Portable Offline Translator

ECE 445 Design Document - Spring 2025

Project #77

Joshua Cudia and Lorenzo Bujalil Silva

Professor: Arne Flifet

1 Introduction

1.1 Problem

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 Solution

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 (MCU), Secondary Processing Subsystem (Compute Module), Audio Subsystem, User Interface Subsystem, Communication 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. Then, the device can be programmed the other way to have the other party translate. Ideally, this will facilitate communication between people without a common language and make life easier while traveling.

1.3 Visual Aid

A common representation of an embedded translation device can currently be seen on the market. However, we have noticed that many times this device will be sold as a package with a variety of functionality including online translation or photo translation.

Figure 1. Retail Translator

This inclusion of functionality provides some better usage for the device, but will also significantly increase the cost. Ideally, all of this additional functionality would be provided by a phone that most people already have. The problem that we are setting to solve is one where the user will have no access to the internet and need to be able to communicate. We want to design a device that is proficient in simplicity. Being able to clearly understand and translate to ensure proper communication when there are no other resources available.

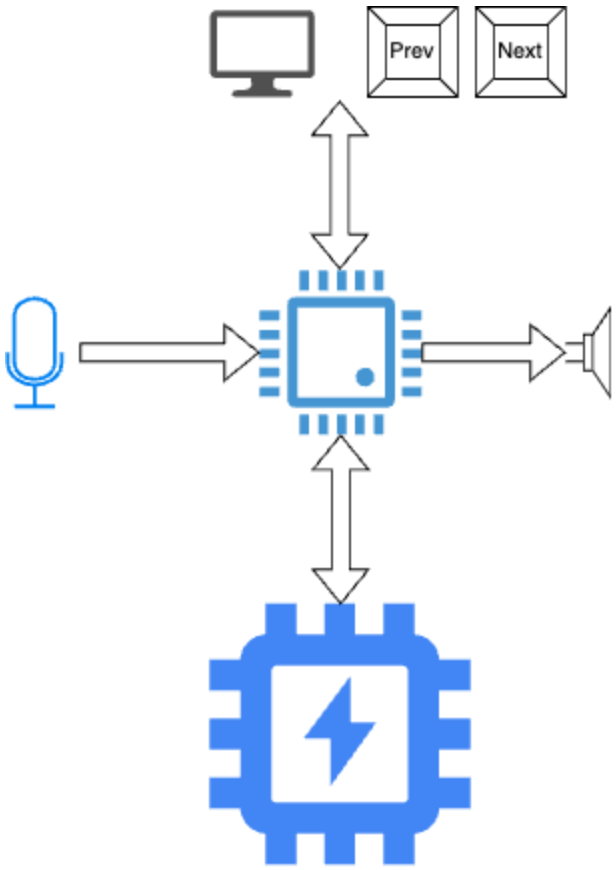


Figure 2. Visual Diagram

1.4 High Level Requirements

1. Translation Latency: This system should 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.

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.

