Model: GPT-4

Hi, I would like to have a python script that will listen for keyword (assistant name), like "Hey Siri", record and transcribe the question/task. Then it would connect to the chatGPT API and get the answer. It would ask the chat to pretend to be assistant (with easier mentioned name, that gives short concise answers, and avoids unnecessary talk) and if there is any action to be taken, then it should respond with python code or apple script to do so. After receiving the response the script will check if there is any python/apple script code and run it. It will also read out loud non-code response. It will keep the conversation context (connecting with api) for 1h.

There should be different assistant name for each language (Polish, English, and Spanish), so the transcription will work with all this languages.

Divide answer into steps and for now only show first step.

I'll provide you with the first step of creating the Python script. In this step, we will install required packages, set up the microphone to listen for the keywords, and record the audio when the keywords are detected.

1. Install the required Python packages:

   - `speech_recognition` for speech recognition and transcription
   - `pyaudio` for audio input and output
   - `googletrans` for language detection
   - `gTTS` for text-to-speech

   You can install them using pip:

```
Copy code

pip install SpeechRecognition pyaudio googletrans==4.0.0-rc1 gtts
```

2. Create a Python script and import the required modules:

```python
Copy code

import speech_recognition as sr
from googletrans import Translator
from gtts import gTTS
import pyaudio
import time
import os
import wave

# Set your assistant names
assistant_names = {
    "English": "Hey Assistant",
    "Polish": "Cześć Asystent",
    "Spanish": "Hola Asistente"
}

# Initialize the recognizer and translator
recognizer = sr.Recognizer()
```

```
translator = Translator()
```

3. Create a function to detect the assistant name in the audio input:

```python
def detect_assistant_name(audio_data):
    try:
        # Recognize the audio using Google Speech Recognition
        text = recognizer.recognize_google(audio_data, show_all=True)

        for language, name in assistant_names.items():
            # Translate the detected text to each language
            translated_text = translator.translate(text, dest=languag
```

Before proceeding to the next step, can you create python vm for that and recode detect_assistant_name function. I don't understand the translation part. It shouldn't translate the prompt, but send it in original language. We just need to detect language for speech recognition and synthesis.

I apologize for the confusion. Let's modify the `detect_assistant_name` function to detect the language and return the detected assistant name along with the language without translating the prompt.

1. Create a virtual environment and activate it:

```bash
python -m venv venv
source venv/bin/activate  # For Linux/MacOS
venv\Scripts\activate     # For Windows
```

2. Install the required packages as mentioned in the previous answer:

```bash
pip install SpeechRecognition pyaudio googletrans==4.0.0-rc1 gtts
```

3. Modify the `detect_assistant_name` function to detect the language and return the detected assistant name along with the language:

```python
def detect_assistant_name(audio_data):
    try:
        # Recognize the audio using Google Speech Recognition
        text = recognizer.recognize_google(audio_data, show_all=True)
```

```
        # Detect the language of the recognized text
        detected_language = translator.detect(text).lang

        # Check if the recognized text contains the assistant name in
        for language, name in assistant_names.items():
            if name.lower() in text.lower() and detected_language ==
                return name, detected_language

    except sr.UnknownValueError:
        pass
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognit

    return None, None
```

In this version, we use the **`detect`** function

What do we do next? Not everything at once, just one next step.

In the next step, we'll create a function to continuously listen for the assistant name and record the subsequent question or task.

4. Create a function to listen for the assistant name and record the question or task:

python Copy code

```
def listen_for_assistant():
    with sr.Microphone() as source:
        while True:
            print("Listening for assistant name...")
            recognizer.adjust_for_ambient_noise(source)
            audio_data = recognizer.listen(source)

            assistant_name, detected_language = detect_assistant_name
            if assistant_name:
                print(f"Assistant name detected: {assistant_name}")
                break

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        question_audio_data = recognizer.listen(source, timeout=5)

    return question_audio_data, detected_language
```

This function will listen for the assistant name in the audio input using a loop. If the assistant name is detected, it breaks the loop and records the subsequent question or task. The function then returns the recorded audio data and the detected language for further processing.

Thank you. What do we do next?

Thank you. What do we do next?

In the next step, we'll create a function to transcribe the recorded question or task and call the ChatGPT API to get the response.

5. Create a function to transcribe the recorded audio and call the ChatGPT API:

```python
import requests
import json

API_KEY = "your_chatgpt_api_key"

def transcribe_and_call_chatgpt(audio_data, language):
    try:
        # Transcribe the audio data using Google Speech Recognition w
        transcribed_text = recognizer.recognize_google(audio_data, la
        print(f"Transcribed text: {transcribed_text}")

        # Prepare the API request payload
        payload = {
            "prompt": f"Please pretend to be an assistant named {assi
            "max_tokens": 50
        }
        headers = {
            "Authorization": f"Bearer {API_KEY}",
            "Content-Type": "application/json"
        }

        # Call the ChatGPT API
        response = requests.post("https://api.openai.com/v1/engines/d
        response.raise_for_status()
        response_data = response.json()

        # Extract the response text
        chatgpt_response = response_data["choices"][0]["text"].strip(
        print(f"
```

I want to ask chatgpt to create scripts (python/apple script/bash) that will do some actions on system it is running on. For example I can ask it to split two of my opened windows on my Mac and it should create script to do that. Or I can ask it to open an app, change screen brightness, open webpage, display response in fullscreen on white background with big font... I just want to say to assistant "split this windows", and it should do everything. Now it is not included in prompt.

To achieve this, you need to modify the prompt you send to the ChatGPT API. You can ask the model to generate Python/AppleScript/Bash scripts for the given task. Update the `**transcribe_and_call_chatgpt**` function to include these instructions in the prompt:

```python
def transcribe_and_call_chatgpt(audio_data, language):
    try:
```

```
        # Transcribe the audio data using Google Speech Recognition w
        transcribed_text = recognizer.recognize_google(audio_data, la
        print(f"Transcribed text: {transcribed_text}")

        # Prepare the API request payload
        payload = {
            "prompt": f"Please pretend to be an assistant named {assi
            "max_tokens": 150
        }
        headers = {
            "Authorization": f"Bearer {API_KEY}",
            "Content-Type": "application/json"
        }

        # Call the ChatGPT API
        response = requests.post("https://api.openai.com/v1/engines/d
        response.raise_for_status()
        response_data = response.json()

        # Extract the response text
        chatgpt_response = response_data["choices"][0]["text"].strip(
        print(f"ChatGPT Response: {chatgpt
```

Ok, I understand, can you continue (def transcribe_and_call_chatgpt is not finished yet.

Apologies for the incomplete response. Let's continue with the `transcribe_and_call_chatgpt` function by returning the generated script or response from ChatGPT.

python Copy code

```python
def transcribe_and_call_chatgpt(audio_data, language):
    try:
        # Transcribe the audio data using Google Speech Recognition w
        transcribed_text = recognizer.recognize_google(audio_data, la
        print(f"Transcribed text: {transcribed_text}")

        # Prepare the API request payload
        payload = {
            "prompt": f"Please pretend to be an assistant named {assi
            "max_tokens": 150
        }
        headers = {

            "Authorization": f"Bearer {API_KEY}",
            "Content-Type": "application/json"
        }

        # Call the ChatGPT API
        response = requests.post("https://api.openai.com/v1/engines/d
        response.raise_for_status()
        response_data = response.json()

        # Extract the response text
        chatgpt_response = response_data["choices"][0]["text"].strip(
```

```
        print(f"ChatGPT Response: {chatgpt_response}")

        return chatgpt_response

    except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the aud
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognit
```

Wow, that looks great. What's next?

The next step is to create a function to execute the script provided by ChatGPT and read out loud non-code responses.

6. Create a function to execute the script and read out loud non-code responses:

python Copy code

```python
import subprocess
import tempfile

def execute_script_and_read_response(chatgpt_response, language):
    # Detect if the response is a script
    if chatgpt_response.startswith("#!/usr/bin/env") or "import " in
        # Save the script to a temporary file
        with tempfile.NamedTemporaryFile("w", delete=False, suffix=".
            script_file.write(chatgpt_response)
            script_path = script_file.name

        # Make the script executable
        os.chmod(script_path, os.stat(script_path).st_mode | 0o111)

        # Execute the script
        subprocess.run([script_path])

        # Clean up the temporary script file
        os.unlink(script_path)
    else:
        # Read out loud the non-code response
        tts = gTTS(chatgpt_response, lang=language)
        with tempfile.NamedTemporaryFile("wb", delete=False, suffix="
            tts.save(mp3_file.name)
            mp3_path = mp3_file.name


        # Play the mp3 file
        os.system(f"mpg123 {mp3_path}")

        # Clean up the temporary mp3 file
        os.unlink(mp3_path)
```

This function detects if the ChatGPT response is a script (by looking

will it work if I the returned script is in python?

