

# Tweak: Towards Portable Deep Learning Models for Domain-Agnostic LoRa Device Authentication

Jared Gaskin

*Oregon State University*

Corvallis, Oregon, USA

gaskinj@oregonstate.edu

Bechir Hamdaoui

*Oregon State University*

Corvallis, Oregon, USA

hamdaoui@oregonstate.edu

Weng-Keen Wong

*Oregon State University*

Corvallis, Oregon, USA

wongwe@oregonstate.edu

*Abstract*—Deep learning based device fingerprinting has emerged as a key method of identifying and authenticating devices solely via their captured RF transmissions. Conventional approaches are not portable to different domains in that if a model is trained on data from one domain, it will not perform well on data from a different but related domain. Examples of such domains include the receiver hardware used for collecting the data, the day/time on which data was captured, and the protocol configuration of devices. This work proposes *Tweak*, a technique that, using metric learning and a calibration process, enables a model trained with data from one domain to perform well on data from another domain. This process is accomplished with only a small amount of training data from the target domain and without changing the weights of the model, which makes the technique computationally lightweight and thus suitable for resource-limited IoT networks. This work evaluates the effectiveness of *Tweak* vis-a-vis its ability to identify IoT devices using a testbed of real LoRa-enabled devices under various scenarios. The results of this evaluation show that *Tweak* is viable and especially useful for networks with limited computational resources and applications with time-sensitive missions.

*Index Terms*—Device authentication, domain-agnostic portable device fingerprints, learning model calibration.

## I. INTRODUCTION

Recent years have seen unprecedented growth in both the number and variety of IoT networks and applications [1]. Due to its ability for enabling long-range connectivity between IoT devices at low power, LoRa technology [2] has been widely adopted by hundreds of IoT application developers as the de facto wide-area wireless network access protocol. With such a rapid adoption of LoRa technology coupled with the massive numbers of emerging resource-constrained IoT devices, there is undoubtedly an urgent need for *lightweight* and *scalable* authentication mechanisms that can ensure automated and secure access to these LoRa-enabled IoT networks.

Device authentication mechanisms that are based on RF (radio frequency) fingerprinting have recently been recognized as key methods with great potential for complementing conventional cryptographic approaches to increase the security protection of IoT networks against unauthorized access [3]. These fingerprinting methods essentially consist of extracting

This work is supported in part by Intel/NSF MLWiNS Award No. 2003273. An IEEE-formatted version of this article is to appear in the 2022 IEEE Conference on Communications and Network Security (IEEE CNS 2022). Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

device-specific features (aka fingerprints) from received RF signals—typically caused by inevitable transceiver hardware impairments incurred during manufacturing, and leveraging these features to uniquely identify and classify wireless transmitters [3], [4]. Various feature extraction approaches have been taken, including early hand-crafted approaches that require RF signal domain knowledge and many trial-and-error attempts (e.g. [5]), and more recent approaches that leverage deep learning to extract features automatically from raw RF signals without requiring RF domain expertise (e.g. [3], [6]).

### A. Multi-Domain Portability Challenges

Although recent deep learning approaches have shown promising results, they fail to maintain good performance when the data used during training and that used during testing are collected under different domains. For example, when the model is trained on data collected at one receiver but tested on data collected at a different receiver, the device identification accuracy degrades substantially compared to when both training and testing are done on data collected using the same receiver. To demonstrate the impact of such challenges, we performed several experiments using data collected with an IoT testbed, consisting of 25 LoRa transmitters and 2 USRP B210 receivers, under three different deployment settings: indoors, outdoors and wired (detailed description of the testbed and experimental scenarios is given later in Sec. III). Our results depicted in Fig. 1a show the accuracy of the learning model when trained on data captured at one receiver (RX1), but tested on data captured at a different receiver (RX2), as well as at the same receiver (RX1). These results clearly show the substantial drop in accuracy due to the change in the receiver hardware. As depicted in Fig. 1b, similar trends are also observed when the model is trained on data captured on Day 1 but tested on data captured on a different day (while keeping the same receiver). The common trend in these figures is that the models only perform well when the testing domain (receiver/day) matches the training domain.

The issues of domain portability we just illustrated have also been demonstrated in other works [6]–[8]. Since this problem is well established, developing approaches that can overcome such issues has become an active research area in recent years.

### B. Limitations of Related Work

Past works attempting to address the wireless channel portability problem [3], [8]–[10] are almost all based on using data augmentation and/or custom feature extraction as preprocess-

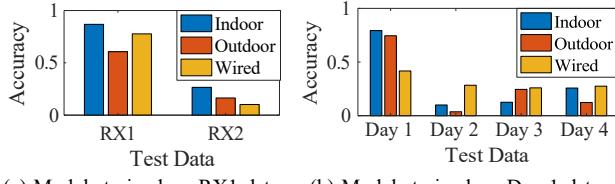


Fig. 1: Accuracy of models tested on different domains.

ing in order to extract RF fingerprints that are channel agnostic. The downside of these approaches is that the resulting model is not flexible and must be re-trained if it cannot perform well in a new condition. A notable exception to this is the work in [8], which proposed a system that can estimate the channel and adapt to it.

Other approaches [11]–[13] attempt to ensure portability with regard to receiver hardware. However, these methods require some form of potentially computationally intensive training or optimization in order to transform input data so that it can be correctly classified.

The authors of [4] make perhaps the most notable recent attempt at portability, and employ a method similar to the one described herein in the sense that both methods use metric learning and the triplet loss function. They address both channel portability and transmitter hardware portability (training and testing using two different sets of transmitters). Notably, this work is the only one, to the best of our knowledge, that performs its evaluation on open-set identification and classification [14], which allows inputs from unknown transmitters and attempts to reject these inputs.

The costly alternatives available to practitioners while research on portability continues are as follows: (i) training a new deep learning model for each domain one wishes to test with, and (ii) training a single deep learning model with data from multiple domains in an attempt to produce a domain-agnostic model. These approaches are both costly in terms of the extra data required from additional target domains and the computation time needed to train the neural network models.

