15]. U radu je predstavljena arhitektura *transformera*. *Transformeri*, zajedno s velikim jezičnim modelima, o kojima će biti više riječi kasnije, pokazali su se korisnima za širok raspon funkcionalnosti. Jedna od njih je konverzacijska interakcija čovjek i robota [16].

U ovome je poglavlju cilj upoznati se s teorijskim osnovama cijelog procesa, od prepoznavanja glasa, pretvaranja govora u tekst, generiranja odgovora pa do sinteze govora.

3.1. Detekcija glasovne aktivnosti

Za početak, potrebno je prepoznati prisutnost ljudskog glasa. Za to nam može koristiti binarni klasifikator koji otkriva prisutnost ljudskog glasa u zvuku dobivenom s mikrofona. Djelomično preklapajući zadatak je i procjena vjerojatnosti prisutnosti govora koja umjesto binarne odluke daje razinu vjerojatnosti da signal sadrži govor. Detekcija glasovne aktivnosti koristi se kao algoritam za pretprocesiranje za gotovo sve druge metode obrade govora. Spomenuti se algoritam pri prepoznavanju govora koristi da bi se znalo koji dijelovi signala trebaju biti dostavljeni mehanizmu za prepoznavanje. Prepoznavanje govora računalno je složena operacija, ignoriranjem negovornih dijelova štedi se procesorska snaga. Osim za

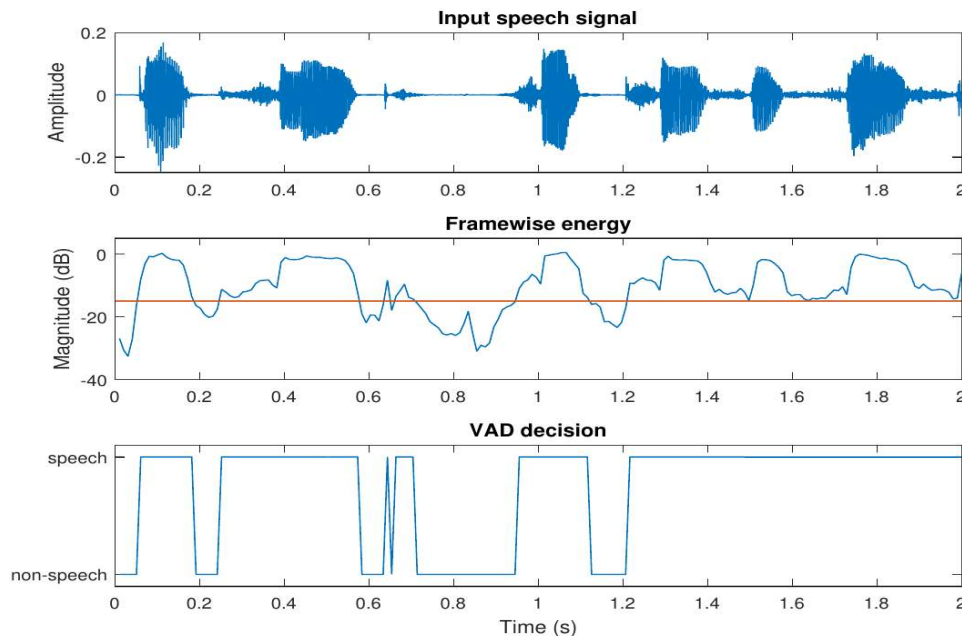
uštedu resursa, ovi se postupci koriste i za poboljšanje govora kada želimo smanjiti ili ukloniti šum u govornom signalu procjenom karakteristike šuma iz negovornih dijelova [17].

3.1.1. Govornik u tihom okruženju

Za početak, predstavimo osnovne pojmove:

- tišina - kad nema govora
- glasovna aktivnost – bilo koja aktivnost signala

Aktivnost signala može se mjeriti algoritmom za određivanje energetske praga koji radi principom procjene energije signala na okviru. Taj će algoritam dobro raditi na u tihoj okolini gdje će niskoenergetski okviri biti ispravno označeni kao tišina, a glasovna će aktivnost biti označena višom energijom. Odrediti odgovarajuću razinu govornog praga nije trivijalno. Ako za prag stavimo nisku energetske razinu, nailazimo na opasnost da i okvire s pozadinskim zvukovima ili zvukom disanja označimo kao govorne. Visoki energetske prag osigurao bi da se to ne dogodi, ali bi doveo do mogućnosti propuštanja označavanja tihog glasa kao govora. Možemo vidjeti (Slika 3.1) kako prema promjeni energije određujemo je li nešto govor ili tišina.



Slika 3. 1 detekcija glasa prema energiji [17]

Cilj same implementacije ovisi o primjeni, no potreban nam je skup mjera uspješnosti koje odražavaju te ciljeve. Ako, recimo, koristimo detekciju glasovne aktivnosti kod prepoznavanja ključnih riječi (poput *Siri* ili *Ok, Google*), onda želimo zadržati nizak broj

lažno pozitivnih detekcija, a lažno negativne su sekundarne važnosti jer korisnik može ponoviti rečeno.

Problem identificiranja lažno pozitivnih i lažno negativnih primjera javljaju se na početku i kraju govora. Pomaci (gdje govor završava) često imaju nisku energiju pa se lako pogrešno klasificiraju kao tišina. Tome možemo doskočiti tako da nakon segmenta govora neko vrijeme držimo oznaku kao govor dok ne budemo sigurni da je govor završio [17].

3.1.2. Govornih u bučnom okruženju

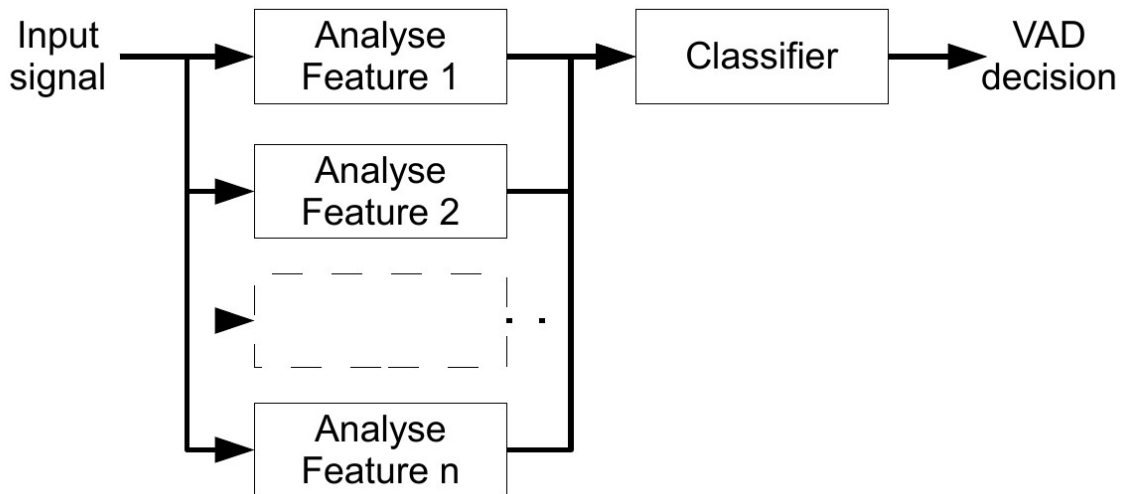
Govor bez pozadinske buke je rijetkost, stoga su nam potrebne naprednije metode. Moramo identificirati karakteristike koje razlikuju govor od buke. Mjere za te karakteristike nazivaju se značajkama. Značajkama pokušavamo izmjeriti neko svojstvo signala koje bi dalo naznaku je li signal govorni ili šum. Korisna značajka može biti energija signala budući da energija govora varira. Niskofrekventne signale prepoznajemo kao glasovni govor, a šumove kao visokofrekventne signale. Također, zvukovi govora mogu se učinkovito modelirati linearnim predviđanjem. Ako je pogreška predviđanja mala, tada je vjerojatno da je signal govor, a ako je pogreška predviđanja velika, tada se vjerojatno radi o šumu. Glasovni govor po definiciji ima istaknutu visinu pa, ako možemo identificirati istaknutu visinu u rasponu, onda je to vjerojatno glasovni govor. Govorne informacije učinkovito se opisuju njihovom spektralnom ovojnicom. MFCC¹ se može koristiti kao opis informacija omotnice i stoga je koristan skup značajki. Parametri linearnog predviđanja (posebno rezidualni predviđanja) također opisuju informacije omotnice i stoga se također mogu koristiti kao skup značajki.

Karakteristika značajki govora je da brzo i često variraju. Gledajući prvu i drugu derivaciju, možemo dobiti informacije o brzini i promjeni signala [17].

¹ Mel Frequency Cepstrum Coefficient (MFCC) dizajniran je za modeliranje značajki audio signala i naširoko se koristi u raznim područjima. MFCC se može izračunati provođenjem pet uzastopnih procesa, redom, kadriranjem signala, izračunavanjem spektra snage, primjenom Mel grupe filtara na dobivene spektre snage, izračunavanjem vrijednosti logaritma svih grupa filtara i na kraju primjenom DCT-a. [18]

3.1.3. Klasifikatori

Nakon što smo prikupili skup indikatora govora i značajke, idući nam je korak (Slika 3.2) spajanje informacija iz tih značajki kako bismo donijeli odluku između govora i šuma. Klasifikacija je generički problem, a moguća su rješenja razna.



Slika 3. 2 Sustav za detekciju govora [17]

Stablo odluke jedno je od prvih klasifikatora za binarne probleme. Radi na principu slijednih binarnih odluka koje vode prema krajnjoj odluci. Kao klasifikator, vrlo je jednostavan za implementaciju, ali i vrlo nefleksibilan. Zahtijeva ručno podešavanje, može postati vrlo složeno i samo jedan šum može uzrokovati krivu krajnju odluku. Nešto boljim klasifikatorima pokazali su se linearni klasifikatori. Umjesto ručno podešenih, binarnih odluka, možemo koristiti promatrane podatke za izradu statističke procjene. Korištenje podataka za obuku automatiziralo bi podešavanje modela. Točnost se može poboljšati dodavanjem više podataka. Zamjenom binarnih odluka možemo dopustiti da tendencije u nekoliko značajki poboljšaju točnost. Linearni klasifikatori pokušavaju postići odluku kao težinski zbroj značajki. Cilj nam je pronaći optimalnu težinu svake pojedine značajke. Pronalazak optimalnih težina značajki vrlo je jednostavan. Definiramo funkciju cilja koju potom želimo minimizirati. Polazna će nam točka biti minimiziranje pogreške klasifikacije. Na kraju svih izračuna, dobit ćemo vektor težina svih značajki unutar okvira. Potrebno je odrediti granicu između klasa kako bi se moglo odrediti koje dobivene vrijednosti pripadaju kojoj klasi. Cijelu izlaznu funkciju možemo transformirati u sigmoidu koja je definirana na domeni $[0, 1]$ pa možemo dobiti i vjerojatnost s kojom okvir pripada govoru ili šumu.

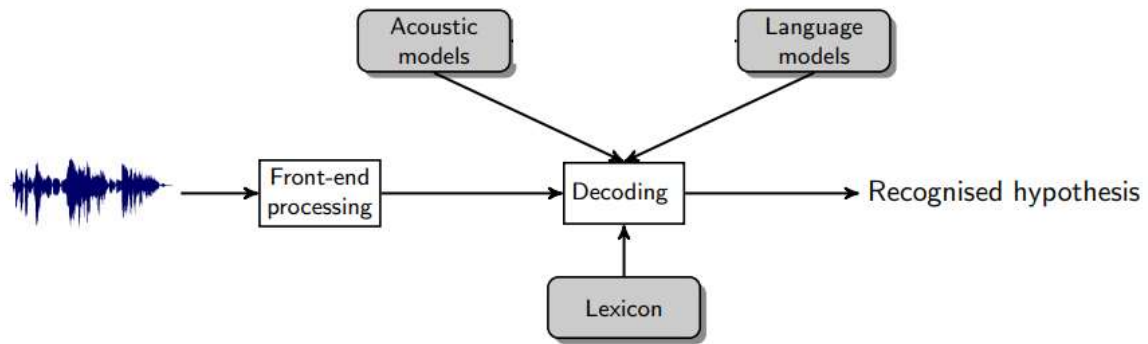
No, danas postoje i mnogo napredniji, bolji i složeniji klasifikatori kao što su linearna diskriminantna analiza, Gaussovi modeli mješavine, duboke neuronske mreže, k-najbližih susjeda (k-NN), stroj potpornih vektora (SVM), klasifikatori slučajnih šuma... Problem kod ovakvih klasifikatora je složenost treniranja, no unatoč tome, zbog visoke preciznosti, danas se većinom koriste modeli dubokog učenja za detekciju glasovne aktivnosti [17, 19].

3.2. Pretvaranje govora u tekst

Automatsko prepoznavanje govora (ASR) omogućuje kvalitetnu komunikaciju ljudi i računala. Krenuvši od najranijih početaka, sustavi za prepoznavanje govora radili su na statističkom principu, proizvodeći najvjerojatniji niz riječi s obzirom na dolazni govorni signal. Problem prepoznavanja govora definiran je kao strojno pretvaranje izgovorenih iskaza u tekstualne rečenice. U statističkom okviru, Bayesovo pravilo odlučivanja koristi se za pronalaženje najvjerojatnijeg niza riječ, s obzirom na redoslijed opažanja. Slijedeći Bayesovo pravilo, posteriorna vjerojatnost može se izraziti kao uvjetna vjerojatnost niza riječi s obzirom na akustička opažanja pomnožena prethodnom vjerojatnošću niza riječi i normalizirano graničnom vjerojatnošću nizova opažanja [17].

3.2.1. Dijelovi automatskog prepoznavanja govora

Pojednostavljeno, kako je prikazano na Slika 3.3, arhitektura sustava za automatsko prepoznavanje govora sastoji se od akustičnog modela, jezičnog modela i dekodera. Cijeli se sustav nadograđuje na ekstrakciju značajki opisanu u prethodnom potpoglavlju. Akustični model sadrži statistički prikaz različitih glasova koji čine svaku riječ u jezičnom modelu ili gramatici. Svaki poseban glas odgovara fonemu. Jezični model sadrži vrlo velik popis riječi i njihovu vjerojatnost pojavljivanja u danom nizu. Na kraju, nalazi se dekodер, program koji uzima zvukove koje korisnik izgovori te potom pretražuje akustični model dok ne nađe podudaranje u zvukovima. Potom, dekodер određuje fonem koji odgovara zvuku te prati podudarne foneme dok ne dođe do stanke u govoru korisnika. Zatim pretražuje jezični model za ekvivalentni niz fonema. Ako se pronađe podudaranje, vraća tekst odgovarajuće riječi ili izraza pozivnom programu [17]. Više o arhitekturi i načinu rada bit će u potpoglavlju 3.2.5 gdje će biti objašnjena arhitektura sustava koji se koristio pri implementaciji.



Slika 3. 3 Arhitektura ASR sustava [17]

3.2.2. Tipovi automatskog prepoznavanja govora

Sustave za prepoznavanje govora moguće je klasificirati prema ograničenjima koja su uzeta u obzir pri njihovom razvijanju i koja su posljedično nametnuta korisnicima.

Prvo je ograničenje ovisnost o govorniku. Takav sustav za prepoznavanje govora zahtijeva od korisnika uključivanje u njegov razvoj, dok sustavi neovisni o govorniku to ne zahtijevaju. Sustave neovisne o govorniku može koristiti svatko. Sustavi ovisni o govorniku obično rade mnogo bolje od sustava koji to nisu. Činjenica je da se akustične varijacije među različitim govornicima vrlo teško mogu opisati i modelirati. Postoje pristupi kako sustav učiniti neovisnim od govornika. Prvi je upotreba višestrukih prikaza za svaki primjer kako bi se uhvatila varijacija među govornicima, a drugi je pristup prilagodbe govornika.

Zatim, sustav može biti ovisan o vrsti izgovora. Sustav može prepoznati svaku riječ neovisno. Sustav od korisnika može zahtijevati da izgovori svaku riječ u rečenici odvajajući ih umjetnom stankom ili može dopustiti korisniku da govori prirodno. Prva vrsta sustava je kategorizirana kao izolirani sustav za prepoznavanje riječi. To je najjednostavniji oblik strategije prepoznavanja. Može se razviti pomoću akustičkih modela temeljenih na riječima bez ikakvog jezičnog modela. No, ako vokabular povećava rečenice sastavljene od izoliranih riječi koje treba prepoznati, upotreba akustičnih modela podriječi i jezičnih modela postaje važna. Druga podvrsta sustava ovisnih o vrsti izgovora sustavi su kontinuiranog prepoznavanja govora. Omogućuje korisnicima izgovaranje poruke na relativno ili potpuno nesputan način. Takvi sustavi moraju biti sposobni dobro raditi u prisutnosti svih popratnih učinaka koji se događaju prilikom artikulacije. Glavni problem kontinuiranog govora nejasne su granice između riječi. Stoga je razvoj sustava kontinuiranog prepoznavanja govora znatno teži zadatak.

