**Voice assistance documentation**

Spring 2024 IOT 263333

Yoav Gilad <Yoav\_ID>

Yagel Maimon 316611078

Adi Yoseph <ADI\_ID>

Guidance: Tom sofer, Yaniv

# Summary

This is a documentation of our IOT project(Voice Assistant).

As part of this project, we need to set up an ESP32 with a microphone that will record our commands and return answers through some speaker based on OpenAI llm model.

The steps we need to overcome are:

# Challenges:

1. [Setup the ESP32 board](#_Setup_the_ESP32) and upload Basic code to it (Blink)
2. [Set up A microphone](#_Setup_A_microphone) on the ESP32 Board and get the sound from the environment.
3. [Convert the sound](#_Covert_to_WAV) into a WAV file and save it.
4. [Hear the saved WAV](#_Ensure_the_WAV) file and ensure it sounds good.
5. [Train a classify model](#_Train_Wake-up_model) that we will use for our Wake-up word(side-work)
6. [Convert the record](#stt) with a speech-to-text tool into a Base64 variable.
7. [Create an OpenAI](#_Send_the_question) user and use the token to send requests and get answers.
8. [Turn the answer](#TextToSpeech) from text to speech again and play it on the speaker.
9. [Final packaging](#finalpacking)

# Documentation

## Setup the ESP32

We use the acknowledge bank to configure the ESP32. In the beginning, we used the Adi computer and it worked fine But with Yagel computer we struggled too much with it because the computer is an IOS operating system so we replaced it with Windows and it worked smoothly. The Blink code was uploaded perfectly as well and when we changed the delay variable the light blinked faster and slower depending on the value we sent to the delay function.

## Setup A microphone

Tom gave us a microphone component (INMP441) and with the help of the Bank we tested the microphone:  
<https://github.com/0015/ThatProject/blob/master/ESP32_MICROPHONE/ESP32_INMP441_SETUP_ESP-2.X/ESP32_INMP441_SETUP_ESP-2.X.ino>

**The main challenge:**

When we connected the microphone to the board it was received short every time.

**The reason that caused that and the solution we found**

the INMP441 component can't setup to the board we used because The shape of the component does not allow connection to the matrix that we are used to due to the fact that the matrix connects rows to the ESP while the component is built so that there is no connection in a line that does not pass between two inputs of the component itself, therefore this caused a short and the solution was to connect wires directly to the ESP instead of to the board that the ESP connects to.

## Convert to WAV

In the repository there is a branch Yoav created that should be good start to create Wav file and understand how to convert the microphone data into Wav file:

<https://github.com/adiyosef9387/IOT-Project---Voice-assistant/tree/platformIO2>

in this branch the main files are:

1. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/platformio.ini> - the only thing that we should take care of is to change *upload\_port* and *monitor\_port* to the port we connect the esp component.
2. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/src/main.ino> - the main loop that will execute the listening to the microphone.
3. <https://github.com/adiyosef9387/IOT-Project---Voice-assistant/blob/platformIO2/Scripts/record_audio.py> - after execute the *main.ino* file we should change *COM\_PORT* to the port that the ESP is connected to (like the ini file).

The flow should be like so:

1. Download the *PlatformIo* extension.
2. In the platformio window pick folder and pick the folder of the project.
3. Pip install ion
4. Pip install pyserial
5. Ctrl+p then platformio:upload
6. Run the python script *record\_audio.py*
7. After finish record interrupt in the terminal the python script running with ctl+c
8. The wav file will be created in the root folder

The main challenge:

We couldn’t find some organized documentation that helped us use the microphone data in WAV.

## Ensure the WAV sound

**The main challenge**

We couldn’t found a way to create this step because the Wav file we created saved on the SPIFFS storage and we couldn’t found solution to send this file to the computer (in this stage we still didn’t plugged the speaker so checking the voice that was record through the esp was not ready yet.

Solution:

We found way to interact with the SPIFFS storage and see the file name inside and the size of them, we saw the file was created and the size looks normal.

## Train Wake-up model

At first, we thought to use the trained model from the reference that was already trained for the word “go” but Tom told us that he expected us to train our own model.