3. Speaker Noise: 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.

2 Design

2.1 Block Diagram

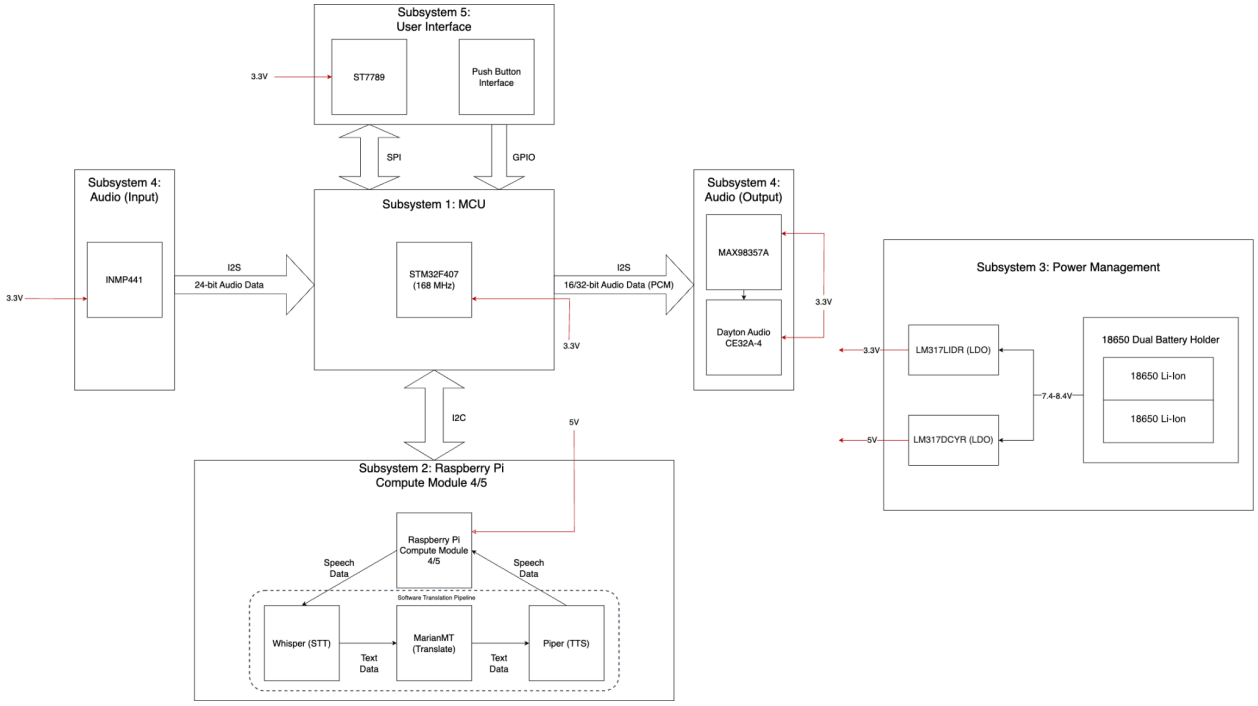


Figure 3. Block Diagram

2.2 Functional Overview & Block Diagram Requirements

2.2.1 MCU (Main Processing System)

The main processing subsystem will manage the workflow for the system, control all I/O, and communicate commands/data to the secondary processing subsystem. When the system powers on a simple interface will be prompted to the user to allow them to select the source and target language to translate. The MCU will support the user inputs through a push button to select the language and will drive the display. Then when the user decides to start translation, through a particular push button, the MCU will change states to start listening on the port for audio data from the microphone. The INMP441 microphone will output a digital signal and communicate over I2S which can be interpreted through our MCU. The MCU will also need to buffer data and need to normalize it to be within the appropriate bit range to be interpreted by the STT model. After preprocessing the data, we will need to set up code to communicate packets of this data over a SPI protocol to the compute module. We also may need to set up some kind of custom protocol to set the compute module to start listening for a data sequence. Then the

compute module will take over and do the translations and conversions to speech and output pulse code modulation data. This data is transmitted again over SPI to the MCU that is listening. We decided to use SPI in this case because we needed high data transfer speeds to communicate the audio data. Since there is only 1 MB of flash memory and 192 KB of SRAM, we are strictly limited by the available memory to store entire audio files. This then will require us to create a circular buffering system where we are collecting data from the speaker from one end of a buffer and then offloading the data when we get to a certain threshold of capacity in this buffer. The microphone will be able to write directly to memory through DMA, and when the CPU has an available time slice it will be able to send off the data quickly to the compute module over SPI. Then the MCU will move to another state to start writing the data to the MAX98357A that will drive the speaker. Then the MCU will move back to a state of user input again to allow the user to translate again. Other than managing the entire workflow for the system, it needs to control the I/O which will include reading inputs from the user on the push buttons and will need to drive an LCD display to show what the user is currently selecting. With enough time, we may also add some status messages onto the LCD display to see what is happening in the system.

We decided to use the STM32F407 for this project because we required high levels of communication between various systems along with the numerous I/O. We also found that it has a LCD parallel interface and JTAG interface. We also have a long reach goal to do some audio manipulation (e.g. filtering, noise reduction) before sending it off to the compute module. We can also expect to support a real time control of the audio and peripheral management.

