Diploma Project

AI-Enhanced SDR-Based RF Signals to Text Platform

George Pettemeridis

University of Cyprus
Department of Computer Science

Department of Computer Science

May 2025

UNIVERSITY OF CYPRUS

Faculty of Pure and Applied Sciences

Department of Computer Science

AI-Enhanced SDR-Based RF Signals to Text Platform

George Pettemeridis

Advisor: Dr. Panayiotis Kollios

The Thesis was submitted in partial fulfillment of the requirements for obtaining the Computer Science degree of the Department of Computer Science of the University of Cyprus

May 2025

Acknowledgements

I would like to express my sincere gratitude to the individuals whose support and contributions have been instrumental in the successful completion of this research.

First and foremost, I extend my heartfelt thanks to my advisor, Dr. Panayiotis Kollios, for their invaluable guidance, continuous support, and insightful feedback throughout this project. Their mentorship played a vital role in shaping the direction and outcome of this work.

I am also thankful to the KIOS Drones Team, whose collective efforts and collaborative spirit created a productive and supportive research environment. Being part of this team has significantly enriched my experience and the quality of this thesis.

In particular, I would like to express my deepest appreciation to Dr. Nicolas Soulis and Maria Karatzia of the Drones Team. Their dedicated assistance, expertise, and encouragement were essential in carrying this project to completion. This work would not have been possible without them.

Thank you all for your contributions and unwavering support.

ABSTRACT

This work presents a system for automating speech transcription from conventional walkie-talkie communication using a combination of low-cost hardware, specialized signal processing software, and advanced speech recognition models. The proposed system is designed to facilitate real-time monitoring that leverages RTL-SDR (RealTek Limited Software Defined Radio) to detect and capture audio signals from active walkie-talkie channels within a specified frequency range. A channel identification procedure is employed to locate the active frequency channel, and then GNU Radio processes the audio signal by converting it into a standardized WAV format suitable for speech-to-text analysis. The audio data is transcribed using the Faster-Whisper model. The proposed system architecture consists of four main components: (1) signal acquisition and channel detection using RTL-SDR, (2) signal processing and audio extraction via GNU Radio, (3) speech-to-text conversion using Faster-Whisper, and (4) speaker diarization (classification) based on direction of arrival (DOA). This modular design aims to provide a reliable and cost-effective solution for real-time transcription applications in fields such as security, emergency response, and situational awareness. Numerous experimental evaluations were conducted to assess the system's performance in terms of channel detection accuracy, transcription quality, classification accuracy, and resilience to environmental factors such as background noise and variable signal strength. Initial results demonstrate that the combination of RTL-SDR and GNU Radio effectively isolates walkie-talkie channels, Faster-Whisper maintains high transcription accuracy, even under noisy conditions, and diarization is possible using DOA measurements of the walkie-talkie signal. This study contributes to the field by demonstrating a practical approach to real-time speech transcription using open-source tools, signal processing, and SDR technology. Future work will further investigate methodologies to improve the noise reduction and processing speed and also enhance the system's adaptability across diverse communication environments.

Keywords: Software-Defined Radio (SDR); Signal Processing; Speech-to-Text; Walkie-Talkie Transcription; Real-Time Monitoring; Direction of Arrival; Artificial Intelligence

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Important abbreviations that have been used in the text and need explanation are briefly presented.

SDR	Software Defined Radio
DOA	Direction Of Arrival
RTL-SDR	Realtek Software Defined Radio
WAV	Waveform Audio File Format
STT	Speech-To-Text
UCA	Uniform Circular Array
ULA	Uniform Linear Array
RTL-SDR	RealTek Limited - Software Defined Radio
GRC	GNU Radio Companion
FFT	Fast Fourrier Transform
VHF/UHF	Very High Frequency/Ultra High Frequency
OS	Operating System

Chapter 1 Introduction

1.1 Background Information

Traditional radio systems rely on fixed hardware configurations designed to operate on specific frequencies and modulation schemes, which makes it difficult for different radio systems to communicate with each other; for example, firefighters may be unable to communicate with police if their systems operate on different frequencies and cannot adapt to each other's modulation formats. This issue is solved by SDR (Software Defined Radio), which replaces dedicated hardware components with software algorithms that allow a single device to tune into various frequencies and modulation schemes dynamically. The flexibility of SDR makes it ideal for applications that require versatile communication options, including telecommunications, emergency response, and military operations.
SDR captures analog signals, such as AM or FM, and converts them into digital signals that a software can process and analyze. Through this analog-to-digital conversion, SDR systems can receive a wide range of signals and apply processing techniques, such as filtering, demodulation, and decoding, to interpret these signals for various purposes. For example, SDR can capture and demodulate audio signals from specific frequencies, converting them as WAV files, which can then be further processed or analyzed by software applications.
This ability to digitally process captured signals also enables more advanced applications, such as determining the origin of a transmission. One such technique is Direction of Arrival (DoA), which identifies the direction from which a transmitting signal is arriving. The transmitted signals include radio waves or sound waves.

1.2 Problem Statement

There is a growing need for real-time monitoring and transcription of voice communications in fields like public safety, emergency management, and security [1]. However, traditional radio systems lack the ability to dynamically scan, capture, and transcribe voice communications across multiple radio frequencies, which can bottleneck coordinated responses. SDR provides a solution by enabling flexible, software-based signal capture, yet most SDR applications only convert audio into transcribed text without going further. Furthermore, previous SDR-to-text setups often struggle with noisy environments and require costly hardware setups for accurate transcription.

Another challenge that arises is the ability to detect where a signal is coming from (eg. an audio signal, a Wi-Fi signal or in a attack scenario, a jammer). Direction of Arrival can also assist emergency responders in identifying the direction of a transmitting SDR, someone talking through a radio, and therefore be able to quickly go to their location and provide any necessary assistance, or be able to identify who is speaking based on the direction of the transmitter.
This project aims to address these challenges by using SDR to capture and process radio communications in real-time, specifically targeting walkie-talkie frequencies. The captured audio is processed and converted to text using OpenAI's Whisper model, an advanced deep-learning-based speech recognition system. In addition the system also utilizes krakenSDR to find the DoA, and identify multiple speakers. This setup leverages SDR's flexibility and Whisper's noise resilience, offering a scalable, cost-effective solution for real-time voice communication transcription and DoA identification with potential applications in security, public safety, emergency managment, and coordination efforts across communication systems.

1.3 Objectives

1. **Identify Walkie-Talkie Channels:** Utilization of RTL-SDR to scan frequencies and identify active channels used by walkie-talkies. Walkie-talkies typically operate in VHF/UHF(Very High Frequency/Ultra High Frequency) bands, so scanning within these frequency ranges is crucial. The goal is to determine the exact frequencies at which transmissions occur and monitor them in real time.
2. **Capture Audio from Walkie-Talkies:** After active channels are identified, the RTL-SDR needs to capture audio from these frequencies. This involves tuning the SDR to the identified frequency and recording the audio in real time, while handling changes in signal strength, background noise, and potential interference.
3. **Convert Audio to Text:** The recorded audio is transcribed into text using OpenAI's Whisper model, known for handling noisy real-world audio.
4. **Identify Speaker:** After transmission for a walkie-talkie has begun, with the help of krakenSDR, identify the direction from which the signal is coming from and therefore identify the speaker.

1.4 Structure of the Thesis

This thesis is organized into five main chapters:
Chapter 1 introduces the research background, outlines the problem statement, and sets the thesis objectives and scope.
Chapter 2 provides a comprehensive literature review, covering the foundational concepts of SDR, the capabilities of RTL-SDR, GNU Radio's role in signal processing, and recent advancements in speech-to-text technologies, with a focus on OpenAI's Whisper model, an overview of DoA technology and introduction to the KrakenSDR.
Chapter 3 explains the technical background necessary to understand the system's components, including an in-depth exploration of SDR fundamentals, walkie-talkie signal characteristics, Whisper's speech recognition process, krakenSDR hardware capabilities and challenges associated with these systems.
Chapter 4 presents the design and implementation of the system architecture, detailing each step of the signal acquisition, audio processing, and transcription pipeline, including diagrams and relevant code snippets.
Chapter 5 presents experiments, testing of the system and their respective results.
Chapter 6 presents coclusion and future work.

Chapter 2 Literature Review

2.1 Overview of SDR Technology

The timeline in Figure 2.1 represents the evolution of SDR (Software-Defined Radio). Initially, radio systems were heavily hardware-based, with each function (modulation, filtering, decoding) implemented using physical components. Over time, advancements shifted these functions into the digital domain, allowing software to handle tasks once managed by dedicated hardware. This evolution led to more flexible, adaptable, and cost-effective radio systems, resulting in fully digital, software-driven designs.

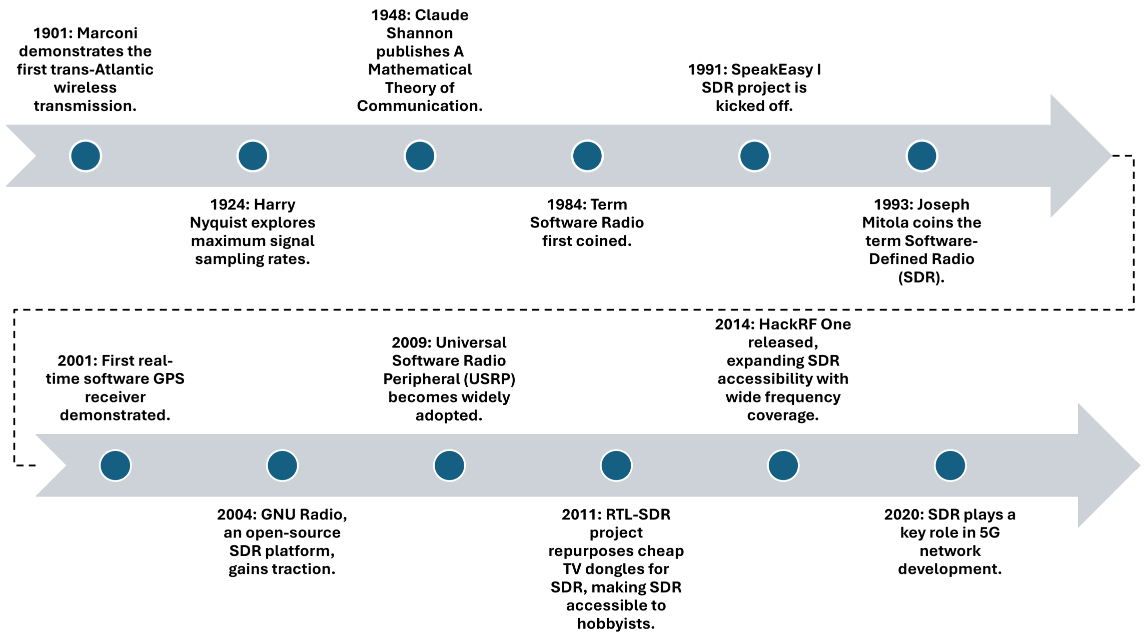

1901: Marconi demonstrates the first trans-Atlantic wireless transmission.
1924: Harry Nyquist explores maximum signal sampling rates.
1948: Claude Shannon publishes A Mathematical Theory of Communication.
1984: Term Software Radio first coined.
1991: SpeakEasy I SDR project is kicked off.
1993: Joseph Mitola coins the term Software-Defined Radio (SDR).
2001: First real-time software GPS receiver demonstrated.
2004: GNU Radio, an open-source SDR platform, gains traction.
2009: Universal Software Radio Peripheral (USRP) becomes widely adopted.
2011: RTL-SDR project repurposes cheap TV dongles for SDR, making SDR accessible to hobbyists.
2014: HackRF One released, expanding SDR accessibility with wide frequency coverage.
2020: SDR plays a key role in 5G network development.

Figure 2.1: Brief timeline of SDR's history.

2.2 GNU Radio and its Role in Signal Processing

GNU Radio is a robust tool for signal processing, ideal for tasks like detecting and processing walkie-talkie channels. Its modular architecture allows users to create signal processing workflows visually using the GNU Radio Companion (GRC).

2.2.1 Architecture Overview

GNU Radio utilizes a flowgraph architecture that allows users to connect various processing blocks, each performing a specific function such as filtering, modulation, or demodulation. This design simplifies the construction of signal processing pipelines and makes it easy to build and modify applications without the need for extensive coding.

2.2.2 Simplifying Signal Manipulation

The GRC serves as a graphical interface for designing these flowgraphs, generating Python code that manages the signal processing tasks automatically. Users can manipulate data streams visually, adjust parameters in real-time, and visualize results through various sinks, including FFT(Fast Fourrier Transfer) displays.

2.2.3 GNU radio-based Walkie-Talkie Applications

In the context of walkie-talkie channel detection, GNU Radio can seamlessly integrate with SDR hardware, helping with real-time signal reception and processing. Users can configure flowgraphs to filter and decode signals from designated channels, visualize frequency spectrum, and even implement features such as audio playback from decoded signals. This capability is essential for effectively analyzing radio communications, especially in dynamic environments.

2.3 Speech-to-Text Technologies

Speech recognition technology has made remarkable advancements over the years. Traditional systems, such as **Kaldi** and **CMU Sphinx**, are built on statistical models like **Hidden Markov Models (HMMs)** and **Gaussian Mixture Models (GMMs)**. These systems require a lot of feature engineering and phonetic modeling, with dependencies on predefined vocabularies and language models.

Recent innovations, particularly OpenAI's **Whisper** model, employ end-to-end deep learning architectures, specifically **Transformers** [2]. Whisper is trained on large-scale datasets that have diverse languages and audio environments. This enables the model to learn representations directly from raw audio to text, thereby minimizing the need for manual feature extraction and complicated pre-processing.

2.3.1 Machine Learning Model Comparisons

The shift from traditional statistical methods to deep learning models like Whisper represents a significant leap in speech recognition technology, facilitating more reliable and versatile applications.

Table 2.1: Comparison of Traditional and Whisper-Like ASR Systems

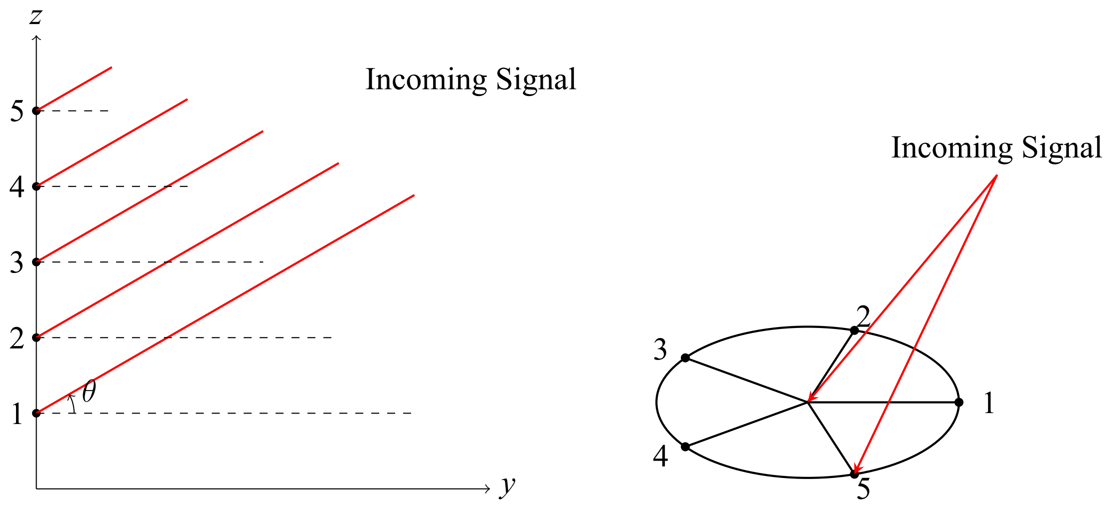
Category	**Traditional Systems**	**Whisper and Similar Models**
Accuracy and Robustness	Performance can be impacted in noisy settings or with accented speech due to limited training data and a dependency on handcrafted features.	Demonstrate higher accuracy and robustness across various languages and acoustic conditions, thanks to extensive training and their capacity to generalize.
End-to-End Learning	Employ distinct components for acoustic modeling, pronunciation lexicons, and language modeling.	Integrates all components into a single model, streamlining the pipeline and reducing potential error sources.
Adaptability and Scalability	Adapting to new languages or domains often requires significant manual effort and data preparation.	Can be fine-tuned or expanded with additional data more efficiently, utilizing transfer learning capabilities.
Resource Requirements	Generally require fewer computational resources for training, although they may not achieve state-of-the-art performance.	Demands substantial computational power for training but delivers superior performance, making it ideal for applications where accuracy is important.

2.4 Direction-of-Arrival (DoA) and KrakenSDR

2.4.1 Overview of DoA Technology

Estimation of DoA is performed with data acquisition by the antenna array and further processed through algorithms specifically designed for that purpose. Such algorithms are based on phase differences or time delay of the signals at each of the antenna elements, determining the source direction. High accuracy can be achieved when the employed algorithm belongs to high-resolution types (eg. Multiple Signal Classification-MUSIC-algorithm) even under closely spaced sources or under considerable noise conditions. Accuracy and resolution in DoA estimation largely depend on the antenna array configuration and on the type of algorithm applied. Among such configurations applied in practice are:

- **Uniform Linear Array (ULA):** A ULA (Figure 2.2(a)) is formed by placing antennas in a straight line with equal spacing between the antennas. Its simplicity of design and ease of implementation make it a very popular choice for many applications. However, ULAs lack in performance regarding the resolving power when the signals are arriving from the same or closely spaced directions.
- **Uniform Circular Array (UCA):** A UCA (Figure 2.2(b)) is formed by placing antennas in a circular configuration. It offers 360-degree coverage and relatively better resolution for some scenarios, such as when the direction of arrival falls in a wide span. A significant drawback is that UCA systems can be more complex to design and may have extra computational burden.