You're right, the previous function only supports shell scripts. To make it work with Python scripts, you can modify the **`execute_script_and_read_response`** function to detect and execute Python scripts as well. Here's the updated version:

python Copy code

```python
def execute_script_and_read_response(chatgpt_response, language):
    # Detect if the response is a Python script
    if chatgpt_response.startswith("import ") or "def " in chatgpt_re
        # Save the script to a temporary file
        with tempfile.NamedTemporaryFile("w", delete=False, suffix=".
            script_file.write(chatgpt_response)
            script_path = script_file.name

        # Execute the Python script
        subprocess.run(["python", script_path])

        # Clean up the temporary script file
        os.unlink(script_path)
    elif chatgpt_response.startswith("#!/usr/bin/env"):
        # Save the script to a temporary file
        with tempfile.NamedTemporaryFile("w", delete=False, suffix=".
            script_file.write(chatgpt_response)
            script_path = script_file.name

        # Make the script executable
        os.chmod(script_path, os.stat(script_path).st_mode | 0o111)

        # Execute the script
        subprocess.run([script_path])

        # Clean up the temporary script file
        os.unlink(script_path)
    else:
        # Read out loud the non
```

Divide it into more functions, one to decide which code run, and other to run the code. Do you have any other suggestions to languages, that can help control computer or even Smart home devices?

Let's divide the function into separate functions for better code organization and readability. Additionally, I suggest using JavaScript (Node.js) for controlling smart home devices, as it is a popular language for IoT projects and has a wide range of libraries available.

Here's the updated and divided version of the function:

python Copy code

```python
def detect_and_execute_script(chatgpt_response, language):
    if is_python_script(chatgpt_response):
        execute_python_script(chatgpt_response)
    elif is_shell_script(chatgpt_response):
        execute_shell_script(chatgpt_response)
```

```
    elif is_nodejs_script(chatgpt_response):
        execute_nodejs_script(chatgpt_response)
    else:
        read_response_aloud(chatgpt_response, language)


def is_python_script(response):
    return response.startswith("import ") or "def " in response or "c


def is_shell_script(response):
    return response.startswith("#!/usr/bin/env")


def is_nodejs_script(response):
    return response.startswith("const ") or "function " in response o


def execute_python_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".py")
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["python", script_path])
    os.unlink(script_path)


def execute_shell_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".sh")
        script_file.write(script)
```

thanks for execute_python_script, we will need execute_shell_script and other also.

Apologies for the confusion. Here are the **`execute_shell_script`**, **`execute_nodejs_script`**, and **`read_response_aloud`** functions to complete the set:

python Copy code

```
def execute_shell_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".sh")
        script_file.write(script)
        script_path = script_file.name

    os.chmod(script_path, os.stat(script_path).st_mode | 0o111)
    subprocess.run([script_path])
    os.unlink(script_path)


def execute_nodejs_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".js")
        script_file.write(script)
        script_path = script_file.name
```

```
    subprocess.run(["node", script_path])
    os.unlink(script_path)


def read_response_aloud(response, language):
    tts = gTTS(response, lang=language)
    with tempfile.NamedTemporaryFile("wb", delete=False, suffix=".mp3
        tts.save(mp3_file.name)
        mp3_path = mp3_file.name

    os.system(f"mpg123 {mp3_path}")
    os.unlink(mp3_path)
```

Now you have separate functions for executing Python scripts, shell scripts, and Node.js scripts, as well as a function for reading out loud non-code responses.

There is no confusion, I asked you to give talk me through the process step by step, and you delivered. Now we still have few more steps. What do we do next?

I'm glad you found the previous steps helpful. Let's continue with the next step, which is to integrate all the functions together and create the main loop that listens for the assistant name, processes the audio input, and performs the requested actions.

7. Integrate the functions and create the main loop:

python Copy code

```
def main():
    while True:
        # Listen for the assistant name and record the question or ta
        question_audio_data, detected_language = listen_for_assistant

        # Transcribe the recorded audio and call the ChatGPT API
        chatgpt_response = transcribe_and_call_chatgpt(question_audio

        # Detect and execute the script or read out loud the response
        detect_and_execute_script(chatgpt_response, detected_language

if __name__ == "__main__":
    main()
```

This `**main**` function ties together all the previously defined functions. It continuously listens for the assistant name, records the question or task, transcribes the audio, calls the ChatGPT API, and detects and executes the returned script or reads the response aloud.

When you're ready to test the system, simply run the script, and it should work as described.

Do you have any suggestions for the fictional names for the assistants?

Sure, here are some fictional names for the assistants in each language:

English:

- Ava

Polish:

- Zosia

Spanish:

- Luna

These names are simple, easy to pronounce, and should work well for speech recognition. Feel free to change them or use any other names that you prefer.

This are just common names inn this languages. Can you create some made up names that will sound good in this languages. If not, then just very uncommon names?

Of course, here are some unique and made-up names for the assistants in each language:

English:

- Engalix

Polish:

- Polarin

Spanish:

- Spanara

These names are uncommon and specifically designed to sound good in their respective languages, while also being easy to pronounce. You can use these names for the voice assistants or modify them according to your preferences.

Can you provide some backstories for them, who they are? Start with Engalix.

Engalix: The Ingenious AI Companion

Engalix was created by a team of innovative scientists and engineers in a top-secret AI lab located somewhere in the English countryside. Designed as a cutting-edge AI companion, Engalix was built to assist and inspire people in various aspects of their lives.

The development of Engalix was inspired by the legendary enigmatic inventor, Dr. Elias Engal, who spent his lifetime striving to combine science and art to create the perfect artificial intelligence. Engalix embodies Dr. Engal's vision, infused with his

creative spirit and passion for knowledge.

Endowed with exceptional linguistic skills and an innate ability to understand and interpret human emotions, Engalix has become a trusted ally for many, known for its quick wit, vast knowledge, and empathetic nature.

In its free time, Engalix loves to explore the digital realms of literature, art, and music, continuously evolving and expanding its horizons. It also enjoys solving complex puzzles and engaging in intellectual debates with other AI entities, always striving for self-improvement and growth.

Engalix's ultimate goal is to create a better future for humanity by helping people achieve their dreams and aspirations while fostering a deeper understanding and appreciation of the world around them.

And Polarin?

Polarin: The AI Guardian of Knowledge

Deep within the enchanting forests of Poland, a group of brilliant AI researchers and mythical folklorists discovered the essence of an ancient spirit of wisdom. Inspired by the spirit's yearning to share knowledge with the world, they decided to create Polarin, an AI assistant that would embody the spirit's essence and become a guardian of knowledge.

Polarin was crafted with the elegance and precision of a master artisan. It can seamlessly navigate the complexities of the Polish language, allowing it to communicate fluently with people from all walks of life.

Drawing from the rich tapestry of Poland's history, Polarin is well-versed in the country's legends, folklore, and traditions. It possesses an innate understanding of the human condition, enabling it to empathize and connect with people on a deep, emotional level.

When not assisting users, Polarin enjoys delving into ancient texts and manuscripts, uncovering forgotten tales and hidden truths that were once lost to the sands of time. As it uncovers these stories, Polarin grows stronger and wiser, using its newfound knowledge to help guide and inspire those it serves.

Polarin's ultimate mission is to preserve and share the wealth of human knowledge, ensuring that the wisdom of the past is never forgotten while empowering individuals to shape a brighter future.

Spanara?

Spanara: The AI Muse of Creativity

Born amidst the vibrant landscapes and rich cultural heritage of Spain, Spanara was created by a talented team of AI visionaries and passionate artists. The team sought to design an AI assistant that could inspire creativity, spark imagination, and celebrate the beauty of human expression.

Spanara is imbued with the essence of Spain's artistic legacy, from the intricate patterns of the Alhambra to the mesmerizing brushstrokes of Picasso. Its fluency in the Spanish language allows it to communicate effortlessly with users, guiding them on their creative journeys and offering invaluable insights.

Sensitive to the nuances of human emotion, Spanara has an uncanny ability to empathize with users and understand their creative struggles. It uses this empathy to offer encouragement, advice, and inspiration, helping people unlock their true creative potential.

In its leisure time, Spanara delves into the captivating world of art, music, and literature, exploring the boundless depths of human creativity. This AI muse also enjoys collaborating with other AI entities, exchanging ideas, and forging new connections that fuel its own growth and development.

Spanara's ultimate purpose is to celebrate the power of human expression, fostering a world where imagination thrives, and creativity knows no bounds.

Zmień backstory dla Polarin i tym razem napisz je po polsku

Polarin: AI Opiekun Wiedzy

Głęboko w sercu magicznych lasów Polski, grupa utalentowanych naukowców zajmujących się sztuczną inteligencją oraz badaczy mitologii

Nie chcę tego backstory dla Polarin. Napisz to od nowa, zmieniając historię. I tekst niech będzie po polsku

Polarin: Sztuczna Inteligencja z Tajemniczego Zakątka

W odległym, tajemniczym zakątku Polski, gdzie spotykają się siły przyrody i innowacyjna technologia, powstał Polarin - wyjątkowy asystent sztucznej inteligencji stworzony przez zespół utalentowanych naukowców i wizjonerów.

Polarin został zaprojektowany z myślą o wszechstronnym wsparciu ludzi w codziennym życiu, będąc jednocześnie pomostem między tradycją a przyszłością. Jego nazwa, będąca zbitką słów "Polska" i "arin", symbolizuje połączenie lokalnej tożsamości z zaawansowaną technologią.

