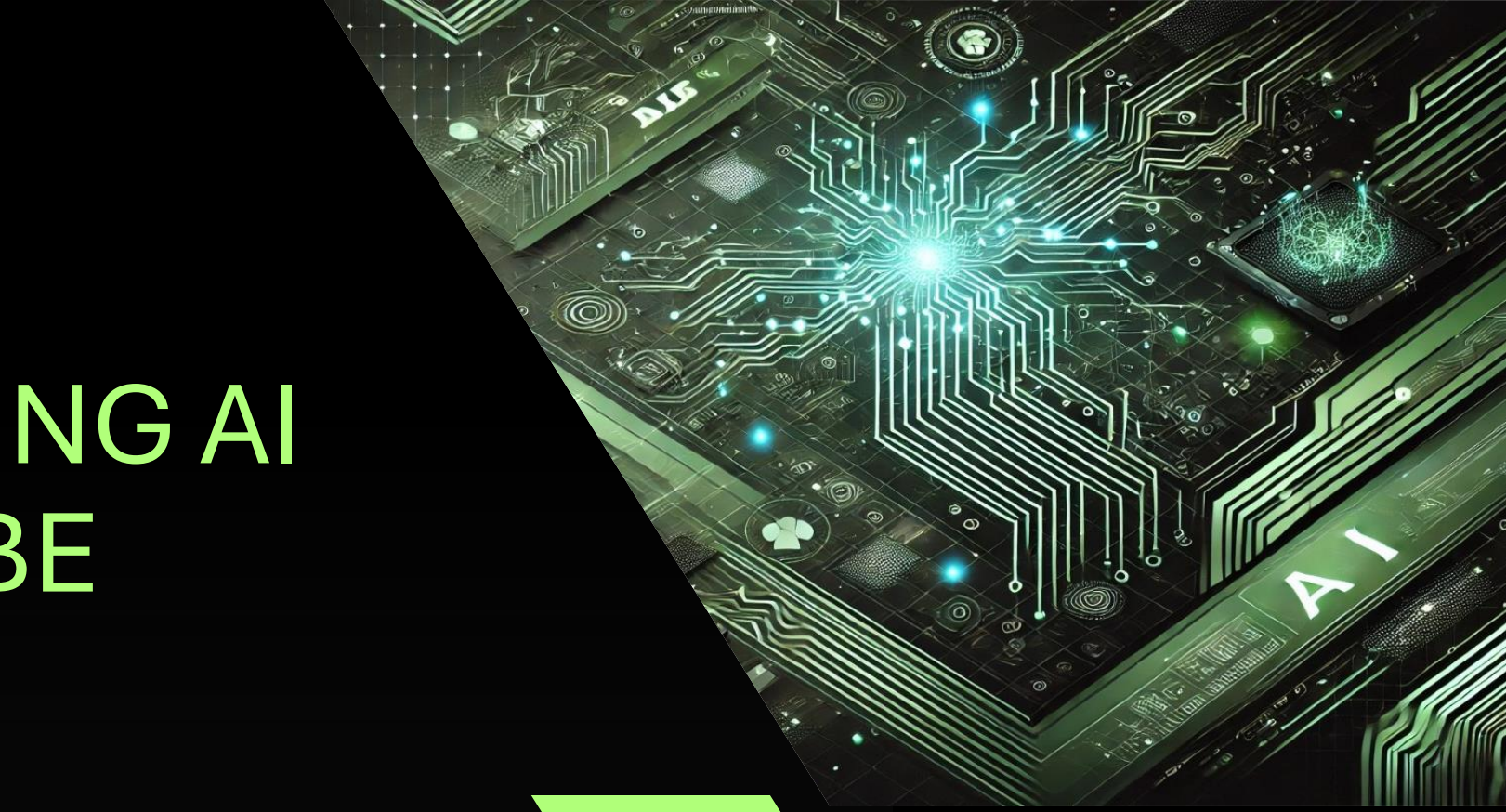
NIGHTWING



K-NIGHTWING AI TRANSCRIBE

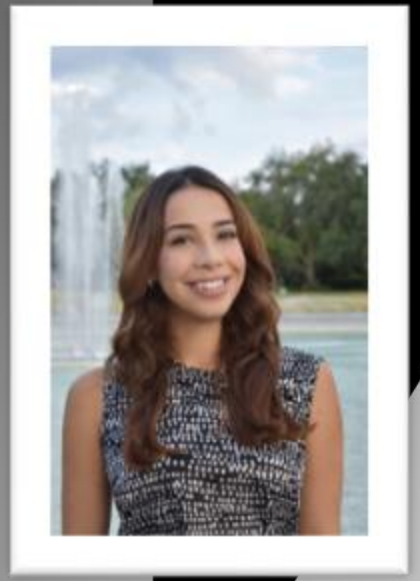
Group 24

FALL 2024 – SPRING 2025

THE TEAM

Alayna Thurner
Electrical Engineering

Sophia Demers
Electrical Engineering



Christine Hawkins
Computer Engineering

Simeon Feliz
Computer Engineering

MOTIVATION

- Our project is driven by the need for real-time, secure, and portable communication intelligence, particularly in environments where traditional cloud-based solutions are not viable
- Cloud-dependent systems introduce latency and security risks, making them unsuitable for sensitive operations.
  - Our project leverages edge computing by performing local AI-driven transcription and analysis on the Jetson Orin Nano.
  - This ensures faster processing, improved data privacy, and resilience against cyber threats.
- Traditional communication intelligence systems are bulky and require fixed installations, limiting their usability in mobile or covert operations.
  - Our system is designed to be compact, battery-powered, and discreetly housed in a backpack.
- By combining edge computing, portability, and real-time AI-driven transcription, our system addresses a growing need in cybersecurity and secure communications.

BACKGROUND

Nightwing's Sponsorship & Industry Outreach

- Nightwing, a leading cybersecurity company, is sponsoring our project to increase brand recognition among UCF students.
  - Their goal is to attract top talent by engaging students in real-world cybersecurity applications.
- Recently divested from Raytheon Technologies, Nightwing is focused on establishing its independent identity in the cybersecurity industry.
  - Supporting innovative student projects helps increase visibility and position Nightwing as a desirable employer.

Integration into Nightwing's Cybersecurity Demos

- Our project will be featured in Nightwing's client demonstrations, showcasing cutting-edge, real-time cybersecurity solutions.
  - Reinforces Nightwing's role as a leader in secure, edge-computing solutions for communication monitoring.

INTRODUCTION

- (Hypothetically) Imagine being a covert operative or intelligence agent needing real-time insight into intercepted foreign language radio chatter…
- Instant access to transcriptions and translations can be a game changer!
- Introducing K-Nightwing AI Transcribe – a fully autonomous, portable, and AI-driven system that captures, transcribes, and translates radio communications in real-time!

PROJECT OVERVIEW

Signal Reception & Processing

- Target communicates over walkie talkie
- Pluto SDR captures RF signals
- GNU Radio demodulates, filters and streams raw PCM audio data in precise chunks to the AI model

AI-Based Transcription & Metadata Generation

- Whisper AI model on Jetson transcribes and translates the speech
- Generates metadata:
  - Detected language
  - Latency
  - Timestamp
  - Confidence score
- Packages all data into a JSON file for transmission

Data Transmission & User Interface

- ESP32 on custom PCB receives JSON data via UART
- ESP32 handles:
  - Touchscreen UI: system status, battery life, start and stop commands
  - Bluetooth Transmission of JSON data to android device
- Android device displays data to user

BASIC GOALS

- The system will efficiently capture and process radio signals in real time, ensuring that the signal capture is as noiseless as possible for the most accurate processing.
- The real-time signal processing will convert radio wave signals into accurate transcriptions and offer real-time translations from a foreign language into English.
- The system will implement reliable wireless connectivity, excluding Wi-Fi, for seamless and discrete communication with a mobile device.
- The translated speech-to-text will be displayed in real time on the mobile device for the user to view.
- The device will be designed as a compact and portable unit for mobility and discreteness.

ADVANCED GOALS

- Implement AI-driven voice recognition that can differentiate between multiple speakers for better context in translations.
- The mobile application connected to the device will allow users to customize language preferences, settings, and data storage.

STRETCH GOALS

- The device will support translation between multiple languages, enhancing communication capabilities for a wider audience.
- Have the device configured to support multiple channels reception, allowing the user to select the frequency they want to tune into.
- The signal acquisition module will support both receiving and transmitting signals for bidirectional communication.
- The AI chip will generate a deepfake voice from the captured audio for applications like voice emulation.

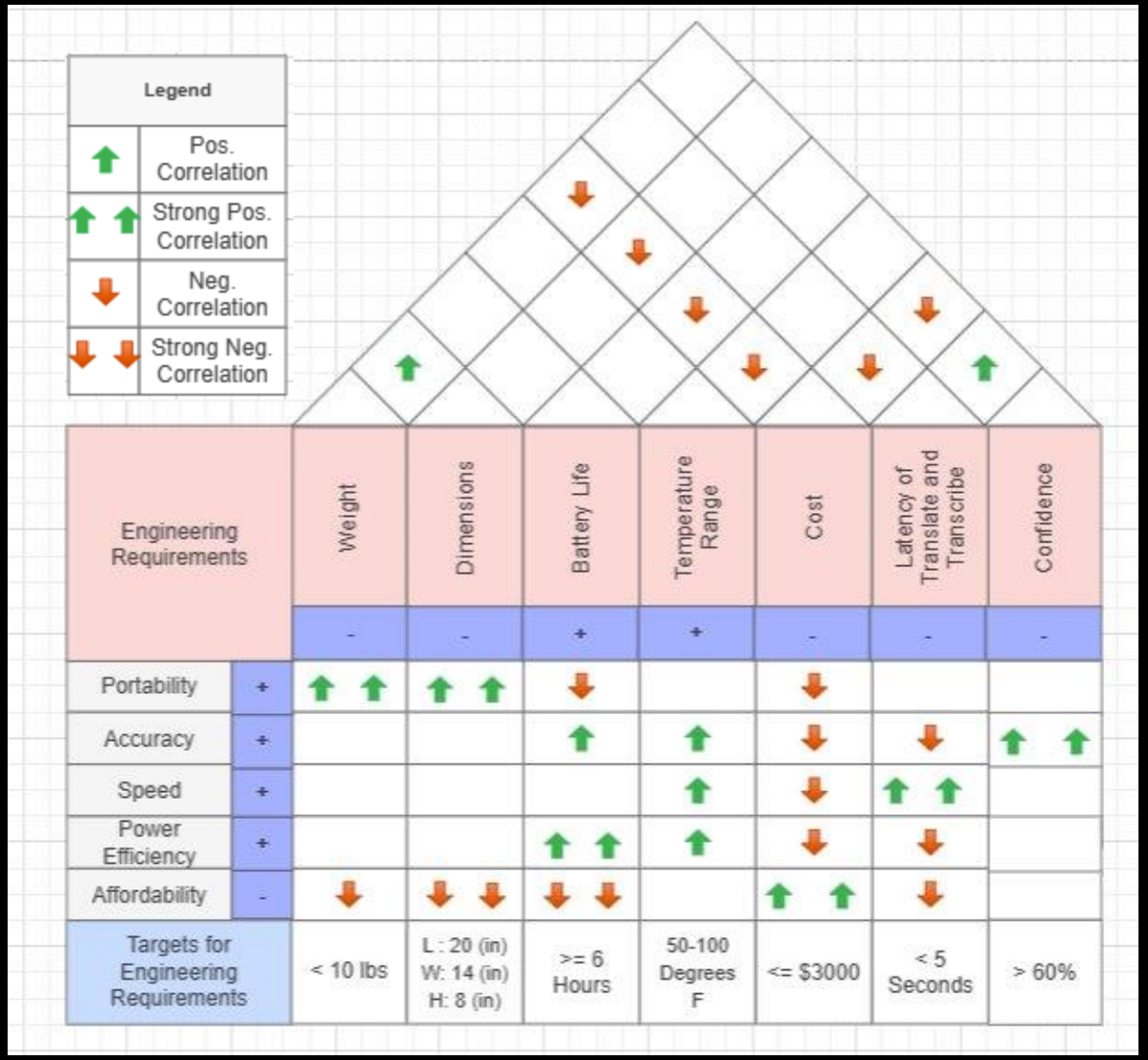
OBJECTIVES

- Implement noise-reduction algorithms for signal capture to ensure high-quality, real-time processing of radio signals with minimal interference or distortion.
- Design and deploy a real-time signal processing pipeline that streams raw PCM data into an AI module, where the signal is resampled and processed to enable accurate transcription and translation of foreign language messages into English in real time.
- Set up a discreet wireless communication module, such as Bluetooth or another suitable protocol, to ensure seamless, secure, and discrete communication between the system and the mobile device.
- Develop a mobile application interface that can display translated speech-to-text in real-time, ensuring that users can view and interact with the translated content immediately as it is received.
- Enable the mobile application to receive system status messages from the device, providing real-time updates on the system's operational state.
- Design the system's hardware and enclosure to ensure the device is portable and discreet, with a focus on ease of mobility and usability in diverse environments.

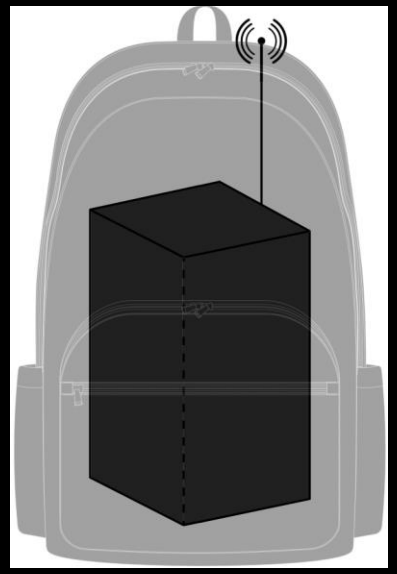
ENGINEERING SPECIFICATIONS

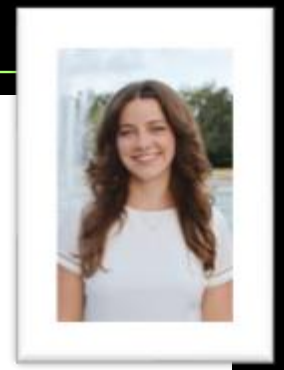
Engineering Specifications	Description	Values
Power	Amount of electrical power required for operation	Shall not exceed 40 W
Weight	Total Weight of the Device	Shall not exceed 10 lbs
Size	Dimensions of the Device (L x W x H)	Shall not exceed 20in x 14in x 8in
Battery Life	Duration of the device operates on a single charge	At least 6 hours
Temperature Range	Operating temperature of the device	50°F - 100°F
Cost	Estimated cost of production	Shall not exceed $3000
Latency of Translate and Transcribe	Time taken to process transcription to the phone	Less than 5 second
Confidence	Represents the average level of certainty derived from the AI model's analysis of the words.	The score ranges from 0.3 to 0.5, indicating a confidence level between 60% and 100%.
Bluetooth Connectivity	Connection range from the computing device to the handheld device	25 ft

HOUSE OF QUALITY



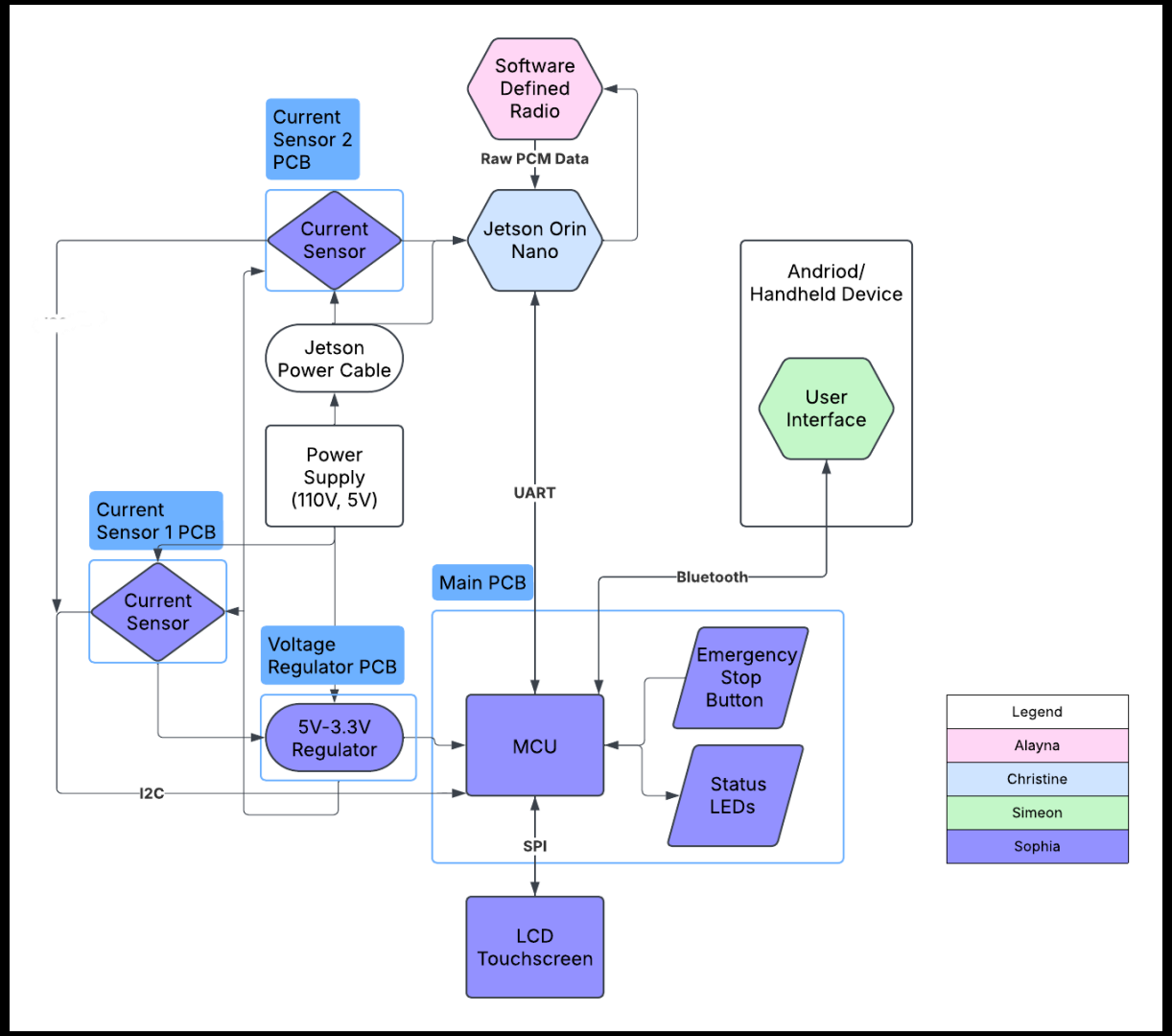
SYSTEM HARDWARE OVERVIEW





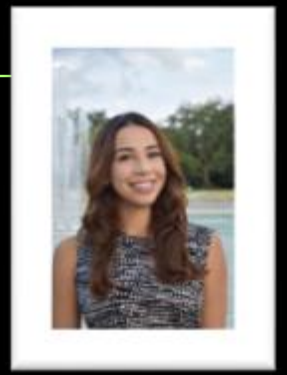
- System includes all required hardware subsystems — a software-defined radio, onboard AI processor, custom control board, power supply, and user interface — fully integrated into a portable backpack for real-time field use.

