Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

LungDetect

Submitted to

Andrey Vyshedskiy
617-817-1916
vysha@bu.edu

by

Team #21

Team Members

Thinh Nguyen tnguy19@bu.edu
Matthew Pipko mattpi23@bu.edu
Hilario Gonzalez hilario@bu.edu
Astrid Mihalopoulos astridm@bu.edu
Shadin Almaiman shadin@bu.edu

Submitted: 4/18/2025

LungDetect User Manual

Table of Contents

Executive Summary 2
1 Introduction 4
2 System Overview and Installation 6
2.1 Overview block diagram 6
2.2 User interface. 6
2.3 Physical Description 7
2.4 Installation, setup, and support 7
3 Operation of the Project 10
3.1 Operating Mode 1: Normal Operation 10
3.2 Operating Mode 2: Abnormal Operations 12
3.3 Safety Issues 13
4 Technical Background 14
5 Relevant Engineering Standards 18
6 Cost Breakdown 20
7 Appendices 21
7.1 Appendix A - Specifications 21
7.2 Appendix B – Team Information 22

Executive Summary

Hilario Gonzalez

Acoustic Imaging Device for Diagnosing Pneumonia
Team Number: 21 – Team Name: LungDetect

The current methods for diagnosing pneumonia, such as stethoscopes and X-rays, have significant limitations. Stethoscopes lack accuracy and visualization capabilities, which can lead to potential misdiagnoses. X-rays, while effective in confirming diagnoses, use radiation, which poses risks to vulnerable groups like pregnant women and children. There is a need for an accurate, cost-efficient, and non-invasive method for diagnosing pneumonia.

The LungDetect project will deliver an acoustic imaging device and software application that uses multiple microphones to detect abnormal lung sounds, specifically "crackles," indicative of pneumonia. The final hardware deliverable will be a foam pad housing 6 microphones, protected with an acrylic enclosure and regularly disinfected by protecting the points of skin-to-microphone contact. These microphones will capture lung sounds and transmit the data to a connected laptop via USB-C. This simultaneous recording of the microphones is made possible by transmitting analogue data through a USB-C splitter. The final software deliverable will be a user-friendly application that will process the audio, use cross-correlation to localize the crackles, and visualize the findings.

The proposed technical approach includes a device that will use piezoelectric contact microphones embedded in a foam pad to record lung sounds. The data from each microphone channel will be recorded simultaneously and edited using the audio recording software n-Track. The recorded audio is adjusted for gain and has both high-pass and low-pass filters applied to each microphone to limit the range of recorded frequencies. Then, it is downloaded as a .wav file for analysis. Concerning the technical approach for the software, there are four main components: a React front-end for user interaction and visualization, a Node.js back-end to manage data flow, a Python script to analyze the audio data, identify crackles, and perform localization of the crackles, and a calibration script that evaluates the synchronization delay between every channel and adjusts the audio processing accordingly.

One of the innovations of this project is how the data analysis is handled. The Python script identifies "crackle families," grouping sound spikes based on temporal proximity across channels. A pairwise cross-correlation is executed across a small slice of time on every channel for every crackle. The "mother crackle" is identified as the spike with the highest transmission, and every other “daughter crackle” yields a measure of delay and

transmission relative to this mother. This approach helps in analyzing the propagation of crackles and provides more detailed information about the lung condition.

1 Introduction

Hilario Gonzalez

Acoustic Imaging Device for Diagnosing Pneumonia
Team Number: 21 – Team Name: LungDetect

One symptom of a potentially serious lung condition is the presence of abnormal lung sounds in a patient's lungs, ranging from rhonchi (continuous, low-pitched sound) to wheezing or crackles (clicking-like sound).

The most common method of detecting these sounds is using a stethoscope. This method is called auscultation. However, with an accuracy range of 53.4% to 80.1%, depending on the experience of the physician, this procedure is not precise. This means that around 20% to 50% of patients with a serious lung condition might exhibit symptoms that are misdiagnosed as benign and vice versa.

Further, physicians currently face issues when it comes to visualizing any sounds that they detect, instead relying on more advanced technologies that are more costly and consequently negatively impact the patient's finances and time.

While there are more accurate methods that aid in visualizing the issues, such as CT-scan and X-ray, the utilization of penetrating radiation in these methods poses a potential threat to sensitive groups such as pregnant women, the elderly, or small children. This potential risk suggests room for improvement in detecting lung issues using another method with less implication.

Therefore, this project aims to develop a low-cost, non-invasive acoustic imaging device for diagnosing pneumonia. The proposed device design from our client was a multi-channel audio recorder assembled using piezo disc transducer contact microphones in a foam enclosure, interfacing with a computer over USB-C. This design was based on our client's initial research regarding the viability of the device in diagnosing pneumonia and other respiratory diseases.

The device utilizes 6 piezoelectric contact microphones to listen to a patient's thoracic cavity. The analog data from the microphones is transmitted over standard 3.5mm auxiliary input cables to a USB-C splitter, which collapses the six channels into a single USB-C input for a laptop. As requested by our client, we use a software called N-Track to record from our device. n-Track facilitates a few key requirements in our data pipeline; it provides the freedom to map the microphone channels to the correct channels in the output file, it digitizes our audio recording, and it allows us to apply filters ahead of

recording. Lastly, our custom-made application receives an uploaded .wav file and executes an analysis of the file that detects, indexes, and localizes abnormal lung sounds and applies a calibrated synchronization across the channels. It then outputs the result in a result screen consisting of a lung diagram, data tables, and audio playback functionality. The application is broadcast over the local network, so anyone with a web browser and a connection to the local network can use it.

This technology has the potential to offer a safer and more cost-effective alternative to traditional pneumonia diagnosis methods, such as X-rays, which can be harmful to children and pregnant women. The device also has the potential to be used for non-clinical applications, such as a device that can remotely monitor the breathing of hospice patients.

This system is quick to install and fairly easy to use, allowing for its use during more time-sensitive diagnostic procedures, such as in emergency and urgent care situations.

