PRONTO: Preamble Overhead Reduction with Neural Networks for Coarse Synchronization

Nasim Soltani, Debashri Roy, and Kaushik Chowdhury
Electrical and Computer Engineering Department, Northeastern University, Boston, MA
{soltani.n, d.roy}@northeastern.edu, krc@ece.neu.edu

***Abstract*—In IEEE 802.11 WiFi-based waveforms, the receiver performs coarse time and frequency synchronization using the first field of the preamble known as the legacy short training field (L-STF). The L-STF occupies upto 40% of the preamble length and takes upto 32 μs of airtime. With the goal of reducing communication overhead, we propose a modified waveform, where the preamble length is reduced by eliminating the L-STF. To decode this modified waveform, we propose a neural network (NN)-based scheme called PRONTO that performs coarse time and frequency estimations using other preamble fields, specifically the legacy long training field (L-LTF). Our contributions are threefold: (i) We present PRONTO featuring customized convolutional neural networks (CNNs) for packet detection and coarse carrier frequency offset (CFO) estimation, along with data augmentation steps for robust training. (ii) We propose a generalized decision flow that makes PRONTO compatible with legacy waveforms that include the standard L-STF. (iii) We validate the outcomes on an over-the-air WiFi dataset from a testbed of software defined radios (SDRs). Our evaluations show that PRONTO can perform packet detection with 100% accuracy, and coarse CFO estimation with errors as small as 3%. We demonstrate that PRONTO provides upto 40% preamble length reduction with no bit error rate (BER) degradation. We further show that PRONTO is able to achieve the same performance in new environments without the need to re-train the CNNs. Finally, we experimentally show the speedup achieved by PRONTO through GPU parallelization over the corresponding CPU-only implementations.**

***Index Terms*—CFO estimation, packet detection, LTF signal, IEEE 802.11, NN-based receiver.**

I. Introduction

The ever-increasing demand for wireless spectrum has led to transformative breakthroughs in the use of massive channel bandwidths (e.g., 2 GHz channels in the 57-72 GHz band) [1], massive multiple input multiple output (MIMO) (with several dozens of antenna elements) [2], [3], and physical layer signal processing innovations (e.g., channel aggregation [4]). All of these methods involve assumptions of availability of specialized hardware, often at the cutting edge of systems design. As opposed to these, in this paper, we propose an approach called PRONTO that reduces the occupancy time of the wireless channel in WiFi networks by shortening packet preambles. Since preambles are present in *every* transmitted packet, even a modest reduction results in a significant cumulative benefit over time. PRONTO is carefully designed to be backwards compatible with legacy WiFi waveforms for general purpose adoption and coexistence with standards-compliant access points (APs) deployed today.

• **Challenges in shortening the preamble:** A typical WiFi packet is composed of a multi-field preamble and data payload.

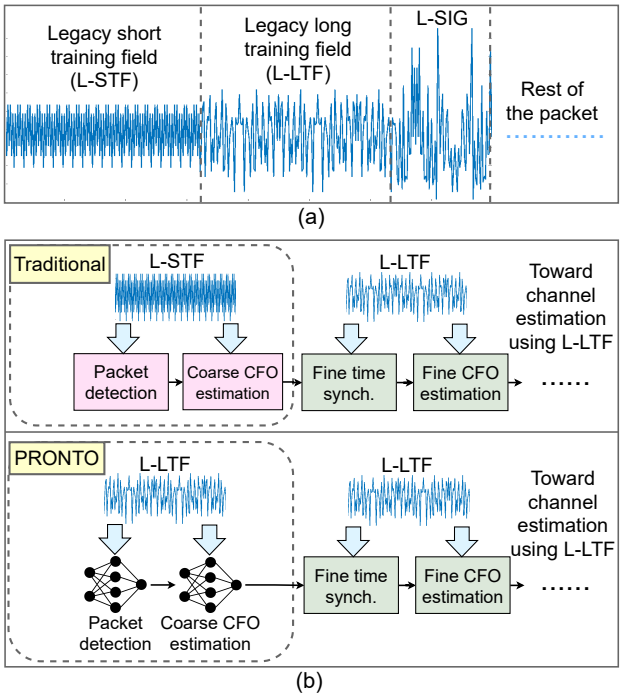

Fig. 1: (a) IEEE 802.11 OFDM-family preamble structure. (b) Traditional packet detection and coarse CFO estimation with L-STF, and PRONTO packet detection and coarse CFO estimation without L-STF.