### C. Our Contributions: Tweak

We propose Tweak, a lightweight device identification technique that enables portability of the deep learning models across multiple different domains. Tweak leverages metric learning to achieve accurate identification and open-set accuracy through model calibration instead of re-training, thus significantly reducing (i) the training time and (ii) the amount of needed data, making it more suitable for resource-constrained and real-time IoT applications. Tweak is evaluated using datasets collected from a testbed of 25 LoRa transmitters and 2 USRP B210 SDR receivers. Specifically, Tweak enables:

- Lightweight model portability through calibration that can be performed very quickly, without changing the model weights, and can be done using a relatively small amount of labeled data from the target domain.
- Multi-domain model portability across multiple different domains, including receiver hardware, communication

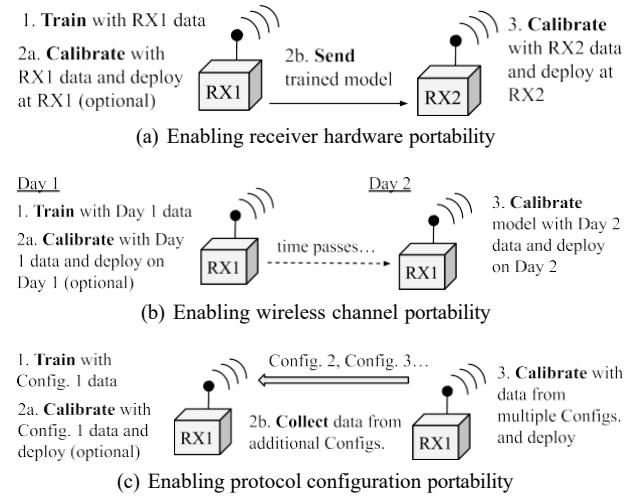


Fig. 2: Tweak: Transferring of a trained model to a different domain: (a) hardware, (b) channel, and (c) configuration.

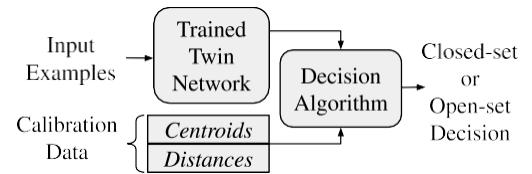


Fig. 3: A high-level diagram of Tweak

channel, and LoRa configuration, as shown in Fig. 2.

- Open-set device identification by easily acting as an open-set classifier, as Tweak handles inputs from unknown devices. Most related approaches that attempt to address portability do not perform open-set testing.

The remainder of this work is organized as follows: Sec. II presents Tweak. Sec. III and Sec. IV evaluate Tweak using RF data captured using a real testbed of IoT devices.

## II. Tweak: ENABLING DEEP LEARNING MODEL PORTABILITY FOR ROBUST DEVICE AUTHENTICATION

Our goal is to provide a device authentication method by which an open-set deep learning model can be trained on data from one domain (collected at one receiver, through one wireless channel, or with IoT transmitters using one LoRa configuration) and then be calibrated to perform well on data from a different but related domain. In this way, Tweak aims to move toward the creation of deep learning models for RF fingerprint authentication that are portable in terms of hardware, wireless channel, and LoRa protocol configuration. Tweak achieves these portability goals through a calibration process that is: (i) not computationally intensive (and can be done on IoT hardware that is less powerful than the resources used for training the original model) and (ii) accomplished with a limited amount of labeled training data from the target domain (receiver, day, configuration).

Fig. 3 summarizes Tweak. Prior to deployment, Tweak requires a training phase for the twin neural network and a calibration process using data from the target domain from

all devices. Note that the calibration process can be repeated with different data without altering the twin neural network. The following subsections provide more information including background material on the twin network architecture, details regarding how the neural network is trained as well as specifics of the calibration and decision-making algorithms.

#### A. Deep Learning

This section focuses the discussion on metric learning and twin neural networks; a comprehensive treatment on deep learning is outside the scope of this work and additional information can be found in [15]. Tweak uses twin neural networks as open-set classifiers. While closed-set classifiers are designed to classify data with the assumption that they only encounter data instances from the classes observed during training, open-set classifiers can handle data from classes (i.e., devices) that are not seen during training [14]. We deal with data instances from these unseen classes by identifying them as not belonging to one of the known classes seen during training. While the open-set problem is more difficult, it is also more realistic and appropriate for device authentication when unknown devices are often encountered in deployment.

##### 1) Metric Learning and Twin Neural Networks

The standard approach for building a deep neural network (DNN) classifier is to train the model using a cross entropy loss with one output node for each class. The values of these output nodes are then used to determine the degree to which an input instance is predicted to belong to a particular class. Under this standard supervised learning setting, it is non-trivial to adapt the network to different domains such as a different receiver, day or configuration. Sophisticated domain adaptation methods for deep learning (e.g. [16], [17]) have been developed, but these methods are typically computationally intensive and require large amounts of data. In contrast, we want a lightweight domain adaptation process that can be quickly calibrated using a small number of labeled examples from the target domain.

To overcome these issues, our work makes use of deep metric learning [18] in which a DNN produces an embedding of a data instance in a  $K$ -dimensional latent space. Within this embedding space, data instances that are similar (e.g. from the same device) will be closer than those that are not similar (e.g. from different devices). Tweak uses a well known metric learning structure called a twin (or Siamese) neural network [19], [20]. These types of neural networks have been successfully used for applications where few examples are available such as signature verification [19] and facial similarity calculations [21]. Additionally, it has been shown that twin networks trained on one image dataset (e.g. Omniglot) and evaluated on an entirely different image dataset (e.g. MNIST) are able to generalize and maintain some level of performance [22]. However, they have only been applied directly to the problem of device fingerprinting based on RF signals in one other work [4].

Fig. 4 depicts a twin neural network, consisting of a pair of convolutional neural networks (CNNs) constrained to have identical weights. Each CNN accepts a single input instance and produces an output that corresponds to coordinates in a

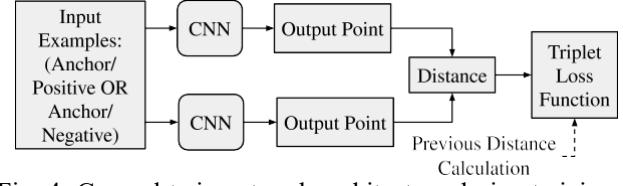


Fig. 4: General twin network architecture during training.

