DE-42 (EE) Ali, Usama, Noor,

Specific Emitter Identification

COLLEGE OF
ELECTRICAL AND MECHANICAL ENGINEERING
NATIONAL UNIVERSITY OF SCIENCES AND TECHNOLOGY
RAWALPINDI
2024

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DE-42 EE
PROJECT REPORT

Specific Emitter Identification

Submitted to the Department of Electrical Engineering
in partial fulfillment of the requirements
for the degree of
Bachelor of Engineering
in
Electrical
2024

Submitted by:
Muhammad Ali Bin Mushtaq
Muhammad Usama Mustafa
Noor-ul-Ain

CERTIFICATE OF APPROVAL

It is to certify that the project **“Specific Emitter Identification”** was done by **NS Muhammad Usama Mustafa**, **NS Muhammad Ali Bin Mushtaq**, **NS Noor-ul-Ain** under the supervision of **Dr. Qasim Umar Khan**.

Submission: This project was submitted to the College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan, as part of the requirement for the degree of Bachelor of Electrical Engineering.

Students:

1. Muhammad Usama Mustafa

NUST ID: 333186 Signature: M Usama

2. Muhammad Ali Bin Mushtaq

NUST ID: 335701 Signature: m.Ali

3. Noor-ul-Ain

NUST ID: 342465 Signature: [illegible]

Approved By:

Project Supervisor: **Dr. Qasim Umar Khan**

Signature: ______________________

Date: __________________________

DECLARATION

We declare that no part of this project thesis has been submitted in support of an application for another degree or qualification. We have not submitted this thesis to any other university or educational institution. We are totally liable for any disciplinary action taken against us based on the nature of the proved offence, including the revocation of our degree.

Students:

1. **Muhammad Usama Mustafa**

NUST ID: 333186 Signature: M Usama

2. **Muhammad Ali Bin Mushtaq**

Signature: m.Ali

NUST ID: 335701

3. **Noor-ul-Ain**

Signature: [illegible]

NUST ID: 342465

COPYRIGHT STATEMENT

Students:

1. Muhammad Usama Mustafa

NUST ID: 333186

Signature: M Usama

2. Muhammad Ali Bin Mushtaq

NUST ID: 335701

Signature: m.Ali

3. Noor-ul-Ain

NUST ID: 342465

Signature: [illegible]

ACKNOWLEDGMENTS

First, we would like to thank Allah Almighty for giving us this amazing opportunity to work at this final year project and giving us strength to complete it. Secondly, we want to thank our supervisor Dr. Qasim Umar Khan for their unwavering support, guidance and help that enabled us to achieve major milestones throughout the project. Without his insightful inputs and humble corrections we would not have been able to complete this project.

ABSTRACT

Specific Emitter Identification (SEI) refers to the identification of transmitters based on their unique characteristics, known as RF fingerprints. SEI can be performed using two main methods: manual feature-based and deep learning-based approaches. In this project, we are developing a novel SEI algorithm that uniquely combines both approaches.

First, the signal is acquired using a software-defined radio. The received signal is then passed through the SEI algorithm, which extracts RF fingerprints from it. The signal is decomposed into five modes using Variational Mode Decomposition (VMD). From each mode, two types of features are extracted: RF-DNA and time-frequency spectrograms. We refer to this combination of RF-DNA with VMD as modified RF-DNA, and it serves as the input to the XGBoost classifier, which classifies the signal among known classes.

If an unknown transmitter is detected, it is sent to the Siamese Neural Network (SNN). The SNN uses both modified RF-DNA and time-frequency spectrograms to convert the inputs into a shared representation space using a 2-channel Convolutional Neural Network (CNN) with bimodal feature fusion. The similarity scores are then calculated by comparing the input with other signals. If a match is found, the transmitter is labeled accordingly; otherwise, it is assigned a new label as an unknown transmitter.

<u>SUSTAINABLE DEVELOPMENT GOALS</u>

Goal 9 – INDUSTRY, INNOVATION, AND INFRASTRUCTURE:

Our project, operating on a portable system, serves to detect potential threats in advance, enabling timely responses to safeguard national security. By offering early indications of emerging risks, it empowers the implementation of effective countermeasures, contributing to the resilience of critical infrastructure and fostering innovation in security protocols.

Figure 1. SGD Goal 9

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Letter like symbols

$\mathcal{H}$	Hilbert transform
s	skewness
k	kurtosis

Greek Letters

α	penalty factor in VMD and bias in triplet loss
ω	Frequency $(\mathrm{rad/s})$
λ	Lagrangian operator
σ	Standard deviation

Acronyms

SEI	Specific Emitter Identification
RFF	Radio Frequency Fingerprint
USRP	Universal Software Radio Peripheral
NI	National Instruments
SVM	Support vector machine
IMF	Intrinsic mode functions
HHT	Hilbert Huang transform
EMD	Empirical mode decomposition
CEEMDAN	Complementary ensemble empirical mode decomposition with adaptive noise
VMD	Variational mode decomposition
UWB	Ultra-wideband
CNN	Convolutional neural network
STFT	Short time Fourier transform
AM-FM	Amplitude-modulated-frequency-modulated
RF-DNA	Radio frequency distinct native attribute
GBDT	Gradient boosting decision tree
XGBoost	Extreme gradient boost
SNR	Signal to noise ratio

Chapter 1 – INTRODUCTION

In the modern world, wireless systems have become an indispensable enabler of modern military operations and civilian activities, and serve as the backbone of communication, surveillance, navigation, and numerous other critical functions. And with the introduction of 5G and upcoming 6G networks, and the increasing number of devices for Internet-of-things (IOT), there is an overwhelming increase in the device density. Because of the massive device density and diversity with respect to the applications and types of devices, security and privacy have become a very important issue that needs to be addressed. Moreover, in the modern warfare and defense, the ability to accurately identify the source of electromagnetic emissions have become a very important task that needs to be done and these objectives cannot be achieved with current wireless security methods, so there is a need for an extraordinary algorithm that can not only provide security for wireless devices but also help monitor the electromagnetic spectrum for a possible attack and Specific Emitter Identification or SEI is the very algorithm that solves the device identification and authorization problem and provides an opportunity for spectrum supervision for military applications.

1.1. Specific Emitter Identification:

Specific Emitter Identification (SEI) involves identifying transmitters based on unique characteristics known as RF Fingerprints. These fingerprints are inherent to the hardware of transmitting devices, making them unique even among transmitters of the same model. RF Fingerprints are consistent and cannot be altered, controlled, or replicated, and they are independent of location and signal type, relying solely on the hardware characteristics. This means that regardless of the transmitted signal, detecting the RF Fingerprint allows for the identification of the transmitting device.

SEI methods fall into two main categories: manual feature-based methods and deep learning-based methods. Manual feature-based methods involve extracting predefined features from the signal and training models to classify transmitters based on these features. In contrast, deep learning-based methods use the signal's IQ data or time-frequency features as input to deep learning models, which learn hidden features for classification.

Traditionally, SEI involves collecting large amounts of data and training numerous machine learning and deep learning models to classify transmitters. However, with the increasing number of devices and scenarios such as fast-paced and highly mobile military operations, there is a need to identify transmitters using minimal data in the shortest possible time. This necessitates a one-shot prediction algorithm capable of identifying transmitters from just a single signal snapshot.

1.1. Scope of the project:

The project Specific Emitter Identification focuses on the following areas:

- The development of a novel RF fingerprint extraction approach using advanced signal processing algorithms.
- Training of a classification algorithm for classification of known transmitters and development of trained model on a real-time system for real-time prediction.
- Development of a one-shot deep learning algorithm for the prediction and labelling of new transmitters as unknown transmitters in real-time.
- Development of novel algorithm that uniquely combines both manual feature-based and deep learning-based algorithms and provides the best results.
- Design and development of an interactive graphical user interface with user-friendly design and interactive display for predictions.

1.2. Report structure:

The rest of the report is structured as follows. Chapter 2 focuses on the background of specific emitter identification shows some of the papers that we have reviewed for this project. Chapter 3 discusses the data collection procedure we adopted to train our models with different signals and the structure of our test devices. Chapter 4 goes into detail regarding our feature extraction algorithm and discusses many signal processing concepts and our novel approach. Chapter 5 and Chapter 6 will focus on the machine learning and deep learning algorithms and their training results where Chapter 7 will show the Graphical user interface and the hardware and software for real-time SEI application with two different receivers and in Chapter 8 we will share the results of our experimentations by testing our models with several parameters including message signal change, carrier frequency change, and SNR.

Chapter 2 – BACKGROUND AND LITERATURE REVIEW

Now we will dive deeper into the previous algorithms and discuss the problems with each specific emitter identification method leading to our novel approach and reason for selection of this approach.

2.1. Background:

Although SEI can be done on both steady state signals and transient signals, there are some difficulties in the transient signals because of their short amount of time and difficultly to control in unknown state [1] [2]. On the other hand, steady state signal detection is relatively easier, and these signals contain more information about RFF. So, it is now very important to identify how we can extract RFF from steady state signals.

2.1.1. Manual Feature-based SEI:

There have been many papers that discuss manual-feature based or traditional way of identifying transmitters. Some of the papers that discuss manual features that can be used for SEI are discussed as follows:

2.1.1.1. RF Fingerprint based on Entropy:

This paper [3] discusses an RF Fingerprint extraction method based on CEEMDAN (Complementary ensemble empirical mode decomposition with adaptive noise) and multidomain joint entropy. First, they decomposed the received signal into multiple Intrinsic Mode Functions (IMF), then represented the modes in multiple spaces by multidimensional phase space reconstruction technique. This way they did the non-linear analysis of original signal in multiple spaces: multidimensional differential approximate entropy space, singular spectral entropy space, and power spectral entropy space. The used Support Vector Machine (SVM) as classification model and achieved a 98.5% accuracy on 5-class USRP dataset and a 94.7% accuracy on 16-class public dataset. They achieved a 85% recognition rate in SNR above 5dB.

2.1.1.2. RF Fingerprint based on spectral and statistical features:

The paper [4] employs Variational Mode Decomposition (VMD) to break down the signal into different Intrinsic Mode Functions (IMFs). It then extracts entropy, statistical, and spectral features from these IMFs, comparing the training results based on the probability of correct classification across various scenarios and modulations. Additionally, the study compares VMD with spectral features against Empirical Mode Decomposition (EMD) with spectral features, concluding that VMD-SF outperforms other feature sets in all scenarios. The paper also demonstrates that VMD is more effective at signal decomposition than EMD for Specific Emitter Identification (SEI).

2.1.1.3. RF-DNA based features:

In [5], the authors introduced a new method for extracting RF Fingerprints (RFF), termed RF Distinct Native Attribute (RF-DNA), for ultra-wideband (UWB) noise radar identification. They calculated various fingerprint features, including variance, skewness, and kurtosis of the signals, and classified them using a Maximum Likelihood classifier. This approach achieved an accuracy of 99.7% for a three-class dataset and 81% for a seven-class dataset. Ultimately, by combining time and frequency domain features, they achieved an accuracy of 97.65%.

2.1.2. Deep learning-based SEI:

Now that we have looked at some of the papers that discussed the manual feature-based approach, we will look at some of the deep learning-based approaches.

2.1.2.1. Single domain prediction:

In the earlier stages of SEI, many papers worked on using only single domain features to predict the transmitters. The paper [6] utilized the symmetrical characteristics of bi-spectrum structure to extract features then used a multi-channel one-dimensional convolutional neural network (CNN) for classification. Although this method performed feature extraction in depth and reduced the loss of RFF, it had high computation complexity because of complex features. Moreover, as the research about SEI progressed, researchers found that single-domain features could not properly explain RF Fingerprint.

2.1.2.2. Time-frequency based features:

As researchers began focusing on time-frequency features for Specific Emitter Identification (SEI) [7] – [10], they employed various time-frequency conversion algorithms such as the Hilbert-Huang Transform, wavelet transform, and Fast Fourier Transform. In [9], a Short-time Fourier Transform (STFT) based SEI algorithm was proposed for signal detection, achieving high performance. However, due to STFT's linear nature and its inability to perform nonlinear analysis, it proved to be a suboptimal approach. Conversely, [10] introduced a method using a wavelet algorithm to extract dynamic wavelet fingerprints, achieving 99% accuracy in recognizing unique RFID tags. The drawback of this approach is the need to select an appropriate basis wavelet function. Among these algorithms, the Hilbert-Huang Transform stands out as an effective method for analyzing nonlinear and non-stationary signals.

2.1.2.3. Empirical Mode Decomposition based approach:

In [7] a Hilbert-Huang transform combined with Empirical Mode Decomposition (EMD) algorithm is proposed. It first decomposes the signal into different modes and then take HHT to obtain time-frequency features. However, a major issue with EMD is the existence of mode mixing and complex calculations [11].

2.2. Literature Review:

For the literature review, we utilized several research papers and learned several algorithms and techniques utilized by researchers for SEI. Some of the core concepts that we have studied are discussed below:

2.2.1. Variational Mode Decomposition (VMD):

Variational Mode Decomposition, proposed in 2014 [11] is a fully intrinsic and adaptive signal processing algorithm used to decompose signals into their principal modes called Intrinsic Mode Functions (IMFs).

2.2.1.1. What is a mode?

A mode, or Intrinsic Mode Function (IMF) is an amplitude-modulated-frequency-modulated (AM-FM) signal, that has an envelope A(t) and an instantaneous phase $\emptyset(t)$. Mathematically, a mode can be written as:

$$u_k(t) = A_k(t)\cos\left(\emptyset_k(t)\right)$$

Where k is the mode number.

2.2.1.2. How does VMD work?

VMD is an iterative method that decomposes a signal based on specific center frequencies ω_k using Euler-Lagrange equations and wiener filtering. It uses Wiener filtering to update the current mode $u_k^n(\omega)$ in frequency domain with respect to current center frequency ω_k^n using the following equation:

$$u_k^{n+1}(\omega) = \frac{\hat{x}(\omega) - \dfrac{\sum_{j \neq k} u_k^n(\omega) + \lambda^n(\omega)}{2}}{1 + 2\alpha(\omega - \omega_k^n)^2}$$

Where n represents the number of iterations. And λ is the Lagrangian operator and α is the penalty factor.
After updating the mode, it updates the center frequency and Lagrangian operator as well until the error is less than the specified threshold.

2.2.1.3. Key features of Variational Mode Decomposition:

1. **Reconstruction of original signal:** When we decompose a signal using Variational Mode Decomposition, we can not only properly decompose the signal but also reconstruct it by simple adding all the modes. Let x(t) be the signal input to VMD and $u_k(t)$ be the signals decomposed by it. We can easily reconstruct x(t) using:

$$x(t)\ reconstructed = \sum_{k=0}^{K} u_k(t)\,, where\ K\ is\ the\ total\ number\ of\ modes.$$

2. **Customizable number of modes:** With VMD, we can customize the number of modes we need. This helps us fixing the number of features for our Machine Learning models and control our data.
3. **Specific bandwidth and center frequency:** The bandwidth of the modes decomposed by VMD have a specific bandwidth and a specific center frequency, hence better decomposition, and better feature extraction for signal processing applications.

Following is a plot that visualizes how VMD works:

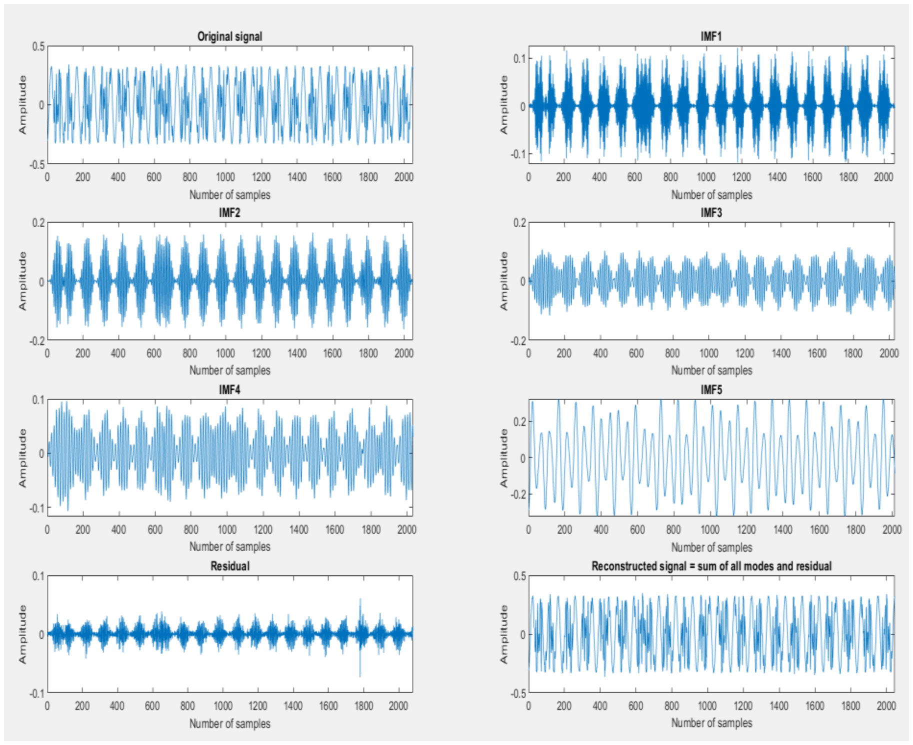


Figure 2. Variational Mode Decomposition of message signal received from dataset.

