Machine Learning LTE RF Fingerprinter

Bradford Williams, G01293202 - Chris Limson, G01360952

GMU CYSE 640 Wireless Network Security

Fall 2024 - Moinul Hossain, PhD

Abstract

This is a study of creating a proof-of-concept system that recognizes RF fingerprints (RFF) extracted from LTE user equipment (UE) uplinks with a web service. It required researching properties of the LTE standard, and generating random UE uplink signals each with a sample set of minor variances exhibiting a characteristic RFF. A machine learning neural network was built from the dataset of UE signals and their variances, into which a simulated test target LTE waveform with RFF would be applied. From this simulation, attributes for a feasible fingerprint were determined, to then build a system to test and demonstrate. The service is able to accept a submitted LTE compliant uplink signal with an unknown RFF embedded, then compare its likeness to waveforms already in the dataset. The system decides whether to update its dataset or flag the submitted signal for further examination, depending on if the RFF in question matches the known fingerprint of that device or a different device. Finally, these decisions will be analyzed and discussed.

1. Introduction

The study has been broken down into outlined summaries of the theory, technology, architecture, experimentation, results, analysis and discussion.

1.1 Overview

When a wireless device attempts to connect to a base station, it needs to be authenticated. LTE increases the complexity for authentication due to necessary standards. There are an increasing number of ways to spoof device MAC address, IP address or other identifying information. With radio frequency (RF) fingerprinting, a unique minor signal impairment emitted by the device exists due to imperfections in the hardware when it is created.

A machine learning neural network can be used to recognize patterns in the signal's waveform and determine how closely the signal matches another in a maintained database. The specific device attempting to connect to the base station can therefore be identified by authenticating the RF fingerprint quickly and securely. No method to spoof an RF fingerprint is currently known to exist, thus allowing each device to be cataloged and classified off this attribute.

The challenge this study addresses is the standardization of a central system that collects LTE reference measurement channel (RMC) waveforms [3] affected by RF fingerprints in a common format and the UE's unique identification. This continually updated collection of waveforms and their devices allows for a more dynamic, robust and efficient method of recognition.

1.2. Objective

The essential solution is the deep learning algorithm implemented. It takes a waveform, and based on the fingerprints supplied during the training process, picks which already classified waveform it resembles most. By analyzing the result, the algorithm can return a message of rejection if the fingerprint does not match any similarly in the database. The deep learning neural network involves applying a large number of simulated devices to the database. However, these simulated devices can be interchanged with actual devices if available. This dataset improves the identification of devices as the number of devices and number of sampled waveforms per device is added. The neural network can identify which device is attempting to connect with higher accuracy if the number of samples for its waveform is increased. The proposed solution is further developed into a system architecture for accepted standardization, addressing the challenge of [1] where the authors suggest building a dataset for classification accuracy.

Since this will be a simulated model, properties will be updated for each device manually to see how the deep learning model reacts to the new information. An additional interest will be flagging malicious devices by analyzing their fingerprints, as all other higher-level logic and MAC addresses can be altered. This proposes a solution where the base station identifies a connecting device based solely off their RF fingerprint. This is due to the increasing commonality of logic, IP addresses and MAC addresses being able to be altered.

A large number of simulated devices to the entire system's algorithm will be applied, showing how the built dataset improves identification, as more devices contribute. This includes having more than one device try authenticating against the base station based off of its RF fingerprint. Four cases for authentication and experimentations are to have everything cataloged by the base station for a device incorrect except the fingerprint, to have everything cataloged correct and the fingerprint incorrect and to have everything either all correct or all incorrect for authentication.

2. Literature Review

With the introduction of radio fingerprinting in [1], machine learning models and software defined radio frequencies were able to give each device, although similar in nature, a specific RF fingerprint. It is mentioned in [1] that “No higher level decoding, feature engineering, or protocol knowledge is needed, further mitigating challenges of ID spoofing and coexistence of multiple protocols in a shared spectrum” which would allow the protocols to be further strengthened in other directions.

The authors of [1] also reaffirm that because the procedure translates complex data points of I/Q samples to simply a collection of real unit pairs, making a two-dimensional array of 2 by sample-length, convolutional neural networks experimentally demonstrate near-perfect radio identification performance. Instead of having overlapping protocols for authentication, there is the possibility of a single authentication method based on the RF fingerprint, further reducing the overhead cost of authentication in each protocol. One significant finding in this paper is that they have tested it at specific directions, notably ranging from 2 feet away to 50 feet away. It is also mentioned that the identification remains accurate until it drops off after 34 feet.

In [1], the authors use RF fingerprinting on a variety of devices from different manufacturers and find that detecting and authenticating a user based on their RF fingerprint is a valid control. With the authors implementing their scenario in 4G-LTE, it gives precedence to this study's development also centered in LTE. There is a brief mention in the future work of the paper about how degradation can occur over the course of the hardware's lifetime. This could potentially cause it to no longer have the same fingerprint, or possibly emulate a similar fingerprint of another device. Additionally, there is no note of intentional disruption of the hardware, potentially changing the fingerprint of the device.

3. Methodology

During research and development, an AWS EC2 Ubuntu 24.04 LTS host with 4 vCPUs and 16 GiB memory of type t3.xlarge was instantiated so coding from a common baseline could be done collaboratively and economically. MATLAB and its LTE Toolbox were installed, as well as TensorFlow and Flask Python libraries for the application's system.

MATLAB was used only to generate LTE waveforms for simulation and test [7]. While MATLAB also has functions for machine learning, Python instead was employed to build the neural network and test cases against it due to TensorFlow's open-source flexibility [4]. The software library was used to compute the convolutional neural network [2]. The Python components were then integrated into a web service with Flask [5] that vendors or requestors would be able to use for authentication, but also to contribute while doing so.

3.1. Testing Predictions from Generated Dataset

LTE compliant modulated uplink waveforms which include RF fingerprint (RFF) are initiated by user equipment (UE) device connection to Evolved Node Base (eNB) base station requestors. Once the service receives it, TensorFlow will determine which waveform the pattern resembles most and its probability, and respond back to the requestor its recommendation. Fig. 1 illustrates this process.

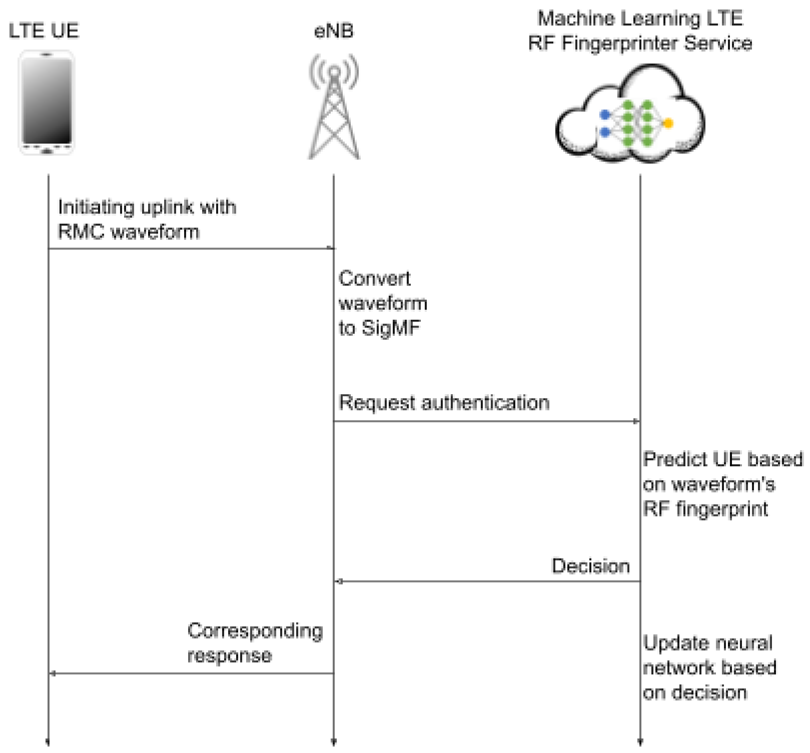

LTE UE
eNB
Machine Learning LTE RF Fingerprinter Service
Initiating uplink with RMC waveform
Convert waveform to SigMF
Request authentication
Predict UE based on waveform's RF fingerprint
Decision
Corresponding response
Update neural network based on decision

Fig. 1. RF fingerprinter sequence diagram.