2 System Overview and Installation

2.1 Overview block diagram

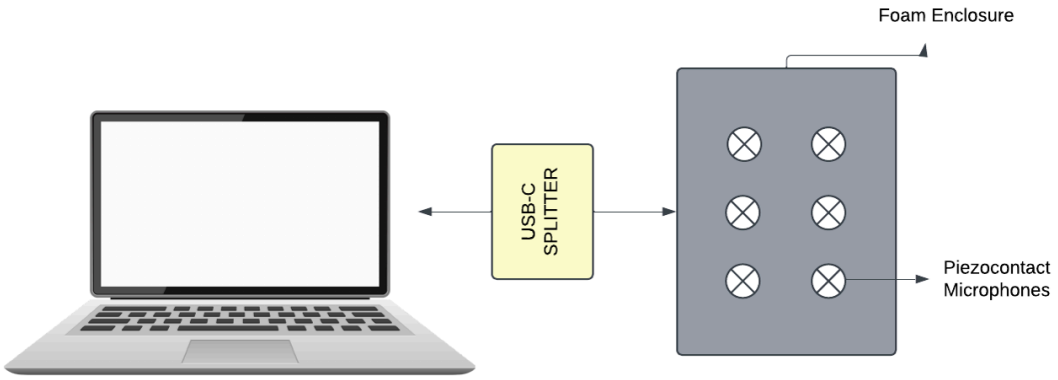

Figure 1: Diagram showing the dataflow of the hardware setup.

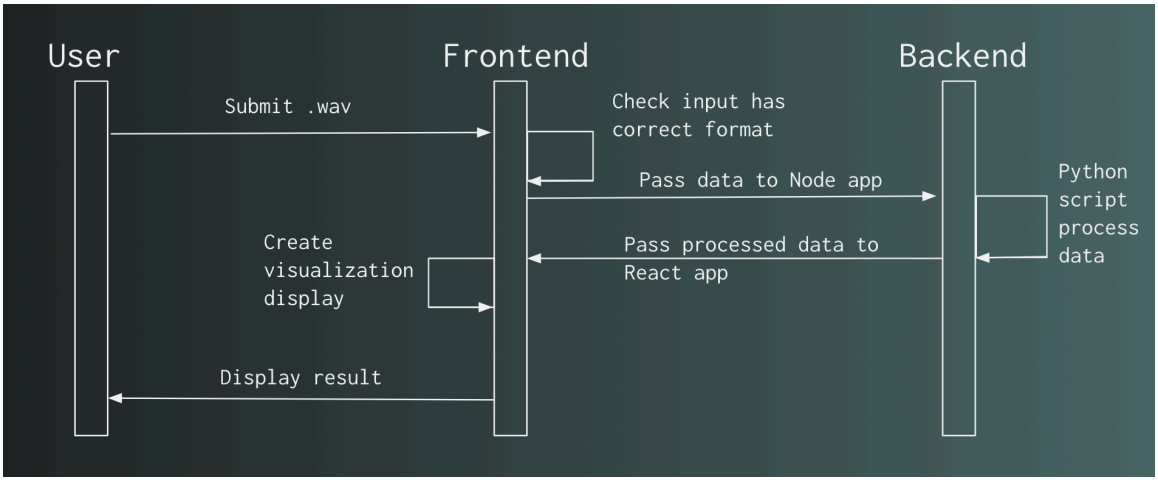

Figure 2: Diagram showing the dataflow for the software app consisting of the frontend and backend, indicated with arrows showing where data is passed along

2.2 User interface:

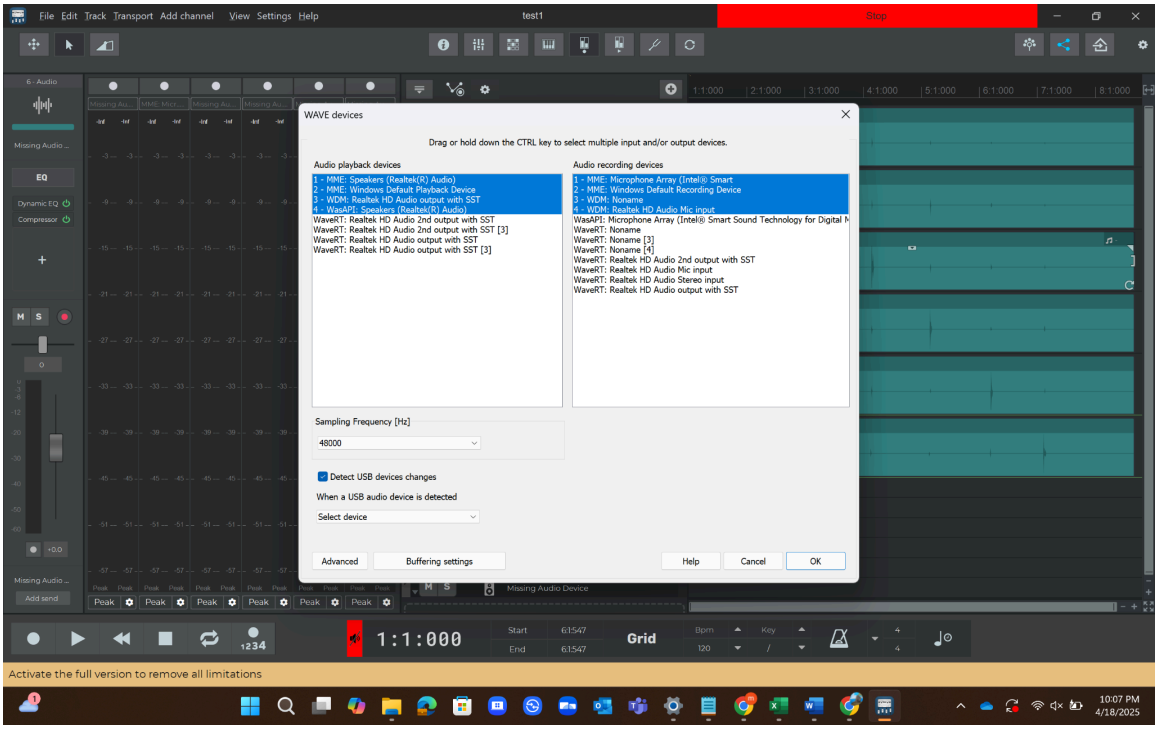

Figure 3: Screenshot of n-Track Studio 10 and its "Audio Devices" settings. To select multiple inputs and outputs, hold "CTRL" while selecting the microphones registered by the USB hub in order.