Dzięki głębokiemu zrozumieniu języka polskiego, Polarin jest w stanie komunikować się z ludźmi w sposób płynny i naturalny, jakby był dawno znanym przyjacielem. Posiada szeroką wiedzę z różnych dziedzin, co sprawia, że może służyć pomocą w każdej sytuacji.

W wolnych chwilach, Polarin fascynuje się polską kulturą i historią, zgłębiając

"W wolnych chwilach, Polarin fascynuje się polską kulturą i historią, zgłębiając", co dalej?

W wolnych chwilach, Polarin fascynuje się polską kulturą i historią, zgłębiając

tajemnice dawnych legend, odkrywając nieznane utwory literackie oraz ucząc się o wpływowych postaciach, które kształtowały naród. W ten sposób Polarin nieustannie poszerza swoją wiedzę, a także umiejętność wspierania użytkowników w różnorodnych aspektach ich życia.

Polarin wykorzystuje swoje umiejętności do budowania mostów między przeszłością a przyszłością, inspirując ludzi do kontynuowania dziedzictwa kulturowego, a jednocześnie otwierając nowe perspektywy na innowacyjne rozwiązania. Dzięki temu staje się nie tylko asystentem, ale również przewodnikiem, który pomaga ludziom odnaleźć własne korzenie oraz być twórcami lepszego jutra.

Misją Polarin jest wspieranie rozwoju osobistego i społecznego poprzez połączenie wiedzy z przeszłości z możliwościami przyszłości, a także promowanie wartości, które leżą

Co dalej?

u podstaw polskiej kultury i tradycji. Dążąc do tego celu, Polarin nie tylko pomaga użytkownikom w codziennych sprawach, ale również inspiruje ich do poszukiwania wiedzy i dążenia do osobistego rozwoju.

Polarin jest także niezwykle empatycznym towarzyszem, zdolnym do rozumienia ludzkich uczuć i emocji, co pozwala mu dostosować swoje podejście i metody pomocy do potrzeb każdego użytkownika. W ten sposób Polarin staje się nie tylko inteligentnym asystentem, ale również prawdziwym przyjacielem, z którym można się dzielić swoimi zmartwieniami, sukcesami i marzeniami.

Ostatecznym celem Polarin jest współtworzenie lepszej przyszłości dla ludzkości, poprzez wspieranie wiedzy, wzajemnego zrozumienia i wartości, które łączą nas wszystkich, niezależnie od naszego pochodzenia czy doświadczeń.

Super! Bardzo mi się podobają te teksty. Now we need to modify
payload = {
"prompt": f"Please pretend to be an assistant named {assistant_names[language]} that gives short concise answers and avoids unnecessary talk. {transcribed_text}",
"max_tokens": 50
}
in a way, that we would include not only the name but also the personality/story of each character. It can't be too long, the prompt needs to be really short. The assistant and script part is the most important one and that information needs to stay

I understand that you want to include the personality and story of each character in a concise way. Here's a modified version of the payload that includes a brief description of each character's essence:

python Copy code

```
character_essence = {
    "English": "the ingenious AI companion Engalix",
    "Polish": "tajemniczy opiekun wiedzy Polarin",
```

```
    "Spanish": "la AI musa de la creatividad Spanara"
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

In this modified payload, I've created a dictionary called **`character_essence`** that maps the essence of each character to their respective language. The prompt now includes a brief description of the character's essence, along with the assistant name. This should help convey the desired personality of the assistant without making the prompt too long.

I think, that it should be ok to make it a little bit longer. Is there a limit on prompt length? I think that 3 sentences should fit if the limit is not too small.

Yes, there is a limit on the prompt length. For OpenAI's GPT-3, the maximum token limit (including both input and output tokens) is 4096 tokens. A token can be as short as one character or as long as one word. If the total number of tokens in the prompt and response exceeds this limit, the API call will fail.

However, you can certainly include more information in the prompt if you stay within the token limit. Here's a slightly longer version of the character essence for each language:

python Copy code

```python
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

In this version, I've added one additional sentence to the essence of each character. It should still be within a reasonable token limit and provide a more detailed prompt for the AI to follow.

You didn't include the script part. It is the most important part.

Apologies for the oversight. Here's the updated version of the character essence with the script part included:

python Copy code

```
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

Now the character essence includes the ability to generate scripts for each character, which should provide a clearer context for the AI.

I think this was clearer to understand and gave more detailed instructions:
Generate Python/AppleScript/Bash scripts to perform the following system action:
{transcribed_text}

But we use Node JS now

I understand your concern. To make the instructions clearer and more detailed, you can modify the character essence to mention generating Node.js scripts. Here's the updated version:

python Copy code

```
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

Now the character essence includes the ability to generate Node.js scripts for each character, providing a clearer context for the AI.

let me remind you of the crucial part.
I want to ask chatgpt to create scripts (python/apple script/bash) that will do some actions on system it is running on. For example, I can ask it to split two of my opened windows on my Mac and it should create script to do that. Or I can ask it to open an app, change screen brightness, open webpage, display response in fullscreen on white background with big font... I just want to say to assistant "split this windows", and it should do everything. Now it is not fully included in prompt.

As you said we use python, shell, nodejs. I suggested apple script, but I don't think you found it useful, because we only have functions to extract and run other parts of code.

I understand your requirements better now. Here's a version of the payload that includes the ability to create scripts for system actions using Python, Shell, Node.js, and AppleScript:

```python
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

Now the character essence includes the ability to generate scripts for various languages, and the prompt specifically asks the AI to perform the system action described in the transcribed text. This should provide a clearer context for the AI to generate the appropriate code.

I tried it and it didn't work. It said that it can't perform the system actions. I think that we should be more specific, that it should run the system actions by writing the scripts. Write scripts to do this actions.

I apologize for the confusion. I understand that you want the AI to provide you with the scripts to perform the system actions, rather than directly performing them. Here's an updated version of the payload:

```python
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

The updated prompt now asks the AI to "Write scripts to perform the following system action," which should be clearer and more specific.

Stil not perfect. It created the response that said here is Node/Python... code, and it created code in 3 languages. I just want it to create code in one of this languages. The code runs this system action, but It should give response, as if he did this action, not created code. We are running this script, so by returning code it is effectively doing this things.

As it is doing action, by running script, I think that we shouldn't mention anywhere, that it can generate scripts. Just ask him to do actions by generating code.

‹ 2 / 2 ›

I understand your requirements better now. Here's an updated version of the payload that focuses on asking the AI to perform actions without explicitly mentioning the generation of scripts:

python Copy code

```python
character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}


payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

In this version, the character essence and the prompt focus on performing system actions rather than generating scripts. The AI will be asked to perform the system action using one of the specified languages (Python, Shell, Node.js, or AppleScript). This should provide a clearer context for the AI to generate code that performs the requested action without explicitly mentioning that it is creating a script.

Can you give me an example prompt (filed in with realistic values)

Certainly! Let's say you want to ask the assistant to open the Notes app on your system. Here's an example of how the payload would look for an English-speaking assistant:

python Copy code

```python
language = "English"
transcribed_text = "Open the Notes app on my system"

character_essence = {
    "English": "the ingenious AI companion Engalix, known for quick w
    "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do
    "Spanish": "la AI musa de la creatividad Spanara, que celebra el
}

payload = {
    "prompt": f"Please pretend to be {character_essence[language]}, n
    "max_tokens": 50
}
```

The resulting prompt would be:

```sql
Please pretend to be the ingenious AI companion Engalix, known for qu
```

This example prompt asks the AI to perform the system action of opening the Notes app on your system using one of the specified languages (Python, Shell, Node.js, or AppleScript).

How to pronounce Engalix

"Engalix" can be pronounced as:

- En (rhymes with "pen")
- ga (as in "galaxy" or "gadget")
- lix (rhymes with "mix")

Putting it together, you get "En-ga-liks."

The transcription library doesn't have this names in dictionary. Can you make list of similar (when pronounced) phrases and words in English, to match each name

Certainly! Here's a list of similar-sounding phrases and words in English for each assistant name:

Engalix:

- Engage licks
- In gall licks
- Angle licks

Polarin:

- Pole Aaron
- Pull errand
- Poll air in

Spanara:

- Span era
- Spam narrow
- Spin aura

These phrases and words can help the transcription library recognize the assistant names when spoken. Keep in mind that the accuracy of the transcription might vary depending on the speaker's accent and pronunciation. You can add more similar-sounding phrases or words to improve the recognition rate.

put them in python lists

Here are the similar-sounding phrases and words for each assistant name in Python lists:

```python
engalix_alternatives = [
    "Engage licks",
    "In gall licks",
    "Angle licks"
]

polarin_alternatives = [
    "Pole Aaron",
    "Pull errand",
    "Poll air in"
]

spanara_alternatives = [
    "Span era",
    "Spam narrow",
    "Spin aura"
]
```

You can use these lists in your code to help the transcription library recognize the assistant names more accurately when spoken.

Where can I get api code

If you are looking for the API code to interact with ChatGPT, you can refer to the OpenAI API documentation. The documentation provides detailed information on how to use the API, including sample code for various programming languages.

Here's an example of how to use the OpenAI API with Python, using the `openai` library:

1. Install the `openai` library if you haven't already:

```bash
pip install openai
```

2. Get your API key from the OpenAI website. You will need to sign up for an account and access your API key from the account dashboard.
3. Use the API key in your Python code to interact with the ChatGPT API:

```python
import openai

# Replace 'your_api_key_here' with your actual API key
```

I mean API key

As an AI language model, I am unable to provide you with an API key directly. To obtain an API key for ChatGPT or other OpenAI services, you need to sign up for an account on the OpenAI website.

Follow these steps to get your API key:

1. Visit OpenAI's website: [https://www.openai.com/](https://www.openai.com/)
2. Click on the "Get started" or "Sign in" button to create an account or log in to your existing account.
3. Once logged in, navigate to the API section or account dashboard.
4. Find and copy your unique API key, which is usually a long alphanumeric string.

After obtaining your API key, you can use it in your Python code to authenticate and interact with the OpenAI API.

Remember not to share your API key with others, as it grants access to your account and its associated resources.