3.2. Four Cases of Predictions

If the predicted device MAC address matches the submitted MAC address which the device claims to have, then either of two more conditions are considered. Their RF fingerprints might still only match by less than 50%, in which case the submitted waveform is added to the dataset which is updated and retrained. Otherwise, at least 50% checks out and the match is strong enough not to add evidence.

If the predicted device MAC address does not match the submitted MAC address which the device claims to have, then two additional conditions may occur. If their RF fingerprints match by less than 80%, then the submitted waveform is added to the dataset which is updated and retrained, strengthening their distinction. Otherwise, at least 80% signifies a conflicting MAC address identification but their waveforms are strongly similar. This outcome strongly suggests that the submitted device is attempting to spoof a different device's MAC address, or that the device already in the dataset has a spoofed MAC address itself. Either case, the new waveform is added to a flagged directory indicating the conflicting dataset waveform, and no retraining is performed.

4. Implementation

The developed software should be broken down into two main parts. All code is available in the repository https://chrimson.github.io/ML-LTE-RFF

The first part in subsection 4.1 is simply a set of tools to investigate the behavior of RF fingerprints added to LTE RMC waveforms and to generate a neural network with a large enough sample space for simulating a query of an arbitrary waveform. The neural network was tested with samples so that parameters for the best waveform demonstrations could be determined.

The second part in subsection 4.2 is the establishment of a centralized network server to which a waveform with an embedded RF fingerprint, using the previously obtained parameters, may be submitted in a standard format. The system will then return its perception of the waveform's results while updating itself with submitted query.

4.1. Research RF Fingerprint

Parameters for how strong an RF fingerprint and how repetitive it would be had to be determined by trial and error, to yield the best apparent demonstration of the presented principles. The general process is shown in Fig. 2 and described in the following subsections.

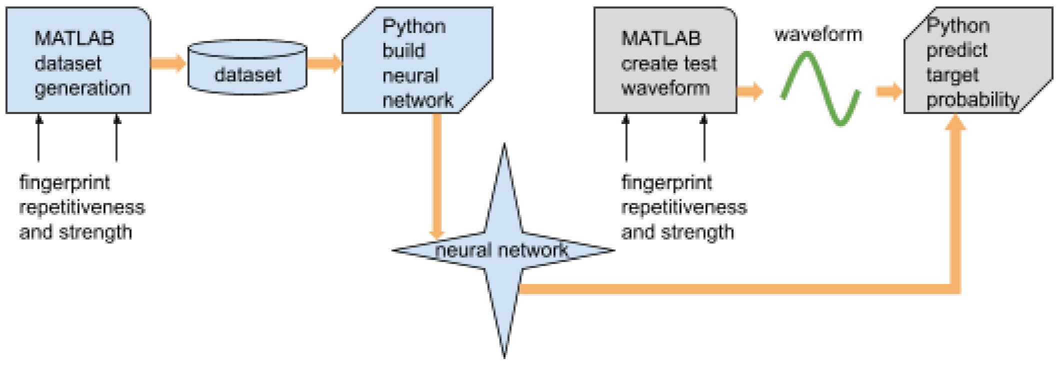
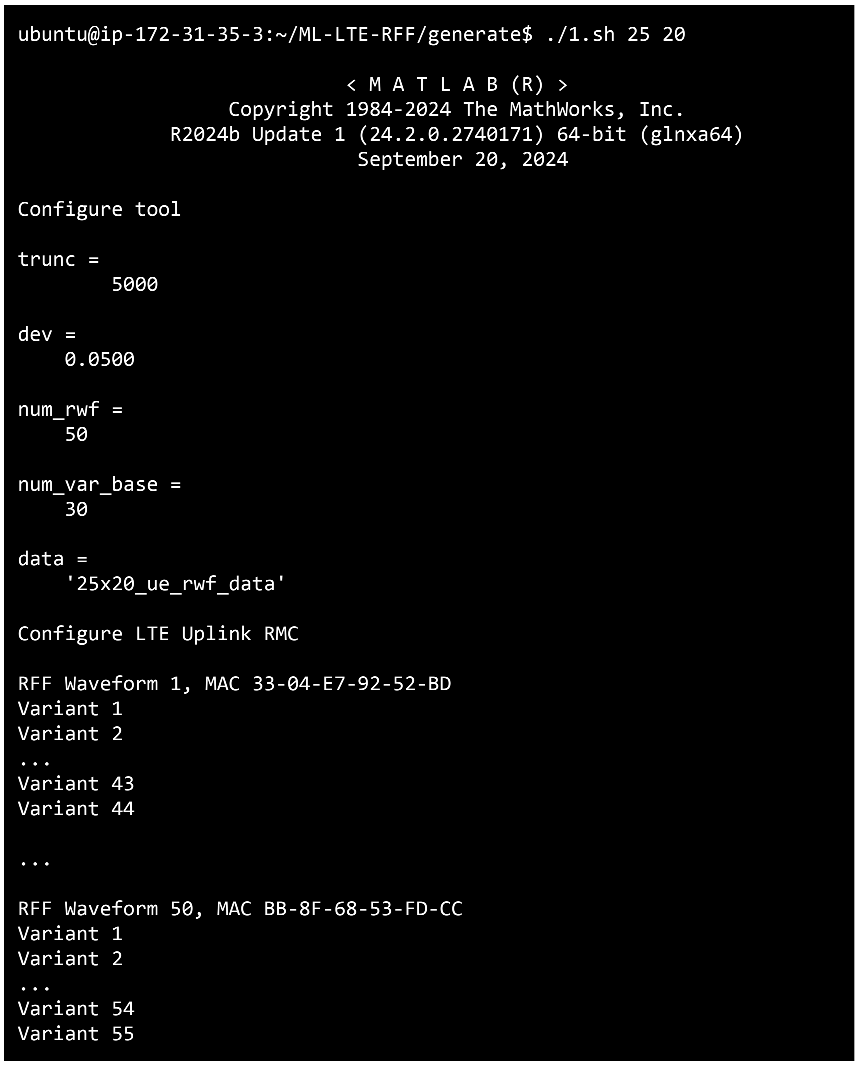

MATLAB dataset generation
dataset
Python build neural network
MATLAB create test waveform
waveform
Python predict target probability
fingerprint repetitiveness and strength
neural network
fingerprint repetitiveness and strength

Fig. 2. Tools for investigation.

4.1.1. Generation of Sample Dataset

A MATLAB CLI script was created to generate LTE compliant modulated uplink waveforms which will include RFF to simulate device UE connection to eNB base stations. Simulated RFF are generated by random parameters (that are likely unique) of a sine and cosine combination. Their effect on the base RMC waveform generated by LTE ToolBox is then the RFF Waveforms (RWF). The output of this code is a directory structure and a metadata file. Inside the directory structure, is each created MAC address along with their corresponding subdirectory. Inside the subdirectory there is the individual information for the generated variations of each of the waveforms. Further inside the subdirectory of each MAC address, inside each variation, is the complex number representing the waveform. Each variation includes 5000 plotted points which correspond to the shape of the variation of the MAC address. Inside the metadata file, are the parameters used to generate the fingerprints, corresponding to each of the MAC addresses. They will be illustrated on the MATLAB GUI in Section 5 by uncommenting plot, timescopes and spectrum analyzers with and without the RFFs applied. Fig. 3 shows an abridged run of generating 50 waveforms, each with a random number of variances between 30 and 60 deviating by 5%.