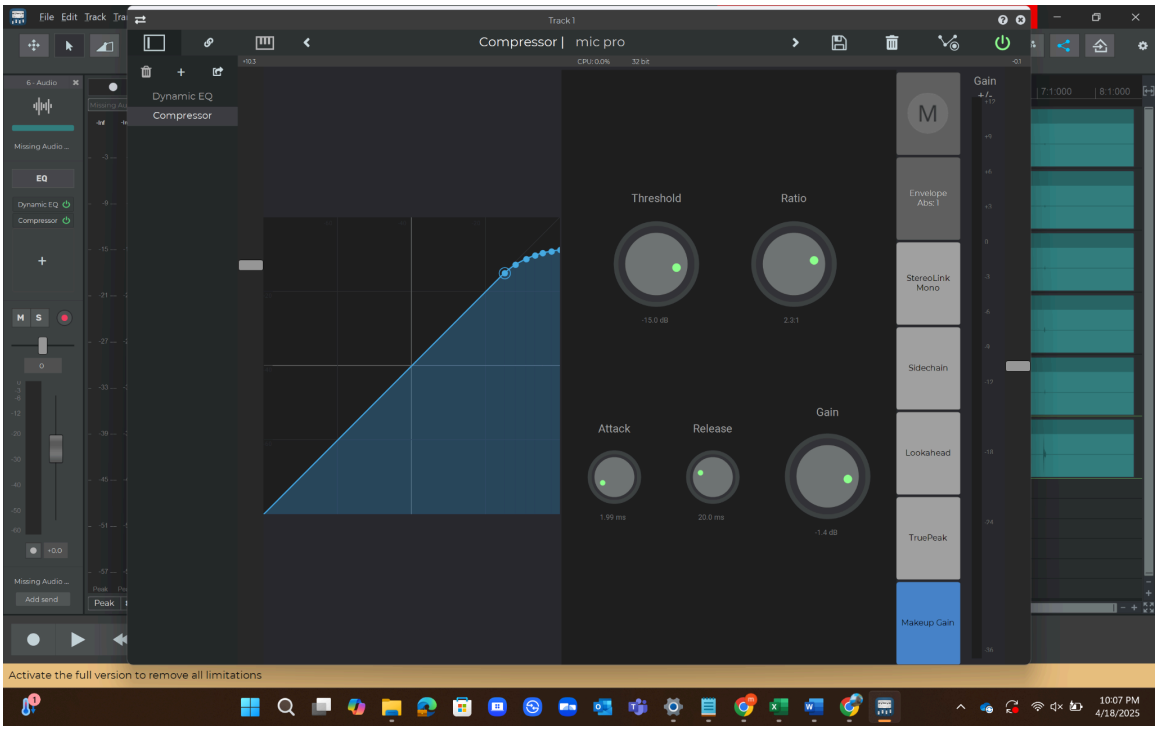

Figure 4: Screenshot of n-Track Studio 10 and its compressor settings. Customized settings can be saved as a preset for easy loading.

Input gain: +10 dB, threshold: -15 dB, ratio: 2.3:1, attack: 2 ms, release: 20 ms, signal gain: -1.4 dB. To apply post-processing effects, click on the "+" on the left side under the track information.

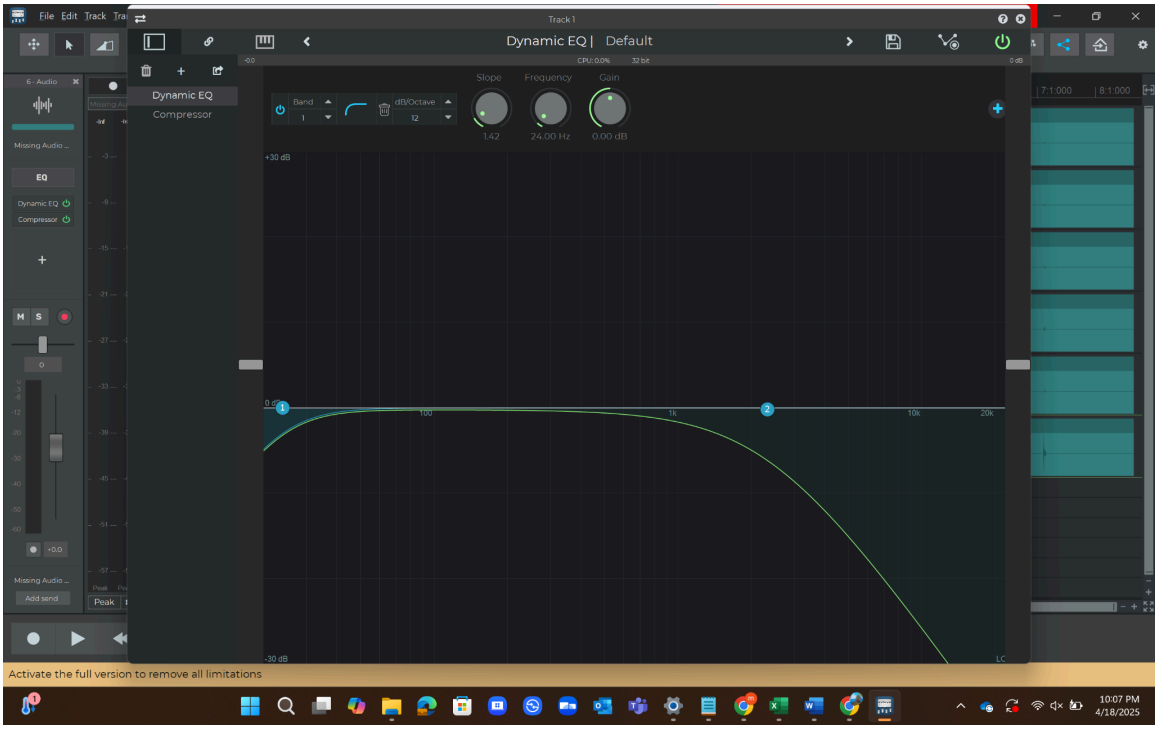

Figure *5: Screenshot of n-Track Studio 10 and its dynamic equalization (EQ) settings. Customized settings can be saved as a preset for easy loading. High-pass filter at 24 Hz, slope = 1.42, gain = 0 dB, 12 dB/octave Low-pass filter at 2300 Hz, slope = 1.00, gain = 0 dB, 12 dB/octave To apply post-processing effects, click on the "+" on the left side under the track information.*