Figure 2 shows the decomposition of a signal collected from an SDR into 5 different modes. We can see that each mode has a specific center frequency and a specific envelope. Additionally, when we add all the modes and the residual signal, we can easily reconstruct the signal. This helps us to remove noise from the signal.

2.2.2. Unintentional Modulation and RF Fingerprint:

As mentioned earlier, RFF are unique features that are added into the signal transmitted by the transmitting device. When a signal is transmitted from a transmitting device, two types of modulation occur, one is the Intentional Modulation (IM), and the other is the Unintentional Modulation (UIM). In this section, we will discuss what UIM is and how it is related to the RFF as explained by [12].

2.2.2.1. Unintentional Modulation:

When a signal is transmitted, because of the hardware structure differences, the amplitude, frequency, and phase of the signal is disrupted by a process called Unintentional Modulation. Although, the signal is affected by all the hardware components including, filters, amplifiers, oscillators, etc. A typical transmitter usually affects the signal in following two ways:

- Change in phase/frequency of transmitter signal because of the phase noise of high frequency oscillators.
- Change in the envelope of the transmitted signal because of the non-linearities of the power amplifier.

So, UIM affects the envelope, phase, and frequency of the transmitted signal which are the RF Fingerprints and now we will look at how we can extract this RFF.

2.2.2.2. Effect of IM on probability distribution:

When the envelope of a signal is affected, the distribution of the signal is also changed. As shown in the following figure:

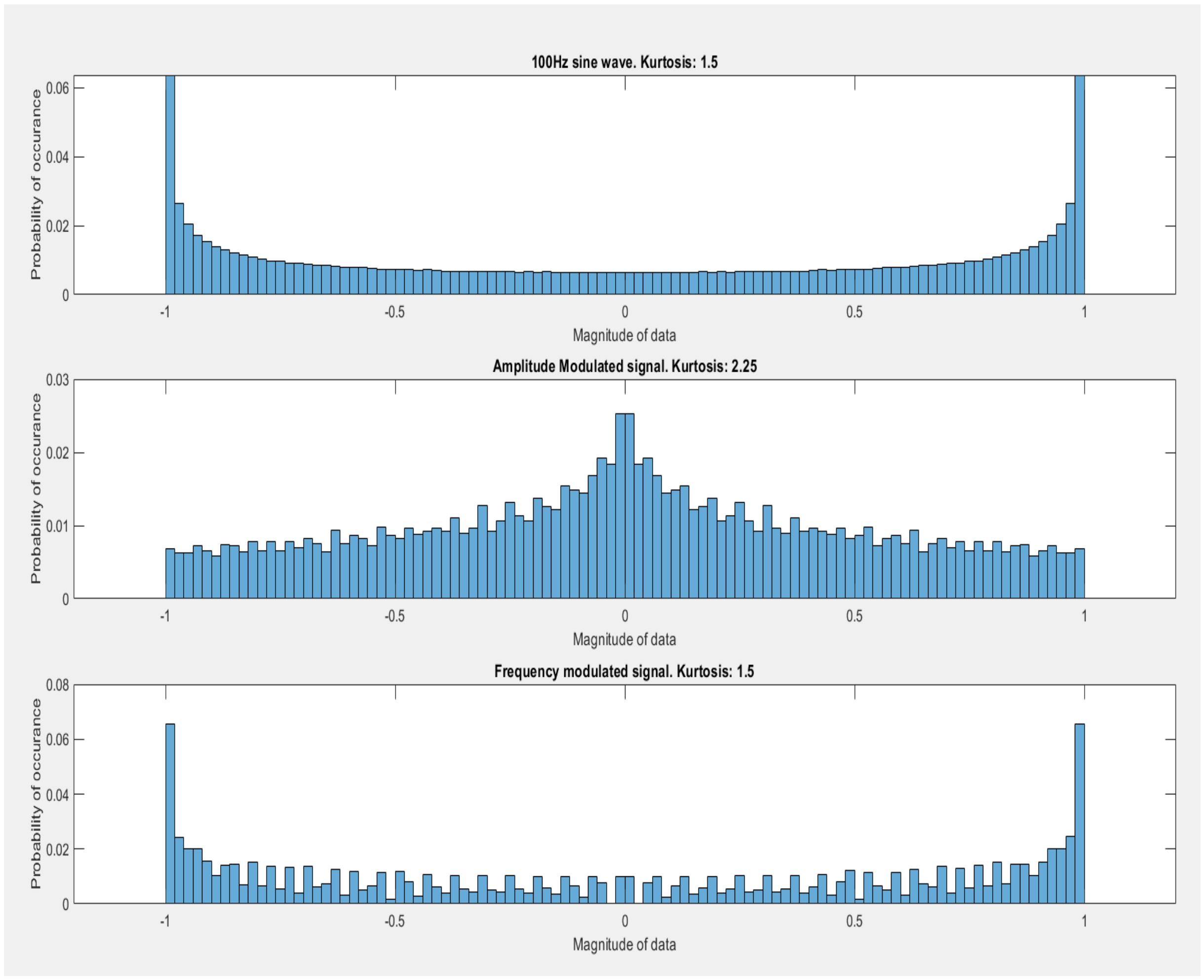


Figure 3. The probability distribution of a single tone before and after amplitude modulation and frequency modulation.

We can see that as the envelope of the signal changes, its distribution and kurtosis changes completely and when the frequency of the signal is affected, it changes the distribution but does not affect the kurtosis.
So, if UIM changes the envelope, phase and frequency of the signal, by focusing on the probability distribution, we can extract RFF from the signal.

2.2.3. RF-DNA and XGBoost for specific emitter identification:

The paper [13] utilizes the RF-DNA feature proposed by Michael Temple and combines it with XGBoost classifier for specific emitter identification and achieves 96.40% accuracy. In this section, we will discuss the RF-DNA feature and XGBoost classifier.

2.2.3.1. RF-DNA feature:

RF-DNA feature is a combination of statistical features of the amplitude, phase, and frequency of the signal showing the irregularities and non-linearities of each aspect of the signal, hence giving an in-depth description of RFF in the signal.
1. For RF-DNA, the signal received is converted to an analytical signal using Hilbert

Transform, using the following equation:

$$x_h(t) = x(t) + \mathcal{H}\big(x(t)\big) * 1j$$

Where x(t) is the received signal. Analytical signal is very helpful as it provides the information about the signal in two domains, we can get the time-frequency information of the signal using this technique.

2. Once the signal is converted into analytical signal, the amplitude, phase, and frequency of each sample is calculated using the absolute, angle, and differential of angle, respectively, of each sample which is a complex number.

3. In this step, the variance, skewness, and kurtosis of the amplitude, phase, and frequency vectors of the analytical signal is calculated.

Mathematically, we can write it as:

$$RF - DNA = [F_A, F_\emptyset, F_f\ ],$$

$where, F = [\sigma^2, s, k]$ and s is the skewness, and k is the kurtosis.

So, by combining the variance, skewness, and kurtosis, we get the complete information about the changes in the probability distribution of the signal affect by UIM (section 2.1.2) and this method helps us extract RFF completely.

2.2.3.2. XGBoost Classifier:

Extreme Gradient Boost (XGBoost) Classifier is an ensemble learning method which is based on the idea of gradient boosting. Although, it works on the similar method, it is completely different from gradient boosting decision tree (GBDT). The term boosting means that by combining different weak models, we can enhance a single weak model and GBDT works on iteratively training an ensemble of decision trees and tries to use the previous model's error residual to train the next decision tree model. But because of its sequential approach, it takes a lot of time to train and has a high tendency to move towards overfitting, but these issues are resolved in XGBoost, which converts its sequential approach to parallel approach and follows a level-wise strategy scanning across gradient values and uses these partial sums to check the quality of splits at every split.
XGBoost has many advantages and is considered the best classification algorithm because of its performance, speed because of regularization, adaptive processing of missing values, pruning and parallel processing.

2.2.4. Bimodal feature fusion for specific emitter identification:

The paper [14] proposes a novel idea of using VMD for specific emitter identification and a unique way to combining both manual features and deep learning to train a CNN classifier. As mentioned in the introduction, there are two main ways to perform specific emitter identification, one is manual feature-based approach and the other is deep learning-based approach but in this paper, the researchers developed a two channel convolutional neural network that took both features as input and later on combined both features and achieved more than 90% for SNR as low as -10dB showcasing the noise resistance of this algorithm.
We will discuss their neural network later in the deep learning chapter as it is the base model for our deep learning model.

CHAPTER 3 – HARDWARE AND DATA COLLECTION

This chapter emphasizes on the project hardware, creation of a personalized dataset, and the incorporation of different parametric nuances (changes in gain, signal's frequency, and devices' location, etc.) to allow for our developed algorithm's increased robustness.

3.1. Project Hardware:

1. USRP-2901: It is a Software-defined radio (Manufactured by National Instruments (NI)) that allows for transmission and reception over a wide range of radio frequencies. The device also includes a high-speed interface (typically Gigabit Ethernet) for data transfer to and from a host computer. For data acquisition, we used these SDR devices that can be configured inside LabView (a graphical programming environment, provided by NI Instruments) and synthesized a broad range of signals, differing in their message contents, as well as, modulation techniques. Finally, all this data was saved onto a computer and taken further for processing.

2. High Performance Laptop: Initially, the signal processing of the RF signals acquired from our USRP devices required a highly efficient processor capable of performing the signal decomposition algorithm discussed in Chapter 3 (i.e., VMD). Furthermore, extracting features from the modes provided by VMD also demanded substantial computational resources. Finally, applying Machine Learning and Deep Learning algorithms required a powerful system to optimize models for improved classification. The spec of which are as follows:

   - CPU: Intel(R) Core (TM) i5-10500H CPU @ 2.50GHz 2.50 GHz
   - GPU: NVIDIA GTX 1650Ti (4 GB).
   - RAM: 16GB
   - Operating system: Windows 11

3.2. Data Collection:

In this section, we will first discuss how we utilized LabVIEW to configure our devices for transmitting and receiving data. Additionally, we will briefly cover the number of classes defined in our dataset. Finally, we will explore the various types of signals we generated to enhance data diversity to optimize models' performance across different modalities.

3.2.1 Design for the LabView Reception file:

We set up a USRP Device as a receiver by using LabView's built-in file named as, "Rx Continuous Async", and configured it as follows:

1. Carrier Frequency: 2.45GHz
2. IQ rate: 100K
3. No of samples: 10,000
4. Gain: 100
5. Time of data collection: 50s

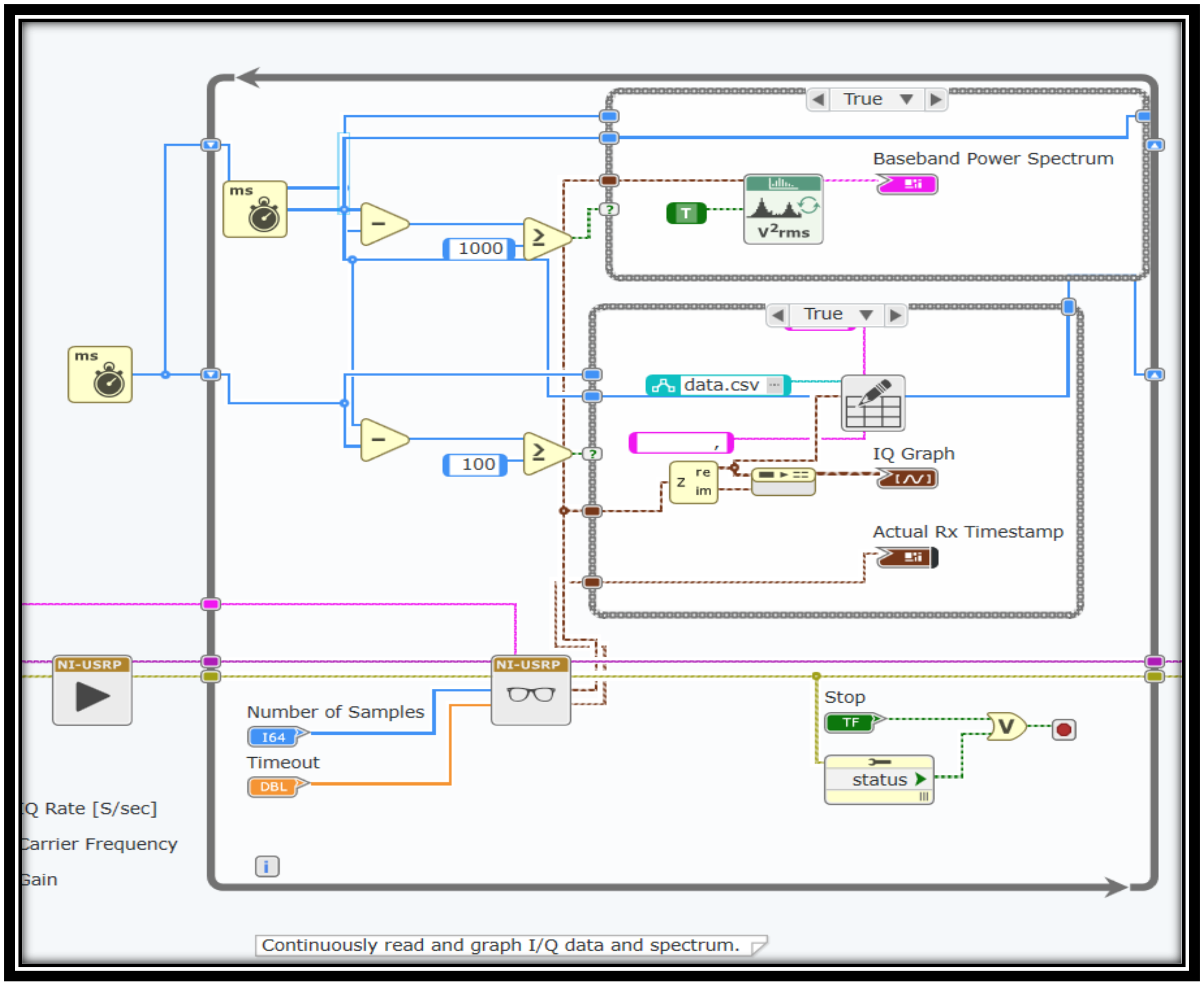


Figure 4. The Reception And Storage File At Receiver

3.2.2. Dataset:

For our project, we defined a total of seven different classes (i.e., 4 known Transmitters, 2 Unknown Transmitter devices, and noise class). Furthermore, they were labeled as:

(i) Known Transmitter 1 => Labeled as (1)
(ii) Known Transmitter 2 => Labeled as (2)
(iii) Known Transmitter 3 => Labeled as (3)
(iv) Known Transmitter 4 => Labeled as (4)
(v) Unknown Transmitter 1 => Labeled as (-1)
(vi) Unknown Transmitter 1 => Labeled as (-2)
(vii) Noise Class => Labeled as (0)

For data collection we will be using the following files that we created using LabView:

1. Single Tone Frequency as done in Transmission and Reception of Single Tone signal using Rx & Tx Continuous Async.

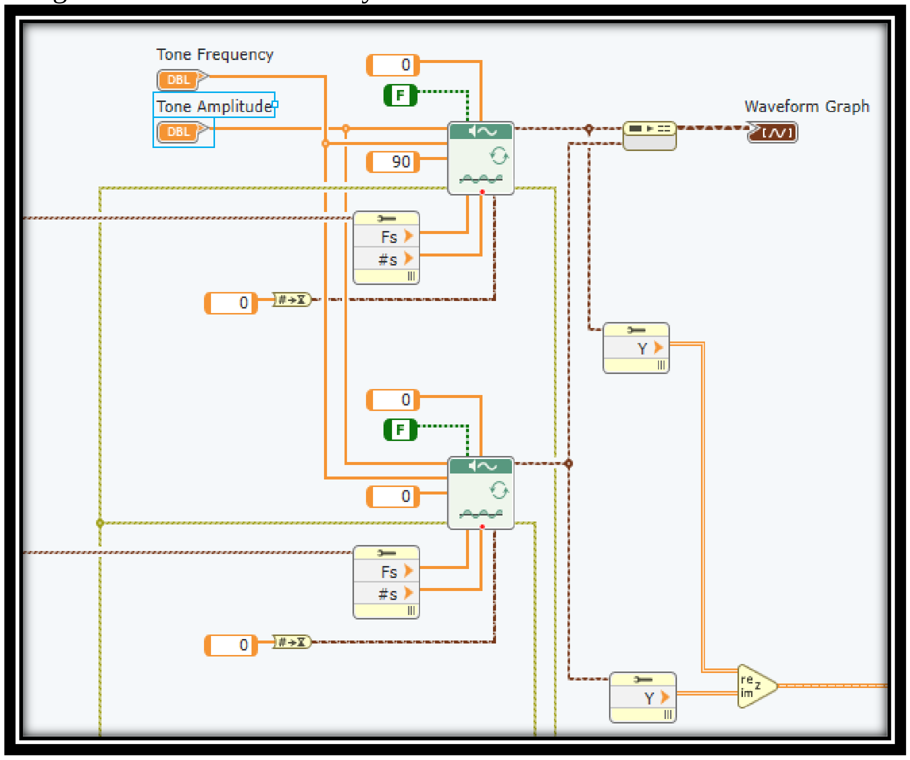


Figure 5. Single Tone Tx File

2. AM Modulated Sine Wave

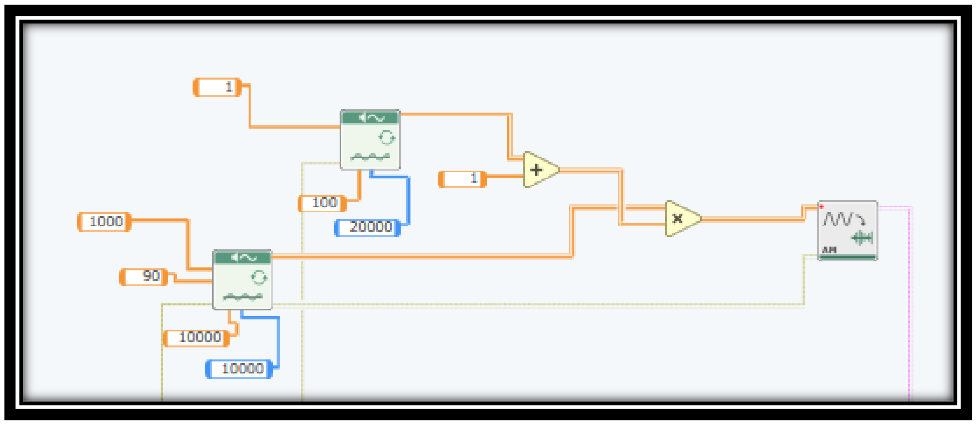


Figure 6. AM Modulated Sine Wave Tx File

3. Simple Audio File Synthesized in MATLAB

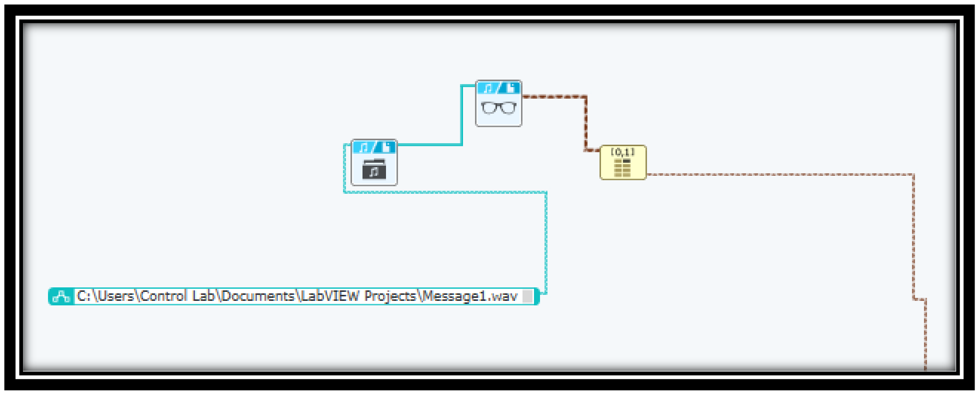

C:\Users\Control Lab\Documents\LabVIEW Projects\Message1.wav
[0,1]

Figure 7. Simple Audio Wave Tx File

4. AM Modulated Audio Signal.

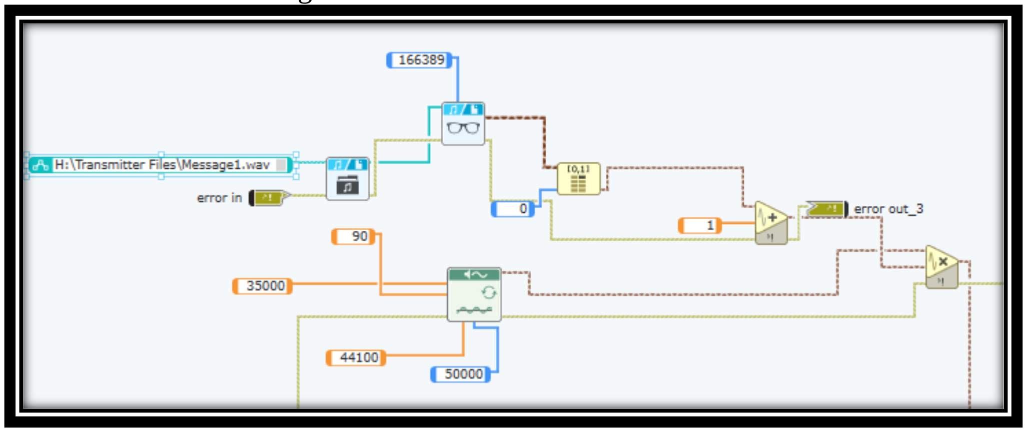


Figure 8. AM Modulated Audio Wave Tx File

3.2.3. DATA VARIATION:

To get variations in the data, we varied the gain of each signal (files, mentioned above) from the lowest possible gain to 100. This will allow us to have data that shows the behavior of features at different gains, hence allowing us to make better identification.
Dealing with noise is an important factor in emitter identification. So, to get more variations of data that has noise, we will use an attenuator at the transmitter side and increase the distance between USRP devices, hence ensuring that most of the data being received is noisy.

CHAPTER 4 – SIGNAL PREPROCESSING AND FEATURE EXTRACTION

As discussed in the literature, our signal preprocessing and feature extraction algorithm must be focused on how we can extract RF fingerprints from received signal with high tolerance to noise. In this chapter, we will discuss our novel specific emitter identification feature extraction and preprocessing algorithms and its key features. As we will be using both manual feature-based approach and deep learning-based approach, there are two features that we will be extracting from received signal. Before we dive deep into the feature extraction, we need to specify our requirements for an amazing SEI application.

4.1. Requirements of RFF extraction algorithm:

As the classification needs to be done solely based on RF fingerprints, we need to eliminate any possibilities including prediction based on carrier signal, message signal, or location of transmitter etc. So, our algorithm must be able to fulfil the following requirements:

- **Message and carrier signal independence:** Our specific emitter identification must be independent of the message signal and the carrier frequency of transmitted signal. It is extremely important to keep the RFF as the only feature for classification which is why it is essential to make sure our feature extraction algorithm is message and carrier signal independent.
- **Bandwidth independence:** Like message and carrier signal, it is important to make sure that the bandwidth of the transmitted signal does not affect the prediction of our model. This is essential because the bandwidth of the transmitter may vary especially in the case of unknown transmitters and in the case of multiple transmitters.
- **Location independence:** It is very important to make sure that our algorithm does not predict based on the location of the transmitter.
- **In-depth and low-cost RFF extraction:** Another important factor for our algorithm is for it to be able to extract RF fingerprint with in-depth details without compromising the computational cost, hence faster feature extraction and faster predictions.

Following the above-mentioned requirements, we have developed two main features for our algorithm.

4.2. Manual feature-based RFF extraction (Modified RF-DNA):

Our novel approach for manual feature-based RFF extraction works on combining the extraordinary signal decomposition technique of VMD and the complete RFF extraction ability of RF-DNA. Here is a step-by-step process of extracting this **modified RF-DNA.**
The received signal is first decomposed into 5 different modes using VMD. As discussed earlier, this allows for better and more in-depth feature extraction. Additionally, by dividing the signal into modes, we can remove the message signal from the signal and only focus on RFF. Amongst these 5 modes, the first two modes are eliminated as the message and carrier signals are mostly dominant in the first two modes and the remaining 3 modes are considered the modes containing the most information about RFF. Once we have the three modes from VMD, we will apply RF-DNA method on each mode and obtain the variance, skewness, and kurtosis of the amplitude, phase, and frequency of the analytical signal of each mode. In addition to these features, we will add the center frequencies of each mode, obtained from the VMD, into the features.
So, the combination of RF-DNA of each mode of the signal obtained by VMD, and the center frequency of each mode is the modified RF-DNA feature extraction method we have used for our Manual feature-based RFF extraction.

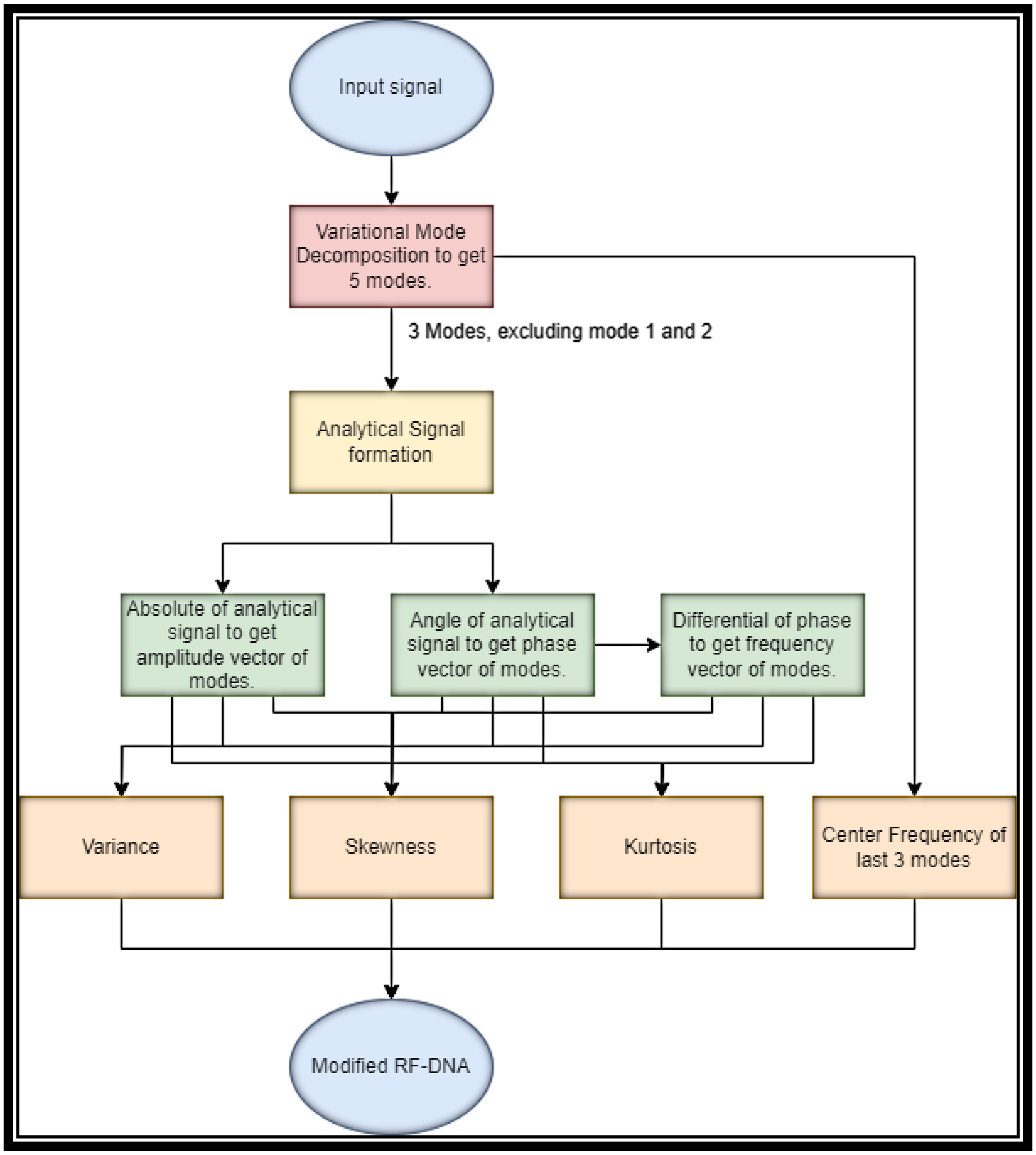

Input signal
Variational Mode Decomposition to get 5 modes.
3 Modes, excluding mode 1 and 2
Analytical Signal formation
Absolute of analytical signal to get amplitude vector of modes.
Angle of analytical signal to get phase vector of modes.
Differential of phase to get frequency vector of modes.
Variance
Skewness
Kurtosis
Center Frequency of last 3 modes
Modified RF-DNA

Figure 9 shows the complete flow chart of the process of extracting modified RF-DNA from input signal.

This modified RF-DNA feature will be used to train a machine learning classifier which is discussed in the next chapter. For now we will move onto the features for deep learning-based SEI.

4.3. Features for deep learning-based SEI:

Like [14] the deep learning model used for this project also utilizes bimodal feature fusion, so for this purpose the features for deep learning-based SEI will be of two types, time frequency spectrogram as spatial features and modified RF-DNA as temporal features.
The temporal features for the bimodal feature fusion will be the modified RF-DNA as discussed in the previous section but for this, our modified RF-DNA will use all the modes from the VMD instead of the final three as the deep learning model will focus on extracting RFF from all the modes.
For the time-frequency spectrogram, we will first take the VMD of the signal, convert each mode to analytical signal, calculate amplitude and frequency of each mode and then we will calculate the spectrogram using the SciPy's spectrogram function which is then plotted using matplotlib's pcolormesh.

