ECE 445

SENIOR DESIGN LABORATORY

FINAL REPORT

Portable Offline Translator

Team #77

LORENZO BUJALIL SILVA
(lorenzo9@illinois.edu)
JOSHUA CUDIA
(cudia2@illinois.edu)

TA: Chi Zhang

May 7, 2025

Abstract

Modern advancements in the field of deep learning has enabled the development of models directly optimized for speech recognition, translation, and synthesis. The open source availability of these models has made it possible for the creation of end-to-end language translation systems. Our project focuses on integrating such a system onto an embedded platform, capable of running entirely offline and anywhere in the world. The system listens for spoken input, transcribes to source language text, translates to target language text, synthesizes target language speech, and plays back the translated speech. We focus on the challenges of performance and resource constraints by integrating multiple subsystems to ensure accurate and low-latency inference directly on-chip. We demonstrate translation in real-time with low latency and intelligible speech output across multiple languages.

Contents

1 Introduction

1.1 Purpose

Traveling is an exciting part of life that can bring joy and new experiences. Trips are the most memorable when everything goes according to plan. However, the language barrier can limit communication with others, causing unnecessary stress on an otherwise enjoyable trip. Although most modern phones provide translation applications, these require a reliable internet connection. In times when the connection is weak or there is no connection at all, translation apps may not be a solution.

1.2 Functionality

We want to solve this problem by building a portable translator that you can ideally use anywhere in the world without internet connection. The idea is to have a small device that can be programmed to make translations between two different languages, then is able to listen what the person says, converts the speech to text, translates the text to the target language, then converts the translated text back to speech, and drives a speaker with the target translated speech. We want to design our translator to encompass a few subsystems: Main Processing Subsystem (ESP32-S3), Secondary Processing Subsystem (Raspberry Pi CM5), Audio Subsystem, User Interface Subsystem, and Power Management Subsystem. Through this design, someone should be able to turn on the device, set the languages up and start talking into the device, and after a few seconds the translated speech will be played. Ideally, this will facilitate communication between people without a common language and make things simpler while traveling.

1.3 Performance Requirements

1.3.1 Translation Latency

During the project proposal, we intended on having our system to be capable of translating spoken input to text and vice versa within 3 seconds to ensure real-time usability. This will be the time that it takes from once the person stops talking to the time that the person is able to hear audio on the speaker. Throughout the semester, we encountered various issues with memory and performance requirements on the compute subsystem that required higher latency than anticipated.

1.3.2 Translation Accuracy

This system should be capable of maintaining an accuracy of at least 90% for common phrases and vocabulary. This is going to be very dependent on the model size, where models that have more parameters are capable of recognizing more languages with higher accuracy and responding better to prompts given. This stage can be calculated through the first recognition model capable of interpreting through a score of 90% on the semantic similarity score. Then on the translation model capable of scoring 90% on a multilingual sentence transformer. Then finally another semantic similarity score of 90% on the text to speech model.

1.3.3 Intelligible Speech Output

The speaker output should be clear and audible within typical decibel ranges (e.g 60db) of normal conversation. This will ensure that we are able to understand what the output language is saying and conversation can flow with ease.

1.4 Subsystem Overview

1.4.1 Block Diagram

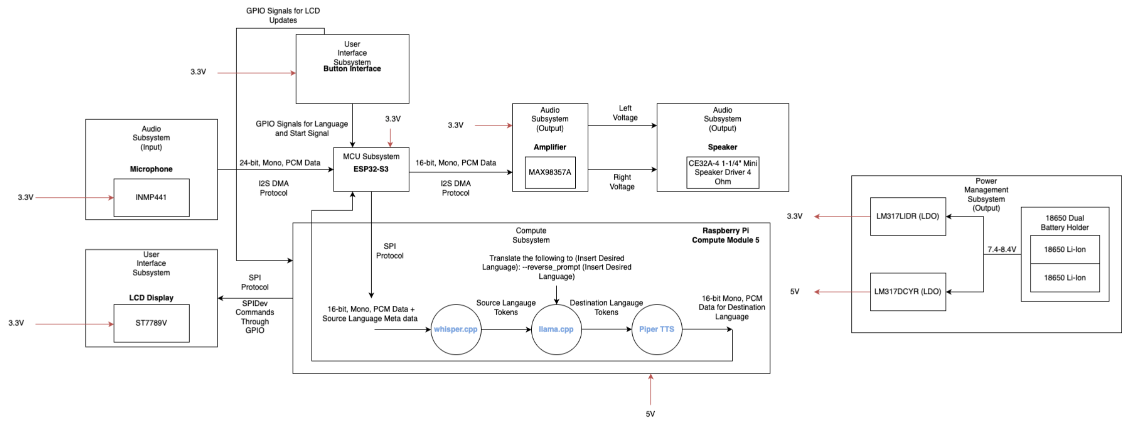


Figure 1: Block diagram of the translator system

1.4.2 Block-Level Changes

We had to make many changes for this block diagram throughout the semester to make our development easier. The first thing that we started working in the semester was developing the inference pipeline prototpe on a general purpose computer which we later integrated it into the compute module. We validated the working of the pipeline working and simulated working from other modules. Later on when integrating the other components including the main processing unit, we had many issues reading data from the audio sensor, and we also had problems in complicated PCB design for the STM32 microcontroller. We were able to change our design to use the ESP32-S3 MCU that easily works with the INMP441 microphone. We were able to read data from the microphone and then we setup the SPI interface to the compute module, we were then able to reconstruct the wave file on the compute module to validate that we were able to transmit data. We also had some plans to drive the LCD display using the main processing unit but then we made the change to manage the display on the CM5 and then multiplex between the SPI transfer and updates to the display.