The preamble is used in many tasks, including establishing synchronization between the transmitter and the receiver by removing relative time/frequency offsets. As shown in Fig. 1(a), the preamble in orthogonal frequency division multiplexing (OFDM) family of IEEE 802.11 waveforms starts with the legacy short training field (L-STF) followed by the legacy long training field (L-LTF) that have bits arranged in pre-set configurations with short and long periods, respectively. First, the L-STF is used to detect the start of the packet (i.e., coarse time synchronization) and then for coarse carrier frequency offset (CFO) estimation. Following this, the L-LTF is used for fine time synchronization and fine CFO estimation. The total CFO is the sum of coarse and fine CFOs, which is later compensated on the whole received packet. Traditional time and frequency synchronization steps are shown in Fig. 1(b). PRONTO revisits the fundamental structure of the WiFi packet by answering the following question: *Can the L-STF be completely eliminated from the preamble, without any adverse impact on decoding?*

• **PRONTO approach using deep learning:** In the absence of part of the preamble, the L-STF, PRONTO must extract relevant information from the remaining fields. Towards this goal, it leverages deep learning using convolutional neural networks (CNNs), which have already been demonstrated to be remarkably capable in physical layer signal classification problems [5] such as modulation recognition [6]–[11] and RF fingerprinting [12]–[16]. In all these problems, CNNs recognize patterns embedded within sequences of in-phase and quadrature (I/Q) samples. Classical methods that perform packet detection and coarse CFO estimation rely on known specific preamble patterns of fixed periods. As opposed to this, once trained, CNNs are able to detect a pattern that is present *somewhere* within their input sequences, regardless of its short or long period, and associate each input with a category (classification) or value (regression). In PRONTO, we leverage this ability by training custom-designed CNNs using L-LTF signals (that follow the now eliminated L-STF). *Our hypothesis here is that PRONTO should be able to predict both the start of packet and the coarse CFO, with only the L-LTF as input to the trained CNNs* (Fig. 1(b)).

• **PRONTO distinction from previous work:** Reducing preamble length in WiFi systems is a real challenge studied before by the related literature. [17] reduces preamble length by dropping a few training symbols and using a complete iterative receiver algorithm. They achieve preamble length reduction at the expense of less accurate channel estimation and higher bit error rate (BER). [18] designs a new shorter WiFi preamble by superimposing different fields in the legacy preamble. However, their method is not compatible with conventional WiFi receivers. PRONTO shifts the burden of packet detection and coarse CFO estimation to CNNs without imposing BER degradation, while the rest of the receiver chain remains the same. We note that our work is distinct from others that use neural networks as an alternative to the legacy processing blocks, but retain the same inputs and outputs as the latter. Applications of such uses of neural networks exist for channel estimation [19]–[22], signal demapping [23], [24], and decoding [25], [26]. Among these works are [27], [28], and [29] that detect packets and estimate CFOs using L-STF signals, similar to the legacy processing blocks. PRONTO is different from [29] that models CFO estimation using L-STF signals as a classification problem, where one of the pre-defined classes is chosen as the CFO category instead of predicting an exact CFO value. PRONTO is distinct from [27], [28], where packet detection using L-STF signals is modeled as a regression problem that yields upto $\sim$8% false alarm rate. In terms of overall objectives, our work comes close to prior research on new waveforms, such as removing pilots in orthogonal frequency division multiplexing (OFDM) symbols [30]. However, approaches like [30] fundamentally change the receiver processing flow. As opposed to this, PRONTO is also designed with the goal of maintaining backward compatibility with legacy waveforms. Through a decision logic included alongside PRONTO, the process-flow seamlessly switches between full preamble-enabled packets (i.e., with L-STF) or PRONTO-capable waveforms (i.e., L-STF missing).