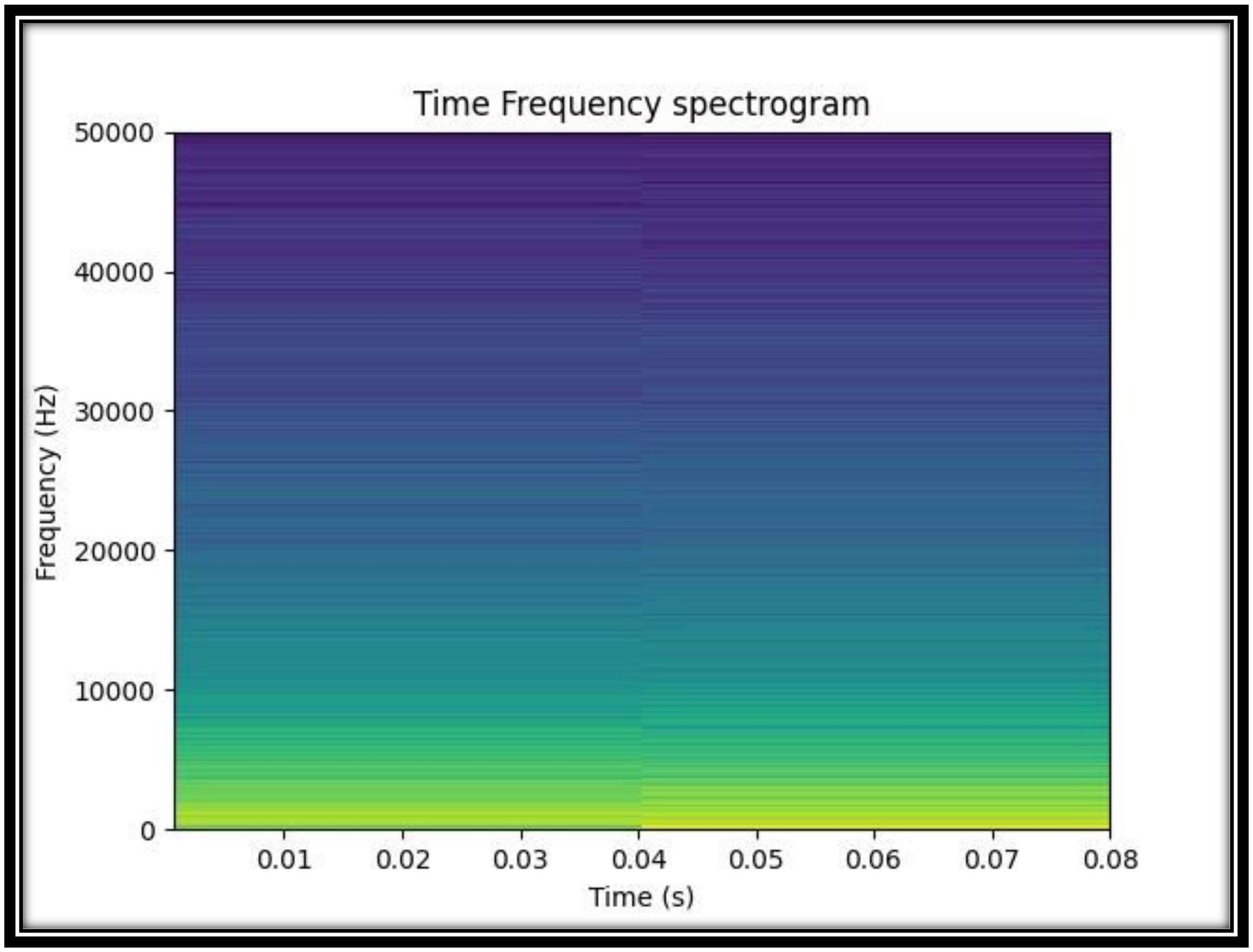


Figure 10. It shows the time frequency spectrogram of a signal received.

So, the modified RF-DNA with all modes and the time frequency spectrogram of all modes are the two features for the bimodal feature fusion algorithm.

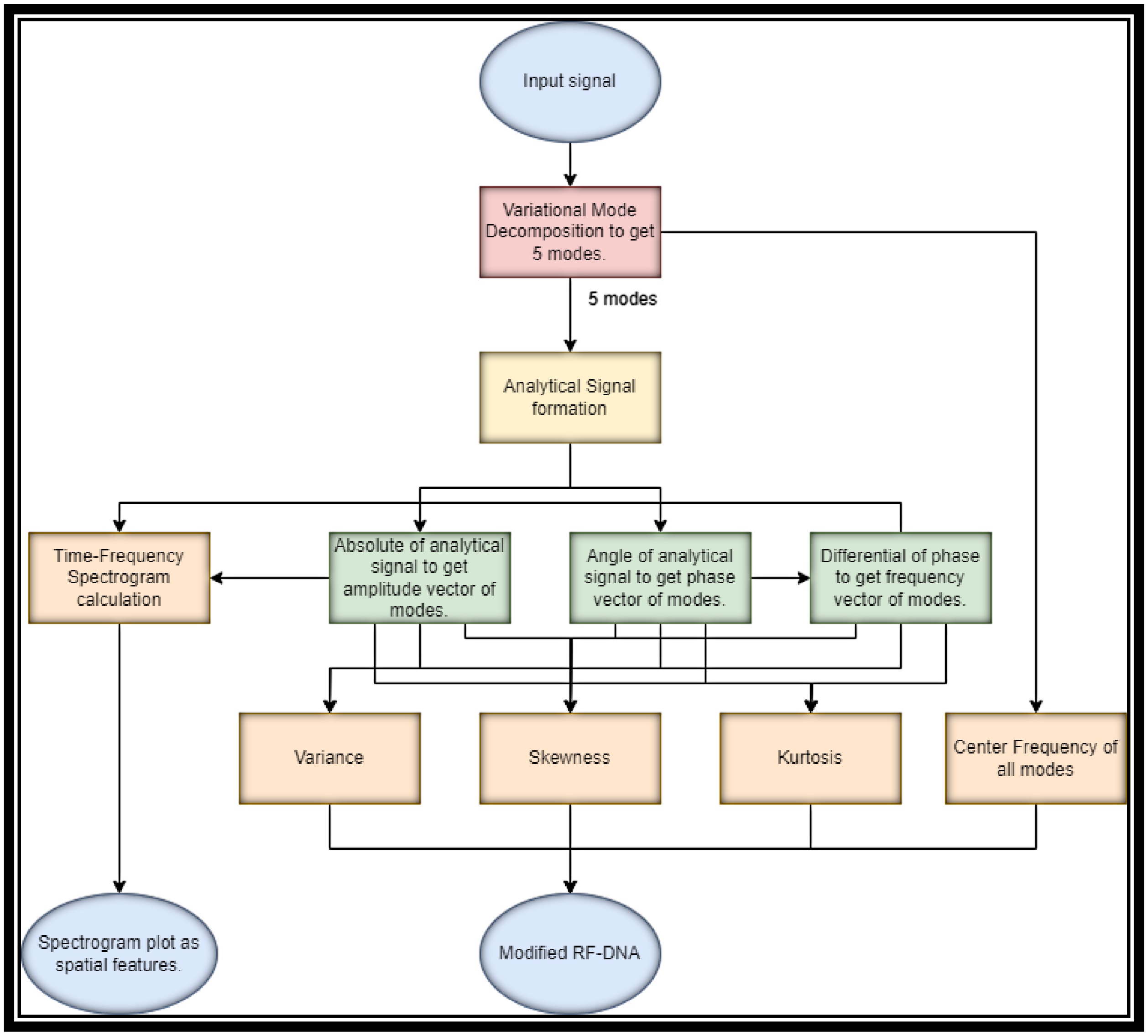

Input signal
Variational Mode Decomposition to get 5 modes.
5 modes
Analytical Signal formation
Time-Frequency Spectrogram calculation
Absolute of analytical signal to get amplitude vector of modes.
Angle of analytical signal to get phase vector of modes.
Differential of phase to get frequency vector of modes.
Variance
Skewness
Kurtosis
Center Frequency of all modes
Spectrogram plot as spatial features.
Modified RF-DNA

Figure 11. It shows the complete flow chart of bimodal feature extraction from received signal.

CHAPTER 5: MACHINE LEARNING CLASSIFICATION AND KNOWN EMITTER DETECTION.

During chapter 4, we discussed feature extraction and briefly described the models used in this project. In this chapter, we will investigate details regarding the classification requirement of our SEI problem and train a classifier for our SEI problem. Additionally, we will discuss why the selected model is the best option for our project.

5.1. Requirements for ML classifier:

Before we start discussing the model, we will investigate the requirements the classifier must fulfil for better specific emitter identification.

- The ML classifier must be able to classify known transmitters into the respective classes, i.e. transmitter 1, 2, 3, and 4, high accuracy.
- The ML classifier must be able to differentiate between known and unknown transmitters and classify all transmitters other than transmitter 1, 2, 3, and 4, as unknown transmitters or 0.
- The ML classifier should not take a lot of time to train and must be more resistant to overfitting.
- The classification time should be as small as possible.

5.2. XGBoost classifier for SEI ML classification:

Based on the requirements for the ML classifier for our SEI application, XGBoost is the classifier that checks all the boxes. As discussed in the section 2.1.3.2, the XGBoost classifier is considered as the best classifier. The combination of L1 and L2 regularization helps to prevent overfitting and its parallel approach gives better and speedier classification time.

5.2.1. Performance evaluation of XGBoost Classifier:

For this project, we trained the XGBoost classifier for classification into 5 classes, that are 0, 1, 2, 3, and 4. Where 0 represents the unknown transmitter, and 1, 2, 3, and 4; represent the respective transmitters. All the transmitted signals, as discussed in chapter 3 were processed and their modified RF-DNA features were extracted and used for training and testing.
Following are some of the performance parameters of XGBoost classifier for our dataset.

Accuracy: 99.018%

Confusion Matrix:

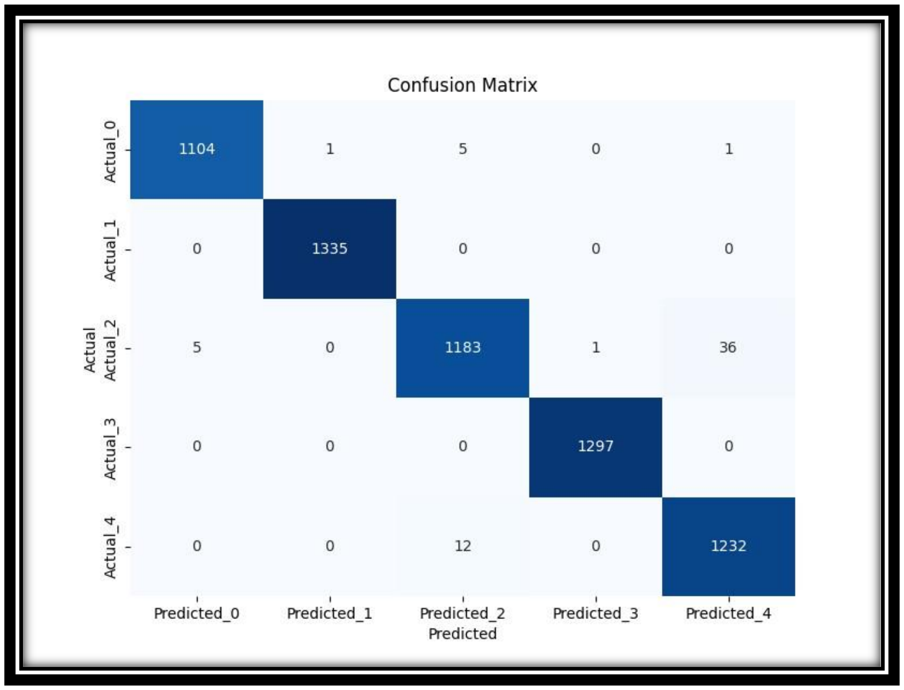


Figure 12. Confusion matrix of XGBoost Classifier

Classification report:

Table 1. Classification report of XGBoost

	Precision	**Recall**	**F1-score**	**Support**
0	1.00	0.99	0.99	1111
1	1.00	1.00	1.00	1335
2	0.99	0.97	0.98	1225
3	1.00	1.00	1.00	1297
4	0.97	0.99	9.98	1244
Accuracy			0.99	6212
Macro avg	0.99	0.99	0.99	6212
Weighted avg	0.99	0.99	0.99	6212

As we can see from these performance parameters, the results are great, and the accuracy is more than 99%.

5.3. Performance comparison with other models:

Now we will look at the accuracies of some other popular classification models and compare the results with XGBoost classifier.

5.3.1. Random Forest Classifier:

As another classifier that works on ensemble approach, RF classifier shows great prediction results and can be considered as the best option after XGBoost classifier.
The performance parameters of RF classifier are:
Accuracy: 98.82%

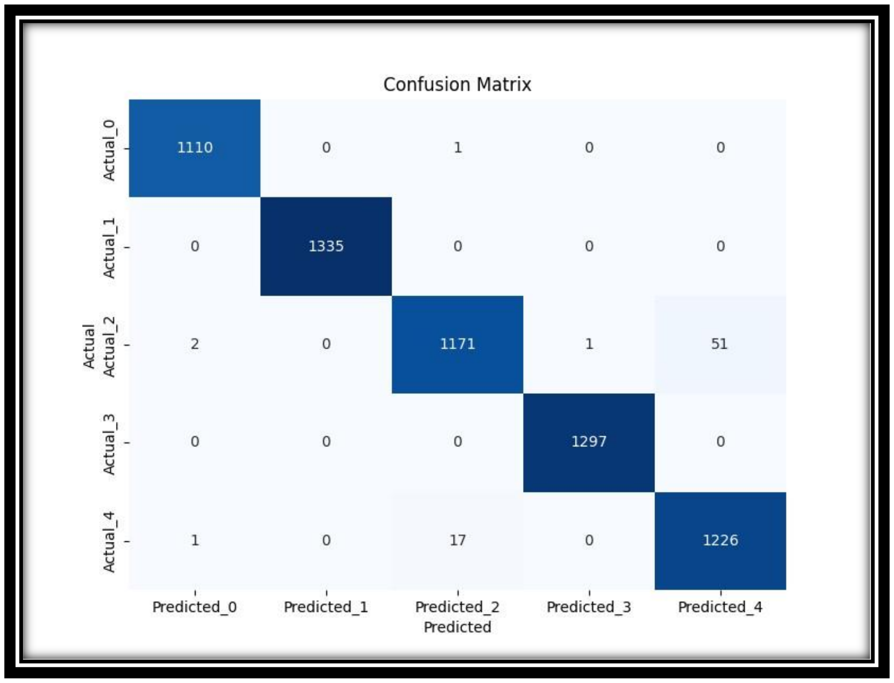


Figure 13. Confusion matrix of Random Forest Classifier

Classification report:

Table 2. Classification report of Random Forest Classifier

	precision	**recall**	**f1-score**	**support**
0	0.997305	0.9991	0.998201	1111
1	1	1	1	1335
2	0.984861	0.955918	0.970174	1225
3	0.99923	1	0.999615	1297
4	0.960063	0.985531	0.97263	1244
accuracy			0.988249	6212
macro avg	0.988292	0.98811	0.988124	6212
weighted avg	0.988374	0.988249	0.988235	6212

We can see that the results of random forest classifier are great, but the accuracy is a bit lower than the XGBoost classifier. Additionally, during real-time testing we noticed that XGBoost gave relatively better real-time prediction accuracies.

5.3.2. Decision Tree Classifier:

Now we will look at the performance of the decision tree classifier.
Accuracy: 97.21%
Confusion Matrix:

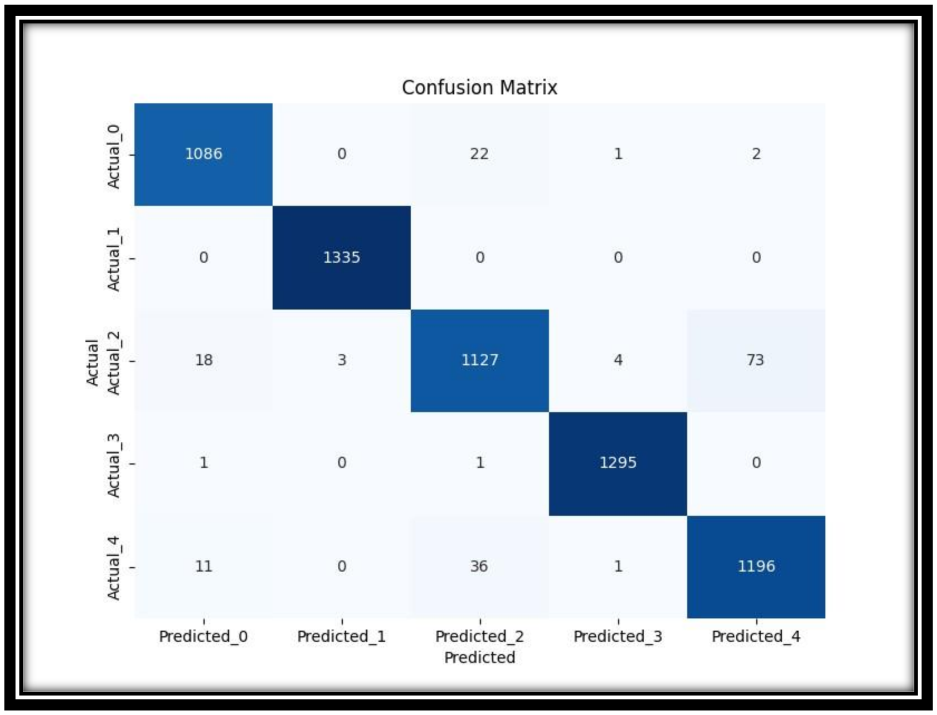


Figure 14. Confusion matrix of decision tree classifier

Classification report:

Table 3. Classification report of decision tree classifier.

	precision	**recall**	**f1-score**	**support**
0	0.973118	0.977498	0.975303	1111
1	0.997758	1	0.998878	1335
2	0.950253	0.92	0.934882	1225
3	0.995388	0.998458	0.996921	1297
4	0.940991	0.961415	0.951093	1244
accuracy			0.972151	6212
macro avg	0.971502	0.971474	0.971415	6212
weighted avg	0.972121	0.972151	0.972064	6212

5.3.3. K-Nearest Neighbor (KNN):

K-Nearest Neighbors (KNN) is a straightforward algorithm that predicts the output based on the majority vote of its nearest neighbors. While it is simple to implement, KNN can be computationally intensive during real-time prediction, as it requires calculating the distance to all training samples. Additionally, the algorithm's performance is sensitive to the choice of the number of neighbors and the distance metric used, which can significantly impact its accuracy and effectiveness.