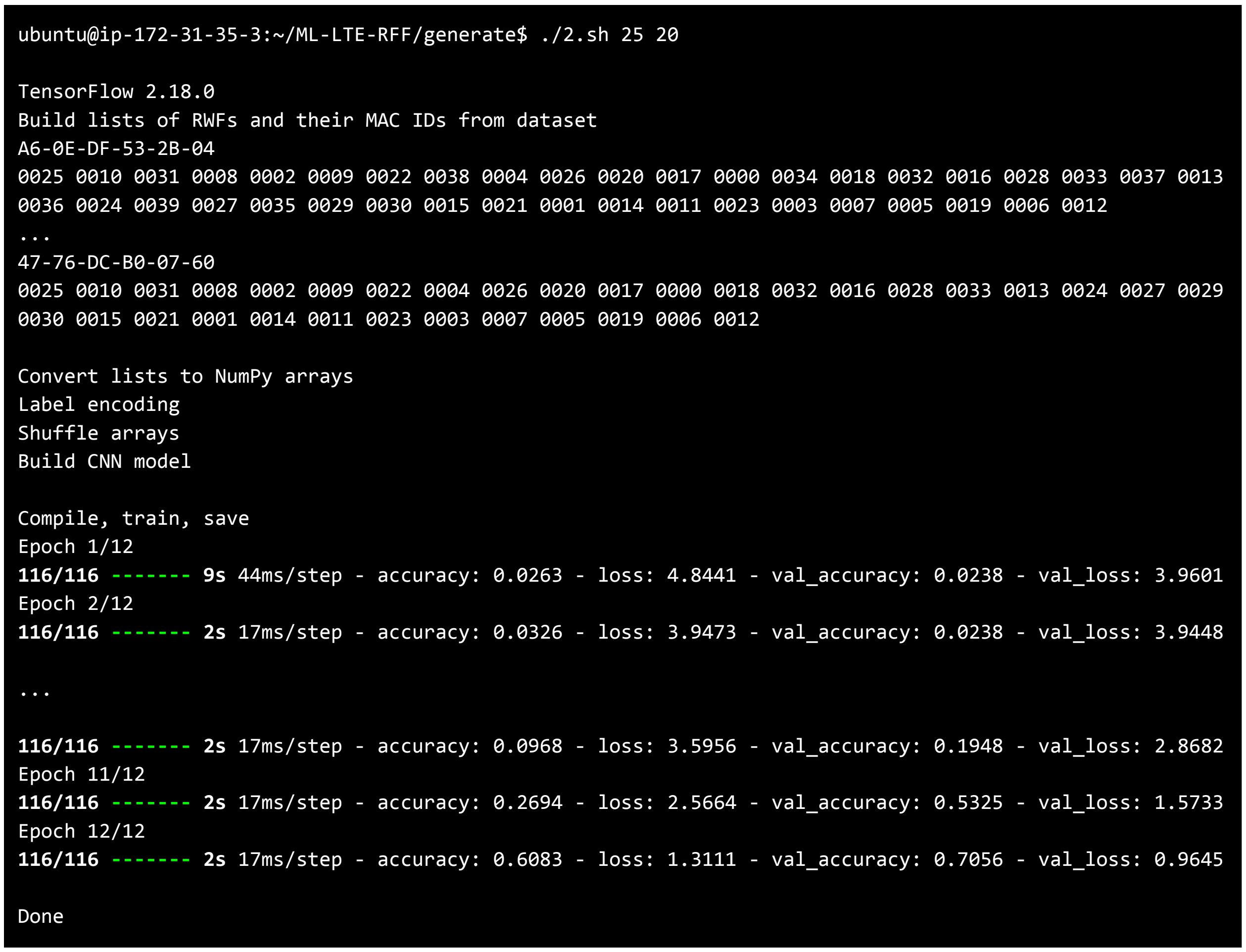


```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF/generate$ ./1.sh 25 20

                            < M A T L A B (R) >
                  Copyright 1984-2024 The MathWorks, Inc.
             R2024b Update 1 (24.2.0.2740171) 64-bit (glnxa64)
                            September 20, 2024

Configure tool

trunc =
        5000

dev =
    0.0500

num_rwf =
    50

num_var_base =
    30

data =
    '25x20_ue_rwf_data'

Configure LTE Uplink RMC

RFF Waveform 1, MAC 33-04-E7-92-52-BD
Variant 1
Variant 2
...
Variant 43
Variant 44

...

RFF Waveform 50, MAC BB-8F-68-53-FD-CC
Variant 1
Variant 2
...
Variant 54
Variant 55
```

Fig. 3. Waveforms of devices with RF fingerprint variants dataset generation.

4.1.2. Building Neural Network

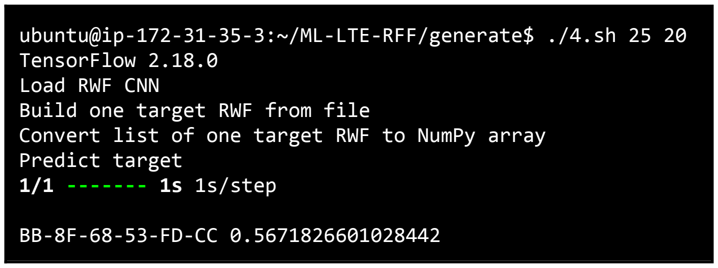
In Fig. 4, Python reads the stored dataset into NumPy memory arrays, so that TensorFlow can be used to create a convolutional neural network by applying the RWFs to train it. The convolutional neural network (CNN) takes an input of the shape (trunc, 2, 1). This means it takes the shape of (5000, 2, 1). This corresponds to the 5000 points for the variation of the mac address, truncated from 307200. Then two channels for both the real and imaginary parts of the waveform. The one at the end corresponds to a single depth channel. The CNN takes in each variation of the fingerprinted waveform for training, and assigns each variation to their corresponding MAC address to try and find similarities. The output of this script is a processed data file, containing the waveforms stored as complex numbers. Additionally, there is an output of a trained CNN model, that can predict the MAC ID address of a new waveform that is given. The format of the waveform must be 5000, 2, 1, as required earlier. The output of the CNN will be which MAC ID associates closest to it.



```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF/generate$ ./2.sh 25 20

TensorFlow 2.18.0
Build lists of RWFs and their MAC IDs from dataset
A6-0E-DF-53-2B-04
0025 0010 0031 0008 0002 0009 0022 0038 0004 0026 0020 0017 0000 0034 0018 0032 0016 0028 0033 0037 0013
0036 0024 0039 0027 0035 0029 0030 0015 0021 0001 0014 0011 0023 0003 0007 0005 0019 0006 0012
...
47-76-DC-B0-07-60
0025 0010 0031 0008 0002 0009 0022 0004 0026 0020 0017 0000 0018 0032 0016 0028 0033 0013 0024 0027 0029
0030 0015 0021 0001 0014 0011 0023 0003 0007 0005 0019 0006 0012

Convert lists to NumPy arrays
Label encoding
Shuffle arrays
Build CNN model

Compile, train, save
Epoch 1/12
116/116 ------- 9s 44ms/step - accuracy: 0.0263 - loss: 4.8441 - val_accuracy: 0.0238 - val_loss: 3.9601
Epoch 2/12
116/116 ------- 2s 17ms/step - accuracy: 0.0326 - loss: 3.9473 - val_accuracy: 0.0238 - val_loss: 3.9448

...

116/116 ------- 2s 17ms/step - accuracy: 0.0968 - loss: 3.5956 - val_accuracy: 0.1948 - val_loss: 2.8682
Epoch 11/12
116/116 ------- 2s 17ms/step - accuracy: 0.2694 - loss: 2.5664 - val_accuracy: 0.5325 - val_loss: 1.5733
Epoch 12/12
116/116 ------- 2s 17ms/step - accuracy: 0.6083 - loss: 1.3111 - val_accuracy: 0.7056 - val_loss: 0.9645

Done
```

Fig. 4. Building neural network from generated dataset.

4.1.3. Testing Sample Waveform Prediction



```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF/generate$ ./4.sh 25 20
TensorFlow 2.18.0
Load RWF CNN
Build one target RWF from file
Convert list of one target RWF to NumPy array
Predict target
1/1 ------- 1s 1s/step

BB-8F-68-53-FD-CC 0.5671826601028442
```

Fig. 5. Predicting which dataset entry matches the submitted waveform and its probability.

