K-Nightwing AI Transcribe: Real-Time Signal Processing, Translation, and Transcription System

Alayna Thurner, Christine Hawkins, Simeon Feliz, and Sophia Demers

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Reliable communication awareness is critical in defense and cybersecurity operations where walkie-talkies are commonly used in the field. Language barriers, environmental noise, and the need for operational secrecy can disrupt the clear understanding of mission-critical information. K-Nightwing AI Transcribe is a portable system designed to perform real-time transcription and translation to English of walkie-talkie communications without relying on internet access. Using a software-defined radio (SDR), the system captures RF signals, processes speech with an embedded AI model, and transmits results to a handheld device over Bluetooth. The entire pipeline runs offline, making it secure, discreet, and suitable for environments where connectivity is limited or prohibited. This paper presents the system's architecture, hardware and software implementation, and real-world performance evaluation. Results demonstrate the system's ability to maintain low latency, high transcription accuracy, and power efficiency in challenging environments.**

***Index Terms* — *Bluetooth* communication, Embedded systems, Low-power computing, Machine learning, Real-time systems, Signal processing, Software-defined radio (SDR), Speech-to-text**

I. INTRODUCTION

In modern defense, cybersecurity, and emergency response environments, clear and immediate communication is critical. Walkie-talkies remain widely used for their portability and line-of-sight connectivity but face challenges such as environmental noise, interference, and language barriers. In multilingual or high-stakes scenarios, these issues can lead to miscommunication, reduced efficiency, or compromised safety.

Most speech-to-text systems rely on internet connectivity, introducing latency and security risks, especially when sensitive audio is sent to cloud servers. Existing offline tools often require proprietary hardware or lack robust translation capabilities, particularly for noisy RF audio. While embedded AI is advancing, few systems integrate it into a real-time, portable platform that processes walkie-talkie transmissions.

K-Nightwing AI Transcribe fills this gap by combining a software-defined radio (SDR), on-device AI for transcription and translation, and Bluetooth communication to an Android device — all fully offline. By processing data locally, it reduces latency and enhances security. Its modular design supports future upgrades to AI models, UI features, and communication protocols, enabling adaptability across a range of mission-critical use cases.

This paper outlines the system's full development cycle. Section II presents the system architecture, followed by hardware (Section III), software (Section IV), and communication and security protocols (Section V). Section VI evaluates performance metrics from real-world tests, and Section VII reflects on lessons learned and future enhancements. Throughout the development process, particular focus was placed on modular design, power efficiency, and reliable end-to-end data flow — all of which were essential to achieving real-time, offline operation in unpredictable field environments.

II. SYSTEM DESIGN AND ARCHITECTURE

The K-Nightwing AI Transcribe system is designed to provide real time, offline speech translation and transcription from walkie talkie transmissions. This section dives into the system design and architecture that is implemented to achieve these qualities.

The core principles guiding the design include real-time performance, covert operation, low power consumption, and portability. These priorities influenced the selection and integration of each component, including the software-defined radio (SDR) for RF capture, embedded AI for speech-to-text processing, and Bluetooth transmission to a mobile device.

Key constraints include transcription accuracy, latency, battery life, computational limits of the Jetson Orin Nano, and robustness in variable environmental conditions. These factors directly impact the system's reliability and field performance.

The system's modularity allows hardware and software components to operate independently, making it easy to upgrade individual elements. Hardware selection emphasized a compact footprint and sufficient processing capability, particularly through the Jetson Orin Nano. The full system is enclosed in a backpack-mounted case to ensure portability and discreet use.

Each component performs a distinct role: the Pluto SDR captures RF signals and sends them to the Jetson Orin

Nano; the Jetson handles speech transcription and translation; and the ESP32 manages Bluetooth communication and displays power information on an LCD using data from two current sensors.

Together, these design choices enable the K-Nightwing AI Transcribe system to deliver secure, real-time speech translation in a compact, scalable, and field-deployable form factor.