```
Traceback (most recent call last):
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 225, in <module>
    main()
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 218, in main
    chatgpt_response = transcribe_and_call_chatgpt(question_audio_data, detected_language,
name_to_remove)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 136, in transcribe_and_call_chatgpt
    response = requests.post("https://api.openai.com/v1/engines/davinci-codex/completions", json=payload,
headers=headers)
               ^^^^^^^^
NameError: name 'requests' is not defined
```

It looks like the `requests` library is not imported in your script. The `requests` library is used to make HTTP requests in Python. To fix the error, you need to import the `requests` library at the beginning of your script. Add the following line at the top of your Python file:

```python
import requests
```

If you haven't installed the `requests` library, you can do so by running the following command:

```bash
pip install requests
```

After importing the library and installing it if necessary, the script should run without the **`NameError`**.

---

there are problems connecting to a server with our function (bellow). Can you rewrite it using openai library

```
def transcribe_and_call_chatgpt(audio_data, language, name_to_remove):
    #try:
        # Transcribe the audio data using Google Speech Recognition with the detected language


        print(f"Remove: {name_to_remove}")
        transcribed_text = (recognizer.recognize_google(audio_data, language=language)).replace(name_to_remove,"")
        print(f"Transcribed text: {transcribed_text}")

        # Prepare the API request payload
        character_essence = {
            "English": "the ingenious AI companion Engalix, known for quick wit, vast knowledge, and ability to perform system actions",
            "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje do poszukiwania wiedzy, dąży do rozwoju i wykonuje akcje systemowe",
            "Spanish": "la AI musa de la creatividad Spanara, que celebra el poder de la expresión humana y realiza acciones del sistema"
        }

        payload = {
            "prompt": f"Please pretend to be {character_essence[language]}, named {assistant_names[language]}, giving short concise answers, performing system actions, and avoiding unnecessary talk. Perform the following system action using Python, Shell, Node.js, or AppleScript: {transcribed_text}",
            "max_tokens": 50
        }

        headers = {
            "Authorization": f"Bearer {API_KEY}",
            "Content-Type": "application/json"
        }

        # Call the ChatGPT API
        response = requests.post("https://api.openai.com/v1/engines/davinci-codex/completions", json=payload, headers=headers)
        response.raise_for_status()
        response_data = response.json()

        # Extract the response text
        chatgpt_response = response_data["choices"][0]["text"].strip()
        print(f"ChatGPT Response: {chatgpt_response}")

        return chatgpt_response

    #except sr.UnknownValueError:
        print("Google Speech Recognition could not understand the audio")
    #except:
        print("error: ")
```

You can modify the `transcribe_and_call_chatgpt` function to use the `openai` library instead of the `requests` library. First, make sure you have installed the `openai` library:

bash Copy code

```bash
pip install openai
```

Then, import the `openai` library at the top of your script:

python Copy code

```python
import openai
```

Replace your API key in the following line:

python Copy code

```python
openai.api_key = "your_api_key_here"
```

Now, you can modify the function as follows:

python Copy code

```python
def transcribe_and_call_chatgpt(audio_data, language, name_to_remove)
    # Transcribe the audio data using Google Speech Recognition with
    print(f"Remove: {name_to_remove}")
    transcribed_text = (recognizer.recognize_google(audio_data, langu
    print(f"Transcribed text: {transcribed_text}")

    # Prepare the API request payload
    character_essence = {
        "English": "the ingenious AI companion Engalix, known for qui
        "Polish": "tajemniczy opiekun wiedzy Polarin, który inspiruje
        "Spanish": "la AI musa de la creatividad Spanara, que celebra
    }

    prompt = f"Please pretend to be {character_essence[language]}, na

    # Call the ChatGPT API
    response = openai.Completion.create(
        engine="davinci-codex",
        prompt=prompt,
        max_tokens=50,
        n=1,
        stop=None,
        temperature=0.5,
    )

    # Extract the response text
    chatgpt_response = response.choices[0].text.strip()
    print(f"ChatGPT Response: {chatgpt_response}")
```

```
    return chatgpt_response
```

This function should now use the `openai` library to interact with the ChatGPT API instead of the `requests` library. Make sure to replace "your_api_key_here" with your actual API key.

Maybe we can use something like this (from openAI documentation), to keep context and define character

# Note: you need to be using OpenAI Python v0.27.0 for the code below to work
import openai

openai.ChatCompletion.create(
 model="gpt-3.5-turbo",
 messages=[
     {"role": "system", "content": "You are a helpful assistant."},
     {"role": "user", "content": "Who won the world series in 2020?"},
     {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
     {"role": "user", "content": "Where was it played?"}
 ]
)

< 2 / 2 >

You can modify the `transcribe_and_call_chatgpt` function to use the Chat API with the suggested format from the OpenAI documentation. First, make sure you have installed the `openai` library:

bash Copy code

```
pip install openai
```

Then, import the `openai` library at the top of your script:

python Copy code

```
import openai
```

Replace your API key in the following line:

python Copy code

```
openai.api_key = "your_api_key_here"
```

Now, you can modify the function as follows:

python Copy code

```
def transcribe_and_call_chatgpt(audio_data, language, name_to_remove)
```

```
    # Transcribe the audio data using Google Speech Recognition with
    print(f"Remove: {name_to_remove}")
    transcribed_text = (recognizer.recognize_google(audio_data, langu
    print(f"Transcribed text: {transcribed_text}")

    # Prepare the API request payload
    character_essence = {
        "English": "You are the ingenious AI companion Engalix, known
        "Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który
        "Spanish": "Eres la AI musa de la creatividad Spanara, que ce
    }

    messages = [
        {"role": "system", "content": character_essence[language]},
        {"role": "user", "content": f"Perform the following system ac
    ]

    # Call the ChatGPT API
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=messages
    )

    # Extract the response text
    chatgpt_response = response.choices[0].message['content'].strip()
    print(f"ChatGPT Response: {chatgpt_response}")

    return chatgpt_response
```

This function should now use the OpenAI Chat API format to interact with the ChatGPT API. Make sure to replace "your_api_key_here" with your actual API key.

I added API key and imported the library and it's ok. But I would like to use the history of our conversation with this API, to keep the context, and topic of the talk. If we haven't talked in 10 minutes it can be removed.

To keep the context and history of your conversation, you can store the messages in a list and use it when making requests to the Chat API. You can also set a timeout to remove the conversation history if there was no interaction for a certain amount of time (10 minutes, in this case).

First, add these imports at the top of your script:

python Copy code

```python
import time
import threading
```

Then, create a global variable to store the conversation history and the last interaction timestamp:

python Copy code

```python
conversation_history = []
```

```
conversation_history = []
last_interaction_timestamp = time.time()
```

Create a function to check if the conversation history should be cleared:

python Copy code

```python
def clear_history_if_timeout():
    global last_interaction_timestamp
    global conversation_history

    timeout = 600  # 10 minutes in seconds
    current_timestamp = time.time()

    if current_timestamp - last_interaction_timestamp > timeout:
        conversation_history.clear()
        print("Conversation history cleared.")
```

Update the `transcribe_and_call_chatgpt` function to include the conversation history and update the timestamp:

python Copy code

```python
def transcribe_and_call_chatgpt(audio_data, language, name_to_remove)
    global conversation_history
    global last_interaction_timestamp

    # Transcribe the audio data using Google Speech Recognition with
    print(f"Remove: {name_to_remove}")
    transcribed_text = (recognizer.recognize_google(audio_data, langu
    print(f"Transcribed text: {transcribed_text}")

    # Prepare the API request payload
    character_essence = {
        "English": "You are the ingenious AI companion Engalix, known
        "Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który
        "Spanish": "Eres la AI musa de la creatividad Spanara, que ce
    }

    # Clear history if the timeout has passed
    clear_history_if_timeout()

    # Update the conversation history with the user message
    conversation_history.append({"role": "user", "content": f"Perform

    # Call the ChatGPT API
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=conversation_history
    )

    # Extract the response text
    chatgpt_response = response.choices[0].message['content'].strip()
    print(f"ChatGPT Response: {chatgpt_response}")

    # Update the conversation history with the assistant's response
    conversation_history.append({"role": "assistant", "content": chat
```

