A logo of a university

Description automatically generated

A close-up of a computer code

Description automatically generated

UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

PROIECT 3

**TEMA: AI voice assistant**

Profesor coordonator:

Dumitrescu Anamaria Studenții: Uța Petre Călin

Dănescu Andrei-Ionuț

Petre Bianca

Grupa: 444C

**Cuprins**

**Capitolul 1: Introducere .........................................................................3**

**Capitolul 2: Obiectivele proiectului........................................................3**

**Capitolul 3: Metodologie..........................................................................4**

**Capitolul 4: Descrierea tehnică...............................................................8**

**Capitolul 5: Testare și rezultate..............................................................9**

**Capitolul 6: Concluzii.............................................................................10**

**Capitolul 7: Bibliografie.........................................................................11**

**Capitolul 8: Anexe...................................................................................12**

1. **Introducere**

Asistenții vocali reprezintă o tehnologie în continuă dezvoltare, redefinind interacțiunile dintre utilizatori și dispozitive.

Proiectul nostru propune dezvoltarea unui asistent vocal inteligent implementat pe platforma Raspberry Pi 4 și utilizarea API-ului Gemini Generative Language pentru procesarea și generarea răspunsurilor.

Soluția este destinată aplicării în diverse domenii, precum case inteligente, educație și asistență personală.

**2. Obiectivele Proiectului**

Dezvoltarea unui sistem capabil să interpreteze comenzi vocale și să ofere răspunsuri precise.

Integrarea unui modul hardware (Raspberry Pi 4) pentru controlul dispozitivelor externe.

Asigurarea unei experiențe fluide pentru utilizatori prin intermediul LLM și TTS.

**3. Metodologie**

**3.1 Cercetare Preliminară**

Am analizat soluțiile existente pe piață, inclusiv Google Assistant și Alexa.

Raspberry Pi 4 l-am ales datorită capacitații sale de procesare, iar Gemini AI l-am preferat pentru precizia sa în procesarea limbajului natural.

**3.2 Alegerea Platformei și a Tehnologiilor Software**

**Hardware:**

1. Raspberry Pi 4

**Raspberry Pi 4 Model B** este un microcomputer de dimensiuni reduse, extrem de versatil, creat de Raspberry Pi Foundation. Acesta este o actualizare semnificativă față de modelele anterioare, oferind performanțe îmbunătățite și o gamă mai largă de funcționalități, ceea ce îl face ideal pentru proiecte educaționale, hobby-uri, IoT, servere sau chiar utilizări ca un mini PC.

**Specificații tehnice principale:**

1. **Procesor**:
   * Broadcom BCM2711, un CPU quad-core ARM Cortex-A72 la 1,5 GHz.
   * Performanță ridicată, potrivită pentru aplicații mai complexe.
2. **Memorie RAM**:
   * Disponibil în variante de 2GB, 4GB, sau 8GB de RAM LPDDR4-3200.
   * Oferă flexibilitate pentru proiecte de la simple la avansate.
3. **Stocare**:
   * Utilizează carduri microSD pentru stocarea sistemului de operare și a datelor.
   * Port USB 3.0 permite conectarea unităților externe de stocare.
4. **Conectivitate**:
   * 2 porturi USB 3.0 și 2 porturi USB 2.0.
   * Ethernet Gigabit pentru rețele rapide.
   * Wi-Fi dual-band (2.4GHz și 5GHz) și Bluetooth 5.0.
5. **Ieșiri video și audio**:
   * Două porturi micro-HDMI, fiecare suportând rezoluții de până la 4K la 60 fps.
   * Suport pentru H.265 și decodare video 4K.
   * Jack de 3,5 mm pentru audio și video compozit.
6. **GPIO**:
   * 40 de pini GPIO, compatibili cu majoritatea componentelor și accesoriilor Raspberry Pi.
7. **Alte caracteristici**:
   * Alimentare prin port USB-C, necesită o sursă de alimentare de minimum 5V/3A.
   * Dimensiuni reduse: 85,6 mm × 56,5 mm.

**Utilizări comune:**

* **Educație**: Ideal pentru învățarea programării și a bazelor electronicii.
* **Servere**: Poate fi utilizat ca server de fișiere, media server (cu software precum Plex) sau server web.
* **Automatizare IoT**: Folosit pentru proiecte de casă inteligentă sau monitorizare.
* **Mini PC**: Suportă sisteme de operare precum Raspberry Pi OS, Ubuntu sau alte distribuții Linux.
* **Gaming retro**: Compatibil cu RetroPie pentru emularea consolelor clasice.