Subsystem Requirements	**Subsystem Verification**
1. The main requirement of the MCU is to orchestrate the state of the system and manage data flow. We can quantify this impact by being able to buffer 10 s of speech data into the pipeline to be translated. Since it is also managing the flow of the system, the end to end performance should be measurable here to be under 500 ms. This would include the	1. Equipment: An oscilloscope and logic analyzer will verify SPI and I2S communication timing, while a DMM ensures proper voltage levels. Debug software (in CubeIDE) will track state transitions, and a function generator will provide test audio signals. 2. Test Procedures: The MCU will be powered on, and correct LCD language selection will be verified via button

time between a user finishing speaking to getting speech out from the speaker.	presses. Audio input from the INMP441 microphone will be monitored to confirm proper I2S transmission and buffering. SPI communication with the compute module will be tested by sending pre-recorded data, ensuring a response time below 500 ms. 3. Presentation of Results: Oscilloscope and logic analyzer readings will confirm protocol timing, while debug logs and testpoint outputs will verify state changes. Voltage and current measurements will ensure power stability, and system latency will be recorded to confirm it remains under 500 ms. Waveform captures of I2S and SPI signals will be documented as verification evidence.

2.2.1 Raspberry Pi Compute Module 4/5 (Secondary Processing Subsystem)

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 STM32 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 are going to be offloading data in real time from the MCU. For this subsystem, we are going to have to build 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. It will also need to have some level of state awareness to know what kind of translation is being requested. We hope to design the PCB so that we can simply plug in the compute module onto the PCB through pinouts.

We need to start to build a software program that will orchestrate the pipeline of data through various ML models that will perform transcription, translation, and speech synthesis. This will also need to support listening to the microcontroller port to decide when it should start to process data. We will need to start

with a general implementation of this data pipeline where we can initially use APIs to query models. Once we get this to work, we can figure out exactly how much memory and storage we will require from these models to determine if we need to do some quantization or add more storage space. This is to have the models on board and able to be queried offline.

Once we get a basic understanding of the capabilities of being able to query a pipeline online, we are able to start pulling in models and generating a system capable of being queried on a higher performance host system (e.g. M2 Mac Pro). We will design a build system capable of pulling various open source projects together to be queried under one pipeline with each of their models being locally stored in memory. Effectively the model framework we are going to be using 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 are going to be using are whisper.cpp and llama.cpp. When the speech data comes through as 24-bit mono PCM, we are going to 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 will 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 will provide this to a text to speech model, Piper, that will interpret the tokens and regenerate the PCM data to be delivered back in SPI to the STM32. From the initial testing, we are able to store larger models within memory to have higher accuracy, but we may need to decrease the performance to be able to actually store the models on the edge.

Once we are able to validate test results on a host system we are able to move to the compute module that we will run some initial testing through a designed I/O board where we can flash the system and simulate input.

