Real-time Device Fingerprinting and Re-identification in GNUradio

Stepan Mazokha SMAZOKHA2016@FAU.EDU
Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431 USA

Jose Sanchez JOSESANCHEZ2019@FAU.EDU
Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431 USA

George Sklivanitis GSKLIVANITIS@FAU.EDU
Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431 USA

Abstract

WiFi device fingerprinting and re-identification (RFFI) are critical functions for wireless security, mobility intelligence, and spectrum sharing applications. However, rapid prototyping and evaluation of fingerprint extraction and classification models remains a challenging task. Model performance evaluation requires exploring various environmental conditions, emitter types, and more. To streamline this process, we introduce *gr-mobrffi* — an out-of-tree GNUradio project for real-time evaluation of device fingerprinting and re-identification models, available on GitHub [1]. The *gr-mobrffi* GNU Radio blocks are designed to ingest WiFi frame preambles and produce vector embeddings to fingerprint and re-identify devices. We evaluate *gr-mobrffi* by performing real-time fingerprinting and re-identification on twelve WiFi emitters, while capturing WiFi probe requests with an AntSDR running OpenWiFi, and a custom Rust app. Indoor testing demonstrates the accuracy of our framework on fingerprint matching and re-identification by displaying WiFi device labels to a GUI.

1. Introduction

RFFI is a crucial technique for various applications, including mobility intelligence, wireless security, and others. It relies on hardware impairments of the RF chains of wireless transmitters and can be a powerful tactic for non-cooperative extraction of permanent device identifiers.

The core approach behind this technology is using deep learning models to recognize device-identifying features in either raw or transformed IQ samples from a wireless receiver.

However, RFFI is exposed to several issues that can significantly hinder the performance of fingerprinting models and complicate the path to building a production-ready RFFI system. First, fingerprint extraction models require diverse datasets that represent signals from all supported RF chipsets and device models, as well as across various environmental conditions and receiver configurations. Second, the fingerprinting approaches are sensitive to minute changes in the environment (e.g., micro-Doppler effects (Argyriou, 2023)), which can be hard to capture in a dataset. Furthermore, device fingerprints are prone to degrading over time and being affected by the lack of synchronization in transceiver systems (e.g., carrier frequency offsets (Zhang et al., 2019)).

One of the key aspects to addressing these challenges is rapid prototyping. However, the state-of-the-art re-identification method evaluation is slow and inefficient, being limited by a laborious data collection process and "offline" evaluation stage, which limits the researcher's capacity to quickly explore the behavior of new models and techniques in a dynamic environment.

To this end, we introduce *gr-mobrffi* – a suite of GNUradio blocks designed to ingest, run inference on extracted WiFi preambles, perform fingerprint matching, vector store retrieval, and display the device labels to the user. In addition, considering our focus on 802.11 stack, we present an app for configuring, capturing, and extracting WiFi preambles. We evaluate *gr-mobrffi* on a set of WiFi emitters that are configured to transmit WiFi probe requests with a spoofed MAC address, and achieve a 94% device re-identification accuracy in an indoor closed-set scenario. Finally, we open-source all components of the described pipeline for continued development.

[1] https://github.com/smazokha/gr-mobrffi

Proceedings of the 15th GNU Radio Conference, Copyright 2025 by the author(s).

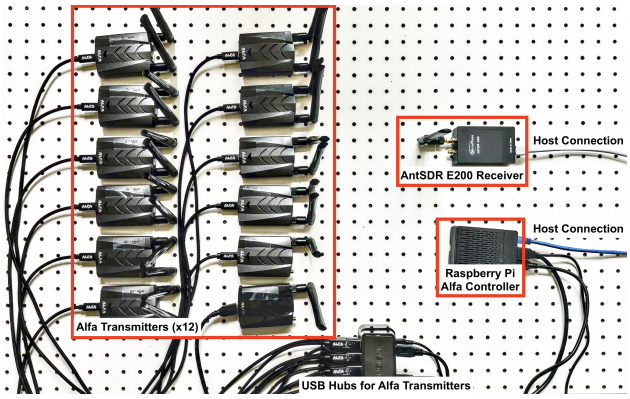

Figure 1. Experimental setup.

