

## Embedded Systems (ES) Lab - Final Project

B08901125 何卓耀 電機四 ([b08901125@ntu.edu.tw](mailto:b08901125@ntu.edu.tw))

B08901143 沈驚毅 電機四 ([b08901143@ntu.edu.tw](mailto:b08901143@ntu.edu.tw))

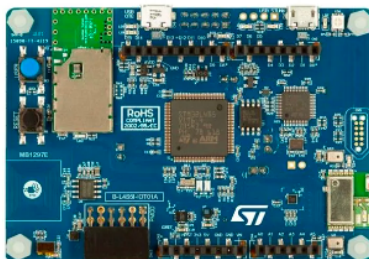
(Github URL: <https://github.com/dWKZ5jSZ8F/voice-lite> )

### Motivation

We are aware of the demand for privacy-aware voice assistants that may be achieved by offline inference, i.e., a Homepod-like artificial assistant. We may picture a household scenario where we strive to provide ease of access for responsive service (inquiry, call for aid etc.). These are often powered by designed systems, and our challenge is to at least implement the functions on a general-purpose embedded system, which gives us an advantage of being a low power device with offline input inference in light of aforementioned privacy concerns, as well as portability for ease of installation.

This may be powered by a lightweight embedded system architecture, which is what we proposed, **VoiceLite**, a lightweight voice recognition system powered by **Jetson Nano** and a STM32 development board **B-L475E-IOT01A**.

### Illustration of our concept



Perform **audio retrieval** and **Button Service**

Connects as **voice input**



Performs basic **speech to text** inference, and **notifies user** if specific keyword is spotted.

We leverage the built-in microphone and connectivity of the STM32 discovery board, and a lightweight ML inference device, Jetson Nano that connects via socket programming to achieve audio transfer through the microphone audio retrieval from STM32.

The Jetson Nano then processes a .wav file as input for its machine learning model Matchboxnet [2] to perform keyword spotting (KS), a typical speech recognition task that aims to detect the existence of pre-trained keywords from an audio input.

We set the detection threshold to be 0.8, in which a successfully detected keyword triggers the system to send an email notifying users of the detected command.

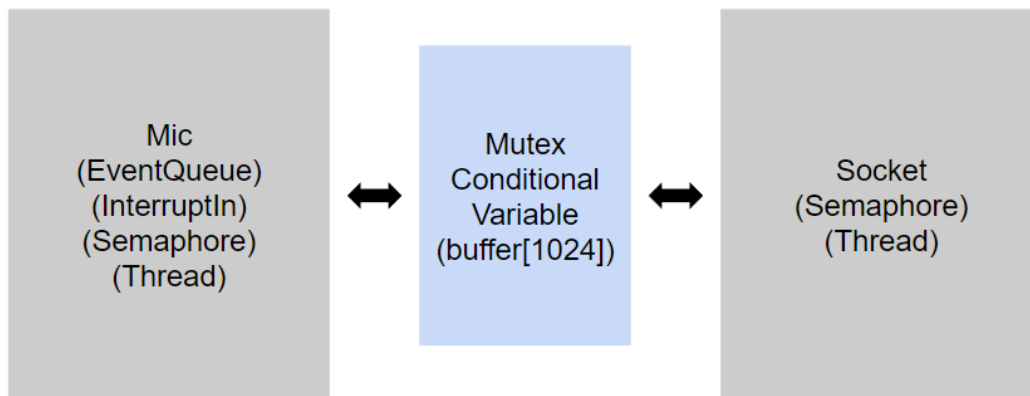
## Techniques, Implementations and Results

### STM32 Development board

We first set up a driver library for the audio sensors (MP34DT01) to retrieve audio signals with reference to a github repo [1]. After ensuring that the microphone module is well-functioning, we use the techniques of socket programming lectured in class and in order to achieve communication between the kits.

For ease in maintenance, we separate the functions into classes acting as two modules. In the class **mic**, we employ two main APIs which are *EventQueue* and *InterruptIn*, to build a button service with the USER button on the board. We create an event by calling the recording function and bound it to the event queue that is set to be dispatching without a timeout. While on the other hand, the **socket** classes dominate the data transfer of audio signals.