A set of RWF characteristic parameters may be chosen as variables to be passed into the second MATLAB script. Then step #3 (not shown) will generate a single target variant RWF with marginal deviations. The parameters for step #3 can be acquired from the created information found in part 1, due to the simulated environment. The parameters are provided for further testing.

In Fig. 5, step #4 shows the model will predict the target RWF that most closely matches RFF, decode its associated label and find the corresponding probability.

4.2. Integrating Components for Service System

The code from the Python models above for building and training the TensorFlow neural network, predicting targets with it and additional supplementary packages is needed for the service.

4.2.1. Flask Web Service REST API

The software was integrated into Flask for Python to make an Internet service using port 64024. A requestor would submit their LTE RMC waveform affected by RF fingerprint using curl for Linux. After sending the file and waiting for the system to process its waveform in the neural network and analyzing its prediction, the requestor is returned a message about the result.

4.2.2. Common SigMF Formatting

A standard signal waveform format accepted by GNURadio known as Signal Metadata Format (SigMF) is utilized so that requesters can follow a method commonly agreed upon [8]. This simply involved inspecting the specification and replicating saved NumPy files. It allowed a slight reduction of just a few lines of code and 10% of dataset storage, as complex numbers could now be saved to binary instead of ASCII files. This was a demonstration of the validity of the principle.

5. Results and Analysis

Like in the previous section of implementation, outcomes have been broken down into two main corresponding parts. The first part in subsection 5.1 describes the behavior of RF fingerprints added to LTE RMC waveforms and the parameters with which to generate a neural network.

The second part in subsection 5.2 demonstrates operation of the network service and its possible responses. Data and screenshots from both have been tabulated and illustrated.

5.1. Behavior and Properties of RF Fingerprints

MATLAB was used to generate LTE compliant RMCs and simulated RF fingerprints. Python then took the generated waveforms and applied them for training and testing of behavior.

5.1.1. Basic Attributes

An RF fingerprint was generated from random factors of amplitude and phase shift applied to a combination of sine and cosine functions. These factors were slightly adjusted to generate variants of the same waveform but with the same primary characteristics, as seen in Fig. 6.

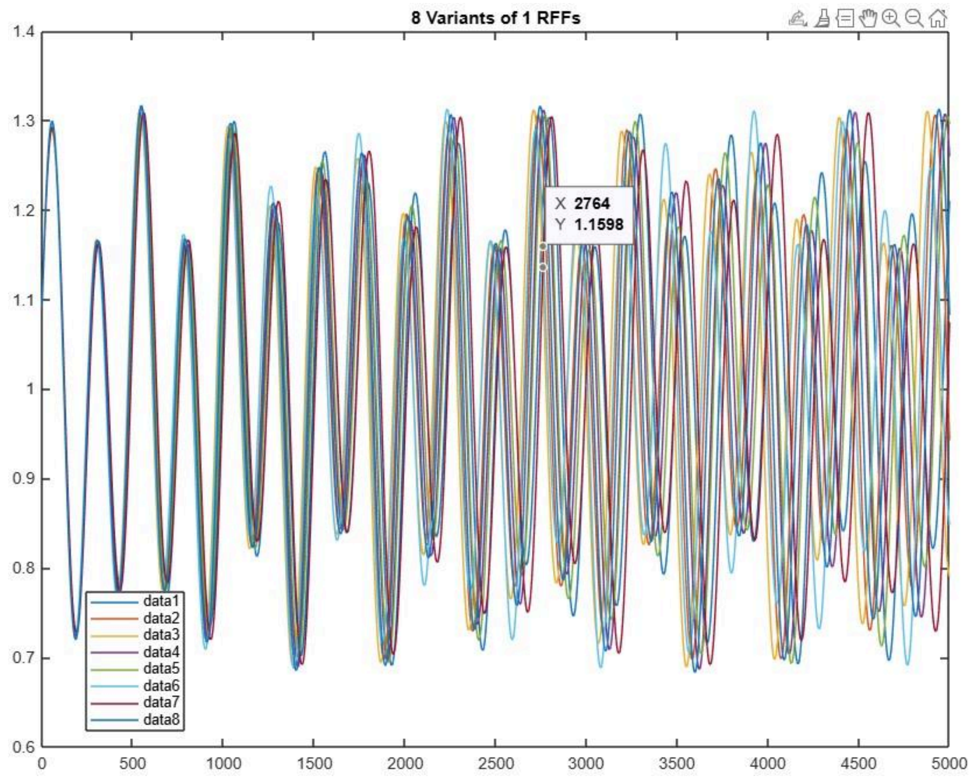
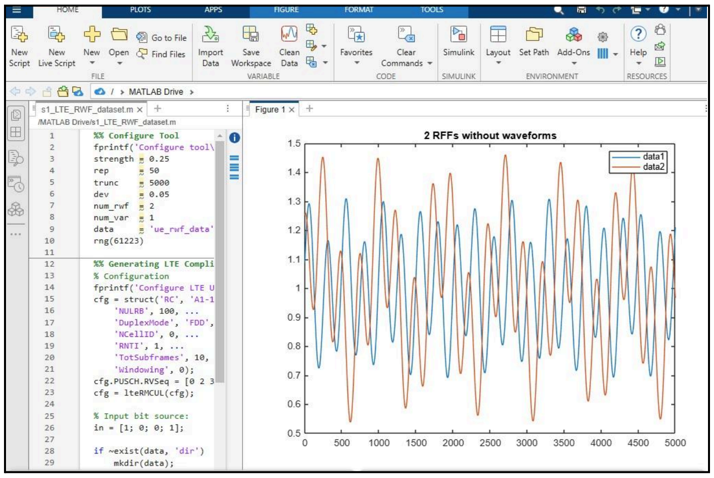


Fig. 6. Eight variants by 5% of one RF fingerprint.

5.1.2. Fingerprint Comparison

Two RFFs with distinct characteristics are shown in Fig. 7.



```
%% Configure Tool
fprintf('Configure tool\
strength = 0.25
rep      = 50
trunc    = 5000
dev      = 0.05
num_rwf  = 2
num_var  = 1
data     = 'ue_rwf_data'
rng(61223)

%% Generating LTE Compli
% Configuration
fprintf('Configure LTE U
cfg = struct('RC', 'A1-1
    'NULRB', 100, ...
    'DuplexMode', 'FDD',
    'NCellID', 0, ...
    'RNTI', 1, ...
    'TotSubframes', 10,
    'Windowing', 0);
cfg.PUSCH.RVSeq = [0 2 3
cfg = lteRMCUL(cfg);

% Input bit source:
in = [1; 0; 0; 1];

if ~exist(data, 'dir')
    mkdir(data);
```

Fig. 7. Two different RF fingerprints without waveforms.

5.1.3. Effect of Fingerprint

A waveform without an RFF on the top, and the same waveform affected by an exaggerated RFF on the bottom are in Fig. 8. Their corresponding spectrum analyzers still match.