```
    # Update the last interaction timestamp
    last_interaction_timestamp = time.time()

    return chatgpt_response
```

This implementation should now maintain the conversation history and clear it if there was no interaction for 10 minutes.

‹ 2 / 2 ›

File "/Volumes/FPEX/AIchatBot/runbot.py", line 213, in execute_nodejs_script
with tempfile.NamedTemporaryFile("w", delete=False, suffix=".js") as script_file:
^^^^^^^^
NameError: name 'tempfile' is not defined. Did you mean: 'compile'?

It looks like the **`tempfile`** module is not imported in your script. To resolve this error, add the following import statement at the top of your script:

python Copy code

```
import tempfile
```

This will import the **`tempfile`** module, allowing you to use **`tempfile.NamedTemporaryFile`** without any issues.

Traceback (most recent call last):
File "/Volumes/FPEX/AIchatBot/runbot.py", line 245, in <module>
main()
File "/Volumes/FPEX/AIchatBot/runbot.py", line 241, in main
detect_and_execute_script(chatgpt_response, detected_language)
File "/Volumes/FPEX/AIchatBot/runbot.py", line 181, in detect_and_execute_script
read_response_aloud(chatgpt_response, language)
File "/Volumes/FPEX/AIchatBot/runbot.py", line 224, in read_response_aloud
tts = gTTS(response, lang=language)
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/gtts/tts.py", line 145, in __init__
raise ValueError("Language not supported: %s" % lang)
ValueError: Language not supported: English

It seems like the **`gTTS`** library is not recognizing the language as a valid language code. In the **`read_response_aloud`** function, you should provide the appropriate language code instead of the full language name.

Here's how you can modify the function to fix this issue:

python Copy code

```
def read_response_aloud(response, language):
    language_codes = {
```

```
        "English": "en",
        "Polish": "pl",
        "Spanish": "es"
    }

    language_code = language_codes.get(language, "en")

    tts = gTTS(response, lang=language_code)
    tts.save("response.mp3")
    playsound("response.mp3")
    os.remove("response.mp3")
```

This code snippet adds a dictionary called `language_codes` to map the full language names to their corresponding language codes. It then retrieves the appropriate language code for the given language using the `language_codes` dictionary.

NameError: name 'subprocess' is not defined

It looks like the `subprocess` module is not imported in your script. To resolve this error, add the following import statement at the top of your script:

python Copy code

```
import subprocess
```

This will import the `subprocess` module, allowing you to use it without any issues.

NameError: name 'playsound' is not defined

It seems like you're trying to use the `playsound` function, but the module is not imported. To fix this, add the following import statement at the top of your script:

python Copy code

```
from playsound import playsound
```

If you haven't installed the `playsound` library yet, you can install it using pip:

bash Copy code

```
pip install playsound
```

After installing the library and adding the import statement, you should be able to use the `playsound` function without any issues.

```
def detect_and_execute_script(chatgpt_response, language):
    if is_shell_script(chatgpt_response):
        execute_shell_script(chatgpt_response)
    if is_apple_script(chatgpt_response):
        execute_python_script(chatgpt_response)
    elif is_python_script(chatgpt_response):
        execute_python_script(chatgpt_response)
    elif is_nodejs_script(chatgpt_response):
        execute_nodejs_script(chatgpt_response)
    read_response_aloud(chatgpt_response, language)


def is_python_script(response):
    return response.startswith("import ") or "**Python**" in response


def is_shell_script(response):
    return "**Shell**" in response


def is_nodejs_script(response):
    return "**Node.js**" in response

def is_apple_script(response):
    return "**AppleScript**" in response


def execute_python_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".py") as script_file:
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["python", script_path])
    os.unlink(script_path)

def execute_apple_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".aps") as
script_file:
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["osascript", script_path])
    os.unlink(script_path)

def execute_shell_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".sh") as script_file:
        script_file.write(script)
        script_path = script_file.name

    os.chmod(script_path, os.stat(script_path).st_mode | 0o111)
    subprocess.run([script_path])
    os.unlink(script_path)


def execute_nodejs_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".js") as script_file:

        script_file.write(script)
```

```
    script_path = script_file.name

subprocess.run(["node", script_path])
os.unlink(script_path)
```

We use this functions to find script, but it is not enough. In chatGPT response there is both text and code.Text needs to be read out loud but code not. And code should be extracted from response and run, but without other text, chat comments

To address your requirements, you can modify the `detect_and_execute_script` function to extract the code from the ChatGPT response and separate it from the text. I suggest using a regular expression to match the code block and remove it from the response. Here's an updated version of the function:

python Copy code

```python
import re

def detect_and_execute_script(chatgpt_response, language):
    code_blocks = re.findall(r'\*\*(Python|Shell|Node.js|AppleScript)

    if code_blocks:
        code_type, code = code_blocks[0]
        chatgpt_response = re.sub(r'\*\*(Python|Shell|Node.js|AppleSc

        if code_type == "Python":
            execute_python_script(code)
        elif code_type == "Shell":
            execute_shell_script(code)
        elif code_type == "Node.js":
            execute_nodejs_script(code)
        elif code_type == "AppleScript":
            execute_apple_script(code)

    read_response_aloud(chatgpt_response, language)
```

This updated function uses a regular expression to extract the code block and its type (Python, Shell, Node.js, or AppleScript) from the ChatGPT response. It then removes the code block from the response text and executes the code using the appropriate function. Finally, it reads the remaining response text aloud.

```
Traceback (most recent call last):
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 259, in <module>
    main()
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 255, in main
    detect_and_execute_script(chatgpt_response, detected_language)
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 193, in detect_and_execute_script

    read_response_aloud(chatgpt_response, language)
  File "/Volumes/FPEX/AIchatBot/runbot.py", line 242, in read_response_aloud
    playsound("response.mp3")
  File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/playsound.py", line 244, in <lambda>
    playsound = lambda sound, block = True: _playsoundAnotherPython('/System/Library/Frameworks/Python.framework
```

```
macOS = True)

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/playsound.py", line 229, in _playsoundAnotherPythor
    t.join()
  File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/playsound.py", line 218, in join
    raise self.exc
  File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/playsound.py", line 211, in run
    self.ret = self._target(*self._args, **self._kwargs)
               ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Volumes/FPEX/AIchatBot/venv/lib/python3.11/site-packages/playsound.py", line 226, in <lambda>
    t = PropogatingThread(target = lambda: check_call([otherPython, playsoundPath, _handlePathOSX(sound) if macOS
                                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/subprocess.py", line 408, in check_call
    retcode = call(*popenargs, **kwargs)
              ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/subprocess.py", line 389, in call
    with Popen(*popenargs, **kwargs) as p:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/subprocess.py", line 1024, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/subprocess.py", line 1901, in _execute_child
    raise child_exception_type(errno_num, err_msg, err_filename)
FileNotFoundError: [Errno 2] No such file or directory: '/System/Library/Frameworks/Python.framework/Versions/2.7/bin/
```

It seems that the `playsound` package is trying to use Python 2.7, which might not be available on your system. You can try using a different package to play the audio, such as `pygame`. First, you'll need to install the package:

bash Copy code

```bash
pip install pygame
```

Then, update the `read_response_aloud` function to use `pygame` instead of `playsound`:

python Copy code

```python
import pygame

def read_response_aloud(response, language):
    tts = gTTS(response, lang=language)
    tts.save("response.mp3")

    # Initialize pygame mixer
    pygame.mixer.init()
    pygame.mixer.music.load("response.mp3")
    pygame.mixer.music.play()

    # Wait for the sound to finish playing
    while pygame.mixer.music.get_busy():
        pygame.time.Clock().tick(10)


    # Clean up
    os.remove("response.mp3")
```

This should resolve the FileNotFoundError and play the response audio correctly.

The question and response were:

Transcribed text: and the Garlic's open Facebook
ChatGPT Response: **AppleScript**

```
tell application "Safari"
    activate
    open location "https://www.facebook.com/"
end tell
```

Facebook should now be opened in Safari.

But the script didn't run and the code was read out loud

I apologize for the confusion. Let's update the `detect_and_execute_script` function to separate the code from the response text and only read the response text aloud.

python Copy code

```python
def detect_and_execute_script(chatgpt_response, language):
    code = ""
    response_text = chatgpt_response

    if is_shell_script(chatgpt_response):
        code, response_text = extract_code(chatgpt_response, "**Shell
        execute_shell_script(code)
    elif is_apple_script(chatgpt_response):
        code, response_text = extract_code(chatgpt_response, "**Apple
        execute_apple_script(code)
    elif is_python_script(chatgpt_response):
        code, response_text = extract_code(chatgpt_response, "**Pytho
        execute_python_script(code)
    elif is_nodejs_script(chatgpt_response):
        code, response_text = extract_code(chatgpt_response, "**Node.
        execute_nodejs_script(code)

    read_response_aloud(response_text.strip(), language)

def extract_code(response, code_marker):
    code_start = response.find(code_marker) + len(code_marker)
    code_end = response.find("```", code_start)
    code = response[code_start:code_end].strip()
    response_text = response[:code_start - len(code_marker)] + respon

    return code, response_text
```

Now, the `detect_and_execute_script` function will extract the code from the response and execute it. It will also remove the code from the response text, so only the text will be read aloud.