(a) Uniform Linear Array (ULA) (b) Uniform Circular Array (UCA)

Figure 2.2: Comparison of ULA and UCA Configurations

Depending on the application requirements for resolution, area of coverage, and system complexity, either ULA or UCA can be employed [3]. The configuration is generally chosen empirically based on the application needs while also considering the trade-offs.

2.4.2 Existing Algorithms for DoA

All the various algorithms developed for estimating DoA can be broadly classified under three categories: spectral-based, subspace-based, and parametric algorithms [4].

Spectral-Based Algorithms: These algorithms include the traditional beamforming method and rely on scanning the spatial spectrum for peaks corresponding to DoA. Though simple and computationally efficient, they are not so effective practically, which is due to poor resolution. Besides, Spectral-Based Algorithms are susceptible to noise, especially when the distance between the sources is small.

Subspace-Based Algorithms: Using subspace techniques like MUSIC and ESPRIT, the signal space can be decomposed into a signal and noise subspace. This assures very good resolution of even closely spaced sources. In fact, the MUSIC algorithm, employs orthogonality between the noise subspace and source steering vectors. Thus, it provides very accurate estimations of DoA.

Parametric Algorithms: Parametric Algorithms model the statistical properties of a signal for providing estimates of DoA and other parameters. A number Parametric Algorithms, such as the maximum likelihood method, give very good accuracy, though at high computational cost; hence, less suitable practically in a real environment.

2.4.3 KrakenSDR

2.4.3.1 What is KrakenSDR?

KrakenSDR is a five-channel coherent software-defined radio aimed at advanced applications that require numerous synchronized radio channels. It has been mainly used in signal processing applications, including DoA estimation, passive radar, beamforming [5]. The key feature differentiating the KrakenSDR from other SDRs is that the platform can achieve phase-coherent operation, which is necessary for correctly measuring the phase difference between the signals received by multiple antennas.
Features:

- **Multi-Channel Coherence:** The five channels of KrakenSDR are phase coherent-unlike the usual SDR, which operates at independent channels. Thus, it targets applications based on direction finding and beamforming.
- **Open-Source Software:** KrakenSDR comes with open-source software that helps in handling complicated applications like DoA estimation. The software helps to visualize the results, interface other systems, and experimental testing.
- **Real-Time DoA Estimation:** KrakenSDR is widely used for real-time direction finding. It accurately estimates the angle of an incoming signal using a five-antenna array, making it valuable for various applications such as signal localization and spectrum monitoring.
- **Application Extension:** It can also be integrated with larger systems, serve independently, and performs a variety of activities (eg. monitoring RF activity or mapping signals geographically)

2.5 Clustering

Clustering is a fundamental unsupervised learning technique used in data analysis and machine learning. Its purpose is to group together data in clusters such that the similarity between the data points inside a cluster is higher than the data points in different clusters. Clustering does not require labeled data, instead, it explores the structure of the dataset.

2.5.1 Clustering Methodologies

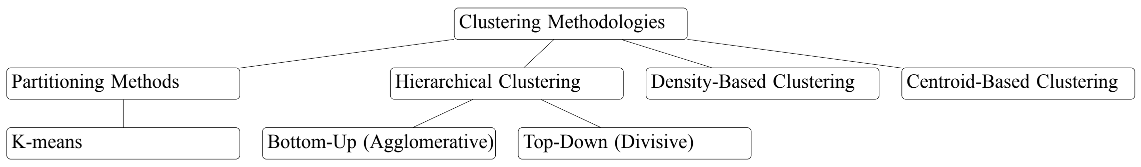


Figure 2.3: Hierarchical Diagram of Clustering Methodologies

2.5.1.1 Partitioning Methods

K-means: The K-means algorithm is popular due to its simplicity and efficiency. It creates clusters multiple times until the data converges to specific clusters. However, it requires the initial centroids to be set, and the performance of the algorithm depends on this initialization.

2.5.1.2 Hierarchical Clustering

This approach follows a tree-like structure and can be categorized into two methods:

- **Bottom-Up (Agglomerative):** Starts with each point of the dataset separately and merges clusters iteratively until a desired threshold is reached.
- **Top-Down (Divisive):** Begins with a single cluster containing the entire dataset and recursively divides it into smaller clusters until a specific threshold is met.

2.5.1.3 Density-Based Clustering

Density-based clustering algorithm defines an epsilon distance and a *"minimum neighbors"* value. If there is sufficient number of neighbors based on the epsilon distance, then a cluster is created. Thus, by correctly tuning the parameters, we can accurately create clusters.

2.5.1.4 Model-Based Clustering

Model-based clustering is a statistical approach to the clustering problem. For each data point, we try to recreate it using a combination of basic probability distribution. That combination is called a *”component”* and it determines in which cluster the data point belongs [6, 7].

2.6 Related Work

The exploration of capturing signals from walkie-talkies with the use of RTL-SDR (Software Defined Radio) technology that is followed by eavesdropping and subsequent processing to convert these signals into text represents a multifaceted area of research that intersects various domains, including signal processing, clustering algorithms, and machine learning. This summary will look at the methods and results from related studies and point out areas where they are lacking.

The initial step in this process involves the capture of walkie-talkie signals using RTL-SDR, a versatile tool that allows for the reception of a wide range of frequencies, making it suitable for various communication signals (e.g., walkie-talkie RF signals). In [8] , the RTL-SDR can sample frequency bands that encompass signals from FM radio, GSM, and other communication protocols, therefore providing a rich source of data analysis. The capability of RTL-SDR to operate across a broad spectrum is crucial for eavesdropping applications, as it enables the interception of communications that might otherwise be inaccessible.

Once the signals are collected, the next phase involves demodulation and conversion to text. This process typically utilizes software platforms like GNU Radio [9], which provide the necessary tools for signal processing. GNU Radio allows the implementation of various demodulation techniques that can convert analog signals into a digital format suitable for further analysis. However, the literature lacks comprehensive studies that detail the specific algorithms and methodologies employed in the demodulation of walkie-talkie signals. Even though some studies examined the GNU radio general capabilities, they do not provide in-depth analyses of the signal processing techniques that are most effective for walkie-talkie communications [10].

Recent studies have recognized OpenAI Whisper model potential in automatic speech recognition (ASR) tasks. For instance, recent evaluations of Whisper reveal its robust transcription capabilities across various accents [11]. However, they highlight that the model's effectiveness could be further optimized through fine-tuning for specific language characteristics or by improving efficiency through various techniques [12]. Thus, while Whisper shows promise in diverse ASR applications, addressing these optimization needs is crucial for elevating its performance benchmarks.

Further, the conversion of signals into a digital format is a critical step in various clustering algorithms to identify the target. This is where the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is employed due to its ability to identify clusters of varying shapes, sizes, and an unkown numbers of clusters, which is essential in cases where the inherently noisy nature of communication signals is present. The algorithm's effectiveness is highlighted in various studies demonstrating its application in network space clustering by exploring its use in ship trajectories. These studies underscore the versatility of DBSCAN in handling complex datasets [13–15].

However, the application of DBSCAN to walkie-talkie signal clustering presents unique challenges. For instance, the algorithm requires careful tuning of its parameters, specifically the neighborhood distance (Eps) and the minimum number of points (MinPts) [13]. The sensitivity of DBSCAN to these parameters can lead to suboptimal clustering results if not appropriately calibrated. This issue is acknowledged in the literature [16], emphasizing on the importance of parameter selection in achieving effective clustering outcomes. Despite this, there remains a lack of empirical studies that specifically address the parameter optimization for walkie-talkie signal clustering.

Moreover, the integration of Direction of Arrival (DoA) estimation techniques with DBSCAN for clustering purposes is an area that requires further exploration. Estimation of DoA can enhance speaker identification by providing spatial information on the source of signals. While some studies have investigated the use of clustering algorithms in conjunction with DoA techniques, the specific application to walkie-talkie signals remains underexplored. This presents an opportunity for future research to develop methodologies that combine these approaches effectively.

Furthermore, the existing literature focuses on the technical capabilities of clustering algorithms like DBSCAN without adequately addressing the limitations and potential biases inherent in these methods. For example, the clustering process can be influenced by the quality of the input data, which may be affected by noise and interference from other signals. A more nuanced understanding of these limitations is necessary to improve the robustness of the methodologies employed in this field [17].

In conclusion, while numerous research attempts have investigated the various components of capturing and processing walkie-talkie signals, significant gaps remain. The integration of RTL-SDR technology with advanced signal processing techniques in GNU Radio, coupled with effective clustering algorithms (such as DBSCAN), presents a promising avenue for research. However, the challenges associated with parameter optimization, the integration of DoA estimation, and the ethical implications of eavesdropping must be addressed to advance this field comprehensively. Future research avenues should aim to fill these gaps, providing a more holistic understanding of the methodologies and their implications in real-world applications.

Through these contributions, this thesis extends the application of SDR beyond signal detection to encompass real-time, content-level monitoring of voice communications, which has potential for significant impact in fields where clear and accurate transcription of radio signals is crucial.

Chapter 3 Technical Background

3.1 Docker Containers

Docker is an open-source platform that enables the automation of deploying, scaling, and managing applications by using lightweight, portable containers. Docker allows applications to run consistently across various environments (eg. development, testing, and production) by packaging the application along with its dependencies and configurations in a container [18]. This chapter provides an overview of Docker's containerization technology, explains its functionality, and highlights its significance in modern development.

3.1.1 Docker Architecture

Docker uses containerization technology, which packages an application and all its dependencies into a single, isolated environment known as a container. Unlike virtual machines, which encapsulate an entire operating system along with applications, Docker containers share the host OS kernel. This kernel-sharing feature makes containers more lightweight and efficient, using fewer system resources and enabling faster start-up times.

The Docker Engine is the core of Docker, which has two main components: the `Docker Daemon` and the `Docker CLI (Command Line Interface)`. The Docker Daemon waits for requests from the Docker CLI and builds and manages Docker objects, such as images and containers. An image can be considered as a kind of pre-configured snapshot of an application and its environment on the other hand, a container is actually a running instance of an image.

Docker uses a layered file system for images, each layer of an image representing the changes from the previous layer. This layering technique enables the efficient management of images and faster builds since Docker can reuse the unchanged layers while creating new images.

3.1.2 Containerization

At the core of Docker functionality is *containerization*, an isolation process for applications in their own environment sharing the host system's OS (Operating System) kernel. Thus, each container contains everything the application needs—libraries, binaries, and even configuration files—making sure it will behave as expected, regardless of the underlying infrastructure. Containerization gives resource efficiency; once created, containers can be stopped or started quickly to enhance system responsiveness.

3.1.3 Docker benefits in DevOps

Key reasons why Docker has been one of the most fundamental tools in the DevOps and cloud-native ecosystem include:

- **Portability:** Docker containers are highly portable across operating systems and cloud providers. Because the application and its dependencies are packaged together, Docker ensures that the behavior is consistent across different environments.
- **Resource Efficiency:** Containers share the host operating system's kernel. Multiple containers can run on the same system with very low overhead, unlike traditional virtual machines, which require their own OS.
- **Scalability:** Docker enables the creation of scalable applications due to the ease of replicating containers, which eases the scaling of applications horizontally.
- **Rapid Development and Deployment:** Docker enables the rapid iteration and deployment of images and the creation of containers. This fast lifecycle thus quickens the development process and encourages practices for continuous integration and delivery.
- **Isolation and Security:** Containers encapsulate an application's code and dependencies, isolating it from other applications. This isolation improves security and decreases compatibility issues.

3.1.4 Docker Compose

Docker Compose simplifies the management of multi-container applications. Using a configuration file `docker-compose.yml`, users may declare services, networks, and volumes using a single format for building, starting, and manipulating containers—while using only a few command lines.
The `docker-compose.yml` file defines all the services and their configurations. One command starts all services, connects them to each other, and creates a network so that containers can communicate with each other easily.

3.2 Fourier Transform (FT)

The **Fourier Transform (FT)** is a mathematical operation that transforms a signal from its time domain into the frequency domain . This transformation decomposes a complex signal into its composing sinusoidal components, each with a specific frequency, amplitude, and phase [19, 20].

Mathematically, the Fourier Transform is defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}\,dt \tag{3.1}$$

where:

- $x(t)$ is the time-domain signal,
- $X(f)$ is the Fourier Transform, which represents the signal in the frequency domain,
- f is the frequency, and
- j is the imaginary unit ($j^2 = -1$).

The Fourier Transform essentially evaluates how much of each frequency component is present in the original time-domain signal, producing a spectrum that represents the amplitude (and sometimes phase) of each frequency.

3.2.1 Fast Fourier Transform

While the Fourier Transform is a very powerful tool in signal analysis, for large datasets direct computation can become very computationally intensive, hence the computational complexity of $O(N^2)$. To overcome this problem, the **Fast Fourier Transform (FFT)** was developed. FFT is an efficient algorithm for computing the Fourier Transform where it reduces the computational complexity to $O(N \log N)$.

The FFT algorithm achieves efficiency in computing Fourier Transform through a divide-and-conquer technique. It first divides the signal into smaller parts, performs calculations over each part, and then combines them. Particularly, FFT becomes efficient when the number of data points, N is a power of 2 and hence can easily divide the computation recursively.

The FFT has wide application in practical fields because of its speed and efficiency, enabling real-time or near-real-time frequency analysis in applications such as audio processing, telecommunications, and medical imaging.

3.2.2 Inverse Fast Fourier Transform (IFFT)

The **Inverse Fast Fourier Transform (IFFT)** is a computational algorithm that, in essence, reverses the FFT. It transforms a signal from the frequency domain back to the time domain. IFFT takes the frequency components and reconstructs the original signal in the time domain, essentially reversing the FFT process. This operation is very important in the applications of signal processing, where, after a modification or filtering is applied in the frequency domain, one has to convert back to a time-domain representation for playback or further analysis of the signal. .

The mathematical expression for the Inverse Fourier Transform is:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} \, df \tag{3.2}$$

In practical applications, the IFFT allows us to take the frequency-domain representation obtained from an FFT and reconstruct the original signal accurately, provided that no essential frequency components were modified or removed.

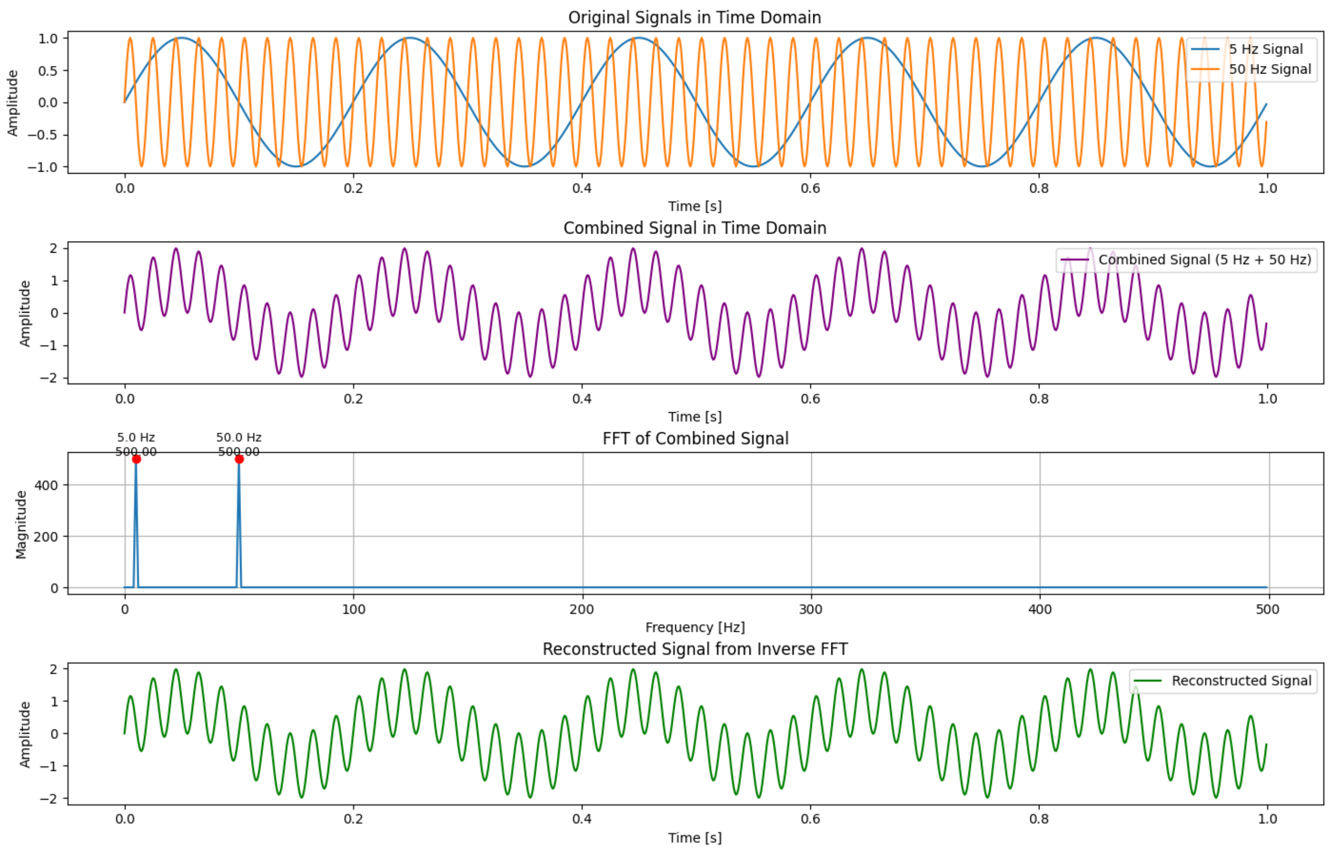


Figure 3.1: An example of FFT and IFFT.

For example in Figure 3.1, the first subplot shows the original 5 Hz and 50 Hz sine wave signals in the time domain [21]. The second plot shows the combined signal. The third plot displays the FFT of the combined signal, highlighting the frequency components at 5 Hz and 50 Hz. The final plot shows the reconstructed signal using the Inverse FFT, which matches the combined signal in the time domain.