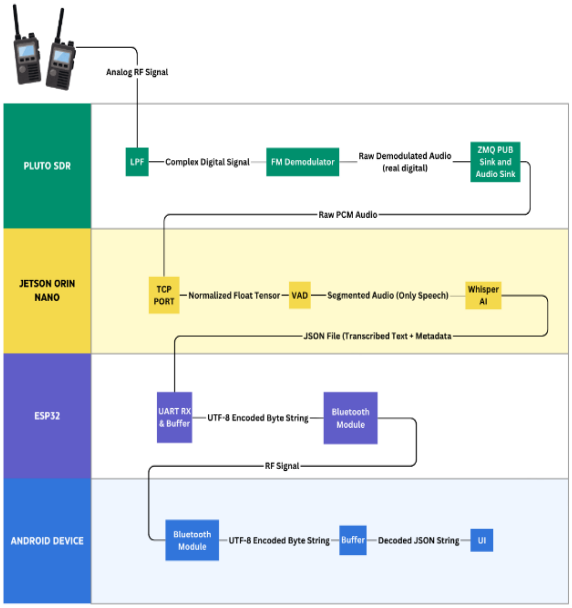

Fig 1: System Signal Flow Diagram

A. System Signal Flow

As shown in Figure 1, our system begins with GNU Radio controlling the Pluto SDR to capture analog RF walkie-talkie signals. These signals are passed through a low-pass filter, demodulated using a narrowband FM receiver, cleaned with squelch and DC blocking, and resampled to 48 kHz. The resulting audio is streamed over a ZeroMQ TCP socket in raw PCM format to support continuous real-time handoff with minimal delay.

The Jetson Orin Nano receives the audio stream in overlapping chunks to prevent data loss at segment boundaries. Each chunk is processed in a separate thread, allowing reception and transcription to run concurrently. The Whisper model performs speech-to-text by converting PCM audio into a normalized float32 array to ensure amplitude levels remain within expected bounds and avoid clipping.

Whisper's multilingual architecture allows it to generalize across speakers, accents, and noisy backgrounds. The small model variant was chosen to balance processing speed and transcription quality.

Once transcription is complete, results are packaged into a compact JSON object containing a timestamp, language, transcribed text, confidence score, and latency. This is sent via UART to the ESP32, which handles Bluetooth communication with the paired Android device.

The ESP32 manages Bluetooth pairing and controls pipeline flow through the attached LCD screen. When a "START" signal is received, it triggers the Jetson to initialize transcription, activate TCP audio reception, and switch to normal power mode. A "STOP" signal halts transcription and reverts to low-power mode. Reliable delivery is ensured by flushing the UART buffer and confirming message completion before proceeding. If disconnected, the ESP32 sends a "STOP" and continuously attempts to reconnect.

On the Android side, the app listens for incoming data, parses each JSON object, and updates the UI in real time with the transcription, timestamp, language, accuracy score, and connection status. Manual reconnection is possible through Bluetooth settings if needed.

The system operates entirely offline, removing dependence on cloud services and reducing both latency and security risk. Components run asynchronously, allowing modular development and enabling future upgrades—such as swapping AI models or adapting wireless protocols—without overhauling system logic.

Together, these software components form a real-time pipeline that converts RF audio into English text on a mobile device.

III. HARDWARE IMPLEMENTATION

This section provides a detailed breakdown of the physical construction and integration of the hardware components that make up the K-Nightwing system. Unlike the previous section, which focused on conceptual interactions, this section dives into how each hardware component was built, connected, and optimized for performance, stability, and power efficiency. The overall assembly process integrates multiple components: Pluto SDR, Jetson Orin Nano, ESP32, custom PCBs, Android device, and touchscreen LCD.

A. Custom PCBs and Power Management

There are 4 custom PCBs distributed throughout the layout of the system encasing. The 4 custom PCBs include a 5V to 3.3V regulator board, an ESP32 main board, and 2 current sensor boards. The regulator board provides power to the ESP32 main board and the current sensor boards. The current sensors are placed at the power input of the Jetson and the regulator board to measure the power consumption of each subsystem. All of the PCBs were designed in Fusion360 and fabricated by JLCPCB.

The 5V to 3.3V regulator board utilizes a buck converter, the MP2338. The MP2338 has an input voltage of 4.5V to 28V [2] which works well within the 5V input from our power bank battery. The regulator also can handle up to 3A of current which is well within the scope of what the system needs as this powers the ESP32 main PCB, TFT LCD screen and the current sensors. The regulator also has a built-in thermal shutdown system so that if the system exceeds 150 degrees celsius it will protect the device and shut down. [2] The layout of the board specifically follows the recommended layout of the regulator found in the data sheet to ensure the best outcome. [2]

The ESP32 main PCB was designed to have the same functionality as an ESP32-WROOM DEVKIT enabling efficient coding and power management. This PCB is powered by the regulator board and facilitates communication with the Jetson Orin Nano, Android mobile device, and LCD touchscreen. With the ESP32 having SPI, I2C and UART interfaces [3], these communication protocols are essential for data exchange and control across the systems components.