HARDWARE BLOCK DIAGRAM



COMPARISION AND SELECTION OF HARDWARE

COMPARISON OF SINGLE BOARD COMPUTERS



Feature	Jetson Orin Nano	Raspberry Pi 4 Model B	BeagleBone Black
CPU	6-core ARM Cortex-A78AE	Quad-core ARM Cortex-A72	Single-core ARM Cortex-A8
GPU	1024-core NVIDIA Ampere GPU + 32 Tensor Cores	Broadcom VideoCore VI (No CUDA support)	No GPU
AI Capabilities	Native CUDA, TensorRT, Deep Learning optimized	Requires external AI accelerators (TensorFlow Lite)	No built-in AI support
Connectivity	Gigabit Ethernet, PCIe, GPIO, I2C, SPI, UART	Bluetooth, Wi-Fi, USB, GPIO, I2C, SPI, UART	Ethernet, USB, GPIO, I2C, SPI, UART
Power Consumption	~7-15W (higher for intensive tasks)	~3-7W	~2-5W
Best Use Case	AI, machine learning, real-time signal processing	General-purpose computing projects	Hardware interfacing, embedded systems

COMPARISON OF THE NVIDIA JETSONS SERIES

Jetson Device	Performance	Power	Price	Storage Capacity	GPU's	Best For
Jetson AGX Orin Series	Delivers up to 275 TOPS	Power configurable between 15W and 60W	$1,599 - $2,299	Available in 64GB ,32GB, and Industrial versions	NVIDIA Ampere architecture with 2048 NVIDIA CUDA cores and 64 tensor cores	Demanding AI workloads and complex real-time processing
Jetson AGX Xavier Series	32 TOPS	Power configurable under 30W	$1,099 - $1,399	Available in 64GB, 32GB, and Industrial versions	NVIDIA Volta™ architecture with 512 NVIDIA CUDA cores and 64 Tensor cores	Very demanding AI tasks and heavy computational workloads
Jetson Xavier NX Series	21 TOPS	Power configurable between 10W and 15W	$399 - $699	Available in 16GB ,8GB	384-core NVIDIA Volta™ GPU with 48 Tensor Cores	Various applications needing solid performance in limited spaces
Jetson TX2 Series	1.33 TFLOPs	Provides up to 2.5X the performance of Jetson Nano	$399 - $699	Available in 4GB	NVIDIA Pascal™ Architecture GPU with 256 CUDA cores	Reliable applications needing moderate processing power
Jetson Orin NX Series	up to 100 TOPS	power configurable between 10W and 25W	$199 - $499	Available in 16GB and 8GB	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores	General applications requiring a mix of performance and efficiency
Jetson Orin Nano Series	deliver up to 40 TOPS	Power options between 7W and 15W	$99 - $199	Available in 8GB and 4GB versions	1024-core NVIDIA Ampere architecture GPU with 32 Tensor Core	Budget-sensitive projects with basic real-time processing needs
Jetson Nano	472 GFLOPS	Power consuming as little as 5W	$99 - $129	Available in 4GB versions.	NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores 0.5 TFLOPs (FP16)	Entry-level projects and simple real-time processing tasks

SDR VS TRADITIONAL RADIO

Criteria	Software Defined Radio (SDR)	Traditional Radio
Flexibility	Can adapt to multiple signal types and frequencies via software	Fixed hardware; requires physical changes for new signal types
Cost	Cost-effective in the long term; no need for hardware replacements for updates	High cost for upgrades or replacements
Ease of Upgrade	Quick software updates, even remotely	Requires physical modifications or complete replacement
Versatility	Supports various communication standards and modulation schemes	Limited to the hardware's original design
Lifetime and Future-Proofing	Long lifespan due to software adaptability	Short lifespan as hardware becomes obsolete
Signal Processing Capabilities	Can handle raw signals and output directly in digital formats (e.g., WAV)	Requires additional steps to convert signals into digital formats
Scalability	Can handle multiple signals on different frequencies simultaneously	New equipment required for expanded capabilities
Use Cases	Ideal for dynamic, evolving environments (e.g., military, emergency response)	Suitable for fixed or singular tasks

COMPARISON OF SDRS

Feature	Lime SDR LimeSDR-USB	Pluto SDR ADALM Pluto	RTL-SDR RTL2832U
Frequency Range	100 kHz – 3.8 GHz	325 MHz – 3.8 GHz	500 kHz - 1.7Ghz
Max Bandwidth	61.44 MHz	20 MHz	2.4 MHz
Transmit Capability	Yes	Yes	No
Channels	6 RX, 4 TX	1 RX, 1 TX	1 Rx
Price	$300 - $500	$150 - $250	< $30
Ease of Use	Advanced	User-friendly	Beginner-friendly
Performance	High Performance, suitable for research	High Performance for most applications	Limited due to lack of transmission
Target Audience	Advanced users, research, industry	Students, projects requiring flexibility	Hobbyists, beginners
Ideal Applications	Extensive signal processing, advanced	Projects needed balanced features	Basic Signal Receiving
Stretch Goal Readiness	Suitable for future expansion (complex)	Meets current and future project needs	Does not meet stretch goals
Software Compatibility	GNU Radio, Pothos, Lime Suite	GNU Radio, MATLAB, Simulink	GNU Radio, SDRSharp
Power Consumption	Moderate to High	Low to Moderate	Low

MCU VS FPGA

Specification	Microcontroller (MCU)	Field-programmable gate arrays (FPGA)
Cost	$2-$50	$10-1000
Power Consumption	Low Power Consumption	High Power Consumption
Customization	Limited Customization, Software-Level Only	High Customization, Software and Hardware-Level
Parallel Processing	Not Available	Available
Ease of Use	Easier to Program: Higher Level Languages	Complex to Program: Higher Level Languages and Hardware Description Languages
Applications	Best for repetitive, simple tasks	Best for complex tasks
Reprogrammability	Limited	Highly Reprogrammable
Integrated Peripherals	Built-in peripherals, timers, GPIO, communication protocols (I2C, SPI, UART)	Requires additional components for peripherals
Real-Time Processing	Suitable for less complex, real-time processing	Suitable for high complex real-time processing

COMPARISON OF MICROCONTROLLERS

Microcontroller	Wireless Connection	Processing Power (MIPS)	Clock Speed	Cost
ESP32	Wi-Fi, Bluetooth	Up to 600	240 MHz	$5-$10
MSP430	None (Requires External Module)	Up to 25	16 MHz	$2-$5
Arduino Uno	None (Requires External Module	Up to 20	16 MHz	$20-$30

COMPARISON OF LCD TOUCHSCREENS

LCD Touchscreen	Input Voltage (V)	Interface	Size (inches)	Cost
Hosyond ST7796S TFT	3.3 - 5	SPI	4.0"	$19
Adafruit TFT Touch Shield	3.3	SPI	2.8"	$35
Nextion Enhanced HMI Touch Display	5	UART	3.5"	$38

COMPARISON OF CURRENT SENSORS

Sensor Model	Measurement Method	Voltage Range (V)	Current Range (A)	Accuracy	Interface
INA219	Shunt Resistor	0 to 26	3.2	0.5%	I2C
ACS712	Hall Effect	Up to 5	5	1.5%	Analog Output (ADC)
WCS1800	Hall Effect	N/A	35	~1.0%	Analog Output (ADC)

POWER DISTRIBUTION REQUIREMENTS

Component	Supply Voltage	Current Draw	Power	Notes
Jetson Orin Nano	19 V	0.79 A	15 W	Has internal regulator to step down voltage
Pluto SDR ADALM Pluto	5V	0.5 A	2.5 W	
ESP32	3.3 V	500 mA	1.65 W	
INA219	3.3 V	<1 mA	< 0.0066 W	Two sensors
Hosyond 4.0" ST7796S TFT	3.3 V	103 mA	0.34 W	
Full Power			**19.51 W**	

COMPARISON OF BATTERIES

	Capacity (mAh)	Capacity (Wh)	Output Voltages (V)	Max Output Power (W)	Current Output (A)	Weight (kg)	Cost
Anker PowerCore III Elite	25600	96	5, 9, 15, 20 (DC)	87	Up to 3	0.58	$ 159.99
Krisdonia Laptop Power Bank	60000	222	5, 9, 12, 15, 20 (USB-A); 110V AC	130	Up to 4 (DC), ~1,2 A (AC)	1.8	$ 229.99
MAXOAK Laptop Power Bank	50000	185	5, 12, 16, 19 (DC)	130	Up to 3	1.3	$ 130

COMPARISON OF VOLTAGE REGULATORS

	Input Voltage (V)	Output Voltage (V)	Max Output Current (A)	Switching Frequency
MP2338 (Monolithic Power Systems)	4.5 to 28	0.5 to 16	3	450 kHz
LM3671 (Texas Instruments)	2.7 to 5.5	0.8 to 3.3	0.6	2 MHz
TPS62085 (Texas Instruments)	2.5 to 6	0.8 to 6	3	2.4 MHz
SY80008C (Silan Microelectronics)	2.5 to 5.5	0.6 to 5	3	1.5 MHz

COMMUNICATION PROTOCOLS

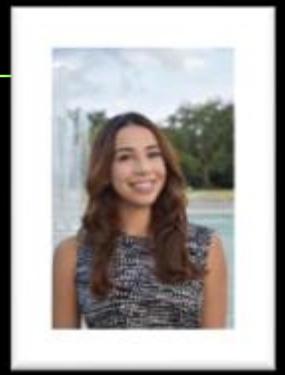
- ESP32 <-> Mobile Device: Bluetooth
- SDR -> Jetson: TCP
- Jetson <-> ESP32: UART
- INA219 -> ESP32: I2C

CONSIDERATION OF ENCLOSURE TYPES AND FEATURES

- Heat management → enclosure needs to alleviate negative heat-based impacts
  - Primary considerations:
    - » Vented enclosure to disperse heating
    - » Placement of components within enclosure to evenly disperse heat
    - » Backpack material needs to be breathable
- Inconspicuous and Mobile → enclosure must be portable and easy to hide
  - Primary Considerations:
    - » Lightweight 3D print
    - » Optimized spacing of components to reduce bulk

COMPARISON AND SELECTION OF SOFTWARE

COMPARISON OF ASR FRAMEWORKS



Framework	Optimized for Low-Resource Devices	Offline Functionality	Real-Time Processing	Translation	Scope	Documentation /Support	Compatibility with Jetson Orin Nano
Vosk	Yes	Yes	Yes (Streaming API)	No	Full speech-to-text transcription	Good, with active community and resources	Yes
Jetson-Voice	Partially (designed for Jetson devices)	Limited	Yes	Limited language support	Primarily command recognition, smaller models	Limited documentation, challenging integration	Yes
NVIDIA Riva	No	Limited	Yes	Multilingual	High-performance ASR	Strong for cloud, but resource-heavy	Limited
Whisper (Open AI)	No	Yes	Yes	Yes Multiple languages	high-accuracy transcription and translation in a wide range of languages	Good documentation and an active developer community	Yes
Picovoice	Yes	Yes	Limited	Limited	Focuses on wake word detection and keywords	Limited, with smaller community	Yes

COMPARISON OF SIGNAL PROCESSING SOFTWARE'S

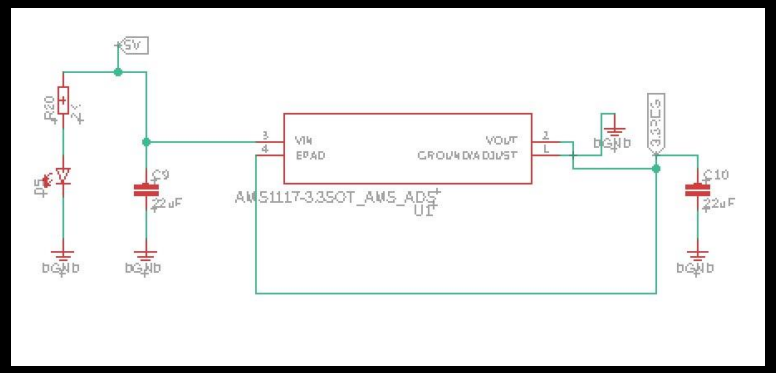
Feature	GNU Radio	MATLAB/Simulink
Cost	Free, open source	Paid license, additional expensive toolboxes
Ease of Use	GUI with drag and drop interface, easy for beginners	Easy to use but requires toolboxes for SDR functionality, MATLAB coding
Flexibility	Highly customizable with Python and C++ support for custom blocks	Less flexible, proprietary, reliant on predefined toolboxes
Community and Support	Active open-source community, frequent updates, extensive forums and documentation	Large commercial support, non open-source
Real-Time Performance	Better for real time applications due to low latency	Real time performance supported; higher latency caused by toolboxes
Compatibility with Pluto SDR	Seamless integration, frequent pairing in industry	Supports but requiers communication toolbox
Programming Language	Python or C++	MATLAB
Documentation	Extensive and free	Comprehensive but requires paid access

COMPARISON OF IDES FOR THE ESP32

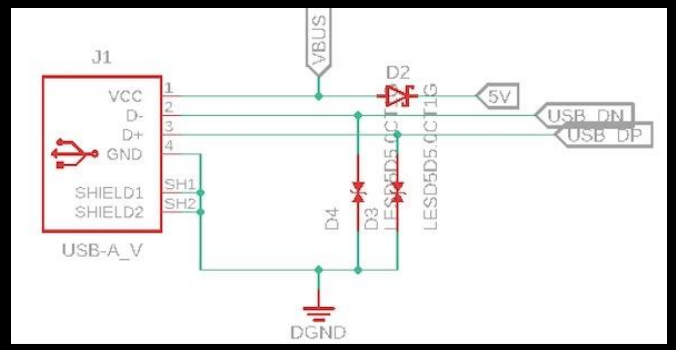
IDE	Familiarity Ranking	Difficulty of Board Integration	Documentation Repository
Arduino IDE	Second	Easy	Large
VS Code	First	Medium	Large
Espressif ESP-IDF	Third	Easy	Moderate