Raspberry Pi 4 este un dispozitiv mic, dar puternic, cu un preț accesibil, fiind preferat atât de entuziaști, cât și de profesioniști datorită versatilității și comunității extinse care sprijină proiectele bazate pe acesta.

1. Boxă Bluetooth cu microfon: Sony SRS-XE300
2. Display cu touch TFT de 3.5 inch

**Software:**

* 1. Gemini API

**Google Gemini** este un model avansat de inteligență artificială dezvoltat de Google DeepMind, conceput pentru a îmbunătăți semnificativ interacțiunea utilizatorilor cu dispozitivele și serviciile Google. Lansat inițial în decembrie 2023, Gemini a evoluat rapid, integrându-se în diverse platforme și dispozitive pentru a oferi funcționalități AI avansate.

**Caracteristici principale:**

* **Model multimodal**: Gemini este proiectat să proceseze simultan diferite tipuri de date, inclusiv text, imagini, audio și video, permițând o înțelegere și generare complexă a conținutului.
* **Dimensiuni variate**: Disponibil în trei versiuni – Ultra, Pro și Nano – fiecare optimizată pentru diferite niveluri de complexitate și utilizare, de la sarcini complexe la aplicații pe dispozitive mobile.
* **Integrare extinsă**: Gemini este integrat în produse precum Bard, Google Workspace (Gmail, Docs, Sheets), Google TV și dispozitive Pixel, oferind asistență AI direct în aceste platforme.
* **Capacități avansate**: Include funcționalități precum generarea de imagini și audio, utilizarea instrumentelor integrate (de exemplu, căutare Google), și agenți AI capabili să execute sarcini complexe sub supravegherea utilizatorului.

**Evoluție și actualizări:**

* **Gemini 1.0**: Lansat în decembrie 2023, a introdus primele modele din seria Gemini, optimizate pentru diferite dimensiuni și capabilități.
* **Gemini 1.5**: Lansat în februarie 2024, a adus îmbunătățiri semnificative în performanță și capacități, inclusiv o fereastră de context extinsă și o nouă arhitectură.
* **Gemini 2.0**: Anunțat în decembrie 2024, introduce capabilități avansate pentru agenți AI, inclusiv utilizarea memoriei, raționamentului și planificării pentru a îndeplini sarcini complexe sub supravegherea utilizatorului.

**Aplicații practice:**

* **Asistență personală**: Gemini poate ajuta utilizatorii în scriere, planificare, învățare și alte activități, oferind suport AI direct pe dispozitivele lor.
* **Control vocal al dispozitivelor**: Integrarea cu Google TV permite utilizatorilor să controleze televizoarele inteligente și alte dispozitive prin comenzi vocale avansate, fără a fi necesară utilizarea comenzilor tradiționale.
* **Productivitate în afaceri**: Integrat în Google Workspace, Gemini oferă instrumente AI care îmbunătățesc eficiența în sarcini precum redactarea documentelor, analiza datelor și gestionarea comunicării.

Prin aceste inovații, Google Gemini reprezintă un pas important în direcția dezvoltării asistenților AI universali, capabili să îmbunătățească interacțiunea utilizatorilor cu tehnologia într-un mod natural și eficient.

* 1. **1. PyAudio**

PyAudio este o librărie Python utilizată pentru a accesa și controla intrările și ieșirile audio, facilitând interacțiunea cu dispozitivele audio precum microfoane și difuzoare. Aceasta este un wrapper pentru biblioteca portaudio, ceea ce permite manipularea fluxurilor audio în timp real.

**Caracteristici principale:**

* Capturarea și redarea fluxurilor audio.
* Configurarea frecvenței de eșantionare, numărului de canale și formatului de date audio.
* Gestionarea simultană a mai multor fluxuri audio.
* Suport pentru diverse formate audio, cum ar fi PCM (Pulse Code Modulation).

**Utilizări comune:**

* Crearea aplicațiilor de recunoaștere vocală.
* Procesarea și analizarea sunetelor în timp real.
* Dezvoltarea de instrumente pentru înregistrare audio.