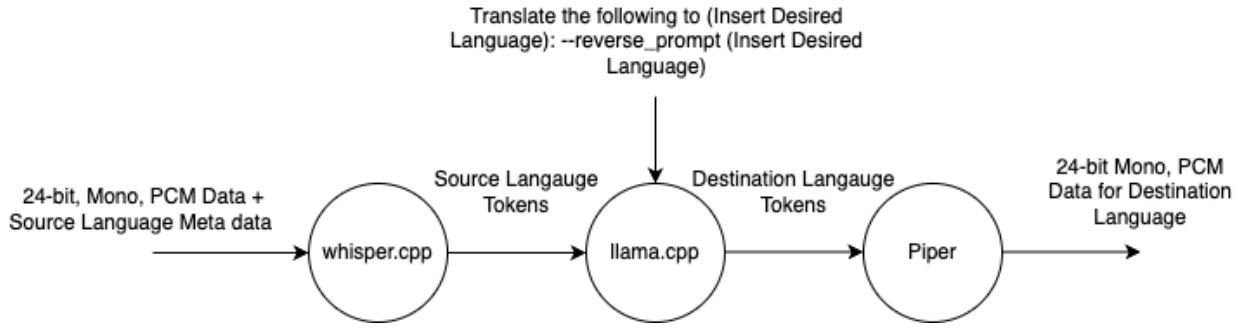


Figure 4. Software Pipeline

Subsystem Requirements	Subsystem Verification
1. The main requirement of this processor would be to ensure appropriate translation of the data. Ideally this processor should have the software infrastructure to translate the data with 90% accuracy across each of the models. This would be measured across an ideal translation. We would also ensure that the data is translated within 300 ms.	1. Equipment: We will use a logic analyzer to verify SPI communication integrity between the compute module and MCU, while an oscilloscope will confirm signal timing. Benchmarking software will measure AI inference time, and memory profiling tools will ensure efficient storage use. 2. Test Procedures: The compute module will boot and establish SPI communication with the MCU, ensuring proper data reception. Audio input will be processed into 24-bit PCM before feeding into whisper.cpp for transcription. The accuracy of whisper.cpp transcription will be compared against expected outputs. Translated text from llama.cpp will be evaluated for correctness, and Piper will generate speech, with PCM data reconstructed and returned to the MCU. The system will be tested under load to ensure a translation time below 300 ms and an accuracy rate of 90% or higher. 3. Presentation of Results: SPI transaction logs will confirm end-to-end data transmission, while debug logs will capture AI model inference times. We will evaluate the accuracy by comparing transcriptions and translations to ground truth datasets. Benchmarking results will document translation speed, and system

	memory usage will be analyzed to ensure feasibility for edge deployment.

2.2.1 Audio I/O Subsystem

We are going to have a LCD display that can let the user decide what languages to translate between and some push buttons to be able to decide. We are also going to have a push button that will start listening on the microphone, then stop listening so we can ensure that all of the data has been stored. In the case that the translation lasts too long, we may add some feature to automatically stop the input of speech so we make sure not to have too much data to translate. This UI subsystem will essentially make it so that this is a usable product.

Subsystem Requirements	**Subsystem Verification**
1. The main requirements of this system would be accuracy in retrieving audio data and delivering understandable speech on the speaker. We would hope to have 95% accuracy in retrieving data from the microphone and driving the speaker to be understood by a person at around 60 dB.	1. Define Equipment: To verify this subsystem, we will first utilize the test points for the I2S communication between the MCU and the INMP441 microphone as well as the I2S communication between the MAX98357A audio amplifier. In order to achieve audible volume, we will measure the speaker output using a decibel meter. 2. Define Test Procedures: We will record test phrases with the microphone and analyze the I2S waveforms using the test points and a logic analyzer. We will benchmark these results with our goal of 95% accuracy. We will also consider the SNR of the audio amplifier with expected values in the MAX98357A datasheet. 3. Define Presentation of Results: Captured I2S waveforms will be stored to verify digital accuracy, while decibel readings will confirm speaker output level. Audio comparison metrics (e.g., signal-to-noise ratio, transcription accuracy) will be documented.

2.2.1 User I/O Subsystem

We are going to have a LCD display that can let the user decide what languages to translate between and some push buttons to be able to decide. We are also going to have a push button that will start listening on the microphone, then stop listening so we can ensure that all of the data has been stored. In the case that the translation lasts too long, we may add some feature to automatically stop the input of speech so we make sure not to have too much data to translate. This UI subsystem will essentially make it so that this is a usable product.

Subsystem Requirements	**Subsystem Verification**
1. The main requirements of this device would be to have a usable interface to be able to select across languages and see the selection on the screen. We hope to achieve a refresh rate on this LCD display of at least 30 Hz.	1. Functional Verification: We will ensure that we are able to send commands to the interface to be able to move between different languages and verify that the correct text appears on the display. We also want to try to run some button tests that will check if the UI is updating correctly depending on what we pressed. We can do some isolated testing for this device by connecting to a development board and writing SPI commands to the device. 2. To verify the refresh rate of this device we can use an oscilloscope to check the VSYNC signal and confirm that the frame update time is less than 1/30 Hz. 3. Finally we can test the communication interface by validating that the SPI commands received at the device are the ones that we intended to send from the STM32.