To maintain two functions working together in a parallel manner, we use different threads for each individual function. However, due to both modules sharing common resources, a character buffer that is used to contain data, *Mutex* and *ConditionalVariable* are used to synchronize the execution of different threads and secure the resources from contamination. In such a way, the socket would only wait until the buffer is contained with proper data and is notified by mic in order to transmit the data to the target receiving end.



For demonstration purposes, the figure below shows the process and end result of the flow. First, the audio and socket module are initialized in a parallel fashion. After the socket connection is established, we then could use the USER button to start recording for roughly about a 2 seconds span. The values in the buffer are also printed out in the terminal for visualization.

```

Hello from the B-L475E-ICT01A microphone (MP34DT01).
Audio Initialized.      (Audio Freq = 16000)
(Press the USER button to record a message.)

Socket Initialized.
10 networks available:
Network: NTU secured: None BSSID: 0:B:86:96:70:c2 RSSI: -80 Ch: 1
Network: eslab305-DL6 secured: WPA2 BSSID: A8:63:7D:c2:de:f1 RSSI: -57 Ch: 3
Network: ntu_peap secured: Unknown BSSID: 0:B:86:96:60:81 RSSI: -75 Ch: 1
Network: NTU secured: None BSSID: 0:B:86:96:60:82 RSSI: -75 Ch: 1
Network: eduroam secured: Unknown BSSID: 0:B:86:96:70:c0 RSSI: -76 Ch: 1
Network: ntu_peap secured: Unknown BSSID: 0:B:86:96:70:c1 RSSI: -76 Ch: 1
Network: eduroam secured: Unknown BSSID: 0:B:86:96:60:80 RSSI: -75 Ch: 1
Network: esys305 secured: WPA2 BSSID: D4:5D:64:e0:9f:88 RSSI: -61 Ch: 6
Network: secured: None BSSID: FA:8F:CA:8b:cd:dc RSSI: -55 Ch: 6
Network: WiFi secured: WPA2 BSSID: C2:E:59:41:3b:21 RSSI: -72 Ch: 6

Connecting to the network...
IP address: 192.168.50.32
Netmask: 255.255.255.0
Gateway: 192.168.50.1

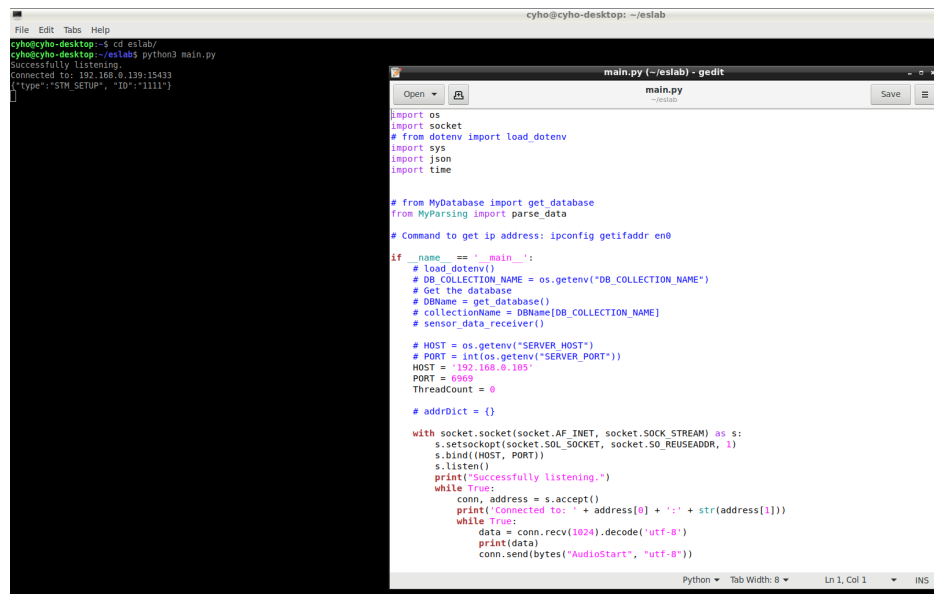
Resolving hostname 192.168.50.89
Address: 192.168.50.89
Opening connection to remote port 6969...

Socket Connected!
Start Recording...
Done recording.
Total complete events: 949, index is 28800
WAV file:
524946462ce1000057415645666d74201000000001000100803e0000007d0000020010006461746100e10000c2fa91fa4dfa43fa02fab4f94ef9ac853f83ef8cbf
7e0f746f857f8eef7a4f72ef81cf93af9a6f8f5f7bbf74ff81af93ef9c1f822f888f77bf780f7bef728f894f81ef958f9baf9fd9aef90cf96df829f8c6f8b4f95e
fbf2fa93fa35fa10fa97fa3bfb84f8e0fb54fcef8b12fbfef93af911f959f944f9c2f9df9b0f959f9a1f9b5fabfffb94fbf6fac2fafbfa74fb9cf8b5efb1bfb82fb2
6f970f915f920f83cf826f96ef941f9a4f816f8ecf76cf819f9c8f915faf2f9bdf9dcf933fa9ffa82faeaf9ccf97af91af9c3f81cf9a6f96cf9eef8c3f8f6f81bf9
c9f820f841f802f9b2f9d7f96df9e9f943fb2ef8b39fa48fa1bfbabfb34fcfefb03fb2ffabef91afa25f8e1fb35fbb3f987f896f7c7f7d3f855f92cf97bf9d4f9acf
aebfa0afb12fbdb5fabcfacdfa2bfb4fbb2fbb69fa2afa10faecf92efadddaf0fa7cfaeaf939f9b2f8daf893f91ffae6f9b6f978f904fa75fabafa04fbacfa85f942
f9fcf87ff836f8d9f774f7d4f7b8f86df99df9a3f999f975fad2fa6faf1fa25f6a6c57fb36fa54faebf92ff9e2f9cbfad7f938f9eef810f9fef811f9c3f809f9c
1fcf1fbb7fbbfbfb0fb38fcdcfb59fb7fa80faa0fa39fb58fb79faaff9b3f9f5f9f5f9c8f947fa2efac2f99af9a7f961f96ef9f1f9acf9aaf81df870f84df8daf7