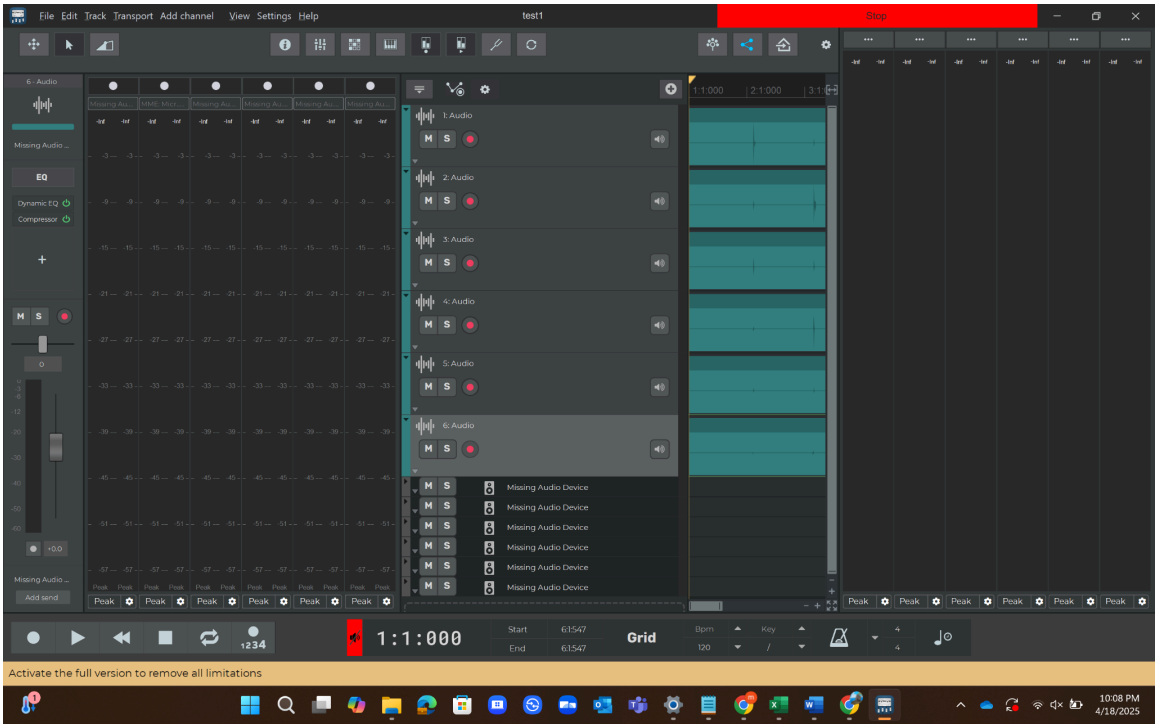

Figure 6: Screenshot of n-Track Studio 10 and its input volume (left side) and output volume (right side) monitors. Use the input volume monitors for testing microphone recording capabilities, and use the output volume monitors for testing audio signal playback.

2.3 Physical Description

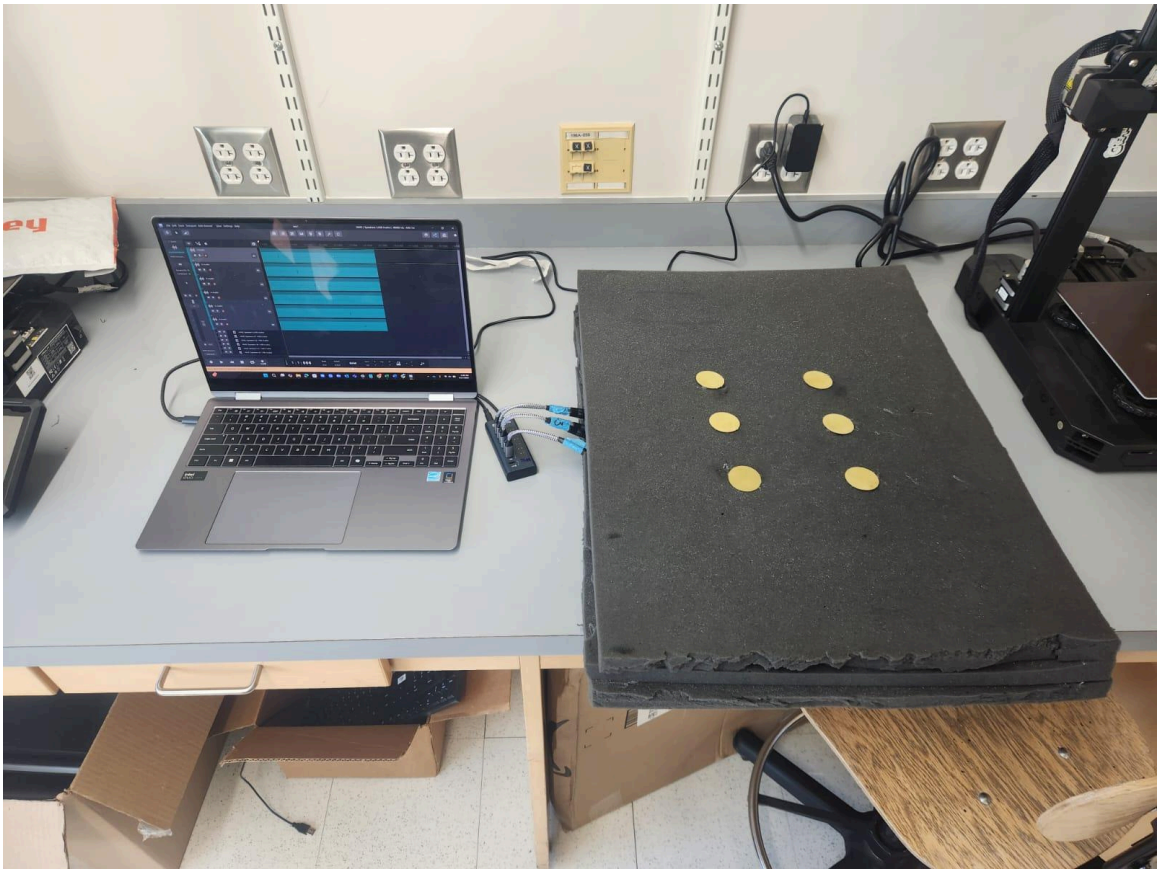

Figure 7: Photograph of the device active and ready to record on n-Track.

2.4 Installation, setup, and support

2.4.1 Hardware installation:

Matthew Pipko and Shadin Almaiman

First, connect the microphones from the foam to the USB splitter to be inserted into the computer.

Making sure all the connections are secure and correct, open up n-Track's "Audio Device" settings and make sure to select a channel and corresponding speaker for each microphone.