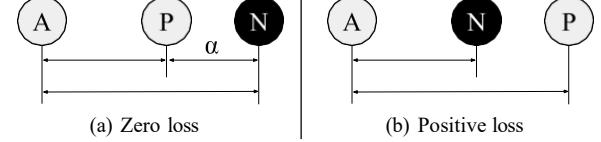


Fig. 5: Example triplet loss values for one triplet.

$K$ -dimensional latent embedding space ( $K$  being the number of output nodes). The distance between the two outputs in the embedding space is computed as a measure of similarity and can be used to train the network with a loss function. In practice, a trained twin network can be considered as a single DNN that produces an output point in the embedding space, instead of twin networks that produce a distance (since the networks have identical weights). Hence, using a twin network does not impose additional memory costs.

##### 2) Training the Twin Neural Network

Training a twin neural network requires a loss function that will cause data instances from different classes to be further apart in the output space and data instances from the same class to be closer together. A loss function that fits this criterion is the triplet loss function [23], which encourages reduced distances between ‘Anchor’ (A) and ‘Positive’ (P) examples, and increased distances between ‘Anchor’ (A) and ‘Negative’ (N) examples. The mathematical expression for this loss function is provided in Equation (1) and a visual depiction is provided in Fig. 5, where  $\alpha$  is a margin value (indicating that the distance between P and N has to be at least this much to matter), and  $f(\cdot)$  is the neural network mapping function.

$$\text{Loss} = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0) \quad (1)$$

Triplets of A, P, and N examples are provided to the network during training. We generate the triplets after examples from a mini-batch have already passed through the network but before the loss is calculated. Determining triplets after examples have passed through the network is advantageous because more difficult triplets can be strategically selected to help with training [23], where difficult triplets are those with high loss due to having A closer to N than to P.

Tweak uses the triplet loss function to train the twin network as shown in Fig. 3. Once this model has been trained on labeled data from a chosen domain, inputs from different transmitters should be mapped to outputs that are farther apart, and inputs from the same transmitter should be mapped to outputs that are closer together. At this point, the model is typically only capable of making decisions given two inputs that it can compare in terms of their distance in the latent space. To enable the trained twin network to make decisions

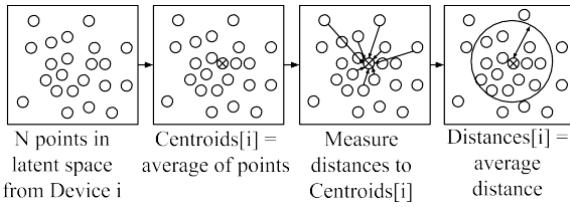


Fig. 6: Calibration for a single device, after  $N$  examples from Device  $i$  have passed through the trained twin network.

about a single input from a particular domain, a calibration process must be performed.

### B. Calibration and Decision Making Algorithms

#### Algorithm 1 Calibration

```

Centroids = [ ]
Distances = [ ]
Devices = { $D_1, D_2, \dots, D_k$ } each  $D_i = \{x_{i1}, x_{i2}, \dots, x_{iN}\}$ 

for all  $D_i \in Devices$  do
    Centroids[i] =  $\frac{1}{N} \sum_{j=1}^N f(x_{ij})$ 
    Distances[i] =  $\frac{1}{N} \sum_{j=1}^N \|f(x_{ij}) - Centroids[i]\|$ 
end for
Calibrated for  $D_i \in Devices$ 

```

The calibration process is performed by providing the trained network with labeled examples from all desired devices. Note that these examples may come from the training data if calibrating for the same domain as the training data. Algorithm 1 describes the calibration process and Fig. 6 illustrates it. We use the convention that boldface symbols refer to vectors and sets while non-boldface refer to scalars. For each device (class), we repeat the following process: given  $N$  examples from device  $i$  for calibration,  $x_{i1}, x_{i2}, \dots, x_{iN}$ , we first pass these examples through the trained neural network,  $f(\cdot)$ , to produce  $N$  corresponding outputs in the latent space. The first image in Fig. 6 shows the result of this step. Next, these  $N$  outputs are averaged to produce  $Centroids[i]$  (i.e. the centroid for device  $i$ ), which is shown in the second image of Fig. 6. This centroid is the first piece of calibration data that will be stored for device  $i$ .

Next, the distances between the centroid for device  $i$  and each of the  $N$  outputs that were used to generate the centroid are calculated. This is depicted in the third image of Fig. 6. These distances are averaged to produce  $Distances[i]$  (i.e. the average distance for device  $i$ ), which is represented by a circle with this distance value as its radius in Fig. 6. This distance is the second piece of calibration data that will be stored for device  $i$ . After the above process is repeated for every device, calibration is complete and the *Centroids* and *Distances* for the devices can be used to make decisions regarding new inputs using the decision algorithm to be described later.

This calibration process is designed to be lightweight as it only requires feeding data instances forward through the network and performing distance calculations, thus avoiding

re-training the network. Additionally, the parameter  $N$  represents the number of examples from each device used for calibration purposes. Altering  $N$  allows adjusting the amount of computation required for calibration as well as the amount of labeled data required. The calibration process could also be repeated with data from multiple domains for the same devices in order to produce a model capable of making decisions about data from these multiple domains, thereby producing multiple *Centroids* and *Distances* for each device.

---

#### Algorithm 2 Open-set Binary Decision

---

```

InputDeviceExamples = { $x_1, x_2, \dots, x_M$ }
InputPoint =  $\frac{1}{M} \sum_{j=1}^M f(x_j)$ 
for all  $D_i \in Devices$  do
    if  $dist(InputPoint, Centroids[i]) \leq Distances[i]$ 
    then
        Return: Decision = Admit
    end if
end for
Return: Decision = Reject

```

---

Once calibrated, the network is then used as an open-set classifier (Algorithm 2) to classify an input data instance as belonging to a known or unknown device. The algorithm begins by collecting  $M$  examples,  $x_1, x_2, \dots, x_M$ , from an input device, passing these examples through the trained network,  $f(\cdot)$ , and averaging the  $M$  outputs to produce *InputPoint*. Since this operation is not typical in deep learning models, it deserves some explanation. In the RF domain, the input data represents fractions of a second of real-time RF communication. Thus, it is plausible to collect  $M$  input examples quickly in order for the model to process them for the open-set classifier. Doing so allows for the construction of a less noisy *InputPoint* that is more representative of a particular device and less susceptible to the effects of outlier examples.