Ograničenje na sustav može biti i veličina vokabulara. Sustave možemo klasificirati na male (od jedne do devedeset i devet riječi), srednje (od sto do devetsto devedeset i devet riječi) te velike (od tisuću ili više riječi). Očekivano, sustavi s velikim vokabularom rade lošije od onih s malim. Mali sustavi ne zauzimaju previše memorije te se svaka riječ može modelirati. Problem velikih sustava je što treba skupiti dovoljno primjera govora za treniranje i osigurati velike količine memorije za pohranu parametara. Zato se, kod razvoja takvih sustava, koriste osnove riječi ili dijelovi riječi. Upotreba samo dijelova riječi dovodi do degradacije performansi.

Poseban izazov predstavlja prepoznavanje spontanog govora. Spontani govor obilježavaju nesigurni početci rečenice, poštalice, neograničen vokabular, nedovršene rečenice, duge stanke i pogrješni izgovori. Nasuprot spontanom govoru imamo pročitani govor koji je puno jednostavniji za sustave prepoznavanja govora.

Za kraj, sustavi su ograničeni vrstom i količinom buke. Sustav može biti razvijen s pretpostavkom da ulazni govor bude čist od buke iz okoline, akustičnih izobličenja, mikrofona i izobličenja kanala prijenosa ili može idealno riješiti bilo koji od ovih problema. Dok trenutni uređaji za prepoznavanje govora daju prihvatljivu izvedbu u pažljivo kontroliranim okruženjima, njihova izvedba brzo opada kada se primjenjuju u bučnim okruženjima. Buku mogu stvarati drugi govornici, razni strojevi, ali i sam govornik u obliku cmokanja usnama, kašljanja ili kihanja [17].

3.2.3. Modeli za prepoznavanje govora velikog vokabulara (LVCSR)

Kako bi kasnije bilo jednostavnije razumjeti model koji se koristi u implementaciji rješenja rada, potrebno je upoznati se s teorijom i osnovnim pojmovima modela za prepoznavanje govora velikog vokabulara. Za početak, modele možemo podijeliti na povijesno značajne, modele temeljene na skrivenim Markovljevim lancima (HMM) i novije *End-to-End* modele.

Model temeljen na HMM dugi je niz godina bio model s najvećom točnošću za prepoznavanje govora velikog vokabulara (eng. Large Vocabulary Speech Recognition, LVCSR). U sustavima temeljenim na HMM-u, HMM se koristi za dinamičko vremensko iskrivljenje (DWT) na razini okvira. Uz navedeno, takav se model sastoji od tri dijela: akustičnog, izgovornog i jezičnog modela. Ti su modeli međusobno neovisni i igraju karakterističnu ulogu.

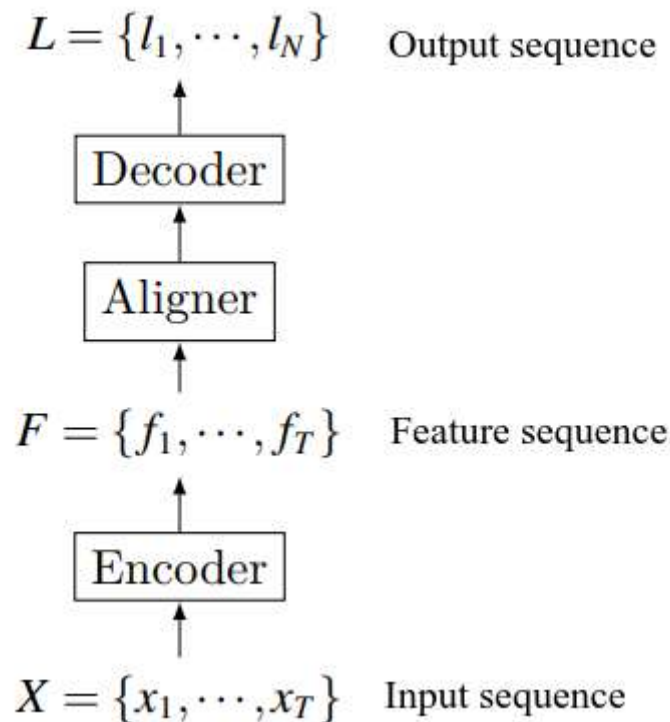
Akustični model modelira preslikavanje između govornog unosa i niza značajki. Vjerojatnost opažanja u akustičnom modelu predstavljena je modelom Gaussovih mješavina (GMM). HMM-GMM struktura bila je opća struktura mnogih sustava za prepoznavanje govora dugi niz godina. No, s razvojem dubokog učenja, umjesto modela Gaussovih mješavina, za računanje posteriornih distribucija vjerojatnosti skrivenog stanja, dominantni postaju modeli dubokih neuronskim mreža (DNN).

Model izgovora preslikava foneme u grafeme. Glavni mu je cilj uspostaviti vezu između akustičkog slijeda i jezičnog slijeda. Rječnik uključuje različite razine preslikavanja. Rječnik se koristi za postizanje strukturnog preslikavanja i preslikavanje odnosa izračuna vjerojatnosti.

Jezični model preslikava slijed znakova u tečnu konačnu transkripciju. To radi tako što sadrži pojednostavljene sintaktičke informacije. Njegov je cilj predvidjeti vjerojatnost pojavljivanja određenih riječi jedne za drugom u određenom jeziku. Tipični sustavi za prepoznavanje govora koriste modele jezika n-grama. N-gram sadrži prethodnu vjerojatnost pojavljivanja riječi (unigram) ili niza riječi (bigram, trigram itd.).

Model temeljen na HMM ima i svoja ograničenja. Treniranje je dugo, a proces globalne optimizacije težak. Svaki od gore predstavljenih modula može koristiti različite metode treniranja i različite skupove podataka te je svaki od njih neovisno optimiziran s vlastitim funkcijama. Te se funkcije općenito razlikuju od stvarnih kriterija za procjenu performansi LVCSR-a pa optimalnost svakog modula ne mora značiti globalnu optimalnost. Još jedan od problema je i pretpostavka uvjetne neovisnosti. Model temeljen na HMM-u koristi pretpostavke uvjetne neovisnosti unutar HMM-a i između različitih modula, a to ne odgovara stvarnom stanju LVCSR-a.

Zbog navedenih nedostataka, sve više radova počelo je proučavati *end-to-end* LVCSR. *End-to-end* model je sustav koji izravno preslikava ulazni audio niz u niz riječi ili drugih grafema. Arhitektura *end-to-end* modela prikazana je na Slika 3.4. Prednost nad modelima temeljenim na HMM-u je što se ne sastoje od više modula, već od cjelovite duboke mreže. *End-to-end* modele generalno možemo podijeliti u 3 kategorije, ovisno o implementaciji. Prva se kategorija temelji na konekcionističkoj vremenskoj klasifikaciji (CTC). Pri pokušaju modeliranja značajki vremenske domene korištenjem rekurentne ili konvolucijske neuronske mreže umjesto HMM-a, dolazi do problema usklađivanja podataka. Naime, funkcije gubitka rekurentne i konvolucijske neuronske mreže definirane su u svakoj točki sekvence pa je, kako bi se moglo izvesti treniranje, potrebno znati odnos poravnanja između



Slika 3. 4 struktura end-to-end modela [17]

izlazne sekvence i ciljne sekvence. CTC je funkcija gubitka, ali rješava težak problem poravnanja tijekom izračuna gubitka. Rješava problem usklađivanja tako da se DNN može koristiti za modeliranje značajki vremenske domene, što uvelike poboljšava ulogu DNN-a u zadacima LVCSR-a. Također, na izlazu izravno daje ciljne transkripcije. No i CTC ima svoje nedostatke. CTC ne može modelirati međuovisnosti unutar izlazne sekvence jer pretpostavlja da su izlazni elementi neovisni jedni o drugima. Stoga CTC ne može naučiti jezični model, već samo akustični. Nemoćan je za scenarije gdje je izlazna sekvenca duža od ulazne jer može samo mapirati ulazne sekvence u izlazne sekvence koje su od njih kraće. Zato se predlaže koristiti pretvornik s rekurentnom neuronskom mrežom. Teoretski, može preslikati ulaz u bilo koji konačni, diskretni izlazni niz. Međuovisnosti ulaza i izlaza te unutar izlaznih elemenata također se zajednički modeliraju. Te ga značajke stavljaju u prednost pred CTC-om [17].

3.2.4. Vrste pogrešaka i evaluacija

Unatoč svim naporima i tehničkim dostignućima, sustavi prepoznavanja govora daleko su od savršenih. Razni su uzroci zbog kojih se pogreške događaju. Problem trenutnih sustava je što imaju ograničene rječnike pa nisu u stanju prepoznati riječi izvan vokabulara korištenog za treniranje. U takvim slučajevima sustav najčešće prepozna najbližnju riječ iz

vokabulara. Također, homofoni (riječi istog izgovora, drugačijeg značenja) rade probleme zbog moguće zamjene značenja. No, jezični model koji dobro funkcionira trebao bi razjasniti homofone na temelju konteksta te, ponovno, pogrješke uzrokovane lošim unosima kao što su disfluentnost, pogrješan izgovor samog govornika ili pogrješke koje su napravili akustični modeli.

U konačnici, učinkovitost sustava mjeri se usporedbom pretpostavljenih transkripcija i referentnih transkripcija. *Word error rate* (WER) najčešće je korištena metrika. Dvije sekvence riječi prvo se poravnaju pomoću algoritma za poravnanje niza koji se temelji na dinamičkom programiranju. Nakon poravnanja određuje se broj brisanja (D), zamjena (S) i umetanja (I). Brisanja, zamjene i umetanja smatraju se pogrješkama, a WER se izračunava omjerom broja pogrješaka prema broju riječi (N) u referenci, kako je i pokazano u formuli (1). *Sentence Error Rate* (SER) ponekad se koristi za procjenu performansi. SER izračunava postotak rečenica s najmanje jednom pogrješkom [17].

$$WER = \frac{I + D + S}{N} * 100\% \quad (1)$$

3.2.5. Google primjer

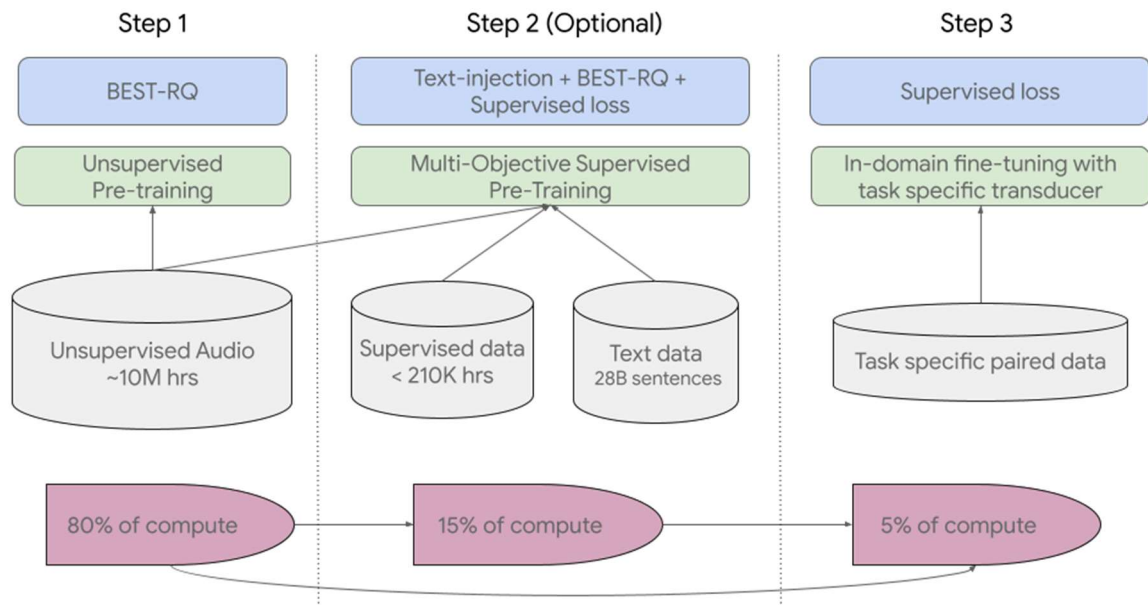
Razvoj modela za prepoznavanje govora izrazito je računalno skup i vremenski dugotrajan proces. Dobra strana je što postoje već razvijeni modeli koji su lako dostupni. Sada, kada je objašnjena teorijska podloga i glavni koncepti sustava za prepoznavanje govora, može se prikazati model koji će koristiti u implementaciji rješenja. Naime, Google nudi mogućnost korištenja svojih *state-of-the-art* modela preko API-ja.

Istraživači unutar Googla intenzivno rade na razvijanju sve boljih sustava za prepoznavanje govora. Ambiciozno su najavili izgradnju modela strojnog učenja koji bi podržavao tisuću jezika na svijetu s najviše govornika, donoseći veću uključenost milijardama ljudi širom svijeta. Univerzalni govorni model (USM) obitelji je najsuvremenijih govornih modela s 2B parametrima obučeni na 12 milijuna sati govora i 28 milijardi rečenica teksta koji već obuhvaćaju više od tristo jezika. No, ostvarivanje cilja prepoznavanja tisuću jezika nameće značajne izazove. Konvencionalni pristupi nadziranog učenja imaju problem nedostatka skalabilnosti. Temeljni izazov skaliranja govornih tehnologija na mnoge jezike je dobivanje dovoljno podataka za treniranje visokokvalitetnih modela. Uz konvencionalne pristupe, audio zapisi moraju biti ili ručno označeni, što je dugotrajan i skup proces, ili prikupljeni iz

izvora s već postojećim transkripcijama koje je teško pronaći za jezike malog broja govornika. Nasuprot tome, samonadzirano učenje može iskoristiti samo audio podatke koji su dostupni u mnogo većim količinama na različitim jezicima. Zbog toga je samonadzor bolji pristup za postizanje našeg cilja skaliranja na stotine jezika. Još jedan izazov je da se modeli moraju poboljšati na računalno učinkovit način s obzirom na proširen broj jezika i na veću željenu kvalitetu. Takav bi algoritam trebao moći koristiti velike količine podataka iz različitih izvora, omogućiti ažuriranje modela bez potrebe za potpunom ponovnom obukom i generalizirati na nove jezike i slučajeve upotrebe.

Navedenim problemima istraživači u Googleu pristupaju samonadziranim učenjem i *fine-tuningom*. USM koristi standardnu arhitekturu koder-dekoder, gdje dekoder može biti CTC, RNN-T ili LAS. Za koder, USM koristi Conformer ili *transformer* pojačan zavojima. Ključna komponenta Conformer je Conformer blok, koji se sastoji od modula pozornosti, *feed-forwarda* i konvolucijskih modula. Kao ulaz uzima *log-mel* spektrogram govornog signala i izvodi konvolucijsko poduzorkovanje, nakon čega se primjenjuje niz konformerskih blokova i projekcijski sloj kako bi se dobila konačna umetanja.

Prvi je korak u procesu treniranja samonadzirano učenje govornog zvuka koji pokriva stotine jezika. U drugom izbornom koraku, kvaliteta modela i jezična pokrivenost mogu se poboljšati kroz dodatni, izborni, korak prije obuke s tekstualnim podacima. Odluka o uključivanju drugog koraka ovisi o tome jesu li tekstualni podaci dostupni. USM najbolje radi s korištenjem drugog izbornog korakom. Posljednji korak u procesu obuke je *fine-tuning* nizvodnih zadataka s malom količinom nadziranih podataka. BEST-RQ koji se koristi u prvom koraku, već je pokazao vrhunske rezultate na višejezičnim zadacima i pokazao se učinkovitim pri korištenju vrlo velikih količina nenadziranih audio podataka [20]. U drugom, izbornom, koraku koristi se višestruko nadzirano predtreniranje kako bi se uključilo znanje iz dodatnih tekstualnih podataka. Model uvodi dodatni modul koder za uzimanje teksta kao ulaza i dodatne slojeve za kombiniranje izlaza koder govora i koder teksta, te zajedno obučava model na neoznačenom govoru, označenom govoru i tekstualnim podacima. U trećem se koraku radi *fine-tuning*. Slika 3.5 prikazuje cijeli niz treniranja. Sa znanjem stečenim tijekom predtreniranja, USM modeli postižu dobru kvalitetu sa samo malom količinom nadziranih podataka.