3.3 Understanding RTL-SDR and Walkie-Talkie Signal Structure

The RTL-SDR is a versatile, low-cost tool for capturing signals from approximately 24 MHz up to about 1.7 GHz. That huge range makes it useful for capturing everything from FM radios to air traffic communications and two-way radios. While conventional radios depend on fixed hardware parts, the software approach utilized by the RTL-SDR provides versatility and adaptability in performing all forms of signal processing.

3.3.1 The Fundamentals of RTL-SDR

The RTL-SDR works by capturing analog signals from the radio frequency spectrum and converting them into digital formats for processing by software like GNU Radio. This conversion is crucial for the RTL-SDR's ability to detect and analyze real-world signals, such as those from walkie-talkies. The device is based on a repurposed DVB-T (Digital Video Broadcasting – Terrestrial) USB dongle, originally designed for TV signal reception. Its tunable frequency range allows it to be adapted for general radio signal reception.

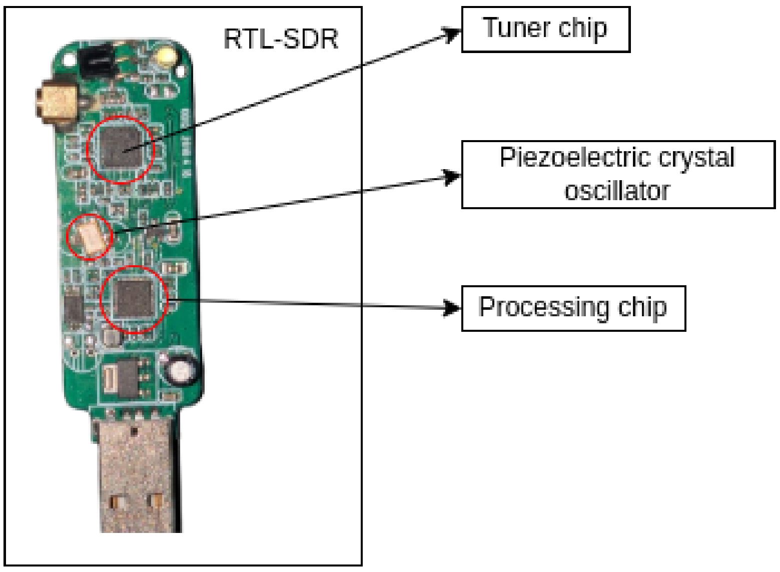


Figure 3.2: Representation of the system structure.

3.3.1.1 Crystal oscillator

The RTL-SDR is a radio signal decoder-software-defined radio reliant on a piezoelectric crystal oscillator, as are most software-defined radio technologies. A small crystal is made to oscillate or vibrate at a specific frequency. It provides a steady timing source and forms the basis of the radio signal tuning process, as seen in Figure 3.2. By default, all the older-generation radios had to replace the crystals manually since each crystal is tuned for certain types of resonating frequencies. Operating such radios required changing from one crystal that had been 'cut' or engineered to resonate at 14 MHz, to a crystal resonating on 28 MHz in order to switch channels from 14 to 28 MHz. However, with such manual tuning, flexibility was restricted by needing physical changes when frequency adjustments had to be made.

However, in a modern software radio system, only one crystal oscillator is used at a specified base frequency, for example, 28.8 MHz. To tune across a wide range of frequencies, electronic multiplication or division is applied to this base frequency. For example, to lock into a signal of 57.6 MHz for a crystal oscillating at 28.8 MHz, it can be resonated at two times the base frequency. This kind of modern solution allows a single crystal to support more frequencies. Nevertheless, some problems persist about frequency stability, given primarily by thermal variations and natural tolerances in the crystal's construction. Over time, this may cause a slight frequency shift, which can result in the SDR being slightly off its target frequency.

In this context, PPM, or ”parts per million,” is one of the key units of measurement used in characterizing the stability and accuracy of a frequency. More precisely, in software-defined radios, PPM describes the deviation of the oscillator frequency from its nominal value. For instance, if an oscillator operates at a frequency of 28.8 MHz and has a PPM value of 10, then it has a potential frequency drift of 10 parts per million, which can be interpreted as a possible variation of ±0.288 kHz.

Some calibration processes are necessary to solve such inaccuracies. First, the SDR can be aligned with a known, stable frequency source, a local FM transmission or a meteorological broadcast. Users then tune the PPM correction factor to make the needed adjustments in frequency drift. This is a necessary process for calibration such that the frequency shown on the SDR corresponds to the real frequency received. The SDR should therefore offer great accuracy in situations where an exact display of frequency becomes very critical, such as the decoding of digital signals or receiving emergency communications.

3.3.2 Modulations

3.3.2.1 Amplitude Modulation (AM)

In AM (Amplitude Modulation), the amplitude of the carrier signal is varied in proportion to the message signal and the frequency remains constant. Information, therefore, is encoded in the changes in amplitude of the carrier wave. AM is normally used in radio broadcasting and is known for its simplicity. However, it's more susceptible to noise, as noise generally affects the amplitude of the signal. Figure 3.3 shows the process of Amplitude Modulation (AM). The top plot displays the carrier signal, a high-frequency sine wave with constant amplitude. The bottom plot shows the data signal, a lower-frequency sine wave that contains the information to be transmitted. The middle plot shows the modulated signal, where the amplitude of the carrier varies according to the data signal.

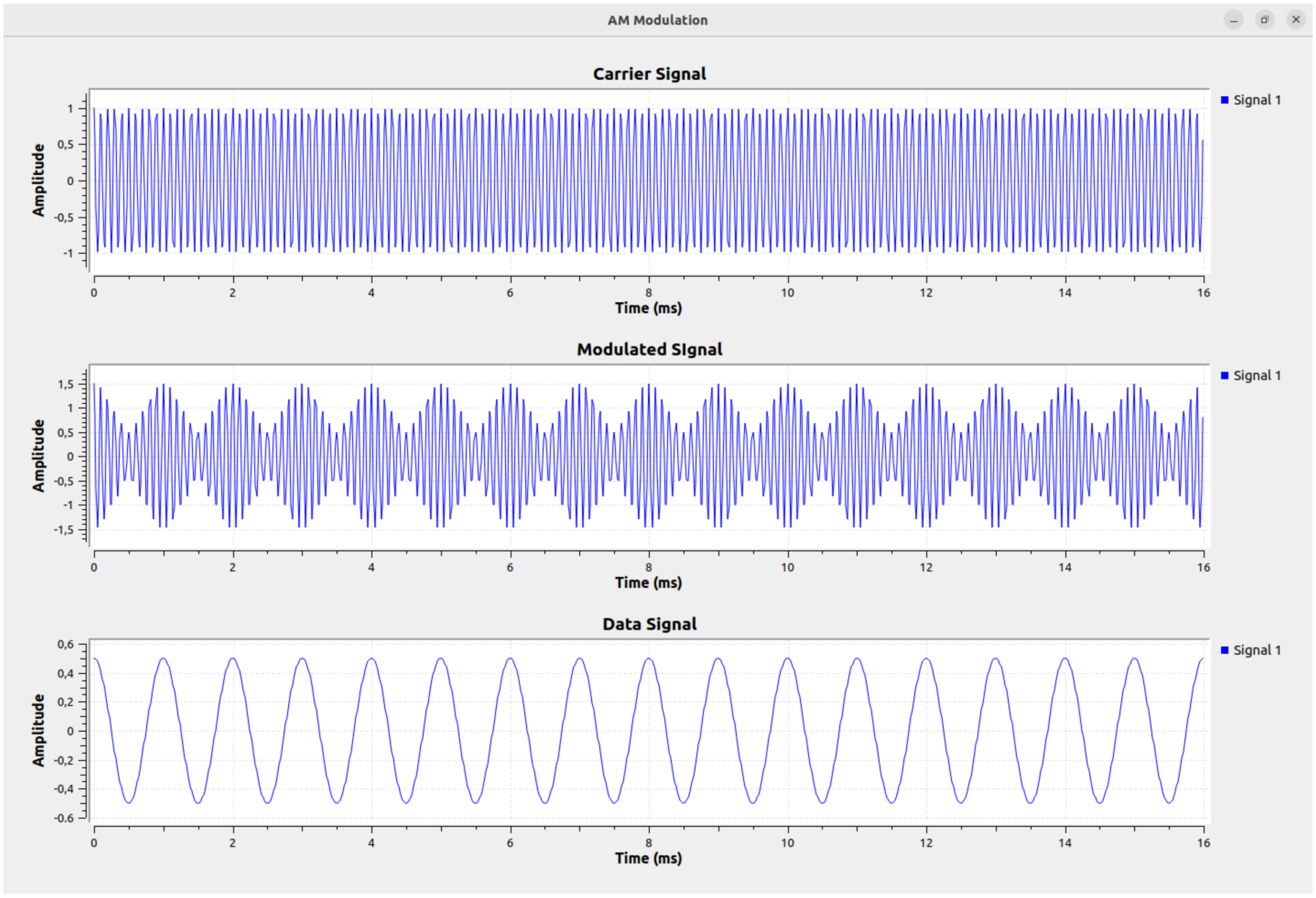


Figure 3.3: Example of Amplitude Modulation.

3.3.2.2 Frequency Modulation (FM)

In FM (Frequency Modulation), the frequency of the carrier signal is varied in accordance with the message signal, while the amplitude remains constant. FM is less prone to noise because noise primarily affects amplitude, making it a popular choice for high-fidelity audio broadcasting, such as FM radio. However, FM requires a larger bandwidth compared to AM, which is a trade-off for its improved noise resistance. Figure 3.4 shows the process of Frequency Modulation (FM). The top plot presents the carrier signal, a high-frequency sine wave with constant amplitude and frequency. The bottom plot shows the data signal, a lower-frequency wave containing the information to be transmitted. In the middle plot, the modulated signal is shown, where the frequency of the carrier varies according to the amplitude of the data signal, while its amplitude remains constant.

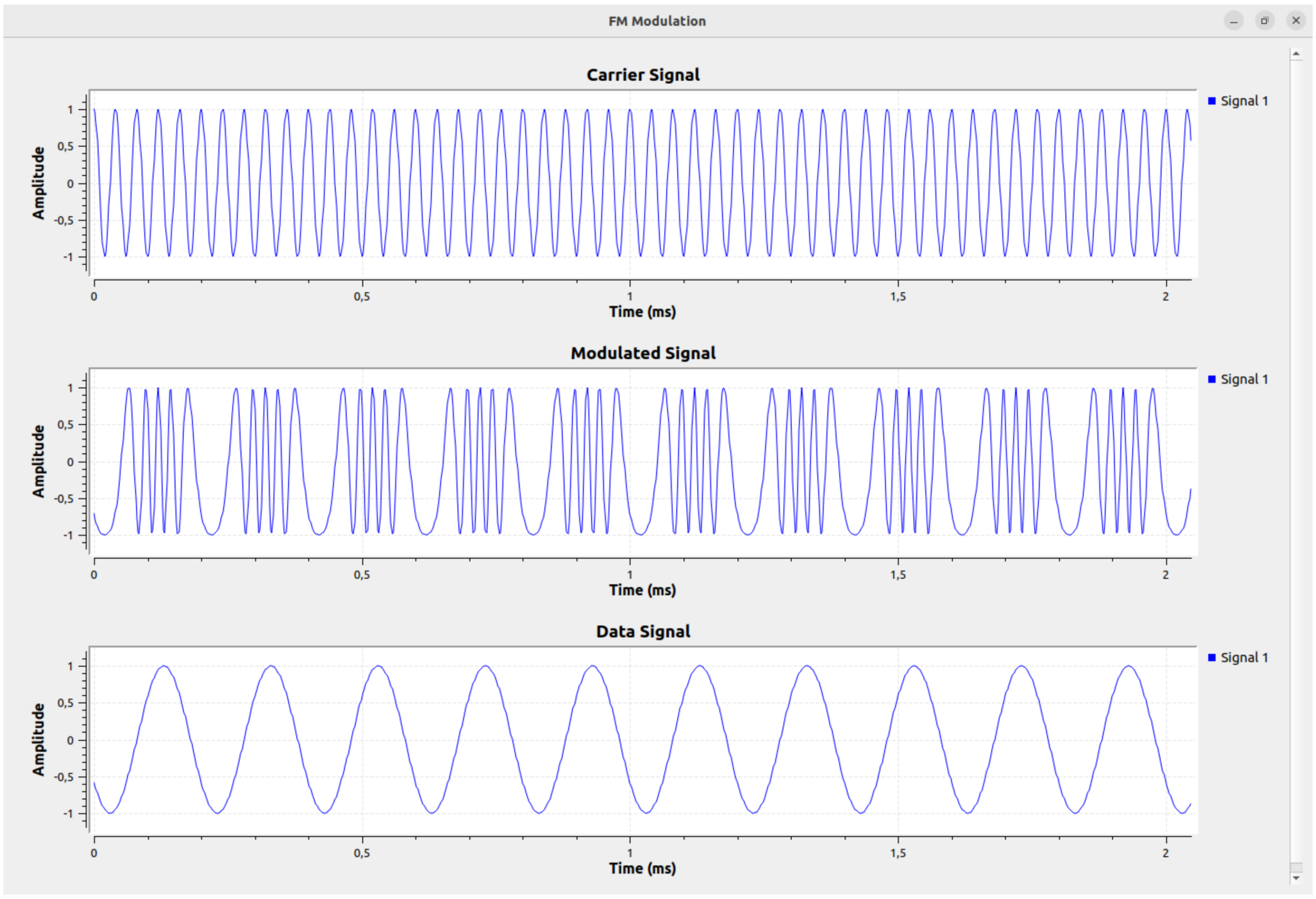


Figure 3.4: Example of FM Modulation.

3.3.2.3 Key Differences Between AM and FM

Table 3.1: Key Differences Between AM and FM

Aspect	**AM**	**FM**
Carrier Signal Variation	The message signal modulates the amplitude of the carrier wave, while the frequency remains constant.	The message signal modulates the frequency of the carrier wave, while the amplitude remains constant.
Noise Sensitivity	More prone to noise because noise affects the amplitude.	Less affected by noise since information is encoded in frequency.
Bandwidth Requirement	Requires less bandwidth.	Needs a larger bandwidth and finds application in fields where signal clarity is preferred over bandwidth efficiency.
Application	Used in ordinary radio broadcasting.	Used in high-quality audio broadcasting and where noise resistance is crucial.

3.3.3 KrakenSDR hardware overview

KrakenSDR is a 5-channel coherent SDR platform designed for demanding RF applications that require synchronized multi-channel receive. The hardware design brings together several components for phase-coherent signal processing, suitable for use in applications like radio direction finding, passive radar, and beamforming. Below are the main hardware components of the system. 3.5 illustrates the top view of the KrakenSDR device. It shows five SMA antenna input ports labeled CH0 to CH4, which are used to connect antennas for signal reception. The device features heatsink fins and a centrally located cooling fan to ensure efficient thermal management during operation. Along the bottom edge, there are ports for power and data connections. Additionally, LED indicators labeled ”PWR” and ”NOISE” are present to display the device’s power status and noise source activity, respectively.

RF Front-End: The system is integrated with five RTL-SDR modules; each has one independent receiving channel. These give a combined frequency range from $24\,$MHz to $1.766\,$GHz and a maximum tunable instantaneous bandwidth of $2.56\,$MHz per channel. There is an SMA connector for antenna input on each channel, enabling flexible deployment with external antennas or custom arrays.

Synchronization Mechanism: Coherent operation over all channels is maintained due to the presence of a common reference clock and LO distribution network. This ensures appropriate phase alignment useful in direction finding and interferometry applications. This ensures LO synchronization to reduce phase drift and guarantees coherence over long operating conditions.

Data Processing: Each RTL-SDR module is responsible for providing raw IQ(Real/Imaginary) data to the core software which is designed to run on an external computing device for data processing. The KrakenSDR is connected to the computing device via a USB-C cable for data transfer.

Connectivity and Power: The KrakenSDR requires DC power from a 5v 2.4A USB-C power supply. It can be powered thourgh a powerbank for portability and avoid the need for a power plug.

Mechanical Design: The KrakenSDR is protected through a small aluminum housing which also reduces RF interference between the channels and utilizes passive thermal management.

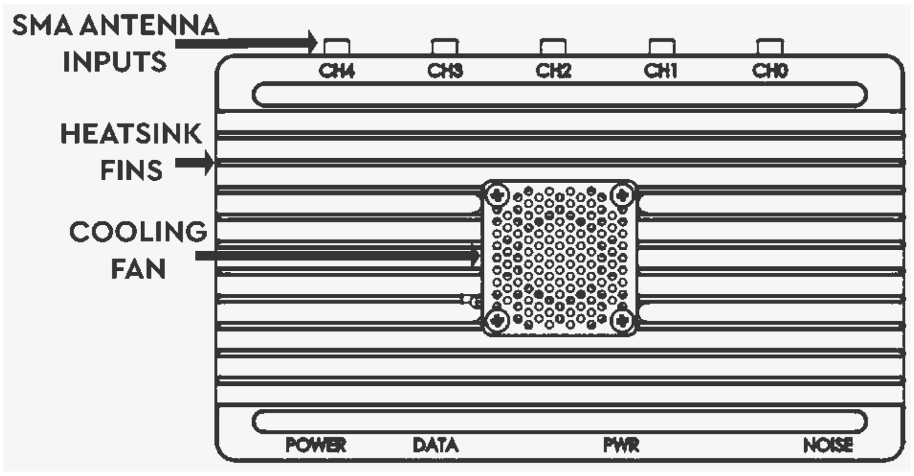


Figure 3.5: KrakenSDR Hardware Descriptions.

3.4 GNU Radio for Signal Processing

GNU Radio is a free, open-source software development toolkit that is specifically designed at signal processing applications. It provides the appropriate tools, which enable the user to build up and manipulate software-defined radio systems. In that respect, it comes with a set of ready signal processing blocks with which radio systems can be designed, tested, and simulated on the computer with or without real RF hardware. This has made it very popular in the academic and commercial sectors for both research into wireless communications and for the more practical implementation of radio systems.