The current sensor boards are used to collect the power consumption values of the subsystem connected to the 5V to 3.3V regulator board as well as the Jeston Orin Nano. The current sensor itself that was chosen is the INA219 current sensor. The INA219 has a wide range of input voltages that can read from 0 to 26V [4] making it perfectly suitable for the two subsystems voltages. The INA219 also has a maximum current measurement range of up to 3.2A [4] and since the Jetson can get up to max 2.75A, this is well within the range that is needed.[4] One of the key reasons for selecting the INA219 was its ability to operate with a low power consumption while providing real-time monitoring of the system's power draw, which is critical for battery management.

B. RF Signal Reception and Processing – Pluto SDR

The Pluto SDR is mounted at the base of the system enclosure and connects directly to the Jetson Orin Nano via a USB-A to micro-USB cable, which supplies both data and power. This direct connection minimizes signal degradation and ensures stable integration with GNU Radio running on the Jetson. The SDR is tuned to 462.562 MHz, corresponding to Channel 1 of the Retevis RT68 walkie-talkies used during development, and operates with a 960 kHz sample rate and automatic gain control (Fast Attack mode) to accommodate variable signal strength in real-world conditions.

A broadband scanner antenna rated for 30 MHz to 1200 MHz reception was used in place of the stock antenna to ensure consistent signal quality within the FRS band. The antenna is routed to the top of the backpack using an adhesive loop mount, keeping it isolated from internal noise sources without requiring external RF enhancement components.

The SDR runs default firmware and requires only standard driver installation, enabling stable performance without custom modifications.

C. AI Processing Unit – Jetson Orin Nano

The Jetson Orin Nano serves as the central AI processing unit, mounted on the bottom layer of a custom 3D-printed enclosure designed for compact component arrangement, airflow, and cable management. Positioned alongside the Pluto SDR, it receives raw RF audio via a local TCP socket.

The Jetson is powered by a 19V rechargeable battery connected through the barrel jack input—necessary for reliable operation under GPU workloads. The battery is mounted externally for improved cooling and easier access during charging or swaps.

The ESP32, LCD, and other components are powered through voltage regulators on the custom PCB, powered by another output on the rechargeable battery of 5V and the regulator board voltage of 3.3V. The Jetson defaults to 7W mode to conserve energy and switches to 15W during transcription, triggered from signals from the ESP32 and controlled through Jetson's nvpmodel and jetson_clocks.

To support real-time performance, the system uses non-blocking input and multithreading. Audio reception and transcription run concurrently, while voice activity detection filters silence to reduce load. Completed transcriptions are formatted into lightweight JSON for efficient transmission.

D. Thermal Management and Portability

The hardware architecture of this design can be negatively impacted by overheating and tank the efficiency of transmission between components greatly. To avoid this, the enclosure that houses these hardware components needed to allow for proper ventilation and cooling. The most heat is generated by the Jetson Orin Nano as it is the largest hardware component of the stack, giving off temperatures ranging from 90°–160°F (32°–71°C). The Pluto SDR tends to operate around room temperature; however, maintaining close quarters with the Jetson Orin Nano is eligible to cause this temperature to rise.

The Jetson Orin Nano contains a small fan that is set to continuously cool the component during use. To

accompany this, passive airflow ventilation was added towards the top of the hardware enclosure to allow for new air to circulate. The walls of the enclosure were designed to be robust enough to hold the hardware without buckling or breaking, yet thin enough to avoid retaining heat.

The design places the hardware components in a tower formation, with the heaviest components on the bottom of the stack and the lighter components on top. With more considerable heat dissipation from the bottom components, the ventilation system allows heat to escape without affecting smaller components located near the top, ensuring functionality in extended use.

A backpack was chosen to conceal this enclosure discretely, as a core characteristic of this design is to remain inconspicuous during use. The backpack's lightweight and flexible fabric assists in maintaining the enclosure's temperature, while maintaining portability and secrecy. This method of transporting the hardware is also a considerable benefit for extended use, as it remains stationary and secured. The size of the enclosure was constrained by the backpack size, as well as concerns regarding bulging and potential tearing of the fabric. To avoid this, the enclosure was 3D printed using a lightweight yet durable filament, with a design focus on height over width, allowing it to fit comfortably into the backpack without risk of bulging or tearing.