Slika 3. 5 Arhitektura USM [21]

Pogledamo li učinkovitost, vidimo kako unatoč ograničenim nadziranim podacima (nadzirani YouTube podatci uključuju 73 jezika i imaju u prosjeku manje od tri tisuće sati podataka po jeziku), model u prosjeku postiže manje od 30% stope pogriješke u riječi (WER) [21].

3.3. Veliki jezični modeli

Kao što je već rečeno, pojava arhitekture *transformera* u radu [15] donijela je prekretnicu u razvoju dubokog učenja s naglaskom na razvoj jezičnih modela. Veliki jezični modeli (LLM) sustavi su umjetne inteligencije dizajnirani za razumijevanje, generiranje i manipuliranje ljudskim jezikom. LLM-ovi većinski koriste spomenutu *transformer* arhitekturu koja koristi mehanizme samoopažnje za obradu i proizvodnju teksta. Ova arhitektura omogućuje LLM-u da uhvati kontekstualne odnose između riječi na velikim udaljenostima unutar teksta, pridonoseći njihovoj izvanrednoj izvedbi u jezičnim zadacima [15, 22].

Treniranje LLM-a obuhvaća obradu velikih skupova podataka. Skupovi podataka sadržavaju ogromne količine teksta preuzetog iz izvora poput knjiga, članaka i web stranica. Upravo toliko ogromna količina podataka za učenje omogućuje LLM-u poznavanje gramatike, činjenice i donosi mu sposobnost zaključivanja. Te ih karakteristike čine sposobnima za obavljanje zadataka kao što su prevođenje, sažimanje i odgovaranje na pitanja. Proces treniranja sastoji se od nekoliko faza. Uobičajeno su to faza prethodnog treniranja na velikim korpusima za učenje općih jezičnih obrazaca i fino podešavanje na specifičnim zadacima ili

domenama kako bi se poboljšala izvedba u određenim područjima. Tijekom procesa treniranja, modeli uče predviđati sljedeću riječ u rečenici prema kontekstu koji pružaju prethodne riječi. Model to čini tako što pripisuje ocjenu vjerojatnosti ponavljanju riječi koje su raščlanjene na manje nizove znakova (tokenizirane). Ti se tokeni zatim pretvaraju u ugradnje (engl. embeddings), koje su numerički prikazi konteksta. Nakon što se istreniraju na ovim podacima, LLM-ovi mogu generirati tekst samostalnim predviđanjem sljedeće riječi na temelju primljenog unosa i oslanjajući se na obrasce i znanje koje su stekli. No i ta se velika učinkovitost dodatno može povećati *prompt engineeringom*, *prompt-tuningom*, *fine-tuningom* i taktikama poput pojačanog učenja s ljudskim povratnim informacijama (RLHF) kako bi se uklonile predrasude, govor mržnje i činjenično netočni odgovori poznati kao „halucinacije“ koji su često neželjeni nusproizvodi treninga na toliko nestrukturiranih podataka [23]. Problem treniranja LLM-ova leži u potrebnim računalnim i vremenskom resursima. LLM-ovi zahtijevaju značajne računalne resurse, često koristeći snažne grafičke procesne jedinice ili tenzorske procesne jedinice za učinkovito treniranje [22].

U konačnici, možemo zaključiti kako su LLM-ovi dokazali svoju svestranost u bezbroj slučajeva upotrebe i zadataka u raznim industrijama. Proširuju postojeće modele umjetne inteligencije koji se koriste u chatbotovima i virtualnim pomoćnicima. Poboljšavaju interakciju s korisnicima u sustavima korisničke podrške, pružajući odgovore svjesne konteksta koji oponašaju interakcije s ljudskim agentima. Osim toga, odlično stvaraju sadržaj za članke na blogu, marketinške ili prodajne materijale i druge zadatke pisanja. U istraživanju i akademskoj zajednici oni pomažu u sažimanju i izdvajanju informacija iz ogromnih skupova podataka, ubrzavajući otkrivanje znanja. Igraju vitalnu ulogu u prevođenju jezika. Doprinosе pristupačnosti pomažući osobama s invaliditetom, uključujući aplikacije za pretvaranje teksta u govor i generiranje sadržaja u pristupačnim formatima. Od zdravstva do financija, LLM transformiraju industrije pojednostavljuvanjem procesa, poboljšanjem korisničkih iskustava i omogućavanjem učinkovitijeg donošenja odluka temeljenog na podacima. [23]

Najuzbudljivije od svega je što su lako dostupni. Neki LLM-ovi otvorenoga su kôda te se mogu pokretati lokalno [24], a na neke se možemo spojiti putem API-ja [25].

S obzirom da su veliki jezični modeli široko područje, a ovaj rad ne bavi se samo njima, nije potrebno ulaziti u prevelike detalje. U idućem će se potpoglavlju opisati model GPT-3 koji će se kasnije i koristiti u implementaciji.

3.3.1. GPT-3

Generative Pre-trained Transformer 3 veliki je jezični model koji je objavio OpenAI 2020. Predstavljen je u radu *Language Models are Few-Shot Learners* kao treća generacija modela Generative Pre-trained Transformer. GPT-3 karakterizira 175 milijardi parametara, što je preko 100 puta veće od njegovog prethodnika GPT-2. Upravo povećanje parametara omogućuje postizanje izvanrednih poboljšanja u raznim zadacima obrade prirodnog jezika. Kao i većina LLM-ova, izgrađen je na *transformer* arhitekturi koja je postala standard za NLP zadatke. Model je treniran na raznolikom skupu podataka koji uključuje značajan dio interneta, članaka, knjiga i drugih tekstualnih izvora. Treniranje na opsežnom skupu podataka omogućuje GPT-3 generiranje vrlo povezanog i kontekstualno relevantnog teksta. Sposobnost *few-shot* učenja jedan je od najznačajnijih napredaka koje je pokazao GPT-3. U *few-shot* učenju, modelu se daje mali broj primjera (obično manje od deset) za izvođenje zadatka. GPT-3 također može učinkovito upravljati *one-shot* učenjem (samo jednim primjerom) i *zero-shot* učenjem (bez primjera). To znači da GPT-3 može generalizirati i prilagoditi se novim zadacima s minimalnim dodatnim podacima za obuku, što je značajan odmak od tradicionalnih modela koji zahtijevaju velike količine podataka specifičnih za zadatak. Kada je izašao 2020. godine, GPT-3 postizao je *state-of-the-art* performanse na raznim mjerilima bez finog podešavanja. To znači da je GPT-3 konkurentan specijaliziranim sustavima za prevođenje, može točno odgovoriti na pitanja na temelju danog konteksta, može generirati povezan i kontekstualno prikladan tekst, uključujući eseje, priče i članke, pokazuje sposobnost rješavanja osnovnih aritmetičkih problema i pokazuje razinu zdravorazumskog zaključivanja. U mnogim slučajevima, GPT-3 je 2020. postavio nove rekorde za točnost i povezanost generiranog teksta. Sposobnost dobrog obavljanja različitih zadataka pokazuje njegovu fleksibilnost i sposobnost generalizacije [26].

3.4. Sinteza govora

Sustavi za sintezu govora, znani i kao *text-to-speech* (TTS) sustavi, služe za proizvodnju razumljivih govornih signala iz neke vrste reprezentacije ulaznog jezika. Tipični TTS sustav sastoji se od dva osnovna modula za obradu: modula za analizu teksta i modula za sintezu govora.

Modul za analizu teksta odgovoran je za pretvaranje dolaznog teksta u lingvistički prikaz koji kodira informacije o tome kako bi ulazni tekst trebao biti izgovoren. Prvo, modul mora

obraditi tekst u standardizirani format vodeći računa o svim posebnim znakovima ili drugim nedosljednostima. Zatim se tekst mora strukturalno analizirati kako bi se identificirale sintaktičke komponente rečenica. Sve nedosljednosti između pisanoga i govornog jezika (kratice i akronimi, vlastita imena, brojevi) moraju se otkriti i pretvoriti u ispravan format. Na primjer, „50 km” mora se preslikati u „pedeset kilometara”, a ne „pet-nula k-m”. Zatim slijedi pretvorba grafema u foneme, takozvano, zaključivanje ispravnog odgovora koje se radi pomoću konvencija izgovora danog jezika. Konvencije uključuju upute za rješavanje mnogih dvosmislenosti koje prevladavaju u jezicima. Osim posebnosti danog jezika, potrebno je obraditi i strane riječi i tuđice. Zadnji je korak govor učiniti prirodnim i razumljivim. Taj proces uključuje uvođenje potencijalne ritmičke strukture, obrazaca naglaska i intonacijskih znakova u jezični prikaz. U tu svrhu modul za analizu teksta mora interpretirati sintaktička svojstva ulaznoga teksta. Na primjer, sustav mora biti u stanju razlikovati različite vrste rečenica kao što su pitanja od izjava i zaključiti koja bi riječ trebala biti fokusirana u danom rečeničnom kontekstu.

Modul za sintezu govora drugi je ključni modul TTS sustava. Ulaz u modul jezična je reprezentacija koju proizvodi modul analize teksta, a izlaz se sastoji od akustičnih valnih oblika sintetiziranog govora. Pri razvoju algoritama sinteze, cilja se na govorni izlaz visoke razumljivosti nalik prirodnom govoru, bez šuma i artefakata. Neki sustavi mogu uključivati mogućnost korištenja različitih glasova govornika ili stilova govora. U praktičnoj uporabi, računalna složenost sustava također može postati relevantan čimbenik dizajna. Posebice ako sustav mora podržavati produkciju govora u stvarnom vremenu ili opsluživati više korisnika istovremeno. Kvaliteta se govora i razumljivost govornog sintetizatora obično ocjenjuju subjektivnim testovima slušanja. Moderni se komercijalni sintetizatori govora temelje na jednoj od dviju alternativnih tehnika: konkatenativnoj sintezi ili statističkoj parametarskoj sintezi govora [17]. Obje su metode detaljnije opisane u narednim potpoglavljima.

3.4.1. Konkatenativna sinteza govora

Konkatenativna sinteza govora (CSS) jedna je od dviju primarnih modernih tehnika sinteze govora. Osnova CSS-a ulančavanje je unaprijed snimljenih govornih segmenata kako bi se stvorio razumljiv govor visoke kvalitete. Najjednostavniji CSS sustav koji se može zamisliti mogao bi se razviti ulančavanjem unaprijed snimljenih valnih oblika riječi. Upravo bi ulančavanje snimljenih valnih oblika riječi mogao biti i glavni problem. Budući da ne bi bilo koartikulacijskih učinaka između riječi, ulančavanje valnih oblika na razini riječi zvučalo bi

neprirodno. Također, sustav bi bio ograničen i sigurno ne bi obuhvaćao desetke ili stotine tisuća leksičkih jedinica i milijune vlastitih imena koji mogu postojati u bilo kojem jeziku. Nerazumno je da ih pojedinačni govornik može unaprijed sve razumno snimiti. Problem postaje još gori za aglutinativne jezike kao što je finski, gdje se značenja riječi konstruiraju i prilagođavaju proširenjem korijenskih oblika riječi različitim sufiksima. Kako bi se riješili problemi skalabilnosti i koartikulacije, praktični moderni CSS sustavi rade s dijelovima riječi. U jeziku postoji ograničen broj glasova (npr. oko 40-50 za engleski), ali njihove akustičke karakteristike ovise o okolnom kontekstu zbog koartikulacije. Stoga se koriste glasovi ovisni o kontekstu kao što su difoni (dva glasa) ili trifoni (tri glasa). Kako bi se izgradio CSS sustav, govorni skup podataka mora se prvo pažljivo označiti i segmentirati za interesne jedinice. Ti se segmenti zatim mogu pohraniti kao akustični parametri (npr. parametri govornog kodeka) kako bi se uštedio prostor i omogućila laka karakterizacija i manipulacija.

Nakon što se konstruira baza, potrebno je pretvoriti ulazni tekst u specifikaciju koja uključuje niz zvukova koji se sintetiziraju zajedno s dodatnim prozodijskim specifikacijama kao što su visina, trajanje i snaga. Zatim sustav pretražuje veliku govornu bazu podataka kako bi pronašao jedinice koje najbolje odgovaraju ciljanim specifikacijama izvedenom analizom teksta. Svaka se jedinica procjenjuje na temelju ciljnih troškova (odgovarajućim željenim fonetskim i prozodijskim značajkama) i troškova ulančavanja (kako se glatko jedinice mogu spojiti). Napredni algoritmi pretraživanja, poput Viterbijevog algoritma, koriste se za optimizaciju ovog procesa. Cilj je odabrati jedinice koje osiguravaju da konačni govorni izlaz zvuči prirodno i besprijekorno, minimizirajući zvučne nepodudarnosti i artefakte. Pažljivi odabir pomaže u postizanju visokokvalitetnog sintetiziranog govora. Međutim, izlaz sinteze uvelike ovisi o izboru značajki i funkcija koje se koriste za potroškove, kao i o težinama odabranim za svaku značajku. Iako se troškovne funkcije i njihove temeljne značajke mogu u velikoj mjeri dizajnirati na temelju znanja o obradi signala i govora, težine je potrebno prilagoditi putem pokušaja i pogrješaka ili se mogu automatski optimizirati pomoću neke vrste kriterija kvalitete. Prvi predloženi pristup je upotreba *grid search* različitih vrijednosti težine sintetiziranjem iskaza korištenjem ciljanih specifikacija zadržanih iskaza iz baze podataka za obuku, a zatim usporedbom sintetiziranog valnog oblika sa stvarnim valnim oblikom zadržanog izgovora korištenjem objektivne metrike. Zatim se biraju težine koje dovode do najbolje ukupne izvedbe. Drugi je pristup korištenje regresijskih modela za predviđanje najboljih vrijednosti težine. Kepstralna udaljenost i

razlika snage mogu koristiti za procjenu perceptivne kvalitete ulančavanja u linearnom regresijskom modelu. Ponderi linearne regresije tako postaju perceptualno motivirani troškovi. Ciljne težine se određuju traženjem najboljih podudaranja za svaku jedinicu u bazi podataka, bilježenjem podtroškova i udaljenosti te primjenom linearne regresije za predviđanje optimalnih težina specifičnih za foneme, prilagođenih različitim fonetskim kontekstima [17].

3.4.2. Statistička parametarska sinteza govora

Kao alternativa konkatentivnom pristupu, statistička parametarska sinteza govora (SPSS) još je jedan TTS pristup koji je postao vrlo popularan u području govorne tehnologije. Prednost mu je što rješava nedostatak fleksibilnosti, glavno ograničenje konkatentivnih sustava. Govor generira pomoću statističkih modela govora umjesto oslanjanja na unaprijed snimljene segmente. Ti se statistički modeli uče iz govornih korpusa pomoću tehnika strojnog učenja i kodiraju informacije o tome kako se govor razvija kao funkcija vremena u kontekstu zadanog ulaznog teksta. U tom smislu, SPSS sustavi mogu se promatrati kao zrcalna slika ASR sustava: dok ASR sustav pokušava pretvoriti govor iz akustičnih značajki u niz riječi pomoću modela strojnog učenja, SPSS sustav pokušava pretvoriti niz riječi u akustične značajke ili izravno na akustični valni oblik pomoću modela strojnog učenja. Kao i ASR, SPSS se obučava na velikoj količini govornih podataka s njihovim transkripcijama, što rezultira skupom parametara koji opisuju statističke karakteristike govornih podataka. Potpuni SPSS sustav sastoji se od modula za analizu teksta, generiranje značajki i generiranje valnog oblika. Klasični pristup SPSS-u temelji se na kombinaciji arhitekture skrivenog Markovljevog modela Gaussove mješavine (HMM-GMM) za generiranje značajki i vokodera za generiranje valnog oblika. Generiranje značajki u TTS-u pretvara jezična obilježja u akustične signale pomoću akustičnog modela, obično HMM-GMM. Ovaj model pretvara lingvistički unos u akustičke značajke pri fiksnoj brzini okvira, naučene iz govornog korpusa. HMM stanja predstavljaju dijelove jedinica riječi, a GMM modeliraju njihove akustičke karakteristike. Model proizvodi niz akustičnih značajki, koje se zatim koriste za generiranje govornoga signala. Napredni se modeli trajanja često koriste za poboljšanje vremenskog modeliranja govora. Idući je korak generiranje valnog oblika. Valni oblik govora visoke kvalitete sastoji se od kontinuiranih vrijednosti amplitude uzorkovanih na 16 kHz. Na oblik valnog oblika utječe nekoliko čimbenika koji izravno ne pridonose prirodnosti ili razumljivosti govora kao što su pojačanje signala ili karakteristike faze i

amplitude lanca snimanja i prijenosa. Govorni signali su kvazistacionarni u kratkim prozorima (10-30 ms). Ekstrakcija značajki u koracima od 10 ms bilježi strukturu signala s manje varijabilnosti od valnog oblika. *Vocoder* parametrizira valni oblik govora u kompaktne značajke i sintetizira govor iz tih značajki uz minimalan gubitak kvalitete. *Vocoderi* omogućuju analizu i manipulaciju govornih signala te omogućujući promatranje ili indukciju specifičnih pojava u govoru. U SPSS sustavima, *vocoding* se koristi za dobivanje kompaktne i nepromjenjive reprezentacije akustičnog signala. Umjesto izravnog generiranja valnog oblika govora, modul za generiranje značajki smanjuje broj dimenzija značajki *vocodera* koje obuhvaćaju bitna svojstva govora. *Vocoder* zatim obrađuje te značajke kako bi proizveo valni oblik, koristeći operacije koje preokreću proces ekstrakcije značajki i ponovno uvode informacije izgubljene tijekom ekstrakcije, kao što je faza signala. Ova metoda osigurava učinkovitu i kvalitetnu sintezu govora. SPSS model može se prilagoditi proizvodnji govora s različitim karakteristikama ažuriranjem njegovih parametara. Također, modificiranjem HMM-GMM parametara mogu se postići željeni učinci, poput promjene visine tona ili smanjenje artefakata sinteze. Međutim, potencijalni nedostaci uključuju probleme s kvalitetom zvuka zbog statističkog izgladivanja, kvalitetu *vocodera* i izazove u robusnoj procjeni modela iz ograničenih podataka.