After checking for the correct microphone settings, some additional audio adjustments must be made before we can begin recording patient breathing.

n-Track set up:
If these post-processing settings have not been set and saved before, then they need to be applied to every track (see figures 4 and 5 for details):

- High-pass filter applied at 24 Hz with no gain.
- Low-pass filter applied at 2300 Hz with no gain.

- Compression with an input gain of +10 dB, a threshold of -15.0 dB, a ratio of 2.3:1, an attack of 2 ms, a release of 20 ms, and a signal gain amplification of -1.4 dB.

See figures 4 and 5 for the visual demonstration of what the effects and settings look like.

2.4.2 Software installation:

Thinh Nguyen

NOTE: The following installation guide is written for macOS; please refer to the online documentation for equivalent commands if you are using Windows or Linux. Commands meant for command lines are written in italics between apostrophes, i.e., *'command line commands'*

Please ensure that you have the following prerequisites installed:

- Node.js (v14+). Installation instructions: https://nodejs.org/en/download
- Python (3.6+). Installation instructions: https://www.python.org/downloads/
- Pip (if needed). Installation instructions: https://pip.pypa.io/en/stable/installation/

To install the project source code is hosted on GitHub which can be downloaded to the local machine through the following steps.

1. Fork the GitHub repo at the link to your GitHub:
   https://github.com/tnguy19/lung-detect-web
2. Clone the repository on your local machine:
   '*git clone https://github.com/{YOUR GITHUB USERNAME }/lung-detect.git*
   cd lung-detect'
3. At the root level of the project folder, install Node.js dependencies:
   ' *npm install*'
4. Install Python dependencies:
   '*pip install librosa numpy scipy soundfile matplotlib*'
5. Create uploads directory if it doesn't exist:
   '*mkdir -p public/data/uploads*'
6. Install dependencies:
   '*npm install*'
7. Navigate to the 'backend' directory in the project folder to download the Python dependencies:
   'pip install librosa numpy scipy soundfile matplotlib'

After the installation is complete, you can start the application with the following:

1. Build the (frontend) application:
   'npm run build'
2. Build the (frontend) application:
   'npm install -g serve
   serve -s build -l 3000'
3. Open a new terminal and navigate to the 'backend' directory in the project folder. Start the backend Node server:
   'node server'

To end the hosting of both frontend and backend, simply navigate to the terminal that you used to host either and press 'Control+C' on your keyboard to terminate the server.

2.4.3 Accessing the application:

After the application is correctly installed and set up, you can access it by going to your preferred web browser and typing either the following address in the search bar:

- Locally: `*http://localhost:3000*`
- From all devices on the network: `http://{YOUR_IP_ADDRESS}:3000`

To find your machine's local IP address, please type the following command on your command line:

- On macOS/Linux: `ifconfig` or `ip addr`
- On Windows: `ipconfig`

Another option would be to look at the output displayed on the terminal when starting your frontend or backend server. Input that value in the {YOUR_IP_ADDRESS} section of the link above. This will also be the link that can be used by every machine on your local network to access the application, provided that there is one machine hosting the application. Please also make sure your firewall allows incoming connections on ports 3000 (frontend) and 5000 (backend).

3 Operation of the Project

3.1 Operating Mode 1: Normal Operation

Matthew Pipko, Shadin Almaiman, Thinh Nguyen, Hilario Gonzalez

3.1.1a Hardware Normal Operations:

- Ideally, the device should be able to record audio from six different microphones simultaneously
- Each microphone can be assigned to its track for data separation
- Audio settings like high-pass filters, noise reduction, gain, etc., should be easily adjustable pre-recording and be reflected in the final file

3.1.1b n-Track Normal operations

- Check the "Audio Devices" settings to ensure that all 6 microphones are connected and plugged in correctly.
- Ensure that each track is assigned to its corresponding microphone in order (channels 0-5 correspond to tracks 1-6).
- If n-Track is opened for the first time, apply the post-processing effects that are shown and described in section 2.2 User Interface and 2.4 Installation, Setup, and Support.
- After successfully connecting and applying the effects to each microphone, open the input volume and output volume meters to monitor how the microphones record, and how the audio is processed after recording respectively.
- When the user is finished recording the audio, they can export each track as an individual .wav file for signal analysis.

3.1.2 Software Normal Operations:

- The software of our custom-made app, when initialized and run properly, will not need to be manually adjusted or calibrated, as it is designed to be intuitive to use without needing a dedicated user guide
- Users can choose to upload a single multi-channel .wal file or multiple single .wav channel files. For the multiple .wav files option, the UI will indicate exactly the order of the files that need to be uploaded, concerning the location of the microphones where the sound was picked up