```

## Jetson Nano

We configured the Jetson Nano to connect via Wi-Fi (or Ethernet when available), both for SSH connection and socket programming. Then, we initiated a socket for the connection between STM32 and Nano. This is leveraged to retrieve our raw string audio input.



```

cyho@cyho-desktop:~/eslab$ python3 main.py
Successfully listening.
Connected to: 192.168.0.139:15433
{"type": "STM_SETUP", "ID": "1111"}

main.py (~/.eslab) - gedit
main.py
import os
import socket
# from dotenv import load_dotenv
import sys
import json
import time

# from MyDatabase import get_database
from MyParsing import parse_data

# Command to get ip address: ipconfig getifaddr en0

if __name__ == '__main__':
    # load_dotenv()
    # DB_COLLECTION_NAME = os.getenv("DB_COLLECTION_NAME")
    # Get the database
    # dbName = get_database()
    # collectionName = dbName[DB_COLLECTION_NAME]
    # sensor_data_receiver()

    # HOST = os.getenv("SERVER_HOST")
    # PORT = int(os.getenv("SERVER_PORT"))
    HOST = '192.168.0.105'
    PORT = 6969
    ThreadCount = 0

    # addDict = {}

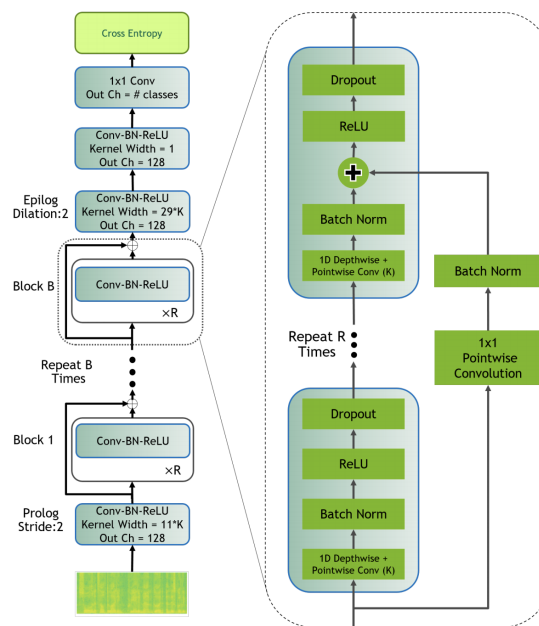
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        s.bind((HOST, PORT))
        s.listen()
        print("Successfully listening.")
        while True:
            conn, address = s.accept()
            print('Connected to: ' + address[0] + ':' + str(address[1]))
            while True:
                data = conn.recv(1024).decode('utf-8')
                print(data)
                conn.send(bytes("AudioStart", "utf-8"))

```

Once a connection is successful, we may retrieve the raw string representation of our audio signal from STM32, and convert it via adding a header onto the beginning of our raw string (see the figure below for the decomposition of 44-byte WAV header), and subsequently convert the raw string by encoding the HEX format raw string to Bytes. See the live retrieval in our Demo.