We summarize our contributions as follows:

- We propose a modified version of the IEEE 802.11 OFDM waveforms where the preamble length is reduced by eliminating the first field (i.e., L-STF). To decode the modified waveform, we propose a deep-learning-based scheme called PRONTO that uses CNNs to detect the packet and estimate coarse CFO using L-LTF. We parameterize our method to be adaptable to signals with different bandwidths, and further propose a generalized decision flow to make the proposed approach compatible with standard waveforms.
- The first objective of PRONTO is to perform packet detection as a classification task that returns the start index of the packet, if a part of L-LTF signal exists in the input. We propose a robust training method by artificially shifting L-LTF signal and injecting periods of noise in the training set through a data augmentation step. The CNN trained on this large variety of inputs can detect L-LTF patterns with 100% accuracy in different noise levels.
- The second objective of PRONTO is to perform coarse CFO estimation, for which we train a regressor CNN to return a coarse CFO value for each detected L-LTF. We propose another data augmentation step to dynamically augment the training set by imposing different CFO values to each training signal. The CNN trained on this large variety of training data, is able to predict coarse CFO values in L-LTFs with errors as small as 3%.
- We validate PRONTO on two over-the-air (OTA) WiFi datasets with multiple SNR levels. We demonstrate that PRONTO causes no degradation in BER compared to the traditional WiFi receiver.
- We provide computation complexity for PRONTO compared to traditional (i.e., `MATLAB`-based) algorithms for packet detection and coarse CFO estimation, as well as run-time on server and edge platforms. We show that PRONTO packet detection and coarse CFO estimation (if properly parallelized on GPUs) can perform upto 2053x and 1.92x faster computation, respectively, compared to their traditional `MATLAB`-based counterparts.

The rest of the paper is organized as follows. Section II describes the legacy methods for packet detection and coarse CFO estimation using L-STF. Section III explains PRONTO scheme in details, as well as the decision flow for backward compatibility. Section IV describes the datasets and the deep learning setup used to evaluate PRONTO system. Section V presents the results and Section VI concludes the paper.

II. Background and Preliminaries

In IEEE 802.11 OFDM waveforms, the standard preamble starts with the L-STF. L-STF contains 10 repetitions of a training symbol with length η. Each training symbol contains one period of a specific pattern, and its length η directly depends on the FFT length $\mathcal{N}$, as $\eta = \frac{1}{4}\mathcal{N}$ [31]. Consequently, the length of full L-STF is $10\eta = \frac{10}{4}\mathcal{N}$. L-STF is modulated with binary phase shift keying (BPSK) modulation. It is not scrambled and has no channel encoding. L-STF is traditionally used for packet detection and coarse CFO estimation, due to its good correlation properties.

A. Packet Detection with L-STF

For detecting the packet, the classical method needs the full L-STF signal and a threshold β, which is a number in the range [0,1], typically close to 1. The classical method finds the start index of the packet by performing an auto-correlation between a portion of L-STF referred to as $\zeta = \text{L-STF}\,(0 : 10\eta - \eta - 1)$, and its delayed version $\zeta_{\text{delayed}} = \text{L-STF}\,(\eta : 10\eta - 1)$, where $\eta = \frac{1}{4}\mathcal{N}$.

Next, the complex correlation, $\rho^{(\tau)}$, between ζ and ζ_{delayed} in time τ is computed as (1), where * is the notation for complex conjugate.

$$\rho^{(\tau)} = \sum_{i=0}^{\eta-1} \zeta(i+\tau) \times \zeta^{*}_{\text{delayed}}(i+\tau+\eta) \tag{1}$$

The energy vector $E^{(\tau)}$ in the correlation window is calculated as in (2).

$$E^{(\tau)} = \sum_{i=0}^{\eta-1} |\zeta_{\text{delayed}}(i+\tau+\eta)|^2 \tag{2}$$

Then, $\rho^{(\tau)}$ is normalized through dividing it by $E^{(\tau)}$ as in (3).

$$\rho^{(\tau)}_{\text{norm.}} = \frac{|\rho^{(\tau)}|^2}{E^{(\tau)} \times E^{(\tau)}} \tag{3}$$

In the normalized correlation vector, $\rho^{(\tau)}_{\text{norm.}}$, we record the indices of the elements that are greater than β as vector I, and the number of elements (peaks) as n_ρ. According to the 802.11 standard, the first peak shows the start of the packet. However, `MATLAB` implementation performs an additional step in the function `wlanPacketDetect` [32] based on [33], to ensure the detected pattern is actually L-STF. In this step: First, the number of detected peaks are checked to be larger than or equal to 1.5 times the training symbol length ($n_\rho \geq 1.5 \times \eta$). Second, the sum of index distances of peaks from the first peak should be smaller than 3 times the symbol length ($\sum_{i=1}^{\eta}(I_i - I_0) < 3 \times \eta$). These criteria ensure that sufficient number of correlation peaks are detected and they are not too distant from each other. In this case, $I_0 - 1$ is returned as the coarse start time index of the packet.