HARDWARE DESIGN

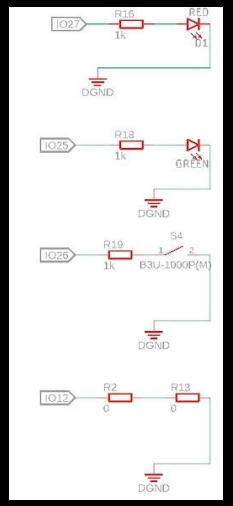
OVERALL SCHEMATIC – MAIN BOARD



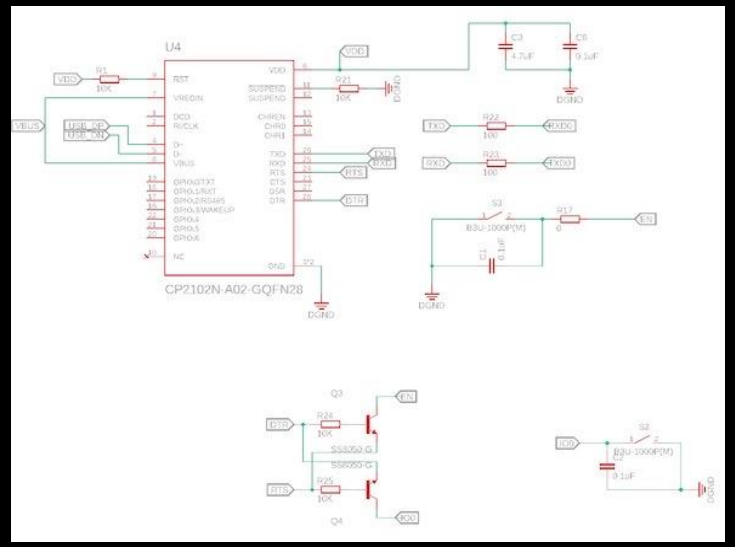
Linear Voltage Regulator 5V-3.3V



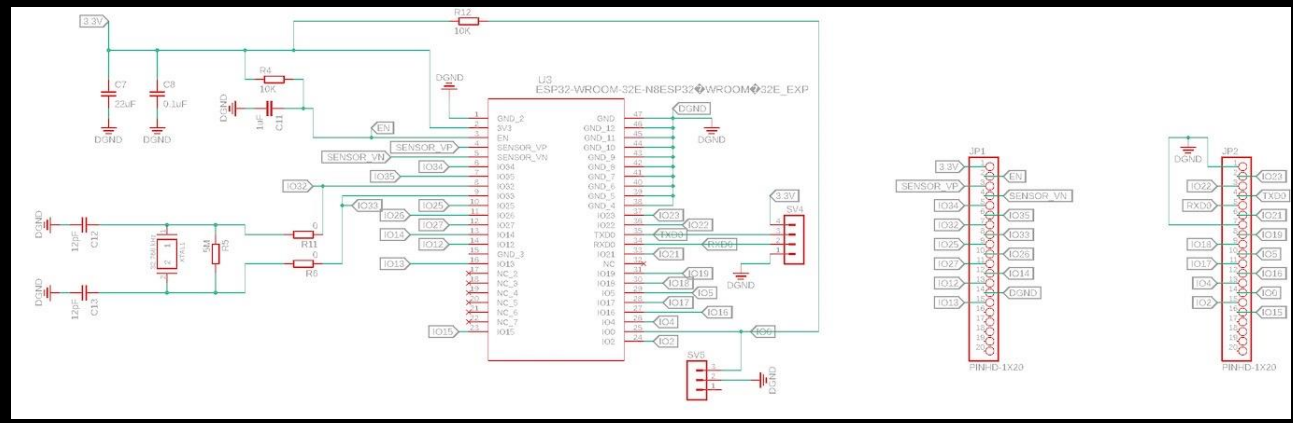
USB-A Connector



Status LED and Shutdown Button

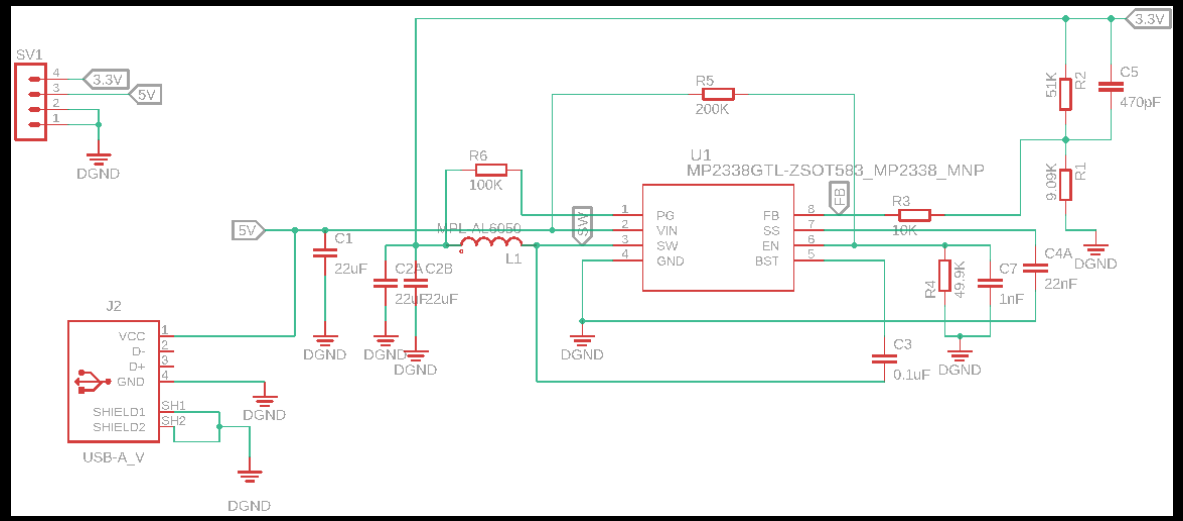


USB to UART Bridge

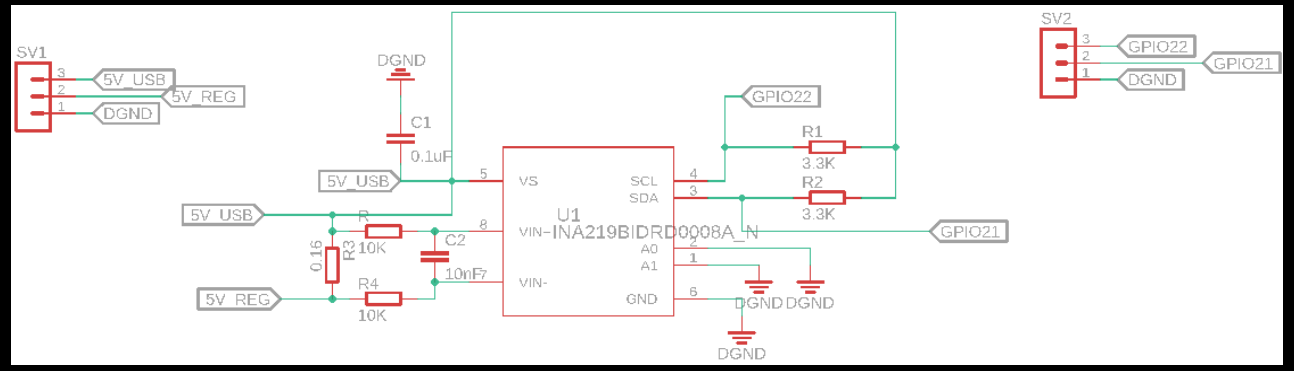


ESP32-WROOM-32E

OVERALL SCHEMATIC – VOLTAGE REGULATOR AND CURRENT SENSOR BREAKOUT BOARDS



Voltage Regulator



Current Sensor(s)

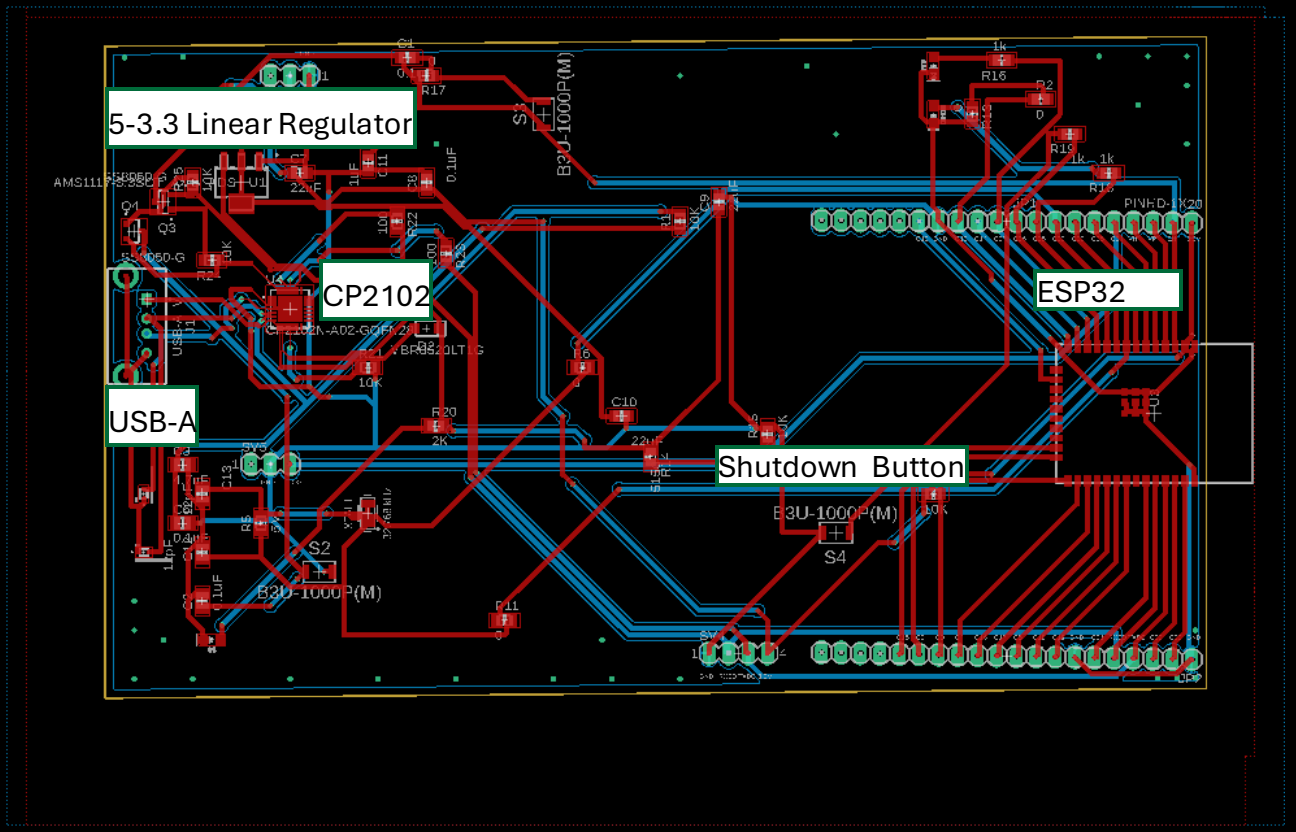
PCB DESIGN

SIGNIFICANT PCB DESIGN - PERIPHERALS

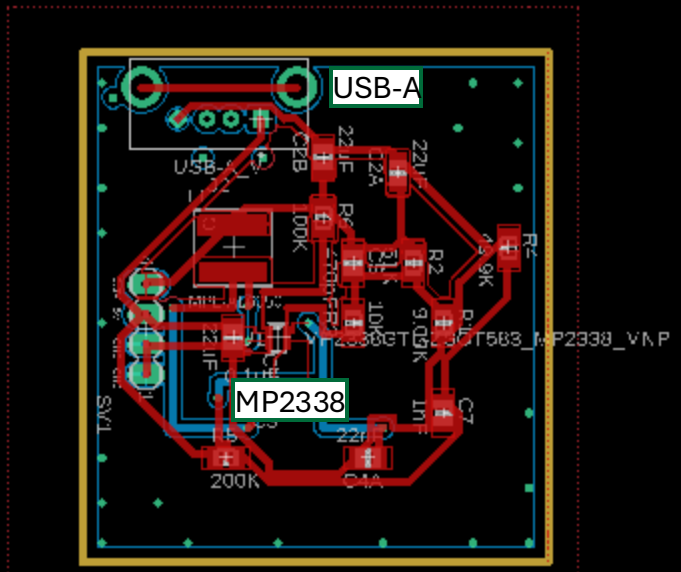
Peripherals	Interface with MCU	Purpose
Touchscreen LCD	SPI	To start and stop the system, Allow user to input starting battery percentage (needed for power consumption measuring)
Current/Power Sensor (Jetson)	I2C/ Analog	Monitors power consumption of Jetson for system diagnostics
Current/Power Sensor (ESP32)	I2C/ Analog	Monitors power consumption of ESP32 for system diagnostics
Emergency Stop Button	GPIO	Allows manual shutdown through MCU logic
UART Status LED	GPIO	Indicates UART Line is Connected to Jetson
USB to UART Bridge	UART	Enable serial communication between USB port and ESP32