2.2.1 Power Management Subsystem

This portable power management system will use 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:

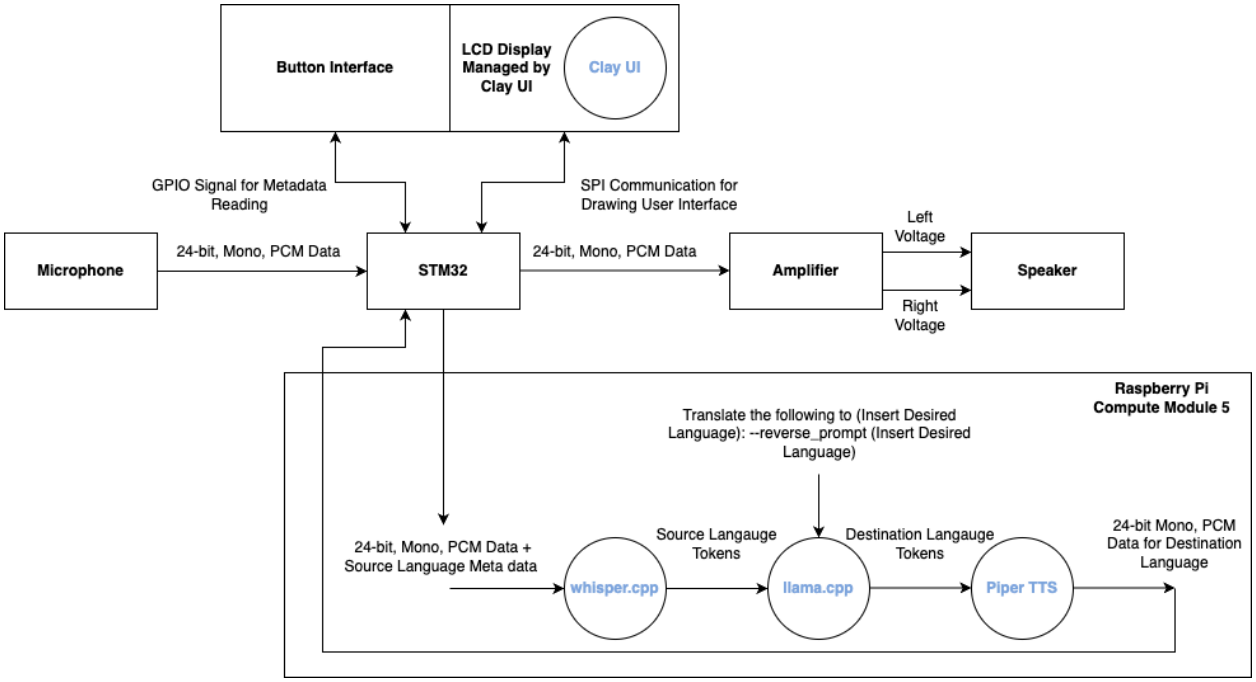
- One will step down the battery voltage to 5V.
- The other will step down the voltage to 3.3V.

Additionally, an 18650 battery holder will securely hold the battery and allow for easy replacement.