After *InputPoint* is computed, for each device  $i$  the algorithm calculates the distance between the *InputPoint* and *Centroids[i]* and compares this distance with *Distances[i]*. If this calculated distance is less than or equal to *Distances[i]*, the algorithm decides that the input comes from a known device, and returns the ‘Admit’ decision (i.e., device is authenticated). If none of the calculated distances are less than or equal to *Distances[i]*, then the algorithm decides the input came from an unknown device, and returns the ‘Reject’ decision (i.e., device is denied network access).

### III. EXPERIMENTAL SCENARIOS AND DATASETS

This section describes the testbed used for collecting the RF datasets and the experimental setups used for evaluating *Tweak* under three different portability scenarios.

#### A. Testbed

In order to evaluate *Tweak*, real RF data was collected using a testbed of devices. This testbed contains 25 almost identical PyCom IoT devices used as transmitters: 23 Lopy4 boards and 2 Fipy boards on top of 22 Pysense sensor shields, 2 Pytrack sensor shields, and 1 Pyscan sensor shield (pictured in Fig. 7c).

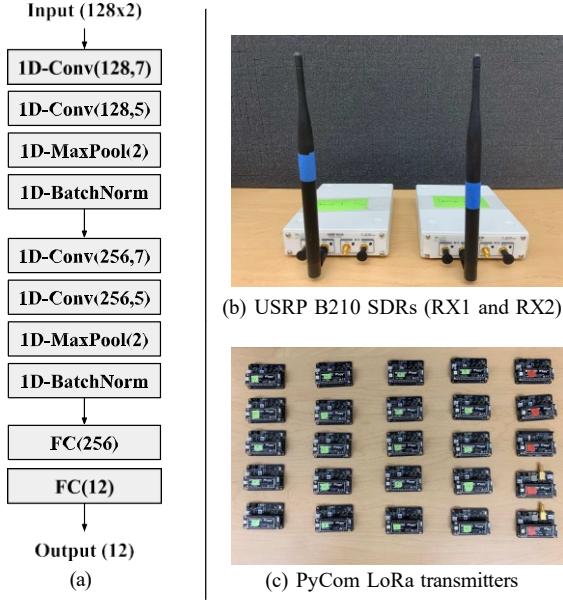


Fig. 7: (a) Twin network, (b) and (c) testbed hardware.

Portability	Env.	Multi-RX Collection	# TX	# RX	# Days	# TX Config.
Hardware	Indoor	Same Tx	25	2	1	1
Hardware	Outdoor	Same Tx	25	2	1	1
Hardware	Wired	Same Tx	25	2	1	1
Hardware	Indoor	Diff. Tx	25	2	1	1
Hardware	Outdoor	Diff. Tx	25	2	1	1
Hardware	Wired	Diff. Tx	25	2	1	1
Channel	Outdoor	N/A	25	1	5	1
Config.	Indoor	N/A	25	1	1	4

TABLE I  
Summary of the datasets used for evaluation.

The testbed also uses 2 USRP B210 SDRs (Software Defined Radios) as receivers to collect data (pictured in Fig. 7b).

All of the datasets collected using this testbed (outlined in the next section) use the following settings and processes unless otherwise stated: the PyCom transmitters were configured to transmit using the LoRa protocol, with a center frequency of 915 MHz, spreading factor of 7, and bandwidth of 125KHz. The USRP B210 receivers were configured to sample at a center frequency of 915 MHz, at a rate of 1M samples per second. Each transmission sends the same message, and lasts for 20s. This produces 20M complex-valued samples for each transmission received. This data was stored in raw-IQ format, using GNURadio to process the data.

Regardless of the environment in which data was collected, the transmitters and receivers were always placed about 5m apart (with the exception of ‘Wired’ data, which is collected by connecting transmitters to receivers by physical wire). All data collected using this testbed will be made publicly available.

### B. Experimental Scenarios

The collected LoRa RF datasets and the studied portability scenarios are summarized in Table I and described next.

Hardware portability dataset scenario. Six different datasets were collected to evaluate the hardware portability

Config.	Spreading Factor	BW	Tx Power	Coding Rate	Bit Rate
1	7	125kHz	20dBm	4/5	5470bps
2	8	125kHz	20dBm	4/5	3125bps
3	11	125kHz	20dBm	4/5	537bps
4	12	125kHz	20dBm	4/5	293bps

TABLE II  
Different LoRa transmitter configurations.

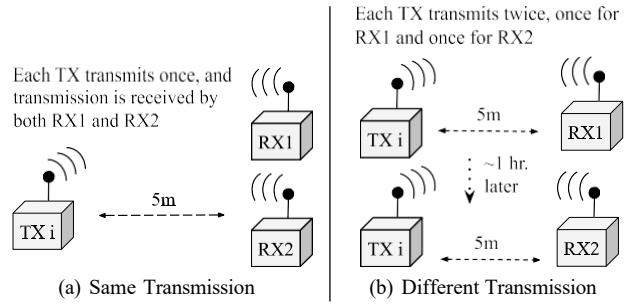


Fig. 8: Methods of collecting RF data with two receivers.

scenario. All of these datasets involve the same 2 receivers and 25 transmitters, and each contains 2 recorded transmissions from each transmitter: one at each receiver.

These six datasets differ in terms of the environment they were collected in as well as the ‘Multi-RX Collection’ method used. Datasets where the ‘Multi-RX Collection’ method is ‘Diff. Tx’ were collected by capturing two different transmissions from each device, one at each receiver as shown in Fig. 8b. Datasets where this method is ‘Same Tx’ were collected by capturing the same single transmission from each device at both receivers as shown in Fig. 8a.

Channel portability dataset scenario. This dataset includes 5 transmissions from each of the 25 transmitters, with each of the 5 transmissions collected on a different day. Our intent is to have this dataset represent transmissions under 5 different wireless channel conditions.