PCB LAYOUT

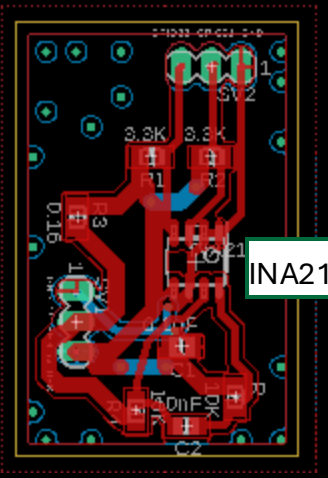
Main PCB



Voltage Regulator PCB

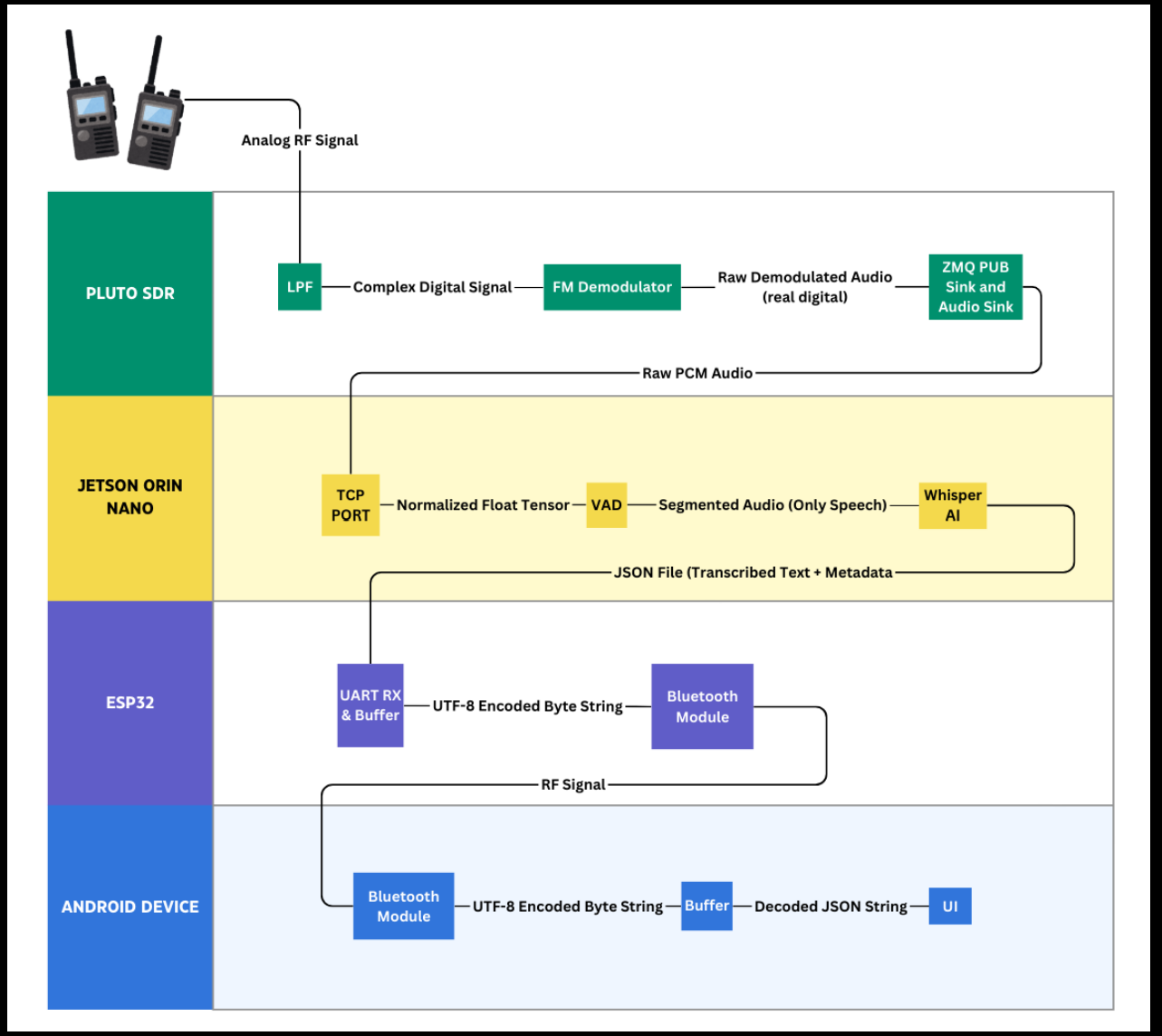


Current Sensor PCB x 2

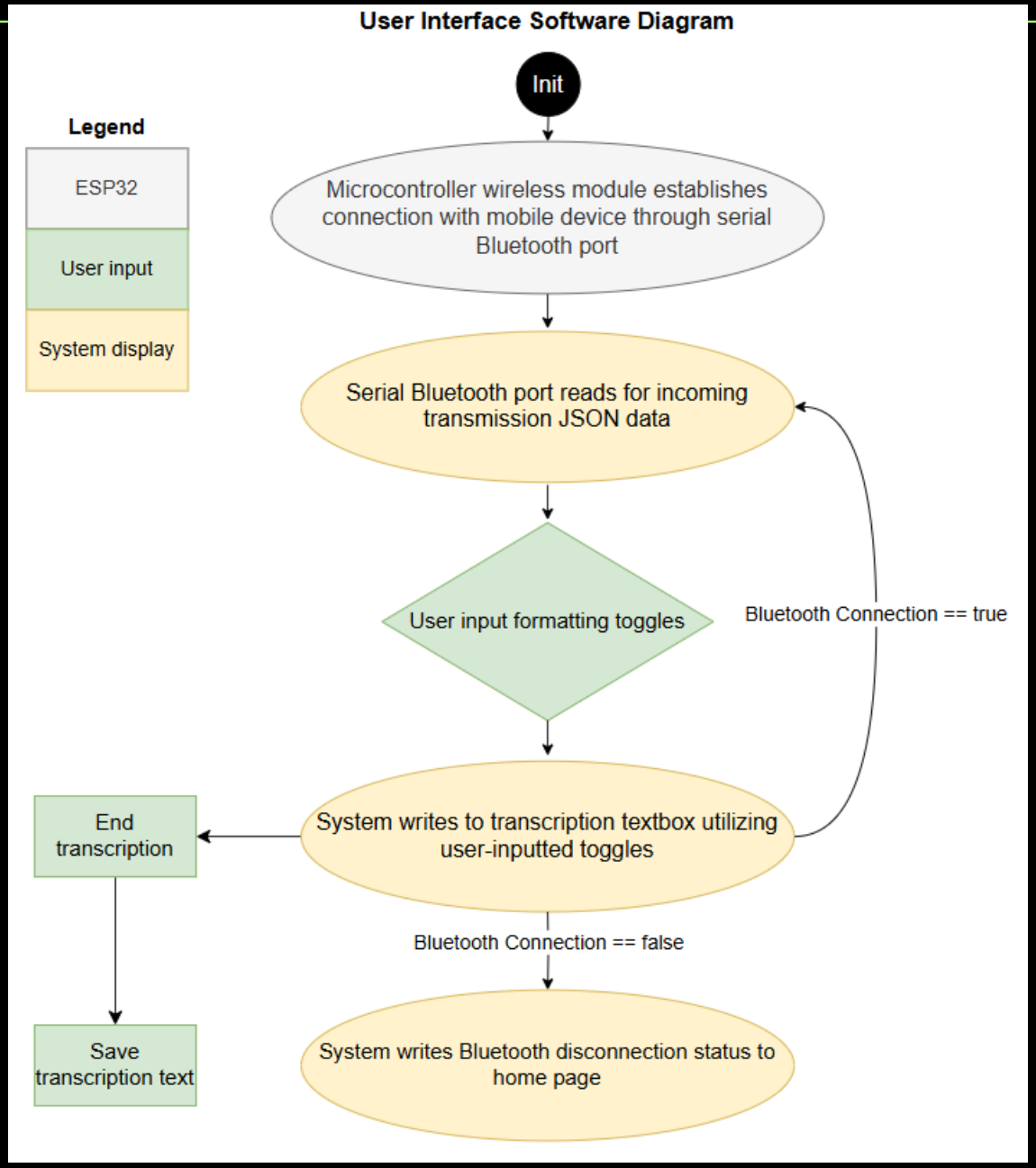


COMMUNICATION DESIGN

SIGNAL FLOW

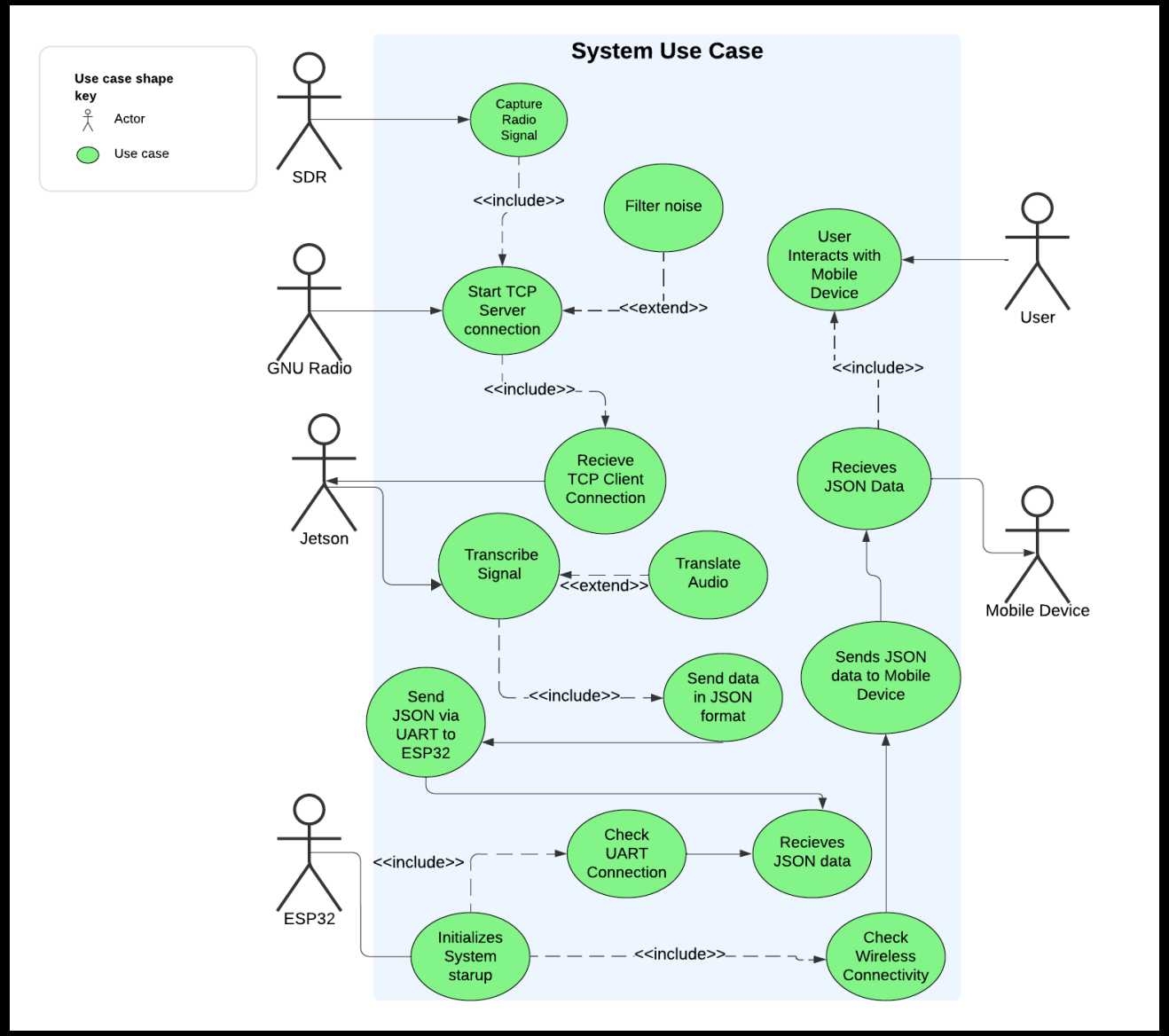

Analog RF Signal
PLUTO SDR
LPF
Complex Digital Signal
FM Demodulator
Raw Demodulated Audio (real digital)
ZMQ PUB Sink and Audio Sink
Raw PCM Audio
JETSON ORIN NANO
TCP PORT
Normalized Float Tensor
VAD
Segmented Audio (Only Speech)
Whisper AI
JSON File (Transcribed Text + Metadata
ESP32
UART RX & Buffer
UTF-8 Encoded Byte String
Bluetooth Module
RF Signal
ANDROID DEVICE
Bluetooth Module
UTF-8 Encoded Byte String
Buffer
Decoded JSON String
UI