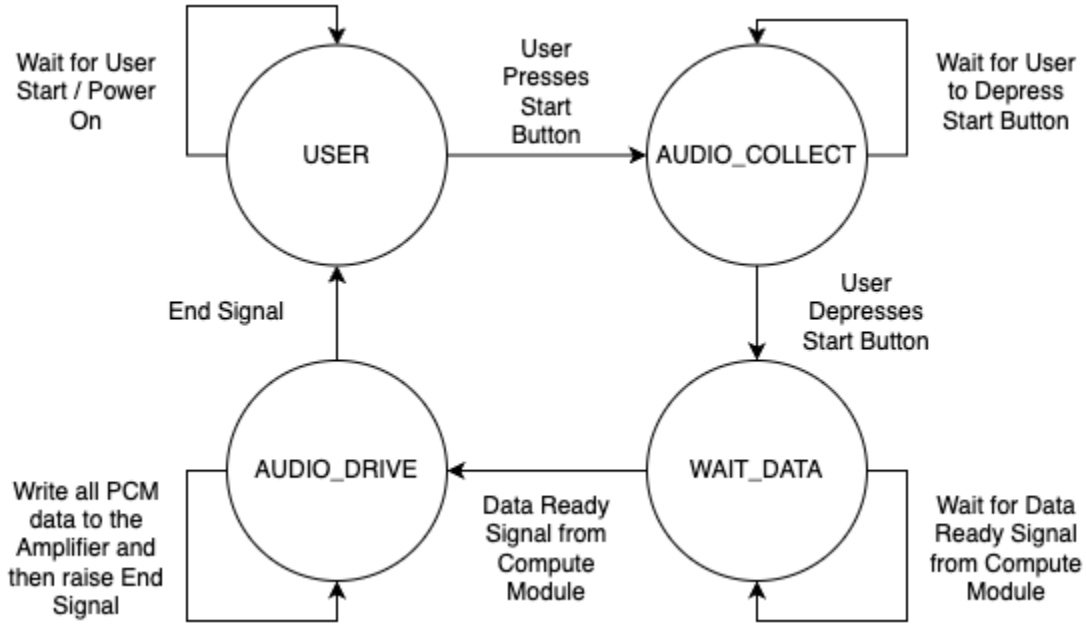
Subsystem Requirements:	**Subsystem Verification**
1. Voltage Stability: ○ The 5V and 3.3V power rails should maintain ±0.1V tolerance to their respective subsystems. 2. Current Supply: ○ The 5V regulator should supply at least 1.5A continuously to support the worst-case current draw from the Raspberry Pi CM4/5. ○ The 3.3V regulator should supply around 400mA to ensure the functionality of the MCU, LCD, and audio subsystem. 3. Thermal Management ○ The LM317 regulators should not exceed 85°C during normal operation, and heat dissipation methods should be implemented to prevent overheating. 4. Protection Mechanisms: ○ Overcurrent protection should be in place to prevent system damage if a component draws excessive power (e.g., PCB fuses).	1. Define Equipment: We will use a digital multimeter (DMM) and oscilloscope will measure voltage stability to ensure the 5V and 3.3V power rails stay within ±0.1V tolerance. 2. Define Test Procedures: The 5V regulator will be tested by applying a 1.5A load, and the 3.3V regulator will be tested with a 400mA load to verify continuous current supply. We will verify the load and line regulation of each output channel during this process. 3. Define Presentation of Results: Voltage readings will be recorded under different loads to confirm stability. Current draw measurements will ensure regulators meet required supply levels.

2.3 Software Design

The basis of the software design will be made up of data processing across audio sensors, microcontrollers, button and LCD user interfaces, and a high performance processing unit.



We can start the analysis of our software design in the STM32 where we have a notion of a state where it will be operating in USER, AUDIO_COLLECT, WAIT_DATA, and AUDIO_DRIVE states.



Through these states we can ensure that we are managing a smooth pipeline where data can transition between submodules with ease. Initially, we will wait for the user to send data to set the source and the

destination languages which will need to be managed through selection of data from the user interface. Then when the source and destination languages have been decided, we will receive a signal from the user to start collecting data from the microphone. Before we start collecting data we will send a signal to the compute module to send the language meta data and initialize it to start listening for PCM data. At this point we need to start buffering data into memory through DMA on the I2S interface. Due to memory limitations of the STM32 we will offload this data when reaching a threshold of data capacity and we will have a high speed SPI interface to be provided to the Raspberry Pi. Once the user has stopped pressing the start button, we will send an acknowledgement signal to the Pi to show the end of data. The data will be received by the compute module and it will start to transcribe the data by doing some preprocessing (Sign-extending) and it will configure the whisper.cpp framework to expect the particular language and it will pull the necessary model into memory. Once it finishes, we will get token data from the source language and we can provide that to the translation model, llama.cpp. In this case we are going to be doing the translation by prompting the model to translate text.

```
std::string model_path = "llama.cpp/models/mistral-7b.Q4_K_M.gguf";
std::string prompt = "-p Translate the following text to " + dest_langauge + ": ' " + escaped_text + " '
--reverse-prompt " + dest_langauge + ":";
```

This prompt will be able to query the llama model and extract the translated text through the reverse prompt. Then we do some setup for the context of the model for the model, and we run inference. Then we will receive the tokens for the translated language and we can use that to show debugging on the LCD interface and we can deliver that to the TTS model, Piper. Then we can configure the model to the particular language we are expecting and request that the generated PCM data should be mono and use 24-bit depth. Then the translated speech is on board the compute module and we can send over a signal to the STM32 to expect receiving the data. At this point we are going to need to do more buffering by receiving partial amounts of the data directly to memory and offloading that to the speaker. We are going to need to develop a quick method that will reduce the delay as much as possible to ensure clear output.