**2. google.generativeai**

google.generativeai este o librărie Python destinată interacțiunii cu modelele de inteligență artificială generativă dezvoltate de Google, precum cele din familia Gemini. Această librărie permite accesul la puterea procesării limbajului natural și generării de conținut bazat pe modele avansate.

**Caracteristici principale:**

* Generarea textului, răspunsuri la întrebări și completarea propozițiilor.
* Posibilitatea de a oferi contexte personalizate pentru interacțiuni mai relevante.
* Suport pentru API-ul Google generative AI pentru integrarea în diverse aplicații.

**Utilizări comune:**

* Dezvoltarea chatbot-urilor inteligente.
* Crearea de conținut automatizat, precum rezumate sau articole.
* Analiza semantică avansată a textelor.

**3. SpeechRecognition**

SpeechRecognition este o librărie Python populară pentru recunoașterea vorbirii. Aceasta oferă un API simplu și intuitiv pentru a converti vorbirea în text, utilizând diverse motoare de recunoaștere vocală, cum ar fi Google Speech API, Sphinx, sau alte servicii online.

**Caracteristici principale:**

* Suport pentru mai multe motoare de recunoaștere vocală.
* Procesarea fișierelor audio sau capturarea sunetului în timp real de la microfon.
* Configurarea limbajelor și recunoașterea multilingvistică.

**4. Descrierea Tehnică**

**4.1 Arhitectura Sistemului**

Sistemul este compus din următoarele module:

* Captarea audio: Microfonul captează comanda vocală.
* Procesare Speech-to-Text: librăria speechrecognition convertește comanda vocală în text.
* Procesare LLM: Gemini analizează textul și generează răspunsul adecvat.
* Răspuns vocal: Modulul TTS transformă textul generat în audio.

**4.2 Implementare Practică**

* Configurare hardware: Instalarea Raspberry Pi OS, conectarea prin bluetooth a boxei.
* Dezvoltare software: Script Python care utilizează API-ul Google Gemini.

**5. Testare și Rezultate**

Testele au fost realizate pentru răspunsul unor întrebări generale.

Tabel 1: Performanța asistentului în diferite scenarii

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenariu** | **Timp de răspuns** | **Precizie recunoaștere** | **Corectitudine răspuns** |
| Întrebări generale | 2 secunde | 95% | 92% |

**6. Concluzii**

Proiectul dezvoltării unui asistent vocal utilizând Raspberry Pi 4 și Gemini API demonstrează viabilitatea creării unui sistem inteligent, compact și economic, capabil să răspundă la comenzi vocale complexe. Raspberry Pi 4, cu capacitățile sale hardware îmbunătățite, oferă o platformă robustă pentru implementarea aplicațiilor de inteligență artificială, în timp ce Gemini API furnizează acces la modele lingvistice avansate, ceea ce facilitează procesarea limbajului natural.

Prin integrarea acestor tehnologii, s-a reușit realizarea unui dispozitiv care poate efectua diverse sarcini, precum gestionarea informațiilor, controlul dispozitivelor smart home și oferirea de răspunsuri contextuale. Acest proiect evidențiază potențialul soluțiilor open-source și modul în care resursele accesibile pot fi utilizate pentru a construi tehnologii competitive din punct de vedere funcțional și eficient din punct de vedere al costurilor.

În concluzie, proiectul reprezintă un pas semnificativ în democratizarea accesului la asistenții vocali și deschide calea pentru îmbunătățiri viitoare, precum optimizarea recunoașterii vocale, creșterea autonomiei dispozitivului sau adăugarea de noi funcționalități personalizate. Astfel, acest tip de soluție contribuie la extinderea utilizării IoT și AI în viața de zi cu zi, oferind o combinație perfectă între inovație tehnologică și aplicabilitate practică.

**6.1 Observații**

Proiectul a demonstrat funcționalitatea și eficiența unui asistent vocal bazat pe Raspberry Pi 4 și GeminiAI. Timpul de răspuns și precizia au fost foarte bune pentru aplicații practice.

**6.2 Direcții Viitoare**

* Integrarea unui model LLM personalizat pentru limba română.
* Dezvoltarea unei interfețe grafice pentru configurare.
* Optimizarea consumului energetic pe Raspberry Pi.