2 Design

2.1 MCU Subsystem

The MCU subsystem is made up of the firmware flashed on the ESP32-S3 to be able to interact with audio peripherals via I2S, transfer audio data to the CM5 [1] via SPI, and finally interact with the GPIO to start the state machine.

2.1.1 Design Details

- **I2S Audio System**:
  - **Microphone Configuration:** We utilized the INMP441 [2] I2S based microphone. We configured it as an I2S master reciever with a 16kHz sampling rate and a 16-bit mono channel. The audio samples are read via DMA into a circular buffer to separate audio capture and transmit.
  - **Speaker Configuration**: We used the MAX98357A 4Ω and $2W$ Class D amplifier with can receive digital 16-bit mono PCM data. We used a I2S master transmit configuration to write the recieved PCM data from the compute module. [3] [4]
- **SPI Communication**: We configured the ESP to work as a SPI slave. The CM5 will work as the master and can request audio chunks from the ESP [5]. In order to notify the CM5 to request data, we sent over a GPIO interrupt to the Pi. The Pi would then poll the ESP until we stop holding the button.
- **Circular Buffer**: We implemented a circular buffer to be able to make sure that the SPI and I2S data communication protocols are better aligned. Since there is some delay in reading the data over SPI, and it takes some time to load the buffer, we need to make sure we are not starving for data and we are not overloading the buffer. Therefore we can make a circular buffer large enough to prevent these issues. We used a read and write pointer to coordinate communication.

-

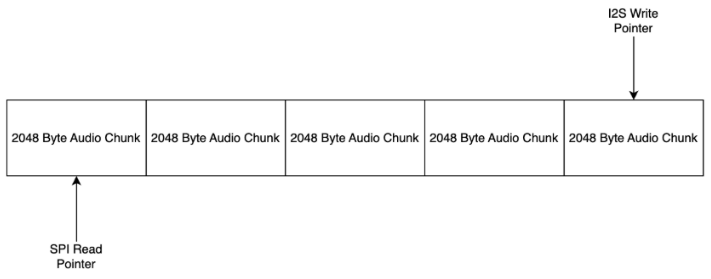


Figure 2: Circular Buffer

- **MCU State Machine**: This is our main state machine for the main processing unit where we are going to synchronize data collection, communication, and transmission.

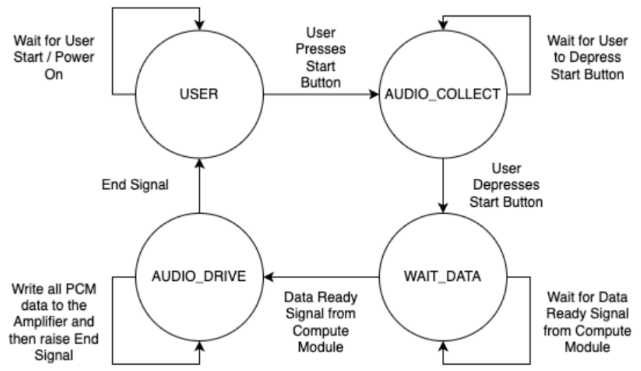

Wait for User Start / Power On
USER
User Presses Start Button
AUDIO_COLLECT
Wait for User to Depress Start Button
End Signal
User Depresses Start Button
AUDIO_DRIVE
WAIT_DATA
Write all PCM data to the Amplifier and then raise End Signal
Data Ready Signal from Compute Module
Wait for Data Ready Signal from Compute Module

Figure 3: MCU State Machine

2.1.2 Design Alternatives

We could have configured the ESP32 to work as a SPI master instead of a slave. We could have been able to transmit all of the data required over to the Pi and manage all the timing on the ESP. The main issue from this was that the Pi does not have support to act as a SPI slave without significant kernel code.