E. System Control and Bluetooth Communication - ESP32

The ESP32-WROOM-32E microcontroller manages Bluetooth transmission and basic system control. It interfaces with the Jetson Orin Nano over UART to receive serialized transcription output and with the touchscreen LCD over a 4-wire SPI connection to process user inputs and display system status. These interfaces were chosen for reliability and compatibility with the system's embedded architecture.

The ESP32 transmits data wirelessly using its onboard Bluetooth antenna, which was sufficient for short-range operation without requiring external components to the Android mobile device. It also coordinates control functions such as transcription toggling and Jetson wake signals. The LCD touchscreen allows users to initiate Bluetooth pairing, begin or pause transcription, and view indicators for battery level and connectivity.

The system was designed to operate hands-free after startup, with the ESP32 handling control coordination and communication tasks in the background.

F. LCD Display

To allow the user to monitor the status of the hardware stack located in the backpack, the touchscreen LCD is located at the top of the enclosure and flushed to the roof for accessibility. This display is a 4.0-inch Thin-Film Transistor Liquid Crystal Display (TFT LCD) and stylus, selected to provide sharp and vibrant imagery showing battery life, power draw, and Bluetooth status. It receives information from the core PCB using SPI and GPIO for input. This SPI configuration also reduces power consumption as it requires fewer data lines, thus minimizing power draw to the component. To provide visibility in a range of lighting conditions, the display is also capable of modifying the Pulse Width Modulation (PWM) of the LED backlight, dimming or brightening the display when desired.

Due to the lightweight nature of the display, the resulting stress it produces on the 3D printed enclosure is quite negligible, while being held securely in place with small screws located at the corners of the display. The placement of the LCD allows for easy user access within the backpack, and concise, readable information relaying.

IV. SOFTWARE IMPLEMENTATION

This section outlines how the software components of the K-Nightwing AI Transcribe system were developed, configured, and optimized to enable real-time, offline transcription. All core components—including signal reception, AI inference, Bluetooth communication, and mobile UI display—were developed and tested to function without reliance on the internet. All components were integrated into a unified pipeline: GNU Radio streamed PCM audio via ZeroMQ, which was received and processed in real time by a multithreaded Python script.

A. Jetson Pipeline Setup

The transcription system runs entirely on the Jetson Orin Nano, allowing it to operate quickly, reliably, and without an internet connection. At the start of development, the transcription pipeline used the original Whisper implementation running entirely on the Jetson Orin Nano's CPU. In this setup, each audio chunk could take more than six seconds to transcribe, making it impossible to maintain real-time performance. In addition to the delay, the output frequently included incorrect or unrelated phrases. These issues were especially noticeable in noisy conditions or when the speaker's pace varied.

To address these problems, the system transitioned to faster-whisper, an optimized version of Whisper that runs inference using CTranslate2 and ONNX Runtime. By compiling CTranslate2 with CUDA and cuDNN enabled, the system offloaded all inference work to the Jetson's GPU, freeing the CPU for communication, buffering, and other tasks. This change dramatically reduced transcription latency—with most segments completing in

under 1 second—and improved accuracy, especially in background noise or fast speech.

To run the Whisper model efficiently on the Jetson GPU, system-level and Python dependencies were installed. The foundation of the setup is the NVIDIA JetPack SDK, which includes CUDA, cuDNN, and TensorRT for GPU acceleration on Jetson. The Whisper model was converted to the ONNX format, allowing it to run through ONNX Runtime with TensorRT acceleration instead of native PyTorch inference.

Python dependencies were managed in a virtual environment and included pyzmq for audio streaming, numpy for PCM processing, faster-whisper for transcription, and pyserial for UART communication with the ESP32. To enable full GPU acceleration, CTranslate2 was compiled with WITH_CUDA=ON and WITH_CUDNN=ON, while Intel MKL was disabled due to ARM incompatibility.

The switch from CPU to GPU inference, combined with the setup of the right libraries and tools, made a noticeable difference in how the system performed. Having installed CUDA-compatible libraries, system tools, and Python packages required to get the Whisper model running with full GPU acceleration, ensured low latency, and better accuracy for translation and transcription.