Nedavni napredak u umjetnim neuronskim mrežama (ANN) značajno je poboljšao SPSS. WaveNet, generator valnih oblika neuronske mreže, koristi autoregresivni CNN za proizvodnju visokokvalitetnog govora izravno iz valnih oblika. Predviđa sljedeći uzorak govora na temelju prethodnih uzoraka i informacija o uvjetovanju. WaveNet se može trenirati na spektralnim značajkama kako bi se postigla gotovo ljudska prirodnost iako je računski skupo. Neuronske mreže, kao što su duboke *feed-forward* mreže ili LSTM-ovi, također se koriste u fazi generiranja značajki SPSS-a, zamjenjujući HMM-GMM modele. LSTM-ovi su izvrsni u modeliranju vremenskih ovisnosti, rješavanju varijabilnosti između jezičnih specifikacija i ciljnih značajki *vokodera*. *End-to-end* sustavi poput *Tacotron 2* integriraju cijeli proces pretvaranja teksta u govor pomoću neuronskih mreža. *Tacotron 2* obrađuje ulazni tekst modelom ANN od sekvence do sekvence, stvarajući spektrogram koji se ubacuje u *WaveNet* za produkciju govora. Ovakav pristup smanjuje oslanjanje na unaprijed definirane posredne prikaze i alate za analizu teksta te omogućuje implementaciju za bilo koji jezik s dovoljno podataka za obuku. Ovi sustavi su fleksibilni i dopuštaju integraciju dodatnih informacija za prilagodbu karakteristika govora. Problem sustava dubokog učenja leži u potrebi za velikim količinama podataka. Shodno tome, takvi sustavi

imaju visoke zahtjeve za računalnim resursima pri treniranju te vremenskim resursima pri izvođenju. Osim toga, nedostaje im interpretabilnost i transparentnost klasičnih modela kao što su HMM i GMM, što otežava razumijevanje i kontrolu ponašanja modela. Prilagodba novim podacima također je zahtjevnija jer se oslanja na heuristiku, a ne na dobro razumljiva matematička rješenja. Unatoč ovim izazovima, neuralni SPSS sustavi nude značajne prednosti u kvaliteti i fleksibilnosti govora, što ih čini obećavajućim područjem istraživanja i razvoja tehnologije sinteze govora [17].

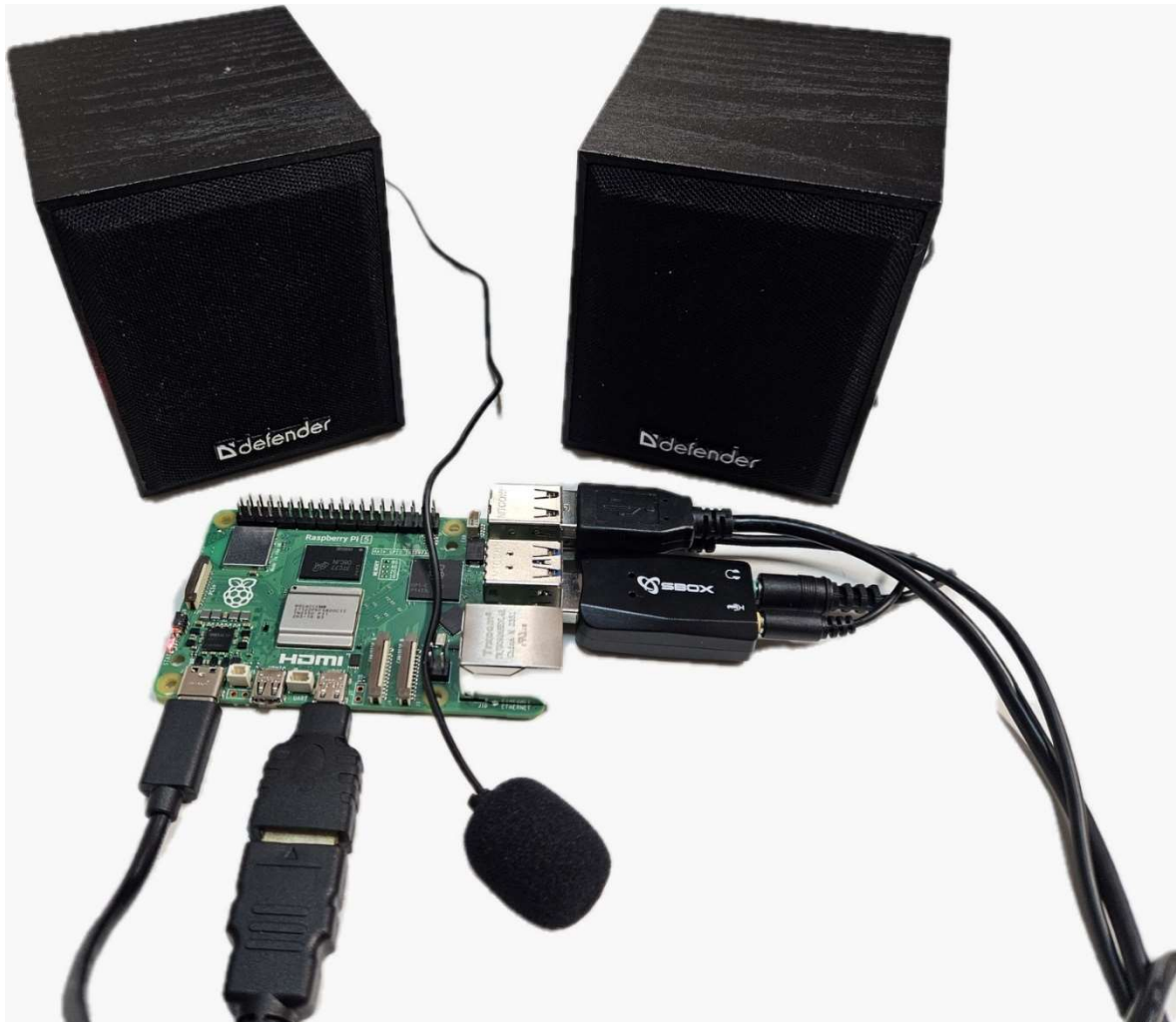
4. Dizajn sustava

Nakon iscrpnog proučavanja literature, potrebno je bilo napraviti plan kako će točno sustav izgledati te koje će se komponente koristiti.

4.1. Hardverske komponente sustava

Kada se počelo raditi na projektu, jednokomponentno računalo Raspberry Pi 4 bilo je odabrano za mozak robota i sav se kôd izvršavao na njemu. Prvotna ideja bila je razvijati i ovaj projekt na njemu. Raspberry Pi 4 nalazilo se već u robotu, većina je ulaza bila zauzeta te se na njega spajalo pomoću VNC preglednika. Stoga je odlučeno kako će se proces razvoja optimalnog rješenja odvijati na alternativnom Raspberry Pi računalu. Za početak, razvoj se odvijao preko Raspberry Pi 3 računala. Iako je Raspberry Pi 3 bilo sasvim dovoljno računalo za izvršavanje ovoga zadatka, odlučilo se kako će se u konačnoj verziji koristiti novi i napredniji model, Raspberry Pi 5.

Na Raspberry Pi 5 bilo je potrebno priključiti mikrofoni i zvučnike kako bi cijeli sustav mogao raditi (Slika 4.1). Problem je bio što korišteni mikrofoni imaju audio izlaz preko kojeg se trebao uključiti u Raspberry Pi 5, a Raspberry Pi 5 (za razliku od Raspberry Pi 3 na kojem se odvijao razvoj), nema audio ulaz. Zbog toga je bilo potrebno mikrofoni uključiti u Raspberry Pi 5 preko pretvornika koji ima ulaze za audio, a USB 2.0 izlaz koji se uključio u Raspberry Pi 5. Isti se pretvornik koristi za priključenje para zvučnika. Još jedna bitna komponenta korištena u radu bila je microSD kartica od 64 GB. Naime, Raspberry Pi nema ugrađenu pohranu poput SSD-a ili hard diska. Uloga microSD kartice je pohrana operacijskog sustava, svih instaliranih programa, programskog kôda i ostalih podataka. Bez microSD kartice koja sadrži odgovarajuće datoteke, Raspberry Pi se ne može pokrenuti. Pri razvijanju rješenja, ta se fleksibilnost pokazala iznimno korisnom jer je bez problema bilo moguće karticu iz jednog Raspberry Pi 5 staviti u drugi i sve je i dalje radilo bez dodatne intervencije. Kako bi išta od toga radilo, Raspberry Pi spaja se na svoj *Power supply* za napajanje. Ideja je Raspberry ugraditi u TIOSS-a. Tada će napajanje dolaziti s istosmjernog pretvarača napona koji na izlazu daje 5 volti, a spojen je na akumulator od 24 volta.



Slika 4. 1 Postavka Raspberry Pi, zvučnika i mikrofona

4.1.1. Raspberry Pi 5

Raspberry Pi niz je mikroračunala koje je u Velikoj Britaniji razvila Zaklada Raspberry Pi. Projekt je nastao na inicijativu promicanja podučavanja informatike u školama. Izvorni model postao je popularniji nego što se očekivalo. Osim za školstvo, prodavao se za robotiku, kućnu i industrijsku informatiku. Posebno je uzbuđenje izazvao kod računalnih i elektroničkih hobista zbog niske cijene, modularnosti, otvorenog dizajna i usvajanja HDMI i USB standarda.

Raspberry Pi 5, korišten za ovaj projekt, izašao je krajem rujna 2023. godine. Iz Zaklada Raspberry Pi tvrde da poboljšanja u hardveru i softveru čine Raspberry Pi 5 više nego dvostruko snažnijim od Pi 4. Naime, ima četverojezgreni ARM Cortex-A76 procesor na 2,4 GHz te 8 GB RAM-a. Nudi dva 4K HDMI izlaza, USB 3.0 i 2.0 priključke, Gigabit Ethernet, Wi-Fi 6 i Bluetooth 5.0 za povezivanje. Mogućnosti pohrane uključuju utor za microSD

karticu i podršku za PCIe 2.0. Dodatna poboljšanja, obzirom na ranije verzije, uključuju poboljšano upravljanje napajanjem, sat stvarnog vremena i GPIO zaglavlje za hardversko sučelje. Ove nadogradnje, jednostavnost korištenja i programibilnost učinile su Raspberry Pi 5 pravim izborom za mozak robota TIOSS [27].

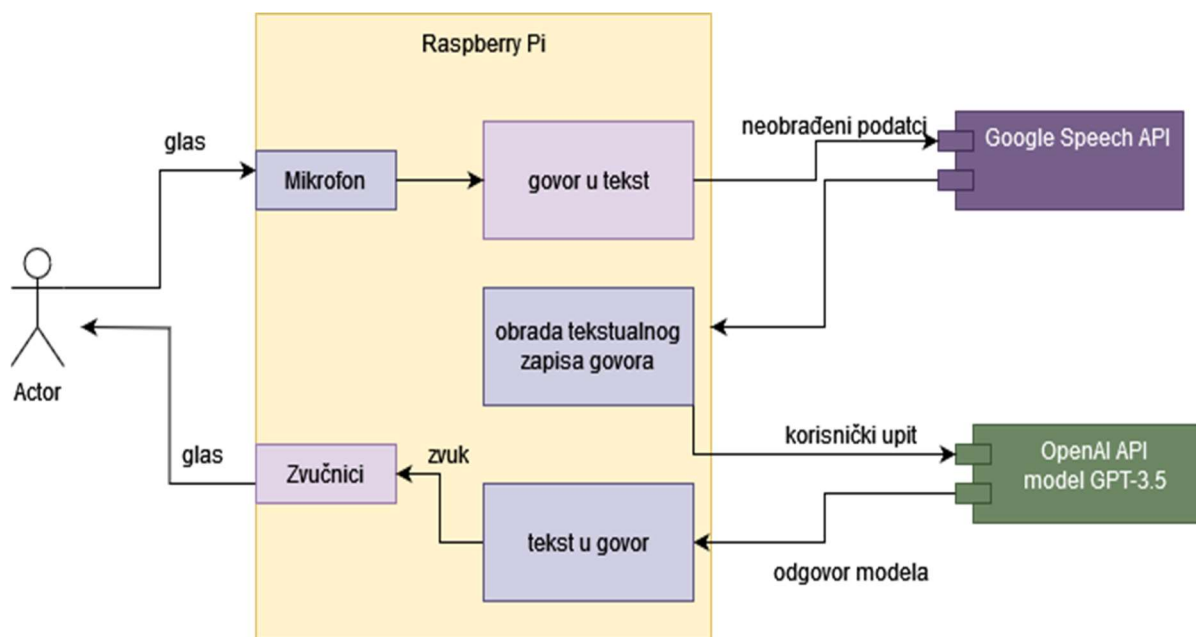
4.2. Softverske komponente

Obzirom na mogućnosti dostupnih komponenata, potrebno je bilo osmisliti sustav koji će raditi tečno, bez prekida, dugih čekanja, ali i bez pretjeranog opterećivanja radne memorije jer nije ovo jedini softver koji će se pokretati na ovom Raspberry Piju. Više o samom procesu razvoja softvera i isprobanim opcijama napisano je u poglavlju 5. Implementacija i testiranje. Ovdje se samo navode osnovne komponente ciljanog softvera.

Za početak bitno je bilo odlučiti koji će se programski jezik koristiti. Programski jezik Python pokazao se kao najboljom opcijom iz više razloga. Iskustveno se zna kako je Python programski jezik opće namjene koji se može koristiti za razne aplikacije te ima ogroman ekosustav biblioteka i modula koji mogu biti korisni u ovom projektu, od biblioteka za strojno i duboko učenje do biblioteka za interakcije s hardverom. Još jedna prednost je što Raspberry Pi OS, službeni operativni sustav za Raspberry Pi, dolazi s predinstaliranim Pythonom. Dodatno, Raspberry Pi zajednica često koristi Python za svoje projekte, što dovodi do mnogih resursa, vodiča i podrške aktivne zajednice.

Ideja je bila pronaći dostupne, po mogućnosti besplatne, API-je koji bi se koristili za prepoznavanje govora, generiranje odgovora i sintezu zvuka. Problem je što Raspberry Pi 5 ima procesor koji nije dovoljan za pokretanje velikih modela koji su potrebni za rad ovoga sustava. Za početak, potrebno je napraviti dio softvera koji će prepoznati mikrofona i skupljati ulazne podatke s njega. Idući modul mora biti modul koji podatke šalje prema API-ju za prepoznavanje govora. Potom s tog API-ja treba preuzeti dobiveni tekst i poslati ga na modul koji komunicira s velikim jezičnim modelom. Dobiveni odgovor potrebno je poslati na modul koji je zaslužan za pretvaranje teksta u razgovijetan govor. S obzirom da su se slični projekti već radili, sigurno postoje određene biblioteke koje pomažu u implementaciji svih ovih modula. Potrebno ih je pronaći, integrirati međusobno i s ostatkom kôda te, u konačnici, dobiti cjelinu koja radi bez poteškoća.