2.1.3 Subsystem Schematic

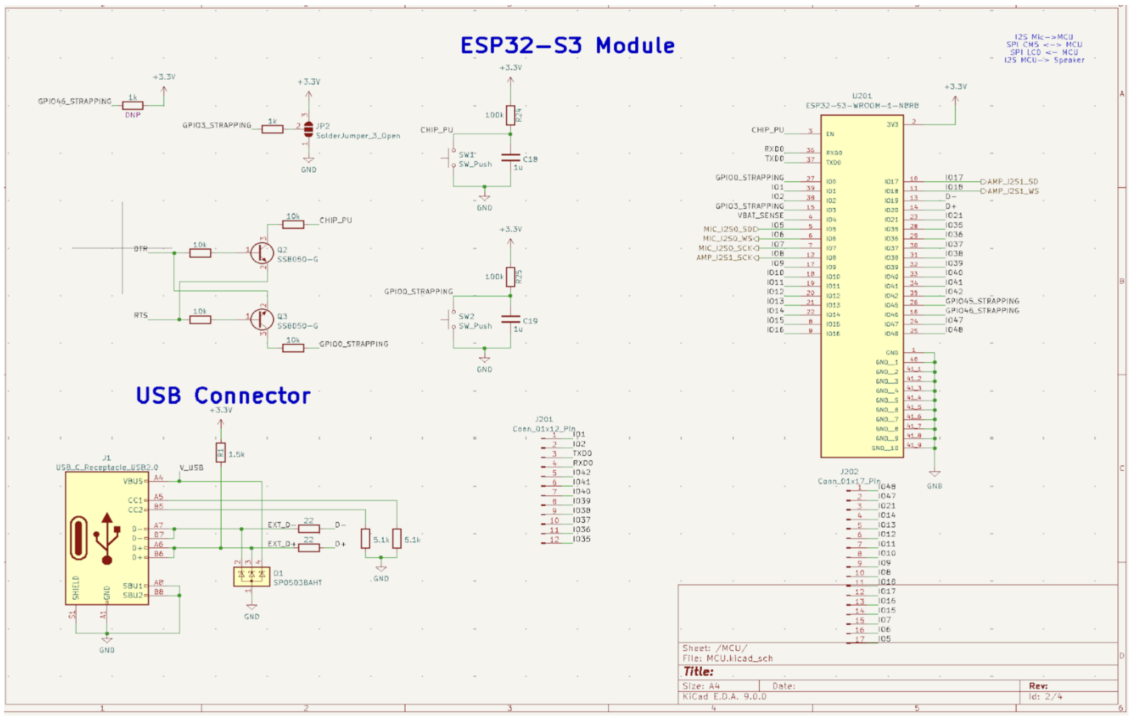


Figure 4: MCU Schematic

2.2 Compute Subsystem

2.2.1 Design Description

The main purpose of the compute module is to offload high compute tasks, including speech to text transcription, translation, and text to speech conversion. It would be too computationally complex to host all three models on the ESP32-S3 while processing I/O data. We specifically chose the RPI Compute Module because it has the computational power to run the AI models, it can interface over SPI to the MCU, it runs Linux, and it has eMMC Flash to store the models on board. We decided that we need to use SPI in this case since we offloaded data in real time from the MCU. For this subsystem, we built an infrastructure around querying the models, reading and sending data to and from the MCU, and a data processing pipeline to move through different stages of translation.

Effectively the model framework we used is the one based around a tensor library for machine learning called ggml that has branching projects capable of doing inference for speech recognition and speech translation. The two projects that we used are whisper.cpp [6] and llama.cpp [7]. When the speech data comes through as 16-bit mono PCM, we reconstruct this data and then do some data processing such as sign extensions to be able to provide it to the 32-bit based whisper.cpp framework that will return tokens in the desired language, from these tokens we have them as a string that will be used to prompt the llama.cpp framework to translate to a desired language with some prompt engineering to extract the desired language extracted. Once we get the translated language, we provide this to a text to speech model, Piper [8], that will interpret the tokens and regenerate the PCM data to be delivered back in SPI to the ESP32.

2.2.2 Design Alternatives

The main design alternative that we can do to this design for the compute subsystem is making changes to the models that we used for inference. We used a multilingual transcription model made up 809 million parameters, this is a very large model that is very memory intensive and requires significant performance to be able to do inference. One way that we can improved this is using a smaller model (e.g. 39 million parameters) that will only require 1 GB of memory.

Another thing that we could have changed in our design was using a trained model that is capable of only doing inference between two languages. Currently we have a prompt based model, using a decoder only transformer. Instead we could use a larger encoder-decoder translator for translation between two particular languages. We could have reduced the generalization of our translator by having a few models capable of doing translation for a particular set of languages.

2.2.3 Subsystem Diagram

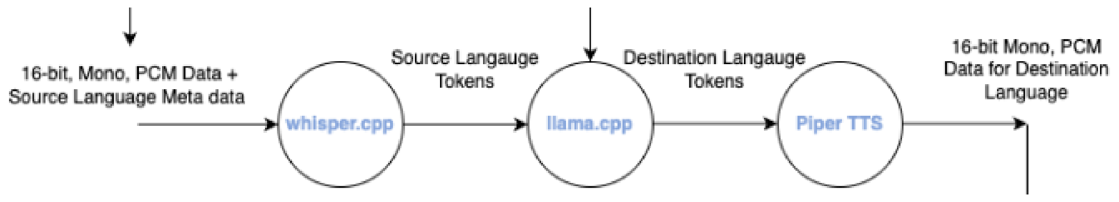

16-bit, Mono, PCM Data + Source Language Meta data
whisper.cpp
Source Langauge Tokens
llama.cpp
Destination Langauge Tokens
Piper TTS
16-bit Mono, PCM Data for Destination Language

Figure 5: Inference Pipeline Diagram

2.3 Audio I/O Subsystem

2.3.1 Design Description

In this subsystem, we had two I2S devices including a microphone and amplifier. These both will manage our analog to digital, and digital to analog conversion for speech input and output. In both cases, we used 16-bit mono audio configuration for our PCM audio.