Name	offset	Size	Value
ChunkID	0	4	"RIFF"
ChunkSize	4	4	405040
Format	8	4	"WAVE"
Subchunk1 ID	12	4	"fmt"
Subchunk1 Size	16	4	16
Audio Format	20	2	1
Num Channels	22	2	2
Sample Rate	24	4	22050
Byte Rate	28	4	88200
Block Align	32	2	4
Bits per Sample	34	2	16
Subchunk2 ID	36	4	"data"
Subchunk 2 Size	40	4	405004

Once the .wav file is post-processed from socket input (pre-processing from the perspective of downstream inference), it can be fed into our machine learning model to perform speech recognition. The model, Matchboxnet [2] is an end-to-end lightweight neural network for speech command recognition (keyword spotting, KS) based on QuartzNet (see the network architecture below). It was trained on 12 keywords from the Google Speech Commands dataset to realize a lightweight keyword spotting model. We use these scenarios ('go' in demo) as our proof of concept. We may perform Automatic Speech Recognition (ASR) using QuartzNet to validate the efficacy of our retrieved audio quality by its end-to-end performance.

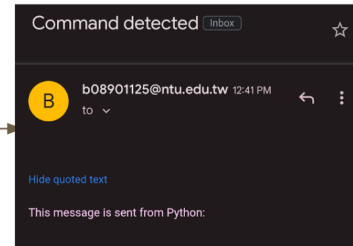


A successful detection triggers an email (demonstrated by using NTU email\*, because Gmail no longer supports Python smtplib [4] method), sending mail to a pre-assigned 'owner mail' (here it is our g.ntu account).

*\* our GitHub codes will not include a readily demonstrable mail function as credentials are left blank.*

```
profiles: {}
[2022-12-23 12:41:06] audio.py:82 - Loading audio 'data/audio/command_go.wav'
Keyword go detected with threshold value: 1.000
Executing response...
res=> (250, b'EX2016-CAS-04.ntu.internal Hello [140.112.174.139]\nSIZE 37740736\nPIPELINING\n8BITMIME\nEN
start tls => (220, b'2.0.0 SMTP server ready')
SMTP login => (235, b'2.7.0 Authentication successful')
Mail sent.

audio stream closed.
[TensorRT] INFO: [MemUsageChange] Init cuBLAS/cuBLASLt: CPU +0, GPU +0, now: CPU 1520, GPU 1941 (MiB)
root@cyho-desktop:/jetson-voices
```



## **Discussions: Issues and Problems Encountered**

### **Incompatibility across versions of development kit**

Originally, we were using a B-L4S5I-IOT01A board. Since there are no driver codes for the audio module in the original BSP library, we then came across a MP34DT01 library dedicated to the B-L475E-IOT01A board. We spent a lot of time trying to debug and apply on the original board, however it turned out to be some missing of fundamental definitions that led to incompatibility. Thus, the last resort was to change our board to B-L475E-IOT01A.

### **Memory Insufficiency**

Throughout the course of multi-threading, we had suffered from the issues of insufficient memory. In order to have both the modules to work parallel, the allocated buffer for audio signals input would be cut down to around 2 seconds. In addition, the target buffer for data transfer would also be limited.

### **Huge overhead for per-wav-file inference**

We script our model to load for each use, which takes around 30 seconds to initialize the inference for each wav file. In fact, we may directly use pyaudio to fetch wav signals into the model through socket programming, which makes STM32 act as a live microphone to mitigate. This cuts down the overhead of initializing our model to readily infer audio inputs, compared to that of doing each wav file separately.

## **Reference**

- [1] MP34DT01 driver library: <https://github.com/janjongboom/b-l475e-iot01a-audio-mbed>
- [2] Matchboxnet: <https://arxiv.org/pdf/1703.05390.pdf>
- [3] Jetson-voice (GitHub): <https://github.com/dusty-nv/jetson-voice>
- [4] Smtplib: <https://docs.python.org/3/library/smtplib.html>