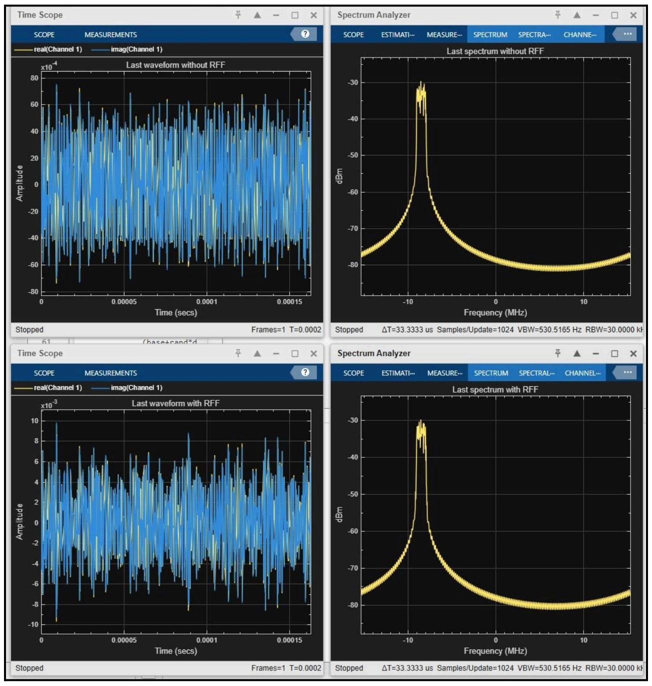
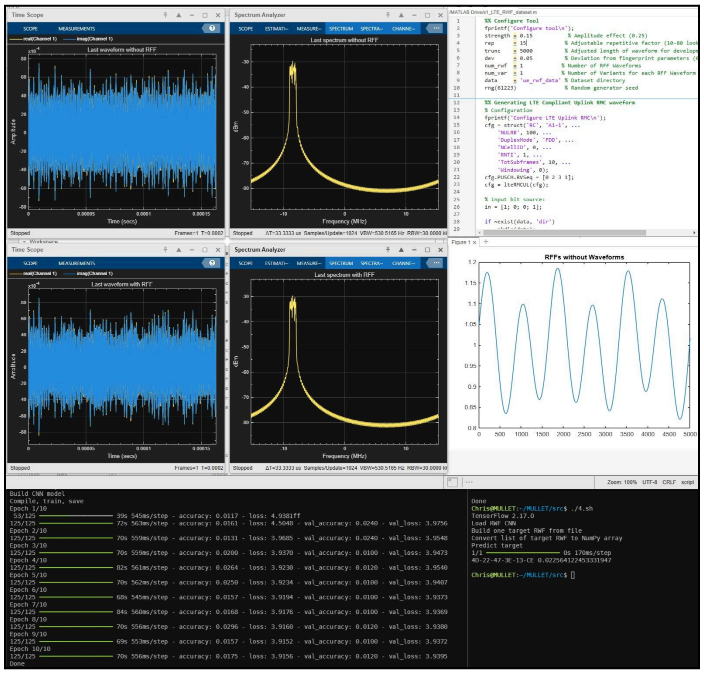


Fig. 8. Waveform with and without an applied RFF.

5.1.4. Neural Network Predictions

After the neural network was modeled and trained with a dataset of 50 LTE RMC waveforms affected by a characteristic RF fingerprint for each, and a random number of variants between 30 and 60 for each, following in Fig. 9, 10 and 11 are three predictions of which device's fingerprint most closely matches trial fingerprints. Fig. 12 summarizes all trials.



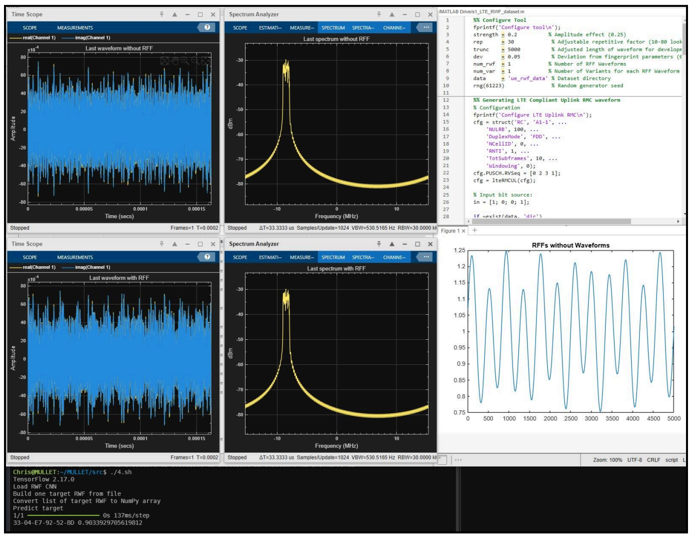
/MATLAB Drive/s1_LTE_RWF_dataset.m

```
%% Configure Tool
fprintf('Configure tool\n');
strength = 0.15          % Amplitude effect (0.25)
rep      = 15            % Adjustable repetitive factor (10-80 look
trunc    = 5000          % Adjusted length of waveform for developm
dev      = 0.05          % Deviation from fingerprint parameters (0
num_rwf  = 1            % Number of RFF Waveforms
num_var  = 1            % Number of Variants for each RFF Waveform
data     = 'ue_rwf_data' % Dataset directory
rng(61223)               % Random generator seed

%% Generating LTE Compliant Uplink RMC waveform
% Configuration
fprintf('Configure LTE Uplink RMC\n');
cfg = struct('RC', 'A1-1', ...
    'NULRB', 100, ...
    'DuplexMode', 'FDD', ...
    'NCellID', 0, ...
    'RNTI', 1, ...
    'TotSubframes', 10, ...
    'Windowing', 0);
cfg.PUSCH.RVSeq = [0 2 3 1];
cfg = lteRMCUL(cfg);

% Input bit source:
in = [1; 0; 0; 1];

if ~exist(data, 'dir')
```

```
Build CNN model
Compile, train, save
Epoch 1/10
 53/125 ━━━━━━━━━━━━━━━━━━━━ 39s 545ms/step - accuracy: 0.0117 - loss: 4.9381ff
125/125 ━━━━━━━━━━━━━━━━━━━━ 72s 563ms/step - accuracy: 0.0161 - loss: 4.5048 - val_accuracy: 0.0240 - val_loss: 3.9756
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 559ms/step - accuracy: 0.0131 - loss: 3.9685 - val_accuracy: 0.0240 - val_loss: 3.9548
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 559ms/step - accuracy: 0.0200 - loss: 3.9370 - val_accuracy: 0.0100 - val_loss: 3.9473
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 82s 561ms/step - accuracy: 0.0264 - loss: 3.9230 - val_accuracy: 0.0120 - val_loss: 3.9540
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 562ms/step - accuracy: 0.0250 - loss: 3.9234 - val_accuracy: 0.0100 - val_loss: 3.9407
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 68s 545ms/step - accuracy: 0.0157 - loss: 3.9194 - val_accuracy: 0.0100 - val_loss: 3.9373
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 84s 560ms/step - accuracy: 0.0168 - loss: 3.9176 - val_accuracy: 0.0100 - val_loss: 3.9369
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 556ms/step - accuracy: 0.0296 - loss: 3.9160 - val_accuracy: 0.0120 - val_loss: 3.9380
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 69s 553ms/step - accuracy: 0.0157 - loss: 3.9152 - val_accuracy: 0.0100 - val_loss: 3.9372
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 556ms/step - accuracy: 0.0175 - loss: 3.9156 - val_accuracy: 0.0120 - val_loss: 3.9395
Done
```

```
Done
Chris@MULLET:~/MULLET/src$ ./4.sh
TensorFlow 2.17.0
Load RWF CNN
Build one target RWF from file
Convert list of target RWF to NumPy array
Predict target
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step
4D-22-47-3E-13-CE 0.022564122453331947

Chris@MULLET:~/MULLET/src$ 
```

Fig. 9. Repetitive 15, Strength 15% results in confidence of 2% in the incorrectly guessed UE match.