2.3.2 Subsystem Schematic

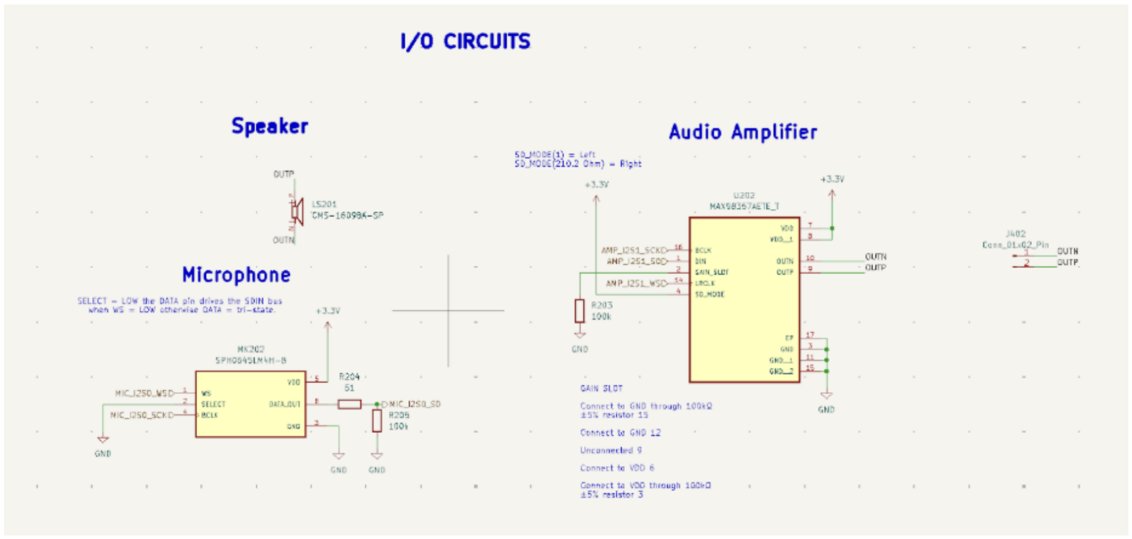


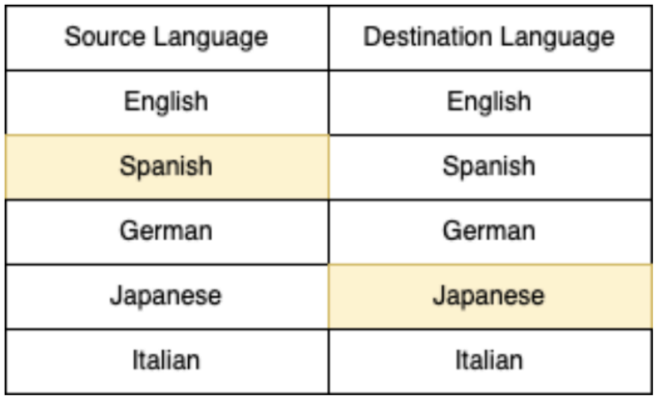
Figure 6: Audio Schematic

2.4 User I/O Subsystem

2.4.1 Design Description

In this subsystem, we had a LCD display [9] allows the user decide which languages to translate between and some push buttons to be able to decide. We also added push buttons that will start listening on the microphone, then stop listening so we can ensure that all of the data has been stored.

2.4.2 Subsystem Diagram



Source Language	Destination Language
English	English
Spanish	Spanish
German	German
Japanese	Japanese
Italian	Italian

Figure 7: LCD Diagram

2.5 Power Management Subsystem

2.5.1 Design Description

This portable power management system uses a Samsung 25R 18650 2500mAh 20A rechargeable Li-Ion battery to supply stable voltage to two power rails. The 5V power rail will be used for the Raspberry Pi Compute Module (CM), while the 3.3V power rail will support the MCU, LCD, and audio subsystem.The system includes two LM317DCYR adjustable LDO regulators for a 3.3V and 5V output.

The current Power Management Subsystem contains two adjustable LDO voltage regulators. To further enhance this subsystem, DC-DC converters can provide efficiency (up to 95%), reduced heat dissipation, and lower power loss. Furthermore, a combination of LDO and DC-DC converters would use a switching regulator for the main power conversion, followed by an LDO for noise-sensitive analog circuits. Lastly, minor improvements to thermal management such as the addition of heats sinks would improve the performance of linear regulators.