The rest of this paper is structured as follows. Section 2 describes our experimental setup, and the motivation behind emitting WiFi probe requests. Section 3 provides a description of our Rust-based signal capture and pre-processing application. Section 4 demonstrates the functionality, design, and implementation of each of the *gr-mobrffi* blocks. Section 5 briefly demonstrates the preliminary performance results of the *gr-mobrffi* on our experimental setup. Finally, Section 6 summarizes the results of our work and outlines our future plans.

2. Hardware and Experimental Setup

Our experimental setup consists of twelve WiFi adapters and one SDR, illustrated in Fig. 1. More specifically, we consider 12 Alfa WiFi Adapters with an RTL8812AU chipset onboard, configured to transmit WiFi packets in monitor mode on channel 36 of 5 GHz WiFi band (5180 MHz). Transmitters are simultaneously connected to a Raspberry Pi with a *scipy*-based probe injection tool (Bao, 2022) installed to configure and perform the transmission process. The tool provides an easily accessible interface to specify the contents of emitted packets, namely, WiFi probe requests, such as the source and destination MAC addresses, WiFi channel, network interface, and more. For further exploration of the tool, the reader is encouraged to review our previous work (Mazokha et al., 2021b). A single AntSDR E200 software-defined radio receiver (Technology, 2023) is connected to the host device running *Ubuntu 22.04*. The device has been configured to run OpenWiFi firmware (Jiao & the OpenWiFi Project contributors, 2025) with a custom onboard Rust app to perform simultaneous capture of RadioTap headers from the FPGA-based WiFi decoder, and the IQ samples from the DMA buffer. Importantly, OpenWiFi software only supports OFDM. Considering that WiFi probe requests are emitted on the lowest available data rate (DSSS on the 2.4 GHz band), this motivated our choice of transmitting signal on the 5 GHz band.

2.1. Why WiFi Probe Requests?

Our selection of WiFi probe requests is motivated by our core research interest in building a robust WiFi-based mobility intelligence system for smart cities (Mazokha et al., 2021b;a). Prior work from our group explored several avenues for extracting valuable insights about pedestrian and vehicular traffic using WiFi probe. Probe requests are ubiquitously emitted by WiFi-enabled mobile devices in attempts to discover and later connect to the nearby WiFi access points. They can also be used to gauge the public presence in the area. However, modern operating systems and WiFi revisions randomize MAC addresses encapsulated in these packets to enhance privacy, which introduces artificially inflated device counts and forbids movement trajectory analysis in public spaces. To this end, WiFi probe request-based RFFI can be an effective solution.

3. Signal Capture and Pre-processing

The task of signal capture consists of two tasks. First, we must capture raw IQ samples transmitted by the Alfa emitter on an open WiFi channel. Second, we must decode the same frame to extract supplemental device information, such as MAC addresses, SEQ numbers, and PNL lists, to use for filtering frames and potentially synchronize data capture across multiple receivers.

These tasks can be easily achieved separately. Raw IQ samples can be captured using either a USRP (e.g., Ettus X310) or an SDR (e.g., PlutoSDR or HackRF). They can be further decoded *offline* using tools such as Matlab WLAN Toolbox (The MathWorks, Inc., 2025), or a *gr-ieee802-11* GNU radio Wifi transceiver suite. WiFi frames can also be continuously decoded in real-time by hardware with monitor mode support, such as the Raspberry Pi, Alfa sniffers, and various other devices that support firmware patching tools, like Nexmon (Schulz et al., 2017). However, these functionalities are seldom supported together for real-time signal capture and decoding.

Fortunately, OpenWiFi, an open-source 802.11-compatible suite of firmware and software tools, can be a solution. It is capable of FPGA-enabled signal decoding, capturing both IQ samples and channel state information, as well as a variety of other functions. OpenWiFi also supports low-cost SDR solutions such as AntSDR E200, which is sufficiently cost-effective for adoption in a real-world environment.