3.4.1 GNU Radio Fundamentals

At the core, GNU Radio uses the *Python programming language* for developing signal processing applications based on Python scripts. With large libraries and intuitive syntax, Python allows for much easier manipulation and testing of radio systems. It thus provides a system that is flexible and, at the same time, simple to manipulate for both beginners and experienced developers. The interface and high-level programming in GNU Radio is done in Python, while much of the central signal processing is done in *C++*, making it computationally efficient.

3.4.2 Signal Processing Blocks and Flowgraphs

In this sense, the modular approach in place within GNU Radio relies on the bricks called *signal processing blocks* or modules, performing a specific job like filtering, modulation, demodulation, and FFT calculations. The blocks are combined into *flowgraphs*, describing the structure and data flow of a signal processing pipeline. Blocks are linked by the user to make a complete signal processing system, where each block represents a different stage in the pipeline.

- **Types of Blocks**: GNU Radio provides a range of pre-built blocks, such as filters, oscillators, mixers, amplifiers, and modulators, which are the building blocks of any radio system.
- **Custom Blocks**: Users can create custom blocks for specialized processing needs, providing flexibility to tailor GNU Radio for unique projects or research applications.

3.4.3 Using GNU Radio with hardware in the loop or in simulation mode

GNURadio versatility allows functioning with or without RD hardware:

- **With Hardware**: GNU Radio interfaces well with SDR hardware such as RTL-SDR, USRP, and HackRF. Users can transmit and receive RF signals in real time. In this environment, GNU Radio will process the signals captured by devices like RTLSDR and perform many practical tasks such as capturing walkie-talkie communications or capturing FM broadcasts.
- **Without Hardware (Simulation Mode)**: In the absence of hardware, GNU Radio can simulate an entire radio system creating and processing signals only in software. Thus, the user can design a prototype of the signal processing part and test the same without any physical RF hardware, perfect for academic research and prototyping.

3.4.4 GNU Radio Companion (GRC)

GNU Radio includes a graphical interface called *GNU Radio Companion (GRC)*, which simplifies the process of building and visualizing flowgraphs. GRC allows users to design signal processing systems by dragging and connecting blocks visually, making it easier to test and troubleshoot designs. This GUI-based approach is especially helpful for beginners to SDR or prefer visual design over coding.

3.5 OpenAI Whisper for Speech-to-Text Conversion

3.5.1 Transformers

Transformers represent a new architecture for training models, initially described in the 2017 work *Attention is All You Need* [2]. This new model represented a radical turn from the formerly traditional RNNs (Recurrent Neural Network) [22], which conventionally processes sequences in a step-by-step manner, taking in one word of input at a time. The main disadvantages of this approach are that RNNs tend to forget important contextual information from the beginning of long sentences, and are computationally expensive to train because parallelization is not well-supported [23].

Transformers overcome these weaknesses with three mechanisms:

1. **Positional Encodings**: The transformer model embeds each word's position in the input sentence, allowing the model to learn word order and contextual relations.

2. **Attention Mechanism**: The transformer does not take one word at a time; instead, it views the whole sentence at once. The model makes explicit which words in the sentence are relevant to the given target word on the basis of long-range contextual dependencies.
3. **Self-Attention**: Using self-attention, the model grasps every word in terms of its neighbours to gain deeper semantic understanding contextually.

These are some of the new features in Transformers that greatly lift their performance in complexity for tasks where an RNN typically falls apart in accuracy.

3.5.1.1 Quantization

This technique decreases the computational and memory load that some modern models—based on transformer architecture—demand. Specifically, quantization involves converting weights and activations into lower-precision data types, like 8-bit integers (int8), rather than relying on the standard 32-bit floating-point data type that is typically used. This allows a reduction in size and accelerates processing with minimal loss of model accuracy.

3.5.1.2 CTranslate2

CTranslate2 [24] is a highly optimized C++/Python library for transformer models which accelerates and reduces memory consumption, hence efficient deployment. Some of the optimization techniques applied in this framework include the following:

1. **Quantization**: CTranslate2 supports quantization methods like INT8 and INT16. Such storage of the weights and the activations of the model using their lower-precision datatypes, as is shown in Section 3.5.1.1, allows CTranslate2 to reduce the memory size and increase the computational throughput. This is done as a part of model conversion, so this does not affect the model's accuracy that much but does improve efficiency. It further supports static and dynamic quantization: wherever possible, the reduced precision during computation is used for saving time with no hit on accuracy.
2. **Efficient Runtime Execution**: The library is internally optimized at a low level for CPU and GPU runtime. CTranslate2 applies the AVX2, AVX512 instruction set for CPUs and add-ons for NVIDIA graphics cards using CUDA. These are able to make full use of the available parallel processing facilities in hardware. In addition to this, CTranslate2 has a fast-working memory allocator and a thread scheduler,

able to cut down the time taken for thread management and memory allocation and hence speed up inferences.

3. **Batch Processing and Parallelism**: CTranslate2 can merge a number of input sequences into one and hence can be utilized more efficiently with the hardware. In other words, it supports the running of multiple sequences in parallel, which raises the efficiency to a higher level.

4. **Memory-Efficient Layers**: CTranslate2 optimizes the transformer layer structure, particularly the memory-intensive operations like self-attention and feed-forward layers. It fuses some operations, reducing the number of memory copies between layers to lower the peak memory consumption. Therefore, the model will run with less latency on systems that have lower RAM or low memory capacity, like mobile and edge devices.

5. **On-Demand Loading of Model Weights**: For larger models, support is available in CTranslate2 for lazy loading of model weights-that is, model weights would be loaded only when required during inference. Regarding on-demand loading, only the most necessary weights would keep a residence in memory at any time. It comes in handy for low-memory systems on which bigger models might be necessary on tiny devices without running out of memory.

6. **Model Pruning and Sparsity Support**: It also optionally includes pruning of models-that is, all the unnecessary connections or neurons in the network get removed so that the model is lightweight. It reduces model size and latency through pruning of the unnecessary parts of the network. The support also provides sparse matrix operations; these can utilize sparsity in the pruned models to further accelerate computation.

These optimizations allow CTranslate2 to deploy transformer-based models in environments requiring high computational efficiency and low memory usage, such as mobile applications, edge devices, and real-time processing scenarios. Each technique is carefully implemented to ensure minimal impact on model accuracy while achieving significant improvements in performance.

3.5.1.3 Inner Workings of the Whisper Model

Whisper is based on a transformer architecture 3.5.1, which is optimized for sequence-to-sequence tasks, making it highly suitable for speech-to-text conversion. It operates by processing audio data—typically in the form of WAV files—and generating text outputs through its encoder-decoder mechanism.

- **Audio Processing and Feature Extraction**: Whisper first transforms the input audio signal into a spectrogram, a visual representation of the audio's frequency spectrum over time. This spectrogram serves as the input to the model, which extracts temporal and frequency-based features crucial for understanding spoken words.
- **Encoder-Decoder Mechanism**: Whisper's transformer model includes an encoder that processes the spectrogram and a decoder that translates the encoded features into text. The encoder maps the audio signal to a series of high-dimensional vectors representing sound features, while the decoder generates textual tokens based on these encoded features. This process enables Whisper to handle various speaking rates, accents, and environmental noise.

Whisper is trained on a large, diverse dataset, which gives it a high degree of generalization across audio conditions and languages. As a result, it performs well on both high-quality and degraded audio, making it robust for practical applications where audio clarity may vary(eg. walkie-talkie communications).

3.5.1.4 Performance Characteristics of Whisper

Whisper's performance is distinguished by its accuracy and noise resilience. Key characteristics include:

- **High Accuracy in Noisy Environments**: Whisper is particularly strong in noisy or challenging environments due to its extensive training data, which includes audio with various background noises. This feature is essential for transcribing walkie-talkie communications, where noise and interference are common.
- **Multilingual Capability**: Whisper can recognize and transcribe multiple languages without the need of language-specific models, which adds flexibility to systems that may encounter multilingual audio data.
- **Robustness to Accents and Audio Quality**: Whisper is capable of accurately transcribing different accents and various audio qualities, including low-bitrate or compressed audio, which makes it suitable for SDR-based applications where signal quality might be variable.

3.5.1.5 Using Faster-Whisper for Local Transcription

The selection of the most suitable model for this thesis was driven by two primary factors: **accuracy** and **speed**, both of which are essential for real-time transcription of walkie-talkie audio. Among the leading models in this field, **OpenAI's Whisper**[25] and **Facebook's Wav2Vec2**[26] are highly regarded due to their robust performance in automated speech recognition (ASR) tasks. Furthermore, several optimized versions of these models have been developed to enhance efficiency for specific applications. Notable examples include **WhisperCPP**[27], a C++ reimplementation of Whisper designed to improve execution speed on certain hardware, and **Faster-Whisper**[28], which utilizes **CTranslate2** 3.5.1.2 for optimized inference.

For evaluation, the **JIWER** (JImilarity Word Error Rate) [29] Python library was used to calculate transcription metrics. JIWER provides functions to compute key ASR metrics by comparing the reference transcription with the model outputs. It generates:

- **Word Error Rate (WER)**: Measures the rate of word-level errors by comparing the predicted transcription with the ground truth, making it a fundamental indicator of word-by-word accuracy.
- **Match Error Rate (MER)**: Focuses on sequence matching errors, providing insight into the structural accuracy of the transcription.
- **Word Information Lost (WIL)**: Evaluates the loss of essential information in the transcription, which is critical for assessing the preservation of context and meaning in real-time ASR.

Given the real-time nature of this application, the **elapsed transcription time** is also a crucial metric. In this study, audio samples were utilized from the **Harvard Speech Corpus** [30], a well-established dataset in ASR research. The transcription times and accuracy metrics are presented in the following figures (see Fig. 3.6 and Fig. 3.7). The tests were run using the same device and specifically on the CPU of the device, some models can utilize CUDA but since CUDA is not available on all devices a more general approach was chosen(CPU).

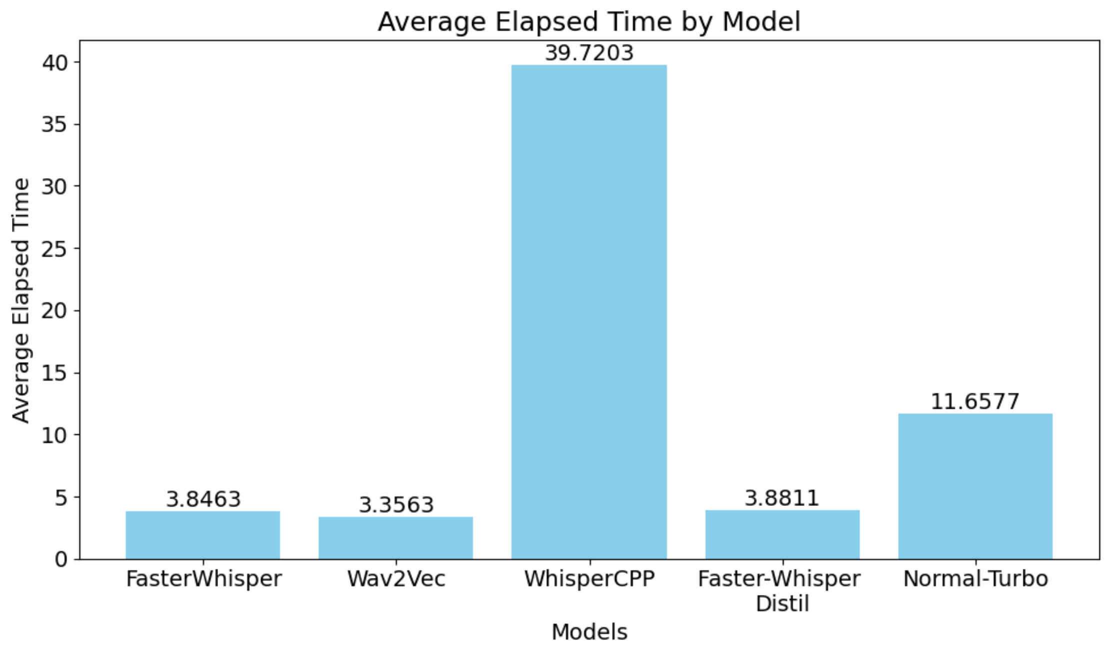


Figure 3.6: Average transcription time for each model (in seconds).

As shown in Fig. 3.6, WhisperCPP and OpenAI's standard Whisper model (Turbo) were considerably slower than other optimized models, rendering them unsuitable for real-time processing requirements. These models, despite their potential accuracy, were excluded from further consideration due to their latency.

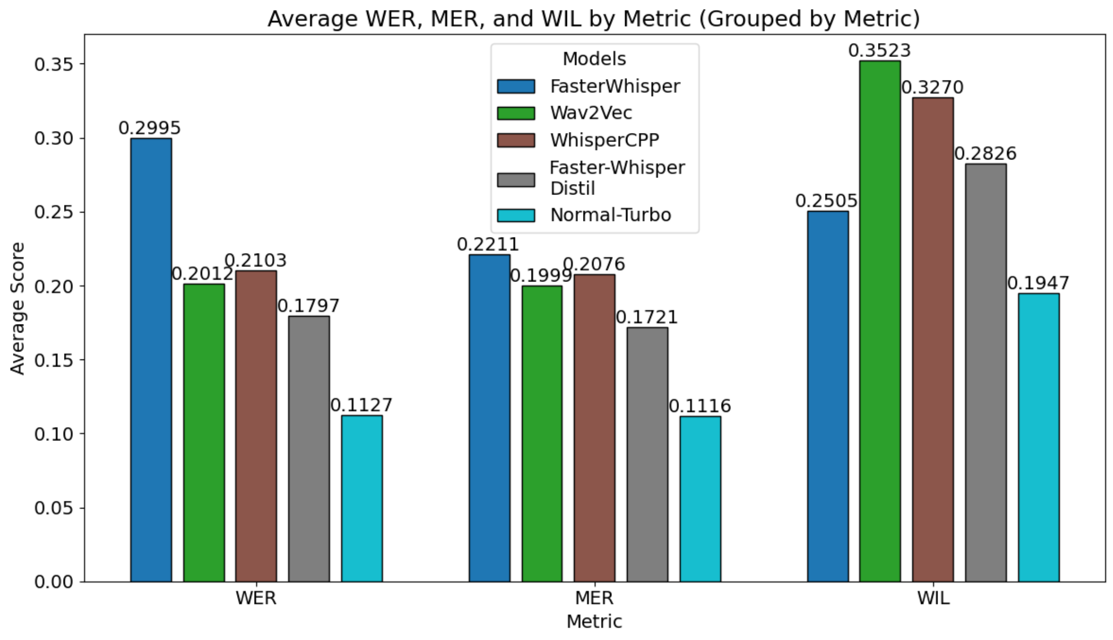


Figure 3.7: Average accuracy metrics (WER, MER, WIL) for each model.

When analyzing accuracy, the standard Whisper model displayed higher WER and WIL values compared to **Wav2Vec2** and the **Distilled version of Faster-Whisper**. However, the Distilled version of Faster-Whisper excelled in both word-by-word accuracy (WER) and contextual information retention (WIL), making it more favorable for real-time applications.

In conclusion, while OpenAI's Whisper model exhibited strong overall accuracy, the **Distilled Faster-Whisper model** emerged as the optimal choice due to its balanced performance in both speed and accuracy. This model not only meets the real-time processing demands but also provides reliable transcription accuracy for walkie-talkie audio, fulfilling the requirements of this thesis.

3.6 Challenges in Real-Time Signal Processing

Real-time signal processing is a set of complex challenges, particularly capturing, analyzing, and then transcribing the audio signals within a Software-Defined Radio framework. Among many, three major areas of challenges emerge from the literature: noise interference mitigation, signal acquisition process synchronization, and transcription delay management. Each of these aspects is very critical assuring that the processing of audio signals is not only effective but also precise; hence, competent methodologies and sophisticated algorithms are required to enhance performance and reliability for real-time applications.

3.6.1 Noise Reduction

Signal processing suffers from deterioration in signal quality due to external noise and interference signals from other electronic devices. Indeed, strong noises pose barriers to obtaining appropriate audio data. Noise cancellation is highly dependent on how effective the mechanism of filtering is in which the desired signal is to be separated from unwanted background noise without hampering the quality of the captured content. However, achieving such a balance in a real-time scenario is a highly computational task because complex algorithms for filtering may introduce latency, hence contrasting with the motivation of real-time processing.

3.6.2 Synchronization of Signal Capture

For multi-frequency/channel SDR applications, synchronization of signal capture is especially of critical importance. Capture must be synchronized at the moment the signal reaches a specific state to avoid timing errors within the recorded data. In the conditions of dynamic frequency hopping or to monitor different channels of walkie-talkies simultaneously, this problem worsens. While exact time-stamping and high-fidelity sampling mechanisms can help some systems meet these demands, they often require hardware that may not be suitable for portable, low-cost devices.

3.6.3 Signal Processing Delays in Transcription

Audio signals can be captured in real-time using a complex model such as OpenAI Whisper; however, this introduces delays and reduces any possible real-time scenarios even further. Faster versions (e.g., SYSTRAN's Faster-Whisper[28]) with smaller machine learning complexity and architecture can provide better efficiency of both inference engines and a wide usage of 8-bit quantization. The signal processing delays can lead to the system's degrading performance in various emergency scenarios, and therefore, the need for real-time speech transcription is becoming a mandate to avoid serious hazards.

3.6.4 DoA Challenges

Determining the direction of arrival (DoA) of a signal is a challenging task. One of the main challenges is dealing with noise, which can strongly impact the estimation's precision as well as the clustering. The other challenge is correctly matching a cluster(Speaker) to its associated transcription value to be interpreted reliably. The final challenge is accurate array calibration, since small misalignments can produce spurious estimates of the DoA.