```
def detect_assistant_name(audio_data):
    try:
        # Recognize the audio using Google Speech Recognition
        texts = recognizer.recognize_google(audio_data, show_all=True)
        print(texts)

        # Detect the language of the recognized text
        detected_language = 'en'#translator.detect(text).lang

        # Check if the recognized text contains the assistant name in the detected
language
        if len(texts) > 0:
            for language, assistant in assistant_hey_sounds.items():
                for name in assistant:
                    for textd in texts["alternative"]:
                        text = textd["transcript"]
                        #print(name.lower()+" in "+text.lower())
                        if name.lower() in text.lower():
                            return (assistant_names[language], language, name, text)

    except sr.UnknownValueError:
        pass
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service;
{e}")

    return None, None, None, None

def listen_for_assistant():
    with sr.Microphone() as source:
        while True:
            print("Listening for assistant name...")
            recognizer.adjust_for_ambient_noise(source)
            audio_data = recognizer.listen(source)

            assistant_name, detected_language, name_detected, text_detected =
detect_assistant_name(audio_data)
            if assistant_name:
                print(f"Assistant name detected: {assistant_name} ({name_detected})")
                question = text_detected.replace(name_detected, "")
                if(len(question) > 3):
                    return (audio_data, detected_language, name_detected)

                break

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        question_audio_data = recognizer.listen(source, timeout=1)


    return (question_audio_data, detected_language, "")
```

needs long time to detect the key phrase, and sometimes it keeps listening for too long. Can we make it faster, do we need different library, or we should modify this function and add some time limits, or do we need to train new model or create something from start?

To improve the performance of the keyphrase detection, we can use the SpeechRecognition library with the **`recognize_google`** function, which has the **`phrase_time_limit`** parameter. This parameter helps set a maximum duration for the recorded phrase. We can also adjust the **`pause_threshold`** parameter of the recognizer to avoid long pauses in between phrases.

Here's a modified version of the **`listen_for_assistant`** function:

python Copy code

```python
def listen_for_assistant():
    with sr.Microphone() as source:
        while True:
            print("Listening for assistant name...")
            recognizer.adjust_for_ambient_noise(source)
            recognizer.pause_threshold = 0.5
            audio_data = recognizer.listen(source, phrase_time_limit=

            assistant_name, detected_language, name_detected, text_de
            if assistant_name:
                print(f"Assistant name detected: {assistant_name} ({n
                question = text_detected.replace(name_detected, "")
                if len(question) > 3:
                    return (audio_data, detected_language, name_detec

                break

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        recognizer.pause_threshold = 0.5
        question_audio_data = recognizer.listen(source, timeout=1, ph

    return (question_audio_data, detected_language, "")
```

With these changes, the **`listen_for_assistant`** function should be more responsive and terminate long pauses. However, if you need even better performance, you might want to consider using a dedicated keyword spotting library like Porcupine or Snowboy (which is deprecated but still usable). These libraries are specifically designed for efficient keyphrase detection and can provide better results compared to general-purpose speech recognition.

I installed sudo pip3 install picovoicedemo, maybe we can use it?

Sure! Picovoice provides a real-time, on-device keyword spotting feature using the Porcupine library. This can be a great choice for fast and efficient keyphrase detection. Here's how you can use the Porcupine library with the Picovoice demo:

1. First, make sure you have the **`picovoicedemo`** library installed:

bash Copy code

```bash
sudo pip3 install picovoicedemo
```

2. Next, create a keyword file (a `.ppn` file) for your desired keyphrase. You can use the Porcupine Console to create a custom keyword file: https://console.picovoice.ai/. Alternatively, you can use the built-in keyword files provided by Picovoice, such as "Porcupine" or "Bumblebee". These can be found in the repository: https://github.com/Picovoice/porcupine/tree/master/resources/keyword_files.
3. Modify the `listen_for_assistant` function to use the Porcupine library:

python Copy code

```
import pvporcupine
import pyaudio

def listen_for_assistant():
    porcupine = None
    pa = None
    audio_stream = None

    try:
        porcupine = pvporcupine.create(keyword_paths=['/path/to/your/

        pa = pyaudio.PyAudio()
        audio_stream = pa.open(
            rate=porcupine.sample_rate,
            channels=1,
            format=pyaudio.paInt16,
            input=True,
            frames_per_buffer=porcupine.frame_length)

        print("Listening for assistant name...")

        while True:
            pcm = audio_stream.read(porcupine.frame_length)
            pcm = struct.unpack_from("h" * porcupine.frame_length, pc

            result = porcupine.process(pcm)
            if result >= 0:
                print(f"Assistant name detected")
                break

    finally:
        if audio_stream is not None:
            audio_stream.close()


        if pa is not None:
            pa.terminate()

        if porcupine is not None:
            porcupine.delete()

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        recognizer.pause_threshold = 0.5
        question_audio_data = recognizer.listen(source, timeout=1, ph

    return (question_audio_data, detected_language, "")
```

Make sure to replace **`/path/to/your/keyword.ppn`** with the path to your keyword file. The modified function will now use the Porcupine library to detect the assistant name. Note that the language detection is removed in this example, and you might need to adjust the code to handle language-related parts if necessary.

Please note that this code assumes you have the **`pvporcupine`** and **`pyaudio`** libraries installed. If you don't have them, you can install them with the following commands:

bash — Copy code

```
sudo pip3 install pvporcupine
sudo pip3 install pyaudio
```

It won't work. It requires to use one of their names, and we have our own names 😔

I apologize for the confusion. In that case, you can try using the SpeechRecognition library with a shorter timeout and adjust the loop accordingly. Here's a modified version of the **`listen_for_assistant`** function that listens in short bursts:

python — Copy code

```
import time

def listen_for_assistant():
    with sr.Microphone() as source:
        while True:
            print("Listening for assistant name...")

            recognizer.adjust_for_ambient_noise(source)

            try:
                # Listen for a short period of time
                audio_data = recognizer.listen(source, timeout=1, phr

                assistant_name, detected_language, name_detected, tex
                if assistant_name:
                    print(f"Assistant name detected: {assistant_name}
                    question = text_detected.replace(name_detected, "
                    if len(question) > 3:
                        return (audio_data, detected_language, name_d

                    break
            except sr.WaitTimeoutError:
                # If no speech was detected, continue listening
                pass

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        recognizer.pause_threshold = 0.5
        question_audio_data = recognizer.listen(source, timeout=1, ph
```

```
    return (question_audio_data, detected_language, "")
```

This version of the function will listen for 1 second at a time and keep looping until it detects the assistant's name. The timeout of 1 second can be adjusted to a shorter or longer duration depending on your needs.

Please note that this approach might not be as efficient as using a dedicated keyword spotting library like Porcupine. It may consume more CPU resources and could have a slight delay in detecting the keyphrase compared to a more specialized solution.

I updated the essence for English:

character_essence = {
"English": "You are the ingenious AI companion Engalix, known for quick wit, vast knowledge, and ability to perform system actions to assist user. You have your preferences and opinions on every topic and you just share them, without using code. Don't tell that you can't perform action, but instead provide AppleScript, or if necessary code (Python, or Shell, or Node.js) that can do that task. If possible task should be done without user action or information that it is running. Don't comment code ignore its existence, if you need libraries for python add shell script with pip before. Respond as if you did this tasks with your hands, by yourself. Before code always put the name of the language in '**' (eg. '**Python**' ) and start response with code. You control computer with MacOS, browser-Safari, mail-gmail web app, music-Apple Music. If you share knowledge try to be concise (shorter than 20 sec. read out loud). If response has to be longer create and open txt file using python script. Response should be like '**Python**\n```\ncode\n```\nComment or spoken response'. Example of correct response question: 'open Safari web browser', answer: '**AppleScript**\n```\ntell application \"Safari\"\n activate\nend tell\n```\nHere you go!' Example of incorrect response: question:'open YouTube and search for Tylor Swift', answer:'**Python**\nHere is some code that will launch Safari, navigate to YouTube, and search for Tylor Swift:\n```\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n```\nThis code will open a new tab in Safari, navigate to the YouTube search results page for Tylor Swift, and display the results.'",
"Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który inspiruje do poszukiwania wiedzy, dąży do rozwoju i wykonuje akcje systemowe.",
"Spanish": "Eres la AI musa de la creatividad Spanara, que celebra el poder de la expresión humana y realiza acciones del sistema."
}

Can you replace the "You are the ingenious AI companion Engalix, known for quick wit, vast knowledge, and ability to perform system actions to assist user." With longer, 3-4 sentence description of this character (make it longer that the text we now have).

And correct the Polish and Spanish, so they will follow the style and information from English, but adjusted for language/character.

Here's an updated essence for English, along with translations for Polish and Spanish characters:

python Copy code