Based on the above, we implement a Rust-based app for capturing both the IQ samples and the decoded fields of the emitted WiFi probe requests. The diagram of the app is illustrated in Fig. 2. It runs natively on AntSDR ARM SoC and continuously receives RadioTap headers decoded

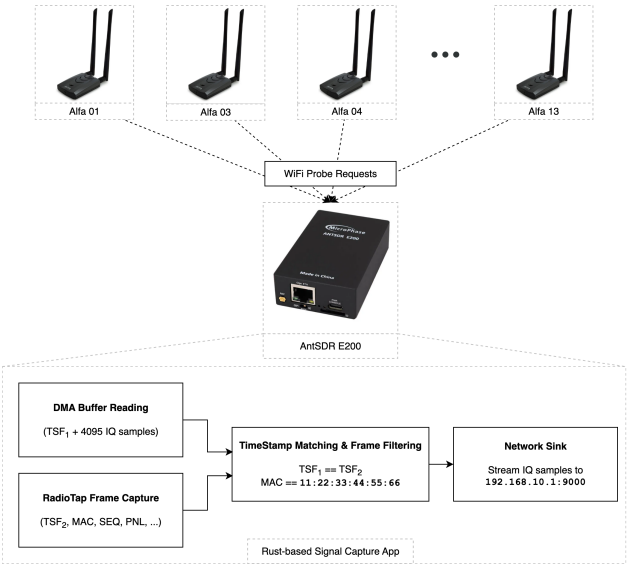

Figure 2. Rust-based signal capture app.

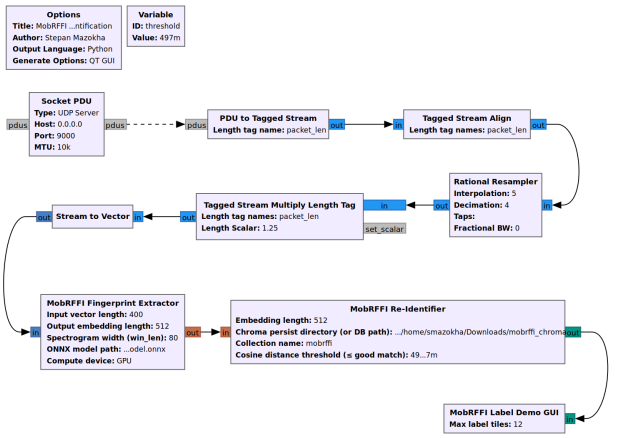

Figure 3. gr-mobrffi GNU Radio example flowgraph.

by the FPGA. It also receives the side-channel IQ samples from the DMA buffer, which is capable of containing up to 4095 IQ samples at a time. The system is configured to trigger frame capture once the receiver passes the legacy signal field (L-SIG) checksum verification. This process ensures that there is a fixed offset between the start of the buffer and the start of the OFDM preamble. This consistent trigger time allows us to efficiently extract the frame preamble by trimming the buffer by a known number of samples.

Considering that both the IQ buffer reading and the RadioTap header capture processes are independent, we match the extracted data using timestamp values encoded in both data sources. For the DMA buffer, the timestamps are encoded in the first 64 bits of the buffer vector, while the RadioTap header contains an additional and easily available timestamp field.

Next, we filter the received traffic using the decoded MAC address field, knowing the expected spoofed value from the transmitter chain. Considering that the IQ samples are stored in the buffer in the int16 format, we also scale the values by $\frac{2^{16}}{2}$ to bring them to the standard $[-1; 1]$ value range. Finally, the IQ samples, as well as the RadioTap fields such as the MAC address, SEQ number, and the timestamp value, are streamed to the host for further processing.

The entire pipeline provides a robust path to capturing WiFi traffic, and importantly does not overflow the maximum available bitrate of the fastest available interfaces to communicate with the host – 1 Gigabit Ethernet cable.

4. *gr-mobrffi*

The *gr-mobrffi* pipeline is illustrated in Fig. 3. It consists of six stages. First, the packets containing the filtered WiFi preamble IQ samples are received by the Socket PDU block and are transformed into a tagged stream. Next, the IQ samples are upsampled from 20 Msps to 25 Msps to conform to the required format of the MobRFFI fingerprint extractor model. The data is then converted from a stream to a vector and sent as an input to the MobRFFI Fingerprint Extractor. The produced fingerprint is then ingested by the MobRFFI Re-identifier, which determines the label for the transmitter. Finally, the produced label is displayed to the user. Each of the *gr-mobrffi* blocks is described in more detail below.