ANDROID UI SOFTWARE FLOWCHART

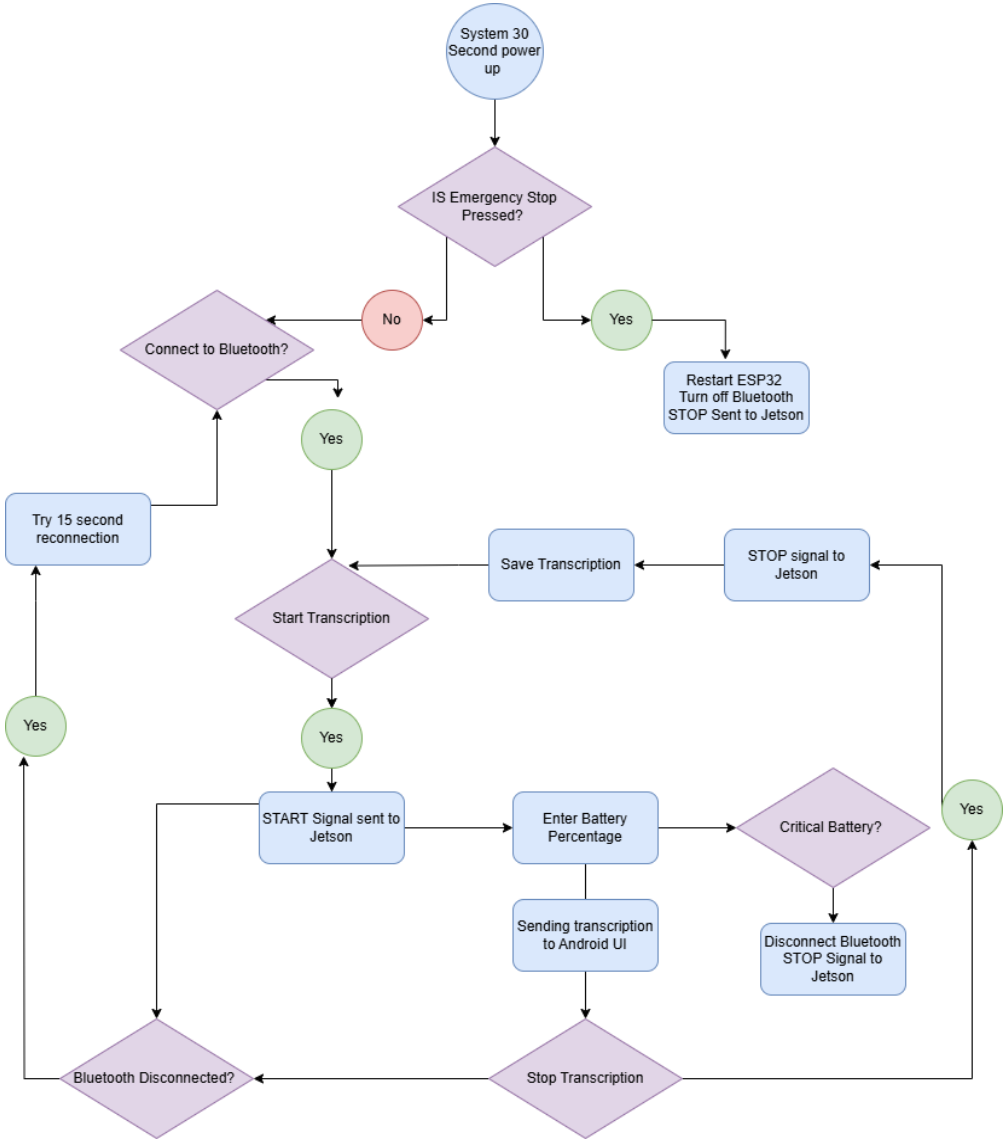


SOFTWARE DESIGN

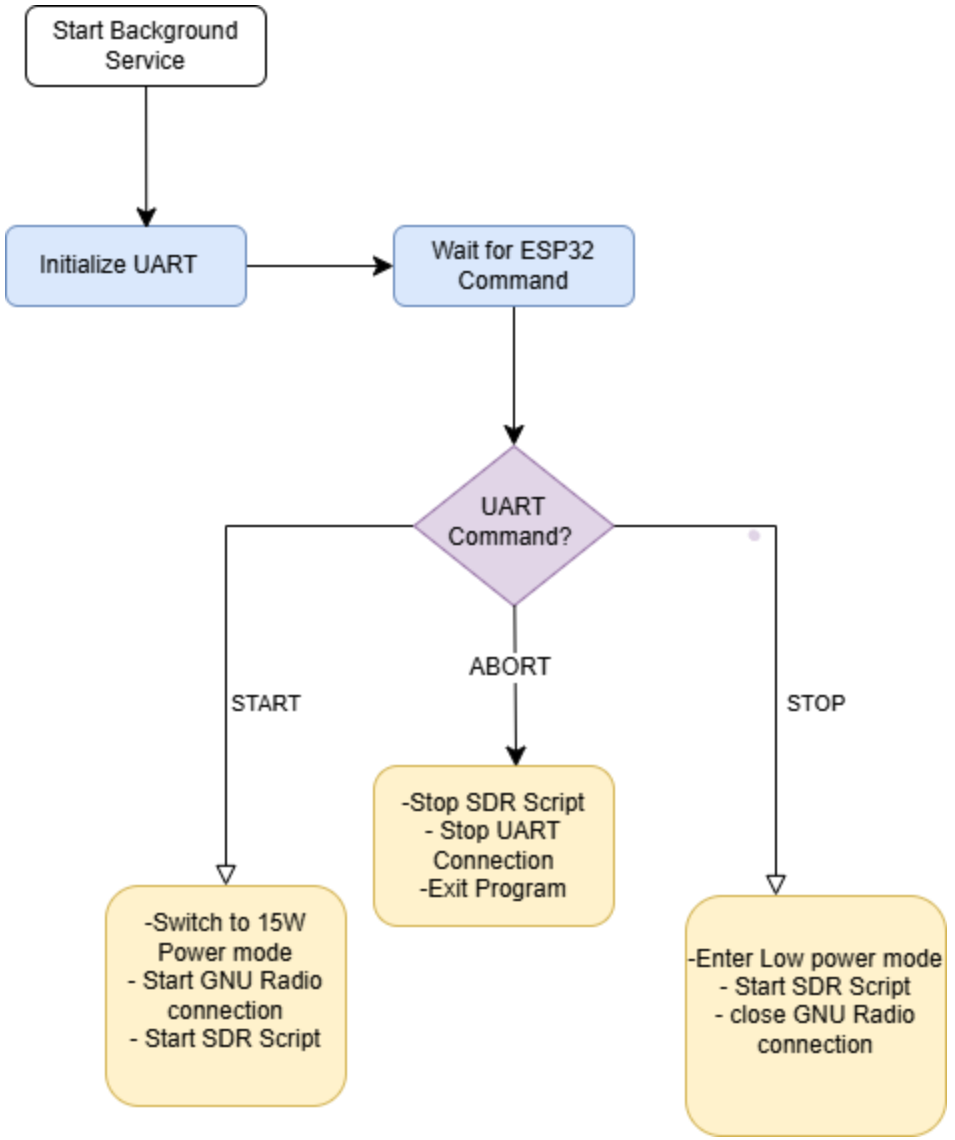
SOFTWARE USE CASE DIAGRAM



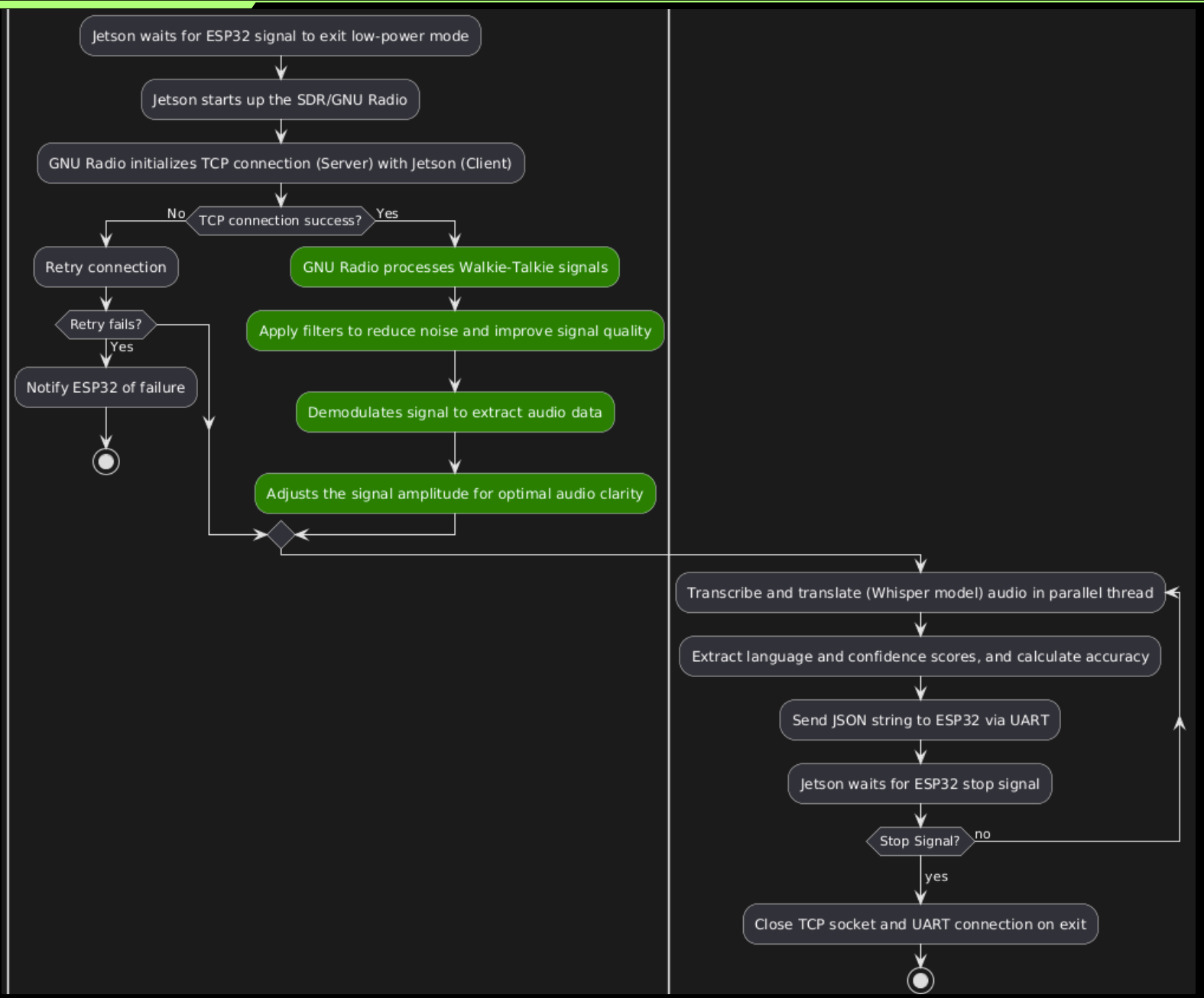
LCD/ ESP32 SOFTWARE FLOWCHART



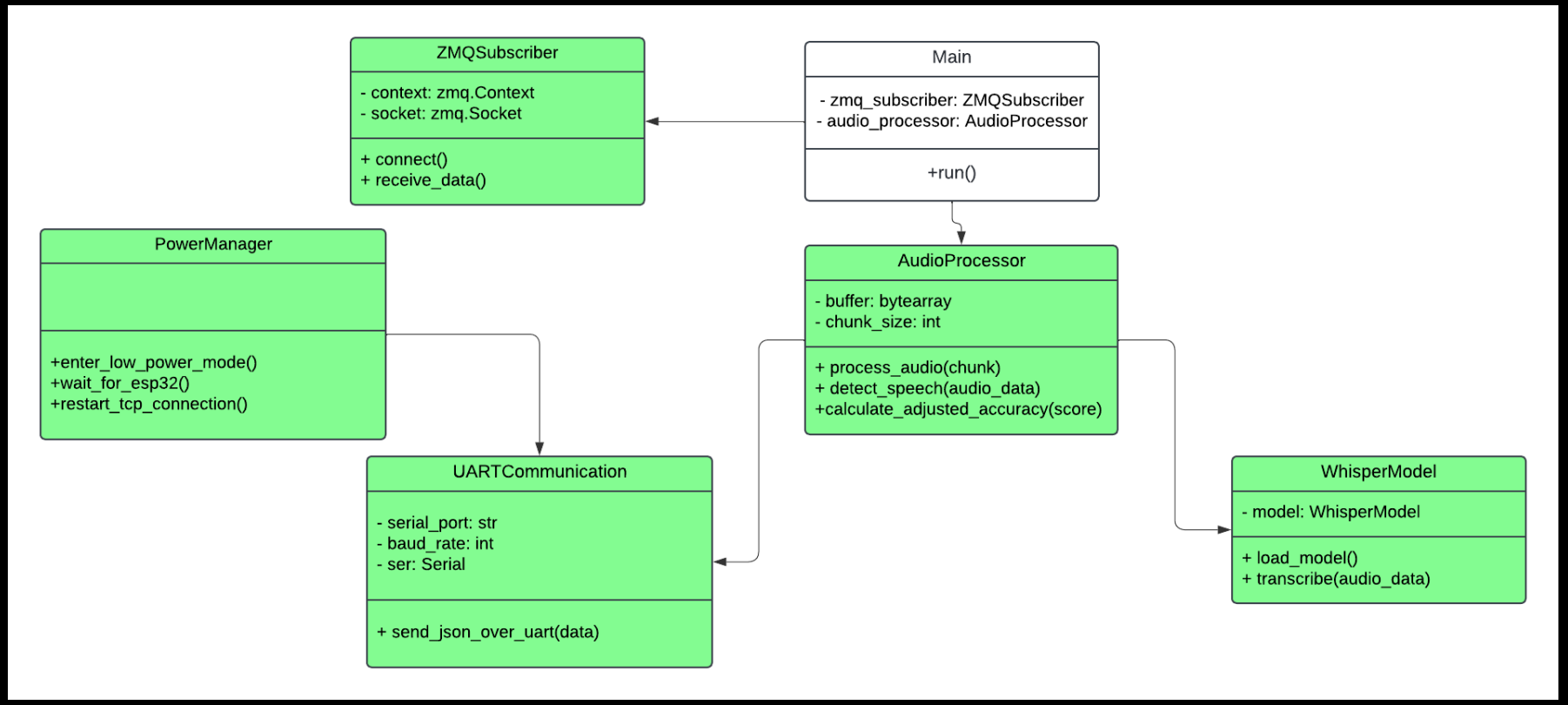
JETSON/ ESP32 SOFTWARE FLOWCHART



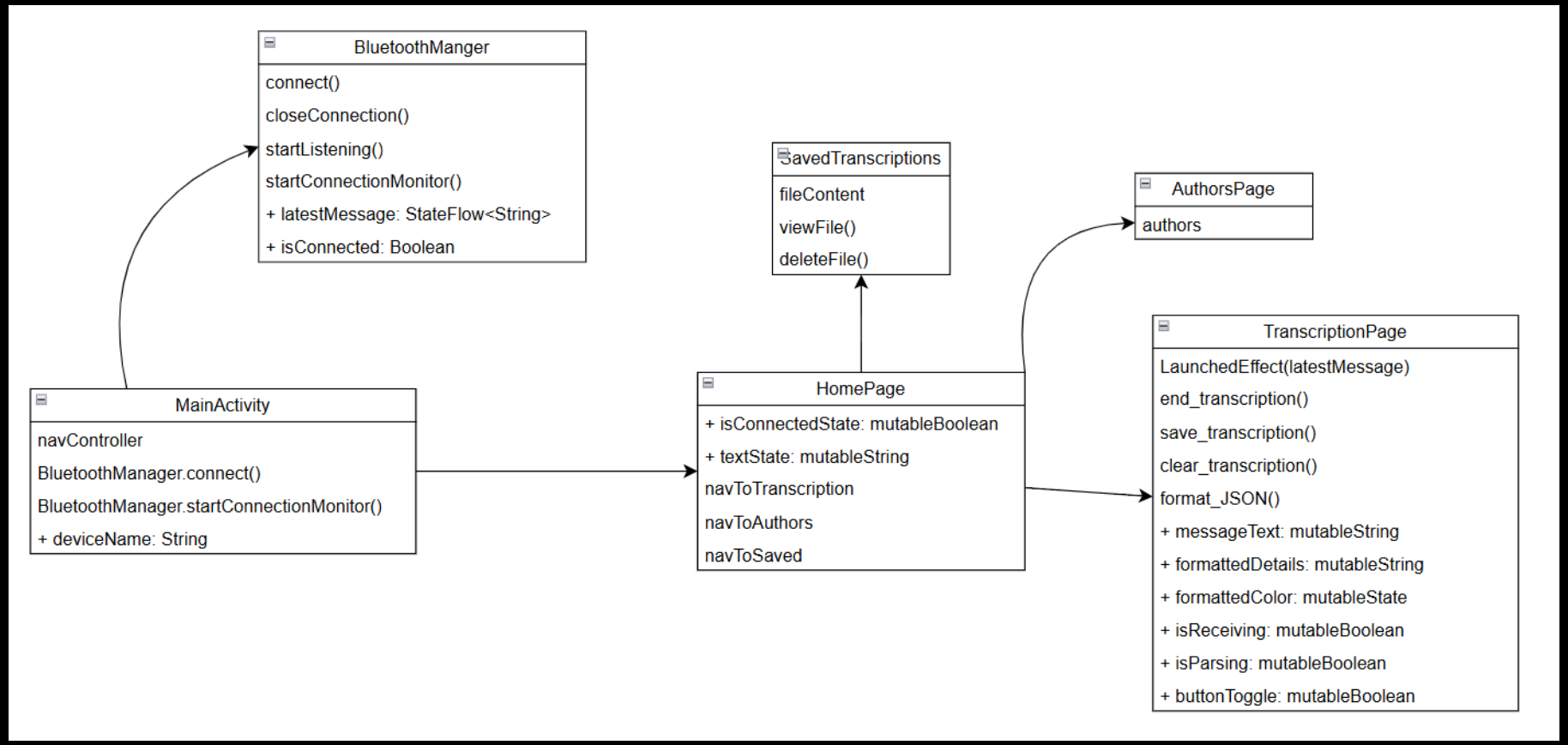
JETSON/SDR FLOW CHART



JETSON UML CLASS DIAGRAM



UI UML CLASS DIAGRAM



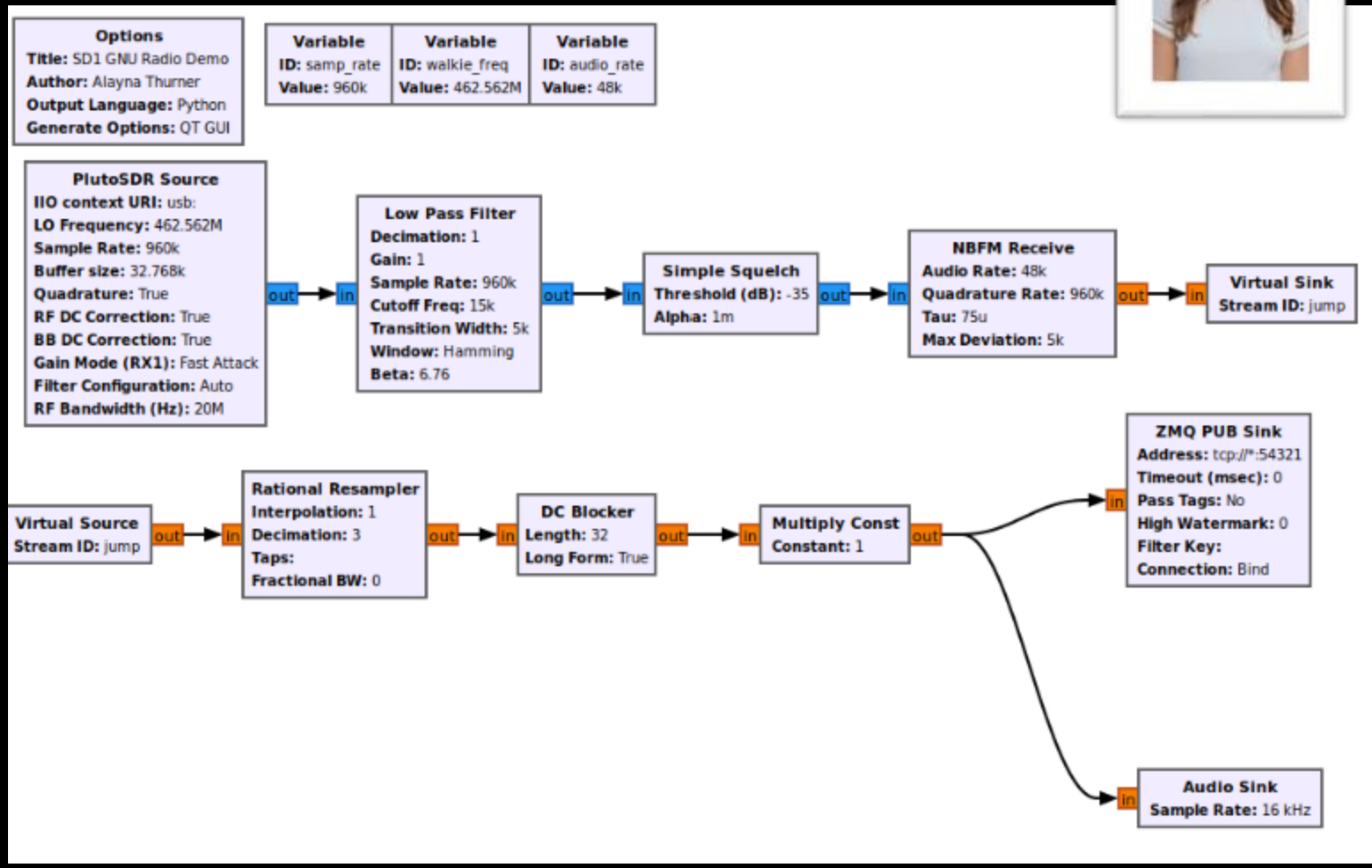
SOFTWARE PROTOTYPE AND TESTING

BLUETOOTH CONNECTION DISTANCE TESTING

Test	Distance Traveled Before Disconnection (ft.)
1	> 25
2	> 25
3	> 25

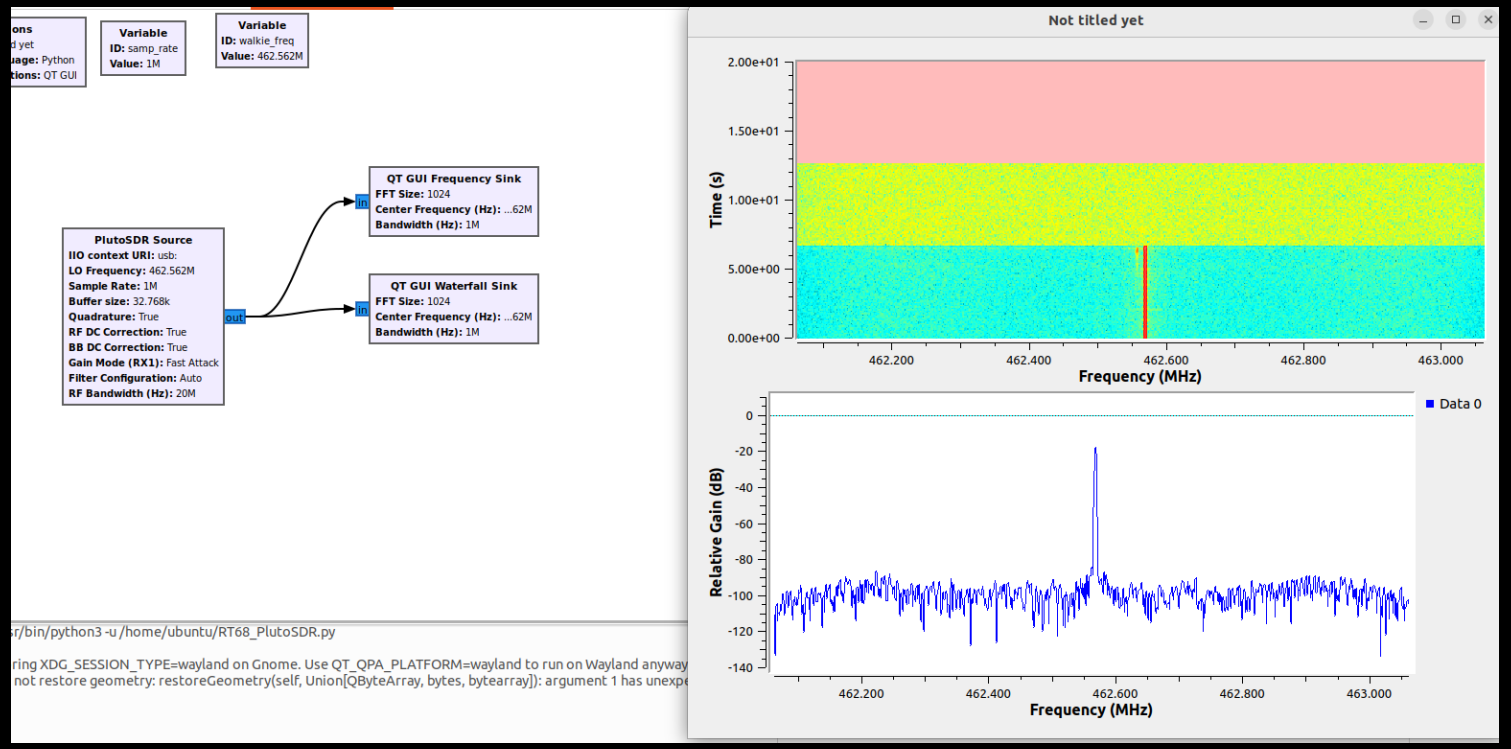
GNU RADIO FLOWGRAPH FOR REAL TIME SIGNAL CAPTURE AND STREAMING

- This flowgraph is responsible for capturing RF signals from the software-defined radio and preparing them for AI transcription.
  - Receives RF input via SDR source block
  - Applies filtering and demodulation to extract clean audio
  - Converts to raw PCM format
  - Streams output as data chunks to Jetson Nano for AI processing
- The entire process runs on the Jetson to reduce latency and ensure the system operates independently of cloud-based tools.