2.4 Tolerance Analysis

2.4.1 Translation Accuracy

Our system aims to achieve full-sentence translation with an accuracy confidence level above 90%. The entire processing pipeline must maintain data integrity to ensure reliable output. Additionally, the translated output should be delivered at a decibel level that is interpretable by the end user.

To improve accuracy, the system will incorporate a filtering mechanism to reduce the impact of input noise. This filtering will help enhance the quality of the translation and ensure that the final output remains clear and intelligible.

We also expect to attempt to improve the accuracy of the translation by reducing the potential loss of data within previous models in the pipeline by running some level of checking and requerying of the model. We will also try to use the largest model we can on board the device that can fit to provide the most amount of performance for the translation. This may require us doing some quantization of the models that we have provided but many of the open source projects that we are working with have a tiny version of their models that has been highly optimized.

2.4.2 Translation Latency

We aim to maintain a translation latency of 500 ms or less to enable real-time usability, especially for users in remote locations. To achieve this, we will:

- Limit buffer sizes within the system to reduce processing delays.
- Optimize data processing on the compute module to minimize potential spikes in latency.

2.4.3 Power Management

To ensure efficient power management, we have included the worst-case current draw for the different voltage domains.

Component (3.3V)	**Peak Current Draw (mA)**
STM32F407IGH7 (MCU)	240.00
ICS-43434 (I2S Microphone)	0.55
ST7789 (LCD Display)	90.00

Component (5V)	Peak Current Draw (mA)
Raspberry Pi Compute Module 5	1600.00
MAX98357AEWL+T	2.40
CMS-16098A-SP (Speaker)	2.75
Total	1935.70

Table 1. Worst Case Current Draw

3 Cost and Schedule

3.1 Cost Analysis

The total cost for parts, as shown in Table 2, is $199.21 before shipping. Considering a 5% shipping cost, which adds $9.96, and a 10% sales tax, which adds $19.21, the total cost will be $229.09.

Designator	Package	Quantity	Designation/Value	Price per Piece ($)
D403,D401,D404,D402	D_DO-41_SOD81_P10.16mm_Horizontal	4	1N4002	0.2
R303,R305,R306,R307,R301	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	5	10k	0.1
J201	PinSocket_1x10_P2.54mm_Vertical	1	Conn_01x10_Female	0.56
C401,C402	C_0603_1608Metric_Pad1.08x0.95mm_HandSolder	2	0.1uF	0.49
SW301,SW303,SW302	SW_SPST_B3S-1000	3	SW_Push	0.57
U401,U402	SOT-223-3_TabPin2	2	LM317_SOT-223	0.64
LS201	CUI_CMS-16098A-SP	1	CMS-16098A-SP	2.41
C309	C_0603_1608Metric_Pad1.08x0.95mm_HandSolder	1	4.7u	0.49
C307,C303,C304,C305,C308	C_0603_1608Metric_Pad1.08x0.95mm_HandSolder	5	100n	0.49
U301	UFBGA-201_10x10mm_Layout15x15_P0.65mm	1	STM32F407IEHx	12.07
U201	21-0896B_9_MXM	1	MAX98357AEWL+T	4.3
MK201	MIC_ICS-43434	1	ICS-43434	3.12
C404,C403	C_0603_1608Metric_Pad1.08x0.95mm_HandSolder	2	10u	0.49
C301,C302	C_0603_1608Metric_Pad1.08x0.95mm_HandSolder	2	20p	0.49
R201,R202	R_0805_2012Metric_Pad1.20x1.40mm_HandSolder	2	100k	0.1