2.5.2 Design Alternatives

The following design alternatives can be applied:

- Switching Regulators (DC-DC Converters)
- Combination of LDO and DC-DC Converter
- Thermal Management Enhancements

2.5.3 Equations & Simulations

The following equation is used to calculate the output voltage Vout for the LM317 voltage regulator:

$$V_{\text{out}} = 1.25 \times \left(1 + \frac{R_2}{R_1}\right)$$

3.3 V calculation:

- $R_1 = 330\,\Omega$
- $V_{\text{desired}} = 3.3\,\text{V}$
- $R_2 = \left(\frac{3.3}{1.25} - 1\right) \times R_1 \approx 540\,\Omega$

5 V calculation:

- $R_1 = 330\,\Omega$
- $V_{\text{desired}} = 5\,\text{V}$
- $R_2 = \left(\frac{5}{1.25} - 1\right) \times R_1 \approx 1\,\text{k}\Omega$

2.5.4 Subsystem Schematic

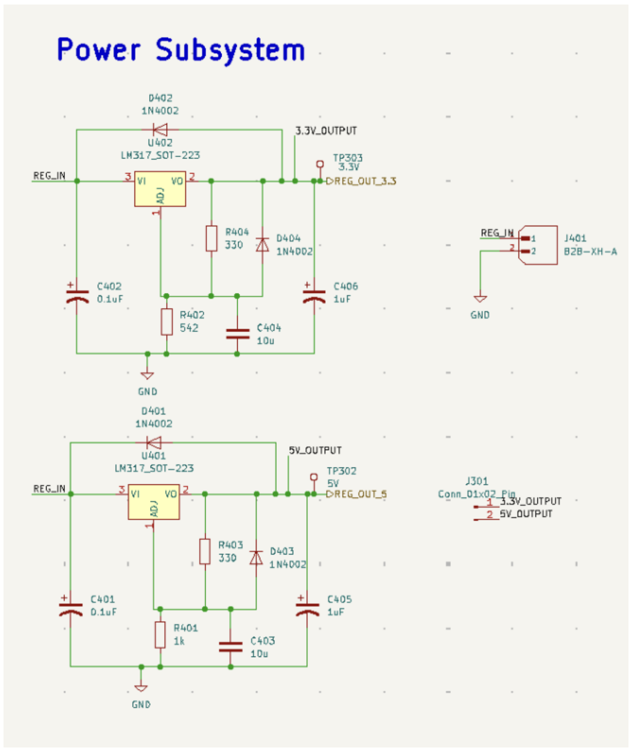
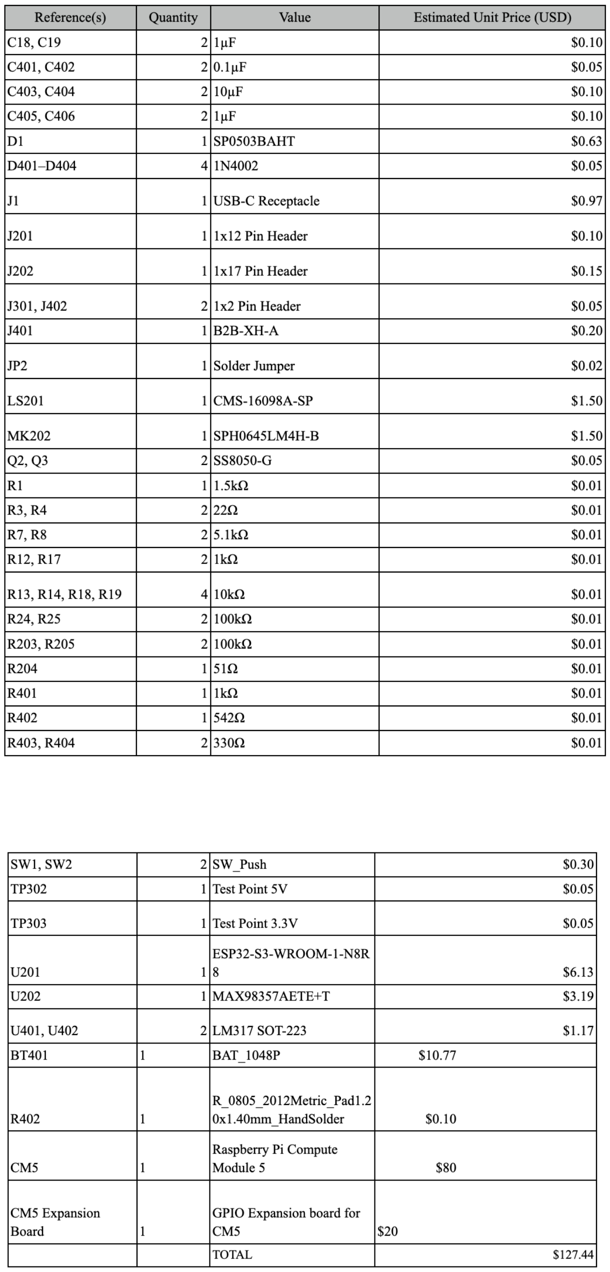