We trained it on the same dataset and achieved the following results:
Accuracy: 88.313%

Confusion Matrix:

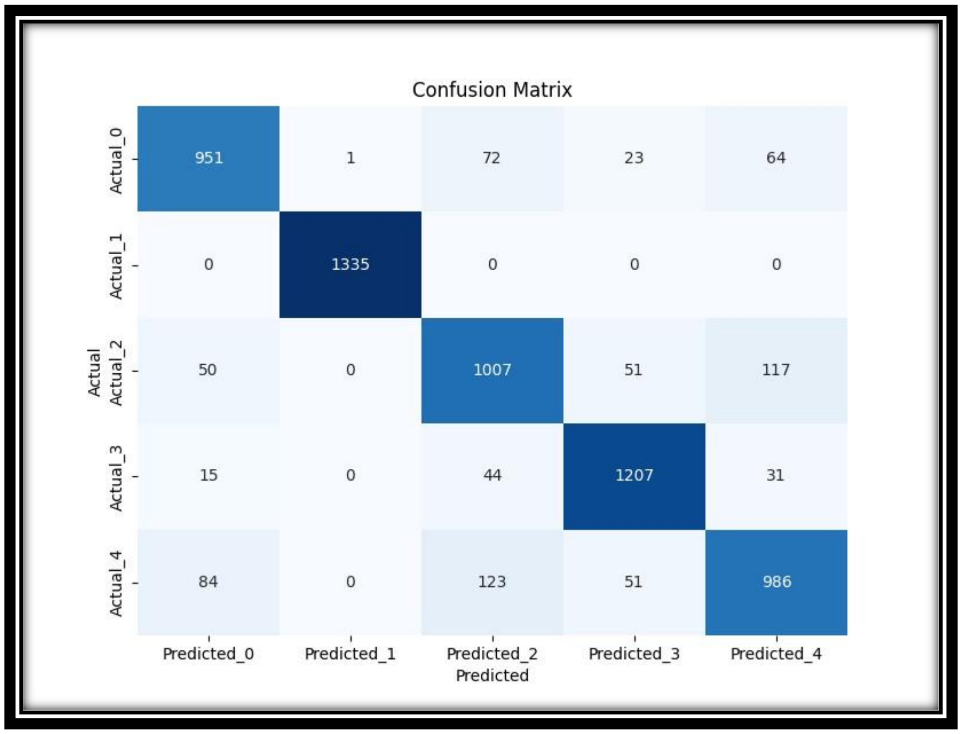


Figure 15. Confusion matrix of KNN classifier

Classification report

Table 4. Classification report of KNN classifier.

	precision	**recall**	**f1-score**	**support**
0	0.864545455	0.855985599	0.860244233	1111
1	0.999251497	1	0.999625608	1335
2	0.808186196	0.822040816	0.815054634	1225
3	0.906156156	0.930609098	0.918219855	1297
4	0.823038397	0.792604502	0.807534808	1244
accuracy			0.883129427	6212
macro avg	0.88023554	0.880248003	0.880135828	6212
weighted avg	0.882756462	0.883129427	0.882836109	6212

5.3.4. Gaussian naïve bayes classifier:

Naïve Bayes is a probability-based classifier that is known for its speed and effectiveness with high-dimensional data. However, it operates under the assumption that all features are independent of one another, which is rarely the case in real-world scenarios. This assumption often leads to lower accuracy, as it fails to account for the dependencies and interactions between features that are typically present in actual datasets.

We trained it on the same dataset and achieved the following results:

Accuracy: 41.47%

Confusion Matrix:

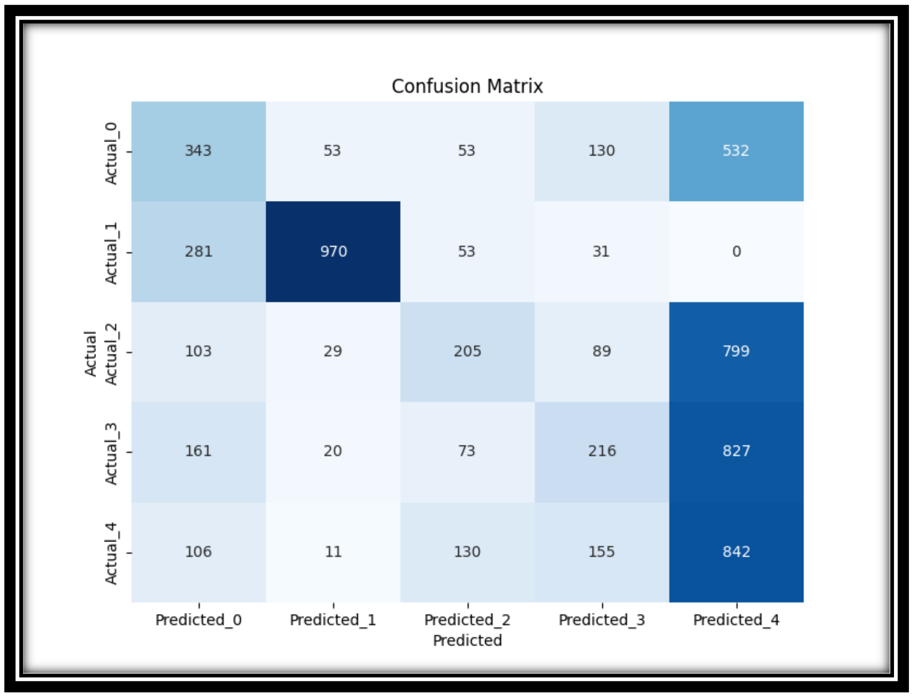


Figure 16. Confusion matrix of Naïve Bayes classifier

Classification report:

Table 5. Classification report of Naïve Bayes classifier.

	precision	**recall**	**f1-score**	**support**
0	0.345070423	0.308730873	0.325890736	1111
1	0.895660203	0.72659176	0.802315964	1335
2	0.398832685	0.167346939	0.235767683	1225
3	0.347826087	0.166538165	0.225234619	1297
4	0.280666667	0.676848875	0.396795476	1244
accuracy			0.414681262	6212
macro avg	0.453611213	0.409211322	0.397200896	6212
weighted avg	0.461675695	0.414681262	0.40368878	6212

5.3.5. Performance comparison of all classification models:

The following table provides the accuracies of all the classification models, and we can compare all of them to see which one provides the best output.

Table 6. Accuracy comparison of all classifiers.

Classifier Name	**Naïve Bayes**	**KNN**	**Decision Tree**	**Random Forest**	**XGBoost**
Accuracy	41.47%	88.313%	97.21%	98.82%	99.018%

So, from the table we can see that the best accuracy is achieved by XGBoost classifier. This is one of the reasons why we selected XGBoost classifier for our application. Additionally, the ability of XGBoost classifier to prevent overfitting and fast prediction is the major reasons it works best for SEI.

5.3.5.1 XGBoost model performance testing and results:

After model training, we tested the model in different scenarios and checked how the real-time accuracy of the model changes.

5.3.5.1.1 Varying message signal:

For the first test, we changed the frequencies of the message signal to check whether the model was message signal independent or not.
The message signal for this experiment is a single tone frequency, which is AM modulated by another sinusoidal signal with frequencies 10x the frequency of single tone. Then this signal is digitally modulated by USRP kits at the carrier frequency of 2.45Ghz. And for the IQ rate of 100k samples per second, i.e. bandwidth of 80kHz, we received the following results:

Table 7. Change in accuracy w.r.t change in message signal

Message signal frequency	**AM carrier signal**	**Reading1**	**Reading2**	**Reading3**	**Reading4**	**Reading5**	**Average%**	**Detected label**	**Actual label**
100	1000	99.3645	95.1726	97.3311	98.8127	98.9	97.91618	1	1
1000	10000	99.2487	96.2307	96.6031	99.4197	98.0649	97.91342	1	1
2000	20000	99.479	57.5632	99.0248	99.6995	50.5697	81.26724	1	1
3000	30000	98.8513	99.5613	98.9454	99.3997	99.1633	99.1842	1	1
4000	40000	97.4218	99.7602	99.0055	97.0613	99.4006	98.52988	1	1
5000	50000	75.3121	97.825	99.653	99.253	98.151	94.03882	1	1
6000	60000	53.4073	68.1458	99.4825	70.251	57.2123	69.69978	1	1
7000	70000	74.1561	97.7465	76.2553	77.2986	68.6164	78.81458	1	1
8000	80000	99.8976	99.8165	73.6377	53.5638	99.589	85.30092	1	1
9000	90000	56.7462	99.6529	96.588	99.3214	97.5352	89.96874	1	1
10000	100000	99.6101	99.8687	81.1752	81.3119	50.5848	82.51014	1	1
20000	200000	62.2334	64.5919	59.3598	70.4355	99.4816	71.22044	1	1
30000	300000	71.8504	99.6477	99.7082	98.6364	99.7457	93.91768	1	1
40000	400000	99.5799	98.646	98.1905	98.7895	55.6806	90.1773	1	1
50000	500000	99.8454	99.7679	73.6047	76.3348	68.3284	83.57624	1	1
60000	600000	59.7033	54.4735	70.3663	99.67	98.1856	76.47974	1	1
70000	700000	76.6172	64.281	52.1021	99.5501	99.5852	78.42712	1	1
80000	800000	79.7743	64.7061	55.1155	61.4006	53.0905	62.8174	1	1

5.4.1. Varying IQ rate:

Now we will try to change the IQ rate of the transmitting device and keep the message signal constant to see how the accuracy changes according to the change in IQ rate.

Table 8. Change in accuracy w.r.t change in IQ rate

IQrate	**Reading1**	**Reading2**	**Reading3**	**Reading4**	**Reading5**	**Average%**	**Detected label**	**Actual label**
100k	99.9788	99.9801	99.9785	99.9817	99.9698	99.97778	1	1
50k	99.9367	99.9794	99 .9485	99.4028	99.9397	99.81436	1	1
25k	99.9266	99.9643	99.9444	99.9655	99.9751	99.95518	1	1
75k	73.3475	53.3132	76.6858	81.0719	83.0082	72.60813	1	1

We can see that by changing the message signal and the IQ rate, the accuracies do get affected but are still more than 70% and the label is not a false result. So, we can say that our algorithm works very well.

CHAPTER 6 – DEEP LEARNING AND SINGLE SHOT IDENTIFCATION OF UNKNOWN TRANSMITTERS

In chapter 4, we discussed the feature extraction from a signal and focused on the combination of manual feature-based and deep learning-based algorithms. We've utilized the manual features in chapter 5 and discussed several machine learning algorithms and selected XGBoost as our classifier. In this chapter, we will focus on the deep learning-based specific emitter identification and the specific algorithm we have selected. But first let's discuss why we need deep learning when our machine learning model is working perfectly.

6.1. Deep learning and artificial neural network:

Deep learning is a subset of machine learning that uses artificial neural networks and deep neural networks to model and solve complex problems. By mimicking the structure of human brain, neural networks are multiple layers of neurons (perceptron) that can extract hidden features and detect complex structures and patterns.

6.1.1. Convolutional neural networks:

Artificial neural networks are of many types, one of the widely used neural networks is convolutional neural networks (CNN). CNN is a specialized neural network that is used to extract hidden features from spatial information like images.

CNNs have three basic layers, a convolutional layer that performs convolution on the input image using a kernel filter of a specific size. The number of times the kernel filter is applied is the depth of the output feature map. After the convolutional layer, there is a pooling and a batch normalization layer with a specific activation function, usually ReLU.

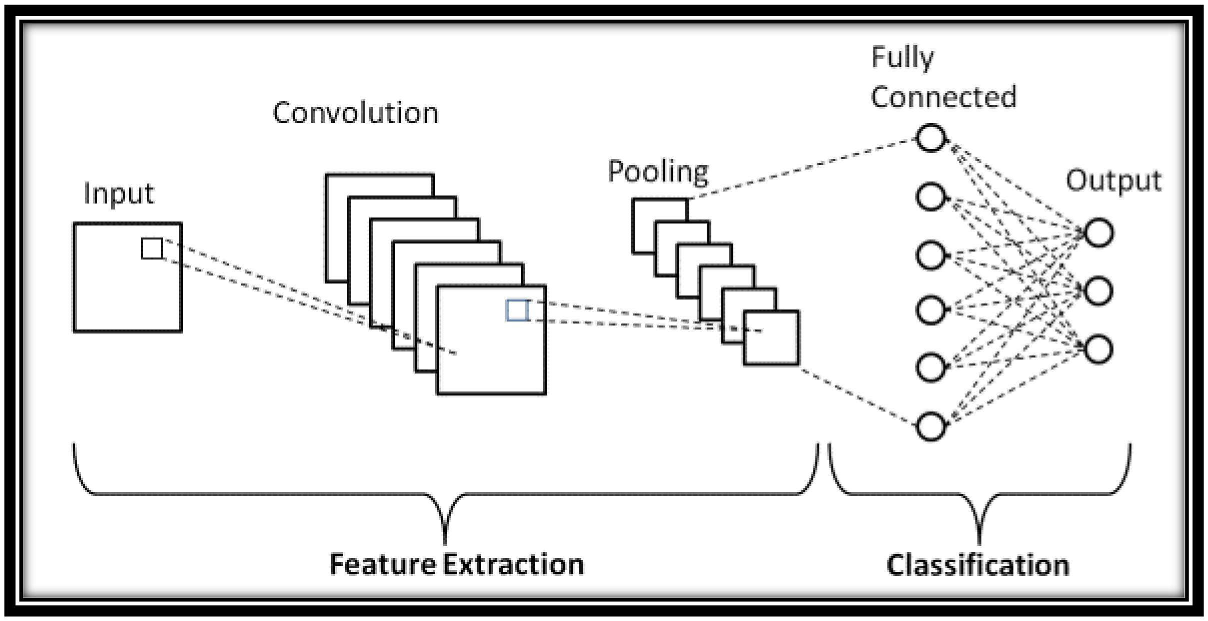


Figure 17. Basic working of a CNN based classifier.

CNNs are used to extract hidden features from spatial data while preserving the spatial information of the network. After the convolutional layers, there are some fully connected neural networks that can be utilized for many purposes. Figure 11 shows how CNN is used for classification.

6.2. Identification, classification, and labelling of unknown transmitters in real-time:

In specific emitter identification, many algorithms work on identifying transmitters using classifiers that are trained on huge amount of data but in real time scenarios it can be extremely difficult to collect that much data of hundreds and thousands of transmitters and it becomes almost impossible to collect data of transmitters that are completely unknown to us and are not in our control.

For example, in military scenarios, the data of transmitters from other countries is not available and it is extremely important to detect, label and remember the transmitters on minimum amount of data. For such type of predictions, one-shot or few-shot algorithms are needed that can not only detect the transmitter as an unknown but also check whether this specific unknown was ever detected earlier or not.
In this project, we proposed and developed a novel way to identifying transmitters by comparing the received signal with the signal of previously identified transmitters using a single shot algorithm called Siamese networks.

6.2.1. Siamese network for specific emitter identification:

Siamese network is a type of neural network that contains two or more identical networks that convert two inputs into a space representation space which is them compared to predict if both inputs are from the same class or not. Usually, Euclidean distance is used to calculate the similarity between two inputs, but cosine similarity and other similar functions can be used.
For specific emitter identification, we trained a Siamese network that utilized the bimodal feature fusion, as proposed in [14], to take modified RF-DNA with all modes, and time frequency spectrogram as spatial features and are passed through a convolutional neural network that combines the features extracted from both inputs, and some fully connected layers to give an output of size 1x32. This output is the shared representation space and to measure the similarity between two inputs, both are converted to their shared representation space and then the Euclidean distance between them is calculated.