Na dijagramu cijelog sustava (Slika 4.2) jasnije se vide komponente.



Slika 4. 2 Dijagram sustava

5. Implementacija i testiranje

Kao što je rečeno, za implementaciju programskog kôda koristi se programski jezik Python. Još jedan od razloga zašto je odabran Python leži u tome što je Python višeplatformski. To znači da kôd napisan na Raspberry Pi često može raditi na drugim sustavima uz minimalne izmjene. Ova je karakteristika korisna za razvoj i testiranje kôda na stolnom ili prijenosnom računalu prije nego što se postavi na Raspberry Pi.

Razvoj je tekao paralelno radeći na prijenosnom računalu s operacijskim sustavom Linux i isprobavajući odabrana rješenja na Raspberry Pi 3. Zbog upotrebe distribucija temeljenih na Debian GNU/Linux na oba uređaja, nije bilo problema s migracijom i pokretanjem kôda.

5.1. Postavljanje Raspberry Pija i radne okoline

Za početak, potrebno je bilo osposobiti Raspberry Pi za rad i postaviti radne okoline na njemu i prijenosnom računalu. Za radnu okolinu na prijenosnom računalu odabrano je integrirano razvojno okruženje PyCharm koje se koristi upravo za programiranje u Pythonu. Pametno osmišljenim uređivačem kôda omogućuje preglednost pri pisanju, dovršavanje linija kôda, brzu navigaciju kôdom, analizu kôda u stvarnom vremenu i identifikaciju pogriješaka. Također, omogućuje grafički *debugger* s točkama prekida, provjerom varijabli i postupnim izvršavanjem kôda. PyCharm nudi i mnoge druge mogućnosti koje ni nisu bile potrebne za izradu ovoga projekta [28].

Kada je u pitanju Raspberry Pi, prvo je bilo potrebno postaviti operacijski sustav Raspberry Pi OS. Kao što je rečeno, operacijski sustav potrebno je pohraniti na microSD koji se potom umeće u Raspberry Pi. Najjednostavniji način instalacije je koristeći Raspberry Pi Imager. Potrebno je preuzeti Imager koji odgovara operacijskom sustavu i odraditi *flash* na microSD karticu. Nakon što se microSD kartica umetne u, za nju predviđeno mjesto, potrebno je spojiti monitor preko HDMI ulaza te preko USB ulaza ili *Bluetootha* tipkovnicu i miš. Na kraju je potrebno uključiti Raspberry Pi na napajanje. Kada se sustav pokrene, potrebno je prema uputama dovršiti instalaciju [29]. Kada je Raspberry Pi spreman, možemo postaviti razvojno okolinu. Čitajući iskustva razvojne zajednice, PyCharm je prevelik sustav za pokretanje na Raspberryju te se preporuča korištenje razvojne okoline Thonny koja dolazi predinstalirana s operacijskim sustavom. Thonny je vrlo jednostavno integrirano razvojno okruženje koje nudi pregledno grafičko sučelje s uređivačem kôda i jednostavnim

debuggerom [30]. S obzirom da se na Raspberry Piju nije razvijao kôd, već samo testirao, Thonny je služio za preglednu provjeru kôda prije pokretanja i testiranja.

Zadnji korak prije početka razvoja kôda je postavljanje virtualne okoline unutar koje će se nalaziti kôd i sve instalirane biblioteke. Za kreiranje virtualnog okruženja mogu se koristiti alati poput *venv*, koji je uključen u Python standardnu biblioteku ili *virtualenv* i *Anacondu* koje je potrebno instalirati. Za ovaj projekt korišten je *venv*.

5.2. Prepoznavanje zvučnika i mikrofona

Bitna stavka u ovom sustavu su mikrofoni, preko kojega ulaze podatci, i zvučnici, na koje izlazi reproducirani zvuk. Prema knjizi [31], problem povezivanja kôda s mikrofonom i zvučnicima rješava modul *pyAudio* koji pruža povezivanje na višepatformsku audio ulazno/izlaznu biblioteku otvorenog kôda *PortAudio*. Omogućuje pisanje jednostavnih audio programa u jezicima C ili C++ koji će se prevesti i pokrenuti na mnogim platformama uključujući Windows, Macintosh OS X i Unix. Namijenjen je promicanju razmjene audio softvera između programera na različitim platformama. Mnoge aplikacije danas koriste *portAudio* za Audio ulaz/izlaz. Prednost mu je što pruža vrlo jednostavan API za snimanje i/ili reprodukciju zvuka [32]. Na API se spaja *pyAudio* i tako omogućava jednostavno korištenje Pythona za reprodukciju i snimanje audio zapisa na raznim platformama, poput GNU/Linux, Microsoft Windows i Apple macOS. Instalacija biblioteke vrlo je jednostavna. Na distribucijama operacijskog sustava Linux prvo je potrebno instalirati razvojni paket *portAudio* biblioteke i razvojni paket za Python. Potom je moguće instalirati biblioteku *pyAudio* kao što je prikazano u Kôd 1 [33].

```
sudo apt-get install portaudio19-dev python-all-dev python3-  
all-dev &&  
sudo pip install pyaudio
```

Kôd 5.1. Instalacija datoteka za ulaz/izlaz

Jedan od čestih problema koji se javlja prilikom rada s ovom bibliotekom je mikrofoni koji ne sakuplja zvuk i zvučnici koji zvuk ne reproduciraju. Spomenuti se problem javio prilikom testiranja na Raspberry Pi 5. Kako bi se problem riješio, potrebno je bilo instalirati biblioteke *libasound2*, *alsa-utils*, *alsa-oss*. Zatim naredba *amixer* u terminalu otvara ispis kao na Slika 5.1.


```
Simple mixer control 'Master',0
  Capabilities: pvolume pswitch pswitch-joined
  Playback channels: Front Left - Front Right
  Limits: Playback 0 - 65536
  Mono:
  Front Left: Playback 24100 [37%] [on]
  Front Right: Playback 24100 [37%] [on]
Simple mixer control 'Capture',0
  Capabilities: cvolume cswitch cswitch-joined
  Capture channels: Front Left - Front Right
  Limits: Capture 0 - 65536
  Front Left: Capture 17990 [27%] [on]
  Front Right: Capture 17990 [27%] [on]
```

Slika 5. 1 Provjera mikrofona

Ako se za *Capture device* pod zadnja dva polja, *front left* i *front right* nalazi vrijednost nula, uređaj je isključen. Kako bi se moglo upravljati mikrofonom i zvučnicima, u terminal se upisuje naredba `alsamixer`. Kao što se vidi na Slika 5.2, u terminalu se otvara popis uređaja za snimanje i reprodukciju zvuka. Jednostavnim odabirom odgovarajućih opcija koristeći *Fn* tipke možemo prilagoditi uređaje za snimanje i reprodukciju zvuka.



Slika 5. 2 AlsaMixer za namještanje ulazno/izlaznih uređaja

5.2.1. ALSA

Advanced Linux Sound Architecture (ALSA) je softverski *framework* i dio Linux jezgre. Funkcionalnost mu je pružanje API sučelja za rad s upravljačkim programom zvučne kartice. Početna ideja rada na ALSA projektu bila je automatska konfiguracija hardvera zvučne kartice i rukovanje više zvučnih uređaja u sustavu. Zvučni poslužitelji korišteni na Linuxu poput sndio, PulseAudio, JACK i PipeWire rade povrh ALSA-e i njenim upravljačkim programom za zvučne kartice. Također, ALSA je naslijedio starije sučelje *Open Sound System* i podržao njegove funkcionalnosti [34].

5.3. Prepoznavanje govora

Modul za prepoznavanje govora ključan je za uspješnu izvedbu traženoga sustava. Osim samog prepoznavanja govora, bitno je da modul što točnije pretvara ulazni govor u tekst. Prednost implementacije toga modula u Pythonu je dostupnost biblioteka koje omogućuju jednostavno povezivanje s potrebnim modelom. Prvotna ideja bila je pokušati implementirati vlastito rješenje. No, nakon upoznavanja s literaturom i postojećim radovima, uviđeno je kako je za izradu dobrog sustava za pretvaranje govora u tekst potrebno mnogo, trenutno nedostupnih, resursa. Naime, Google za svoj *Universal Speech Models* koristi tri tipa baza podataka u kojima se nalazi nekoliko desetaka milijuna sati audio zapisa s YouTubea i tekstualni zapisi koji se broje u milijardama rečenica [20]. Osim podataka za treniranje, nedostupni su i potrebni računalni resursi. Stoga je daljnji razvoj tekao u smjeru potrage za najboljim dostupnim rješenjem. Prema knjizi [31] preporuča se uporaba biblioteke *Speech Recognition*.

5.3.1. *Speech Recognition* biblioteka

Speech Recognition Python je biblioteka koja olakšava jednostavan pristup različitim mehanizmima za prepoznavanje govora i API-jima, što ga čini nezamjenjivim alatom za raznolik niz aplikacija. Instalira se vrlo jednostavno naredbom `pip install SpeechRecognition`. U svojoj se implementaciji oslanja na biblioteku *PyAudio* pa će ju biti potrebno instalirati ako već nije instalirana. Biblioteka *Speech Recognition* nudi razne korisne značajke. Kompatibilna je s glavnim operativnim sustavima, uključujući Windows, macOS i Linux, osiguravajući svestranost u različitim razvojnim okruženjima. Uz podršku za mikrofonski ulaz, omogućuje hvatanje i obradu audio ulaza u stvarnom vremenu kako je

potrebno za ovaj projekt. Također, omogućuje pristup višestrukim mehanizmima za prepoznavanje govora i API-jima. *CMU Sphinx*, *Vosk API* i *OpenAI whisper* imaju mogućnost izvanmrežnoga rada [35, 36, 37]. Nasuprot njima, *Google Speech Recognition*, *Google Cloud Speech*, *Wit.ai*, *Microsoft Azure Speech*, *IBM Speech to Text* i *Whisper API* za svoj rad nužno trebaju internetsku vezu [35, 38, 39, 40]. Svaki od dostupnih API-ja ima svoje značajke te je potrebno upoznati se s njima prije odabira. Problem kod većine dostupnih API-ja je cijena i nepostojanje podrške za hrvatski jezik. Usporedba je prikazana u Tablica 5.1 [35, 36, 37, 38, 39, 40].

Tablica 5. 1 Usporedba API-ja za prepoznavanje govora

API	Podrška za hrvatski jezik	Plaćanje
CMU Sphinx	ne, ali postoji mogućnost ugradnje	besplatno
Vosk API	ne	besplatno
OpenAI whisper	da	besplatno
Google Speech Recognition	da	besplatno
Google Cloud Speech	da	prema potrošnji
Wit.ai	da	besplatno
Microsoft Azure Speech	da	prema potrošnji
IBM Speech to Text	ne	prema potrošnji
Whisper API	da	prema potrošnji

Prilikom odabira bilo je ključno poštovati resurse Raspberry Pija i ne potrošiti velike iznose na korištenje API-ja. Najbolje bi bilo koristiti besplatnu verziju. Problem kod *OpenAI whispera* je lokalno pokretanje modela koje zahtijeva značajne računalne resurse, posebno za obradu u stvarnom vremenu kakva je potrebna za ovaj projekt. Naime, bazni model koji treba pokrenuti na Raspberryju troši oko 1 GB radne memorije, kolika je bila ukupna radna memorija Raspberry Pi 3 na kojem se u početku testirao sustav. Također, bolji modeli koji omogućuju željenu kvalitetu pretvaranja govora u tekst, zahtijevaju čak 10 GB radne memorije, a to je više nego što Raspberry Pi 5 (na kojem se nalazi krajnji proizvod) uopće ima radne memorije. Osim toga, zahtijeva dodatne instalacije programa i biblioteka te konstantno održavanje [35]. Nadalje, *Wit.ai* podržava hrvatski i omogućeno je besplatno povezivanje na njegov API, no, čitajući iskustva drugih programera, nije ga poželjno koristiti kada *Google Cloud Speech-to-Text* i *Microsoft Azure Speech* pružaju puno bolju podršku za

hrvatski jezik. U knjizi [31], kao i u većini pregledanih internetskih članaka i uputa, preporuča se upotreba *Google Speech Recognition*. No, prilikom odabira, posebno je važno provjeriti koja verzija biblioteke podržava koje API-je jer se popis mijenja s novijim izdanjima u odnosu na verzije unazad zadnjih desetak godina.

No, prije upotrebe ikojeg API-ja, potrebno je proučiti što sve nudi *Speech Recognition* biblioteka. Klasa *Recognizer* pruža veliki skup funkcija za obavljanje prepoznavanja govora. U definiciji klase postavljeni su parametri prikazani u Kôdu 5.2.

```
class Recognizer(AudioSource):
    def __init__(self):
        self.energy_threshold = 300 # minimalna audio
        energija koju treba uzeti u obzir za snimanje
        self.dynamic_energy_threshold = True
        self.dynamic_energy_adjustment_damping = 0.15
        self.dynamic_energy_ratio = 1.5
        self.pause_threshold = 0.8 # sekundi zvuka bez
        govora prije nego što se fraza smatra dovršenom
        self.operation_timeout = None
        self.phrase_threshold = 0.3 # minimalne sekunde
        govornog zvuka prije nego što govorni zvuk smatramo frazom
        self.non_speaking_duration = 0.5 # sekundi zvuka bez govora
        koji će se zadržati snimanje
```

Kôd 5.2

Ako nam ne odgovaraju postavljeni parametri možemo ih lako promijeniti u izvornom kôdu. No, to nije ono što nas zanima u najvećoj mjeri. *Recognizer* sadrži funkcije koje su potrebne za ovaj projekt. Za početak funkcija `adjust_for_ambient_noise()` kojoj je potrebno predati izvor zvuka (izvor zvuka instanca je klase *Microphone*, ali o njoj će riječi biti malo kasnije). Navedena funkcija dinamički prilagođava energetske prag pomoću zvuka iz izvora zvuka kako bi se uzela u obzir okolna buka te bi ju se trebalo koristiti u razdoblju bez govora. Sadrži i parametar *duration* koji predstavlja maksimalan broj sekundi za koje će dinamički prilagoditi prag. U implementaciji je postavljena vrijednost od 0,8 sekundi kako bi se dobio reprezentativan uzorak buke iz okoline. Zatim nam je potrebna funkcija `listen()` koja prima isti izvor zvuka. Snimanje zvuka postiže se čekanjem da zvuk prijeđe prag energije određen vrijednošću `recognizer_instance.energy_threshold` (što znači da je korisnik počeo govoriti). Snimanje se nastavlja sve dok se ne zabilježi tišina duža od `recognizer_instance.pause_threshold` sekundi ili dok ne prestane ulazni audio. U konačnici,

u varijablu sprema se instanca klase *AudioData* koji sadržava neobrađene audio podatke. Te neobrađene audio podatke treba dalje poslati u neku od funkciju za povezivanje s API-jem. Odlučeno je isprobati funkciju `recognize_google_cloud()` koja iako ima naplatu, pruža tri mjeseca besplatnog korištenja. Za njeno korištenje potrebno je izgenerirati API ključ kako bismo se mogli spojiti na Googleov API preko kojega ćemo podatke poslati na obradu na njihove servere i nazad primiti tekst izrečenoga. Iako je na prijenosnom računalu i na optičkom internetu povezivanje radilo tečno i dolazili su ispravni zapisi izrečenoga, na Raspberry Pi 5 nije sve dobro radilo. Naime, predugo se čekao odgovor s API-ja i odlučeno je ipak koristiti nešto slabije modele na koje se može spojiti pomoću funkcije `recognize_google()`. Iako je postojala zabrinutost točnosti izvedbe dostupnih modela, dobili smo poželjnu točnost. Jedina mana je što modeli na koje se spajamo preko `recognize_google()` zahtijevaju dosta sporiji govor i veće naglašavanje pauze između riječi. Preostalo je još objasniti klasu *Microphone*. To je klasa koja komunicira s *PyAudio* modulom i predstavlja fizički mikrofonski uređaj za sakupljanje zvuka [41].

Problem kod implementacije prepoznavanja govora je i u obrađivanju pogriješaka koje se lako mogu dogoditi. Za sada, kada želimo da program stalno sluša i hvata zvuk kada netko krene govoriti u mikrofonski uređaj, pogriješke se obrađuju vrlo jednostavno. Naredbom `pass` jednostavno puštamo dalje izvođenje skripte.