Figure 8: Power Schematic

3 Costs & Schedule

3.1 Costs

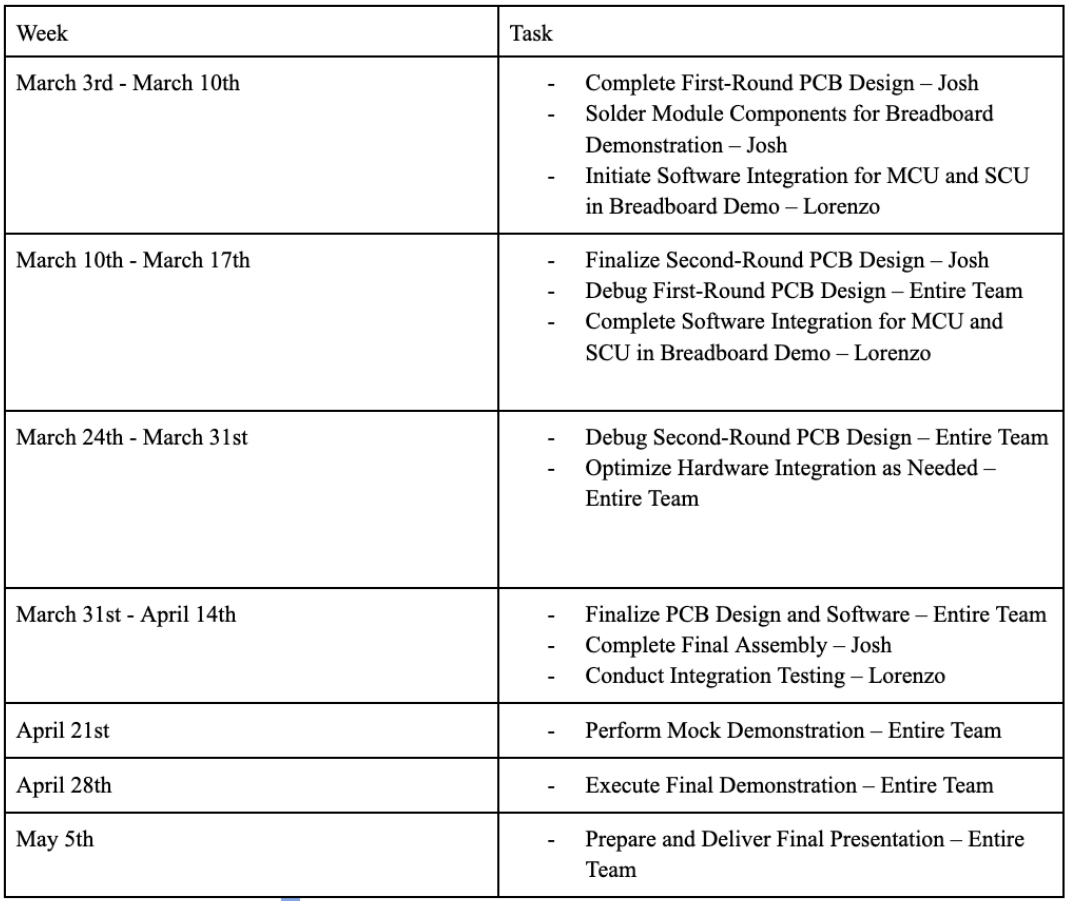
The total cost for parts, as shown in table below, is $127.44 before shipping. Considering a 5% shipping cost, which adds $6.37, and a 10% sales tax, which adds $12.74, the total cost will be $146.56.



Reference(s)	Quantity	Value	Estimated Unit Price (USD)
C18, C19	2	1µF	$0.10
C401, C402	2	0.1µF	$0.05
C403, C404	2	10µF	$0.10
C405, C406	2	1µF	$0.10
D1	1	SP0503BAHT	$0.63
D401–D404	4	1N4002	$0.05
J1	1	USB-C Receptacle	$0.97
J201	1	1x12 Pin Header	$0.10
J202	1	1x17 Pin Header	$0.15
J301, J402	2	1x2 Pin Header	$0.05
J401	1	B2B-XH-A	$0.20
JP2	1	Solder Jumper	$0.02
LS201	1	CMS-16098A-SP	$1.50
MK202	1	SPH0645LM4H-B	$1.50
Q2, Q3	2	SS8050-G	$0.05
R1	1	1.5kΩ	$0.01
R3, R4	2	22Ω	$0.01
R7, R8	2	5.1kΩ	$0.01
R12, R17	2	1kΩ	$0.01
R13, R14, R18, R19	4	10kΩ	$0.01
R24, R25	2	100kΩ	$0.01
R203, R205	2	100kΩ	$0.01
R204	1	51Ω	$0.01
R401	1	1kΩ	$0.01
R402	1	542Ω	$0.01
R403, R404	2	330Ω	$0.01
SW1, SW2	2	SW_Push	$0.30
TP302	1	Test Point 5V	$0.05
TP303	1	Test Point 3.3V	$0.05
U201	1	ESP32-S3-WROOM-1-N8R8	$6.13
U202	1	MAX98357AETE+T	$3.19
U401, U402	2	LM317 SOT-223	$1.17
BT401	1	BAT_1048P	$10.77
R402	1	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	$0.10
CM5	1	Raspberry Pi Compute Module 5	$80
CM5 Expansion Board	1	GPIO Expansion board for CM5	$20
		TOTAL	$127.44