B. Coarse CFO Estimation with L-STF

The coarse CFO is derived from the phase of complex conjugate of a portion of the L-STF, multiplied by a second portion of the L-STF [33]. The time-domain method for coarse CFO estimation, as implemented in `MATLAB` function `wlanCoarseCFOEstimate` [34], needs 9 short training symbols for coarse CFO estimation. These 9 symbols are chosen with a distance of 1 symbol from each other. The L-STF (consisting of $10\times\eta$ short training symbols) starts with a default guard interval of $\mathcal{G} = \frac{3}{4}\eta$, and these samples are considered as offset from the beginning of the signal. After $\mathcal{G}$, two portions of the L-STF, $\chi = \text{L-STF}\,(\mathcal{G} + (0 : 8 \times \eta - 1))$ and $\xi = \text{L-STF}\,(\mathcal{G} + (0 : 8 \times \eta - 1) + \eta)$ each with length of 8 training symbols are separated and composed.

Next, coarse CFO is calculated using the phase of multiplication of complex conjugate of χ multiplied by ξ as shown in (4), where $\phi(x) = \arctan\,(\frac{\text{Imag}(x)}{\text{Real}(x)})$ and f_s is sampling frequency in Hz.

$$\text{CFO}_{\text{coarse}} = \frac{\phi(\chi^{*} \times \xi) \times f_s}{2\pi \times \eta} \tag{4}$$

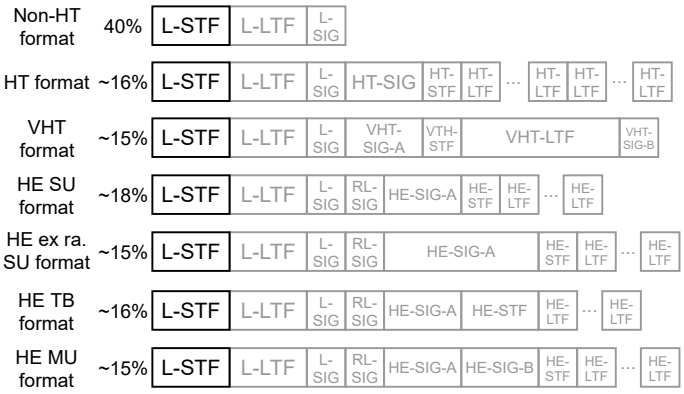

Fig. 2: Different IEEE 802.11 preamble formats and the percentages that the L-STF occupies in each preamble [35].

C. Eliminating the L-STF

The L-STF occupies upto 40% of the preamble and is considered as an integral component of existing and emerging standards of IEEE 802.11 WiFi, as represented in Fig. 2.

Apart from the L-STF, the legacy long training field (L-LTF) plays a key role in fine time synchronization, fine CFO estimation, and channel estimation. Hence, it is also included in all the preamble formats seen in Fig.2. However, the L-LTF has a different pattern with longer training symbol period, and cannot substitute for the L-STF using classical signal processing blocks. In this regard, we propose to use custom-designed neural networks to exploit the intrinsic patterns within the *L-LTF symbols* for packet detection and coarse CFO estimation. This obviates the need for L-STF and suggests that it could be completely removed from the preamble.

III. PRONTO System Description

In this section, we present the details of our proposed PRONTO framework that performs packet detection and coarse CFO estimation without the L-STF. First, we discuss L-LTF extraction which is used to compose the training, validation, and test sets for PRONTO. Then, we explain the CNN-based PRONTO modules for packet detection and coarse CFO estimation, respectively. Finally, we propose a generalized decision flow to make PRONTO compatible with coarse synchronization of standard waveforms.

A. L-LTF Extraction

Consider a PRONTO-enabled waveform that does not contain the L-STF. In this case, PRONTO buffers the received signal in I/Q format and streams these samples to the packet detection module. The objective of this module is to detect potential L-LTF signals that may be embedded somewhere within the streaming samples. PRONTO uses CNNs for both packet detection and coarse CFO estimation, therefore we first describe the procedure for training these models. To

setup a training/test pipeline using an OTA standard WiFi dataset, we need to create the labeled training/test datasets by extracting coarsely time-synchronized (detected) L-LTF signals denoted as $X(t)$s. To do this, the start of each packet in the standard dataset is detected through the traditional method (Section II-A), and depending on the waveform in use we extract L-LTF signal, $X(t)$, with known length and known start index, l, in the standard waveform. For the input data provided to the PRONTO coarse CFO estimation module, we label each $X(t)$ with the coarse CFO, f, for the associated packet estimated through the traditional method using the corresponding L-STF (Section II-A). The dataset of $X(t)$ signals recorded and labeled in this manner is later partitioned into training, validation, and test sets.