	C311,C306	C_0603_1608Metric_Pad1.08x0.95mm_ HandSolder	2	1u	0.49
	R302,R304	R_0805_2012Metric_Pad1.20x1.40mm_ HandSolder	2	2k2	0.1
	C310	C_0603_1608Metric_Pad1.08x0.95mm_ HandSolder	1	10n	0.49
	J301	PinSocket_2x03_P2.54mm_Vertical	1	Conn_02x03_Odd_Even	0.56
	Y301	Crystal_HC49-U_Vertical	1	Crystal	52
	R401	R_0805_2012Metric_Pad1.20x1.40mm_ HandSolder	1	1k	0.1
	R403,R404	R_0805_2012Metric_Pad1.20x1.40mm_ HandSolder	2	330	0.1
	C406,C405	C_0603_1608Metric_Pad1.08x0.95mm_ HandSolder	2	1uF	0.49
	BT401	BAT_1048P	1	1048P	10.77
	R402	R_0805_2012Metric_Pad1.20x1.40mm_ HandSolder	1	542	0.1
	CM5	Raspberry Pi Compute Module 5	1	CM5	80
	CM5 Expansion Board	GPIO Expansion board for CM5	1	CM5 Expander	20
				Total	$199.21

Table 2. Itemized List of Components and Cost

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

Table 3. Schedule for Project Progression

3.3 Risk Analysis

The development and use of a portable translator introduces several potential risks that must be considered to ensure its reliability and usability. One major risk is hardware failure, particularly within the Main Processing Subsystem (MCU) or Secondary Processing Subsystem (Compute Module). If these components malfunction, the device may be unable to process translations accurately, leading to communication failures. Additionally, since the device operates without an internet connection, there is a risk that the translation algorithms may not be as comprehensive or up-to-date as online alternatives, potentially resulting in inaccurate translations. Another concern is audio subsystem reliability; if the microphone or speaker malfunctions, users may struggle to input or hear translations, reducing the device's effectiveness. Furthermore, since this device is meant to be portable, inefficient power consumption could lead to short battery life, limiting the device's usability during travel. To mitigate these risks, we will implement robust error-handling mechanisms, optimize power efficiency, and rigorously test the device under various conditions.

4 Discussion of Ethics and Safety

4.1 Open Source Usage

Our project is based on various open source projects. We have adapted these projects to meet our specific needs. We acknowledge our responsibility to properly attribute the original authors and comply with licensing requirements. We will ensure that all intellectual property is properly cited and comply with their licenses.

4.2 Battery Safety

To comply with industry standards, including IEEE 1725-2021 for rechargeable battery safety, we will incorporate protection mechanisms against overcharging, overheating, and short circuiting.

4.3 Translation Misuse

Our project can be prone to the sensitive nature of translating sensitive language which could cause harm to others. We intend to censor this language as much as possible to prevent from any possible misuse of this device. We intend to develop a table of commonly used sensitive words and censor appropriately if a particular word is found.

5 References

[1] Piper TTS, "Piper TTS Model," [Online]. Available: https://github.com/rhasspy/piper.

[2] G. Gerganov, "whisper.cpp," [Online]. Available: https://github.com/ggerganov/whisper.cpp.

[3] Raspberry Pi Ltd., *Raspberry Pi Compute Module 4/5 Datasheet*, [Online]. Available: https://datasheets.raspberrypi.com/cm4/cm4-datasheet.pdf.

[4] STMicroelectronics, *STM32F407 Datasheet*, [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html.

[5] TDK InvenSense, *INMP441 Datasheet*, [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf.

[6] Maxim Integrated, *MAX98357A Datasheet*, [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf.

[7] Dayton Audio, *Dayton Audio CE32A-4 Datasheet*, [Online]. Available: https://www.daytonaudio.com/images/resources/285-103-dayton-audio-ce32a-4-spec-sheet.pdf.

[8] Adafruit, *ST7789 Datasheet*, [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/2-0-inch-320-x-240-color-ips-tft-display.pdf.

[9] G. Gerganov, "llama.cpp," [Online]. Available: https://github.com/ggml-org/llama.cpp.

[10] N. Barker, "clay," [Online]. Available: https://github.com/nicbarker/clay.