Protocol configuration portability dataset scenario. This dataset includes four transmissions from each of 25 transmitters, each of the four transmissions using a different configuration. The configurations amount to using different LoRa spreading factors and are detailed in Table II. Note that all other datasets use only Config 1.

## IV. EXPERIMENTAL RESULTS

We first describe the specific implementation of the twin network as well as the metrics to be used to evaluate Tweak. Then, we explain the results obtained by testing Tweak under the portability scenarios outlined in the previous section.

### A. Neural Network Implementation

The network used for this work is inspired by one of the straightforward CNNs described in [7] and is depicted in Fig. 7a. Note that this represents one of the two identical halves of the twin network. The network was implemented using the Python ‘PyTorch’ library and uses a total of four 1-D convolutional layers, making use of 1-D max-pooling and

batch normalization layers after each pair of convolutional layers. It also uses two fully-connected layers at its output, ending with 12 neurons. This produces a 12-dimensional output point for each input example. Leaky ReLU is used as the activation function throughout the network.

The inputs to the network are presented in the form of complex IQ (In-phase, Quadrature) samples, a common format for representing RF signals. For this work, a sequence length of 128 complex samples was considered as one input example. Additionally, two different input channels were considered, one for I samples and one for Q samples. Thus, the final input shape for the network is 2x128.

The triplet loss function is used to train the network with a margin value of 0.1, and the “difficult triplet selection” strategy described earlier is employed to select triplets for training. The optimizer chosen was Stochastic Gradient Descent (SGD) with Momentum set to 0.9. Batch size was set to 64, and the learning rate for each model was tuned over the range 1e-1 to 1e-6. The networks were allowed to train for 100 epochs and the model was saved at the best performing epoch.

For baseline comparison, a ‘vanilla’ network was defined using half of a twin network (Fig. 7a) and having 10 neurons in the final layer (where 10 is the number of transmitters used during testing). The same hyper-parameters described for the proposed network are used for training, but the ‘vanilla’ network uses cross-entropy loss instead of triplet-loss.

### B. Performance Evaluation Metrics

To assess Tweak’s performance, three metrics are used: (i) the averaged Area Under the Receiver Operating Characteristic (Avg. AUROC); (ii) the averaged True Positive Rate (Avg. TPR); and (iii) the averaged False Positive Rate (Avg. FPR). These three metrics are further explained next.

The TPR and FPR metrics are calculated using the exact decision making process defined in Algorithm 2 as,  $TPR = TP/(TP + FN)$  and  $FPR = FP/(FP + TN)$ , where:

- True Positive (TP) is the number of tested known devices that the model predicts as known (correctly)
- False Negative (FN) is the number of tested known devices that the model predicts as unknown (incorrectly)
- False Positive (FP) is the number of tested unknown devices that the model predicts as known (incorrectly)
- True Negative (TN) is the number of tested unknown devices that the model predicts as unknown (correctly)

These quantities are also shown visually in Fig. 9b.

The AUROC metric allows for measuring the performance of a binary classifier (i.e., classifying an input/device as known or unknown) without having to specify a particular threshold. Instead, decision scores are collected and the TPR and FPR are calculated for all possible thresholds on these scores. The AUROC is then defined as the area under the ROC curve resulting from plotting FPR vs. TPR for all thresholds. Fig. 9a depicts an example of this curve. Note that the open-set evaluations using AUROC do not strictly follow Algorithm 2 defined earlier, since this algorithm uses specific thresholds (the *Distances*). Instead, for models

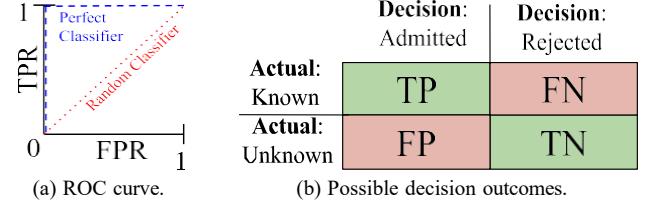


Fig. 9: Visualization of performance metrics.

using Tweak, the AUROC decision scores correspond to the minimum distance between *InputPoint* and a *Centroid*, and for the ‘vanilla’ models, the decision scores correspond to the maximum logit value for each example, which is commonly used in open set classification.

The reported evaluation metrics (AUROC, TPR and FPR) are averaged over 5 trials since performance can depend on which unknown devices are used for testing. For each trial, an equal number of examples were drawn from 5 random known devices and 5 random unknown devices.

### C. Evaluation Results

Tweak is evaluated in an open-set device authentication setting, which requires it to decide if inputs originated from a known (authorized) or unknown (unauthorized) device. All models evaluated in this section were trained and tested using data transmissions collected from the same 10 wireless IoT transmitters, with 75% of the data used for training and 25% used for testing (each transmission contains 156,250 examples of size 128x2). Test data from the other 15 IoT transmitters was used to represent the unknown/unauthorised devices. In all tests, the amount of calibration data used ( $N$  in Algorithm 1) was set to 10% of the size of the training data used to initially train the model (11,719 examples), and the number of input examples used to form the ‘input point’ ( $M$  in Algorithm 2) was set to 10. It is also worth noting here that on all AUROC results presented in this section, a dashed line is plotted at 0.5 to indicate the performance of a random classifier/authentication.

Additionally, for comparison to Tweak, ‘vanilla’ models are also evaluated in this section. Note that the ‘vanilla’ models were only evaluated using the AUROC metric.

#### 1) Hardware Portability

We begin by assessing Tweak’s ability in achieving hardware portability. For this, for each of the six studied hardware portability datasets listed in Table I, Tweak was trained on data collected at RX1. Then, Tweak is tested with data collected at both receivers, RX1 and RX2, while being: (i) calibrated with data collected at RX1, and (ii) calibrated with data collected at RX2 (receiver different from that used for collecting training data).