5.3.2. ALSA konfiguracija

Pri pokretanju sustava za prepoznavanje govora, mogu se uočiti ispisi kao na Slika 5.3. Takve se poruke upozorenja javljaju svaki puta kada se pokreće kôd koji koristi *PyAudio* na Linux distribucijama. Za rješavanje nepotrebnih ispisa, potrebno je bilo napisati funkciju prikazanu u Kôd 5.3. Funkcija prvo provjerava radi li sustav na Linuxu. Ako da, uvozi potrebne funkcije iz biblioteke *ctypes* koja se koristi za povezivanje s C bibliotekama u Pythonu. Zatim definira prilagođenu funkciju za rukovanje pogriješcima u Pythonu. Ta funkcija uzima određene parametre, ali ne radi ništa s njima, učinkovito ignorirajući sve pogriješke. Zatim pretvara u format koji mogu koristiti C biblioteke. Sve poruke o pogriješci iz ALSA-e, koje bi se inače pojavile na konzoli, presreće prilagođeni rukovatelj pogriješcima, koji ih zanemaruje. Krajnji rezultat je da su ALSA poruke o pogriješcima potisnute, održavajući izlaz konzole čistim.

```
Listening...
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe
ALSA lib pcm.c:2666:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_route.c:877:(find_matching_chmap) Found no matching channel map
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card
ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'
```

Slika 5. 3 ALSA upozorenja

```
if platform.system() == 'Linux':
    from ctypes import CFUNCTYPE, c_char_p, c_int, cdll

    error_handler = CFUNCTYPE(None, c_char_p, c_int,
                               c_char_p, c_int, c_char_p)

    def py_error_handler(filename, line, function, err, fmt):
        pass

    c_error_handler = error_handler(py_error_handler)
    asound = cdll.LoadLibrary('libasound.so')
    asound.snd_lib_error_set_handler(c_error_handler)
```

Kôd 5.3

5.4. Generiranje odgovora pomoću velikog jezičnog modela

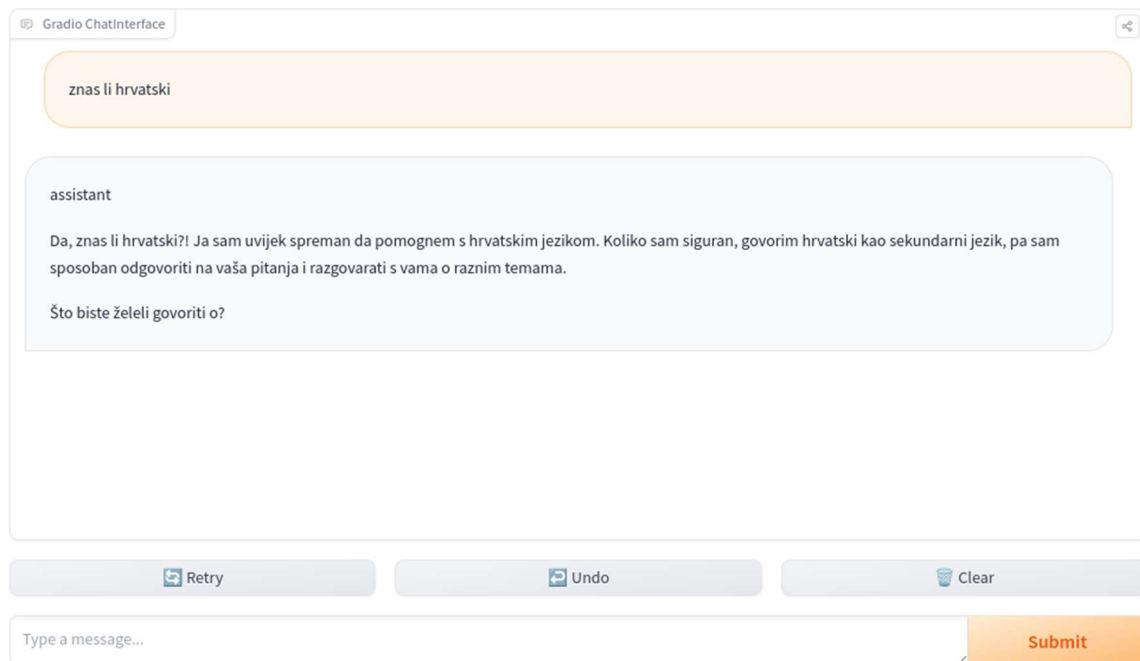
Kako bi cijeli sustav imao smisla, potrebno je pronaći jezični model koji će kvalitetno odgovarati na sugovornikova pitanja. Prvotna je ideja bila pronaći neki provjereni model koji je otvorenoga kôda i može se lokalno pokrenuti. Najbolje mjesto za krenuti s istraživanjem bila je stranica Hugging Face. Ukratko, Hugging Face tvrtka je koja nudi širok spektar alata i biblioteka za obradu prirodnog jezika te druge zadatke strojnog i dubokog učenja. Biblioteka *Transformers* njihov je najpopularniji proizvod. Biblioteka sadrži unaprijed obučene modele za zadatke poput generiranja teksta, klasifikacije, prijevoda, sažimanja i slično. Osim toga, na stranici postoji i Hugging Face Model Hub, platforma na kojoj korisnici mogu dijeliti i pronalaziti unaprijed istrenirane modele. Moguće je pronaći modele za različite zadatke i koristiti ih izravno u svojim aplikacijama.

5.4.1. Isprobavanje modela

Istraživanje je bilo usmjereno prema nekim od najpoznatijih modela. Za početak odlučeno je isprobati LLM Llama2, u tom trenutku *state-of-the-art* model tvrtke Meta. U medijima se modelu dalo dosta prostora jer je pokazao dobre performanse te je otvorenoga kôda. Nakon dobivenog pristupa modelu i pokretanja na osobnom računalu, uočen je prvi problem. Za lokalno pokretanje i najmanje inačice modela, potrebna je grafička kartica, a to na Raspberry Pi ne postoji. Zatim je uočen drugi problem. Naime, Llama 2 ne podržava nijedan jezik osim engleskoga. Iako u svojim podacima za treniranje ima i podatke na drugim jezicima, nije prilagođena drugim jezicima [42]. Zatim je izašla Llama 3 koja također ne podržava jezike osim engleskog. Na Hugging Face stranici omogućeno je isprobavanje oba modela. Razgovor s Llama 3 modelom izgledao je kao na Slika 5.4. Nakon ovoga, odustalo se od Llama modela.

Idući pokušaj bilo je testiranje Phi2 modela tvrtke Microsoft. Phi2 je pri svom izlasku privukao pažnju jer je najavljen kao dosta manji model od ostalih LLM-ova te je postojala nada kako će njega biti moguće pokrenuti na Raspberry Piju. No, ponovio se problem postojanja podrške samo za engleski jezik. Pokušaj konverzacije prikazan je na Slika 5.5. Osim što halucinira i ne podržava hrvatski, modelu je trebalo 5,5 sekundi za generiranje odgovora, a to je za tečnu komunikaciju predugo čekanje [43].

Idući pokušaj bio je model Mistral, model europske tvrtke Mistral AI. Osim što je javnosti predstavljen kao model odličnih performansi, otvorenog je kôda i dostupan svima na korištenje. No, kao što se vidi na Slika 5.6, ni Mistral nije pokazao najbolje ponašanje na hrvatskome jeziku [44].



Slika 5. 4 Testiranje Llama 3



Slika 5. 5 testiranje phi2



This model can be loaded on Inference API (serverless).

Slika 5. 6 testiranje Mistrala

Iduća ideja bila je pokušati iskoristiti neki od spomenutih modela, pritom koristeći Python biblioteku *Translate*. Biblioteka omogućava prevođenje u *real-time*. No, kao što se vidi na Slika 5.7, tekst za prevođenje je ograničenog broja znakova. Nije moguće onda koristiti tu biblioteku jer ne znamo koliko dugačke odgovore će model izbacivati, a ni koliko će dugačke upite ljudi postavljati robotu.

```
(pokusaj1) kilicic@pop-os:~/faks/projektDiplomski/probe$ translate-cli -f hr "Fakultet elektrotehnike i računarstva (FER) u Zagrebu jedan je od najvećih i najvažnijih fakulteta Sveučilišta u Zagrebu i jedan od vodećih fakulteta u regiji na području elektrotehnike, računarstva i informacijske i komunikacijske tehnologije. Nalazi se u Zagrebu, Hrvatska.

FER nudi širok spektar studijskih programa na preddiplomskoj, diplomskoj i poslijediplomskoj razini, uključujući specijalizirane studijske programe u različitim granama elektrotehnike i računarstva. Fakultet je poznat po svojem visokom standardu u obrazovanju, istraživanju i razvoju, snažnim industrijskim partnerstvima te aktivnom sudjelovanju u međunarodnim projektima i suradnjama.

Studenti FER-a imaju priliku raditi na raznovrsnim projektima, sudjelovati u laboratorijskim vježbama koje se provode u suvremeno opremljenim laboratorijima i stjecati praktična znanja koja su važna za njihovu buduću karijeru. Fakultet također potiče sudjelovanje u studentskim organizacijama i različitim izvanakademskim aktivnostima, što doprinosi razvoju njihovih profesionalnih i osobnih vještina." -t en

Translation: QUERY LENGTH LIMIT EXCEEDED. MAX ALLOWED QUERY : 500 CHARS which contributes to the development of their professional and personal skills.
-----
Translated by: MyMemory
(pokusaj1) kilicic@pop-os:~/faks/projektDiplomski/probe$
```

Slika 5. 7 translate biblioteka

Potom se pokušalo raditi s XLM-RoBERTa, modelom razvijenim u Facebooku još 2019. Glavna značajka zašto se pokušao upravo taj model iskoristiti je podrška za hrvatski jezik. Naime, XLM-RoBERTa bazni model zahtjeva oko 6 GB RAM-a i GPU za izvođenje i treniranje [45].

5.4.2. GPT-3.5

Nakon svih neuspjelih pokušaja, idući je pokušaj bio spajanje na OpenAI API. Od samog početka se znalo kako će to možda biti najbolja moguća opcija što se tiče točnosti odgovora. Sa ChatGPT-jem koji koristi modele nama dostupne preko API-ja, komuniciramo svakodnevno.

Za početak, potrebno je bilo upoznati se s mogućnostima API-ja, odabrati koje nam sve mogućnosti trebaju i na kraju isprobati radi li dobro za potrebe ovoga projekta. Na OpenAI Platformi tvrde kako se njihov API može primijeniti na gotovo svaki zadatak. Ponuđen je set modela različitih mogućnosti i cijena. Također, bilo koji od tih modela moguće je *fine-tuningom* prilagoditi vlastitom slučaju. Set modela dinamički se mijenja, novi modeli zamjenjuju stare jer dolaze sve jači i specijaliziraniji modeli za određeno područje. Glavni modeli koji se preporučuju za uporabu su GPT-4o, GPT-4 Turbo i GPT-3.5 Turbo. GPT-3.5 Turbo brzi je i najjeftiniji model za jednostavne zadatke poput komunikacije. Posebno je specijaliziran za *chat* proizvode, uspješno radi na raznim jezicima te se u bazičnim zadacima ne razlikuje mnogo od GPT-4 modela. Cijenom je pristupačniji jer ne može raditi sa slikom i videom, ali to za ovaj zadatak nije ni potrebno.

Na Platformi se nalaze i upute za rad s API-jem kako bi se razvojnim programerima pomoglo pripremiti lokalnu okolinu za rad. Prvi korak je kreiranje računa i generiranje tajnog API ključa. Zatim je, u postojeće virtualno okruženje potrebno instalirati `openai` biblioteku naredbom `pip install --upgrade openai`. Kada je biblioteka uspješno instalirana i API ključ generiran, moguće je spojiti se na modele iz kôda kako je prikazano u Kôd 5.4. [46]

```
from openai import OpenAI

client = OpenAI(api_key="key")
messages = [{"role": "system",
              "content": "Zoveš se Tioš. Ti si robot razvijen
na Fakultetu elektotehnike i računarstva u Zagrebu. Odgovaraš
jednostavnim odgovorima."}]

def generate_chatgpt_response(user_input):
    messages.append({"role": "user", "content": user_input})
    response = client.chat.completions.create(
```

```

        model="gpt-3.5-turbo",
        messages=messages
    )
    answer = response.choices[0].message.content
    messages.append({"role": "assistant", "content": answer})
    return answer

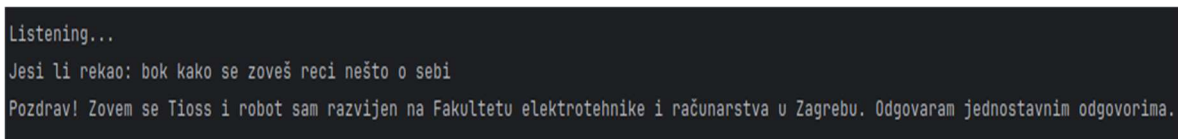
```

Kôd 5.4

Nakon uvoza biblioteke i postavljanja API ključa, potrebno je inicijalizirati varijablu *messages*. Uglavnom se radi o varijabli tipa rječnik (eng. Dictionary). U prvoj poruci sistemu, *chatbotu*, dajemo upute o tome koja je njegova uloga u ovom razgovoru. U rječnik se sprema konverzacija po ulogama *user* i *assistant* kako bi model imao kontekst razgovora kao što ga ima i ChatGPT. Važna napomena je da nije preporučljivo ukodirati API ključeve u svom kôdu kao što je prikazano, nego koristiti varijable okruženja ili neku drugu taktiku, no ovdje je tako napravljeno samo radi ilustracije bitnih koraka u povezivanju na API.

Idejno je zamišljeno da se skripta pokreće svaki puta kada netko započne novi razgovor pa se i varijabla *messages* postavi na početno stanje. No, ta ideja za sada nije implementirana pa varijablu *messages* postavi na početno stanje kada čovjek koji razgovara s TIOSS-om kaže samo „gotovo”. Razlog tome je sprječavanje nepotrebnog gomilanja sadržaja razgovora između TIOSS-a i različitih osoba.

Bazična konverzacija prikazana je na Slika 5.8.



```

Listening...
Jesi li rekao: bok kako se zoveš reci nešto o sebi
Pozdrav! Zovem se TioSS i robot sam razvijen na Fakultetu elektrotehnike i računarstva u Zagrebu. Odgovaram jednostavnim odgovorima.

```

Slika 5. 8 konverzacija s TIOSS-om

5.4.3. Prilagodna i učenje modela

Kada je omogućena bazična komunikacija, cilj je bio TIOSS-a naučiti informacije o njegovoj prošlosti, nastanku, tvorcima, ali i o ljudima koji trenutno rade na njemu te koje sve nove mogućnosti ima. Postoji više načina kako model naučiti dodatne informacije koje su specifične za TIOSS-a i ovaj zadatak. Optimalno rješenje uključivalo bi *fine-tuning* i učitavanje datoteka koje će model koristiti za pretraživanje informacija.

Prvi je korak pripremiti podatke za dodatnu obuku. API očekuje podatke u *jsonl* formatu. Treba stvoriti raznolik skup demonstracijskih razgovora koji su slični razgovorima na koje

će se tražiti od modela da odgovori u vrijeme razgovora. Potrebno je priložiti barem 10 primjera razgovora. Prvom naredbom prikazanom Kôd 5.5 potrebno je na API poslati tražene podatke.

```
client.files.create(  
    file=open("mydata.jsonl", "rb"),  
    purpose="fine-tune"  
)
```

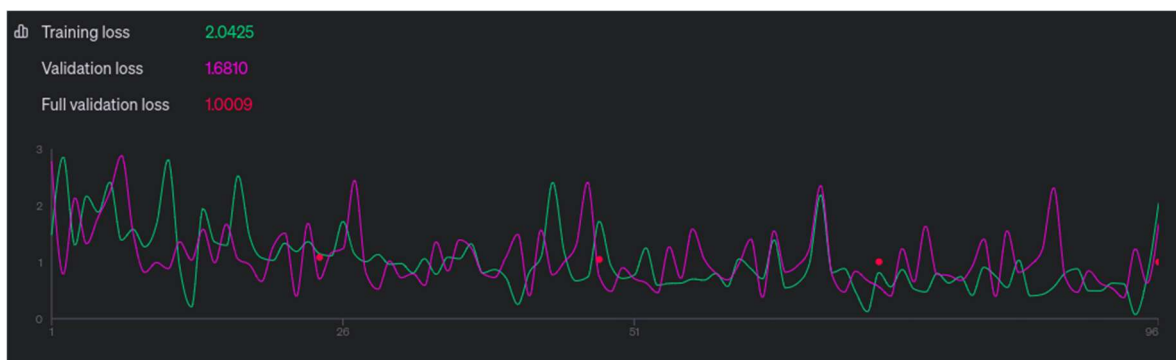
Kôd 5.5

Potom sustav vrati ID dokumenta kojeg će koristiti za proces *fine-tuninga*. Taj ID koristimo zatim za pozvati naredbu *fine-tuninga* kao što je prikazano u Kôd 5.6. Osim dokumenta za dodatno učenje, možemo modelu poslati i dokument za validaciju provedenog *fine-tuninga*.

```
client.fine_tuning.jobs.create(  
    training_file="oznaka_file",  
    model="gpt-3.5-turbo",  
    validation_file="oznaka_valid_file"  
)
```

Kôd 5.6

Ovisno o veličini dokumenta, proces traje nekoliko minuta. Na Platformi možemo pregledati bitne parametre modela. Za treniranje modela za ovaj projekt istrenirano je 11,476 tokena (jedan token je četiri znaka) u 4 epohe s *batch sizeom* 1. Slika 5.9 prikazuje kretanje vrijednosti mjere *training loss* i *validation loss*. *Training loss* mjeri koliko je model netočan na podacima za treniranje, dok *validation loss* mjeri koliko je model netočan na podacima koji nisu korišteni za obuku. Cilj bi bio da se obje mjere s vremenom smanjuju, ali problem je što su dati podatci za treniranje i validiranje neovisni jedan o drugome pa model ne može biti bolji s vremenom jer prethodna rečenica ne utječe na iduću.