B. RF Signal Processing – GNU Radio

GNU Radio was installed directly on the Jetson Orin Nano to control the Pluto SDR and extract walkie-talkie audio in real time. Standard driver support for the ADALM-PLUTO and the libiio library were required, along with the appropriate Out-of-Tree (OOT) modules for IIO-compatible SDRs. The flowgraph was built using GNU Radio Companion and tuned specifically to 462.562 MHz, matching Channel 1 of the Retevis RT68 FRS walkie-talkies used during testing.

The SDR signal path uses narrowband FM (NBFM) demodulation. Incoming RF is filtered with a low-pass filter to suppress out-of-band noise, then passed through an NBFM receive block for demodulation. A squelch block mutes background noise during periods of silence, and a DC blocker removes baseline drift caused by minor DC offsets in the signal. These elements were tuned through iterative testing to produce clean, intelligible voice audio suitable for transcription.

Audio is resampled to 48 kHz and streamed as raw PCM using a ZeroMQ (ZMQ) TCP sink block. This continuous data stream is received by the Jetson Orin Nano with minimal latency. Initial testing used a file-based WAV approach, but it introduced significant I/O bottlenecks and buffering delays. Switching to real-time PCM streaming allowed the AI pipeline to process incoming audio more smoothly and with reduced lag.

Table 1: Whisper Models

Model	**Latency**	**Accuracy**
Tiny	~0.2 - 0.4 sec	Low
Base	~0.3 - 0.9 sec	Moderate
Small	~0.7 - 1.2 sec	High
Medium	~1.5 – 2.7 sec	Very High

Table 2: Transcription Parameters

Parameter	**Value**
Temperature	0.0
Beam_size	8
Log_prob_threshold	-0.8
Multilingual	True
Vad_filter	True
Vad_parameters	{"threshold": 0.02, "min_speech_duration_ms": 2}

C. AI-Based Transcription (Whisper AI)

This part of the pipeline is responsible for converting speech into text in real time. To meet the system's low-latency requirements, careful selection of the model variant and tuning of transcription parameters were essential. Several models and configurations were tested to find the best compromise between accuracy, speed, and resource efficiency on embedded hardware.

Different Whisper model variants were tested as seen in Table 1. The tiny model was the fastest but often inaccurate or gave incomplete transcriptions. The medium model had better accuracy but averaged about three seconds per audio chunk, making it unsuitable for real-time use. The base model showed slight improvements over the tiny model but still produced inaccurate transcriptions, especially when translating from Spanish to English. The small model was the best choice, averaging latency of about one second, with more reliable accuracy and better results in Spanish to English translation. Performance also improved by enabling INT8 quantization, which reduced memory usage and sped up inference without significantly impacting transcription quality.

To improve speed and transcription quality, several tuning parameters were applied as seen in Table 2. The

temperature filter is set to 0.0 for deterministic outputs, ideal for structured speech. The beam size is set to 8 for a balance between accuracy and latency. A larger value could improve output but would increase latency, which is not useful for real-time use. The log probability threshold is set to -0.8 to filter out low-confidence words. A VAD filter detects and removes silence before audio is sent to the model, reducing computation. The multilingual setting is enabled to allow transcription and translation of both English and Spanish without pre-selecting a language. Audio is grouped into overlapping chunks of about five seconds to prevent data loss at segment boundaries and give the model enough context to make accurate predictions.

D. Bluetooth Data Transmission – ESP32

Arduino IDE was used to initialize the ESP32 Serial Bluetooth connection, while code within the Android Studio software was designed to instantiate a handshake with this Serial Bluetooth port. Arduino's BluetoothSerial library was used to implement Bluetooth Classic, allowing the ESP32 to become discoverable and establish the proper port connection. The ESP32 is hardwired to the Jetson Orin Nano through a UART channel and receives the formatted JSON file.

The JSON file type was chosen because it is optimized for transmitting data of different data types. The Jetson Orin Nano transmits JSON objects containing key-pair sets of data. This allows the ESP32 to identify the characteristics of the data being transferred, which are then assigned to a pre-built object within the ESP32 code. Once a Bluetooth connection is established, the JSON object is serialized and sent to the mobile app.

In debugging the Bluetooth module, testing was performed across long distances, through physical obstructions, and in environments with other wireless signals. In all cases, the connection remained strong and consistently delivered reliable data transmission.

E. Touchscreen LCD Interface

The touchscreen UI was developed using a lightweight graphics library compatible with the ESP32 to reduce resource usage while maintaining responsive touch input. When the user interacts with the screen, touch inputs are captured through a resistive touch controller and processed by the ESP32 to trigger system-level commands, such as waking the Jetson Orin Nano from standby or beginning a transcription session.