B. PRONTO Packet Detection Module

In the first module of PRONTO scheme, we propose a CNN-based solution to detect packets by detecting their L-LTF signals. Our solution can be generalized to detect any other types of signals (e.g., proprietary L-LTF sequences with customized patterns and periods) as long as a specific pattern exists within them. Since the role of packet detection module is to find the start index of a specific pattern in a sequence of time domain samples with *discrete* indices, we model packet detection as a *multi-class classification* problem, which we solve using a deep CNN. The input and output dimensions of the CNN, and the details of data flow in the packet detection module are shown in Fig. 3 and explained in the following.

CNN Input Dimensions. Inputs of the packet detector CNN are sequences of length $\mathcal{L}$ that contain shifted versions of the L-LTF signal, $X(t)$, in the time domain. To determine the size of the input sequence, we start by studying the L-LTF length. In an OFDM system, the L-LTF signal consists of 2.5 long training symbols, each with length equal to the FFT length, $\mathcal{N}$. The smallest input sequence with length $\mathcal{L}$ to fit one full L-LTF signal (i.e., start index = 0 with respect to the input sequence) is calculated as $\mathcal{L} = 2.5 \times \mathcal{N}$. We prepare the inputs for the CNN by separating their real and imaginary components and forming inputs of dimensions $(\mathcal{L}, 2)$.

CNN Output Dimensions. To model packet detection as a classification problem, we map *detection* or *no detection* of a packet to certain classes. Each input sequence with length $\mathcal{L}$ is classified as detection classes, if a portion of L-LTF signal, $X(t)$, exists within it. In cases where a portion of $X(t)$ is present, we associate different classes with different start indices of $X(t)$ with respect to the input with length $\mathcal{L}$. We denote the start index (shift) of $X(t)$ in the input sequence of length $\mathcal{L}$, as k, which ranges from 0 to a specific number K (i.e., $k \in \{0, ..., K\}$). For the CNN to be able to distinguish an L-LTF pattern from a generic pattern of bits, there should be at least one period of the L-LTF in the input. Therefore, K is defined as the largest L-LTF shift in the range [0, $\mathcal{L}$-1], wherein at least $\mathcal{N}$ samples of L-LTF are in the input sequence of length $\mathcal{L}$. K is calculated as $K = \mathcal{L} - \mathcal{N}$.

We associate each L-LTF shift k ($k \in \{0, ..., K\}$) with one class, and hence, the set of classes is given as $\mathbb{I}^{\text{PD}} = \{0, \cdots, K\}$, where $\text{len}(\mathbb{I}^{\text{PD}}) = (K + 1)$. Apart from these $K + 1$ classes where the L-LTF pattern can be detected in the input, there is one additional class that includes all the cases where the L-LTF pattern cannot be recognized. This class includes cases where L-LTF exists in the input but it cannot be detected because $k > K$, as well as the cases where the input does not contain any part of the L-LTF. Hence, we revise the set of classes as $\mathbb{I}^{\text{PD}} = \mathbb{I}^{\text{PD}} \bigcup \{(K + 1)\}$, and the total number of classes are updated to $\text{len}(\mathbb{I}^{\text{PD}}) = (K + 2)$. Finally, the basis vector (or one hot representation) of the class set $\mathbb{I}^{\text{PD}}$ is represented as $Y^{\text{PD}} \in \{0, 1\}^{\text{len}(\mathbb{I}^{\text{PD}})}$.

Fig. 4 shows an example of defining the classes with L-LTF of length $\mathcal{L} = 160$, that occurs in 5 MHz, 10 MHz, and 20 MHz bandwidths. In Fig. 4(a), the start of L-LTF, k, in the input sequence is shifted between 0 and K to create $K + 1$ classes, where the L-LTF pattern can be detected in the input, giving rise to classes indexed as 0 to 96. Fig. 4(b) shows different cases where the L-LTF pattern cannot be detected in the input, and hence, the packet is not detected. All these cases are labeled as class 97. Therefore, total number of classes for $\mathcal{L} = 160$ is calculated as 98.