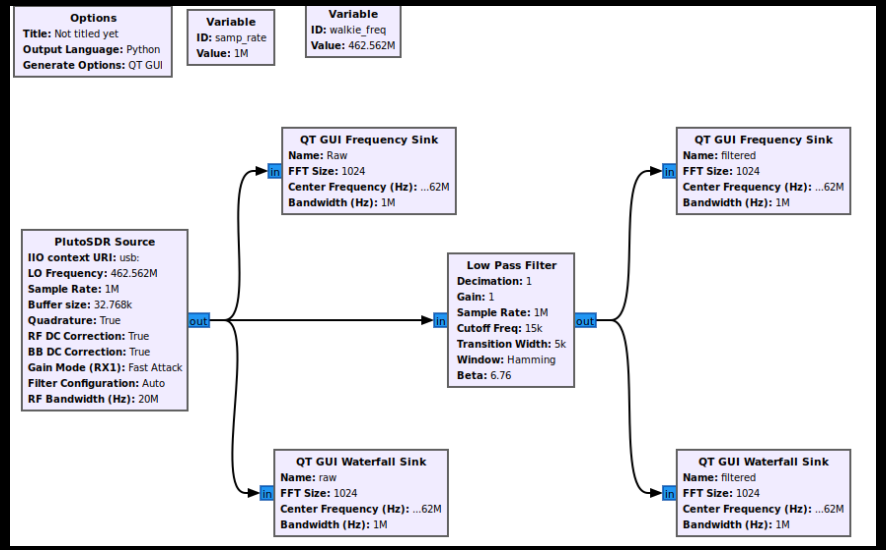
GNU RADIO – TEST 1: CONFIRM SIGNAL RECEPTION

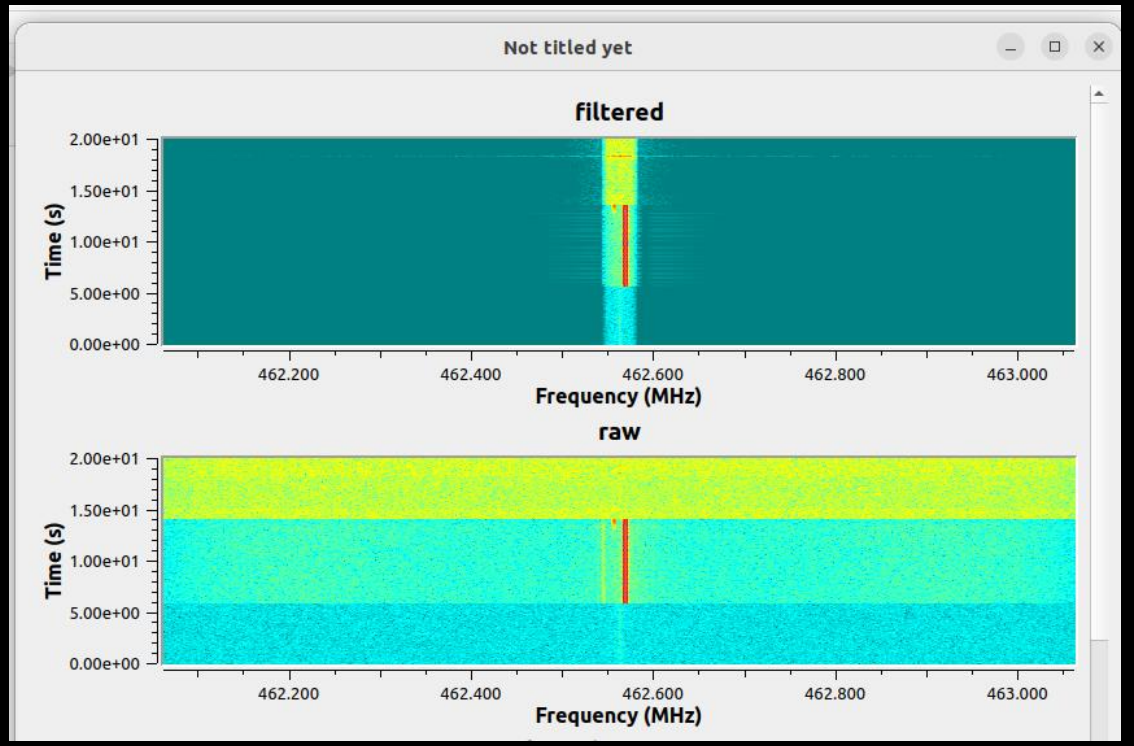
- PlutoSDR source block configured for walkie-talkie signal reception at 462.5625 MHz.
- Frequency and waterfall sinks confirm successful capture and stability.



GNU RADIO – TEST 2: FILTER OUT NOISE

- Low Pass Filter applied to isolate the target transmission and reduce surrounding wideband noise. Visual output shows a cleaner, more focused signal with improved contrast against the noise floor.

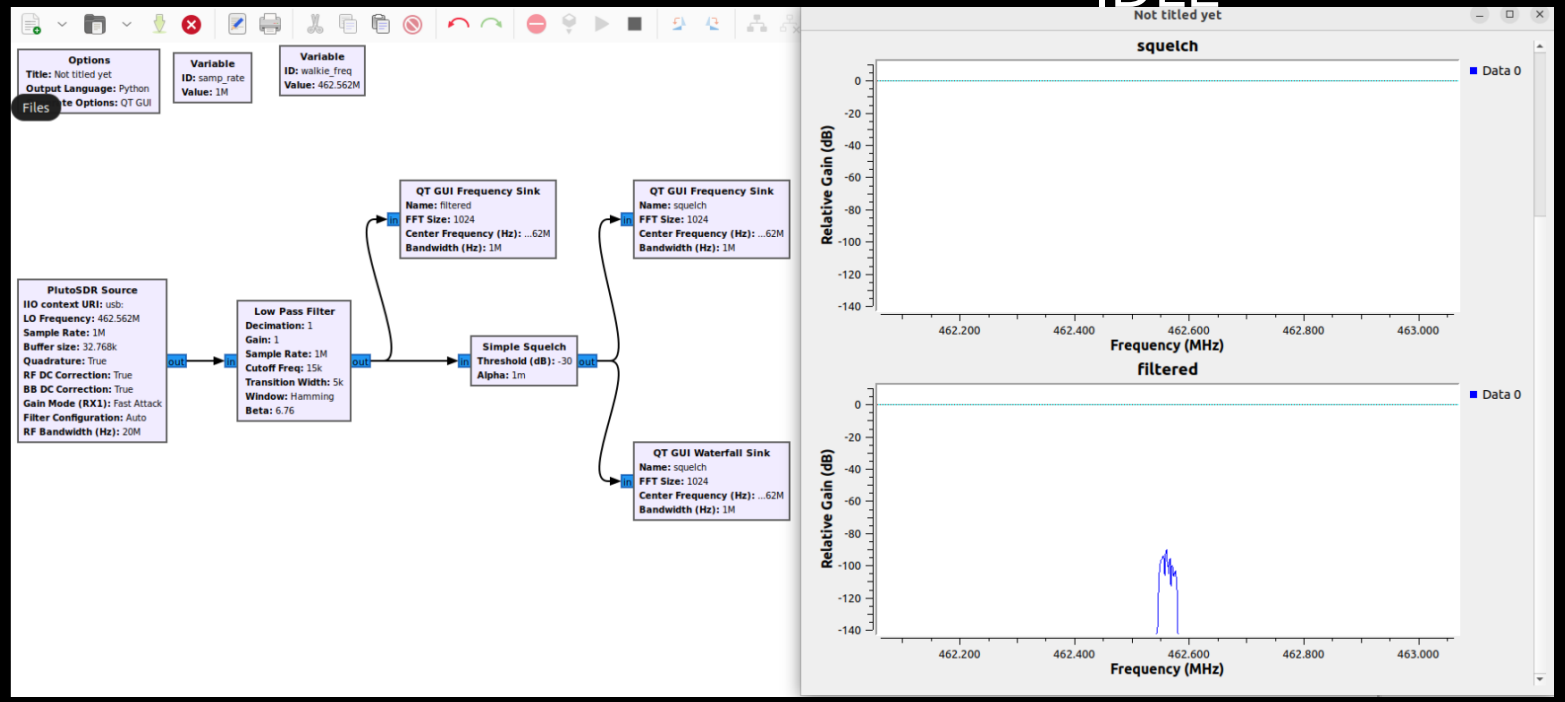




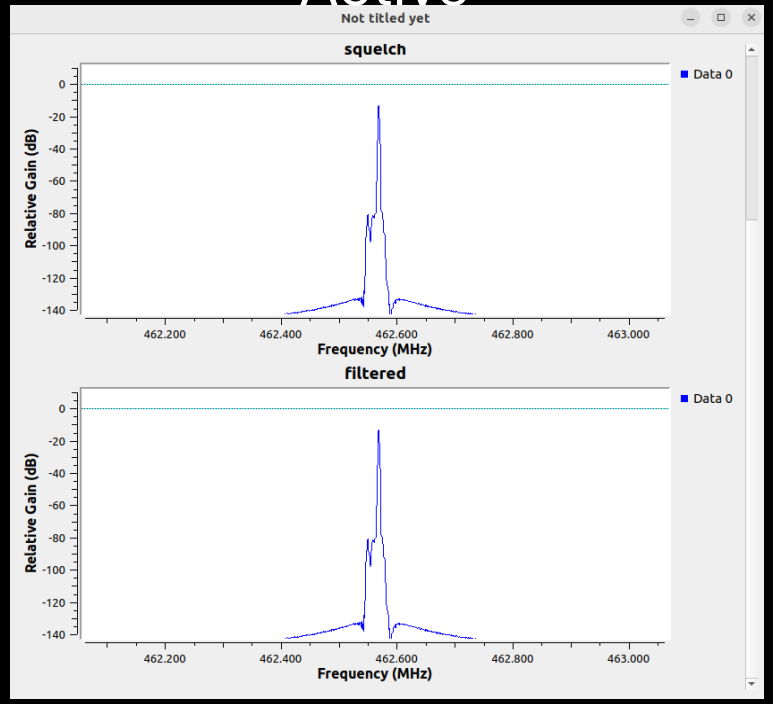
GNU RADIO - TEST 3: BLOCK LOW POWER IDLE NOISE

- A Simple Squelch filter was added to eliminate background noise when no transmission is present. The output remains clean during idle periods while preserving active signals.

IDLE

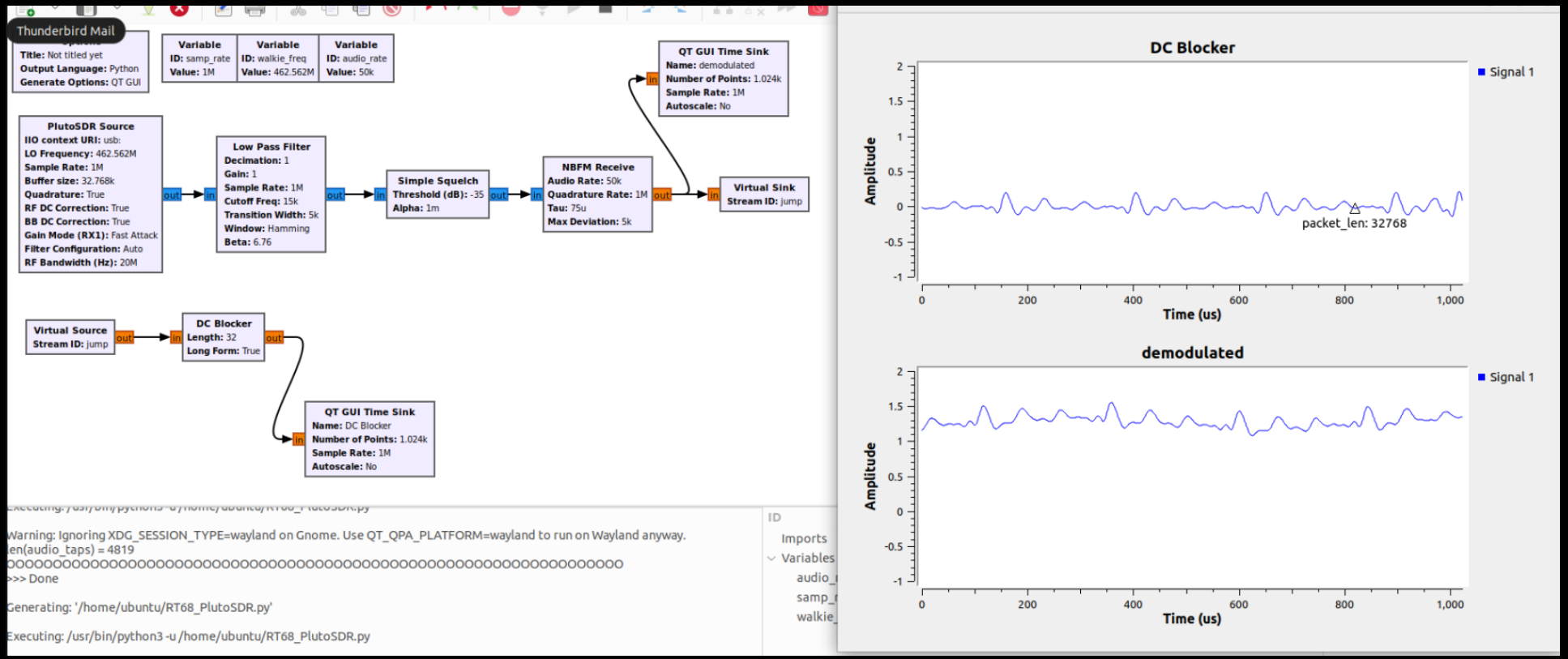


Active



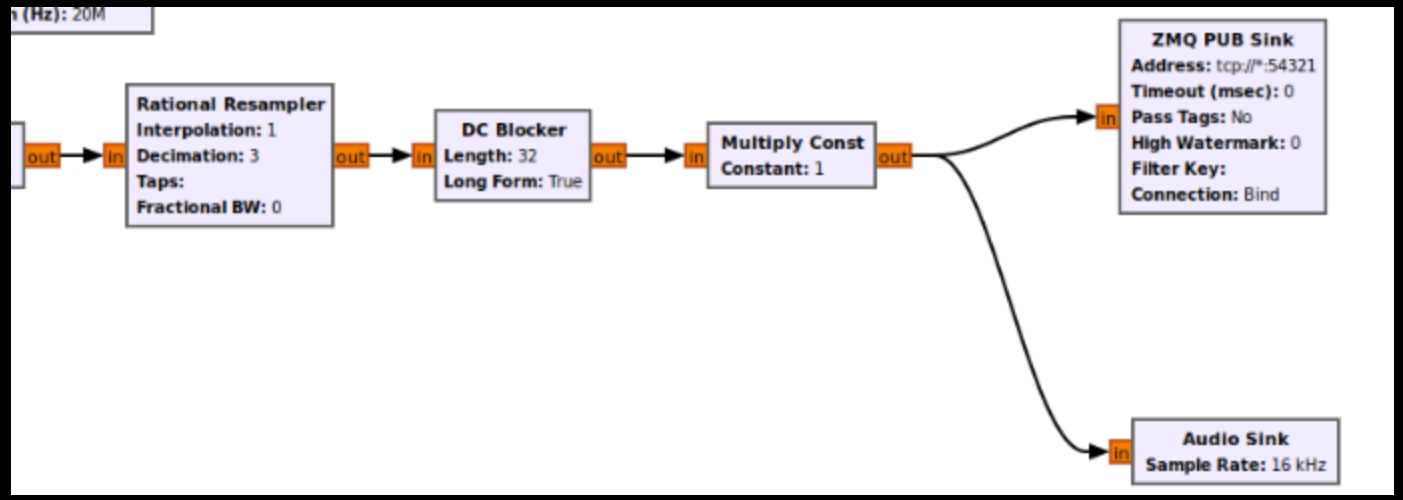
GNU RADIO – TEST 4: DEMODULATE AND CENTER

- The NBFM Receiver block demodulates the FM signal into audio. A DC Blocker was added to remove bias and center the waveform for clean playback.