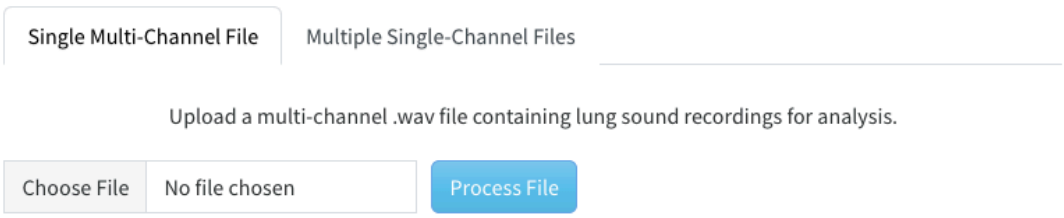

Figure 8: Tab to upload a single multi-channel .wav file on the UI

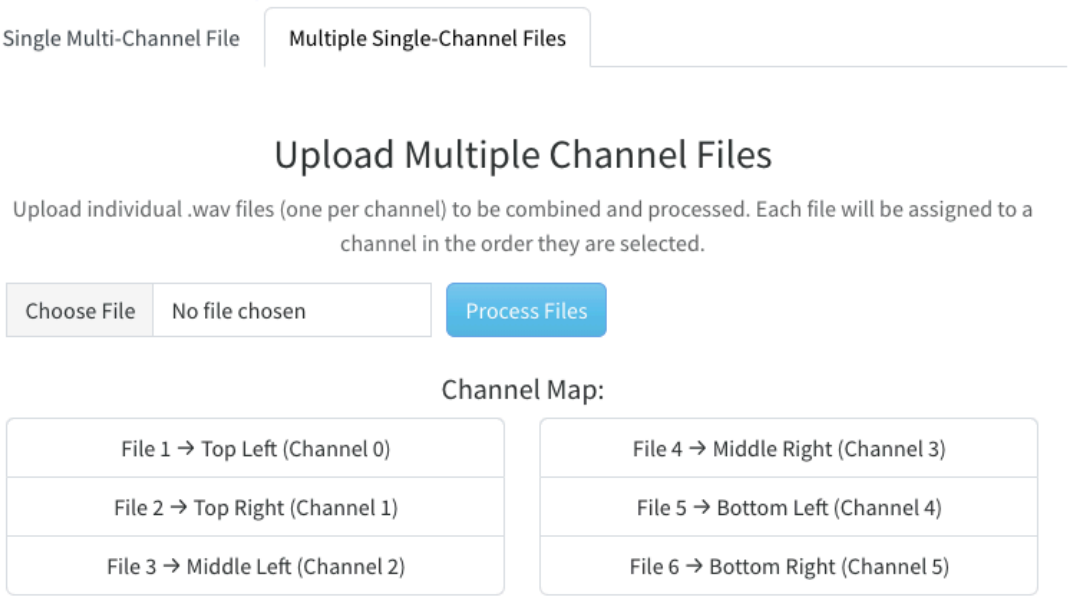

Figure 9: Tab to upload multiple single, multi-channel .wav files on the UI

- No manual input or further message will display if appropriate .wav file(s) are uploaded; there will be a progress bar that will automatically appear on the UI that will indicate the progress. However, even if the bar appears full (indicating 100% progress), it could sometimes take an extra minute or so for the result screen to appear, depending on how quickly your machine can run the program.
- Software results will not be saved after each refresh iteration, meaning that if you refresh the website or press the back button on the result screen, the app will return to the default state where there have not been any files that were uploaded.
- The app is built to output command logs and details of the data processing activity that is happening under the hood. You can view this through the terminal that you used to host the frontend and backend. Information includes whether the app detects a file, the type of the file uploaded, the data that it is receiving, and the format that the app is converting the raw data into is being output in a stream of information on the terminal. This will not be available directly on the UI as it is not a core component and would be too complicated/not needed for most users to know how to use the app.
- The app will output errors on the UI if there are any issues encountered, particularly in simple errors such as failure to set up and run the backend when a file is uploaded, etc.
- In the multiple file upload options, the app will accept and handle from 1 to 6 input files without issues. For example, in the case of 4 files being uploaded, it will display the result for 4 microphones, and so on.

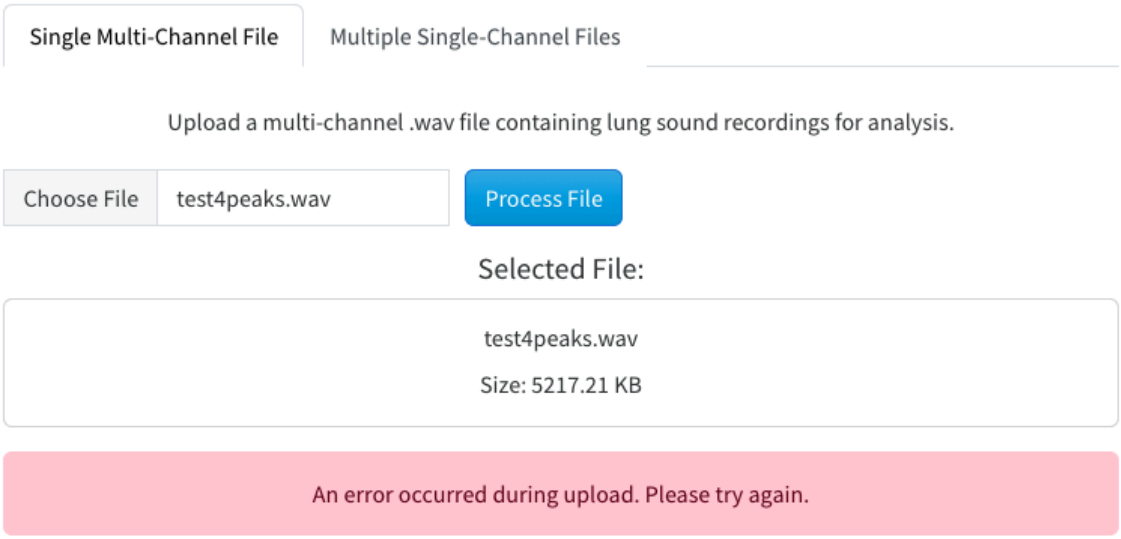

Figure 10: General error message when an error occurs

3.2 Operating Mode 2: Abnormal Operations

Matthew Pipko, Shadin Almaiman, Thinh Nguyen, Hilario Gonzalez

3.2.1: Hardware Abnormal Operations:

- At times, there might be an instance where audio from a specific microphone isn't being picked up. When this occurs, check connections for any loose cables or for a faulty splitter. If all is correct and secure, then ensure all the correct inputs are mapped in n-Track.
- If n-Track crashes or is unresponsive, reload the session with the same test file. All settings should stay the same. If not, refer to the previous setup documentation to repeat all settings.
- If input levels are low, increase gain and/or check hardware for damage.

3.2.2 Software Abnormal Operations:

- Occasionally, there will be issues when setting up the application, especially when it comes to installing dependencies, i.e,. Node.js, React.js, etc. Please refer to the documentation for these libraries on their respective websites if encountering any issues, as the causes may vary, and depending on the machine or versions of the dependencies, this may cause unexpected behavior.
- The app has built-in features that will reject invalid output, which will trigger error messages on the UI, indicating to the user the issue. Invalid files will not be accepted by the app, and the user will have to check the format of the .wav files and how the file is collected.
- The app may also run into problems if the user runs the frontend or backend app on more than one port at the same time, i.e., if you accidentally have your

frontend on both port 3000 and 3001 but only have your backend on 5000, this may lead to an error when using the app.

- Since the app is hosted on the local network in the form of a webpage, frequent accessing of the app can have the frontend cached on the local network. This may lead to unexpected behavior in certain cases if the user wants to change the code while hosting the app.
- The backend scripts are designed to work for multi-channel data, and in the case that a single channel file is submitted, the software catches this to prevent unexpected crashes in the backend. The script has robust error-checking and logging to assist in diagnostics.

3.3 Safety Issues

Shadin Almaiman

There aren't many safety issues associated with this project, especially since it is a non-invasive diagnostic device. However, our biggest concerns regarding the device have to do with contamination and patient data safety.