**7. Bibliografie**

[1] <https://www.raspberrypi.com/>

[2] <https://github.com/techmakerai/Hands-on-Tutorial-Voice-Assistant-on-Raspberry-Pi>

[3] <https://www.youtube.com/watch?v=uV6hJQcuW4w&ab_channel=TechMakerAI>

[4] <https://ai.google.dev/>

[5] https://pypi.org/project/PyAudio/

**7. Anexe**

Script Python complet

from datetime import date

from io import BytesIO

import threading

import queue

import time

import os

# turn off the welcome message from pygame package

os.environ['PYGAME\_HIDE\_SUPPORT\_PROMPT'] = '1'

os.environ["GRPC\_VERBOSITY"] = "ERROR"

os.environ["GLOG\_minloglevel"] = "2"

os.environ["GRPC\_DNS\_RESOLVER"] = "native"

os.environ["GRPC\_DEFAULT\_AUTHORITY"] = "ipv4-only"

import google.generativeai as genai

from gtts import gTTS

from pygame import mixer

import speech\_recognition as sr

import sounddevice

mixer.pre\_init(frequency=24000, buffer=2048)

mixer.init()

# add your Google Gemini API key here

my\_api\_key = ""

if len(my\_api\_key) < 5:

    print(f"Please add your Google Gemini API key in the program. \n " )

    quit()

# set Google Gemini API key as a system environment variable or add it here

genai.configure(api\_key= my\_api\_key)

# model of Google Gemini API

model = genai.GenerativeModel("gemini-1.5-flash")

# start the chat model

chat = model.start\_chat(history=[])

today = str(date.today())

# Initialize the counters

numtext = 0

numtts = 0

numaudio = 0

# thread 1 for text generation

def chatfun(request, text\_queue, llm\_done, stop\_event):

    global numtext, chat

    response = chat.send\_message(request, stream=True)

    shortstring = ''

    ctext = ''

    for chunk in response:

        try:

            if chunk.candidates[0].content.parts:

                ctext = chunk.candidates[0].content.parts[0].text

                ctext = ctext.replace("\*", "")

                if len(shortstring) > 10 or len(ctext) >10:

                    shortstring = "".join([shortstring, ctext])

                    text\_queue.put( shortstring )

                    print(shortstring, end='') #, flush=True)

                    shortstring = ''

                    ctext = ''

                    # time.sleep(0.2)

                    numtext += 1

                else:

                    shortstring = "".join([shortstring, ctext])

                    ctext = ''

        except Exception as e:

            continue

    if len(ctext) > 0:

        shortstring = "".join([shortstring, ctext])

    if len(shortstring) > 0:

        print(shortstring, end='')

        text\_queue.put(shortstring)

        numtext += 1

    if numtext > 0:

        append2log(f"AI: {response.candidates[0].content.parts[0].text } \n")

    else:

        llm\_done.set()

        stop\_event.set()

    llm\_done.set()  # Signal completion after the loop

# convert "text" to audio file and play back

def speak\_text(text):

    global slang, rled

    mp3file = BytesIO()

    tts = gTTS(text, lang = "en", tld = 'us')

    tts.write\_to\_fp(mp3file)

    mp3file.seek(0)

    print("AI: ", text)

    try:

        mixer.music.load(mp3file, "mp3")

        mixer.music.play()

        while mixer.music.get\_busy():

            time.sleep(0.2)

    except KeyboardInterrupt:

        mixer.music.stop()

        mp3file = None

        rled.off()

    mp3file = None

    rled.off()

# thread 2 for tts

def text2speech(text\_queue, tts\_done, llm\_done, audio\_queue, stop\_event):

    global numtext, numtts

    time.sleep(1.0)

    while not stop\_event.is\_set():  # Keep running until stop\_event is set

        if not text\_queue.empty():

            text = text\_queue.get(timeout = 1)  # Wait for 1 second for an item

            if len(text) > 0:

                # print(text)

                try:

                    mp3file1 = BytesIO()

                    tts = gTTS(text, lang = "en", tld = 'us')

                    tts.write\_to\_fp(mp3file1)

                except Exception as e:

                    continue

                audio\_queue.put(mp3file1)

                numtts += 1

                text\_queue.task\_done()

        #print("\n numtts, numtext : ", numtts , numtext)

        if llm\_done.is\_set() and numtts == numtext:

            #time.sleep(0.3)

            tts\_done.set()

            mp3file1 = None

            #print("\n break from the text queue" )

            break

# thread 3 for audio playback

def play\_audio(audio\_queue,tts\_done, stop\_event):

    global numtts, numaudio

    #print("start play\_audio()")

    while not stop\_event.is\_set():  # Keep running until stop\_event is set

        mp3audio1 = BytesIO()

        mp3audio1 = audio\_queue.get()

        mp3audio1.seek(0)

        mixer.music.load(mp3audio1, "mp3")

        mixer.music.play()