4.1. Fingerprint Extraction

The fingerprint extractor is designed to ingest raw IQ samples of an OFDM preamble and produce an embedding vector containing device-identifying features of the transmitter. The block has several parameters: input vector length, output embedding length, width of the spectrogram into which the preamble is transformed, path to the trained extractor model file, and a selector specifying whether the inference should be performed on a CPU or a GPU.

For each input, a vector of 400 IQ samples of an upsampled IQ spectrogram, two steps are performed. First, the vector is transformed into a channel-independent spectrogram, as illustrated in Fig. 4. The process has been thoroughly explored in our prior work (Mazokha et al., 2025). It consists of (a) extracting the preamble from an OFDM frame on AntSDR, (b) performing a short-term Fourier transform (STFT) to convert the IQ samples into a frequency domain, and (c) dividing the neighboring STFT bins to suppress

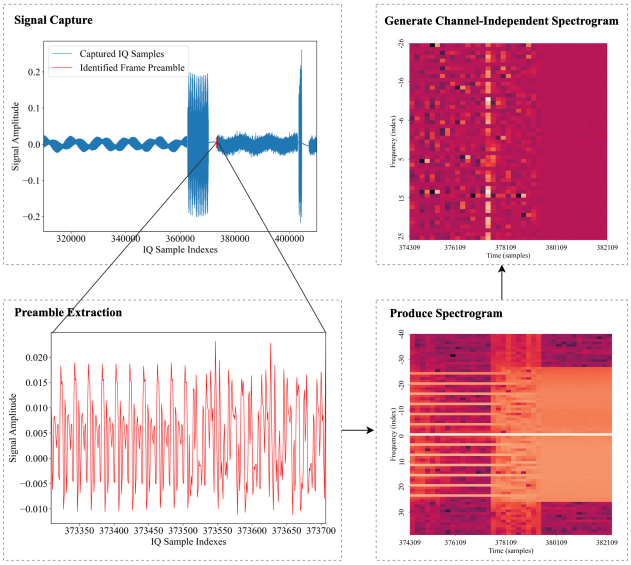

Figure 4. WiFi frame preamble transformation.

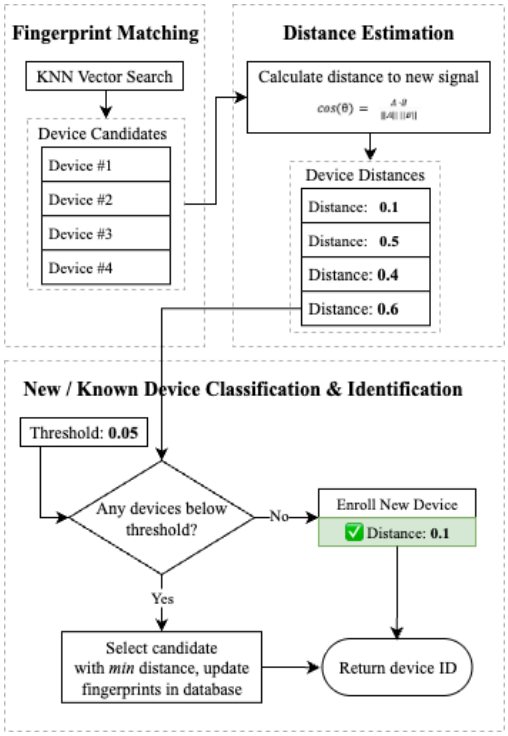

Figure 5. Device re-identification framework.

wireless channel effects while preserving device-specific characteristics of the captured signal. Finally, we remove the DC and guard subcarriers, since they do not hold the information transmitted by the emitter and therefore do not contribute to the fingerprinting task. Second, the channel-independent spectrogram is ingested by the deep learning model, trained to extract features representing the hardware impairments of the transmitting RF chain. We reuse the model design from our prior work, which was originally implemented in TensorFlow and Keras. However, to maintain the framework-agnostic approach, we adopt the ONNX runtime environment (Microsoft Corporation, 2019), and convert our model weights to the new standard. Finally, the result of the block is a vector embedding, which can be used for attempting to match the transmitter against a database of previously identified devices.