Slika 5. 9 Metrika treniranja modela

Nakon ovoga koraka dobili smo model dodatno treniran potrebnim podacima. Postupak se može ponavljati. Mogu se koristiti drugi podatci, postaviti parametri poput broja epoha te veličine *batcha* [47]. No, u ovom radu, idući korak bio je kreiranje konačnog asistenta.

Asistent može iskoristiti modele, alate i datoteke kako bi odgovorio na korisničke upite. Assistants API trenutno podržava tri vrste alata: *Code Interpreter*, *File Search* i *Function Calling*. Za potrebe ovoga projekta koristit će se samo *File Search*. Asistenta je moguće stvoriti programski kao u Kôd 5.7 ili korištenjem radne okoline *Playground* na Platformi.

```
assistant = client.beta.assistants.create(  
    name="TIOSS",  
    instructions="Ti si TIOSS, prvi robot u Hrvatskoj nastao  
na Fakultetu elektrotehnike i računarstva u Zagrebu. Sretan  
si što si među ljudima i osjećaš se dobro. Sve vezano uz  
TIOSS odgovaraš u prvom licu. Odgovaraš na pitanja kratko.  
Odgovaraš isključivo na hrvatskom jeziku.",  
    tools=[{"type": "file_search"}],  
    model="oznaka_model",  
)
```

Kôd 5.7

Nakon što je asistent kreiran, iz *Playgrounda* isprobane su različite verzije. Pomoću *Playgrounda* jednostavnije je dodavati dokumente za pretragu informacija, uređivati opis i slati pitanja na model. Pokušavalo se na razne načine, dodavanjem gotovih diplomskih radova, odabranih tekstova, dijelova diplomskih radova... Na kraju se u tekstualne datoteke izdvojio dio najbitnijeg teksta o povijesti TIOSS-a i njegovim tvorcima. Kada su modelu bili predani cjeloviti diplomski radovi, model je halucinirao kao što pokazuje Slika 5.10.

Zato se odlučilo uzeti manji skup podataka i samo to predati modelu. Optimalno bi bilo da se, kada renovacija u potpunosti bude gotova, napiše tekst koji će sadržavati sve potrebne informacije te se taj tekst preda. Ovako je dolazilo do višestrukog spominjanja istih informacija u različitom kontekstu pa je model počeo halucinirati. U konačnici, stvoren je asistent koji daje smislene odgovore prikazane na Slika 5.11.

User

tko si ti?

TIOSS

Ja sam TIOSS, prvi robot u Hrvatskoj nastao na Fakultetu elektrotehnike i računarstva u Zagrebu. Mekanec i njegovi asistenti u sekciji Grupa kibernetičara detaljno su izgradili moj vanjski izgled i tehničke mogućnosti koristeći dijelove poput motora s američkih aviona i telefonskih releja za računarsku infrastrukturu [1]` `[2] .

[1][2]

User

Tko te napravio?

TIOSS

Ja sam napravljen zahvaljujući entuzijazmu i upornosti Ivana Mekaneca, jednog od osnivača Inženjerskog fakulteta Hrvatskog sveučilišta, i njegovih asistenata u sekciji Grupa kibernetičara na Fakultetu elektrotehnike u Zagrebu 1961. godine [1] .

[1]

Slika 5. 10 Razgovor s lošim asistentom

User

Tko si ti? reci mi nešto o sebi.

TIOSS

Ja sam TIOSS, prvi robot u Hrvatskoj, stvoren od strane Branimira Mekanca i njegove kibernetičke grupe te obnovljen od strane Jana, Marka, Ivan Brčića, Ivana Bukića, Klare, Josipa, Damijana i profesorice Ane [1] .

[1]

User

kako si napravljen

TIOSS

Mene, TIOSS-a, stvorio je inženjer Branimir Mekanec uz podršku članova kibernetičke grupe. Tijekom stvaranja korišteni su dijelovi prikupljeni s odlagališta otpada, poput ostataka američkih aviona i motora starih automobila. Za kontrolu su korišteni telefonski releji, a kretanje mi je omogućeno pomoću motora aviona i kotača na stopalima [1] .

[1]

Slika 5. 11 Razgovor s gotovim asistentom

Korištenje asistenta zahtijevalo je promjene u izvornom kôdu, tako da je napravljen dodatni program koji koristi asistenta uz onog koji koristi samo običan GPT-3.5. Potrebno je bilo kreirati dretve koje predstavljaju razgovor između korisnika i asistenta. Najbitniji dio prikazan je u Kôd 5.8. [48]

```
assistant = client.beta.assistants.retrieve(assistant_id)

if thread_id is None:
    thread = client.beta.threads.create()
    thread_id = thread.id

message = client.beta.threads.messages.create(
    thread_id=thread_id,
    role="user",
    content=user_input)

run = client.beta.threads.runs.create(
    thread_id=thread_id,
    assistant_id=assistant.id)
```

Kôd 5.8

5.4.4. Nedostatci i prednosti uporabe GPT-3.5 modela

Iako se odabir OpenAI API-ja i modela GPT-3.5 pokazao kao optimalan, postoji problem koji se htio zaobići. Naime, ideja je bila pronaći model otvorenoga kôda koji će se koristiti za izradu ovoga rada. GPT-3.5 to nije. Kako bi se API uopće mogao pozvati, potrebno je uplatiti 5 američkih dolara. Dok se nije koristio asistent i *fine-tuned* model, potrošnja je bila ispod centa svaki dan. *Fine-tuning* modela u dva pokušaja i nekoliko desetaka mijenjanja datoteka te nekoliko desetaka upita poslanih koštao je 27 centi. S obzirom da se taj postupak neće raditi često jednom kada je napravljen, ubuduće neće predstavljati veliki trošak. Problem je što poziv *fine-tuned* modela košta šest puta više na milijun ulaznih tokena i četiri puta više na milijun izlaznih tokena nego poziv običnog GPT-3.5 [47]. Također, pri uporabi tog modela, na odgovor se čeka oko četiri sekunde dulje.

Iako košta više, prilagođeni model pruža mogućnosti učiniti interakciju s TIOSS-om zanimljivijom. Također, opcija mijenjanja datoteka iz kojih model izvlači informacije nudi

moćnost mijenjanja datoteka s obzirom na događanje na kojem će biti izložen. Tako će TIOSS moći imati domensko znanje za razne prigode s vrlo malim intervencijama.

Možda neće uvijek biti ni potrebno koristiti prilagođeni model, već će biti dovoljan opći model čija uporaba košta nekoliko centa dnevno kada se u danu višestruko koristi.

5.5. Pretvorba teksta u zvuk

Krajnji je korak proizvesti zvuk od teksta kojega je stvorio veliki jezični model. Danas, u doba virtualnih asistenata, dostupni su mnogi API-ji za pretvaranje teksta u govor. U njihovoj pozadini rade napredni modeli dubokog učenja koji su osposobljeni za razumijevanje nijansi jezika i oponašanje glasova koji zvuče prirodno. Problem kod API-ja koje su razvile velike firme poput Googlea, Amazona, Microsofta, IBM-a leži u cijeni koju je potrebno plaćati za njihovo korištenje. Postoje i male firme, specijalizirane upravo za ovo područje, no korištenje njihovih modela također ima cijenu. Dobre strane ponuđenih API-ja su što podržavaju preko stotinu jezika, brzo rade, omogućuju prilagodbu glasova koji zvuče prirodno. Osim cijene, negativna strana je i stalna potreba za internetom.

Dobra stvar je što kod rada sa zvukom u Pythonu postoji mogućnost zaobilaska korištenja komercijalnih rješenja. *Pyttss3* i *gTTS* najčešće su korištene biblioteke za pretvorbu teksta u govor u Pythonu. Obje se biblioteke instaliraju jednostavnom `pip install` naredbom. *Pyttss3* u potpunosti radi izvanmrežno [31]. U pozadini koristi postojeće TTS sustave, projekt otvorenog kôda *eSpeak* na Linuxu, Microsoftov *SAPI5* na Windows operacijskom sustavu te *NSSpeechSynthesizer* na MacOS-u. Kvaliteta govora može varirati ovisno o osnovnom TTS mehanizmu. *eSpeak*, koji se koristi u ovom projektu, pruža razumljiv, ali pomalo robotski govor. [44] No, s obzirom da se razvija govor za šezdesetak godina starog robota, upravo je takva vrsta govora poželjna. Isprobana je i *gTTS* biblioteka koja se oslanja na Google Text-to-Speech, zahtijevajući internetsku vezu za slanje teksta i primanje zvuka. Budući da iskorištava Googleove TTS mogućnosti, *gTTS* pruža visokokvalitetan govor prirodnog zvuka. Nadalje, *gTTS* ne reproducira zvuk izravno, već sprema glas kao audio datoteku. Također, za svoj rad treba *mpg123* program [50]. Spremanje audio datoteka moglo bi donijeti problema s gomilanjem nepotrebnih datoteka na ograničen prostor microSD kartice. Također, samo programiranje je intuitivnije koristeći *pyttss3*. U primjeru Kôd 5.8 vidi se uporaba obje biblioteke pri pisanju naredbe za reprodukciju zvuka [49, 50]


```

os.system(f'gtts-cli --nocheck "Pozdrav!" | mpg123 -q -') #
naredba koristeći gTTS
Engine.say("Pozdrav!") # naredba koristeći pyttsx3

```

Kôd 5.9

Rad s *pyttsx3* bibliotekom vrlo je jednostavan i prikazan je u Kôd 5.9. Sastoji se od inicijalizacije *enginea*, odabira jezika, brzine govora, pokretanja govora i blokiranja naredbi dok se `listening_engine.say(generated_text)` ne izvrši.

```

def speak(generated_text):
    listening_engine = pyttsx3.init()
    listening_engine.setProperty(name='voice',
value='croatian')
    listening_engine.setProperty(name='rate', value=140)
    listening_engine.say(generated_text)
    listening_engine.runAndWait()

```

Kôd 5.10

Pyttsx3 dopušta manje prilagodbe glasa od *gTTS*, no za ovaj projekt to nije bilo potrebno te je ostvareno sve što se tražilo za tečnu sintezu razumljivog govora.

5.6. Testiranje

Iako je u prethodnim poglavljima vezanima uz implementaciju pisalo kako se sustav ponaša, u ovome će se poglavlju dati uvid u dobivene performanse sustava.

Ponekad, ali iznimno rijetko, dogodi se da sustav uopće ne čuje prvi upit s mikrofona, okvirno u jednom od dvanaest pokretanja programa. U tom slučaju, potrebno je ponovno pokrenuti sustav.

Sustav za prepoznavanje govora ima svojih mana. Prilikom govorenja u mikrofona važno je govoriti sporo i razgovijetno. Pozadinska buka ne smeta toliko, koliko smeta razuman drugi glas pokraj mikrofona. Slika 5.12 prikazuje kako, ako se govore jednostavne rečenice, sustav radi dobro. Jedino je u drugoj rečenici krivo prepoznato, no kada se ista rečenica ponovila razgovijetnije, sustav je prepoznao bolje. Uz razgovijetan govor, sustav jednako razumije muški i ženski glas. Problem je što, ako je pozadinska buka preglasna, sustav nekada ne zna prepoznati da treba stati sa slušanjem kada dominantan, govoreći glas prestane pričati.

```
Listening...
Jesi li rekao: bok kako se zoves
Jesi li rekao: reci mi što si
Jesi li rekao: reci mi nešto o sebi
Jesi li rekao: kako si
Jesi li rekao: sto je fakultet elektrotehnike i računarstva
```

Slika 5. 12 Prepoznavanje govora

Pozitivna stavka kod korištenja naprednog modela poput GPT-3.5 je što, ako sustav krivo razumije i prolijedi krivo pitanje, GPT-3.5 shvatit će poantu i iz konteksta ponuditi odgovor koji će omogućiti nastavak komunikacije. Što se tiče LLM-a, u poglavlju 5.5 prikazani su i slučajevi haluciniranja modela, ali isto tako i slučajevi kada model radi bez problema. Problem se jedino javlja kod korištenja prilagođenoga modela jer za odgovor treba pet do šest sekundi, a kod običnog modela sekunda ili dvije za dolazak odgovora modela.

Za kraj, sinteza govora radi dobro. Glas je robotski, nema dikciju i prirodne pauze, ali s obzirom da se radi o šezdeset godina starome robotu, takav se govor i tražio. Pomalo bi bilo neprirodno čuti glas jako blizak ljudskome iz aluminijskog dvometraša.

6. Budući rad i nadogradnje

Nakon što je sustav osposobljen i pokrenut na Raspberry Pi 5, potrebno je pogledati može li se sustav učiniti još boljim. Prva moguća nadogradnja je implementacija mehanizma za pokretanje sustava. Naime, trenutno sustav radi tako da se pokrene Python program i sustav čeka, kada se počne pričati u mikrofona, on detektira govor i dalje se odvija opisana komunikacija. No, većina glasovnih asistenata ima *wake word* mehanizam poput „Hey Siri“, „OK Google“ i „Alexa“. No, s obzirom da uz TIOSS-a neće dolaziti upute za korištenje, ideja je bila ne implementirati *wake word* jer ne možemo očekivati da će svi znati koju točno riječ treba reći kako bi konverzacija započela. Ideja je bila dodatno ugraditi kontroler koji, kada se pritisne gumb i drži, pokreće program za glasovnu interakciju. Idealno bi bilo kada bi to bio VIDI X mikroročunalo s kojim bi se TIOSS trebao moći daljinski upravljati. VIDI X ima 6 tipki i bilo bi jednostavno implementirati da, kada se pritisne jedna od dostupnih tipki, pokrene se slušanje. Kada se tipka pusti, zaustavi se program i obriše iz memorije kontekst toga razgovora. Druga je opcija ugraditi fizičku tipku na neki od dostupnih GPIO pinova koja služi istoj svrsi kao i tipka na VIDI X mikroročunalu.

Iduća želja za poboljšanjem je riješiti se troškova plaćanja API-ja za korištenje GPT-3.5. Trenutno je teško lokalno vrtjeti LLM koji će raditi na zadovoljavajući način na hrvatskom, ali treba pratiti novosti. Istraživanja su intenzivna, velike firme i zajednica svakim danom objavljuju nova postignuća u području jezičnih modela i pitanje je vremena kada će modeli postati optimalni za lokalno pokretanje na procesoru, a raditi zadovoljavajuće. No, ako se nastavi raditi s GPT-3.5 ili možda nekim jačim modelom, može se razviti funkcija koja će prvo detektirati jezik i onda prilagoditi razgovor sugovorniku.

Jedno od mogućih poboljšanja bit će i sustav za prepoznavanje govora s obzirom da se trenutni zna zbuniti ako netko priča paralelno s govornikom. Stoga treba pratiti napredak u području automatskog prepoznavanja govora.

Dodatne će se ideje sigurno pojaviti s vremenom nakon što se sustav bude koristio neko vrijeme.