In summary, real-time signal processing in SDR applications involves navigating a balance among computational efficiency, signal quality, and prompt transcription. These challenges necessitate careful consideration of hardware capabilities, software optimization, and algorithm selection to ensure reliable performance in time-sensitive environments.

Chapter 4 System Architecture and Implementation

4.1 System Overview

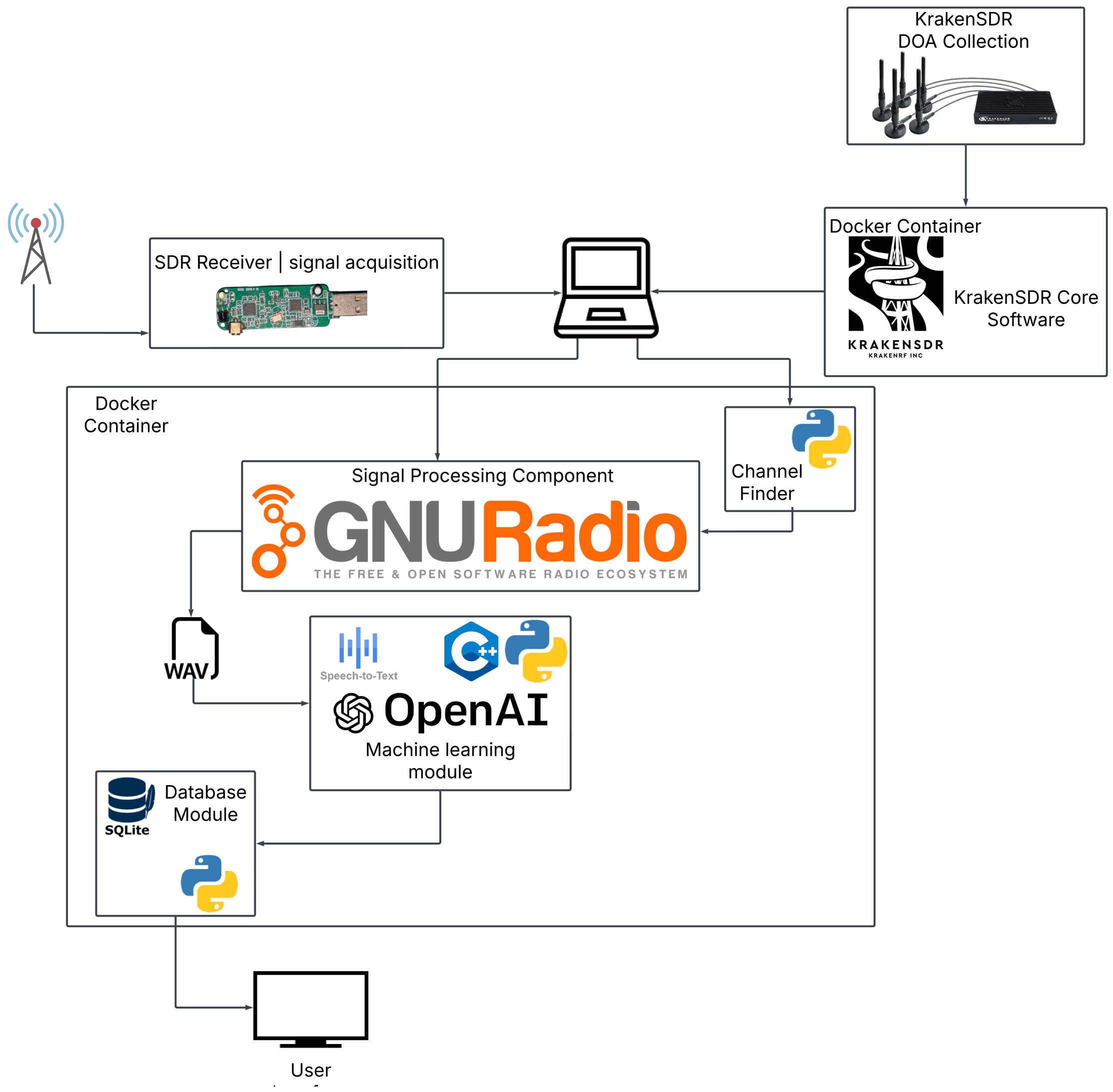
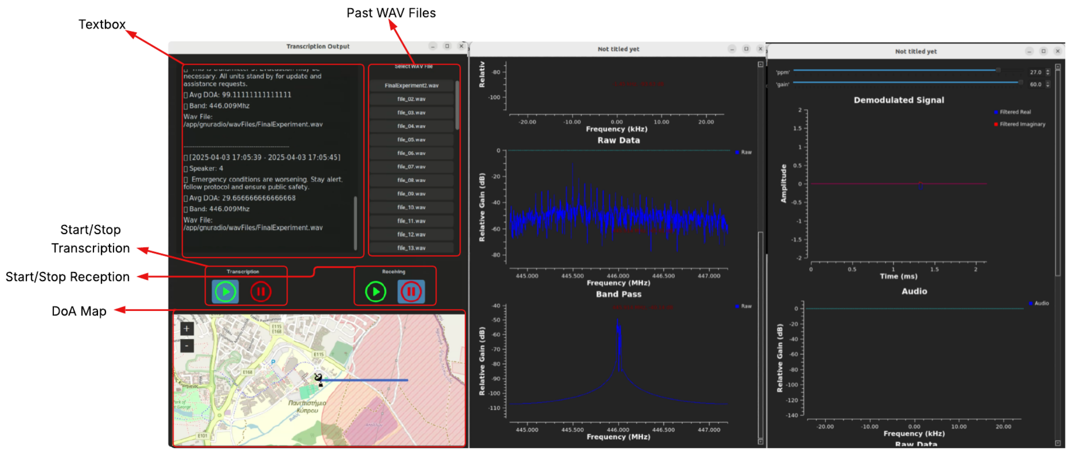


Figure 4.1: Representation of the system structure.

4.1.1 Frontend architecture

Figure 4.2 showcases the frontend of the platform. Specifically, at the top part (leftmost) there is a text box displaying the timestamp of the transcribed text as well as the speaker classified. On the right part of the text box, the user can query the database of previously transcribed WAV files. Start/stop buttons for transcription and reception are located at the bottom of the Transcription Output window (leftmost), enabling/disabling the WAV-to-text model and the RTL-SDR, respectively. The map below the buttons displays the receiver, with a blue line representing the DOA of the received signal. On the right of the Transcription Output window there is the signal processing window from the GNU Radio where it showcases the Raw received data, the results of passing the signal through the Band Pass filter and the Demodulated signal (rightmost) as well as the audio received after demodulation.


necessary. All units stand by for update and assistance requests.
Avg DOA: 99.11111111111111
Band: 446.009Mhz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav

[2025-04-03 17:05:39 - 2025-04-03 17:05:45]
Speaker: 4
Emergency conditions are worsening. Stay alert, follow protocol and ensure public safety.
Avg DOA: 29.666666666666668
Band: 446.009Mhz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav

Figure 4.2: Frontend system overview.

4.1.2 Back end architecture

The backend system is responsible for signal acquisition, processing, transcription, and data storage. It operates entirely within a Docker container (3.1) and includes the following components:

1. **RTL-SDR Signal Acquisition**
    - The RTL-SDR hardware (Figure 3.2) captures raw radio signals from walkie-talkies.
    - These signals are streamed into the backend for further processing.
2. **GNU Radio Signal Processing**
    - GNU Radio (Subsection 2.2) processes the raw signals, applying filtering and demodulation techniques (Subsection 3.3.2).
    - The processed signals are converted into audio files in `.wav` format, suitable for transcription.

3. **Channel Finder**
    - A Python-based script scans a predefined frequency range and identifies active channels.
    - The script configures GNU Radio to tune to the selected frequency for optimal signal capture.
4. **Speech-to-Text Conversion**
    - The audio files generated by GNU Radio are transcribed into text using the OpenAI Whisper model (Subsection 3.5).
    - Metadata, including timestamps and source frequency, is added to the transcriptions.
5. **SQLite Database for Data Storage**
    - The transcribed text, along with associated metadata, is stored in an SQLite database.
    - This database serves as the central repository for transcription data, enabling efficient querying and retrieval.

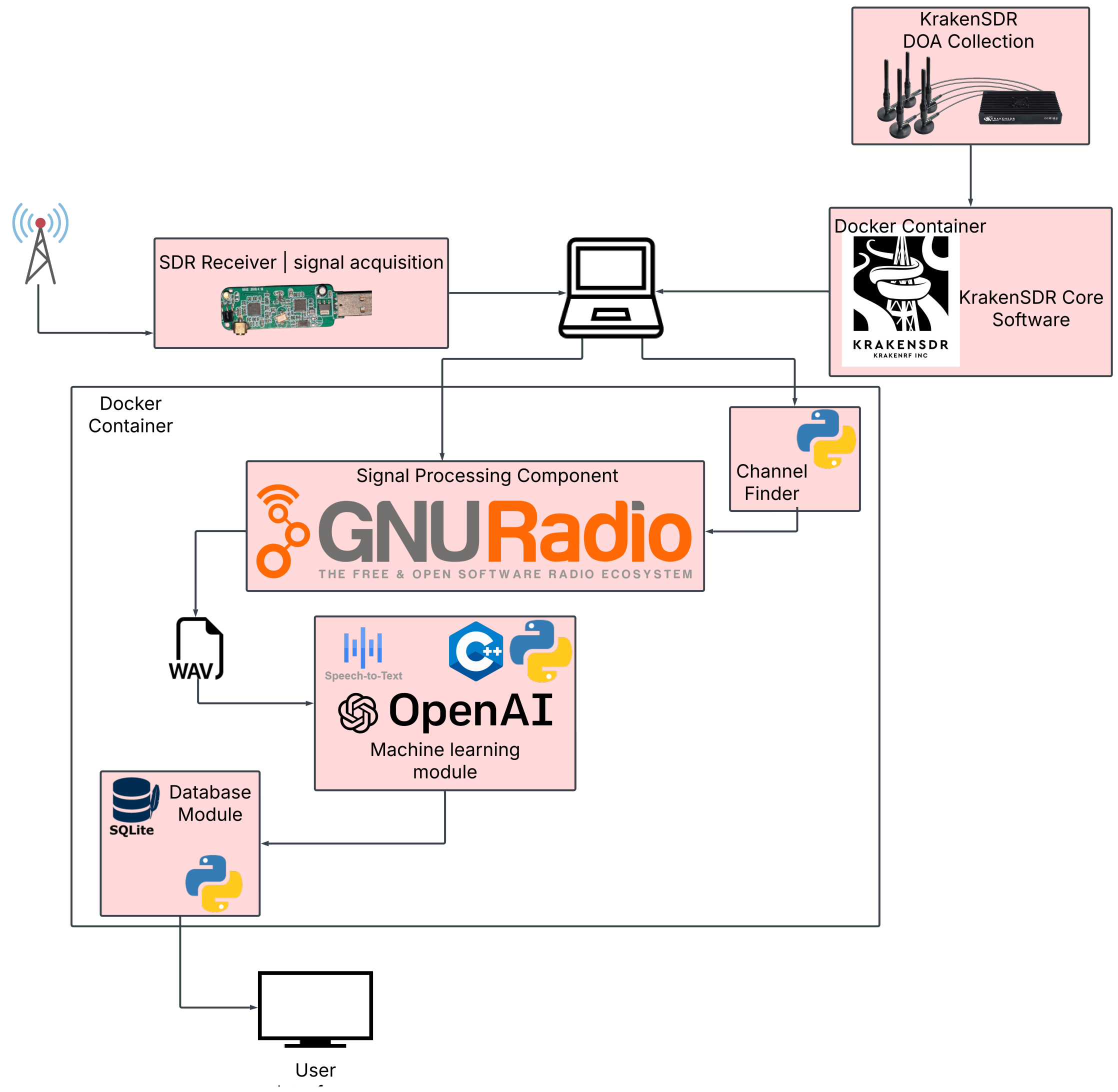


Figure 4.3: Backend System Architecture.

The backend Figure (4.3) (red parts showcase the backend parts) operates autonomously once initiated by the frontend, processing signals and storing the results for later retrieval and visualization.

4.2 Calibrating the RTL-SDR

4.2.1 Parts Per Million (PPM)

As previously discussed in Subsection 3.3.1.1, it is important to calibrate the RTL-SDR (Software Defined Radio) due to frequency deviations that may compromise the accuracy of received signals. This is done by finding a specific known frequency. Since the specific frequencies used by the walkie-talkie are known, an appropriate channel can be selected. Channel 3 is selected at 446.03125 MHz, and then these steps are followed:

The steps below outline the process for calibrating the RTL-SDR device. This involves measuring the frequency offset, calculating the correction in PPM, and plotting the power spectrum to visualize the calibration results.

1. **Initialize the RTL-SDR device:**
   - Set the sampling rate.
   - Set the center frequency to the known frequency (e.g., walkie-talkie channel frequency).
   - Enable automatic gain control.
   - Apply an initial PPM correction if available.
2. **Capture radio samples:**
   - Collect a block of raw IQ samples from the SDR device.
3. **Perform frequency analysis:**
   - Compute the FFT (Fast Fourier Transform) of the captured samples.
   - Shift the FFT output to center the frequency components.
   - Calculate the power spectrum from the FFT results.
   - Generate the corresponding frequency axis.
4. **Identify the peak frequency:**
   - Find the frequency with the highest power in the spectrum.
5. **Calculate frequency offset and PPM correction:**
   - Compute the difference between the known frequency and the measured frequency.
   - Calculate the recommended PPM correction using the formula:

$$\text{PPM correction} = \frac{\text{Known Frequency} - \text{Measured Frequency}}{\text{Known Frequency}} \times 10^6$$

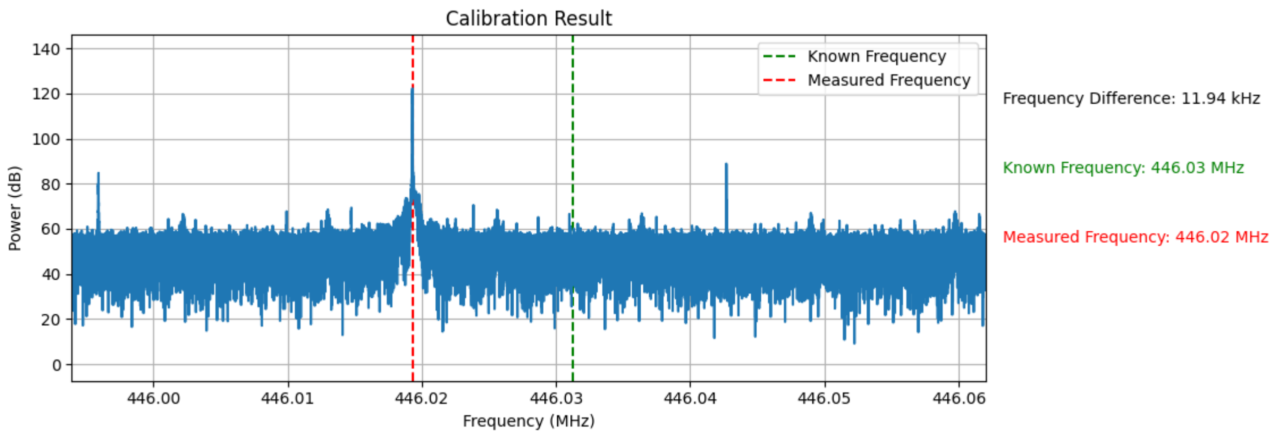
6. **Visualize results:**
   - Plot the power spectrum, highlighting the known and measured frequencies.

    - Display the frequency offset and recommended PPM correction.

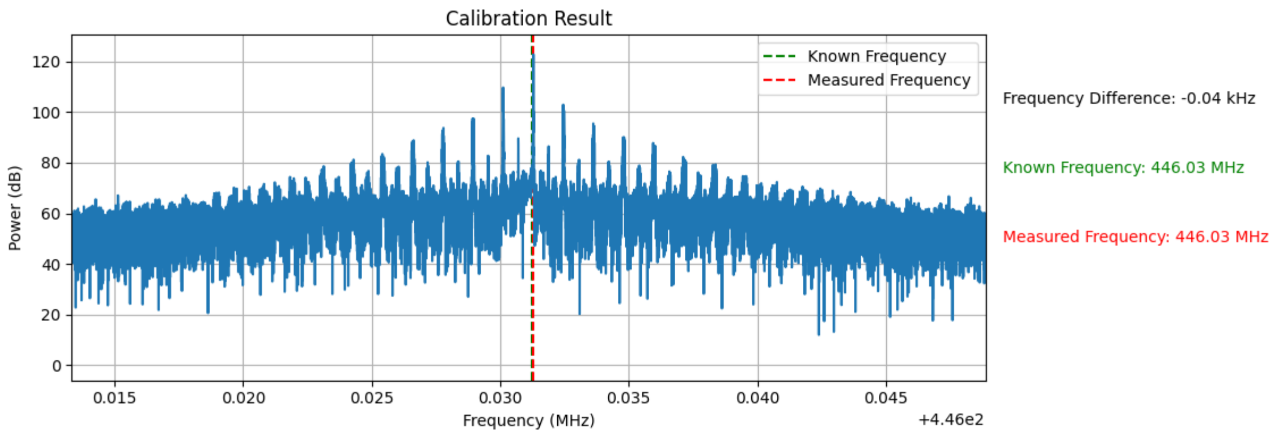
7. **Close the RTL-SDR device:**
    - Release resources and terminate the SDR connection.

4.2.1.1 Calibration Procedure Output

- **Setup:** The known frequency represents a stable signal source (e.g., walkie-talkie channel).
- **Frequency Offset:** The peak in the power spectrum indicates the actual received frequency.
- **PPM Correction:** This value adjusts for hardware-specific offsets in the SDR, improving future frequency accuracy.
- **Visualization:** The plot provides a graphical confirmation of the calibration results.