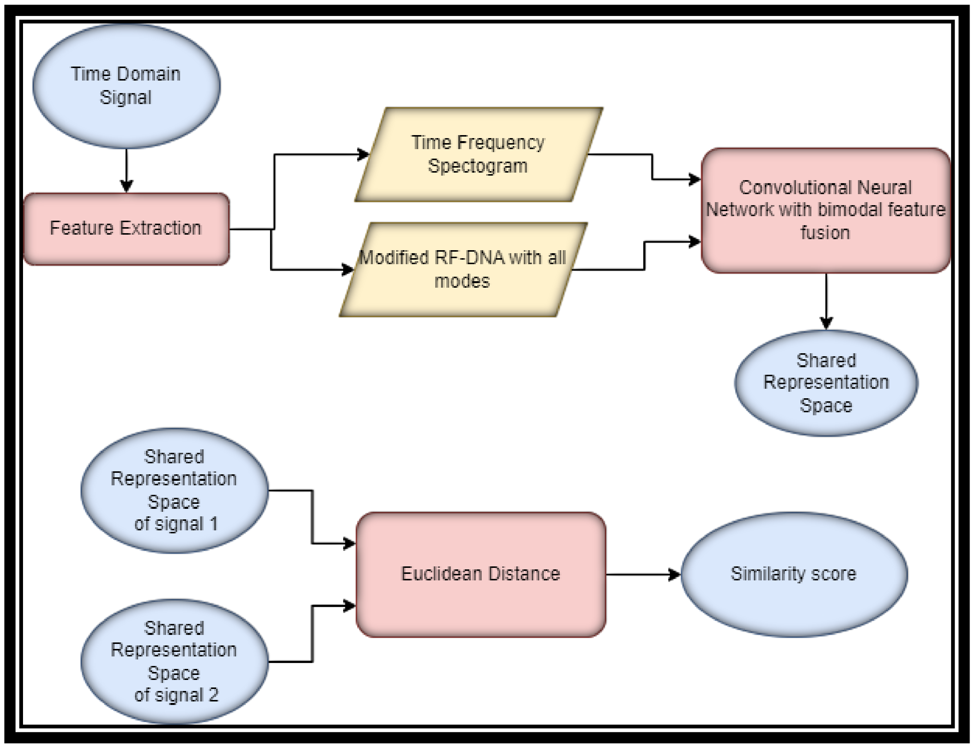

Time Domain Signal
Feature Extraction
Time Frequency Spectogram
Modified RF-DNA with all modes
Convolutional Neural Network with bimodal feature fusion
Shared Representation Space
Shared Representation Space of signal 1
Shared Representation Space of signal 2
Euclidean Distance
Similarity score

Figure 18. Flow diagram of specific emitter identification with Siamese networks.

In figure 12, we can the visual representation of how two signals can be compared with each other to calculate their similarity score. This Euclidean distance is then compared with a constant and if it is lower than that constant we say that the inputs are from same transmitter otherwise they are from different transmitters.

6.2.1.1. Loss function for Siamese networks:

To train the Siamese network for our project, we utilized triplet loss. Triplet loss teaches algorithms to recognize the similarity and difference between items and uses the group of three items, called triplets, containing an anchor, an item from the same class (positive) and an item from the different class (negative). The major goal of triplet loss is to make the model learn that the anchor is closer to the positive and different from the negative.
Mathematically, triplet loss can be written as:

$$\sum_{i}^{N}[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha]$$

Where, f(x) is the share representation space of anchor, positive and negative and i represents the ith input and α is the bias. The goal is to minimize the whole term by minimizing the first term and maximizing the second term and bias acts as threshold.

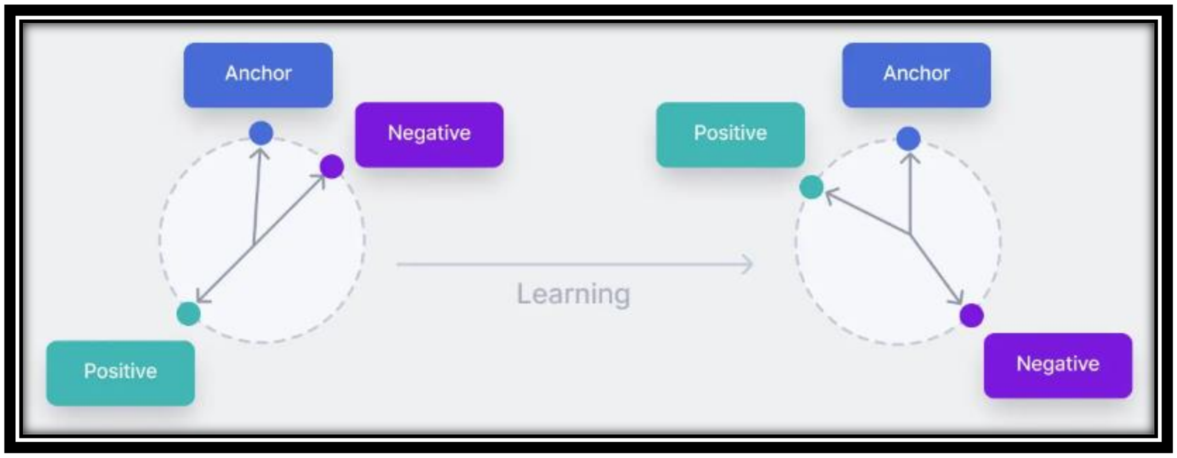


Figure 19. Working of a triplet loss

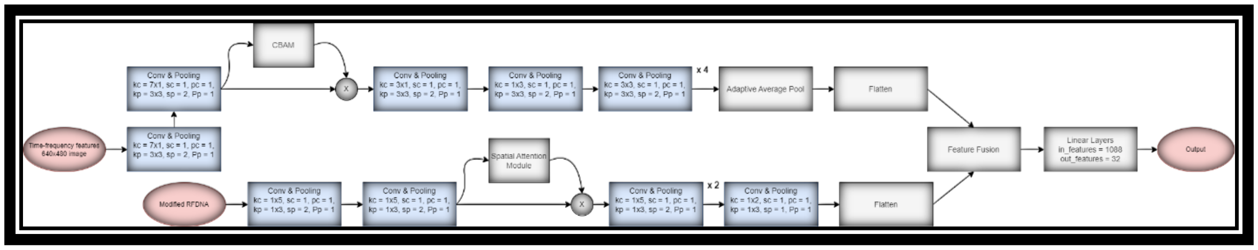
The python implementation of the triplet loss is as follow:

```
class TripletLoss(nn.Module):
    def __init__(self, margin=2.0):
        super(TripletLoss, self).__init__()
        self.margin = margin

    def forward(self, anchor, positive, negative):
        distance_positive = F.pairwise_distance(anchor, positive)
        distance_negative = F.pairwise_distance(anchor, negative)
        loss = torch.mean(torch.clamp(distance_positive - distance_negative + self.margin,
min=0.0))
        return loss
```

6.2.1.2. Convolutional neural network layer with bimodal feature fusion:

As shown in figure 12, we are using a 2 channel CNN layer for bimodal feature extraction and fusion. Now we will look at the neural network architecture.



CBAM
Conv & Pooling kc = 7x1, sc = 1, pc = 1, kp = 3x3, sp = 2, Pp = 1
X
Conv & Pooling kc = 3x1, sc = 1, pc = 1, kp = 3x3, sp = 2, Pp = 1
Conv & Pooling kc = 1x3, sc = 1, pc = 1, kp = 3x3, sp = 2, Pp = 1
Conv & Pooling kc = 3x3, sc = 1, pc = 1, kp = 3x3, sp = 2, Pp = 1
x 4
Adaptive Average Pool
Flatten
Time-frequency features 640x480 image
Conv & Pooling kc = 7x1, sc = 1, pc = 1, kp = 3x3, sp = 2, Pp = 1
Spatial Attention Module
Feature Fusion
Linear Layers in_features = 1088 out_features = 32
Output
Modified RFDNA
Conv & Pooling kc = 1x5, sc = 1, pc = 1, kp = 1x3, sp = 2, Pp = 1
Conv & Pooling kc = 1x5, sc = 1, pc = 1, kp = 1x3, sp = 2, Pp = 1
X
Conv & Pooling kc = 1x5, sc = 1, pc = 1, kp = 1x3, sp = 2, Pp = 1
x 2
Conv & Pooling kc = 1x2, sc = 1, pc = 1, kp = 1x3, sp = 1, Pp = 1
Flatten

Figure 20. 2-channel CNN with bimodal feature fusion

Where the convolutional block attention module (CBAM), spatial attention module (SAM), and channel attention module (CAM) are used to enhance the distinguishability between fingerprint features and to improve weight of important parts of subsequent feature. The structures of these attention modules can be seen as follows:

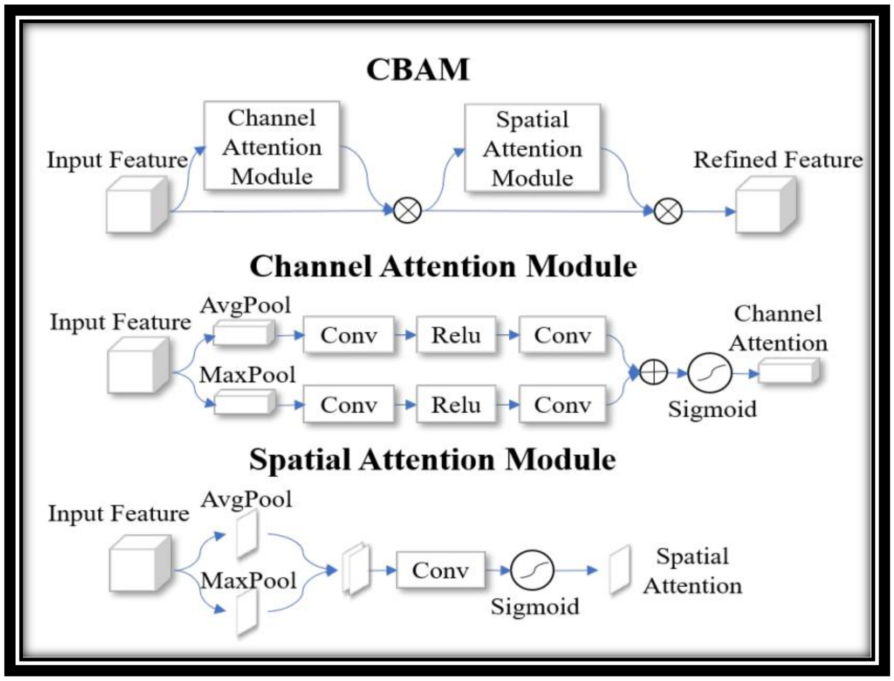


Figure 21. The structures of attention modules.

Where the python implementation of this neural network is as follows:

```
class SiameseNetwork(nn.Module):
    def __init__(self, num_classes, batch_size):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv1d(in_channels=1, out_channels=32, kernel_size=3, padding=1,
stride=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm1d(32),
            nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3, padding=1,
stride=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm1d(64)
        )
        self.conv2 = nn.Sequential(
            nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3, padding=1,
stride = 1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm1d(128),
            nn.Conv1d(in_channels=128, out_channels=256, kernel_size=3, padding=1,
stride = 1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm1d(256)
        )
        self.conv3 = nn.Sequential(
            nn.Conv1d(in_channels=256, out_channels=512, kernel_size=2, padding=1,
stride=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(512)
        )
        self.convs = nn.Sequential(
            nn.Conv2d(in_channels = 3, out_channels = 64, kernel_size=(7,1), stride=1,
padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(1,7),
stride=1, padding=0),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64)
        )
        self.convs2 = nn.Sequential(
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,1),
stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(1,3),
stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
    stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
    stride=1, padding=1),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=(3,3), stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.AdaptiveAvgPool2d(output_size=(1,1)),
            nn.Flatten()
        )
        self.avg2d = nn.AvgPool2d(kernel_size=(3,3), stride=1, padding=1)
        self.max2d = nn.MaxPool2d(kernel_size=(3,3), stride=1, padding=1)
        self.conv_cam = nn.Sequential(
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
    stride=1, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.Conv2d(in_channels = 64, out_channels = 64, kernel_size=(3,3),
    stride=1, padding=1),
            nn.ReLU(),
            nn.BatchNorm2d(64)
        )
        self.avg = nn.AvgPool1d(kernel_size=1,padding=0, stride=1)
        self.max = nn.MaxPool1d(kernel_size=1, stride=1, padding=0)
        self.avg2 = nn.AvgPool2d(kernel_size=(3,3),padding=1, stride=1)
        self.max2 = nn.MaxPool2d(kernel_size=(3,3), stride=1, padding=1)
        self.spm = nn.Sequential(nn.Conv1d(in_channels=128, out_channels=64,
    kernel_size=1, padding=0, stride=1),
                        nn.ReLU(),
                        )
        self.spm2 = nn.Sequential(nn.Conv2d(in_channels=128, out_channels=64,
    kernel_size=(3,3), padding=1, stride=1),
                        nn.ReLU())
        self.soft = nn.Softmax(dim=1)
        self.fc1 = nn.Flatten()
        self.linear = nn.Sequential(
                        nn.Linear(in_features=1600, out_features=512),
                        nn.BatchNorm1d(512),
                        nn.ReLU(),
                        nn.Linear(in_features=512, out_features=256),
                        nn.BatchNorm1d(256),
                        nn.ReLU(),
                        nn.Linear(in_features=256, out_features=128),
                        nn.BatchNorm1d(128),
                        nn.ReLU(),
                        nn.Linear(in_features=128, out_features=64),
                        nn.BatchNorm1d(64),
                        nn.ReLU(),
                        nn.Linear(in_features=64, out_features=32),
                        nn.BatchNorm1d(32),
                                nn.ReLU(),
                                )
        # self.fc2 = nn.Linear(512, num_classes)

    def cam(self, x):
        x1 = self.avg2d(x)
        x1 = self.conv_cam(x1)
        x2 = self.max2d(x)
        x2 = self.conv_cam(x2)
        x = x1+x2
        x = self.soft(x)
        return x

    def cbam(self,x):
        x2 = self.cam(x)
        x = x2*x
        x2 = self.sam2(x)
        x = x2*x
        return x

    def vmd_s(self,x):
        x = self.convs(x)
        x2 = self.cbam(x)
        x = x*x2
        x = self.convs2(x)
        return x

    def sam(self,x):
        x3 = self.avg(x)
        x4 = self.max(x)
        x2 = torch.cat([x3, x4], dim=1)
        x2 = self.spm(x2)
        x2 = self.soft(x2)
        return x2

    def sam2(self,x):
        x3 = self.avg2(x)
        x4 = self.max2(x)
        x2 = torch.cat([x3, x4], dim=1)
        x2 = self.spm2(x2)
        x2 = self.soft(x2)
        return x2

    def vmd_t(self,x):
        x = x.unsqueeze(1)
        x = self.conv1(x)
        x2 = self.sam(x)
        x2 = x2.expand_as(x)
        x = x2*x
        x = x.squeeze(2)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.fc1(x)
        return x
```

```
            def forwarda(self, x):
                temporal = x['temporal'] #temporal features VMDT
                tf = x['tf'] #Time frequency features VMD_S
                temporal = self.vmd_t(temporal)
                tf = self.vmd_s(tf)
                x = torch.cat([temporal, tf], dim=1)
                x = self.linear(x)
        return x
```

6.2.1.3. Siamese neural network (SNN) training and results:

To train the SNN, we took the data of 6 USRP devices, extracted features and trained using triplet loss. The training parameters are as follows:

Table 9. Summary and training parameters for Siamese Neural Network

Parameter name	**Parameter value**
Train size	80% of dataset
Test size	20% of dataset
Time-frequency spectrogram image size	620x480
Device	Cuda
Batch size	8
Learning rate	0.001 for first 5 epochs and 0.0001 for later.
No of epochs	15
Loss function	Triplet loss
Optimizer	Adam with momentum 0.9
Programming language	Python
Library	Pytorch
Distance setpoint	1.8
Achieved accuracy	90%

So, from the table we can see that when the distance setpoint is set to 1.8, the achieved accuracy is 90%.

6.3 The Fusion of the DL Model with the Chosen ML Model

The novelty of this project lies in the combined usage of both Machine Learning (ML) and Deep Learning (DL) models. The ML model is responsible for the classification of known transmitters and the identification of any unknown signals or potential noise. On the other hand, the DL model works in conjunction with the ML model and takes additional inputs such as time-spectrogram features. These features are passed through a detailed architecture that includes attention blocks, which prioritize decisive parameters crucial for classifying signals emitted by unknown devices.

6.3.1 Workflow Description:

1. **ML Model**:
    - **Classification of Known Transmitters**: The ML model classifies signals from known transmitters.