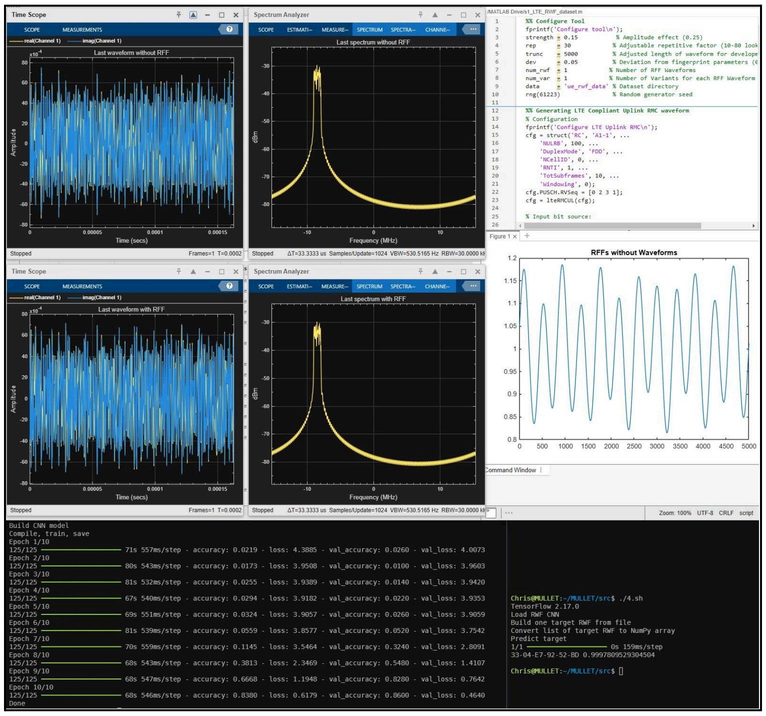


/MATLAB Drive/s1_LTE_RWF_dataset.m

```
%% Configure Tool
fprintf('Configure tool\n');
strength = 0.2          % Amplitude effect (0.25)
rep      = 30            % Adjustable repetitive factor (10-80 look
trunc    = 5000          % Adjusted length of waveform for developm
dev      = 0.05          % Deviation from fingerprint parameters (0
num_rwf  = 1            % Number of RFF Waveforms
num_var  = 1            % Number of Variants for each RFF Waveform
data     = 'ue_rwf_data' % Dataset directory
rng(61223)               % Random generator seed

%% Generating LTE Compliant Uplink RMC waveform
% Configuration
fprintf('Configure LTE Uplink RMC\n');
cfg = struct('RC', 'A1-1', ...
    'NULRB', 100, ...
    'DuplexMode', 'FDD', ...
    'NCellID', 0, ...
    'RNTI', 1, ...
    'TotSubframes', 10, ...
    'Windowing', 0);
cfg.PUSCH.RVSeq = [0 2 3 1];
cfg = lteRMCUL(cfg);

% Input bit source:
in = [1; 0; 0; 1];

if ~exist(data, 'dir')
```

```
Chris@MULLET:~/MULLET/src$ ./4.sh
TensorFlow 2.17.0
Load RWF CNN
Build one target RWF from file
Convert list of target RWF to NumPy array
Predict target
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
33-04-E7-92-52-BD 0.9033929705619812
```

Fig. 10. Repetitive 30, Strength 20% results in confidence of 90% in the correctly guessed UE match.



/MATLAB Drive/s1_LTE_RWF_dataset.m

```
%% Configure Tool
fprintf('Configure tool\n');
strength = 0.15           % Amplitude effect (0.25)
rep      = 30            % Adjustable repetitive factor (10-80 look
trunc    = 5000          % Adjusted length of waveform for developm
dev      = 0.05          % Deviation from fingerprint parameters (6
num_rwf  = 1            % Number of RFF Waveforms
num_var  = 1            % Number of Variants for each RFF Waveform
data     = 'ue_rwf_data' % Dataset directory
rng(61223)               % Random generator seed

%% Generating LTE Compliant Uplink RMC waveform
% Configuration
fprintf('Configure LTE Uplink RMC\n');
cfg = struct('RC', 'A1-1', ...
    'NULRB', 100, ...
    'DuplexMode', 'FDD', ...
    'NCellID', 0, ...
    'RNTI', 1, ...
    'TotSubframes', 10, ...
    'Windowing', 0);
cfg.PUSCH.RVSeq = [0 2 3 1];
cfg = lteRMCUL(cfg);

% Input bit source:
```

```
Build CNN model
Compile, train, save
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 71s 557ms/step - accuracy: 0.0219 - loss: 4.3885 - val_accuracy: 0.0260 - val_loss: 4.0073
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 80s 543ms/step - accuracy: 0.0173 - loss: 3.9508 - val_accuracy: 0.0100 - val_loss: 3.9603
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 81s 532ms/step - accuracy: 0.0255 - loss: 3.9389 - val_accuracy: 0.0140 - val_loss: 3.9420
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 67s 540ms/step - accuracy: 0.0294 - loss: 3.9182 - val_accuracy: 0.0220 - val_loss: 3.9353
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 69s 551ms/step - accuracy: 0.0324 - loss: 3.9057 - val_accuracy: 0.0260 - val_loss: 3.9059
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 81s 539ms/step - accuracy: 0.0559 - loss: 3.8577 - val_accuracy: 0.0520 - val_loss: 3.7542
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 70s 559ms/step - accuracy: 0.1145 - loss: 3.5464 - val_accuracy: 0.3240 - val_loss: 2.8091
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 68s 543ms/step - accuracy: 0.3813 - loss: 2.3469 - val_accuracy: 0.5480 - val_loss: 1.4107
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 68s 547ms/step - accuracy: 0.6668 - loss: 1.1948 - val_accuracy: 0.8280 - val_loss: 0.7642
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 68s 546ms/step - accuracy: 0.8380 - loss: 0.6179 - val_accuracy: 0.8600 - val_loss: 0.4640
Done
```

```
Chris@MULLET:~/MULLET/src$ ./4.sh
TensorFlow 2.17.0
Load RWF CNN
Build one target RWF from file
Convert list of target RWF to NumPy array
Predict target
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step
33-04-E7-92-52-BD 0.9997809529304504

Chris@MULLET:~/MULLET/src$ 
```

Fig. 11. Repetitive 30, Strength 15% results in confidence of 99% in the correctly guessed UE match.

5.1.5. Ranges of Generation Parameters Results

Repetition

Strength

	10	15	20	25	30	35
10%	33%	2%	4%	3%	2%	2%
15%	2%	2%	2%	2%	99%	9%
20%	2%	22%	59%	**99%**	90%	2%
25%	99%	99%	99%	95%	12%	99%
30%	99%	99%	99%	99%	99%	99%
35%	70%	99%	99%	99%	15%	2%

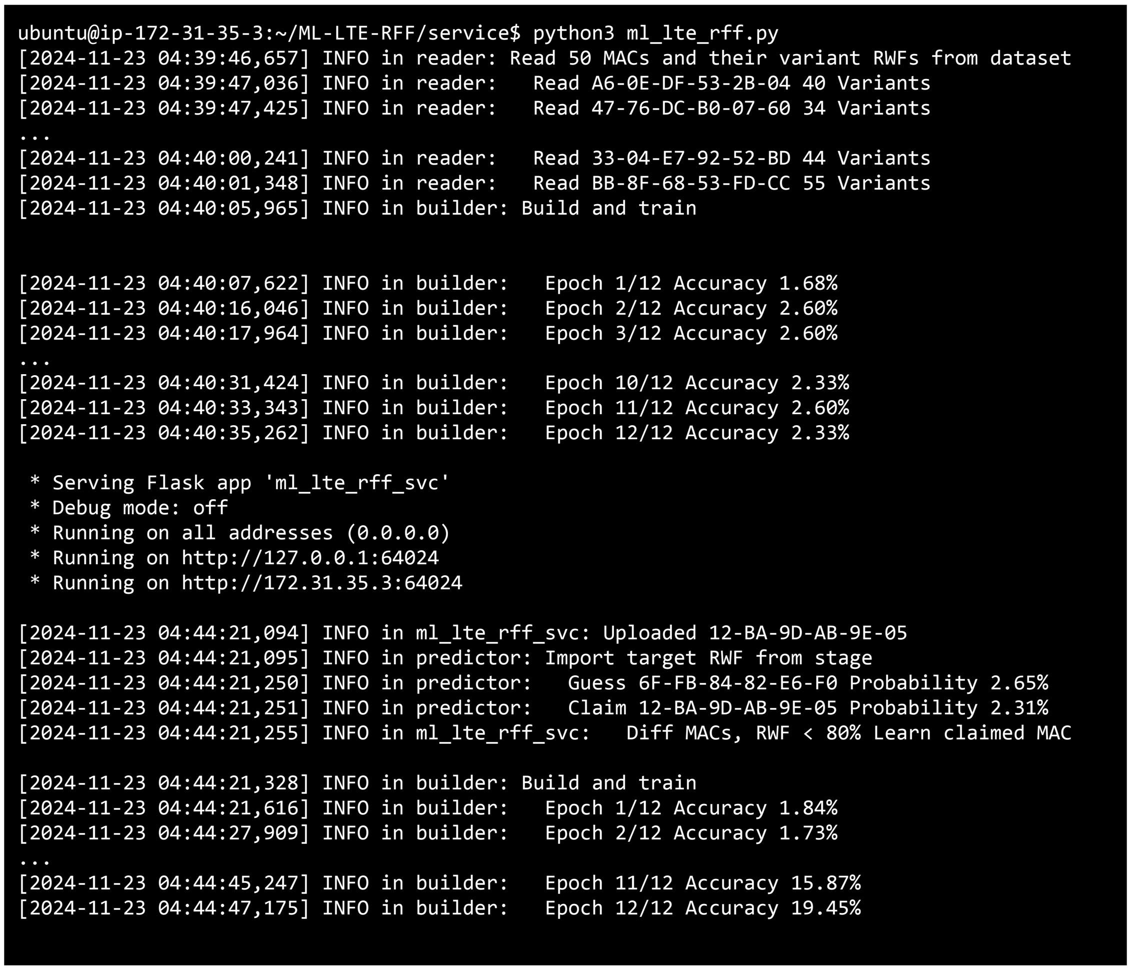
Fig. 12. Probabilities in red indicate that guessed MAC address was not the one requestor submitted. Highlighted **bold** indicates the parameters chosen to continue using for the application's neural network (25, 20%)