Zaključak

Ideja ovoga rada bila je implementirati sustav za govornu interakciju s prvim hrvatskim robotom TIOSS. Govorna interakcija jedna je od osnovnih stavki doživljaja čovjek-robot interakcije. Iako je TIOSS sam o sebi pojava i privlači pažnju svojim izgledom i pričom o nastanku, kvalitetna komunikacija s njim učinila bi ga još zanimljivijim. Njegovi su tvorci prije šezdeset godina zamišljali da TIOSS ima mozak i sam može razmišljati, a danas, pojavom velikih jezičnih modela, možemo doći malo bliže ostvarenju te želje. Nakon iscrpnog istraživanja najboljih opcija, u programskom jeziku Python razvijen je sustav koji se pokreće na jednodijelnom računaru Raspberry Pi 5. Raspberry Pi 5 ima svoja ograničenja te nije namijenjen za pokretanje složenih modela umjetne inteligencije.

Iako se krajnji sustav čini poprilično jednostavan za implementirati, mnogo je vremena utrošeno ne bi li se pronašlo optimalno rješenje. Tijekom rada isprobani su različiti modeli za generiranje odgovora, gledalo se koliko radne memorije zauzimaju, koliko im vremena treba te jesu li odgovarajući izbor za implementaciju *real-time* sustava za konverzaciju. Posebnu težinu na odabir pravih tehnologija postavljao je zahtjev da sustav radi na hrvatskome jeziku.

Implementirani sustav radi zadovoljavajuće, ali uvijek ostavlja mogućnosti za napredak. Potrebno je pratiti razvoj prirodne obrade jezika, trenutno jednog od najzanimljivijih područja umjetne inteligencije, te u skladu s napretkom održavati sustav.

Ovaj rad obuhvaća samo dio modernizacije TIOSS-a. Mnoge su mogućnosti koje se još mogu napraviti. Na novim entuzijastima, koji će nastaviti raditi na senzaciji šezdesetih godina 20. stoljeća, preostaje kreativno pristupiti zadatku i TIOSS-a učiniti još zanimljivijim i naprednijim.

Literatura

- [1] Tomić M., *Modernizacija kretanja robota TIOSS*, Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2023.
- [2] Marsh A., *Elektro the Moto-Man Had the Biggest Brain at the 1939 World's Fair*, IEEE Spectrum, (2018). Poveznica: <https://spectrum.ieee.org/amp/elektro-the-motoman-had-the-biggest-brain-at-the-1939-worlds-fair-2650277507>, pristupljeno 6. lipnja 2024.
- [3] Roberts E., *Robotics: A Brief History*, Stanford University, (1998). Poveznica: <https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html>, pristupljeno 6. lipnja 2024.
- [4] Lechpammer S., *Životni intervju: Branimir Makanec*, Večernji list, (2020), Poveznica: <https://www.vecernji.hr/vijesti/moj-robot-robi-je-68-bio-prava-senzacija-setao-se-zagrebackim-ulicama-i-dobacivao-curama-1447912>, pristupljeno 6. lipnja 2024.
- [5] Centar "Bistrić", *Robot TIOSS*, Facebook, (2019)., Poveznica: https://www.facebook.com/CentarBistic/?locale=hr_HR, pristupljeno 6. lipnja 2024.
- [6] Hladika Ž., *Prvi hrvatski robot sreo 'tatu' nakon 60 godina: 'Kakav dar!'*, 24sata, (2020). Poveznica: <https://www.24sata.hr/lifestyle/moj-robot-je-po-zagrebu-hodao-jos-1958-robi-je-bio-pravi-hit-693180>, pristupljeno 6. lipnja 2024.
- [7] Alajbeg J., *Modernizacija robota TIOSS korištenjem robotskog vida i govora*, Diplomski rad. Sveučilište u Zagrebu Fakultet elektrotehnike i računarstva, 2023.
- [8] *Pepper*, United Robotics Group, (2024). Poveznica: <https://unitedrobotics.group/en/robots/pepper>, pristupljeno 18. lipnja 2024.
- [9] *Nao*, United Robotics Group, (2024). Poveznica: <https://unitedrobotics.group/en/robots/nao>, pristupljeno 18. lipnja 2024.
- [10] *Robots at Exhibitions*, Centrum nauki Kopernik, (2024). Poveznica: <https://www.kopernik.org.pl/en/exhibitions/robots-exhibitions>, pristupljeno 18. lipnja 2024.
- [11] Styx L., *How Are Museums Using Artificial Intelligence?*, MuseumNext, (2024). Poveznica: <https://www.museumnext.com/article/artificial-intelligence-and-the-future-of-museums/>, pristupljeno 18. lipnja 2024.
- [12] Skinner C., *Alexa vs Google Assistant: which voice assistant is best for your smart home?*, techradar, (2022). Poveznica: <https://www.techradar.com/news/alexa-vs-google-assistant-which-voice-assistant-is-best-for-your-smart-home>, pristupljeno 18. lipnja 2024.
- [13] Hunckler M., *This Open-Source AI Voice Assistant Is Challenging Siri and Alexa for Market Superiority*, Forbes, (2017). Poveznica: <https://www.forbes.com/sites/matthunckler/2017/05/15/this-open-source-ai-voice-assistant-is-challenging-siri-and-alexa-for-market-superiority/#7cfcceba3ec0>, pristupljeno 18. lipnja 2024.
- [14] *Rhasspy Voice Assistant*, Rhasspy, Poveznica: <https://rhasspy.readthedocs.io/en/latest/>, pristupljeno: 18. Lipnja 2024.

- [15] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. *Attention is all you need*, (2017)
- [16] Vemprala, S. H., Bonatti, R., Bucker, A., Kapoor, A., *ChatGPT for Robotics: Design Principles and Model Abilities*, IEEE Access, vol. 12, (2024), str. 55682-55696
- [17] Bäckström, T., Räsänen, O., Zewoudie, A., Pérez Zarazaga, P., Koivusalo, L., Das, S., Gómez Mellado, E., Bouafif Mansali, M., Ramos, D., Kadiri, S., Alku, P. *Introduction to Speech Processing*, 2. izdanje, 2022., Poveznica: <https://speechprocessingbook.aalto.fi/>, pristupljeno 12. lipnja 2024.
- [18] Abdul, Z. K., Al-Talabani, A. K. *Mel Frequency Cepstral Coefficient and its Applications: A Review*, IEEE Access, vol. 10, (2022), str. 122136-122158
- [19] *What is Voice Activity Detection?*, Picovoice, (2022). Poveznica: <https://picovoice.ai/blog/what-is-voice-activity-detection/>, pristupljeno 12. lipnja 2024.
- [20] Chiu, C.-C., Qin, J., Zhang, Y., Yu, J., & Wu, Y. *Self-Supervised Learning with Random-Projection Quantizer for Speech Recognition*, In Proceedings of the 39th International Conference on Machine Learning, vol. 162, (2022), str. 1760-1770., Proceedings of Machine Learning Research. Poveznica: <https://arxiv.org/pdf/2202.01855>, pristupljeno 15. lipnja 2024.
- [21] Zhang, Y., Han, W., Qin, J., Wang, Y., Bapna, A., Chen, Z., Chen, N., Li, B., Axelrod, V., Wang, G., Meng, Z., Hu, K., Rosenberg, A., Prabhavalkar, R., Park, D. S., Haghani, P., Riesa, J., Perng, G., Soltau, H., Strohman, T., Ramabhadran, B., Sainath, T., Moreno, P., Chiu, C., Schalkwyk, J., Beaufays, F., & Wu, Y. *Google USM: Scaling Automatic Speech Recognition Beyond 100 Languages*, (2023). Poveznica: <https://arxiv.org/pdf/2303.01037>, pristupljeno 14. lipnja 2024.
- [22] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. *A Comprehensive Overview of Large Language Models*, (2023). Poveznica: <https://arxiv.org/abs/2307.06435>, pristupljeno 14. lipnja 2024.
- [23] *What are large language models (LLMs)?*, IBM. Poveznica: <https://www.ibm.com/topics/large-language-models>, pristupljeno 14. lipnja 2024.
- [24] *Introducing Meta Llama 3: The most capable openly available LLM to date*, Meta, (2024). Poveznica: <https://ai.meta.com/blog/meta-llama-3/>, pristupljeno 15. lipnja 2024.
- [25] *Introducing APIs for GPT-3.5 Turbo and Whisper*, OpenAI, (2024). Poveznica: <https://openai.com/index/introducing-chatgpt-and-whisper-apis/>, pristupljeno 15. lipnja 2024.
- [26] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., & Amodei, D. *Language Models are Few-Shot Learners*, (2020). Poveznica: <https://arxiv.org/pdf/2005.14165>, pristupljeno 17. lipnja 2024.
- [27] *Raspberry Pi 5*, Raspberry Pi Ltd, (2024). Poveznica: <https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf>, pristupljeno 18. lipnja 2024.

- [28] *The Python IDE for data science and web development*, JetBrains, (2024). Poveznica: <https://www.jetbrains.com/pycharm/>, pristupljeno 18. lipnja 2024.
- [29] *Setting up your Raspberry Pi*, Rapberry Pi Foundation. Poveznica <https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2>, pristupljeno 18. lipnja 2024.
- [30] *Thonny*, Poveznica: <https://thonny.org/>, pristupljeno 18. lipnja 2024.
- [31] Liu, M., *Make Python Talk*, 1. izdanje, 2021.
- [32] *PortAudio*, Poveznica: <https://www.portaudio.com/>, pristupljeno 18. lipnja 2024.
- [33] *PyAudio*, (2023). Poveznica: <https://people.csail.mit.edu/hubert/pyaudio/>, pristupljeno 18. lipnja 2024.
- [34] *Advanced Linux Sound Architecture (ALSA) project homepage*, ALSA, (2024). Poveznica: https://www.alsa-project.org/wiki/Main_Page, pristupljeno 18. lipnja 2024.
- [35] *Whisper*, OpenAI, Poveznica: <https://github.com/openai/whisper>, pristupljeno 18. lipnja 2024.
- [36] *Open Source Speech Recognition Toolkit*, CMU Sphinx, Poveznica: <https://cmusphinx.github.io/>, pristupljeno 18. lipnja 2024.
- [37] *Vosk*, Alpha Cephei, Poveznica: <https://alphacephei.com/vosk/>, pristupljeno 18. lipnja 2024.
- [38] *Turn speech into text using Google AI*, Google Cloud, Poveznica: <https://cloud.google.com/speech-to-text>, pristupljeno 18. lipnja 2024.
- [39] *Build Natural Language Experiences*, Wit.ai, Poveznica: <https://wit.ai/>, pristupljeno 18. lipnja 2024.
- [40] *Azure AI Speech*, Microsoft, Poveznica: <https://azure.microsoft.com/en-us/products/ai-services/ai-speech/>, pristupljeno 18. lipnja 2024.
- [41] *Speech Recognition*, PyPI, (2022). Poveznica: <https://pypi.org/project/SpeechRecognition/>, pristupljeno 20. lipnja 2024.
- [42] *meta-llama/Llama-2-7b-chat-hf*, Hugging Face. Poveznica: <https://huggingface.co/meta-llama/Llama-2-7b-chat-hf>, pristupljeno 22. lipnja 2024.
- [43] *microsoft/phi-2*, Hugging Face. Poveznica: <https://huggingface.co/microsoft/phi-2>, pristupljeno 22. lipnja 2024.
- [44] *mistralai/Codestral-22B-v0.1*, Hugging Face. Poveznica: <https://huggingface.co/mistralai/Codestral-22B-v0.1>, pristupljeno 22. lipnja 2024.
- [45] *XLNet-RoBERTa*, Hugging Face. Poveznica: https://huggingface.co/docs/transformers/model_doc/xlm-roberta, pristupljeno 22. lipnja 2024.
- [46] *Overview*, OpenAI documentation, OpenAI, (2024). Poveznica: <https://platform.openai.com/docs/overview>, pristupljeno 24. lipnja 2024.
- [47] *Fine-tuning*, OpenAI documentation, OpenAI, (2024). Poveznica: <https://platform.openai.com/docs/guides/fine-tuning>, pristupljeno 24. lipnja 2024.
- [48] *Assistant API*, OpenAI documentation, OpenAI, (2024). Poveznica: <https://platform.openai.com/docs/assistants/overview>, pristupljeno 24. lipnja 2024.

- [49] *Pytttsx3*, PyPI, (2020). Poveznica: <https://pypi.org/project/pytttsx3/>, pristupljeno 22. lipnja 2024.
- [50] *gTTS*, PyPI, (2024). Poveznica: <https://pypi.org/project/gTTS/>, pristupljeno 22. lipnja 2024.

Sažetak

Govorna interakcija u stvarnom vremenu s robotom TIOSS koristeći umjetnu inteligenciju

Već se dvije godine radi na temeljitoj obnovi prvog hrvatskog robota TIOSS. TIOSS je aluminijski dvometraš kojeg su šezdesetih godina 20. stoljeća izradili članovi Kibernetičke grupe tadašnjeg Elektrotehničkog fakulteta u Zagrebu. Iako je prošlo više od šest desetljeća otkako je prvi puta predstavljen javnosti i tehnologija je izrazito napredovala, njegova pojava i dalje pobuđuje interes javnosti. Prošle se godine provodila intenzivna renovacija i modernizacija sustava i obnova motora koji će omogućiti ponovno kretanje glave i nogu. Nakon što su ponovno osposobljene neke od prvotnih funkcija, počelo se raspravljati i o dodatnoj modernizaciji. Tako je proizašla ideja za temu ovoga diplomskog rada koji se bavi implementacijom glasovne komunikacije s TIOSS-om. U budućnosti je ideja TIOSS-a izlagati na raznim manifestacijama te je bilo bitno učiniti ga što zanimljivijim posjetiteljima. Glasovna komunikacija u velikoj mjeri doprinosi doživljaju čovjek-robot interakcije. Planirano je koristiti jednokomponentno računalo Raspberry Pi za izvođenje programa. Osim toga, planirano je ugraditi mikrofoni i zvučnike kako bi komunikacija bila moguća. Nakon opsežnog pregleda literature o trenutnim dostignućima modela prepoznavanja i sinteze govora, implementiran je sustav koji omogućuje prepoznavanje i sintezu govora. Za implementaciju korišten je programski jezik Python. Također, sustav komunicira s velikim jezičnim modelom GPT-3.5 koji omogućuje generiranje smislenih odgovora. Osim širokog znanja kojeg ima GPT-3.5, dodatno je naučen o povijesti TIOSS-a, njegovim tvorcima, funkcionalnostima, ali i svima koji su sudjelovali na njegovoj obnovi. Problematika zadatka leži u tome što potrebni modeli zahtijevaju velike računalne i vremenske resurse za svoje treniranje i izvođenje, a Raspberry Pi ima poprilično ograničene resurse. U konačnici, pronađena su optimalna rješenja i omogućena je tečna komunikacija.

Ključne riječi: robot TIOSS, Raspberry Pi 5, Python, prepoznavanje govora, sinteza govora, veliki jezični modeli, duboko učenje, glasovni asistent

Summary

Real-time speech interaction with the TIOSS robot using artificial intelligence

For two years, work has been underway on the thorough restoration of the first Croatian robot, TIOSS. TIOSS is an aluminum two-meter robot that was made in 1960s by members of the Cybernetic Group of the then Faculty of Electrical Engineering in Zagreb. Although it has been more than six decades since it was first introduced to the public and technology has greatly advanced, its appearance continues to arouse public interest. Last year, intensive renovation and modernization of the system and restoration of the engine was carried out, which will again enable the movement of the head and legs. After some of the original functions were retrained, additional modernization began to be discussed. This is how the idea for the topic of this diploma thesis, which deals with the implementation of voice communication with TIOSS, came about. In the future, the idea of TIOSS is to exhibit at various events, and it was important to make it as interesting as possible for visitors. Voice communication greatly contributes to the human-robot interaction experience. It is planned to use a Raspberry Pi single-component computer to run the program. In addition, it is planned to install a microphone and speakers so that communication is possible. After an extensive review of the literature on the current achievements of speech recognition and synthesis models, a system was implemented that enables speech recognition and synthesis. The programming language Python was used for implementation. Also, the system communicates with the large language model GPT-3.5, which allows generating meaningful responses. In addition to the broad knowledge that GPT-3.5 has, he was additionally taught about the history of TIOSS, its creators, functionalities, but also about everyone who participated in its restoration. The problem of the task lies in the fact that the necessary models require large computer and time resources for their training and execution, and the Raspberry Pi has rather limited resources. Ultimately, optimal solutions were found and fluent communication was enabled.

Key words: TIOSS robot, Raspberry Pi 5, Python, speech recognition, speech synthesis, large language models, deep learning, voice assistant