Data Augmentation. We load to the memory the L-LTF signals, $X(t)$s, of the training set, batch-by-batch. Next, we pass each training $X(t)$ through a *data augmentation* function for packet detection, denoted as $F^{\text{PD}}_{\text{aug.}}(.)$, to dynamically generate different inputs and their corresponding true classes for the packet detector CNN. The goal of the data augmentation block is to artificially create different shift variations in the training input to the CNN, so that the trained packet detector CNN can detect L-LTF signals in different settings with a variety of time shifts and added noise. In this regard, the augmentation function performs a series of steps including shifting the input L-LTFs, as well as inserting noise and random data portions along the shifted L-LTF, to increase resiliency of the classifier, especially for unseen channels and conditions.

First, the mean of $X(t)$ is calculated as $\mu_{X(t)}$. Second, an empty sequence of $X^{\text{PD}}_{\text{aug.}}(t)$ is created with the same dimensions as those of $X(t)$. Third, the L-LTF signal $X(t)$ is shifted k indices ($k \in \{0, ..., K\}$) and inserted in $X^{\text{PD}}_{\text{aug.}}(t)$ (that has length $\mathcal{L}$), as shown in Fig. 4. Fourth, the power of $X(t)$ is calculated as $P_{X(t)} = \frac{1}{\mathcal{L}}|X(t)|^2$, and a random vector with random length in range [0,k], is drawn from complex Gaussian distribution with mean=$\mu_{X(t)}$ and variance=$P_{X(t)}$. This vector emulates the potential random data with the same power as L-LTF, and is inserted in $X^{\text{PD}}_{\text{aug.}}(t)$ alongside the shifted $X(t)$. Fifth, the power of noise $P_{N(t)}$ in the L-LTF signal is calculated, and if there are empty periods left in $X^{\text{PD}}_{\text{aug.}}(t)$, they are filled with random values drawn from complex Gaussian distribution with mean $\mu_{X(t)}$ and variance $P_{N(t)}$. The function $F^{\text{PD}}_{\text{aug.}}(.)$ returns $X^{\text{PD}}_{\text{aug.}}(t)$ as well as the shift of L-LTF, k, as the true class index, as shown in (5). The data augmentation block is highlighted in Fig. 3.

$$X^{\text{PD}}_{\text{aug.}}(t)\ ,\ k = F^{\text{PD}}_{\text{aug.}}(X(t),\ P_{N(t)},\ P_{X(t)}) \tag{5}$$

RMS Normalization. At this stage our augmented signal $X^{\text{PD}}_{\text{aug.}}(t)$ contains the shifted $X(t)$ along with periods of data and noise with different amplitudes. In order for the packet detector CNN to be able to detect signals in different environments with different amplitudes, we perform a root

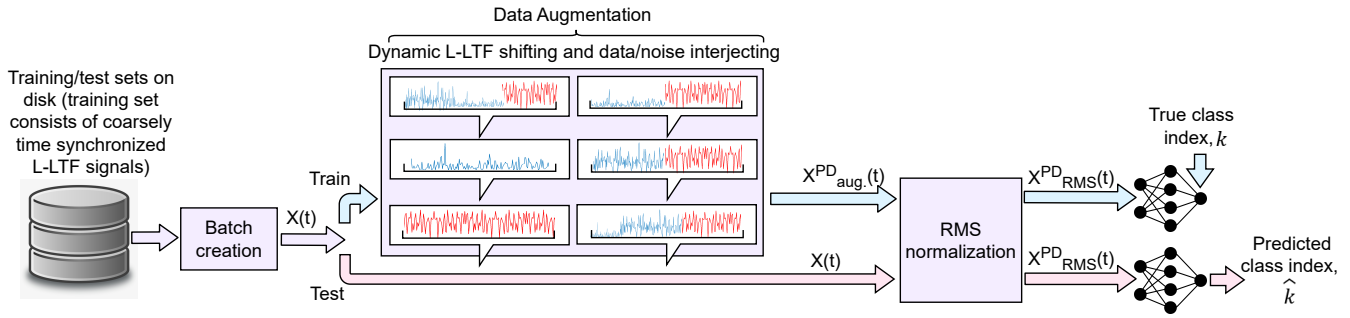

Fig. 3: PRONTO training/test pipeline for packet detection as a classification problem, with dynamic L-LTF shifting block in the training path.