**Main challenge:**

So we tried to figure out how to do that from the next repo:

<https://github.com/ICST-Technion/S22_VoiceControl/tree/main/SplitNetwork_ESP32>

we tried to start with this and struggled a little bit with it.

**The first attempt to train the model:**

Tom said that considering the circumstances we can use:

<https://docs.edgeimpulse.com/docs/edge-ai-hardware/mcu/seeed-xiao-esp32s3-sense>  
Which is an online service that trains split networks and supports specific microphones we got from Tom.

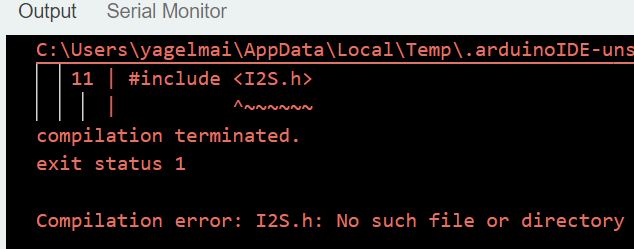
**The main challenge with the train model via online service**

We connected the microphone with the ESP board to the computer but when we got to the compile and uploaded the program to the ESP board we had to choose if our ESP32 version is 2.0.x or 3.0.x

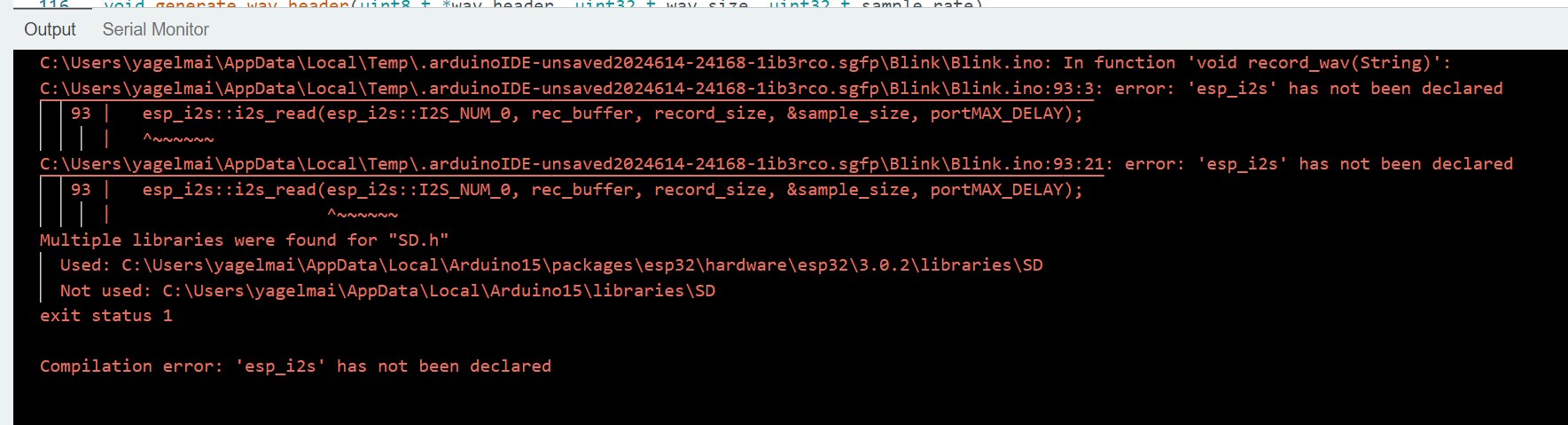
A screenshot of a computer

Description automatically generated

So as we configured at the esp32 setup it is 2.0.x so we tried to upload this code but got an error that the I2S.h file did not exist:



And when we tried to upload the 3.0.x code we got the esp\_i2s is not declared:



**Solution**:

we understood that the model is trained with some amount WAV files that are labels as “good” and some amount of WAV files that label as bad and the situation so In order to train the model we needed to have a lot of WAV files that contain records with some words.