(a) Comparison between known and measured frequency before calibration



(b) Comparison between known and measured frequency after calibration (27PPM)

Figure 4.4: PPM Correction of RTL-SDR

As seen in Figure 4.4, before calibrating the PPM, there is a frequency difference of 12 kHz between the actual frequency and the measured frequency. While this may not seem concerning, it is significant in this case, where the channels of the walkie-talkies are very close together. For example, channel 2 operates at 446.01875 MHz and channel 3 at 446.03125 MHz, with only a 12.5 kHz difference between them. With such a frequency inconsistency, attempting to listen on channel 2 would result in the RTL-SDR receiving data from channel 3 instead. After applying a 27 PPM calibration, the frequency difference is reduced to just 0.04 kHz, which is acceptable for this purpose.

4.3 Channel Scanning and Signal Capture

4.3.1 Channel Scanning

The following pseudocode outlines the logic for scanning predefined frequencies, analyzing the signal spectrum, detecting peaks, and identifying the closest active channel.

Listing 4.1: Channel Finding Pseudocode

```
// Initialize the SDR device
initialize SDR
set sample rate to 2.048 MHz
enable automatic gain control
apply PPM correction (e.g., 23 ppm)

// Define a list of predefined frequencies (channels)
frequencies = [446.00625 MHz, 446.01875 MHz, ..., 446.09375 MHz]

// Define a target frequency for scanning
target_frequency = 446.05 MHz

// Function to scan a frequency
function scan_frequency(frequency, sdr):
    set SDR center frequency to frequency
    capture samples from SDR
    return samples

// Function to compute power spectrum
function detect_signal(samples):
    perform FFT on samples
    compute power spectrum from FFT results
    return power spectrum

// Function to detect peaks in the power spectrum
function detect_peaks(power, multiplier = 10):
    calculate threshold as multiplier * median(power)
    find peaks above threshold
    return peaks and peak properties

// Main function for channel finding
function find_channel():
    initialize SDR and configure parameters

    // Scan the target frequency
    samples = scan_frequency(target_frequency, sdr)
    power = detect_signal(samples)
    peaks, properties = detect_peaks(power)

    // Check for significant peaks
    if peaks are not empty:
        find the peak with the highest amplitude
        calculate the frequency of the highest peak
        match the peak frequency to the closest predefined channel
        return the closest channel and its number
    else:
        print "No significant peaks detected"
        return None

    close SDR connection
end function

// Execute the channel finding process
find_channel()
```

4.3.2 Signal Capture

The signal capture happens via Gnuradio (Section 2.2), the blocks used and their purpose are explained below.

4.3.2.1 GNU Radio Modules

The flowchart shown in Figure 4.5 demonstrates the key blocks used in GNU Radio. Below is an explanation of each block, its role in the flowchart, and its generic purpose in signal processing applications.

1. **RTL-SDR Source**
   - **Role in the Flowchart:** Configures the RTL-SDR device to capture raw IQ samples from a specified center frequency (e.g., 446.031 MHz) with a sample rate of 2.4 MHz. Gain and frequency correction are applied.
   - **Generic Purpose:** Interfaces with RTL-SDR devices to receive RF signals, enabling their conversion into digital IQ data for further processing.
2. **Band Pass Filter**
   - **Role in the Flowchart:** Passes only the frequency components within a specific range, defined by the low cutoff frequency (-6.25 kHz) and high cutoff frequency (+6.25 kHz), suppressing out-of-band noise.
   - **Generic Purpose:** Isolates the desired signal from surrounding noise or interference in the frequency spectrum.
3. **Low Pass Filter**
   - **Role in the Flowchart:** Further filters the signal, removing high-frequency components outside the desired range (cutoff frequency: 15 kHz), ensuring only relevant frequency components are preserved.
   - **Generic Purpose:** Removes noise and reduces bandwidth, focusing on the essential signal.
4. **Noise Threshold**
   - **Role in the Flowchart:** Demodulates the Wideband FM (WBFM) signal, converting it into a baseband audio signal with reduced bandwidth. The ‘audio decimation‘ parameter of 10 reduces the sample rate(by choosing 1 in N datapoints), optimizing the signal for audio playback or analysis.

- **Generic Purpose:** Filters out low-power noise or irrelevant signals in the signal chain, improving signal clarity.

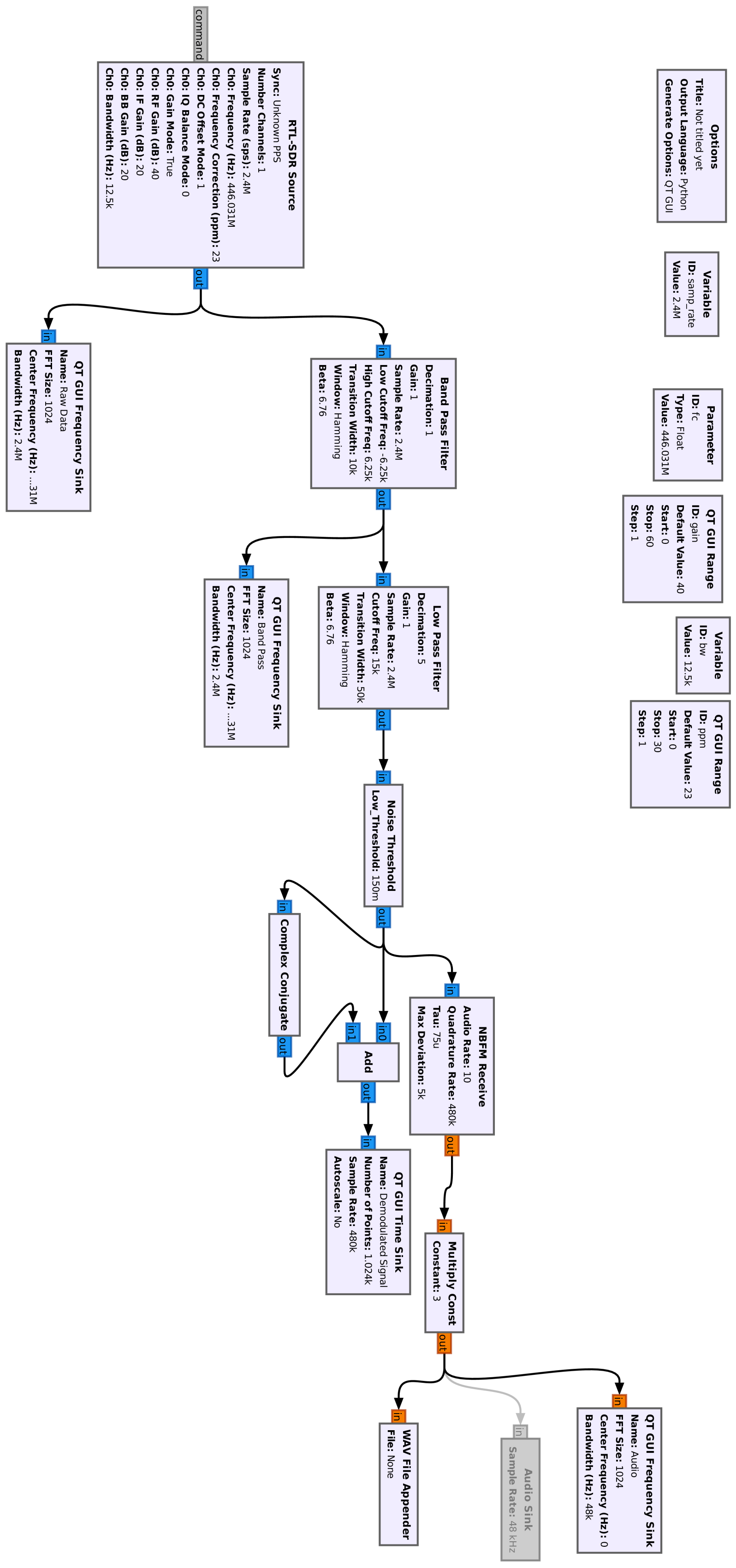
5. **WBFM Receive**
    - **Role in the Flowchart:** Demodulates the Wideband FM (WBFM) signal, converting it into a baseband audio signal with reduced bandwidth (audio decimation: 10).
    - **Generic Purpose:** Converts FM-modulated signals into audio signals for further analysis or playback.
6. **Multiply Constant**
    - **Role in the Flowchart:** Multiplies the signal by a constant value (3), scaling its amplitude for visualization or storage.
    - **Generic Purpose:** Adjusts signal amplitude for normalization or to match downstream processing requirements.
7. **Add**
    - **Role in the Flowchart:** Combines two input signals: the main signal and its complex conjugate, to enhance or process the signal further.
    - **Generic Purpose:** Used to sum signals or perform mathematical operations on multiple inputs.
8. **Complex Conjugate**
    - **Role in the Flowchart:** Calculates the complex conjugate of the input signal, often used for phase correction or correlation tasks.
    - **Generic Purpose:** Provides signal manipulation for advanced processing tasks, such as phase alignment.
9. **WAV File Appender**
    - **Role in the Flowchart:** Stores the processed audio signal in a WAV file for offline playback or further analysis.
    - **Generic Purpose:** Writes audio data to a standard file format for storage or post-processing.
10. **QT GUI Frequency Sink (Raw Data and Band Pass)**
    - **Role in the Flowchart:** Displays the frequency-domain representation of the signal at various stages (e.g., raw IQ data, bandpass filtered signal).

- **Generic Purpose:** Provides real-time spectral analysis for monitoring signal frequency components.

11. **QT GUI Time Sink**

    - **Role in the Flowchart:** Displays the time-domain waveform of the demodulated signal, showing amplitude variations over time.
    - **Generic Purpose:** Visualizes real-time time-domain characteristics of the processed signal.

Two custom blocks, **Noise Threshold** and **WAV File Appender**, were created in Python to fulfill the specific requirements of this pipeline. These blocks were developed following the GNU Radio documentation for Embedded Python Blocks, ensuring seamless integration and efficient processing.



Options
Title: Not titled yet
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2.4M

Parameter
ID: fc
Type: Float
Value: 446.031M

QT GUI Range
ID: gain
Default Value: 40
Start: 0
Stop: 60
Step: 1

Variable
ID: bw
Value: 12.5k

QT GUI Range
ID: ppm
Default Value: 23
Start: 0
Stop: 30
Step: 1

RTL-SDR Source
Sync: Unknown PPS
Number Channels: 1
Sample Rate (sps): 2.4M
Ch0: Frequency (Hz): 446.031M
Ch0: Frequency Correction (ppm): 23
Ch0: DC Offset Mode: 1
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: True
Ch0: RF Gain (dB): 40
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20
Ch0: Bandwidth (Hz): 12.5k

QT GUI Frequency Sink
Name: Raw Data
FFT Size: 1024
Center Frequency (Hz): ...31M
Bandwidth (Hz): 2.4M

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 2.4M
Low Cutoff Freq: -6.25k
High Cutoff Freq: 6.25k
Transition Width: 10k
Window: Hamming
Beta: 6.76

QT GUI Frequency Sink
Name: Band Pass
FFT Size: 1024
Center Frequency (Hz): ...31M
Bandwidth (Hz): 2.4M

Low Pass Filter
Decimation: 5
Gain: 1
Sample Rate: 2.4M
Cutoff Freq: 15k
Transition Width: 50k
Window: Hamming
Beta: 6.76

Noise Threshold
Low_Threshold: 150m

Complex Conjugate

Add

NBFM Receive
Audio Rate: 10
Quadrature Rate: 480k
Tau: 75u
Max Deviation: 5k

QT GUI Time Sink
Name: Demodulated Signal
Number of Points: 1.024k
Sample Rate: 480k
Autoscale: No

Multiply Const
Constant: 3

WAV File Appender
File: None

Audio Sink
Sample Rate: 48 kHz

QT GUI Frequency Sink
Name: Audio
FFT Size: 1024
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Figure 4.5: Gnuradio flowchart signal capture.

4.3.2.2 Noise Threshold Block

The **Noise Threshold** block is designed to filter out weak signals by applying a threshold to the signal's magnitude. Signals with magnitudes below the specified threshold are set to zero, effectively removing low-power noise and enhancing the signal's quality.

Listing 4.2: Noise Threshold Block

```
// Custom GNU Radio block: Noise Threshold
// Input: Complex signal
// Output: Complex signal with weak signals zeroed out

function NoiseThreshold(low_threshold):
    initialize block with input and output signatures as complex signals
    set low_threshold to the provided value

function process(input_signal):
    magnitude = compute magnitude of input_signal
    output_signal = set values of input_signal to 0 if magnitude < low_threshold
    return output_signal
```

4.3.2.3 WAV File Appender

The **WAV File Appender** block is used to save processed audio signals into a WAV file. It appends data to an existing WAV file or creates a new one if it does not exist. This ensures that the processed audio can be stored for offline analysis or playback. The block manages file operations, including updating the WAV file header when the flowgraph stops.

Listing 4.3: WAV File Appender Block

```
// Custom GNU Radio block: WAV File Appender
// Input: Audio samples
// Output: None (data is saved to a file)

function WAVFileAppender(file_path):
    initialize block with input signature as float
    set file_path to the provided value
    if file exists:
        open file for appending and read header
    else:
        create a new WAV file with appropriate parameters

```

```
function process(input_signal):
    convert input_signal from float to int16
    append converted data to the WAV file

function stop():
    update WAV file header with the correct frame count
    close the file
```

4.4 Speech-to-Text Pipeline Using Whisper

The WAV file is processed by the speech-to-text pipeline to transcribe spoken audio into text. To achieve real-time processing, the pipeline reads and processes the audio incrementally as the WAV file is being created. This approach ensures minimal latency between audio capture and transcription.

4.4.1 Real-Time Audio Processing

Real-time processing is implemented through a combination of efficient audio reading, silence detection, and transcription. The key components of this pipeline are described below:

4.4.1.1 Audio Reading Function

The `read_new_audio` function reads new audio data from the WAV file incrementally, starting from the last processed position [Listing 4.4]. It ensures the audio is prepared for transcription by:

- Reading only the newly written audio frames from the specified position in the file.
- Resampling the audio to 16 kHz to meet Whisper's input requirements.
- Converting the audio to mono by averaging across channels.
- Returning the new audio data, the updated position, and the corresponding time offset.

Listing 4.4: Pseudocode for `read_new_audio`

```
function read_new_audio(file, last_position):
    try:
        new_data, sample_rate = load audio from file starting at last_position
        if no frames read:
            return None, last_position, 0.0

        time_offset = last_position / sample_rate
        resample new_data to 16 kHz
        convert new_data to mono
        update last_position with the number of frames read
        return new_data, updated last_position, time_offset
    except error:
        return None, last_position, 0.0
```

4.4.1.2 Silence Detection Function

The `is_silence` function determines whether a segment of audio represents silence by comparing its maximum amplitude to a predefined threshold. This function is critical for tasks such as detecting when to pause transcription or identifying silent sections for post-processing.

Listing 4.5: Pseudocode for `is_silence`

```
function is_silence(data, threshold=0.1):
    return true if maximum amplitude of data < threshold
```

4.4.1.3 Silence Checking Function

The `checkForSilence` function monitors the audio file for consecutive silent segments of a specified duration. It reads new audio using `read_new_audio` and increments a silence duration counter when silence is detected. If non-silent audio is encountered, the counter is reset. Once the specified duration of silence is reached, the function returns the updated file position.

Listing 4.6: Pseudocode for `checkForSilence`

```
function checkForSilence(file, plot_stop_flag, last_position, seconds=5):
    initialize silence_duration = 0
    sample_rate = 16000

    while silence_duration < seconds and plot_stop_flag is not set:
        new_data, last_position, time_offset = read_new_audio(file, last_position)
        if new_data is None:
            wait 1 second
            continue

        duration = length of new_data / sample_rate
        if is_silence(new_data):
            increment silence_duration by duration
        else:
            reset silence_duration to 0

        wait 1 second

    if silence_duration >= seconds:
        return last_position
```

4.4.1.4 Integration with Transcription

The `transcribe_audio` function coordinates the transcription process by:

- Incrementally reading new audio data from the file.
- Sending the data to Whisper for transcription.
- Adjusting transcription timestamps to account for the file's offset timing.
- Storing the results in a database for retrieval.

The integration of these components enables seamless real-time transcription while the WAV file is being created.

Listing 4.7: Pseudocode for `transcribe_audio`

```
function transcribe_audio(wavFile, model):
    last_position = 0
    connect to database

    while stop_flag is not set:
        if wavFile exists:
            new_data, last_position, time_offset = read_new_audio(wavFile,
                last_position)
            if new_data is not None:
```

```
            segments = transcribe new_data with model
            for segment in segments:
                adjusted_start = segment.start + time_offset
                adjusted_end = segment.end + time_offset
                store transcription in database
    else:
        print "wavFile does not exist."

    wait 1 second
```

4.5 DoA Implementation

The KrakenSDR system timestamps and stores DoA values for analysis in the future. DoA values and speech transcriptions with timestamps are then combined in an attempt to perform speaker diarization through the following sequence of operations:

1. Set a start timestamp at audio record (e.g., 15:00:00)
2. Transcribe speech segments with start/end timestamps in relation
3. DoA values match with a starting time.
4. DBSCAN clustering clusters similar DoA readings
5. Centroids of clusters represent individual speakers

Table 4.1: Example of audio Transcription Data.

Audio Start	**Audio End**	**Transcription**
15:00:00	15:00:10	This is location 1

Table 4.2: Example of direction of arrival (DoA) measurements over time.

Timestamp	**DoA (°)**
15:00:00	190
15:00:01	189
15:00:02	190
15:00:03	191
15:00:04	192
15:00:05	190
15:00:06	189
15:00:07	190
15:00:08	190
15:00:09	191
15:00:10	189

Table 4.3: Example of diarization results.