The UI also displays key status indicators, including battery level, Bluetooth connection state, and transcription activity. These updates are rendered dynamically based on internal system messages passed through the ESP32's firmware. To ensure reliable operation in field conditions, the interface was designed to require minimal user interaction after startup, allowing the device to operate unattended once transcription has begun.

Power efficiency was considered during development by minimizing screen redraws and avoiding continuous polling of the touch interface when inactive. Although no formal low-power mode is implemented for the screen itself, the system architecture allows the touchscreen and ESP32 to operate in an energy-efficient state when idle. During development, the screen was tested for responsiveness and proved fast enough to handle all required UI actions without perceptible input lag.

E. Android Mobile Application

The mobile application was developed in Android Studio, a full-scale application software development kit (SDK) that allows the data being transmitted from the ESP32 Bluetooth connection to be displayed and formatted according to user preferences by utilizing Jetpack libraries. The application displays the current Bluetooth connection status, togglable filters for the timestamp, language, accuracy, and latency components of the received JSON within the transcription page, and a monitor on the battery life of the hardware system.

The Android application receives transcription data very similarly to the ESP32, by first defining and instantiating an object containing the message, timestamp, language, latency, and accuracy characteristics of the transcription. The software then utilizes the port connection with the ESP32 to recognize any serialized transcription data passed to the wireless communication channel, before assigning each data value to its corresponding key-value pair in the transcription object.

Error messages are displayed in case of faulty behavior, such as turning off Bluetooth after pairing, using the app before a connection is established, or powering off the ESP32 after connection. In all cases, the user is notified and prompted to reconnect.

One major challenge in integrating this application to the rest of the design was creating a Bluetooth connection that remains connected as the user navigates through different pages on the application. This was resolved by generating a global Bluetooth connection that is not delegated to the transcription page of the application only. The graphical interface of the application came with a steep learning curve as well, being resolved with practice and research of the SDK's available libraries and permissions.

V. COMMUNICATION, SECURITY, AND DATA INTEGRITY

Due to the system's prioritization of security, reliability, and efficiency, several tradeoffs and optimizations were considered. In real-time transcription, data loss, poor transmission latency, and security vulnerabilities can critically impact performance. Using a secure Bluetooth architecture introduces processing overhead, but the system's low data volume makes this acceptable, allowing security and robustness to take priority over throughput. BLE was considered to optimize power consumption, but it was ruled out because it doesn't maintain a constant connection—an essential requirement for this system. Bluetooth Classic was selected instead for its reliable connection, and its slightly higher power draw was determined to be negligible.

Bluetooth connectivity is managed through a handshake between the ESP32 and the mobile application. The user connects to the ESP32 via phone settings, while the ESP32 monitors the connection status. If disconnected, it attempts reconnection during a timed window; otherwise, it requires manual pairing.

A. Security and Data Integrity

Transcription data integrity is critical to system design, and unauthorized access is restricted. Bluetooth uses a pairing handshake to secure communication, limiting data access to paired devices. This is protected using AES-CCM encryption, which adds negligible latency for the JSON data size used in the system.

The ESP32 applies CRC and HEC to detect transmission errors. Corrupted packets are discarded and handled using Forward Error Correction (FEC), and if necessary, the ARQ protocol prompts retransmission. This architecture is widely used in Bluetooth systems and follows standards from the Bluetooth SIG.

A known vulnerability occurs during disconnection and reconnection, where spoofing attacks are possible. The system mitigates this by requiring a valid pairing before any data is sent and by including reactive reconnection logic to minimize risk.

VI. TESTING AND PERFORMANCE EVALUATION

To evaluate the real-world performance of the K-Nightwing AI Transcribe system, a series of controlled and exploratory tests were conducted using Retevis RT68 walkie-talkies and simulated field scenarios. Tests focused on measuring latency, power consumption, Bluetooth reliability, and transcription accuracy across a range of operating conditions. Although the system was not formally tested by external field operators, development team members conducted walk-and-talk trials indoors and outdoors, mimicking realistic usage patterns including variable distance, environmental noise, and intermittent movement. Performance data was collected through internal software timers, Bluetooth signal monitoring, and log outputs from the Jetson and ESP32 components. Testing emphasized real-time responsiveness and operational stability under conditions expected in defense or field-oriented deployments, with particular attention to whether the system could operate hands-free after initialization.