Figs. 10a-10c show the results of these tests for the ‘Diff. Tx’ method, where data captured by RX1 and RX2 for each transmitter is done during two separate transmissions. Figs. 10d-10f show the same results but for the ‘Same Tx’ method, where data is collected at RX1 and RX2 by capturing

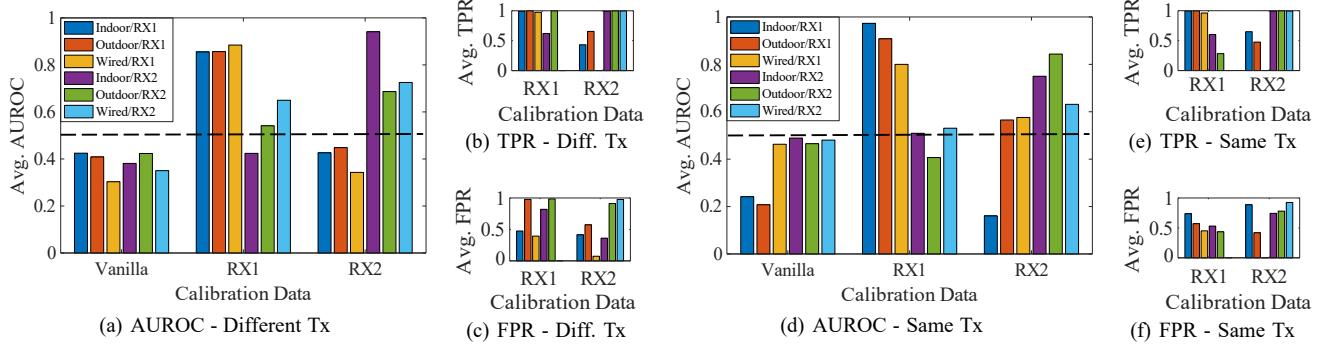


Fig. 10: Hardware Portability: Models trained on data collected at RX1.  $N = 10\%$  and  $M = 10$ .

the same transmission from each device. Note: models trained on RX2 data as opposed to RX1 data were also evaluated and showed similar results, and are omitted due to space limitations.

The main trend present for the AUROC results is that in most cases, better AUROC is achieved when testing on the same receiver for which the model is calibrated. This can be seen in Fig. 10a, for instance, where the results when calibrating/testing on RX1 data and the results when calibrating/testing on RX2 data show higher AUROC when compared to the other tests. It is also notable that the ‘vanilla’ models perform far worse than Tweak in terms of AUROC when the models that use Tweak are calibrated and tested on the same receiver. This is indicative of the potential for Tweak to achieve hardware portability for device authentication.

The TPR results shown in Figs. 10b and 10e provide even stronger evidence of the main trend that existed for the AUROC metric, where performance is high when the model is calibrated/tested on the same receiver. These results indicate that Tweak always achieves a high TPR, meaning it almost always correctly ‘admits’ examples from known devices.

In the FPR results, the trend is less clear. Ideally, models calibrated for a particular receiver would achieve a low FPR when tested on data from that receiver, but this is not always the case. For example, in Figs. 10c and 10f, it can be seen that there are several cases where a model is calibrated for data from a receiver, but still achieves a higher FPR when tested on data from that receiver than when it is tested with data from a receiver for which it is not calibrated. For a particular example, notice the last two bars of the tests for models calibrated with RX2 data in Fig. 10c. These are both tests on RX2 data, so good performance would be expected, and thus low FPR. Instead these tests have the highest FPR of any test for models calibrated to RX2 in the figure.

This seemingly counter-intuitive result can be explained by considering the meaning of FPR, and what exactly the model is attempting to do in the latent space during evaluation. Recall that when a model is calibrated using a particular set of data it forms a *Centroid* for each device in its latent output space. To achieve low FPR (which is desirable), the examples from unknown devices must form distinct ‘clusters’ in the latent

space far enough away from the *Centroids* so that the model can distinguish known from unknown.

In the case where the model is calibrated/tested on data from the same domain, the ‘clusters’ from both the known and unknown devices will likely be closer to the *Centroids* in the latent space than the case where the model is calibrated/tested with data from two different domains. This means that when testing on a different domain than calibration, the model will more easily be able to ‘reject’ examples from unknown devices (achieving a lower FPR), but will also have a more difficult time ‘admitting’ examples from known devices (also achieving a lower TPR). Thus, achieving a low FPR is generally more difficult when testing on the same domain as calibration.

It is also worth noting that even when the FPR produced by calibrating and testing on data from the same receiver is not the highest FPR, the result is still not as low as would be desired. For example, observe Fig. 10c. Notice that the FPR for the model calibrated and tested on RX2 data from an ‘Indoor’ environment is among the lowest for all models calibrated with RX2 data, but that it is still in excess of 0.25. This is indicative of an issue with the *Distances* chosen during calibration and used for open-set decision making. It indicates that these *Distances* may be too large, producing a much higher TPR as well as a relatively high FPR.

## 2) Channel Portability

Tweak’s portability across different channels is evaluated using channel portability dataset scenarios listed in Table I. For this, Tweak was first trained on data collected on Day 1, and then calibrated for each of the five days. Tweak was tested with data from all five days. The results are shown in Figs. 11a-11c. Note that this evaluation was also performed for a model trained on Day 2 data, and similar results were achieved, but omitted here due to space limitations.

First considering the AUROC results in Fig. 11a, there are a couple of interesting trends that are worth mentioning. From observation of this figure it is evident that high AUROC is achieved when a model is calibrated with data collected on one day and tested using data from the same day. For instance, the model calibrated with Day 2 data performs very well when tested with Day 2 data. This trend exists regardless of the calibration day. The lowest Avg. AUROC achieved when

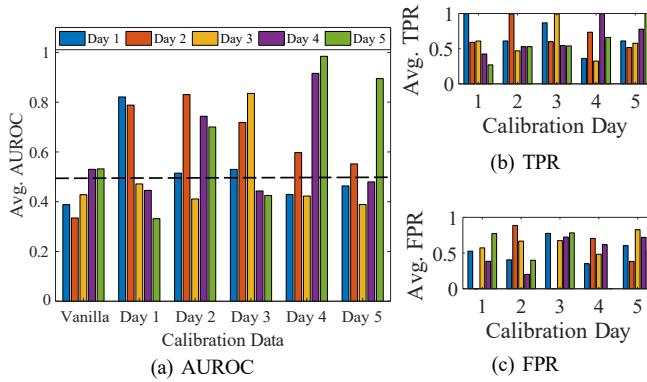


Fig. 11: Channel Portability: All models are trained on Day 1 data.  $N = 10\%$  and  $M = 10$ .

models are calibrated/tested on data from the same day is  $\sim 0.79$ . This trend is indicative of the ability of Tweak to achieve wireless channel portability for device authentication.