Start Time	**End Time**	**Text**	**Speaker**
15:00:00	15:00:10	This is location 1	1

Listing 4.8: Speaker Diarization Algorithm

```
function process_transcription(start, end, wavFile):
    // Retrieve temporal DoA data
    ref_time = get_reference_time()
    absolute_start = ref_time + start
    absolute_end = ref_time + end
    DoA_data = retrieve_DoA(absolute_start, absolute_end)

    if no DoA_data:
        return empty list
    // Cluster processing
    initialize dbscan with eps=20, min_samples=5
    clusters = dbscan.fit_predict(DoA_data)
    // Calculate cluster centroids
    valid_clusters = filter_noise(clusters)
    centroids = calculate_centroids(DoA_data, valid_clusters)

    // Speaker matching process
    initialize transcription_speakers as empty list

    for each centroid in cluster_centroids:
        set matched = False

        // Check against known speakers
        for each (known_centroid, speaker_id) in cluster_to_speaker:
            if absolute(centroid - known_centroid) <= 10: // Threshold check
                append speaker_id to transcription_speakers
                set matched = True
                break loop // Exit speaker search
        // Handle new speaker case
        if not matched:
            add centroid to cluster_to_speaker with next_speaker_id
            append next_speaker_id to transcription_speakers
            increment next_speaker_id by 1

    // Recursive processing
    if multiple speakers detected:
        mid_point = (start + end) / 2
        process_transcription(conn, start, mid_point, wavFile)
        process_transcription(conn, mid_point, end, wavFile)
    else:
        save_results(conn, start, end, speakers[0], wavFile)

    return speakers
```

Chapter 5 Experimental Results

The system analysis was divided into three parts:

- Walkie-Talkie Signal Capture Analysis
- Speech-To-Text Analysis
- DoA Classification Analysis

5.1 Walkie-Talkie Signal Capture Analysis

The signal capture analysis involved transmitting a signal using an off-the-shelf walkie-talkie and capturing it at varying distances. The distances captured were between 100 and 600 m with increments of 100 m. A transmitter (walkie-talkie) was set up on top of a house at about 4 meters height and 0m as seen on the Figure 5.1. The receiver was then gradually moved away from the transmitter while recording the transmitted signal. The signal was captured and analyzed using FFT [3.2.1]. Specifically, the receiver was tuned to a specific frequency (446.00625 MHz) and collected IQ data points, which were then processed through the FFT to extract the power spectrum. It is important to note that the amplitude values provided by the FFT are not absolute measurements but rather relative values used for analysis. Figure 5.1 shows the experimental setup.

Transmitter: Motorola T92 H2O 446MHz.
Receiver: The RTL-SDR receiver was calibrated as explained in Section 4.2.

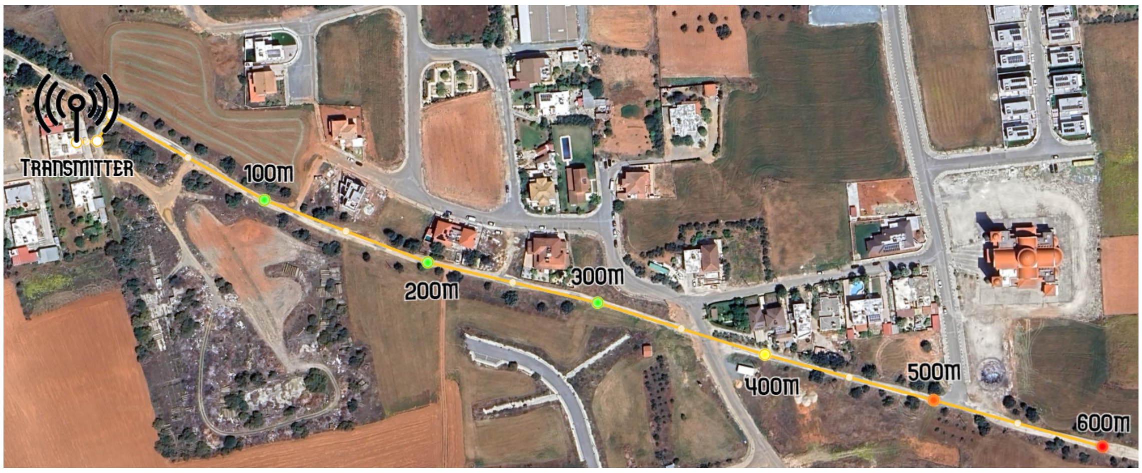


Figure 5.1: Map showcasing setup for Distance Experiment.

5.1.1 Walkie-Talkie Signal Capture Analysis Results

Figure 5.2 presents the relationship between distance and received power, with the distance represented on the x-axis and the corresponding received power displayed on the y-axis. The illustration demonstrates how variations in distance influence the amplitude of the received power.

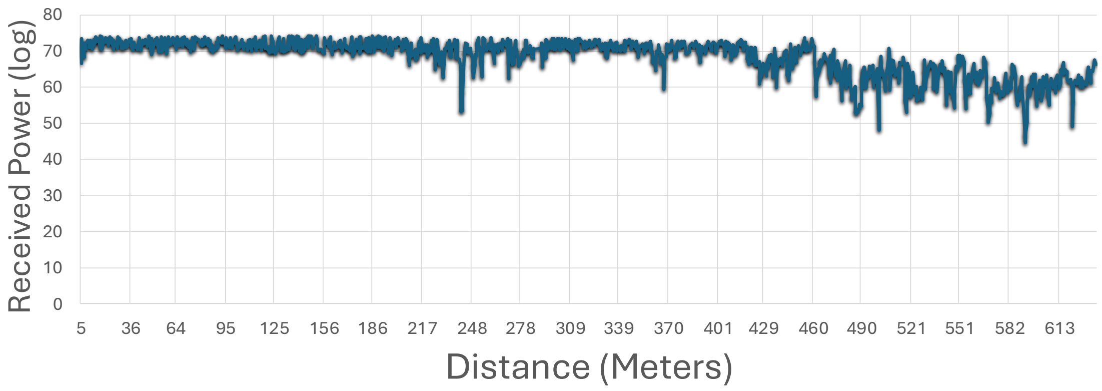


Figure 5.2: Graph showing the power transmission compared to the distance.

5.2 Speech-To-Text Analysis

To evaluate the speech-to-text accuracy and distance capture capability, a predefined phrase was used as the ground truth. The phrase was:

”This is a distance test {number}”

The {number} corresponds to the distance in meters, where:

- 1 → 100m
- 2 → 200m
- 3 → 300m
- 4 → 400m
- 5 → 500m
- 6 → 600m

The test was conducted with the following distances as also shown in Table 5.1:

Table 5.1: Test phrases and corresponding distances

Text	**Distance**
This is a distance test 1	100m
This is a distance test 2	200m
This is a distance test 3	300m
This is a distance test 4	400m
This is a distance test 5	500m
This is a distance test 6	600m

5.2.1 Results

The recorded speech-to-text outputs with corresponding timestamps are as follows:

- **[0.00 - 5.18]:** ”This is a distance test one.” (100m)
- **[6.50 - 9.98]:** ”This is in distance test two.” (200m)
- **[11.68 - 15.86]:** ”This is a distance test three.” (300m)
- **[17.34 - 21.36]:** ”This is a distance test four.” (400m)
- **[21.36 - 23.36]:** ”Thank you.”
- **[29.92 - 30.12]:** ”I.”
- **[30.12 - 31.16]:** ”This is a great”
- **[31.16 - 31.42]:** ”chance”
- **[31.42 - 31.98]:** ”and C.”
- **[31.98 - 32.92]:** ”See.”

The analysis of the results indicates a clear correlation between distance and transcription accuracy. Within the range of 100 meters to 400 meters, the transcription process is successful. However, as we extend beyond 500 meters, the accuracy of the transcriptions deteriorates significantly, resulting in fractured and incomplete outputs. This aligns with the observations presented in Figure 5.2, which illustrates that the signal power remains robust and stable up to approximately 410 meters. Beyond this threshold, there is a notable decline in signal strength and stability. This instability likely contributes to the incorrect and fractured transcription. Consequently, it can be inferred that maintaining signal integrity is critical for achieving accurate transcription results, particularly in scenarios that require communication over extended ranges.

A spectral analysis was conducted on the captured audio to further examine the signal quality. The waveform representations of the recorded audio at different distances are shown in Figure 5.3.

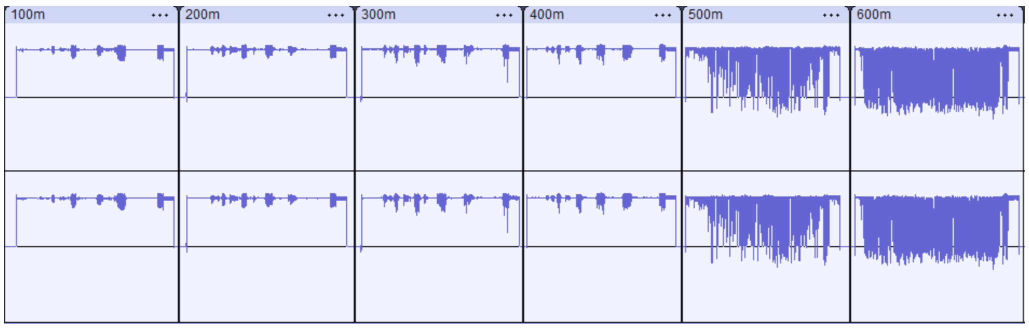


Figure 5.3: Waveform analysis of the captured audio at various distances.

The spectral analysis confirms again that at 400m distance, the signal is strong enough with speech patterns that can be understood. However above 500m there is a noticeable increase in background noise, causing significant distortion in the captured audio. This noise interference can justify the speech-to-text system produced wrong or fragmented transcriptions at these distances.

Signal deterioration can occur naturally as the signal weakens over distance. Although the path between the obstacles and receiver was mostly clear, a few trees were present and could affect the signal slightly. Vegetation can absorb or scatter signals, especially at certain frequencies. However, in this study, such small obstacles are considered normal and expected in typical real-world conditions.

In summary, all tests conducted showed that the signal remained generally stable and transcription was possible for up to 400m. Beyond this distance, the signal became too unstable, with excessive noise preventing accurate transcription.

5.3 DoA Classification Analysis

For the Direction of Arrival (DoA) Classification Analysis, an experiment was conducted in an outdoor field setting. In this experiment, the receiver, specifically the KrakenSDR, was positioned in a static location. Meanwhile, the transmitting device was moved to three predetermined locations within the field. These locations were selected based on their known coordinates and established as the ground truth for the analysis.

To evaluate the effectiveness of the DoA data collected, clustering was performed on the data obtained from these distinct positions. The clustering process involved analyzing the signals received at each of the fixed locations to identify patterns and groupings that could indicate the direction from which the signals originated. By comparing these clusters against the ground truth locations, we aimed to assess the accuracy of the DoA classification methodology. This comprehensive approach not only facilitated a structured evaluation of the DoA data but also contributed to a deeper understanding of the system's performance in real-world scenarios.
The figure below shows the setup for the experiment.

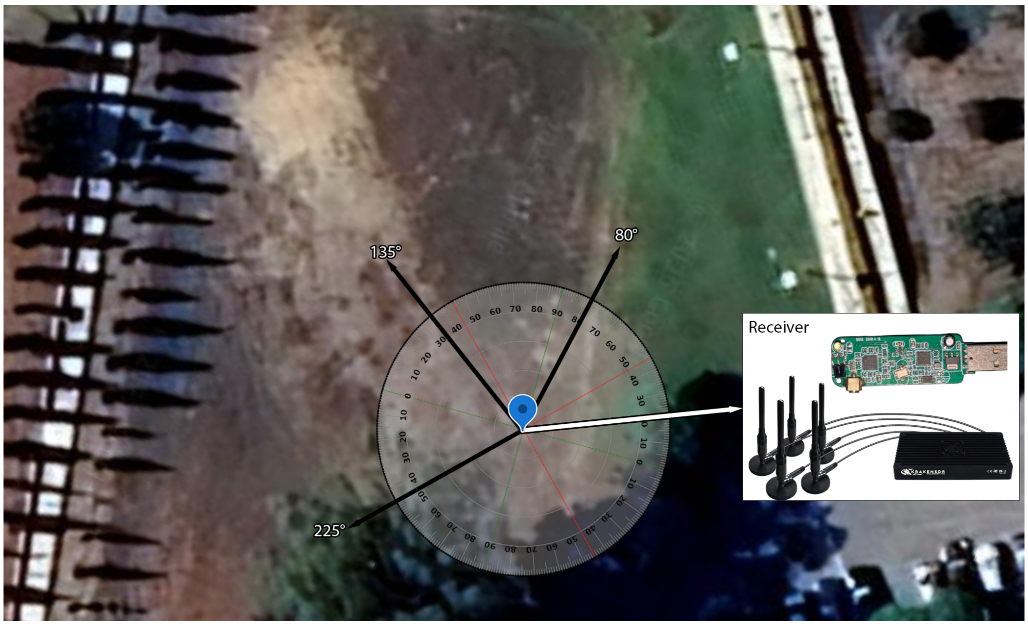


Figure 5.4: DoA experiment setup.

5.3.1 DoA Classification Analysis Results

The DoA was clustered and placed on a polar map with 0 Degrees representing north. The 3 different colors and symbols show the 3 clusters respectively, while the black color shows the ground truth which is the actual position as captured in the experiment.

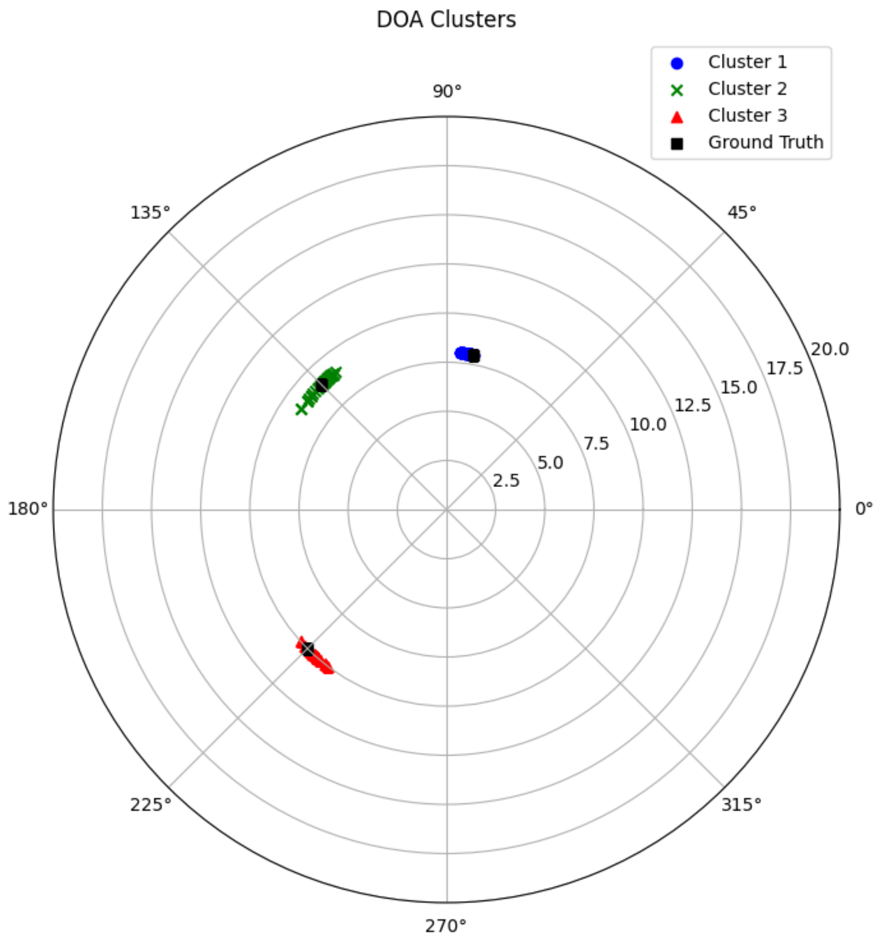


Figure 5.5: DoA experiment results.

As seen by the Figure 5.5, the captured DoA values are close to the ground truth hence using this system, speakers can be assigned to the data.

5.4 Real Life Experiment

In this section a real life experiment will be analyzed that was conducted similarly to the DoA experiment setup Figure 5.5. Trasmitters were set up on 4 locations around the receiver as show in Figure 5.6.