Table 3: Latency Results

Test Type	Input Phrase	Avg Latency	Min	Max
Simple Words	Good morning	0.732	0.684	0.758
Longer Sentence	The weather is nice and sunny outside	0.763	0.709	0.982
Spanish Translation	"Hola, cómo estás"	0.761	0.705	0.9
Background Noise Present	"Test message with static noise"	0.734	0.687	0.773
Testing from Far	"Testing from far away"	0.74	0.7	0.77
Fast Speech	"Hey check this now right away..."	0.733	0.703	0.74

A. Latency and Real-Time Performance

Latency was measured as the time between initial speech detection and the completion of transcription. The `speech_detected_time` was captured immediately after audio energy exceeded a set threshold, indicating active speech. The `transcription_end` timestamp was taken after the Whisper model finished processing each audio chunk. Each test phrase was run five times, and the average latency was recorded. This method captured the full processing delay, from RF audio reception to transcription output.

The Eq. for calculating the latency:

$$total_{latency} = transcription_{end} - speech_{\text{detected time}}$$

The system consistently maintained sub-second latency across various conditions. Table 3 summarizes results for different scenarios, including longer sentences, background noise, and Spanish inputs. The average latency ranged from **0.732 to 0.763 seconds**, with the longest delay observed during complex or multilingual sentences.

Bluetooth transmission to the Android device added minimal delay due to the small JSON payload size. There was no visible lag, and text appeared instantly on the Android app once received. Latency was further minimized through multithreading, CUDA acceleration, and VAD-based silence trimming.

B. Power Consumption and Efficiency

To test the power consumption of each subsystem, current sensors were attached to the input of the two subsystems to gauge the power consumption of the Jetson and the ESP32 with the LCD intact. The tests conducted were collecting the measurements during low power mode, in an idle state, in an active state, and during peak conditions (constant active state - multiple sentences being translated and transcribed in a row). The table below shows the values collected.

Table 4:Power States

State	Jetson	ESP32
Low Power Mode	~7736 mW	~554 mW
Idle	~9296 mW	~456 mW
Active	~13662 mW	~852 mW
Peak Active	~14474 mW	~938 mW

Jetson's Power consumption is configurable between 7W and 15W allowing the system to balance performance and efficiency as needed. To support long-term deployment, the Jetson alternates between low-power mode (7W) and performance mode (15W) depending on the START or STOP signal the ESP32 signal sends via UART. On receiving a START command, it escalates to high-performance mode and reinitializes the GNU Radio TCP connection. Conversely, when receiving a STOP signal, it drops back to low-power mode to preserve thermal budget and power consumption.

C. Bluetooth Transmission Range and Reliability

The maximum stable Bluetooth range was observed to be approximately 120–140 feet, with no noticeable differences between indoor and outdoor environments. In the presence of multiple signals such as Wi-Fi and other Bluetooth devices, the system operated as intended with no connection wavering. Tests included distancing the ESP32 from the mobile device until disconnection occurred and operating it in signal-dense environments with physical obstacles. Because the system uses Bluetooth Classic, the power level remains static.

The primary failure points were establishing an automatic reconnection feature within the application on the mobile device, and maintaining the Bluetooth connection as the user navigated between the different pages within the app. Since the ESP32 streams data over the Bluetooth port, no information is lost during disconnection and is received once the connection is re-established via the ESP32's packet retransmission protocol. At longer distances, slight delays occur, but transmission remains unaffected.

Table 5: Accuracy Results – Person 1

Test Type	Example Words	Expected Confidence	Actual Confidence	Expected Accuracy	Actual Accuracy
Longer Sentence	The weather is nice and sunny outside	-0.2 to -0.4	-0.26	80-100%	100%
Simple Words	Good morning	-0.3 to -0.5	-0.41	70-90%	84.96%
Complex words	Entrepreneur	-0.5 to -0.7	-0.69	40-70%	44.5%
Foreign language	Hola, como estas	-0.3 to -0.7	-0.43	40-90%	80.36%
Nonsense Words	Zarglonis	-0.7 to -1.0	-0.77	10-40%	32.16%

Table 6: Accuracy Results – Person 2

Test Type	Example Words	Expected Confidence	Actual Confidence	Expected Accuracy	Actual Accuracy
Longer Sentence	The weather is nice and sunny outside	-0.2 to -0.4	-0.32	80-100%	95.73%
Simple Words	Good morning	-0.3 to -0.5	-0.42	70-90%	82.59%
Complex words	Entrepreneur	-0.5 to -0.7	-0.62	40-70%	53.4%
Foreign language	Hola, como estas	-0.3 to -0.7	-0.38	40-90%	87.95%
Nonsense Words	Zarglonis	-0.7 to -1.0	-0.79	10-40%	30.55%

D. Transcription Accuracy and AI Performance

Each segment of audio is assigned an average log-probability score by the whisper model, reflecting how confident the system is in its interpretation of that portion of speech. A custom accuracy mapping function was implemented to translate confidence scores into a percentage.