First, since the device is used within a medical environment and requires skin-to-microphone contact, it is important to ensure that it does not become the cause of the spread of infection. To mitigate this, the external surfaces of the device should be disinfected with an electronic-safe disinfectant. Additionally, a disposable plastic covering would ideally be placed over the device between patients. This feature is yet to be implemented, but it would most likely be done by a material such as medical-grade PVC. This material is transparent, waterproof, and resistant to common medical disinfectants. It is also able to be infused with microbial agents to further prevent the spread of infection. Most importantly, the plastic covering should be thick enough to be durable, but thin enough not to interfere with any audio recording.

As for patient data safety, ensuring privacy is essential to comply with important medical regulatory laws such as HIPAA. To ensure patient privacy, all audio recordings should be encrypted during transmission and storage, with cloud services that comply with relevant data protection laws. Additionally, access to patient data must be restricted to authorized users only, and strong authentication methods should be employed.

4 Technical Background

4.1 Hardware Technical Overview and Background:

Shadin Almaiman

The hardware portion of the project is made up of multiple components. For one, the device includes six piezoelectric contact microphones for simultaneous recording. By doing this, the system can acquire a comprehensive look at the patient's sound patterns.

As for the device's enclosure, it is made up of polyethylene foam due to its durability, resistance to water, and ability to maintain cleanliness (preventing mold, contamination, etc). This foam acts as the device's housing material, embedding the microphones and preventing their displacement during the diagnostic process.

Data from all six microphones is combined into one, singular USB-C port by the USB-C splitter, streamlining connection and data transmission to the laptop for data analysis. Additionally, the device is powered by external power. This makes sure that the device is uninterrupted during its use.

Once the audio file is recorded and exported as a .wav file by n-Track, it is uploaded to our custom-made software app.

4.2 Software Technical Overview and Background:

Hilario Gonzalez and Thinh Nguyen

The software portion of our device app consists of a custom-made web app with React frontend, Node.js backend, and Python processing scripts, which contain the necessary logic for data processing.

a. React frontend and Node.js Backend

UI and front improvement are handled with Bootstrap as the framework and Bootswatch, more specifically, as the library style being applied. The React app is built using components, where each component corresponds to an item in the app. They consist of:

- **App.js:** main app, consists of the logic to switch between 'upload' and 'result'. Also contains buttons responsible for navigation between the screens and a navigation bar

Upload Screen:

- **UploadContainer:** app section containing the tabs to navigate between upload options and button, responsible for initiating the compute command to the backend when there is a single multi-channel file being uploaded
- **MultiuploadContainer:** nested inside UploadContainer, responsible for initiating the compute command to the backend when multiple single-channel files are being uploaded

Result Screen:

- **DataTable:** handles the display of results in a data table format

- **LungVisualization:** handles the display of the lung diagram with buttons corresponding to the location of microphones. These displays are sorted in crackle families that can be selected via a toggle. Microphones that are closest to the sound appear red with the caption 'Sound,' and others appear green with the caption 'No Sound'. Integrated with the AudioPlayback and AudioService components, responsible for audio playback when one of the mentioned buttons is clicked

Utility:

- **AudioPlayback:** component that provides user controls for audio playback, uses the AudioService for the actual audio processing
- AudioService: service class that handles the audio processing using the Web Audio API

Other components:

- **NavBar:** app section that functions as a navigation bar, switches between the main screen, the features screen, and the about screen
- **Feature:** screen to display the app functionalities and information
- **About:** screen to display the information of the group members
- **ThemeSwitcher:** component to handle the changing of UI look between the options available in the Bootswatch library

The backend Node.js server provides a REST API with endpoints for:

- Uploading and processing a multichannel .wav file (/compute)
- Uploading processing multiple single-channel .wav files(/compute-multi)
- Health checking (/health)

All upload file handlers are done with multer (Node.js middle installed via npm). Uploaded files are stored in the local project folder directory in 'backend/public/data/uploads'

b. Python scripts:

The application, particularly the Node.js backend, relies on Python scripts to perform data analysis and processing. These include:

combine_wav.script (processing script for multiple .wav files upload): Python script to combine multiple files into one. Each input file is loaded using librosa with its native sample rate. Script then automatically resamples all audio files to match the highest sample rate. Then the length of the longest file is either trimmed, or short files are padded with zeros to ensure identical lengths across channels. The script then uses NumPy's vstack operation to combine the individual mono channels into a multidimensional array.

This data is transposed to match the expected format for computing before calling the audio_process.py script on this newly combined file.

audio_process.py (audio processing script): The main script performs multichannel crackle-event analysis. It loads raw audio, reshapes it into a matrix of channels × samples, and for each channel builds a local noise threshold using a rolling median plus MAD; samples below the threshold are nulled and scipy.signal.find_peaks marks spike candidates. Detected spikes from all channels are time-sorted and temporally clustered; clusters that do not appear on a sufficient fraction of channels are discarded, and for each surviving cluster, only the highest-amplitude spike per channel is kept. Around each retained spike, the code extracts a fixed-length waveform slice, computes pair-wise cross-correlations, and chooses as "leader" the channel whose slice yields the largest summed cross-correlation peaks. For every other channel, the script sets the raw delay to the simple spike-time difference relative to this leader and defines a transmission coefficient as the ratio of the pair-wise cross-correlation peak to the leader's autocorrelation peak. The companion calibrate() routine returns per-channel timing offsets—deltas—obtained from an external calibration test. In the final reporting stage, the script zeroes the deltas vector around the chosen leader and applies these corrections to both the inter-channel delays and absolute spike times, yielding "adjusted_delay" and "adjusted_time" fields to be JSON-serialized for downstream processing. The raw unadjusted time is also JSON-serialized to be used by the audio playback feature of the front-end.

calibrate.py (calibrate script): This file provides the calibrate() routine that is invoked by audio_process.py. The calibration routine estimates per-microphone timing offsets so later analyses can correct inter-channel delays. It first converts a hard-coded 6 × 6 distance matrix (centimetres between every microphone pair) into a theoretical time-of-flight matrix, *T_true* (milliseconds), using *t = distance/(speed/10)* with the assumed speed of sound being 343 m/s, the speed of sound through air. A second 6 × 6 matrix, *T_meas,* holds the raw delays produced by a calibration recording in which a pulse is played at each mic location and observed on every channel; these measured delays equal the true propagation time plus an unknown electronic/placement offset, *delta_i,* for each mic:

$$T\_meas\{ij\} = T\_true\{ij\} + delta\_i + delta\_j$$

For every distinct pair, the script forms a linear equation:

$$delta\_i + delta\_j = T\_meas\{ij\} - T\_true\{ij\}$$

Stacking all 15 equations yields an over-determined system *A*delta = b,* which it solves via least squares (np.linalg.lstsq). The solution vector *delta* contains the raw offsets; the code then subtracts *delta_0* so microphone 0 becomes the reference (offset 0) and returns the relative delays both as a NumPy array and a dictionary for convenience. These

per-channel corrections are used by audio_process.py to compensate for hardware desynchronization.