        #print("Numaudio: ", numaudio )

        while mixer.music.get\_busy():

            time.sleep(0.2)

        numaudio += 1

        audio\_queue.task\_done()

        #print("\n numtts, numaudio : ", numtts , numaudio)

        if tts\_done.is\_set() and numtts  == numaudio:

            mp3audio1 = None

            #print("\n no more audio/text data, breaking from audio thread")

            break  # Exit loop

# save conversation to a log file

def append2log(text):

    global today

    fname = 'chatlog-' + today + '.txt'

    with open(fname, "a", encoding='utf-8') as f:

        f.write(text + "\n")

        f.close

# define default language to work with the AI model

slang = "en-EN"

# Main function

def main():

    global today, slang, numtext, numtts, numaudio, messages

    rec = sr.Recognizer()

    mic = sr.Microphone()

    rec.dynamic\_energy\_threshold=False

    rec.energy\_threshold = 400

    sleeping = True

    # while loop for conversation

    while True:

        with mic as source:

            rec.adjust\_for\_ambient\_noise(source, duration= 0.5)

            try:

                print("Listening ...")

                audio = rec.listen(source, timeout = 10 ) #, phrase\_time\_limit = 30)

                text =  rec.recognize\_google(audio, language=slang) #   # rec.recognize\_wit(audio, key=wit\_api\_key ) #

                # print(text)

                if len(text)>0:

                    print(f"You: {text}\n " )

                else:

                    #print(f"Unable to recognize your speech. Program will exit. \n " )

                    continue

                # AI is in sleeping mode

                if sleeping == True:

                    # User can start the conversation with the wake word "Jack"a

                    if "jack" in text.lower():

                        request = text.lower().split("jack")[1]

                        # User said wake word, AI is awake now,

                        sleeping = False

                        # start a new conversation

                        chat = model.start\_chat(history=[])

                        append2log(f"\_"\*40)

                        today = str(date.today())

                        messages = []

                        # if the user's question is none or too short, skip

                        if len(request) < 2:

                            speak\_text("Hi, there, how can I help?")

                            #append2log(f"AI: Hi, there, how can I help? \n")

                            continue

                    # if user did not say the wake word, nothing will happen

                    else:

                        #print(f"Please start the conversation with the wake word. \n " )

                        continue

                # AI is awake

                else:

                    request = text.lower()

                    if "that's all" in request:

                        append2log(f"You: {request}\n")

                        speak\_text("Bye now")

                        append2log(f"AI: Bye now. \n")

                        sleeping = True

                        # AI goes back to speeling mode

                        continue

                    if "jack" in request:

                        request = request.split("jack")[1]

                if len(request) == 0:

                    #print(f"Unable to recognize your question. Program will exit. \n " )

                    continue

                # process user's request (question)

                append2log(f"You: {request}\n ")

                #print(f"AI: ", end='')

                # Initialize the counters before each reply from AI

                numtext = 0

                numtts = 0

                numaudio = 0

                # Define text and audio queues for data storage

                text\_queue = queue.Queue()

                audio\_queue = queue.Queue()

                # Define events

                llm\_done = threading.Event()

                tts\_done = threading.Event()

                stop\_event = threading.Event()

                # Thread 1 for handling the LLM responses

                llm\_thread = threading.Thread(target=chatfun, args=(request, text\_queue,llm\_done,stop\_event,))

                # Thread 2 for text-to-speech

                tts\_thread = threading.Thread(target=text2speech, args=(text\_queue,tts\_done,llm\_done, audio\_queue, stop\_event,))

                # Thread 3 for audio playback

                play\_thread = threading.Thread(target=play\_audio, args=(audio\_queue,tts\_done, stop\_event,))

                llm\_thread.start()

                tts\_thread.start()

                play\_thread.start()

                # wait for LLM to finish responding

                llm\_done.wait()

                llm\_thread.join()

                tts\_done.wait()

                audio\_queue.join()

                stop\_event.set()

                tts\_thread.join()

                play\_thread.join()

                print('\n')

            except Exception as e:

                continue

if \_\_name\_\_ == "\_\_main\_\_":

    main()