GNU RADIO – TEST 5: RESAMPLE AND AMPLIFY

- The received audio was resampled to 16 kHz to meet Whisper model input requirements. Resampling was done in GNU Radio to reduce Jetson processing load. A Multiply Constant block was added to boost volume for clearer playback.



LATENCY TEST 1-3

SAME SENTENCE FOR LATENCY TESTING
ELIMINATES VARIABILITY FROM SENTENCE LENGTH, COMPLEXITY, AND PRONUNCIATION

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-End Latency	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Testing one two three, can you hear me?	3.55	3.552	0.002	0.935	0.937	Clean transmission
2	Testing one two three, can you hear me?	7.167	7.172	0.005	0.938	0.943	Clean transmission
3	Testing one two three, can you hear me?	10.946	10.951	0.005	0.832	0.837	Clean transmission
4	Testing one two three, can you hear me?	14.774	14.78	0.006	0.952	0.958	Clean transmission
5	Testing one two three, can you hear me?	19.091	19.098	0.007	0.889	0.896	Clean transmission

DIFFERENT SPEAKER
TESTING WITH DIFFERENT VOICES ENSURES THAT LATENCY REMAINS CONSISTENT REGARDLESS OF INDIVIDUAL SPEAKER CHARACTERISTICS.

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-End Latency	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Testing one two three, can you hear me?	3.778	3.778	0	0.965	0.965	
2	Testing one two three, can you hear me?	7.524	7.527	0.002999999998	0.969	0.972	
3	Testing one two three, can you hear me?	11.285	11.287	0.002	0.954	0.956	
4	Testing one two three, can you hear me?	15.176	15.178	0.001999999999	0.954	0.956	
5	Testing one two three, can you hear me?	19.002	19.004	0.002	0.989	0.991	

BACKGROUND NOISE
TESTING LATENCY WITH BACKGROUND NOISE CONFIRMS THAT THE SYSTEM CAN STILL OPERATE IN REAL TIME WITHOUT ADDED DELAY DUE TO ENVIRONMENTAL INTERFERENCE.

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-To_End	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Testing one two three, can you hear me?	3.592	3.595	0.003	0.938	0.941	
2	Testing one two three, can you hear me?	7.433	7.435	0.002	1.01	1.012	
3	Testing one two three, can you hear me?	11.171	11.186	0.015	1.3	1.315	
4	Testing one two three, can you hear me?	14.968	14.986	0.018	1.2	1.218	
5	Testing one two three, can you hear me?	18.3662	18.381	0.0148	1.295	1.3098	

LATENCY TESTS 3-5

Spanish Language

The system supports translation and multilingual transcription. It's important to verify that latency stays within acceptable bounds when processing non-English speech, which reflects multilingual field deployment.

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-To_End	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Probando uno dos tres, ¿me escuchas?	3.189	3.21	0.021	0.826	0.847	
2	Probando uno dos tres, ¿me escuchas?	7.277	7.297	0.02	0.964	0.984	
3	Probando uno dos tres, ¿me escuchas?	11.116	11.133	0.017	1.104	1.121	
4	Probando uno dos tres, ¿me escuchas?	14.93	14.947	0.017	1.078	1.095	
5	Probando uno dos tres, ¿me escuchas?	18.93	18.95	0.02	0.825	0.845	

Distance

Users may not always speak close to the walkie-talkie. Testing latency when the input signal is weaker or less clear ensures that the system remains responsive even under degraded input conditions.

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-To_End	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Testing one two three, can you hear me?	3.689	3.69	0.0009999999995	0.879	0.88	5 feet
2	Testing one two three, can you hear me?	7.549	7.554	0.005000000001	0.835	0.84	15 feet
3	Testing one two three, can you hear me?	11.34	11.343	0.003	0.955	0.958	30 feet
4	Testing one two three, can you hear me?	15.061	15.065	0.004	0.962	0.966	40 feet
5	Testing one two three, can you hear me?	18.841	18.845	0.004	0.966	0.97	50 feet

Fast Speech

In high-pressure or emergency scenarios, users often speak quickly. Measuring latency in these cases ensures that the system can keep up with fast-paced communication without lagging or failing to deliver timely resul

Run #	Sentence	Jetson Timestamp (log_timestamp)	ESP32 Receive Time (s)	End-To_End	Processing Latency (s)	Calculated Latency (s)	Notes (optional)
1	Testing one two three, can you hear me?	3.863	3.863	0	0.969	0.969	
2	Testing one two three, can you hear me?	7.337	7.338	0.001	0.98	0.981	
3	Testing one two three, can you hear me?	11.475	11.477	0.002000000001	0.936	0.938	
4	Testing one two three, can you hear me?	14.783	14.785	0.002000000001	0.967	0.969	
5	Testing one two three, can you hear me?	14.785	14.787	0.002000000001	0.977	0.979	

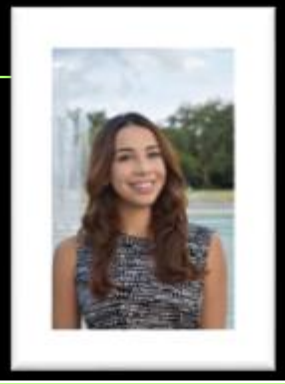
LATENCY

- **Definition of Latency**
  - Latency = Time from **end of transcription** to the moment transcribed result is **sent over UART**.
  - Transcription only starts when **speech is detected**, reducing unnecessary processing.
- **Speech Detection & Processing**
  - Voice activity detection triggers Whisper model.
  - **start_time** is captured when processing a transcribed segment begins.
  - **end_time** occurs when JSON is sent over UART (+0.002s offset).
- **UART Handling**
  - UART transmission to mobile (ESP32/Android) is **threaded** and incurs **negligible delay**.
  - Latency stops once the data is **handed off to UART**.
- **Extra Metadata**
  - Timestamps also mark when **speech was first detected**.
  - This offers **context** separate from backend latency.

Test Scenario	Avg Latency	Std. Deviation	Max Latency	Min Latency
Baseline	0.9142	0.04888455789	0.958	0.837
Different Speaker	0.968	0.01450861813	0.991	0.956
Background noise	1.15916	0.1729815828	1.315	0.941
Spanish language	0.9784	0.1313613337	1.121	0.845
distance	0.9228	0.05920472954	0.97	0.84
fast speech	0.9672	0.01723948955	0.981	0.938
Average Latency:	0.98496	0.07403005195	1.056	0.8928333333

TESTING CONFIDENCE

- avg_logprob = Average log probability of transcribed segments.
  - Measures the model's confidence in its transcription.
  - Higher values (closer to 0) → Higher confidence.
  - Lower values (e.g., -1.0) → More uncertainty and potential errors.



Test Type	Example Words/Phrases	Expected Confidence	Actual Confidence	Expected Confidenc	Actual Confidence
Simple Words	Good morning	-0.2 to -0.4	-0.318	80-100%	85.28%
Simple Words	Thank you	-0.2 to -0.4	-0.261	80-100%	92.38%
Simple Words	How are you?	-0.2 to -0.4	-0.388	80-100%	76.56%
Longer Sentences	The weather is nice and sunny outside.	-0.3 to -0.5	-0.341	70-90%	82.36%
Longer Sentences	She is studying engineering at the university.	-0.3 to -0.5	-0.448	70-90%	68.98%
Complex Words	Onomonopia	-0.5 to -0.7	-0.568	40-70%	54.02%
Complex Words	Entrepreneur	-0.5 to -0.7	-0.585	40-70%	51.86%
Complex Words	Artificial Intelligence	-0.5 to -0.7	-0.691	40-70%	38.66%
Foreign Language	Gracias	-0.5 to -0.7	-0.607	40-70%	49.15%
Foreign Language	Hola, como estas	-0.5 to -0.7	-0.512	40-70%	61.04%
Nonsense Words	Zarglonis	-0.8 to -1.0	-0.797	10-30%	25.39%
Nonsense Words	Hoggle Throp	-0.8 to -1.0	-0.87	10-30%	16.25%

TESTING CONFIDENCE BY SPEAKER

- Speaker #1

Run #	Sentence	avg_logprob	Adjusted Confidence (%)	Transcription Match?	Notes
1	Testing one two three.	-0.42	83	✅	Short + familiar words (Baseline)
2	“Frequency is set to 146.52 megahertz.”	-0.705	41.5	✅ Partially	Technical vocabulary
3	“I need backup at checkpoint three immediately, do you copy?”	-0.38	88.82	✅	Long + structured command
4	Probando uno dos tres, ¿me escuchas?	-0.46	77.05	✅	Multilingual test
5	“The quick brown fox jumps over the lazy dog.”	-0.67	47.095	✅	Phonetically diverse sentence
6	“Hey… um, I think it’s working now… maybe?”	-0.18	100	✅	Informal + disfluencies
			72.91083333		

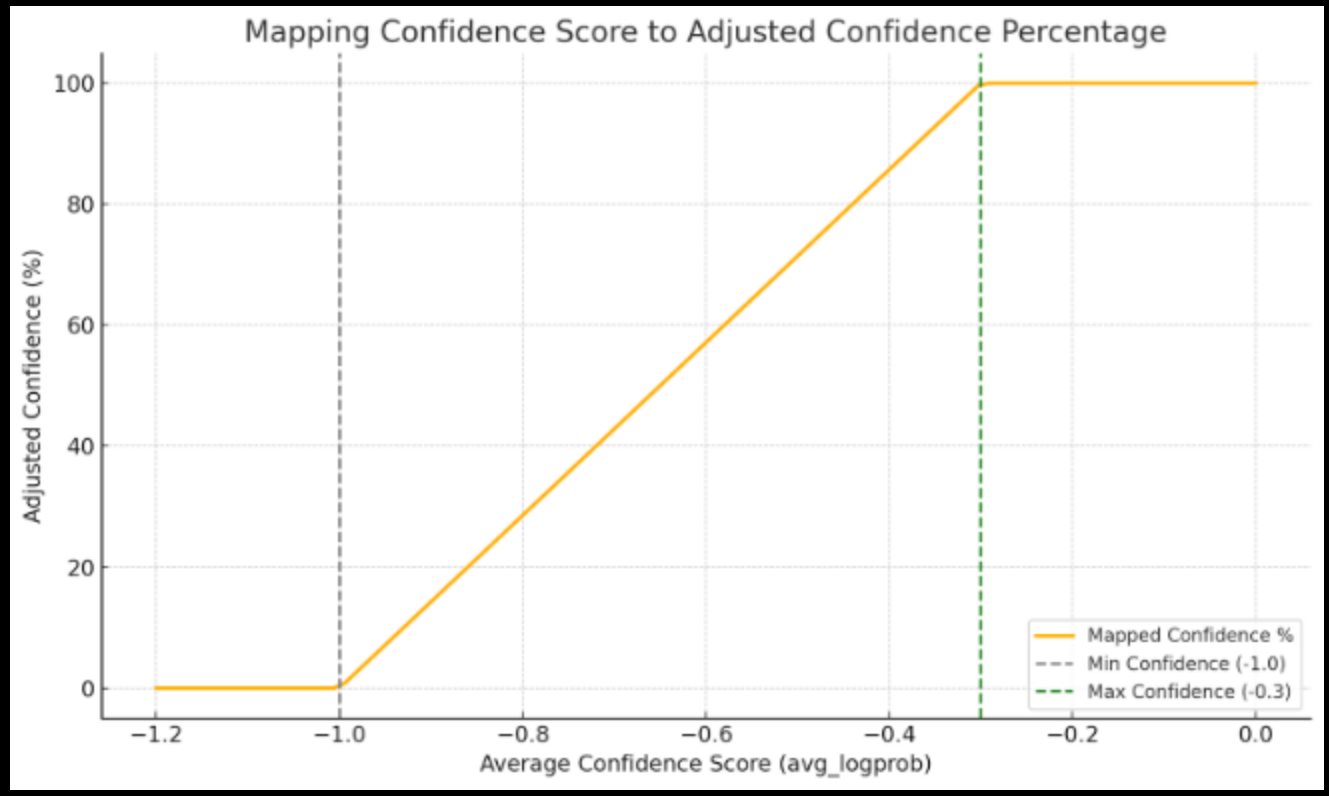
TESTING CONFIDENCE BY SPEAKER

- Speaker #2

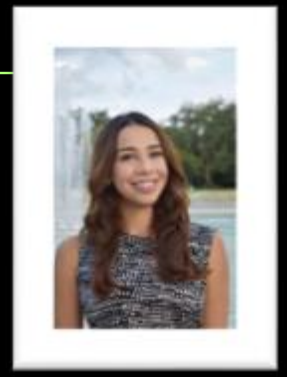
Run #	Sentence	avg_logprob	Adjusted Confidence (%)	Transcription Match?	Notes
1	Testing one two three.	-0.59	59.25	✅	Short + familiar words (Baseline)
2	“Frequency is set to 146.52 megahertz.”	-0.45	77.6	✅	Technical vocabulary
3	“I need backup at checkpoint three immediately, do you copy?”	-0.4	85.05	✅	Long + structured command
4	Probando uno dos tres, ¿me escuchas?	-0.73	37.415	✅ Partially	Multilingual test
5	“The quick brown fox jumps over the lazy dog.”	-0.5055	70.55	✅	Phonetically diverse sentence
6	“Hey… um, I think it’s working now… maybe?”	-0.283	90.95	✅	Informal + disfluencies
			70.13583333		

CONFIDENCE SCORE

- The model's confidence score (represented by avg_logprob) reflects how certain Whisper is about the words it has transcribed. This confidence depends on several key factors:
  - Context and previous words – Whisper uses context-aware decoding, so the confidence of a word depends partially on the words that came before it.
  - Sentence complexity – Longer or more grammatically complex sentences are harder to transcribe reliably, which may lower confidence.
  - Word familiarity – Whisper performs best on common or predictable vocabulary. Rare terms or jargon can reduce certainty.
  - Pronunciation and clarity – Slurred, rushed, or unclear speech can reduce transcription confidence.
  - Audio quality and background noise – Clean input leads to higher confidence; noisy, distorted, or distant speech lowers it.
- The x-axis shows avg_logprob (Whisper's confidence).
- The y-axis shows the calculated confidence percentage our system assigns.



JETSON SENDING TRANSCRIPTION (JSON STRING) TO ESP32



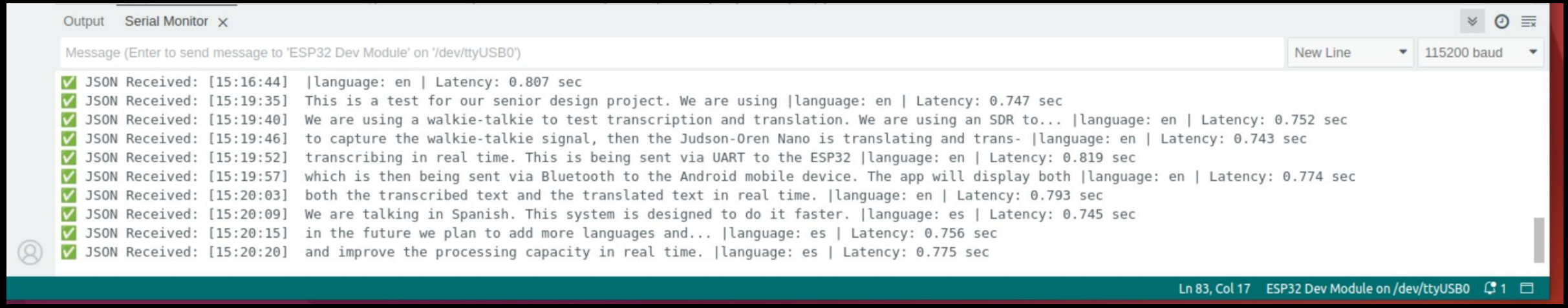
- **Real-Time Data Processing:**
  - The JSON is converted into a string and sent over UART
  - Transmits data using multi-threading

- timestamp (when speech was detected)
- text (transcribed message)
- language (detected language)
- latency (how long the transcription took)
- accuracy (how accurate the transcription is)

```
Sent to ESP32: {"timestamp": "15:19:35", "text": " This is a test for our senior design project. We are using", "language": "en", "latency": 0.747}

Sent to ESP32: {"timestamp": "15:19:40", "text": " We are using a walkie-talkie to test transcription and translation. We are using an SDR to...", "la
nguage": "en", "latency": 0.752}

Sent to ESP32: {"timestamp": "15:19:46", "text": " to capture the walkie-talkie signal, then the Judson-Oren Nano is translating and trans-", "languag
e": "en", "latency": 0.743}

Sent to ESP32: {"timestamp": "15:19:52", "text": " transcribing in real time. This is being sent via UART to the ESP32", "language": "en", "latency":
0.819}

Sent to ESP32: {"timestamp": "15:19:57", "text": " which is then being sent via Bluetooth to the Android mobile device. The app will display both", "l
anguage": "en", "latency": 0.774}

Sent to ESP32: {"timestamp": "15:20:03", "text": " both the transcribed text and the translated text in real time.", "language": "en", "latency": 0.79
3}

Sent to ESP32: {"timestamp": "15:20:09", "text": " We are talking in Spanish. This system is designed to do it faster.", "language": "es", "latency":
0.745}

Sent to ESP32: {"timestamp": "15:20:15", "text": " in the future we plan to add more languages and...", "language": "es", "latency": 0.756}

Sent to ESP32: {"timestamp": "15:20:20", "text": " and improve the processing capacity in real time.", "language": "es", "latency": 0.775}
```

RECEIVING TRANSCRIPTION (JSON STRING) FROM JETSON

- **Successful UART Communication:** Parsing JSON messages over UART from the Jetson, confirming a stable data link between the two devices.
- **Real-Time Processing and Storage:** Buffers incoming JSON messages, extracts key fields (text, latency, language, timestamp, accuracy)

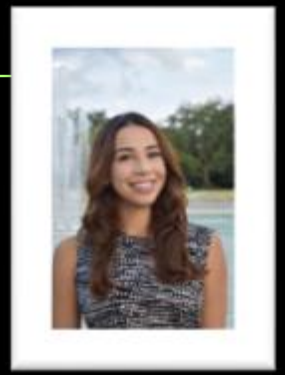


SOFTWARE IMPROVEMENT AND OPTIMIZATIONS

FASTER WHISPER AND GPU UTILIZATION

- Switched from original Whisper AI to Faster-Whisper, a more optimized version using ONNX Runtime.
  - Lower memory footprint and improved real-time transcription speed.
- Whisper now runs on the Jetson Orin Nano's GPU using CUDA acceleration.
  - This significantly reduces transcription latency compared to CPU-based execution.
  - Reduced transcription time per chunk → Faster real-time processing.
  - Lower CPU usage → Frees up resources for other tasks (e.g., handling JSON packaging, UART communication).
  - Better energy efficiency → Optimized GPU usage improves overall power consumption for extended runtime.
  - Real-time transcription is now more responsive and accurate.