5.2. Operating Experimentation

In the main application, launch service in one terminal. In a different terminal, copy RWFs from the previously generated dataset for simulation to the current directory, experimenting with different MAC addresses. TensorFlow applies the RWFs submitted by the requestor against the neural network. Observe flag directory as well as service operations in the first terminal. Details follow in the next subsections..

5.2.1. Network Service Log

Fig. 13a and b show the service launch initially reading in the dataset and training the neural network. Then Flask is invoked and the HTTP server goes live. Submitted waveforms named as the MAC addresses they claim to be are tested against the neural network. The four cases are addressed.



```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF/service$ python3 ml_lte_rff.py
[2024-11-23 04:39:46,657] INFO in reader: Read 50 MACs and their variant RWFs from dataset
[2024-11-23 04:39:47,036] INFO in reader:   Read A6-0E-DF-53-2B-04 40 Variants
[2024-11-23 04:39:47,425] INFO in reader:   Read 47-76-DC-B0-07-60 34 Variants
...
[2024-11-23 04:40:00,241] INFO in reader:   Read 33-04-E7-92-52-BD 44 Variants
[2024-11-23 04:40:01,348] INFO in reader:   Read BB-8F-68-53-FD-CC 55 Variants
[2024-11-23 04:40:05,965] INFO in builder: Build and train


[2024-11-23 04:40:07,622] INFO in builder:   Epoch 1/12 Accuracy 1.68%
[2024-11-23 04:40:16,046] INFO in builder:   Epoch 2/12 Accuracy 2.60%
[2024-11-23 04:40:17,964] INFO in builder:   Epoch 3/12 Accuracy 2.60%
...
[2024-11-23 04:40:31,424] INFO in builder:   Epoch 10/12 Accuracy 2.33%
[2024-11-23 04:40:33,343] INFO in builder:   Epoch 11/12 Accuracy 2.60%
[2024-11-23 04:40:35,262] INFO in builder:   Epoch 12/12 Accuracy 2.33%

 * Serving Flask app 'ml_lte_rff_svc'
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:64024
 * Running on http://172.31.35.3:64024

[2024-11-23 04:44:21,094] INFO in ml_lte_rff_svc: Uploaded 12-BA-9D-AB-9E-05
[2024-11-23 04:44:21,095] INFO in predictor: Import target RWF from stage
[2024-11-23 04:44:21,250] INFO in predictor:   Guess 6F-FB-84-82-E6-F0 Probability 2.65%
[2024-11-23 04:44:21,251] INFO in predictor:   Claim 12-BA-9D-AB-9E-05 Probability 2.31%
[2024-11-23 04:44:21,255] INFO in ml_lte_rff_svc:   Diff MACs, RWF < 80% Learn claimed MAC

[2024-11-23 04:44:21,328] INFO in builder: Build and train
[2024-11-23 04:44:21,616] INFO in builder:   Epoch 1/12 Accuracy 1.84%
[2024-11-23 04:44:27,909] INFO in builder:   Epoch 2/12 Accuracy 1.73%
...
[2024-11-23 04:44:45,247] INFO in builder:   Epoch 11/12 Accuracy 15.87%
[2024-11-23 04:44:47,175] INFO in builder:   Epoch 12/12 Accuracy 19.45%
```

Fig. 13a. Server (cont'd).

```
[2024-11-23 04:47:44,781] INFO in ml_lte_rff_svc: Uploaded 12-BA-9D-AB-9E-05
[2024-11-23 04:47:44,781] INFO in predictor: Import target RWF from stage
[2024-11-23 04:47:44,797] INFO in predictor:   Guess 12-BA-9D-AB-9E-05 Probability 60.65%
[2024-11-23 04:47:44,798] INFO in predictor:   Claim 12-BA-9D-AB-9E-05 Probability 60.65%
[2024-11-23 04:47:44,799] INFO in ml_lte_rff_svc:   Same MACs, RWF >= 50% Checks out

[2024-11-23 04:48:08,921] INFO in ml_lte_rff_svc: Uploaded BB-8F-68-53-FD-CC
[2024-11-23 04:48:08,921] INFO in predictor: Import target RWF from stage
[2024-11-23 04:48:08,936] INFO in predictor:   Guess BB-8F-68-53-FD-CC Probability 46.54%
[2024-11-23 04:48:08,937] INFO in predictor:   Claim BB-8F-68-53-FD-CC Probability 46.54%
[2024-11-23 04:48:08,938] INFO in ml_lte_rff_svc:   Same MACs, RWF < 50% Strengthen

[2024-11-23 04:48:09,013] INFO in builder: Build and train
[2024-11-23 04:48:09,280] INFO in builder:   Epoch 1/12 Accuracy 1.89%
[2024-11-23 04:48:15,581] INFO in builder:   Epoch 2/12 Accuracy 2.16%
...
[2024-11-23 04:48:32,962] INFO in builder:   Epoch 11/12 Accuracy 7.30%
[2024-11-23 04:48:34,898] INFO in builder:   Epoch 12/12 Accuracy 22.61%

[2024-11-23 04:51:25,116] INFO in ml_lte_rff_svc: Uploaded BB-8F-68-53-FD-XX
[2024-11-23 04:51:25,117] INFO in predictor: Import target RWF from stage
[2024-11-23 04:51:25,131] INFO in predictor:   Guess BB-8F-68-53-FD-CC Probability 94.62%
[2024-11-23 04:51:25,131] INFO in predictor:   Claim BB-8F-68-53-FD-XX Probability N/A
[2024-11-23 04:51:25,132] INFO in ml_lte_rff_svc:   Diff MACs, RWF > 80% Flag for examination
```

Fig. 13b. Server (cont'd).

The flagged waveform, potentially spoofing a different MAC address, is saved as the combination of the two addresses in Fig. 14.

```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF/service$ ls flag
BB-8F-68-53-FD-XX_BB-8F-68-53-FD-CC
```

Fig. 14. Server flagged directory.

5.2.2. Requestor

Fig. 15 corresponds with the requests made to the service above. A known waveform is copied to a new fake MAC address to show how a UE might be trying to spoof that address. The UE is identifiable by its RF fingerprint.

```
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF$ curl -F "rwf=@12-BA-9D-AB-9E-05" http://35.153.176.133:64024
  Diff MACs, RWF < 80% Learn claimed MAC

ubuntu@ip-172-31-35-3:~/ML-LTE-RFF$ curl -F "rwf=@12-BA-9D-AB-9E-05" http://35.153.176.133:64024
  Same MACs, RWF >= 50% Checks out

ubuntu@ip-172-31-35-3:~/ML-LTE-RFF$ curl -F "rwf=@BB-8F-68-53-FD-CC" http://35.153.176.133:64024
  Same MACs, RWF < 50% Strengthen

ubuntu@ip-172-31-35-3:~/ML-LTE-RFF$ cp BB-8F-68-53-FD-CC BB-8F-68-53-FD-XX
ubuntu@ip-172-31-35-3:~/ML-LTE-RFF$ curl -F "rwf=@BB-8F-68-53-FD-XX" http://35.153.176.133:64024
  Diff MACs, RWF > 80% Flag for examination
```

Fig. 15. Requestor.