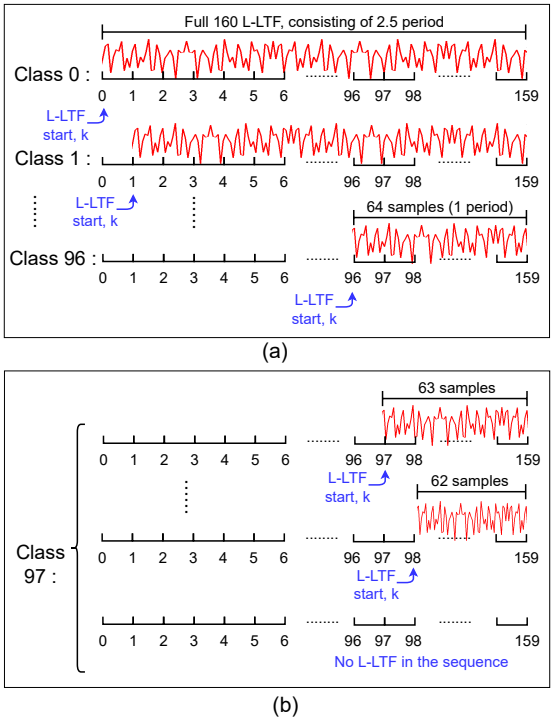

Fig. 4: The position of L-LTF in the 160 sample NN input for different packet detection classes. (a) Classes 0 to 96 correspond to the index of packet start, where we have 1 period of L-LTF signal in the sequence. (b) Class 97 shows different cases where less than a period of L-LTF exists in the sequence and the L-LTF pattern cannot be detected.

mean square (RMS) normalization for each augmented signal $X^{\text{PD}}_{\text{aug.}}(t)$, as in (6).

$$X^{\text{PD}}_{\text{RMS}}(t) = \frac{X^{\text{PD}}_{\text{aug.}}(t)}{\sqrt{\frac{1}{\mathcal{L}}\sum |X^{\text{PD}}_{\text{aug.}}(t)|^2}} \tag{6}$$

The RMS normalization block that follows the data augmentation block is shown in Fig. 3.

CNN Modeling. Recall that real and imaginary components of $X^{\text{PD}}_{\text{RMS}}(t)$ are separated to form an input of dimensions $(\mathcal{L}, 2)$, which is used for training and testing the CNN along with one hot representation, $Y^{\text{PD}} \in \{0,1\}^{\text{len}(\mathbb{I}^{\text{PD}})}$, of the true class index. The proposed CNN-based classifier predicts the occurrence probability of each class, as in (7), where γ is *Softmax* and $F^{\text{PD}}_{\theta}(.)$ denotes the classifier CNN.

$$\hat{Y}^{\text{PD}} = \gamma\left(F^{\text{PD}}_{\theta}(X^{\text{PD}}_{\text{RMS}}(t))\right) \qquad F^{\text{PD}}_{\theta} : \mathbb{R}^{\mathcal{L}\times 2} \mapsto \mathbb{R}^{\text{len}(\mathbb{I}^{\text{PD}})} \tag{7}$$

As shown in Fig. 3, we train the CNN with $X^{\text{PD}}_{\text{RMS}}(t)$ using *catergorical cross-entropy* loss for several hundreds of epochs, to create millions of variations in the L-LTF pattern, as the shifts and data and/or noise chunks happen randomly for each $X(t)$ in each epoch. During the test phase, we do not need the augmentation block described in (5), as the sampled wireless input by nature resembles the augmented L-LTF sequences in Fig. 3. Therefore, we only process the test sequences through the RMS normalization block in (6) and feed them to the trained CNN model.

The predicted class index, $\hat{k}$, is determined using the probability vector, $\hat{Y}^{\text{PD}}$, as in (8).

$$\hat{k} = \underset{\hat{k}\in\{0,\ldots,\,\text{len}(\mathbb{I}^{\text{PD}})-1\}}{\arg\max} \; (\hat{Y}^{\text{PD}}) \tag{8}$$

C. PRONTO Coarse CFO Estimation Module

The second module of PRONTO is designed to estimate coarse CFO using the L-LTF signals, once a packet is detected in the system. We model coarse CFO estimation as a *regression* problem where the intrinsic properties within the L-LTF patterns are exploited by a CNN to estimate the coarse CFO (see Fig. 5).