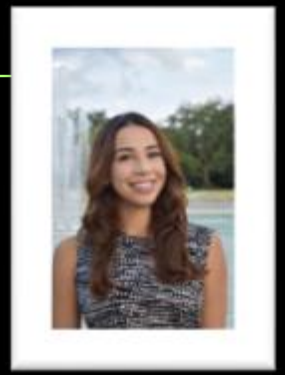
MODEL TRANSCRIPTION PARAMETERS



THESE PARAMETERS BALANCE CONFIDENCE, SPEED, AND EFFICIENCY FOR A LOW-LATENCY, REAL-TIME TRANSCRIPTION SYSTEM.

Parameter	Value	Reason
task="translate"	Required	The system is designed to transcribe and translate from any spoken language into English.
temperature=0.0	Fixed	Ensures deterministic and factual output with no randomness. Ideal for technical speech.
log_prob_threshold=-0.5	Fixed	Filters low-confidence words, helps accuracy without significant latency cost.
multilingual=True	Required	Enables language recognition and multilingual support, which is core to the system.
vad_parameters	Fixed (0.06, 50ms)	Reasonable VAD threshold; only the filter is toggled, not these values.
vad_filter	Variable (on/off) Yes	Helps the system remove silence and background noise before transcribing. This improves accuracy by making sure the model only processes actual speech. However, enabling VAD can add a small delay because the system spends time detecting when speech starts and stops.
beam_size	Variable (e.g., 2–8) 8	Controls how many possible transcriptions the model compares. A higher beam size increases accuracy but also adds latency. A lower beam size reduces latency but may slightly lower accuracy.

WHISPER OPEN AI MODELS



Model	Latency (Estimated)	Accuracy	Translation	Feasibility
Tiny	~0.2 - 0.4 sec	Low	Poor, struggles with accuracy	Best for real-time
Base	~0.3 - 0.9 sec	Moderate	Decent but struggles with complex speech	Feasible, but some translation errors
Small	~0.7 - 1.2 sec	High	Good translation ability	Feasible, good balance of speed & accuracy
Medium	~1.5 – 2.7 sec	Very High	More accurate, handles accents better	Less feasible, higher latency

HARDWARE PROTOTYPE AND TESTING

ESP32 – MAIN BOARD FUNCTIONALITY

Test Category	Test Description	Status
Voltage Rails	Measured 5V and 3.3V outputs with multimeter	✓
Power LED	Power LED turns on when powered	✓
Buttons	Measured voltage drop to 0V when pressed	✓
Flashing Code	Successfully uploaded code multiple times via UART	✓
Bluetooth Test	Verified Bluetooth communication with phone	✓
SPI Test	Confirmed SPI functionality with touchscreen display	✓

Test Category	Test Description	Status
I2C Bus Test	Communicated with INA219	✓
ADC Input Test	Verified analog readings	✓

CURRENT SENSOR – ESP32

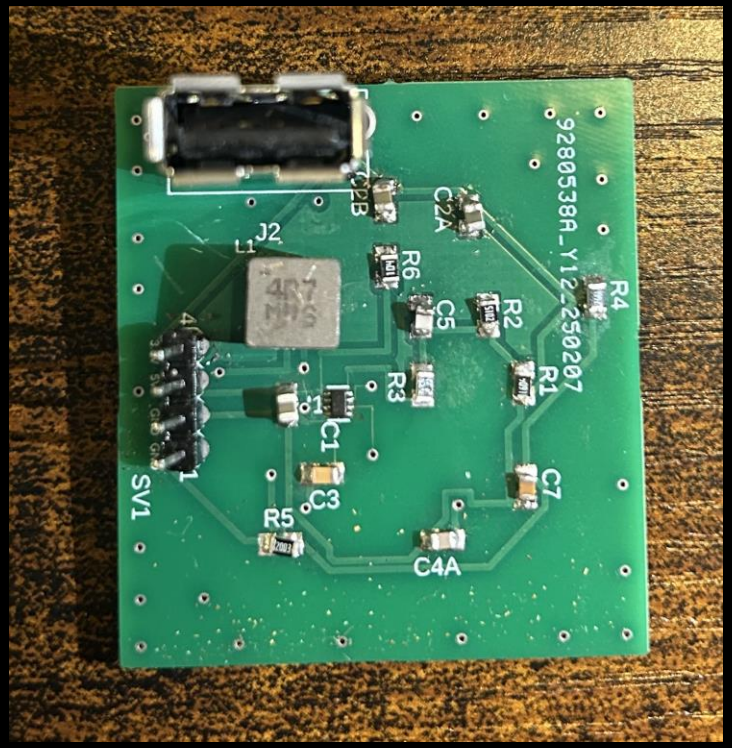
- Compared INA219 sensor readings to multimeter and clamp meter to verify accuracy.
- INA219 values were within ~10% of clamp and multimeter, validating calibration.

Measuring Device	Bus Voltage	Shunt Voltage	Current	Power
Multimeter	4.9V	27mV	100.7mA (measured 0.268 ohm shunt resistor)	0.49W
Ina219 current sensor	4.7V	29.99mV	111.91mA	0.52W
Current clamp	(4.9V)	-	130mA	0.64W

5V-3.3V SWITCHING REGULATOR

- Tested the functionality of the 5V to 3.3V switching regulator

Test Load	Voltage Output
100 mA	3.32 V
200 mA	3.31 V
400 mA	3.3 V
600 mA	3.29 V



BATTERY RUNTIME

- To Calc Remaining Energy (Wh)
  - (Battery %/100) x 222 Wh = Remaining Energy (Wh)
- To Calc Estimated Runtime (Hrs)
  - Runtime (hrs) = (Remaining Energy (Wh))/ Power Consumption (W)

Battery %	Remaining Energy (Wh)	Estimated Runtime (hrs)	ESP32 Behavior
100%	222	11.1	Normal Operation
75%	166.5	8.32	Normal Operation
50%	111	5.55	Normal Operation
25%	44.4	2.22	Normal Operation
10%	22.2	1.11	Shutdown and send Stop

MAJOR CHALLENGES

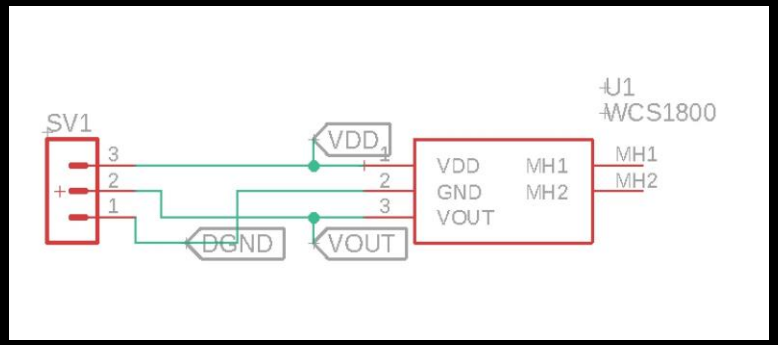
CURRENT SENSOR WOES

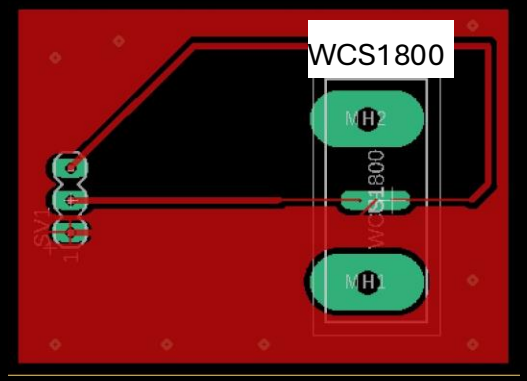
- Design of Current Sensor intended for 5V measurement with 5V input voltage
  - With the Jetson, the voltage we would measure is 19V and in the design this is connected to the input Vcc of the INA219 as well as the I2C pins to the ESP32
    - Our solution was to pull up the leg of the INA219 and apply the 5V directly to the pin from there, and we neglected to consider the I2C pins connected to the ESP32
    - Simultaneously when we were manually measuring the current, we short ground and 19V
      - As a result:
        - ESP32 fried – this was confirmed by a measured short between 3.3V and ground
        - Our TX and RX pins on the Jetson also were fried – this was confirmed by software testing of UART, transmitting and receiving to itself was un-successful

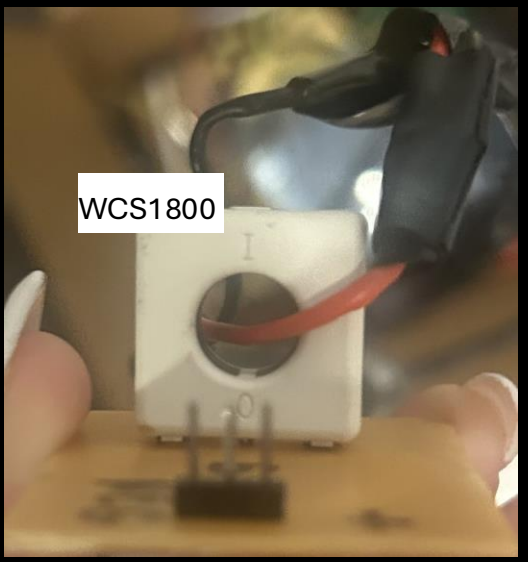
TO THE SOLUTION…

CURRENT SENSOR WINS

- We designed a NEW Current Sensor PCB that was non-invasive and electrically isolated for the Jetson
  - WCS1800
  - Plus, we got to mill our own board!

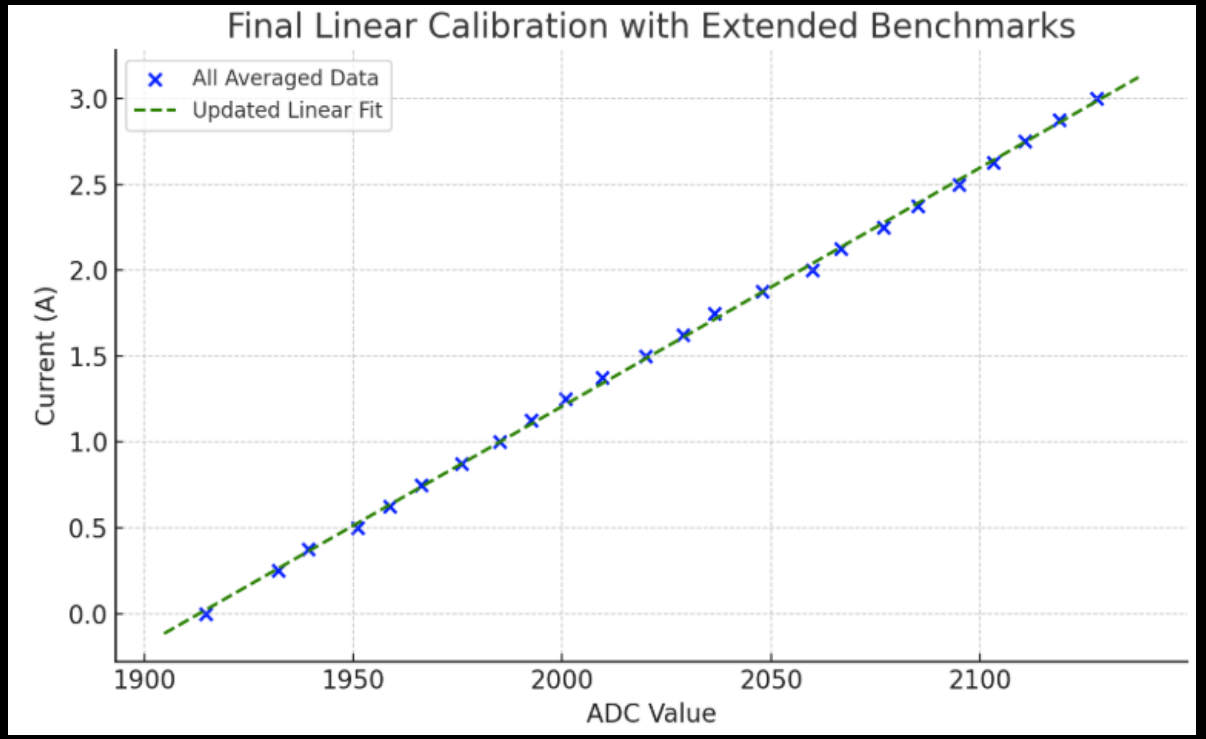






CURRENT SENSOR WINS

- Current Sensor Calibration
  - For sensor calibration
    - Read ADC values at known current levels (also measured with current clamp), the more values I collected, the more accurate I was able to get it
    - Using those to derive a linear relationship between ADC value and actual current (slope + offset)
    - Using that formula to estimate current during real-time use



ADMINISTRATIVE CONTENT

BUDGET

	Item	Quantity	Unit Cost ($)	Total Cost ($)
0	ESP32-WROOM-32E Devkits	2	10.0	20.0
1	Jetson Orin Nano Devkit	1	598.0	598.0
2	SDR Pluto	1	233.0	233.0
3	Backpack Antenna	1	10.27	10.27
4	Electronics Starter Kit	1	19.99	19.99
5	128 GB SD Card	1	25.0	25.0
6	Android Phone	1	150.0	150.0
7	Walkie-Talkie Set	1	34.99	34.99
8	ESP32-WROOM-32E MCUs	10	5.0	50.0
9	JLCPCB	1	107.0	107.0
10	Krisdonia Battery	1	229.0	229.0
11	Barrel Jack Cord	1	13.98	13.98
12	Current Sensors	1	8.19	8.19
13	USB Cord	1	5.99	5.99
14	LCD Screens	2	18.99	37.98
15	USB-A to C	2	6.99	13.98
16	USB-A to A	2	6.99	13.98
17	Total			1667.35

WORK DISTRIBUTION

Alayna Thurner	• Built the full GNU Radio pipeline for signal capture, filtering, demodulation, and streaming • Developed LCD touchscreen interface from scratch: pin config, touch calibration, and display logic • Optimized Whisper AI for GPU deployment to meet real-time performance goals • Led full system integration, including cross-subsystem debugging, final testing, and demo readiness • Managed all project logistics: sponsor coordination, milestone planning, team check-ins, documentation, and scripting
Sophia Demers	• Solely responsible for all custom PCB designs: mainboard, voltage regulator, and current sensing boards • Independently resolved hardware issues under pressure, including remilling boards and replacing failed sensors • Calibrated and tested power monitoring systems; validated system wiring and power delivery • Regularly worked in-lab to consult with EE faculty and complete designs under tight hardware timelines
Christine Hawkins	• Deployed Whisper AI and voice activity detection pipeline on Jetson Orin Nano • Developed Bluetooth communication on ESP32 and extended the LCD interface UI • Completed Android app functionality and debugging in the final phase of the project • Created the project website and supported integration testing during the final system build
Simeon Feliz	• Designed and 3D-printed the enclosure for housing internal system components • Contributed to development of the Android app UI and Bluetooth integration

NIGHTWING