   - **Detection of Unknown Signals**: It also identifies the presence of unknown signals or potential noise.
2. **DL Model**:
   - **Enhanced Feature Extraction**: The DL model takes in time-spectrogram features along with other inputs (Modified RF-DNA).
   - **Attention Mechanism**: It uses attention blocks to focus on critical parameters that are key to classifying signals from unknown devices.
3. **Combined Model Operation**:
   - **Shared Representation**: The DL model transforms these features into a shared representation.
   - **Signal Classification**: The combined model checks if it has previously encountered the signal from a particular transmitter.
   - **Decision Making**: If the signal is recognized, it is classified accordingly. If it is unrecognized, the signal is stored in the database and assigned a new class.

6.3.2 Advantages and Application:

- **Improved Accuracy**: The combined approach leverages the strengths of both ML and DL models to achieve higher accuracy in signal classification.
- **Real-time Signal Identification**: The models work in real-time to identify and classify signals, making them suitable for dynamic and changing environments.
- **Adaptive Learning**: The system learns continuously, updating the database with new signals and improving classification accuracy over time.

CHAPTER 7 – SPECIFIC EMITTER IDENTIFICATION SOFTWARE AND HARWARE SETUP

The process of specific emitter identification is done is three steps, data acquisition, feature extraction and model prediction. In chapter 3 we discussed how we collected data, chapter 4 focused on feature extraction, and in chapters 5 and 6 we discussed the machine learning and deep learning methods for our application. In this chapter, we will focus on the specific emitter identification software and its hardware setup for real-time specific emitter identification.

7.1. Hardware Setup and data acquisition:

The hardware setup for the specific emitter identification real-time application consists of a receiver, a processor, and a display. The signal acquired from the receiver is sent to the SEI application which is being run in the processor and is responsible for the feature extraction, machine learning prediction, and Siamese network unknown emitter identification. We will discuss the SEI application in detail in the next section and in this section, we will focus on the hardware.

7.1.1. Receiver for SEI application:

There are two main receivers that are being used for the SEI application. One is a USRP2901 device, and the other is RTLSDR receiver.

7.1.1.1. NI-USRP2901:

As discussed in chapter 3, NI-USRP2901 is a software defined radio developed by national instruments and can be configured using LabVIEW's communication toolbox. USRP devices are a great option for specific emitter identification application as they provide a huge range of center frequencies, IQ rates, number of samples per snapshot, and high-quality signal transmission and reception. The only downside about it is the lack of compatibility with python and the need for LabVIEW for signal acquisition.
While reception, the SEI models work best with the following configuration:

- Carrier Frequency: 2.45GHz
- IQ rate: 100k samples per second
- Gain: 0 dB
- No of samples per snapshot: 10000

And the LabVIEW code is as shown below:

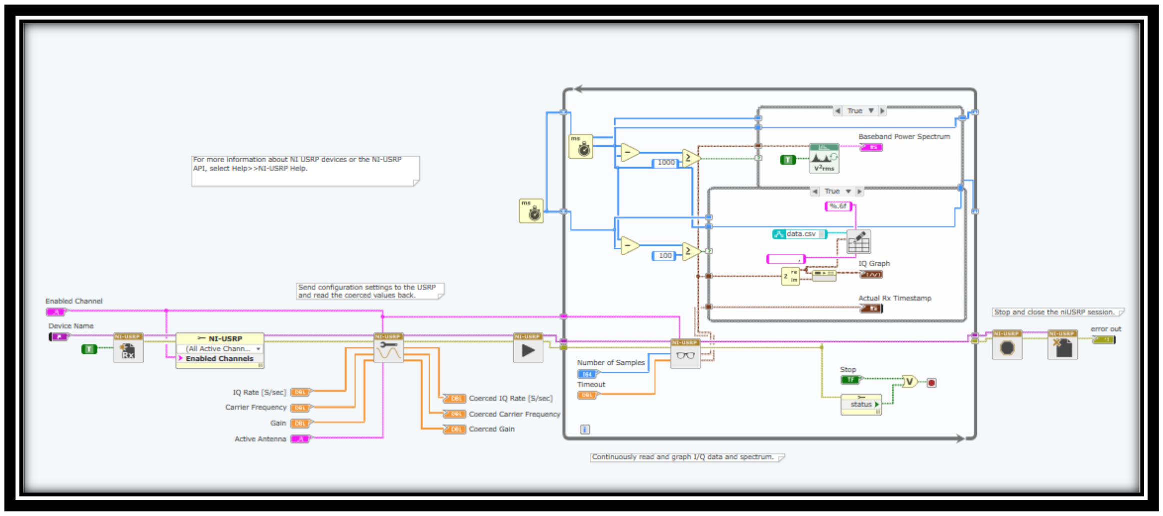

For more information about NI USRP devices or the NI-USRP API, select Help>>NI-USRP Help.
Send configuration settings to the USRP and read the coerced values back.
Enabled Channel
Device Name
NI-USRP
(All Active Chann...
Enabled Channels
IQ Rate [S/sec]
Carrier Frequency
Gain
Active Antenna
Coerced IQ Rate [S/sec]
Coerced Carrier Frequency
Coerced Gain
Number of Samples
Timeout
True
Baseband Power Spectrum
1000
100
%.6f
data.csv
IQ Graph
Actual Rx Timestamp
Stop
status
Stop and close the niUSRP session.
error out
Continuously read and graph I/Q data and spectrum.

Figure 22: Signal acquisition LabVIEW code

7.1.1.2. RTLSDR:

RTLSDR is a cheap software defined radio that can be configured using different ways, one of which is using python's pyrtlsdr library. RTLSDR works in a 24MHz to 1.7GHz center frequency range and has a bandwidth of 2.4Mhz, i.e. an IQ rate of 3Mhz. RTLSDR's utility is its availability in python making it a good and cheap receiver for a complete specific emitter identification software.
SEI application with RTLSDR receiver works best with the following configuration:

- Carrier frequency: 1GHz
- IQ rate: 3Mhz
- Samples per snapshot: 10000
- Gain: Auto

7.1.2. Processing unit and display for SEI application:

The major processing unit for the specific emitter identification application is the MSI gf63mini laptop with its built-in display. The high computational complexity of the SEI application demands a high-performance laptop, and this is the major reason for the use of this hardware for SEI application. The specifications of this processing unit are as follows:

- CPU: Intel(R) Core(TM) i5-10500H CPU @ 2.50GHz   2.50 GHz
- GPU: NVIDIA GTX 1650Ti.
- RAM: 16GB
- Operating system: Windows 11

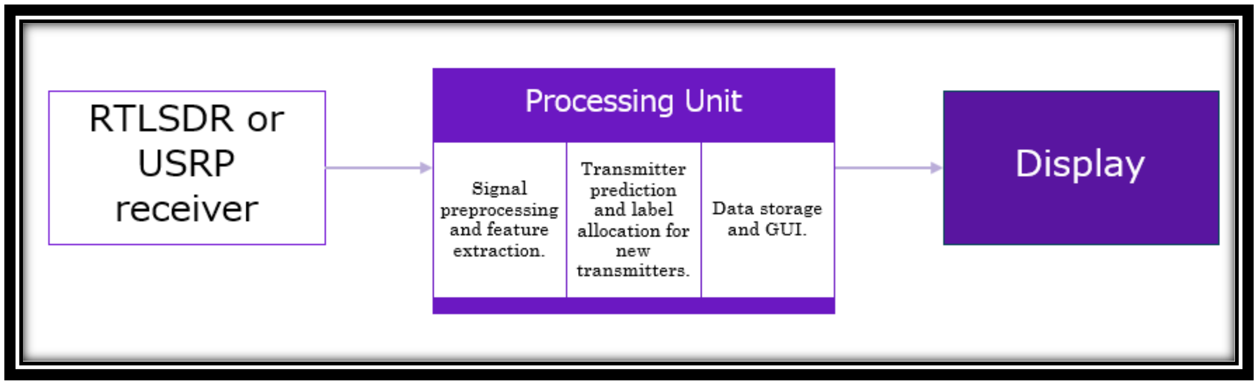

RTLSDR or USRP receiver
Processing Unit
Signal preprocessing and feature extraction.
Transmitter prediction and label allocation for new transmitters.
Data storage and GUI.
Display

Figure 23. Block diagram of hardware configuration.

7.2. Specific emitter identification application:

The specific emitter identification application is a software developed in python using Kivy framework for GUI. This software has two main versions, one suitable when USRP kit is being used as receiver and the other when RTLSDR is being used. Although the basic algorithm is the same there are minimal differences in data acquisition and transmitter configuration i.e. GUI. But before we discuss the differences, let's discuss the complete specific emitter identification algorithm containing the feature extraction and prediction with both XGBoost classifier and Siamese Network.

7.2.1. Complete specific emitter identification algorithm:

For the specific emitter identification project, SEI is done in three steps after data acquisition: feature extraction, known emitter classification or unknown emitter detection, and then unknown emitter classification and real-time labelling.

7.2.1.1. Feature Extraction:

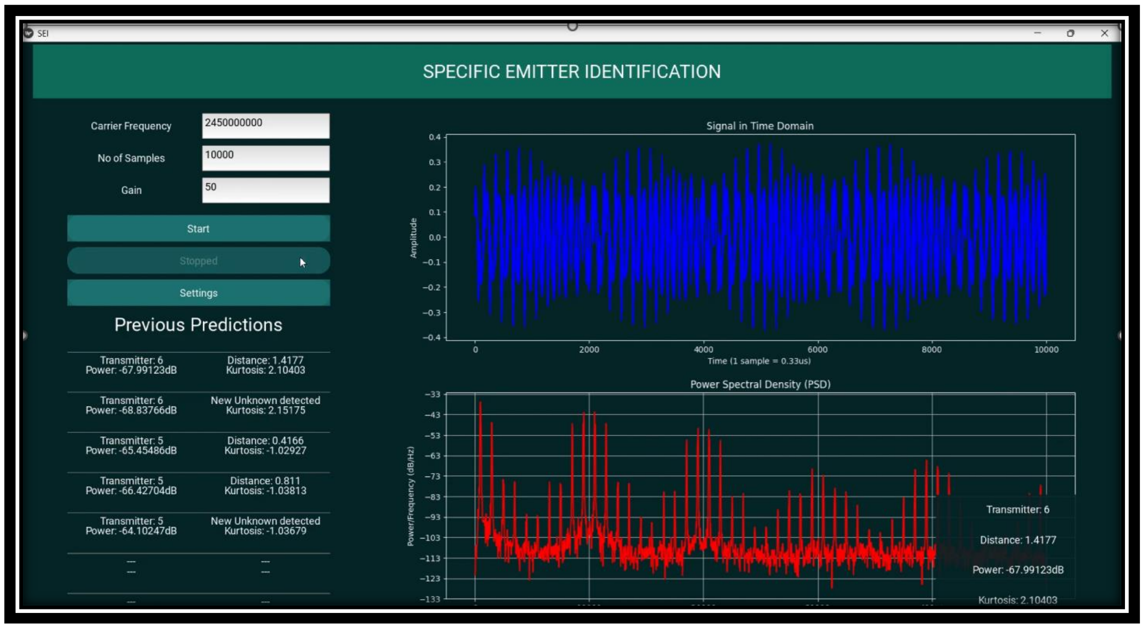
When the data is received from the receiving device, it is sent to the feature extraction function that extracts the modified RF-DNA features and the time-frequency spectrogram from the signal. The details of these algorithms are discussed in chapter 4. In addition to the feature extraction, other information from the received signal is also extracted, including the signal power, its kurtosis, and its power spectral density (PSD) which is displayed on the GUI which will be discussed later. Additionally, the extracted features are sent to the prediction function responsible for both ML and SNN predictions.

7.2.1.2. XGBoost classification and unknown emitter detection:

At this stage, extreme gradient boost pretrained classifier is loaded using python's joblib library and is used to predict the transmitter of the received signal. At this stage the classifier either outputs the specific transmitter, i.e. 1, 2, 3, and 4, or detects the unknown transmitter by giving an output of 0. If the unknown transmitter is detected, the program proceeds to the Siamese neural network's function, otherwise the application displays the results and the prediction parameters including the prediction accuracy and the predicted transmitter.

7.2.1.3. Siamese network classification and unknown emitter labelling:

If the unknown is detected by the XGBoost classifier, the signal is sent to the Siamese Network's function that converts the signal to its shared representation space using the 2-channel bimodal feature fusion network and then calculates the Euclidean distance between the shared representation space of previously stored and labelled signals. If the lowest of all the distances is less than 1.8, we label the received signal as the same label as the matched transmitter but if no match is found, meaning if the lowest Euclidean distance is greater than 1.8, then the signal is said to be from a completely unknown transmitter, and it is assigned a new label and stored in the database.
For example, if we have previously detected three unknowns and labelled them in real-time as -1, -2, and -3, and if the received signal doesn't match with any of these, then the signal will be saved with the label of -4 and our application will saw that a new unknown was discovered.



Carrier Frequency: 2450000000
No of Samples: 10000
Gain: 50

Previous Predictions

Transmitter: 6 Power: -67.99123dB	Distance: 1.4177 Kurtosis: 2.10403
Transmitter: 6 Power: -68.83766dB	New Unknown detected Kurtosis: 2.15175
Transmitter: 5 Power: -65.45486dB	Distance: 0.4166 Kurtosis: -1.02927
Transmitter: 5 Power: -66.42704dB	Distance: 0.811 Kurtosis: -1.03813
Transmitter: 5 Power: -64.10247dB	New Unknown detected Kurtosis: -1.03679
--- ---	--- ---
---	---

Transmitter: 6
Distance: 1.4177
Power: -67.99123dB
Kurtosis: 2.10403

Figure 24. New transmitter detection GUI display

So, the complete flow diagram of our specific emitter identification algorithm can be displayed as:

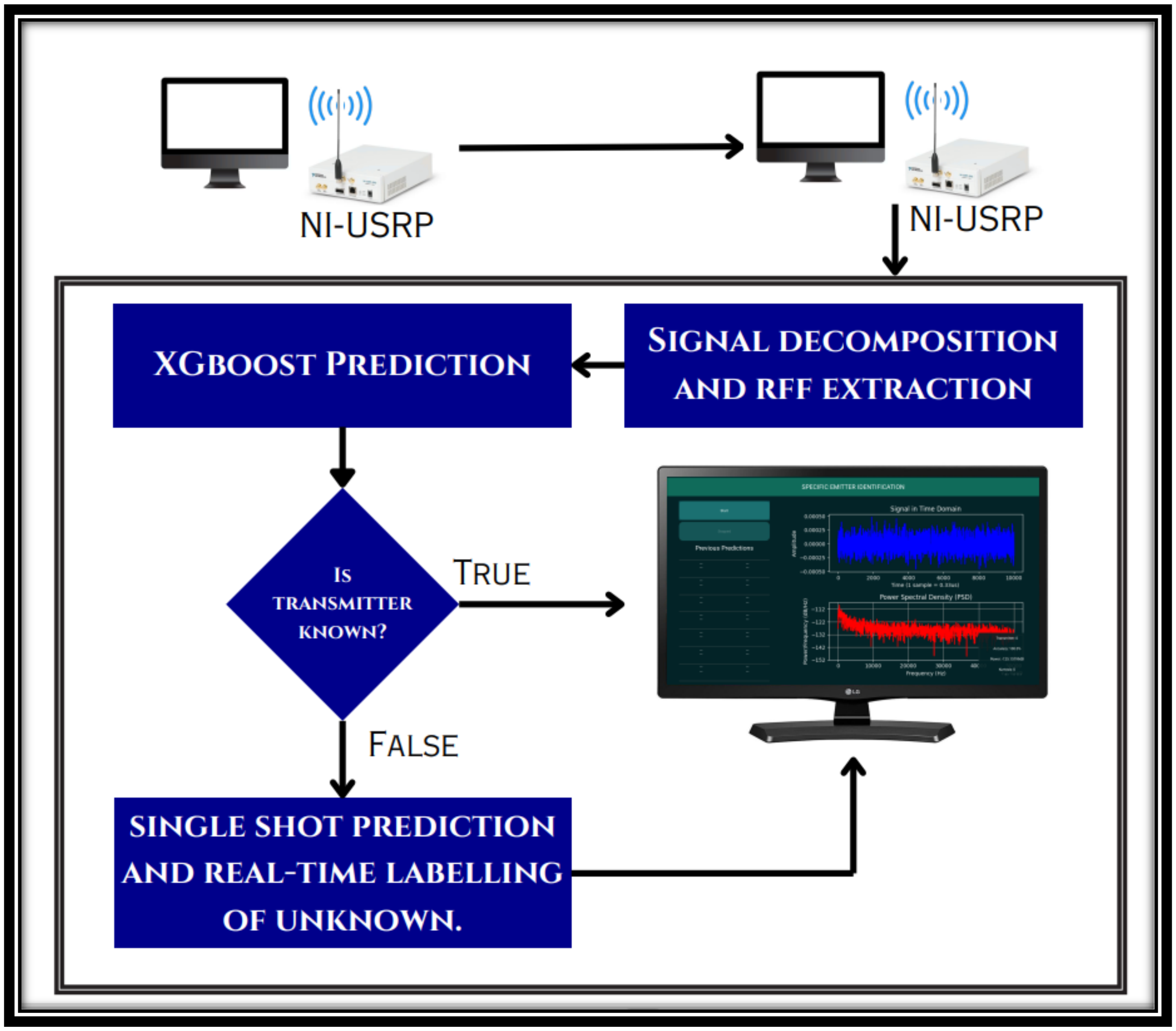
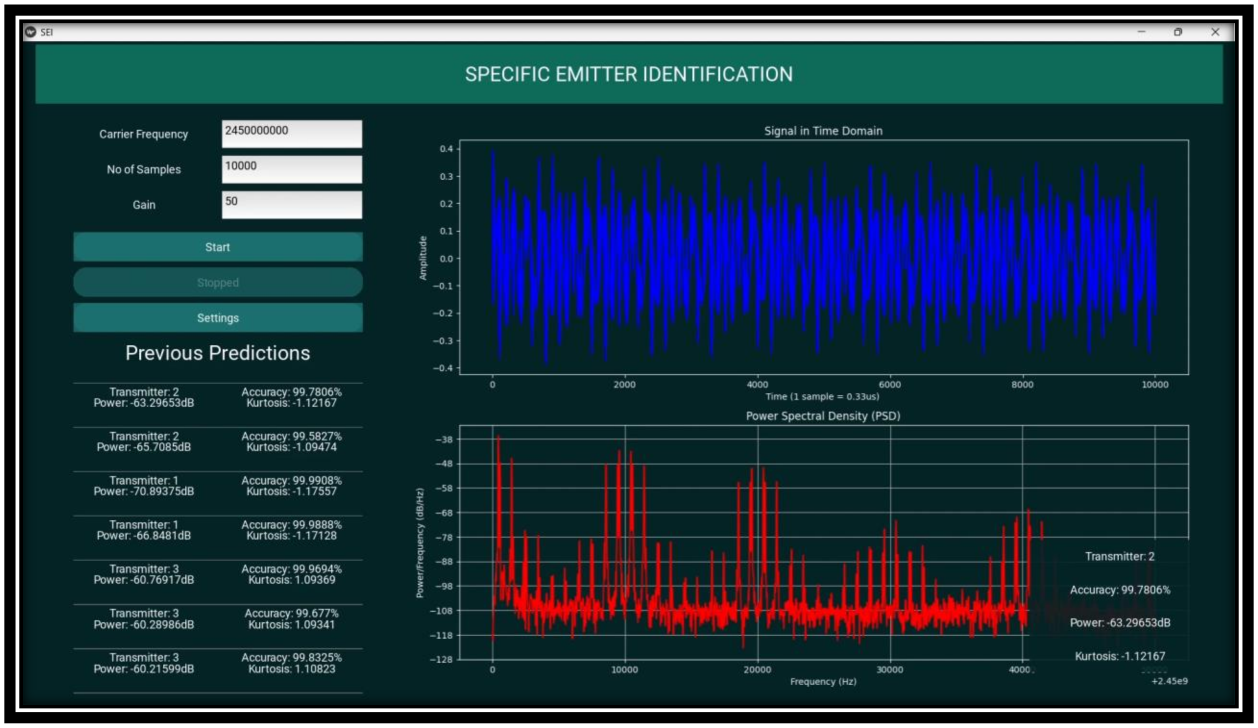

NI-USRP
NI-USRP
XGBOOST PREDICTION
SIGNAL DECOMPOSITION AND RFF EXTRACTION
IS TRANSMITTER KNOWN?
TRUE
FALSE
SINGLE SHOT PREDICTION AND REAL-TIME LABELLING OF UNKNOWN.
SPECIFIC EMITTER IDENTIFICATION
Signal in Time Domain
Power Spectral Density (PSD)

Figure 25. Specific emitter identification algorithm flow diagram.

7.2.2. Graphical user interface:

Now we will discuss the graphical user interface of SEI application. The SEI application's GUI was designed in Figma and developed using python's kivy framework. The GUI consists of input fields for RTLSDR configuration and to control the center frequency, number of samples, and the gain of RTLSDR, along with start and stop buttons. The GUI developed for USRP as a receiver does not have these input fields and only contains a start and stop button.
In addition to this, there is a section to display the previous predictions and their properties including predicted transmitter, accuracy/distance, kurtosis, and power. One the right, there are two graphs, one displays the signal in time domain and the other displays the power spectral density (PSD) of the signal. Additionally, there is a block that displays the current prediction.



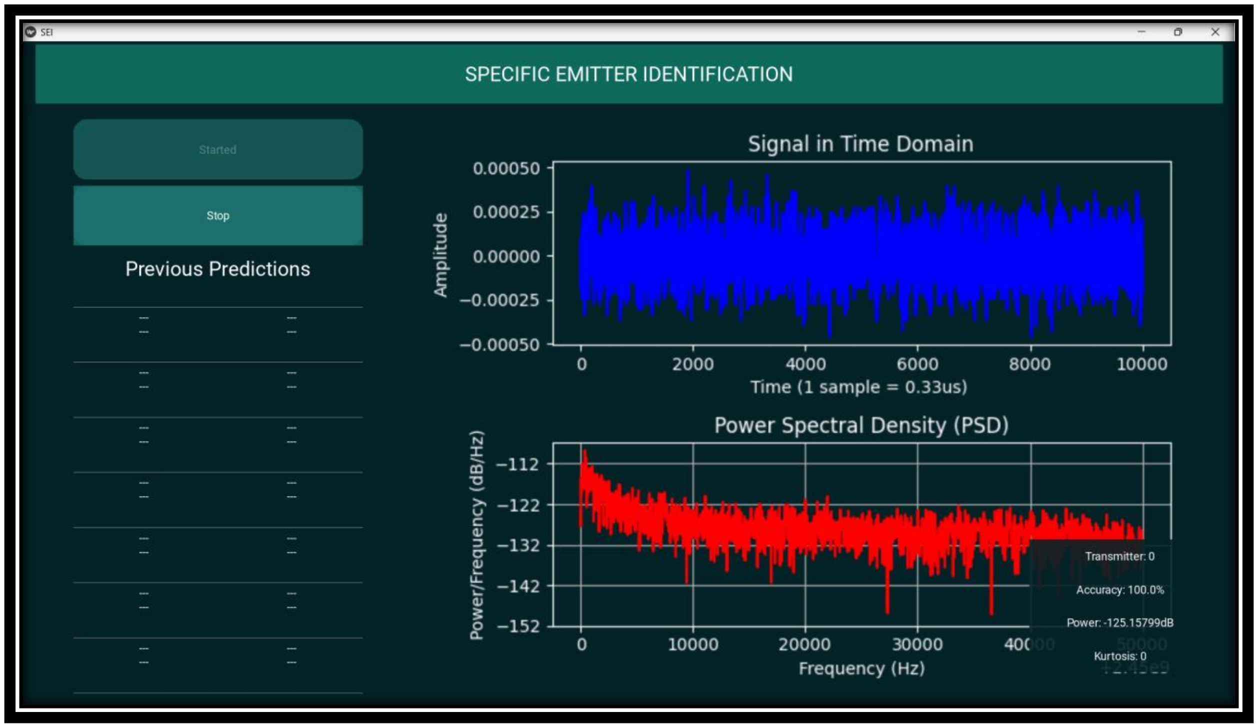
Carrier Frequency 2450000000
No of Samples 10000
Gain 50

Previous Predictions

Transmitter: 2 Power: -63.29653dB	Accuracy: 99.7806% Kurtosis: -1.12167
Transmitter: 2 Power: -65.7085dB	Accuracy: 99.5827% Kurtosis: -1.09474
Transmitter: 1 Power: -70.89375dB	Accuracy: 99.9908% Kurtosis: -1.17557
Transmitter: 1 Power: -66.8481dB	Accuracy: 99.9888% Kurtosis: -1.17128
Transmitter: 3 Power: -60.76917dB	Accuracy: 99.9694% Kurtosis: 1.09369
Transmitter: 3 Power: -60.28986dB	Accuracy: 99.677% Kurtosis: 1.09341
Transmitter: 3 Power: -60.21599dB	Accuracy: 99.8325% Kurtosis: 1.10823

Transmitter: 2
Accuracy: 99.7806%
Power: -63.29653dB
Kurtosis: -1.12167

Figure 26. GUI of SEI application when receiver is RTLSDR.



Previous Predictions

-- --	-- --
-- --	-- --
-- --	-- --
-- --	-- --
-- --	-- --
-- --	-- --
-- --	-- --

Transmitter: 0
Accuracy: 100.0%
Power: -125.15799dB
Kurtosis: 0

Figure 27. GUI of SEI application when receiver is USRP2901.

7.2.3. Noise floor:

To make sure the algorithm does not classify noise as a transmitter, a noise floor of -120dB is set. This value was selected based on experimentation and if the power of received signal is lower than -120dB, it is labelled as 0.

CHAPTER 8 – CONCLUSION AND FUTURE WORK

In this final chapter, we will conclude our report and discuss some ideas that can be implemented on this project in future.

8.1. Conclusion:

Specific Emitter Identification (SEI) involves identifying transmitters based on their unique RF fingerprints unintentionally modulated into their signals. This project introduces a novel approach by combining manual feature-based methods and deep learning-based methods to enhance SEI. The received signal is processed to extract Modified RF-DNA using Variational Mode Decomposition (VMD) and RF-DNA, achieving a 99% accuracy in transmitter prediction with an XGBoost classifier. The classifier also identifies unknown transmitters not among the known four. If an unknown transmitter is detected, the signal is forwarded to a Siamese network, which uses a 2-channel Convolutional Neural Network (CNN) with bimodal feature fusion to extract RF fingerprints from the time-frequency spectrogram and Modified RF-DNA. This model compares the unknown signal with known transmitter signals and achieves 90% accuracy in determining whether they are from the same transmitter. A complete SEI application has been developed for this process, capable of running on a laptop using either a USRP2901 or RTLSDR as the receiver platform.

8.2. Future work:

Although this project completes its requirements, there are many things that can be added to this project. Some of them are:

- This project can be implemented on a high-performance minicomputer to make it robust and available for a small device.
- The complete algorithm can be improved by improving the RFF extraction from the signal and making the prediction better with less tendency towards overfitting.
- The models presented in this project can be trained on huge amounts of data consisting of hundreds and thousands of transmitters for better and more accurate predictions and making the models ready for unseen scenarios.
- The computational complexity of the algorithm can be decreased.
- The models can be trained to detect multiple transmitters at once by utilizing the time division multiplexing (TDM) and frequency division multiplexing (FDM).

ANNEXURE

SEI AS A COMPLEX ENGINEERING PROBLEM.

Specific emitter identification is a complex engineering problem that has the following characteristics:

WP1 – Depth of knowledge required.

Advanced engineering knowledge at the level of WK3, WK4, WK5, WK6, and WK8 is required for specific emitter identification as this level of understanding is necessary to apply basics of signal processing, electromagnetic theory, machine learning, deep learning, and statistics. It is very important for us as engineers to understand the characteristics of electromagnetic waves, non-linearities of devices, and the theory behind radio frequency fingerprints.

WP2: Range of conflicting requirements.

SEI has a wide range of technical, engineering, and other issues that we faced and had to overcome.

- SEI requires high precision calculations and identification versus the processing power available in real-time scenarios.
- SEI also needs highly accurate data in the minimum amount of time, hence requires high speed.
- There are many environmental constraints that need to be addressed for specific emitter identification including the channel noise.

WP3: Depth of analysis required.

Because of no specific definition of what an RF fingerprint is, the problem has no obvious solution and requires abstract thinking and an original approach. We developed novel algorithms and techniques for signal feature extraction and classification.

WP6: Extent of stakeholder involvement and level of conflicting requirements:

SEI involves many groups of stakeholders with varying needs. Some of them are:

- Military and defense industries require high precision and secure identification systems to ensure national security.
- Many commercial entities require specific emitter identification for managing and identifying signals in crowded spectra.
- Many regulatory bodies can use this technology for spectrum management and legal constraints.

	WP1						WP2	WP3	WP4	WP5	WP6	WP7
	WK3	WK4	WK5	WK6	WK7	WK8						
PLO1 (WA1)	X											
PLO2 (WA2)		X						X			X	
PLO3 (WA3)			X									
PLO4 (WA4)						X	X					
PLO5 (WA5)				X								
PLO6 (WA6)												
PLO7 (WA7)												
PLO8 (WA8)												

REFERENCES

[1] I. S. Mohamed, Y. Dalveren and A. Kara, “Performance Assessment of Transient Signal Detection Methods and Superiority of Energy Criterion (EC) Method,” in IEEE Access, vol. 8, pp. 115613-115620, 2020.Z.

[2] U. Satija, N. Trivedi, G. Biswal and B. Ramkumar, “Specific Emitter Identification Based on Variational Mode Decomposition and Spectral Features in Single Hop and Relaying Scenarios,” in IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, pp. 581-591, March 2019

[3] WEI J Y, YU L, ZHU L, et al. RF fingerprint extraction method based on CEEMDAN and multidomain joint Entropy. Wireless Communications and Mobile Computing, 2022, 2022: 5326892

[4] SATIJA U, TRIVEDI N, BISWAL G, et al. Specific emitter identification based on variational mode decomposition and spectral features in single hop and relaying scenarios. IEEE Trans. on Information Forensics and Security, 2018, 14(3): 581–591

[5] M. Lukacs, P. Collins, and M. Temple, “Classification performance using ’rf-dna’ fingerprinting of ultra-wideband noise waveforms,” Electronics Letters, vol. 51, no. 10, pp. 787–789, 2015.

[6] L. Yang, P. Zhen, J. Wang, J. Zhang and D. Guo, “Radar Emitter Recognition Method Based on Bispectrum and Improved Multichannel Convolutional Neural Network,” 2021 6th International Conference on Communication, Image and Signal Processing (CCISP), 2021, pp. 321-328.

[7] Y. Yuan, Z. Huang, H. Wu, and X. Wang, “Specific emitter identification based on Hilbert–Huang transform-based time–frequency–energy distribution features,” IET Commun., vol. 8, no. 13, pp. 2404–2412, Sep. 2014.

[8] R. Kabiri and S. V. Shojaeddini, “specific emitter identification novel method based on time-frequency features,” 2015.

[9] X. Wang, G. Huang, Z. Zhou and J. Gao, “Radar emitter recognition based on the short time fourier transform and convolutional neural networks,” 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), 2017, pp. 1-5.

[10] C. Bertoncini, K. Rudd, B. Nousain, and M. Hinders, “Wavelet fingerprinting of radio-frequency identification (RFID) tags,” IEEE Trans. Ind. Electron, vol. 59, no. 12, pp. 4843–4850, Dec. 2012.

[11] K. Dragomiretskiy and D. Zosso, “Variational Mode Decomposition,” in IEEE Transactions on Signal Processing, vol. 62, no. 3, pp. 531- 544, Feb.1, 2014

[12] Zhao Yurui, Wang Xiang, Sun Liting, and Huang Zhitao, “Specific emitter identification based on frequency and amplitude of the signal kurtosis” Journal of Systems Engineering and Electronics, Vol. PP, No. 99, April 2023, pp. 1 – 11.

[13] Yipeng Zhou, Rui Zhou, Hailong Wang, Chunyu Wang, Xiaofeng Wang, and Yan Yu, “A Specific Emitter Identification Method Based on RF-DNA and XGBoost” 2022 7th International Conference on Intelligent Computing and Signal Processing (ICSP).

[14] Jinling Su, Heng Liu, Lui Yang, "Specific Emitter Identification Based on Variational Mode Decomposition and Bimodal Feature Fusion" 2023 IEEE 3rd International Conference on power, Electronics and Computer Applications (ICPECA).