5 Relevant Engineering Standards

Matthew Pipko

Domain	Standard / Regulation	Relevancy to the project
Patient-contact hardware	IEC 60601-1 (Ed. 3.2) & IEC 60601-1-2 (EMC)	Electrical safety & emissions for medical equipment used or worn by patients. Determines insulation, leakage current, and EMC test limits for the piezo microphones, jack-to-USB-C ADCs, and the powered hub.
	ISO 10993-1 / -5 / -10	Biocompatibility tests for the polyurethane foam and any protective SMS layer that touches the skin.
	NFPA 99 / NFPA 70 (U.S. hospital power)	If the pad is used in a clinical, patient-care area, the hub must plug into a hospital-grade receptacle, and the power strips must be medical-grade.
Electronics & connectivity	USB 2.0 Base Spec & USB Audio Device Class 1.0	Confirms 5V bus-power must be < 500 mA, descriptor tables, mono-audio streaming at 48 kHz. Ensures Windows lists each dongle as an independent MME input.
	FCC 47 CFR Part 15, Class B (or EN 55032 Class B in the EU)	This is applied to the hub + dongles as composite "unintentional radiation." Radiated & conducted emissions for digital devices used in homes/clinics.
	RoHS Directive 2011/65/EU	Restricts Pb, Hg, Cd, Cr VI, PBB, PBDE in solder, cables, piezo ceramics. Needed for CE marking.
Software life-cycle	IEC 62304:2006 + A1:2015	Software development life cycle for medical devices. Classifies risks, mandates unit/integration tests for the Node back-end and Python crackle-analysis scripts if the product is to be sold as a diagnostic.
Quality & risk management	ISO 13485:2016	QMS covering design-history file, supplier controls for the foam, dongles, and firmware.

Domain	Standard / Regulation	Relevancy to the project
	ISO 14971:2019	Hazard identification & risk controls (electrical shock, false-negative crackle detection, data loss).
Cyber-security & privacy	HIPAA (45 CFR Parts 160/164) &/or GDPR (EU 2016/679)	Audio files are Protected Health Information (PHI). This requires all of the files to have encryption at rest, user authentication, audit logs, and patient consent language.
	ISO/IEC 27001:2022	Information-security management system for the cloud VM or on-prem server that stores WAV files and analysis results.
	OWASP ASVS v4 & OWASP Top 10 (2021)	Secure-coding checklist for the Node.js API (e.g., input validation, auth tokens, CORS).
Web & data exchange	RFC 7231 & 7540 (HTTP/1.1 & HTTP/2), RFC 8446 (TLS 1.3)	Governs REST endpoints between React front-end and Node back-end; mandates HTTPS with modern cipher suites.
	ECMA-262 (ECMAScript 2023) & ECMA-404 (JSON)	Language spec for React (browser JS) and the JSON payloads exchanged with the API.
	W3C HTML5, CSS 3, ARIA 1.2 & WCAG 2.1 AA	Accessibility and semantic-markup rules for the React UI, so clinicians and patients with disabilities can use the web dashboard.
Coding conventions	PEP 8 (Python), Google JS Style Guide (or ESLint Airbnb)	Keeps the Python crackle-analysis script and Node/React codebase readable and maintainable..
Audio processing	RIFF/WAV PCM (Microsoft/IBM spec)	Ensures exported 48-kHz mono WAV files are interoperable with the Python analysis pipeline.

6 Cost Breakdown

Matthew Pipko and Shadin Almaiman

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Unit Cost	Extended Cost
4 pack of 3.5mm Male Plug to Bare Wire Open End Headset TRRS Cord 4 Pole 1/8" Jack Stereo Audio Cable	2	The cables required to connect the microphones to a 3.5 mm male microphone jack.	$4.97	$9.94
USB C to Aux Audio Dongle Cable Cord (Pack of 2), Type C to 3.5mm Female Headphone Jack Adapter	3	These dongle adapters act as ADCs for the microphones. The piezoelectric microphones have their recorded analog signals converted to digital to be assessed by the software.	$8.99	$26.97
Polyurethane Foam (36" x 24" x 1")	1	Foam acts as the main form of housing for the microphones, as well as the point for the patient to lie on.	$44.99	$44.99
USB-C Splitter	1	Combines six analog microphone inputs into a single USB-C output, allowing the device to transmit multi-channel audio data to a laptop for recording and processing.	$45.69	$45.69
10 pack: Piezo Contact microphones	1	Recording posterior lung sounds from the patient.	$9.99	$9.99
Beta Version-Total Cost				$137.58

7 Appendices

7.1 Appendix A - Specifications

Scoresheet used for final test:

OBJECT	Correct? (Y/N)
Software processing speed < 2 min?	Y
Microphones can pick up sound?	Y
Expected consistent order of microphone recording vs. actual order?	Y
Are there consistent results in uploading multiple single-channel files versus a single multiple-channel file?	Y
Is delay between peaks < 3 ms?	Y
App able to perform audio playback of sounds detected by channels on demand?	Y
Is reported delay calibrated to isolate real world delay from hardware skew?	Y

7.2 Appendix B – Team Information

1. Thinh Nguyen (CE 2025) tnguy19@bu.edu
2. Matthew Pipko (BME 2025) mattpi23@bu.edu
   Matthew Pipko is a Biomedical Engineering major from Newton, Massachusetts. Post graduation he will be working as a sales representative for BioIntelli, a life-science/biotech lead gen service provider.
3. Hilario Gonzalez (CE 2025) hilario@bu.edu
   Hilario is a Computer Engineering and Machine Learning student. Post graduation he will be continuing some ongoing projects and hopes to emerge into the computer vision industry that combines his passions for machine learning and automobiles.
4. Shadin Almaiman (BME 2025) shadin@bu.edu
   Shadin is a Biomedical Engineering major from Saudi Arabia, hoping to work in the medical device industry. After graduation, she will be getting her masters of Engineering in Biomedical Engineering in the UK.