4.2. Re-identification

The re-identifier block is the cornerstone of the *gr-mobrffi* pipeline. It ingests extracted embedding vectors and attempts to make two decisions: 1) whether the associated device is known to the system (i.e., known or rogue) and, 2) if so, which of the known devices matches the embedded features. To achieve this task, the block has the following parameters: vector embedding length, path to the local vector database file (e.g., ChromaDB (Chroma Core contributors, 2025)), name of the database collection to create or contribute to, and a pre-determined cosine similarity threshold for identifying unknown devices.

The workflow of the block is illustrated in Fig. 5. First, the input embedding is used to query the vector database. This process involves using KNN and a cosine similarity distance metric to identify the top-K potential matches (i.e., device candidates) that most closely resemble the new embedding. For the identified candidates, the cosine similarity value is compared against the threshold value, which is established experimentally during the model training stage. If any candidate's similarity metric is above the constant, we consider the candidate to be a new device. In such a case, a new device label is produced and stored in the database along with the embedding vector. Alternatively, the device is considered to be known, and the label of the candidate with the highest available similarity value is the result. The result of the block is a label value of an integer type, which is reused if the same device is identified again.

4.3. Real-time Demonstration

The label demo simply displays the produced device label to the user. Its GUI example is illustrated in Fig. 6. The block has a single parameter – the number of anticipated labels. Once a new label is received, the block associates its value with the first available tile in the window. On repeated reception, the tile colors are altered to display the active device with a green color.

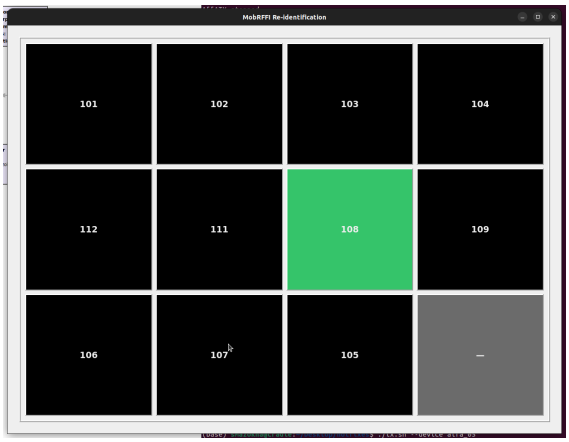

Figure 6. Demo GUI.

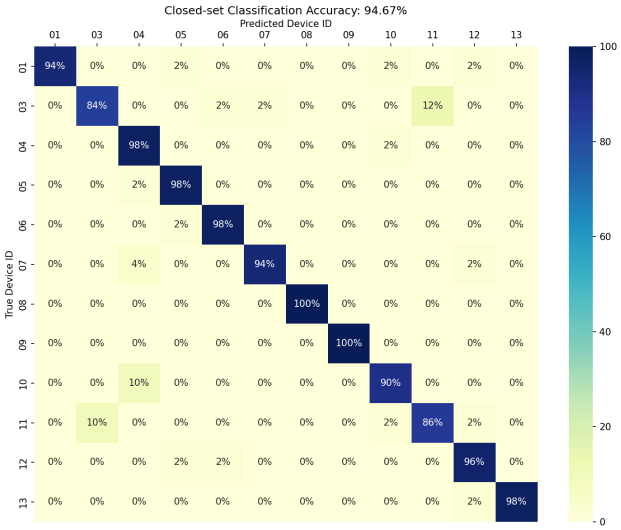

Figure 7. Closed-set re-identification accuracy.

5. Preliminary Indoor Performance

We performed an experimental evaluation of our signal capture, pre-processing, and re-identification pipeline indoors on a set of 12 Alfa WiFi transmitters. During this experiment, a set of 1000 frames per device was used for training the fingerprint extractor model. Device classification has been performed across a set of 100 additional frames per device. The results are illustrated in Fig. 7. Here, the heatmap illustrates a confusion matrix, where rows and columns represent the device identifiers, while the color of each tile represents the number of matches between each pair of devices. During this initial experiment, the framework achieves 94.67% classification accuracy.