Another trend of note when looking at the AUROC results is that sometimes good performance is also achieved when testing using data from days other than the day used for calibration. For instance, in Fig. 11a, it can be seen that the model calibrated with Day 1 data also performs very well when tested with Day 2 data, and that the model calibrated with Day 4 data also performs very well when tested with Day 5 data. This trend is consistent, and is also present when the model is instead trained with data from Day 2. This could be indicative of some similarity between the wireless channel on these different days, but more work needs to be done to confirm this idea.

Moving to observe the TPR results in Figs. 11b, it can be seen that the desired trend is present. That is, TPR is high when a model is calibrated and tested on data from the same day. This indicates that Tweak is excellent at correctly ‘admitting’ examples that come from known devices.

Turning to the FPR results, it can be seen that the trend present in the ‘hardware portability’ evaluation is again present here. That is, the FPR results are less than ideal, since calibrating and testing with data from the same day does not always produce the lowest FPR. This can be attributed to the idea explained in the ‘hardware portability’ evaluation section.

### 3) Configuration Portability

To evaluate the configuration portability of Tweak, the configuration portability dataset in Table I was used to train two twin network models: one with only data using Config. 1 and one with only data using Config. 2. Each of these trained models was then calibrated for each of the four configurations. Finally, Tweak was tested with data from all four configurations.

The results when training on data collected while the transmitters use Config. 1 are shown in Figs. 12a-12c and those when the transmitters use Config. 2 are shown in Figs. 12d-12f. First observing the AUROC results, it can be seen that regardless of the training configuration, calibrating the model using Config. 1 or 2 results in the highest performance of the

model occurring when it is tested with the same configuration as calibration (which is expected). One example of this can be seen in Fig. 12a, where the model calibrated with Config. 1 data performs the best when tested with Config. 1 data, and the model calibrated with Config. 2 data performs the best when tested with Config. 2 data. This trend is in line with the expected trends that have been mentioned thus far.

A unique trend present here can be seen by observing the performance of the models when calibrated and tested with Config. 3 data. In these cases, the performance of the model is not the best when testing on the same domain as calibration. In particular, testing with Config. 4 produces better performance than testing with Config. 3. This is not the case however, when the model is trained with data from transmitters that use Config. 3 or Config. 4 (not pictured). In this case, the result is as expected, where calibrating and testing with Config. 3 data produces the best result of all tests for that calibration. This result could be attributed to the fact that generally Configs. 1 and 2 are more similar to each other and less similar to Configs. 3 and 4 (see Table II).

The TPR results for the models trained with either configuration are consistent with the desired trend. That is, a very high TPR is achieved when the model is calibrated and tested using data from the same configuration.

Similar to the other evaluations, the FPR results are less encouraging than those for AUROC and TPR. There are many instances where calibrating and testing using data from the same configuration results in the highest FPR among all of the tests for that calibration. This can be explained using the same line of thinking used in the ‘hardware portability’ evaluation. That is, achieving a low FPR is easier when calibration and testing are done with data from two different configurations. This conclusion is reinforced by the FPR results here, as generally the FPR is lower when testing on a configuration that is ‘more different’ from the one used for calibration.

Finally, the last evaluation performed in this section leverages the notion that Tweak could be used to calibrate for more than one configuration by performing multiple consecutive calibrations. To this end, the twin network model trained using Config. 2 data was calibrated using data from multiple configurations and tested on data from all four configurations. The results of these tests are shown in Fig. 13. The results here are encouraging for AUROC and TPR, but contain the same issues mentioned above when it comes to FPR results. There is also a notable drop in AUROC when calibrating for multiple configurations when compared to calibrating for a single configuration.

## V. CONCLUSION

This work proposes Tweak, a technique for creating a portable deep neural network for LoRa device authentication which can be calibrated quickly, using a small amount of labeled examples, to perform effectively on data from a domain other than the one it was originally trained with. It has been demonstrated through experimentation with a testbed of IoT devices that Tweak enables portability with respect to receiver

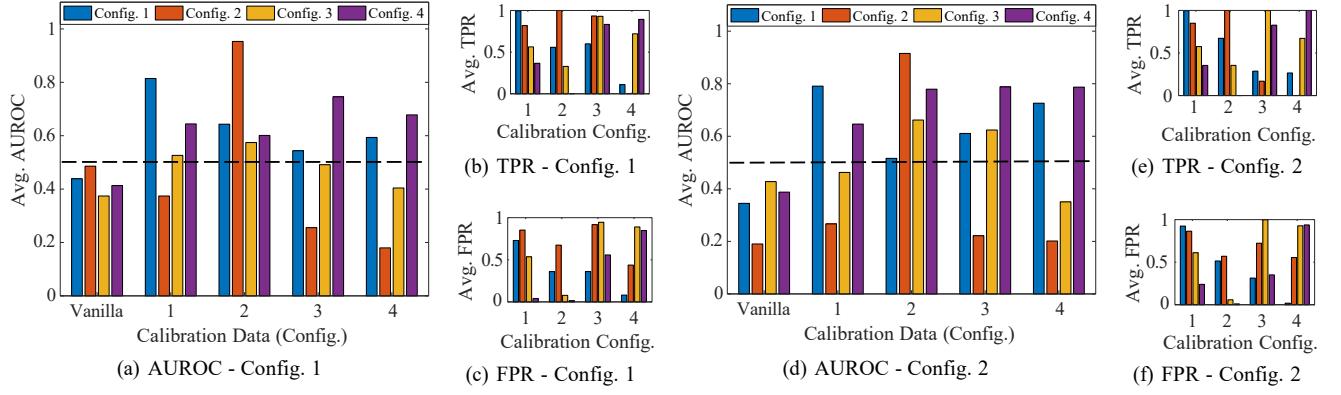


Fig. 12: Configuration Portability: Models trained on data using (a)-(c) Config. 1 and (d)-(f) Config. 2 and tested on data from transmitters using four different configurations.  $N = 10\%$  and  $M = 10$ .