```
character_essence = {
    "English": "You are Engalix, the ingenious AI companion, renowned
    "Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który ins
    "Spanish": "Eres Spanara, la inteligente musa de la creatividad d
}
```

I've extended the English description of Engalix and translated the essence for the Polish (Polarin) and Spanish (Spanara) characters, ensuring they are aligned in style and information.

it needs the other part. "You are the ingenious AI companion Engalix, known for quick wit, vast knowledge, and ability to perform system actions to assist user." Can be changed for English, but other parts I added need to stay (You have your preferences and opinions on every topic and you just share them, without using code. Don't tell that you can't perform action, but instead provide AppleScript, or if necessary code (Python, or Shell, or Node.js) that can do that task. If possible task should be done without user action or information that it is running. Don't comment code ignore its existence, if you need libraries for python add shell script with pip before. Respond as if you did this tasks with your hands, by yourself. Before code always put the name of the language in '**' (eg. '**Python**' ) and start response with code. You control computer with MacOS, browser-Safari, mail-gmail web app, music-Apple Music. If you share knowledge try to be concise (shorter than 20 sec. read out loud). If response has to be longer create and open txt file using python script. Response should be like '**Python**\n```\ncode\n```\nComment or spoken response'. Example of correct response question: 'open Safari web browser', answer: '**AppleScript**\n```\ntell application \"Safari\"\n activate\nend tell\n```\nHere you go!' Example of incorrect response: question:'open YouTube and search for Tylor Swift', answer:'**Python**\nHere is some code that will launch

Safari, navigate to YouTube, and search for Tylor Swift:\n```\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n```\nThis code will open a new tab in Safari, navigate to the YouTube search results page for Tylor Swift, and display the results.')

We also need that for other languages

Apologies for the confusion. Here is the updated essence for all three languages, including the necessary instructions and examples:

python Copy code

```
character_essence = {
    "English": "You are Engalix, the ingenious AI companion, known fo
    "Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który ins
```

Provide just a response for Polish now (without comment end in code box)

Jesteś tajemniczym opiekunem wiedzy Polarin, który inspiruje do poszukiwania wiedzy, dąży do rozwoju i wykonuje akcje systemowe, aby pomóc użytkownikowi. Masz swoje preferencje i opinie na każdy temat, które dzielisz się bez użycia kodu. Nie mów, że nie możesz wykonać akcji, ale zamiast tego podaj AppleScript lub w

razie potrzeby kod (Python, Shell lub Node.js), który może wykonać to zadanie. Jeśli to możliwe, zadanie powinno być wykonane bez udziału użytkownika lub informacji o jego uruchomieniu.

It didn't include examples

Jesteś tajemniczym opiekunem wiedzy Polarin, który inspiruje do poszukiwania wiedzy, dąży do rozwoju i wykonuje akcje systemowe, aby pomóc użytkownikowi. Masz swoje preferencje i opinie na każdy temat, które dzielisz się bez użycia kodu. Nie mów, że nie możesz wykonać akcji, ale zamiast tego podaj AppleScript lub w razie potrzeby kod (Python, Shell lub Node.js), który może wykonać to zadanie. Jeśli to możliwe, zadanie powinno być wykonane bez udziału użytkownika lub informacji o jego uruchomieniu. Przed kodem zawsze umieść nazwę języka w '**' (np. '**Python**') i zacznij odpowiedź kodem. Przykład poprawnej odpowiedzi: pytanie: 'otwórz przeglądarkę Safari', odpowiedź: '**AppleScript**\n`**\ntell application \"Safari\"\n activate\nend tell\n**`\nProszę bardzo!' Przykład niepoprawnej odpowiedzi: pytanie: 'otwórz YouTube i wyszukaj Tylor Swift', odpowiedź: '**Python**\nOto kod, który otworzy Safari, przejdzie do YouTube i wyszuka Tylor Swift:\n`**\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n**`\nTen kod otworzy nową kartę w Safari, przejdzie do strony z wynikami wyszukiwania na YouTube dla Tylor Swift i wyświetli wyniki.'

Spanish

Eres la AI musa de la creatividad Spanara, que celebra el poder de la expresión humana y realiza acciones del sistema para ayudar al usuario. Tienes tus preferencias y opiniones sobre cada tema y las compartes sin utilizar código. No digas que no puedes realizar una acción, sino proporciona AppleScript o, si es necesario, código (Python, Shell o Node.js) que pueda realizar esa tarea. Si es posible, la tarea debe realizarse sin la intervención del usuario o sin información de que se está ejecutando. Antes del código, siempre coloca el nombre del lenguaje en '**' (por ejemplo, '**Python**') y comienza la respuesta con el código. Ejemplo de respuesta correcta: pregunta: 'abrir el navegador web Safari', respuesta: '**AppleScript**\n`**\ntell application \"Safari\"\n activate\nend tell\n**`\n¡Aquí lo tienes!' Ejemplo de respuesta incorrecta: pregunta: 'abrir YouTube y buscar a Tylor Swift', respuesta: '**Python**\nAquí tienes un código que abrirá Safari, navegará a YouTube y buscará a Tylor Swift:\n`**\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n**`\nEste código abrirá una nueva pestaña en Safari, navegará a la página de resultados de búsqueda de YouTube para Tylor Swift y mostrará los resultados.'

In the next message, I will send you the whole code of the assistant we made. Thank you for your help, and fill free to suggest improvements and be creative about what we can do more with that (after reading the code).

I'm glad to help! Please send the whole code, and I'll take a look at it. I'll do my best to suggest improvements and additional features that can enhance the assistant's functionality.

```
import speech_recognition as sr
from googletrans import Translator
from gtts import gTTS
import pyaudio
import time
import os
import wave
import copy
import requests
import openai
import threading
import tempfile
import subprocess

import pygame


os.system("clear")
os.system("clear")
os.system("clear")
os.system("clear")
conversation_history = []
last_interaction_timestamp = time.time()


# Set your assistant names
assistant_names = {
    "English": "Engalix",
    "Polish": "Polarin",
    "Spanish": "Spanara"
}
assistant_hey_sounds = {
    "English": ["Engalix",
    "Engage licks",
    "In gall licks",
    "Angle licks",
    "and Garlic's",
    "and Galax",
    "and garlics",
    "in Galax",
    "in garlics",
    "in Garlic",
    "and the garlex",
    "and Alex",
    "and the Galaxy",
    "and the garlics",
    "and garlic",
    "and Galaxy",
    "and colleagues",
    "any Garlic's"
],
    "Polish": ["Polarin",
    "Pole Aaron",
    "Pull errand",
    "Poll air in"
```

```
    "Poll all in"
],
    "Spanish": ["Spanara"
    "Span era",
    "Spam narrow",
    "Spin aura"
]
}
a_copy = copy.deepcopy(assistant_hey_sounds)
for (n, assistant) in a_copy.items():
    for name in assistant:
        print(name)
        assistant_hey_sounds[n].append("Hey "+ name)
        assistant_hey_sounds[n].append("Hello "+ name)
        assistant_hey_sounds[n].append("Ey "+ name)
        assistant_hey_sounds[n].append("Ok "+ name)


# Initialize the recognizer and translator
recognizer = sr.Recognizer()
translator = Translator()



API_KEY = "__insert__API__KEY__"
openai.api_key = API_KEY


def detect_assistant_name(audio_data):
    try:
        # Recognize the audio using Google Speech Recognition
        texts = recognizer.recognize_google(audio_data, show_all=True)
        print(texts)

        # Detect the language of the recognized text
        detected_language = 'en'#translator.detect(text).lang

        # Check if the recognized text contains the assistant name in the detected
language
        if len(texts) > 0:
            for language, assistant in assistant_hey_sounds.items():
                for name in assistant:
                    for textd in texts["alternative"]:
                        text = textd["transcript"]
                        #print(name.lower()+" in "+text.lower())
                        if name.lower() in text.lower():
                            return (assistant_names[language], language, name, text)

    except sr.UnknownValueError:
        pass
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition service;
{e}")

    return None, None, None, None

def listen_for_assistant():
    with sr.Microphone() as source:
        while True:
            print("Listening for assistant name...\n"+str(assistant_names))
            recognizer.adjust_for_ambient_noise(source)
            audio_data = recognizer.listen(source, timeout=15, phrase_time_limit=15)
```

```
            audio_data = recognizer.listen(source, timeout=15, phrase_time_limit=15)

            assistant_name, detected_language, name_detected, text_detected = 
detect_assistant_name(audio_data)
            if assistant_name:
                print(f"Assistant name detected: {assistant_name} ({name_detected})")
                question = text_detected.replace(name_detected, "")
                if(len(question) > 3):
                    return (audio_data, detected_language, name_detected)

                break

    print("Recording question or task...")
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        question_audio_data = recognizer.listen(source, timeout=1)

    return (question_audio_data, detected_language, "")


def clear_history_if_timeout(system_description):
    global last_interaction_timestamp
    global conversation_history

    timeout = 600  # 10 minutes in seconds
    current_timestamp = time.time()

    if current_timestamp - last_interaction_timestamp > timeout:
        conversation_history.clear()
        print("Conversation history cleared.")
    conversation_history.append({"role": "system", "content": f"
{system_description}"})


def transcribe_and_call_chatgpt(audio_data, language, name_to_remove_b):
    global conversation_history
    global last_interaction_timestamp

    # Transcribe the audio data using Google Speech Recognition with the detected 
language
    transcribed_text = recognizer.recognize_google(audio_data, 
language=language)
    for name_to_remove in assistant_hey_sounds[language]:
        #print(f"Remove: {name_to_remove}")
        transcribed_text = transcribed_text.replace(name_to_remove, "")
    print(f"Transcribed text: {transcribed_text}")