Figure 9: Costs Table

3.2 Schedule



Week	Task
March 3rd - March 10th	- Complete First-Round PCB Design – Josh - Solder Module Components for Breadboard Demonstration – Josh - Initiate Software Integration for MCU and SCU in Breadboard Demo – Lorenzo
March 10th - March 17th	- Finalize Second-Round PCB Design – Josh - Debug First-Round PCB Design – Entire Team - Complete Software Integration for MCU and SCU in Breadboard Demo – Lorenzo
March 24th - March 31st	- Debug Second-Round PCB Design – Entire Team - Optimize Hardware Integration as Needed – Entire Team
March 31st - April 14th	- Finalize PCB Design and Software – Entire Team - Complete Final Assembly – Josh - Conduct Integration Testing – Lorenzo
April 21st	- Perform Mock Demonstration – Entire Team
April 28th	- Execute Final Demonstration – Entire Team
May 5th	- Prepare and Deliver Final Presentation – Entire Team

Figure 10: Schedule Table

4 Requirements & Verification

4.1 Completeness of Requirements

- **Microcontroller Subsystem (ESP32-S3)**
  - Captures 16kHz, 16-bit mono PCM audio via I2S Microphone.
  - Drives 16kHz, 16-bit mono PCM audio via I2S Amplifier.
  - Buffers audio data into a circular buffer of 16 chunks made up of 1024 samples within 1.024 seconds.
- **Compute Subsystem (Raspberry Pi Compute Module 5)**
  - Runs STT, translation, and TTS models locally within 50 seconds for end-to-end latency and with 90% accuracy in semantic similarity.
- **Power Subsystem**
  - Regulates 5V for Raspberry Pi and 3.3V rails for MCU, and peripherals.

4.2 Appropriate Verification Procedures

- **Microcontroller Subsystem (ESP32-S3)**
  - Verified I2S audio capture and transmission via a oscilloscope to validate data packets being sent are aligned. We can also validate the clock speed for the desired frequency.
  - Create a sample audio array made up of the size of the buffer and read sample data from the microphone, then using the timer function on the ESP to check if the buffer was filled in 1.024 seconds.
- **Compute Subsystem (Raspberry Pi Compute Module 5)**
  - Run the pipeline with a sample PCM audio file and then log the timestamp that it takes at the start of transcription and then the end of the speech synthesis. We can validate that the time it takes is less than 50 seconds. We came up with this time because of the average of the times that we had for most language translations.
- **Power Subsystem**
  - Validated voltage levels at 3.3V and 5V rails using DMM and bench-top power supply.
  - Utilized oscilloscope under full workload to analyze load and line regulation.

4.3 Quantitative Results

4.3.1 Compute Subsystem

- **Translation Latency**: We configured out our translation latency by taking a dataset of audio data and sending into our inference pipeline and evaluating the total runtime and the timestamps for each models along with the memory utilization. We validated that our design is capable of a runtime of under 50 seconds.
  - Configuration:
    - llama.cpp: mistral-7b.Q4_K_M.gguf
    - whisper.cpp: ggml-tiny.bin
  - Results:
    - Total Runtime: 38s
    - Whisper Inference Duration: 19s
    - Translation + TTS Duration: 6s
    - Peak RSS: 4500 MB
    - Peak VSZ: 4900 MB

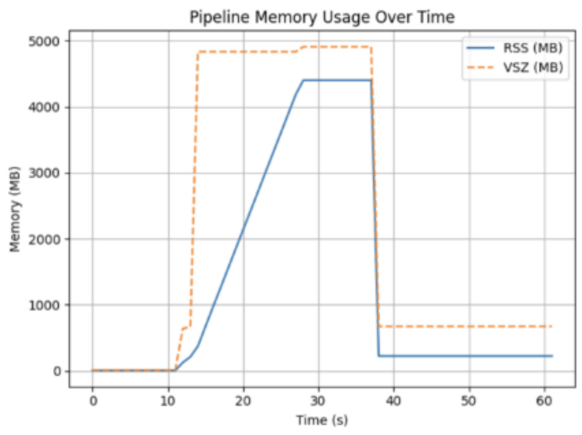


Figure 11: Translation Pipeline Memory Usage

- **Translation Accuracy**: For translation accuracy, we would take the dataset of samples of PCM data with their transcription to validate the edit distance for the whisper model and then the translation for semantic similarity.
  - Configuration:
    - llama.cpp: mistral-7b.Q4_K_M.gguf
    - whisper.cpp: ggml-tiny.bin