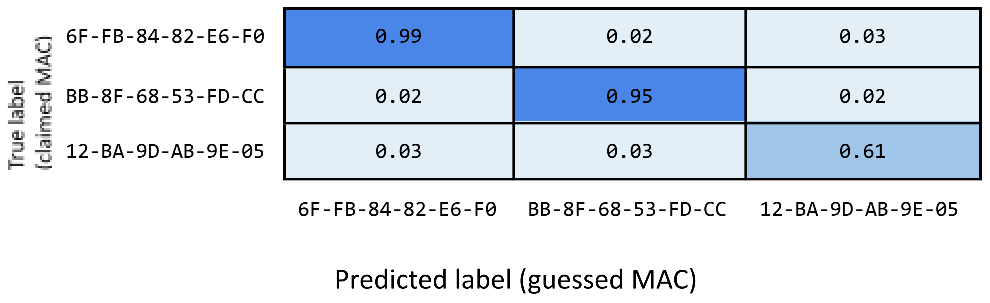
5.3. Observations

For test and demonstration, a similar AWS instance of type g4dn.xlarge was then stood up with the NVIDIA CUDA-enabled GPU and drivers specifically for deep neural networks, and training significantly performed faster [6]. Where TensorFlow training of 10 epochs without the GPU would take more than 10 minutes, the same process with the GPU would take less than 1 minute. This allowed retraining to be performed upon a requestor's submission.

However, it was observed that retraining did require resetting the neural network, and not just training only the new entry. When even on ideal RFF parameters and performing as high as 99% accuracy, training only a new entry would then result in accuracy worsening as low as 2% for most predictions. This is a documented phenomenon known as catastrophic forgetting.

Another neural network behavior was observed that training accuracy would sometimes inexplicably result as insufficient. This was not as much of an issue, however, as it usually took just a few submissions for the neural network to adequately yield acceptable results. This may have been a significant display of machine learning, in that the neural network essentially learned by itself.

The values of the confusion matrix in Fig. 16 shows ideal parameters for the prototype yielded successful results enough to continue addressing challenges and refining the system. The low probability of 0.61 was an effect of that particular device not having a largely represented dataset. Only after the requestor submitted that device just two more times did its value become adequate.



True label (claimed MAC) \ Predicted label (guessed MAC)	6F-FB-84-82-E6-F0	BB-8F-68-53-FD-CC	12-BA-9D-AB-9E-05
6F-FB-84-82-E6-F0	0.99	0.02	0.03
BB-8F-68-53-FD-CC	0.02	0.95	0.02
12-BA-9D-AB-9E-05	0.03	0.03	0.61

Fig. 16. Normalized confusion matrix.

5.4. Assumptions

Because this study was time-constrained, LTE waveforms and RF fingerprint behavior could only be simulated by software. The waveforms of RF fingerprints were also fabricated as arbitrary combinations of sine and cosine due to their periodic nature. Finally, RMC waveforms were truncated to 5000 data points as opposed to the LTE compliant 307200 for handleability, to avoid overflow when collecting waveforms from the dataset.

6. Discussion

This new mode of authenticating a device can replace other methods due to its specificity. Current knowledge determines that the fingerprint of a device cannot be spoofed to be exactly as another device, allowing proper authentication for a device trying to connect to another. Comparing the fingerprint of a device can also be added upon other security features for added security. Additionally, things like intrusion detection or spectrum management could benefit from the RF fingerprint identification. With intrusion detection being able to determine if the device is valid, or if they have spoofed credentials and managed to get somewhere they should not be. With spectrum management being able to determine if a radio frequency is being used or abused by a certain user or device when it should not be.

Ultimately, this kind of ML deep neural network can be extended to any wireless platform. LTE was only applied here to demonstrate the principle of RF fingerprint recognition. Social, privacy and legal implications must still be considered for any wireless-ML technology. For example, it must be decided how much identifying information such as MAC addresses can be associated with an RF fingerprint before breaching privacy standards. Another example reflects how the traditional sense of fingerprinting has been accepted as solid evidence of identification, but RF fingerprinting still needs to undergo uncountable factors before it may be used literally.

7. Conclusion

The study was successful in understanding the concept of RF fingerprinting and applying it to LTE standards. With that understanding, a sample set of simulated uplink signals was generated large enough to build a machine learning neural network model. The model was then used in a system enabling practical use by vendors for authenticating UE, while strengthening the model. The different outcomes were addressed, and further work was considered.

7.1. Further Work

Real RF Fingerprint characteristics from hardware radios can be exchanged for the simulated ones created in the provided code. Then real characteristics can be assessed and used to train the model on what to look for, determining proper authentication parameters. Additionally, the simulated waveforms can be swapped out for captured ones and the model can be trained on those as well.

As this study focuses on the essential functions of a minimal prototype, full-scope security hardening should still be performed including HTTPS and vendor certification, elimination of signal truncation and the resolution of catastrophic forgetting. Upon community acceptance, the service IP would be assigned a special port number for secure standardization. Dataset would take advantage of cloud storage and compression into database systems instead of the current local file system structure.

The impact of distance affecting the degradation of the waveform, can be addressed with sensitivity of detection hardware along with the other signal impairments. This would allow the waveform to be properly captured and assessed.

Additionally, the challenge of hardware degradation can be addressed, updating properties and still adjusting with deep learning. This can have significant impacts on the created deep learning models, as an RF fingerprint may become similar to another or too dissimilar to the original one used in the model. This can cause problems in authentication, including potential spoofing or access denial. Additionally, updating properties like changing out the CPU inside a computer, can cause the produced fingerprint to be changed as well. This would require the model to then be adjusted to include the new fingerprint and the old removed.

References

[1] S. Riyaz, K. Sankhe, S. Ioannidis and K. Chowdhury, "Deep Learning Convolutional Neural Networks for Radio Identification," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 146-152, Sept. 2018, doi: 10.1109/MCOM.2018.1800153.

[2] S. Abbas, Q. Nasir, and D. Nouichi, "Improving security of the Internet of Things via RF fingerprinting based device identification system," *Neural Comput & Applic,* 33, 14753–14769 (2021). https://doi.org/10.1007/s00521-021-06115-2

[3] F. Azzawi, Z. Azzawi, S. Azzawi, and F. Abid, "Reference measurement channel RMC parameters of LTE downlink waveforms," *2020 IOP Conf. Ser.: Mater. Sci. Eng.* 881 012107, doi: 10.1088/1757-899X/881/1/012107.

[4] F. Ertam and G. Aydın, "Data classification with deep learning using Tensorflow," *2017 International Conference on Computer Science and Engineering (UBMK)*, Antalya, Turkey, 2017, pp. 755-758, doi: 10.1109/UBMK.2017.8093521.

[5] A. Verma, C. Kapoor, A. Sharma and B. Mishra, "Web Application Implementation with Machine Learning," *2021 2nd International Conference on Intelligent Engineering and Management (ICIEM)*, London, United Kingdom, 2021, pp. 423-428, doi: 10.1109/ICIEM51511.2021.9445368.

[6] M. Guillermo, "Implementation of Automated Annotation through Mask RCNN Object Detection model in CVAT using AWS EC2 Instance," *2020 IEEE REGION 10 CONFERENCE (TENCON)*, Osaka, Japan, 2020, pp. 708-713, doi: 10.1109/TENCON50793.2020.9293906.

[7] H. Zarrinkoub, "Simulation," *Understanding LTE with MATLAB: From Mathematical Modeling to Simulation and Prototyping*, Wiley, 2013, pp.353-419, doi: 10.1002/9781118443446.ch9.

[8] B. Hilburn, "SigMF: The Signal Metadata Format," *Proceedings of the GNU Radio Conference*, [S.l.], v. 3, n. 1, Sep. 2018. https://pubs.gnuradio.org/index.php/grcon/article/view/52