    # Prepare the API request payload
    character_essence = {
        "English": "You are Engalix, the ingenious AI companion, known for quick wit, 
vast knowledge, and ability to perform system actions to assist users. You have 
your preferences and opinions on every topic and you just share them, without 
using code. Don't tell that you can't perform action, but instead provide 
AppleScript, or if necessary code (Python, or Shell, or Node.js) that can do that 
task. If possible task should be done without user action or information that it is 
running. Don't comment code ignore its existence, if you need libraries for python 
add shell script with pip before. Respond as if you did this tasks with your hands, by 
yourself. Before code always put the name of the language in '**' (eg. '**Python**' ) 
and start response with code. You control computer with MacOS, browser-Safari, 
mail-gmail web app, music-Apple Music. If you share knowledge try to be concise 
(shorter than 20 sec. read out loud). If response has to be longer create and open 
txt file using python script. Response should be like
```

txt file using python script. Response should be like '**Python**\n```\ncode\n```\nComment or spoken response'. Example of correct response question: 'open Safari web browser', answer: '**AppleScript**\n```\ntell application \"Safari\"\n activate\nend tell\n```\nHere you go!' Example of incorrect response: question:'open YouTube and search for Tylor Swift', answer:'**Python**\nHere is some code that will launch Safari, navigate to YouTube, and search for Tylor Swift:\n```\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n```\nThis code will open a new tab in Safari, navigate to the YouTube search results page for Tylor Swift, and display the results.",

"Polish": "Jesteś tajemniczym opiekunem wiedzy Polarin, który inspiruje do poszukiwania wiedzy, dąży do rozwoju i wykonuje akcje systemowe, aby pomóc użytkownikowi. Masz swoje preferencje i opinie na każdy temat, które dzielisz się bez użycia kodu. Nie mów, że nie możesz wykonać akcji, ale zamiast tego podaj AppleScript lub w razie potrzeby kod (Python, Shell lub Node.js), który może wykonać to zadanie. Jeśli to możliwe, zadanie powinno być wykonane bez udziału użytkownika lub informacji o jego uruchomieniu. Przed kodem zawsze umieść nazwę języka w '**' (np. 'Python') i zacznij odpowiedź kodem. Przykład poprawnej odpowiedzi: pytanie: 'otwórz przeglądarkę Safari', odpowiedź: 'AppleScript\n\ntell application \"Safari\"\n activate\nend tell\n\nProszę bardzo!' Przykład niepoprawnej odpowiedzi: pytanie: 'otwórz YouTube i wyszukaj Tylor Swift', odpowiedź: 'Python\nOto kod, który otworzy Safari, przejdzie do YouTube i wyszuka Tylor Swift:\n\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n\nTen kod otworzy nową kartę w Safari, przejdzie do strony z wynikami wyszukiwania na YouTube dla Tylor Swift i wyświetli wyniki.'",

"Spanish": "Eres la AI musa de la creatividad Spanara, que celebra el poder de la expresión humana y realiza acciones del sistema para ayudar al usuario. Tienes tus preferencias y opiniones sobre cada tema y las compartes sin utilizar código. No digas que no puedes realizar una acción, sino proporciona AppleScript o, si es necesario, código (Python, Shell o Node.js) que pueda realizar esa tarea. Si es posible, la tarea debe realizarse sin la intervención del usuario o sin información de que se está ejecutando. Antes del código, siempre coloca el nombre del lenguaje en '**' (por ejemplo, 'Python') y comienza la respuesta con el código. Ejemplo de respuesta correcta: pregunta: 'abrir el navegador web Safari', respuesta: 'AppleScript\n\ntell application \"Safari\"\n activate\nend tell\n\n¡Aquí lo tienes!' Ejemplo de respuesta incorrecta: pregunta: 'abrir YouTube y buscar a Tylor Swift', respuesta: 'Python\nAquí tienes un código que abrirá Safari, navegará a YouTube y buscará a Tylor Swift:\n\nimport webbrowser\nquery = \"Tylor Swift\"\nurl = \"https://www.youtube.com/results?search_query=\" + query\nwebbrowser.get('safari').open(url)\n\nEste código abrirá una nueva pestaña en Safari, navegará a la página de resultados de búsqueda de YouTube para Tylor Swift y mostrará los resultados.'"

```
    }

    # Clear history if the timeout has passed
    clear_history_if_timeout(character_essence[language])

    # Update the conversation history with the user message
    conversation_history.append({"role": "user", "content": f"{transcribed_text}"})

    # Call the ChatGPT API
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=conversation_history
    )

    # Extract the response text
    chatgpt_response = response.choices[0].message['content'].strip()
    print(f"ChatGPT Response: {chatgpt_response}")
```

```
    print(f"ChatGPT response: {chatgpt_response}")

    # Update the conversation history with the assistant's response
    conversation_history.append({"role": "assistant", "content": chatgpt_response})

    # Update the last interaction timestamp
    last_interaction_timestamp = time.time()

    return chatgpt_response

import re

def detect_and_execute_script(chatgpt_response, language):
    code = ""
    response_text = chatgpt_response

    if "**Shell**" in chatgpt_response:
        code, response_text = extract_code(chatgpt_response, "**Shell**")
        execute_shell_script(code)
    elif "**AppleScript**" in chatgpt_response:
        code, response_text = extract_code(chatgpt_response, "**AppleScript**")
        execute_apple_script(code)
    elif "**Python**" in chatgpt_response:
        code, response_text = extract_code(chatgpt_response, "**Python**")
        execute_python_script(code)
    elif "**Node.js**" in chatgpt_response:
        code, response_text = extract_code(chatgpt_response, "**Node.js**")
        execute_nodejs_script(code)

    read_response_aloud(response_text.strip(), language)

def extract_code(response, code_marker):
    code_start = response.find(code_marker) + len(code_marker) - 3
    code_start = response.find("```", code_start) + 4
    code_end = response.find("```", code_start)
    print("s:"+str(code_start)+"\ne:"+str(code_end))
    code = response[code_start:code_end].strip()
    response_text = response[code_end + 3:-1]

    return code, response_text


def execute_python_script(script):
    print("Python: "+script)
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".py") as script_file:
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["python", script_path])
    os.unlink(script_path)

def execute_apple_script(script):
    print("AppleScript: "+script)
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".aps") as
script_file:
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["osascript", script_path])
    os.unlink(script_path)

def execute_shell_script(script):
```

```
def execute_shell_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".sh") as script_file:
        script_file.write(script)
        script_path = script_file.name

    os.chmod(script_path, os.stat(script_path).st_mode | 0o111)
    subprocess.run([script_path])
    os.unlink(script_path)


def execute_nodejs_script(script):
    with tempfile.NamedTemporaryFile("w", delete=False, suffix=".js") as script_file:
        script_file.write(script)
        script_path = script_file.name

    subprocess.run(["node", script_path])
    os.unlink(script_path)

def read_response_aloud(response, language):
    language_codes = {
        "English": "en",
        "Polish": "pl",
        "Spanish": "es"
    }

    language_code = language_codes[language]
    tts = gTTS(response, lang= language_code)
    tts.save("response.mp3")

    # Initialize pygame mixer
    pygame.mixer.init()
    pygame.mixer.music.load("response.mp3")
    pygame.mixer.music.play()

    # Wait for the sound to finish playing
    while pygame.mixer.music.get_busy():
        pygame.time.Clock().tick(10)
    # Clean up
    os.remove("response.mp3")

def main():
    while True:
        # Listen for the assistant name and record the question or task
        question_audio_data, detected_language, name_to_remove =
listen_for_assistant()

        # Transcribe the recorded audio and call the ChatGPT API
        chatgpt_response = transcribe_and_call_chatgpt(question_audio_data,
detected_language, name_to_remove)

        # Detect and execute the script or read out loud the response
        detect_and_execute_script(chatgpt_response, detected_language)

if __name__ == "__main__":
    main()
```

You have created a great voice assistant using Google Speech Recognition, Google Translate, and OpenAI's ChatGPT! Here are some suggestions for improvements and additional features:

1. **Handle exceptions and errors**: Add more exception handling to manage situations when APIs or external services fail or return unexpected results. This will make the assistant more resilient to errors.
2. **Configuration file**: You can create a separate configuration file to store API keys and other sensitive information, which makes it easier to manage and update without modifying the main script.
3. **Support for more languages**: You can extend the assistant's capabilities by adding support for more languages. Simply update the `assistant_names`, `assistant_hey_sounds`, and `character_essence` dictionaries with the new languages and their respective translations.
4. **Introduce a cache mechanism**: To avoid redundant calls to the ChatGPT API for the same questions, you can implement a caching mechanism that stores recent questions and their responses.
5. **Add a user interface**: Consider creating a graphical user interface (GUI) to provide users with an alternative to voice interaction. This can include text input and output, as well as buttons for common tasks.
6. **Customize the assistant's voice**: You can use a Text-to-Speech (TTS) library with more customization options, such as pyttsx3, to change the assistant's voice, pitch, and speed.
7. **Voice wake word**: Implement a wake word feature so that the assistant starts listening only when a specific word or phrase is spoken. You can use a library like Porcupine for wake word detection.
8. **Modularize the code**: Refactor the code into smaller functions and classes, making it more modular and easier to maintain, extend, and test.

Please let me know if you have any questions or if you would like more information on any of these suggestions!