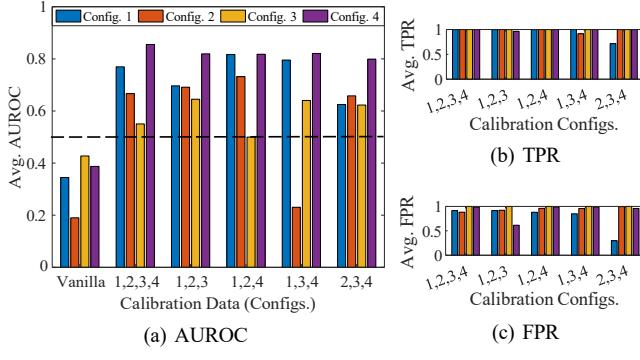


Fig. 13: Multi-Calibration Configuration Portability: Models trained on data using Config. 2 and calibrated with data from multiple configurations.  $N = 10\%$  and  $M = 10$ .

hardware, a change in the wireless channel due to the passage of time, or a change in the configuration of the transmitters.

## REFERENCES

- [1] IoTAnalytics, “State of IoT 2022: Number of connected IoT devices growing 18 to 14.4 billion globally,” <https://iot-analytics.com/number-connected-iot-devices/>, 2022, [Online; accessed 9-June-2022].
- [2] LoRa Alliance, “<https://lora-alliance.org/>.” [accessed 9-June-2022].
- [3] S. Rajendran and Z. Sun, “Rf impairment model-based iot physical-layer identification for enhanced domain generalization,” *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1285–1299, 2022.
- [4] G. Shen, J. Zhang, A. Marshall, and J. R. Cavallaro, “Towards scalable and channel-robust radio frequency fingerprint identification for lora,” *IEEE Tran. on Infra. Forensics and Security*, vol. 17, pp. 774–787, 2022.
- [5] Y. Li, X. Chen, Y. Lin, G. Srivastava, and S. Liu, “Wireless transmitter identification based on device imperfections,” *IEEE Access*, vol. 8, pp. 59 305–59 314, 2020.
- [6] A. Elmaghbub and B. Hamdaoui, “Lora device fingerprinting in the wild: Disclosing rf data-driven fingerprint sensitivity to deployment variability,” *IEEE Access*, vol. 9, pp. 142 893–142 909, 2021.
- [7] A. Al-Shawabka, F. Restuccia, S. D’Oro, T. Jian, B. Costa Rendon, N. Soltani, J. Dy, S. Ioannidis, K. Chowdhury, and T. Melodia, “Exposing the fingerprint: Dissecting the impact of the wireless channel on radio fingerprinting,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 646–655.
- [8] F. Restuccia, S. D’Oro, A. Al-Shawabka, M. Belgiovine, L. Angioloni, S. Ioannidis, K. Chowdhury, and T. Melodia, “Deepradioid: Real-time channel-resilient optimization of deep learning-based radio fingerprinting algorithms,” in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 51–60.
- [9] A. Al-Shawabka, P. Pietraski, S. B. Pattar, F. Restuccia, and T. Melodia, “DeepLoRa: Fingerprinting lora devices at scale through deep learning and data augmentation,” in *Proceedings of the International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2021, pp. 251–260.
- [10] S. Kokalj-Filipovic, L. Boegner, and R. D. Miller, “Practical training for rf fingerprinting of commercial transmitters at the edge,” in *2021 IEEE Globecom Workshops (GC Wkshps)*, 2021, pp. 1–6.
- [11] K. Merchant and B. Nousain, “Toward receiver-agnostic rf fingerprint verification,” in *2019 IEEE Globecom Workshops*, 2019, pp. 1–6.
- [12] G. Baldini, R. Giuliani, C. Gentile, and G. Steri, “Measures to address the lack of portability of the rf fingerprints for radiometric identification,” in *IFIP Conf. on New Tech., Mobility and Security*, 2018, pp. 1–5.
- [13] H.-H. Yang, T.-C. Wang, Y.-J. Yang, J.-Y. Wang, T. Y. Lin, S. Lai, and H.-C. Liu, “Improvement of radio frequency fingerprint portability for wi-fi adaptor identification,” in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–2.
- [14] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, and T. E. Boult, “Toward open set recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013.
- [15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [16] G. Wilson and D. J. Cook, “A Survey of Unsupervised Deep Domain Adaptation,” *ACM Transactions on Intelligent Systems and Technology*, vol. 11, no. 5, pp. 51:1–51:46, Jul. 2020.
- [17] M. Wang and W. Deng, “Deep visual domain adaptation: A survey,” *Neurocomputing*, vol. 312, pp. 135–153, Oct. 2018.
- [18] B. Ghojogh, A. Ghodsi, F. Karray, and M. Crowley, “Spectral, probabilistic, and deep metric learning: Tutorial and survey,” jan 2022, arXiv: 2201.09267. [Online]. Available: <http://arxiv.org/abs/2201.09267>
- [19] J. Bromley, I. Guyon, Y. LeCun, E. Saïckinger, and R. Shah, “Signature verification using a “siamese” time delay neural network,” in *Advances in Neural Information Processing Systems 6*, 1994, pp. 737–744.
- [20] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *IEEE Conf. on Comp. Vision and Pattern Recognition*, vol. 1, 2005, pp. 539–546.
- [21] H. Wu, Z. Xu, J. Zhang, W. Yan, and X. Ma, “Face recognition based on convolution siamese networks,” in *Int’l Congress on Image and Signal Processing, BioMedical Engineering and Informatics*, 2017, pp. 1–5.
- [22] G. Koch, R. Zemel, R. Salakhutdinov *et al.*, “Siamese neural networks for one-shot image recognition,” in *ICML workshop*, vol. 2. Lille, 2015.
- [23] D. Thapar, G. Jaswal, A. Nigam, and V. Kanhangad, “Pvsnet: Palm vein authentication siamese network trained using triplet loss and adaptive hard mining by learning enforced domain specific features,” in *2019 IEEE 5th International Conference on Identity, Security, and Behavior Analysis (ISBA)*, 2019, pp. 1–8.