    - Speech Dataset: LibriSpeech (16kHz PCM + transcriptions)
  - Results:
    - **Average Levenshtein Accuracy**: 92.3%
    - **Average Semantic Similarity**: 90.5%

4.3.2 Power Subsystem

- **3.3V Channel**:
  - Target Voltage: 3.3V
  - Measured Voltage: 3.24V (±0.01V tolerance)
  - Load Current: Up to 500mA (measured at full load)
  - Ripple Voltage: < 20mV (measured at full load)
- **5V Channel**:
  - Target Voltage: 5.0V
  - Measured Voltage: 5.13V (±0.02V tolerance)
  - Load Current: Up to 1A (measured at full load)
  - Ripple Voltage: < 30mV (measured at full load)

5 Conclusion

5.1 Accomplishment

Built a portable, offline translator that performs real-time speech to speech translation. We integrated embedded systems ESP32-S3 and Raspberry Pi CM5 for on-chip inference while dealing with real world bottlenecks from embedded machine learning. We were able to integrate various subsystems including the main processing unit firmware and allowed it to communicate with the secondary processing unit to do real time inference. We were also able to build a power subsystem for power regulation for different voltage levels for each processor and peripherals. Additionally, we were successful in our speech translation pipeline capable of doing translation to and from more than 50 languages. Our transcription model, whisper, was multilingual and capable of resisting to variable noise and accents. We then took the transcribed text from the model into the prompt based translation model capable of supporting multiple languages and generates high accuracy text. Finally we had our TTS model, Piper, capable of generating the necessary translated speech data in 16-bit PCM audio format. Finally, we designed a custom PCB with reliable power deliver with voltage regulation and ESD protection. We validated all power pathways through test points to ensure the system remained functional.

5.2 Uncertainties

Some of the main uncertainties for this project was the speech translation pipeline latency on embedded hardware. What we found out about this project is that embedded machine learning is difficult to optimize fully to reduce the latency. We had to cut back on many functionality points of the device to ensure that the device would be able to be usable in a real world setting. We also would need to reduce the performance of the system by using quantized models with less parameters and potentially less understood languages. Using these models will put less strain on the memory and power usage, and then have a better performance. We could have also integrated some level of cooling for the compute module to ensure that the heat levels of the Pi were low to have good performance levels.

5.3 Future Work

In the future, we hope to improve the functionality of the translator by improving the translation latency. We want to get rid of the prompt based translation. The prompt based decoder model is more GPT-like and is very general. In a more specialized case, we could use trained models for particular language translation that are optimized to work on embedded hardware. These local models would be able to use much less memory therefore improving latency. Another thing we could do is expand the storage capacity of the Pi so that we can store more of these specialized models and then multiplex between them when needing to load them into memory for different use cases. We also want to resolve PCB related issues caused at boot mode. We hope to add test points and bench-top equipment to be able to debug the PCB issues.

5.4 Ethical Considerations

- **Open Source Usage**: Our project is based on many open source projects. We have adapted these projects to meet our projects needs. We cited the original authors and comply with their licenses.
- **Battery Safety**: We incorporated mechanisms for protecting overcharging, overheating, and short circuiting. We did this to comply with industry standards, including IEEE 1725-2021 for rechargeable battery safety.
- **Translation Misuse**: Our project can be subject to translating sensitive language that can cause harm to other people. We integrated models that able to prevent possible generation of foul language.

References

[1] Raspberry Pi Ltd. "Raspberry Pi Compute Module 4/5 Datasheet." (2025), [Online]. Available: https://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf (visited on 05/07/2025).

[2] TDK InvenSense. "INMP441 Datasheet." (2025), [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf (visited on 05/07/2025).

[3] Dayton Audio. "Dayton Audio CE32A-4 Datasheet." (2025), [Online]. Available: https://www.daytonaudio.com/images/resources/285-103-dayton-audio-ce32a-4-spec-sheet.pdf (visited on 05/07/2025).

[4] Maxim Integrated. "MAX98357A Datasheet." (2025), [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/MAX98357A.pdf (visited on 05/07/2025).

[5] Espressif Systems. "ESP32-S3-WROOM Datasheet." (2025), [Online]. Available: https://www.espressif.com/en/products/modules/esp32-s3-wroom-series (visited on 05/07/2025).

[6] G. Gerganov. "whisper.cpp." (2025), [Online]. Available: https://github.com/ggerganov/whisper.cpp (visited on 05/07/2025).

[7] G. Gerganov. "llama.cpp." (2025), [Online]. Available: https://github.com/ggml-org/llama.cpp (visited on 05/07/2025).

[8] Piper TTS. "Piper TTS Model." (2025), [Online]. Available: https://github.com/rhasspy/piper (visited on 05/07/2025).

[9] Waveshare.com. "2inch LCD Module." (2025), [Online]. Available: https://www.waveshare.com/wiki/2inch_LCD_Module#Underlying_hardware_interface (visited on 05/07/2025).