We understood that the main gap is with recording ourselves because it take too much time and the records should stand in some specifics conditions (different people in variety places with variety conditions like with AC or not, the accents of the speaker etc..) so we understood that we need some workaround for this small step

We found this dataset:

<https://storage.cloud.google.com/download.tensorflow.org/data/speech_commands_v0.02.tar.gz>

that contain dozens of directories that are labeled by name, with each folder containing thousands of WAV files containing one-second and 1600 sample rate recordings. The awake word that was chosen is MARVIN since it is the word with the most recordings in the dataset, we trained the model based on this word and we have about 2,100 records with the fit conditions. We created a spectrogram(which is a way to present audio as an Image), this spectrogram converted into hex variables and saved inside the project as c file we upload this file t the esp code and any time esp alive it get record from the environment convert it to spectrogram and check how much is the same as the trained variable, if its 0.8 accurate and more the awake-word detected state is on.

## Convert record to text

**Storage challenge**

At first we thought to save the record as a Wav file inside the esp32 memory but we saw that any record of question take about 32KB so we can imagine that the answer should be about this size too so any QA should take more than 50KB.

So if the flash memory is about 4MB and we store there another things like the code and the library we can get fullstack overflow for short conversation.

**Solution:**

We didn’t saved the record as WAV file. Instead, we send the record as a stream to google speech to text service right away, then we used the answer from google as is.

## Send the question to OpenAI

The next task was to send the question as text to OpenAI llm model.

We used OpenAI documentation to find the right way for sending http request to OpenAI and get the answer.

At the beginning we ha a lot of errors, we understood we should create user and pay for it, then we found the right way to send the json to OpenAI.

## TextToSpeech

Now that we have the answer we wanted to have the ability to manage full conversation like we have in the real world! So we wanted the answer to play on the speaker.

At the beginning we thought to do it with as we did with text to speech google service but again the same issue (storage issue) came up, so we searched for workaround and we found this library:

esphome/ESP32-audioI2S

that fits to our I2S speaker and have the ability to turn text to speech (and for free!) The solution was that google translate API is free to use and text to speech isn’t so it sent to google translate the text but ask it to play the answer as speech. This solution helped us saving money and easy to use for sending stream.

One last thing is to adjust the voice to Marvin, because by default google translate is woman voice we search for a male voice it the google translate option, we found-out that the Italian voice sound more like male so we change the request to be as Italian voice (its not harm the answer in any way because the request we use is not translate but read the request we sent as is with google translate voice).

## Final packing

Now that the whole flow work as expected we ask Shira to help us with appropriate box for the project we brought her dimensions but we didn't think about the small things like

1. the position occupied by the hollow area inside the box that fits the screw
2. the ESP lamp is not visible
3. the microphone is a bit hidden and doesn't sound as good as before.

So we:

1. attached the board so that it would be narrower and fit into the box.
2. connected a new bulb to the ESP, drilled a new hole for the lamp to come out of and
3. widened the hole for the microphone.

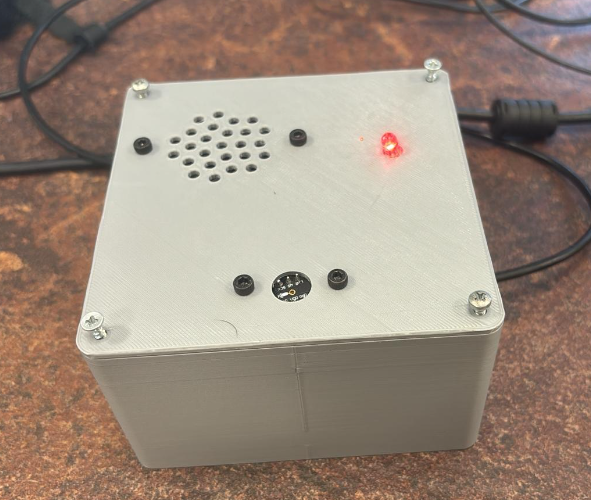
The microphone still not good as before but its doing much better job now!

We decided that for the user there is 3 main stages:

1. Green-light: Marving alive and waiting for wake-up word(you can wake him up by saying “Marvin”).



1. Red-light: Marving woke-up and load data (need to wait for a second).



1. Blue-light: Marving waiting for your question or answering to you right now.



Now the Project is fully ready!

# Thanks!

A big thanks must to be said to the IOT stuff aspecially to Yaniv our guidance, Tom the main moderator and Shira the assistant