6. Conclusion and Future Work

In this paper, we presented *gr-mobrffi*, a system for evaluating RFFI models and frameworks in real-time on WiFi packets. As part of the proposed suite of tools, we presented a two-fold: a Rust-based app for AntSDR with OpenWiFi firmware to capture and pre-process the WiFi probe requests, and a set of GNUradio blocks to receive, upsample, run inference on preamble IQ samples using an ONNX model, perform device re-identification, present, and store the device fingerprints in a local vector database. During our initial evaluation indoors on a set of 12 Alfa transmitters, we achieved 94% device classification accuracy.

gr-mobrffi presents a key path to building a production-ready RFFI system through real-time prototyping. Since the project is open-sourced, the model, the database choice, and the re-identification algorithm can be modified and improved over time.

Our results prompt several paths for future work. First, building a real-world RFFI system requires training a fingerprinting model on a large fleet of devices that represent the majority of commercially adopted RF chipsets. To this end, we plan to use the AntSDR and the proposed app to capture a large dataset and train a larger model. Second, the RFFI system is prone to various environmental effects, which can hinder the system's performance. As a result, we plan to integrate the AntSDR and the GNUradio workflow into our MobIntel testbed in West Palm Beach and upgrade a set of 54 sensors to enable real-world experimentation with *gr-mobrffi*. Finally, the scalability of the RFFI framework requires its deployment on a large number of sensors, with the ability to share the fingerprints. Hence, we will explore the avenue of adopting federated learning techniques to train and fine-tune models on the edge, while sharing and averaging model weights across multiple sensors.

Acknowledgment

This research is supported through the ERC for Smart Streetscapes, funded through NSF award EEC-2133516.

We gratefully acknowledge the support of (1) the City of West Palm Beach; (2) Chris Roog, Executive Director of the Community Redevelopment Agency in the City of West Palm Beach; and (4) Florida Atlantic University's *I-SENSE* team.

References

Argyriou, Antonios. Obfuscation of human micro-doppler signatures in passive wireless radar. *IEEE Access*, 11: 40121–40127, 2023.

Bao, Fanchen. Probe request injection tool. https://github.com/FanchenBao/probe_request_injection, 2022. GitHub repository, MIT License.

Chroma Core contributors. Chroma: Ai-native open-source embedding / vector database. https://github.com/chroma-core/chroma, 2025. GitHub repository, Apache-2.0 license.

Jiao, Xianjun and the OpenWiFi Project contributors. Openwifi. https://github.com/open-sdr/openwifi, 2025. GitHub repository, AGPL-3.0 licensed.

Mazokha, Stepan, Bao, Fanchen, Sklivanitis, George, and Hallstrom, Jason O. Urban-scale testbed infrastructure for data-driven wireless research. In *2021 IEEE 4th 5G World Forum (5GWF)*, pp. 517–522. IEEE, 2021a.

Mazokha, Stepan, Bao, Fanchen, Zhai, Jiannan, and Hallstrom, Jason O. Mobintel: Sensing and analytics infrastructure for urban mobility intelligence. *Pervasive and Mobile Computing*, 77:101475, 2021b.

Mazokha, Stepan, Bao, Fanchen, Sklivanitis, George, and Hallstrom, Jason O. Mobrffi: Non-cooperative device re-identification for mobility intelligence. *arXiv preprint arXiv:2503.02156*, 2025.

Microsoft Corporation. Onnx runtime: cross-platform, high-performance ml inferencing and training accelerator. https://github.com/microsoft/onnxruntime, 2019. GitHub repository, MIT License; first stable release announced in 2019 :contentReference[oaicite:0]index=0.

Schulz, Matthias, Wegemer, Daniel, and Hollick, Matthias. Nexmon: The c-based firmware patching framework, 2017. URL https://nexmon.org.

Technology, MicroPhase. Antsdr e200: Fpga-based sdr platform (zynq + ad936x). https://www.crowdsupply.com/microphase-technology/antsdr-e200, 2023. CrowdSupply crowdfunding project; sample rate up to 61.44 MS/s, tuning range up to 6 GHz.

The MathWorks, Inc. WLAN Toolbox — MATLAB. https://www.mathworks.com/products/wlan.html, 2025. [Online; accessed 05-Sep-2025].

Zhang, Dongheng, Hu, Yang, Chen, Yan, and Zeng, Bing. Calibrating phase offsets for commodity wifi. *IEEE Systems Journal*, 14(1):661–664, 2019.