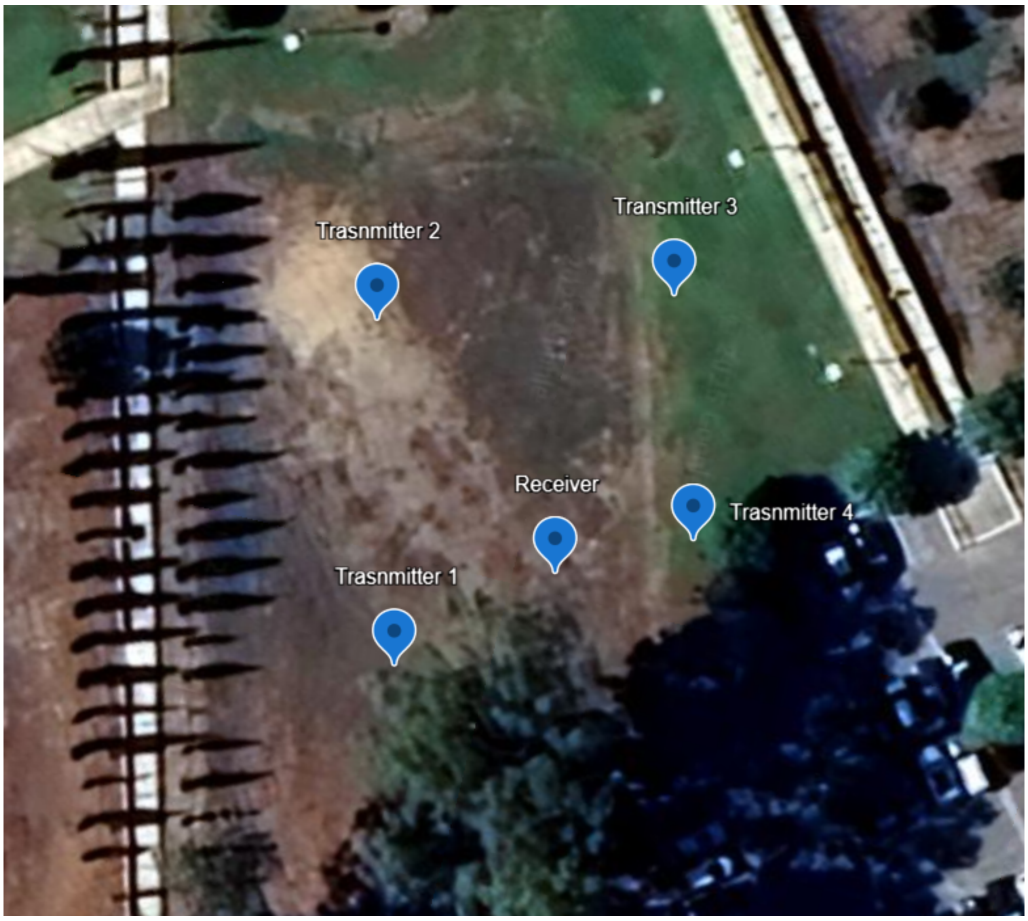
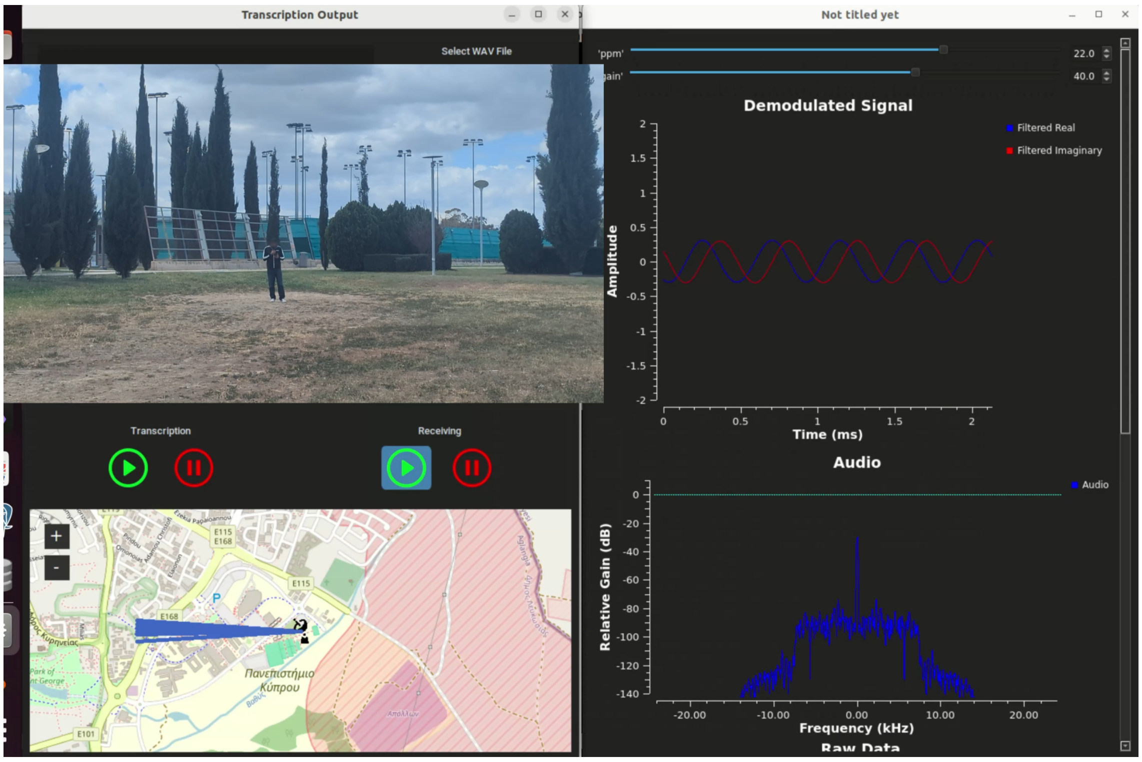


Figure 5.6: DoA experiment results.

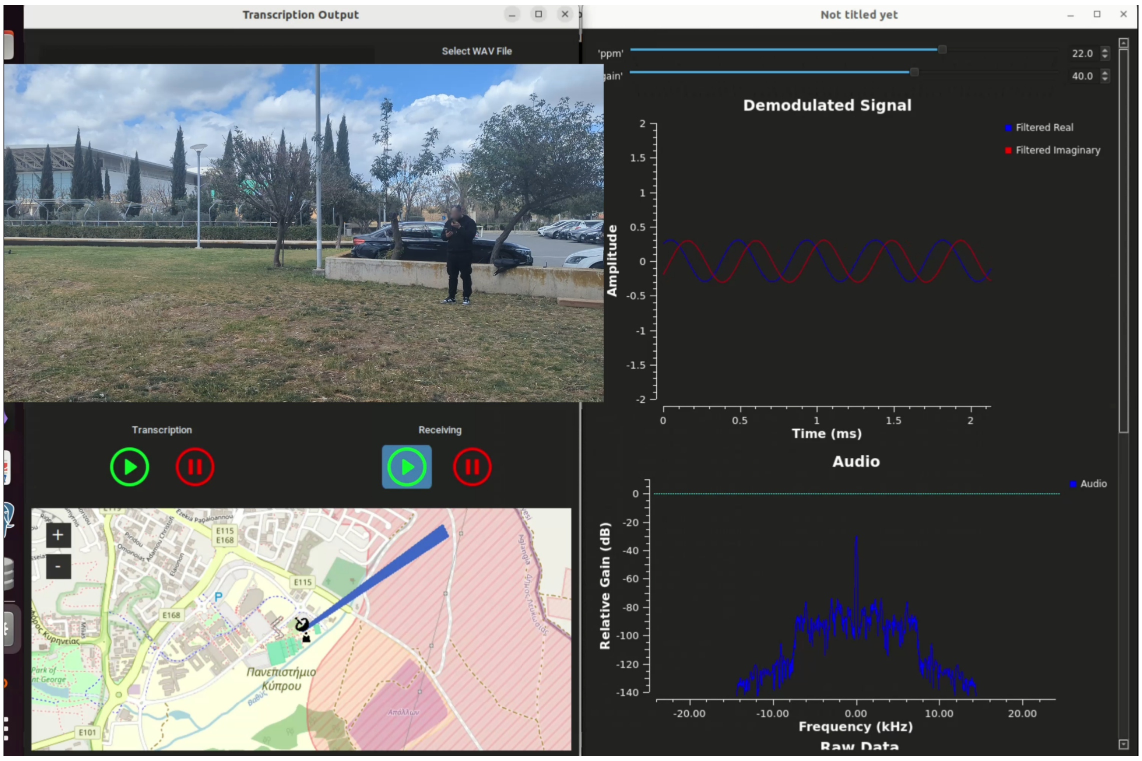
For each transmitter, a specific phrase was chosen:

1. *"This is Transmitter 1. I am transmitting to report severe flooding in the area. Proceed with caution and await further instructions."*
2. *"This is Transmitter 2. Roads are becoming impassable due to rising water. Emergency response teams, please coordinate accordingly."*
3. *"This is Transmitter 3. Evacuation may be necessary. All units, stand by for updates and assistance requests."*

4. *”This is Transmitter 4. Emergency conditions are worsening. Stay alert, follow protocol, and ensure public safety.”*



(a) Transmitter 2



(b) Transmitter 4

Figure 5.7: Front-end platform results for transmitter 2 and transmitter 4

Figure 5.7 shows the configurations of Transmitter 2 and Transmitter 4, alongside their respective DoAs and accompanying audio signals.

```
-> [2025-04-17 11:44:44 - 2025-04-17 11:44:50]
-> Speaker: 1
-> This is transmitter 1. I am transmitting to report severe flooding in the area.
-> Avg DOA: 249.25
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
--------------------------------------------------
-> [2025-04-17 11:44:50 - 2025-04-17 11:44:53]
-> Speaker: 1
-> Proceed with caution and await further instructions.
-> Avg DOA: 249.25
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
--------------------------------------------------
-> [2025-04-17 11:44:59 - 2025-04-17 11:45:05]
-> Speaker: 2
-> This is transmitter 2. Roads are becoming impassable due to rising water.
-> Avg DOA: 179.37
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
--------------------------------------------------
-> [2025-04-17 11:45:05 - 2025-04-17 11:45:09]
-> Speaker: 2
-> Emergency response teams, please coordinate accordingly.
-> Avg DOA: 179.37
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
--------------------------------------------------
-> [2025-04-17 11:45:18 - 2025-04-17 11:45:26]
-> Speaker: 3
-> This is transmitter 3. Evacuation may be necessary. All units stand by for update and assistance requests.
-> Avg DOA: 99.11
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
--------------------------------------------------
-> [2025-04-17 11:45:35 - 2025-04-17 11:45:41]
-> Speaker: 4
-> Emergency conditions are worsening. Stay alert, follow protocol and ensure public safety.
-> Avg DOA: 29.67
-> Band: 446.009 MHz
Wav File: /app/gnuradio/wavFiles/FinalExperiment.wav
```

Figure 5.8: Real-Life experiment results.

Figure 5.8 illustrates that the system effectively transcribed the audio from the walkie-talkies and accurately assigned speaker identities in this scenario.

5.5 Ethical Considerations

This research explores the technical capabilities of Software-Defined Radio (SDR) for analyzing radio frequency signals. All activities were conducted in compliance with applicable legal and regulatory frameworks. This work aims to contribute to the understanding of RF technologies, and it is intended for informational and educational purposes only. We advocate for the responsible and ethical use of these technologies, emphasizing respect for privacy and the legal conditions governing use.

Chapter 6 Conclusion and Future work

6.1 Conclusion

This proof-of-concept study successfully demonstrated a system capable of detecting and analyzing walkie-talkie communications using software-defined radio (SDR) and advanced AI tools. By identifying active VHF/UHF channels with an RTL-SDR, capturing audio transmissions, and converting them to text using OpenAI's Whisper model, we established an effective pipeline for real-time audio processing. Additionally, by integrating direction-finding capabilities via the KrakenSDR, the system could approximate the direction of the transmission, enabling speaker identification based on signal direction. This end-to-end approach presents a practical solution for monitoring and interpreting short-range radio communications, with potential applications in emergency response, situational awareness, and security monitoring. The findings from the signal capture analysis experiment, as detailed in Section 5.1, indicate accuracy in transcription when distances are maintained below 500 meters. This suggests effective signal integrity and processing capability within this range. Furthermore, the Direction of Arrival (DoA) Capture analysis presented in Section 5.3 demonstrates reliability to ascertain the orientation of incoming signals.

Additionally, the real-life experiment outlined in Section 5.4 further supports these results, revealing that the system not only captures and transcribes audio effectively but also assigns speakers accurately based on their respective transmitter's signal direction. Such findings collectively highlight the system's potential for real-world applications in emergency management, public safety and coordination across communication systems.

6.2 Future Work

While this study provides an extensive research on transcribing Walkie-Talkie signals as well as identifying a speaker based on the DoA of the incoming signal there are still areas that warrant further exploration.

A higher quality Software Defined Radio (SDR), such as the HackRF, could potentially enhance both the range and overall quality of the captured signals. By offering improved sensitivity and advanced processing capabilities, high-quality SDRs can detect weaker signals that lower-grade equipment might miss.

A fine-tuned model trained on specific phrases from walkie-talkies could potentially help the model transcribe text with better results since it will learn to understand voice from walkie-talkie.

A further improvement is a localization system using a dual-receiver setup with two KrakenSDR units. By placing these receivers at known locations, it will be possible to capture the direction-of-arrival (DoA) values of incoming signals simultaneously. These measurements can then be processed using triangulation algorithms to estimate the position of the transmitting source. This approach would yield better results compared to the single-receiver and clustering method, that shows only who is speaking. Future research will focus on the synchronization between the KrakenSDR devices, refining the signal processing techniques to reliably extract DoA values under varying environmental conditions, and validating the method through both simulated and real-world experiments.

Another idea is to get the entire system airborne, on a drone. This will help capture the signal much more efficiently since there will be fewer obstacles, allowing the RTL-SDR to capture the signal at greater distances. The system could also be easily moved to another location where attention is needed.

BIBLIOGRAPHY

[1] G. Baldini, T. Sturman, A. Dalode, A. Kropp, and C. Sacchi, "An emergency communication system based on software-defined radio," *EURASIP Journal on Wireless Communications and Networking*, vol. 2014, pp. 1–16, 2014.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] F. Zubir, M. K. A. Rahim, T. Masri, and M. N. Karim, "Comparison between circular array and linear array microstrip antenna," in *2008 IEEE International RF and Microwave Conference*, 2008, pp. 422–426.

[4] P. J. Chung, M. Viberg, and J. Yu, "Chapter 14 - DOA Estimation Methods and Algorithms," in *Academic Press Library in Signal Processing: Volume 3*, A. M. Zoubir, M. Viberg, R. Chellappa, and S. Theodoridis, Eds. Elsevier, 2014, vol. 3, pp. 599–650.

[5] L. Trapani, F. Taylor, E. Gattis, Y. Chen, S. Lo, D. Akos *et al.*, "Testing a coherent software defined radio platform for detection of angle of arrival of rf signals," in *Proceedings of the 36th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2023)*, 2023, pp. 3829–3836.

[6] I. C. Gormley, T. B. Murphy, and A. E. Raftery, "Model-based clustering," *Annual Review of Statistics and Its Application*, vol. 10, no. 1, pp. 573–595, 2023.

[7] P. D. McNicholas, "Model-based clustering," *Journal of Classification*, vol. 33, pp. 331–373, 2016.

[8] R. W. Stewart, L. H. Crockett, D. S. W. Atkinsons, K. W. Barlee, D. H. Crawford, I. G. Chalmers, and E. Sozer, "A low-cost desktop software defined radio design environment using MATLAB, Simulink, and the RTL-SDR," *IEEE Communications Magazine*, vol. 53, no. 9, pp. 64–71, 2015.

[9] "Gnu radio project." [Online]. Available: https://www.gnuradio.org/

[10] A. Radhi, H. Akkar, and H. Abdullah, "Sdr-based intelligent cooperative spectrum sensing for cognitive radio systems," *Engineering and Technology Journal*, vol. 41, no. 2, pp. 1–11, 2023.

[11] C. Graham and N. Roll, “Evaluating openai’s whisper asr: performance analysis across diverse accents and speaker traits,” *JASA Express Letters*, vol. 4, no. 2, 2024.

[12] M. Na and M. Chung, “Optimizing vocabulary modeling for dysarthric speech recognition,” in *Computers Helping People with Special Needs: 15th International Conference*, 2016, pp. 507–510.

[13] X. Zhang and S. Zhou, “Woa-dbscan: application of whale optimization algorithm in dbscan parameter adaption,” *IEEE Access*, vol. 11, pp. 91 861–91 878, 2023.

[14] Y. Zhang, X. Yuan, M. Li, G. Zhao, and H. Wang, “Multi-density adaptive trajectory clustering algorithm for ships based on ais data,” *IEEE Access*, vol. 11, pp. 108 198–108 210, 2023.

[15] D. Deng, “Dbscan clustering algorithm based on density,” in *7th International Forum on Electrical Engineering and Automation (IFEEA)*, 2020, pp. 949–953.

[16] S. Akbar and M. Khan, “Critical analysis of density-based spatial clustering of applications with noise (dbscan) techniques,” *International Journal of Database Theory and Application*, vol. 7, no. 5, pp. 17–28, 2014.

[17] M. Ezuma, F. Erden, C. Anjinappa, O. Özdemir, and I. Güvenç, “Detection and classification of uavs using rf fingerprints in the presence of wi-fi and bluetooth interference,” *IEEE Open Journal of the Communications Society*, vol. 1, pp. 60–76, 2020.

[18] I. Docker, “Docker: Accelerated container application development,” 2013. [Online]. Available: https://www.docker.com/

[19] E. O. Brigham and R. E. Morrow, “The fast fourier transform,” *IEEE Spectrum*, 1967.

[20] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier transform-algorithms and applications*. Springer Science & Business Media, 2011.

[21] M. Mazzei, *Signal (electrical engineering)*. Research Starters, 2023.

[22] I. D. Mienye, T. G. Swart, and G. Obaido, “Recurrent Neural Networks: A Comprehensive Review of Architectures, Variants, and Applications,” *Information*, vol. 15, no. 9, p. 517, 2024.

[23] M. V. Vildyaeva, E. A. Egorova, and A. B. Vavrenyuk, “Using a neural network to convert a radio signal from an rtl-sdr receiver to text,” in *2020 IEEE Conference of*

Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2020, pp. 1449–1451.

[24] G. Klein and O. Contributors, "Ctranslate2: Fast inference engine for opennmt models," gitHub repository. [Online]. Available: https://github.com/OpenNMT/CTranslate2

[25] OpenAI, "Openai: Advancing digital intelligence for the benefit of humanity." [Online]. Available: https://openai.com/

[26] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv:1904.05862 [cs.CL]*, 2019.

[27] G. Gerganov, "Whisper.cpp: High-performance whisper implementation in c++," 2024. [Online]. Available: https://github.com/ggerganov/whisper.cpp

[28] SYSTRAN, "Faster-whisper: Efficient whisper model implementation," 2024. [Online]. Available: https://github.com/SYSTRAN/faster-whisper

[29] Jitsi, "Jiwer: Similarity measures for automatic speech recognition evaluation," gitHub repository. [Online]. Available: https://github.com/jitsi/jiwer

[30] P. Demonte, "Speech corpus - harvard - raw audio." [Online]. Available: https://salford.figshare.com/articles/media/Speech_corpus_-_Harvard_-_raw_audio/7862666

[31] A. M. Wyglinski, R. Getz, T. Collins, and D. Pu, *Software-defined radio for engineers*. Artech House, 2018.

APPENDICES

APPENDIX I

Title of Appendix