The Eq for Accuracy Mapping:

$$accuracy = \frac{100 * (score - \min c\,onfidence)}{\max c\,onfidence - \min c\,onfidence}$$

Segments with clear pronunciation and strong contextual cues—such as those found in longer or complete sentences—tend to receive higher scores, typically close to -0.3, indicating high confidence. In contrast, segments containing short, unclear speech, or nonsense words usually score lower, closer to -1.0, indicating low confidence. To convert these scores into a readable accuracy percentage, a linear mapping was used: scores at or above -0.3 were treated as 100% accurate, while scores at or below -1.0 were mapped to 0%. This normalization provided a simple, interpretable accuracy metric for each transcription segment.

To validate consistency, test phrases were spoken by two individuals with different accents and tonal variations. Despite these differences, the accuracy scores remained closely aligned, confirming the robustness of the model's multilingual handling.

Calculating accuracy is important because it helps users understand how trustworthy each transcription is. Showing a confidence percentage alongside the translated text gives users immediate feedback on how well the system understood the audio signal especially useful in noisy settings or when translating between languages.

VII. CONCLUSION AND FUTURE WORK

The K-Nightwing AI Transcribe system proves that real-time transcription and translation of walkie-talkie communications is achievable using fully offline, embedded hardware. By combining software-defined radio, on-device AI inference, and Bluetooth communication, the system delivers secure, low-latency speech-to-text conversion in a portable, wearable form factor. It is well-suited for mission-critical applications requiring mobility and data privacy, with potential use in defense, emergency response, and remote fieldwork.

Future improvements include support for multiple radio channels, multilingual translation, and enhanced UI features. As the system transitions to Nightwing for demonstration and client use, further development will be guided by customer needs. Its modular design ensures adaptability to new requirements, environments, and technologies

ACKNOWLEDGEMENT

The authors wish to acknowledge the assistance and support of their mentor Mr. Bayardo Payan and Nightwing .

ENGINEERS

Alayna Thurner is a senior pursuing her Bachelor Degree in Electrical Engineering. Following graduation, she will join Nightwing's CODEX business unit as a Vulnerability Research and Reverse Engineer, delivering innovative solutions to both commercial and government clients.

Christine Hawkins is a senior pursing her Bachelor Degree in Computer Engineering. Upon graduation, she plans to join the Global Information Security team as a Software Engineer at Walt Disney Company, where she will contribute to the design and development of secure web

Sophia Demers is a senior pursuing her Bachelor's Degree in Electrical Engineering. She is currently an Electrical Engineering Intern at ACD Telecom. Upon graduation, she plans to work in System and Integration Testing, with a focus on ensuring hardware and software reliability in real-world environments.

Simeon Feliz is a senior pursuing his Bachelor's Degree in Computer Engineering, with a minor in Intelligent Robotic Systems. Upon graduation, he plans on working as a Test and Systems Integration Engineer, bridging the gap between Machine Learning and Artificial Intelligence.

REFERENCES

[1] Lenovo, "What is Cyclic Redundancy Check (CRC) and How Does it Work?," *Lenovo US*, [Online]. Available: https://www.lenovo.com/us/en/glossary/cyclic-redundancy-check. [Accessed: Mar. 26, 2025].

[2] Monolithic Power Systems, "MP2338GTL Datasheet," [Online]. Available: https://www.monolithicpower.com/en/documentview/productdocument/index/version/2/document_type/Datasheet/lang/en/sku/MP2338GTL.

[3] Espressif Systems, *ESP32-WROOM-32E / ESP32-WROOM-32UE Datasheet*, [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf.

[4] Texas Instruments, *INA219 Precision Digital Current and Power Monitor Datasheet*, [Online]. Available: https://www.ti.com/lit/ds/symlink/ina219.pdf.