CNN Input and Output Dimensions. The inputs to the coarse CFO estimator CNN are batches of L-LTF signals, $X(t)$s, of dimensions $(\mathcal{L}, 2)$ where I and Q are passed through 2 separate channels to form the second dimension. Since we consider coarse CFO estimation as a regression problem, the label space of the PRONTO CFO estimator CNN converges to a single valued prediction represented by 1 neuron ($\text{len}(\mathbb{I}^{\text{CFO}}) = 1$).

RMS Normalization. To start with training, validation, and test phases, we normalize each L-LTF signal, through RMS normalization as in (9).

$$X^{\text{CFO}}_{\text{RMS}}(t) = \frac{X(t)}{\sqrt{\frac{1}{\mathcal{L}}\sum |X(t)|^2}} \tag{9}$$

The RMS normalization step in the training/test pipeline is shown in Fig. 5.

Data Augmentation (CFO Augmentation). For training a robust CFO estimator, different variations of the input perturbed with different CFO values should be provided to the CNN. To do this, we insert an intermediate CFO augmentation block in the training pipeline, as shown in Fig. 5.

Before adding *augmenting* CFO values to the normalized signal, $X^{\text{CFO}}_{\text{RMS}}(t)$, the inherent coarse CFO, f, of the signal

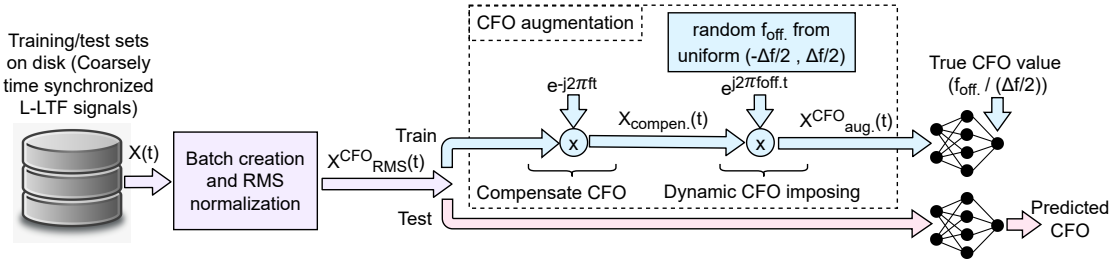

Fig. 5: PRONTO training/test pipeline for CFO estimation as a regression problem, with CFO augmentation block in the training path.

must be compensated. This f is previously recorded for each L-LTF during L-LTF extraction, and is required only during the training process, as shown in Fig. 5. The inherent coarse CFO, f, is compensated on $X^{\text{CFO}}_{\text{RMS}}(t)$ with sampling frequency f_s, as given in (10). Here, l is the L-LTF start index in the original waveform as defined in Section III-A.

$$X_{\text{compen.}}(t) = X^{\text{CFO}}_{\text{RMS}}(t) \times e^{-j2\pi f t/f_s}, t = l, \ldots, l+\mathcal{L}-1 \quad (10)$$

After compensating the CFO, f, an augmenting CFO needs to be applied to $X_{\text{compen.}}(t)$. We choose our augmenting CFO to be in range $[-\frac{\Delta f}{2}, \frac{\Delta f}{2}]$, where Δf is sub-carrier frequency spacing. In the augmentation step, we draw a random CFO value, $f_{\text{off.}}$, from a uniform distribution given as $f_{\text{off.}} \sim U(-\frac{\Delta f}{2}, \frac{\Delta f}{2})$, and apply it to the signal $X_{\text{compen.}}(t)$, as shown in (11).

$$X^{\text{CFO}}_{\text{aug.}}(t) = X_{\text{compen.}}(t) \times e^{j2\pi f_{\text{off.}} t/f_s}, \quad t = 0, \ldots, \mathcal{L}-1 \quad (11)$$

The augmentation function $F^{\text{CFO}}_{\text{aug.}}(x)$ is defined as a cascade of (10) and (11), with the assumption that $l = \mathcal{L}$ (i.e., the L-STF and L-LTF have the same length) as in (12).

$$F^{\text{CFO}}_{\text{aug.}}(x) = x \times e^{j2\pi(f_{\text{off.}}(t-\mathcal{L})-ft)/f_s}, t = \mathcal{L}, \ldots, 2\mathcal{L}-1 \quad (12)$$

During augmentation, for each $X^{\text{CFO}}_{\text{aug.}}(t)$, $f_{\text{off.}}$ is recorded as the true CFO label. For a regression problem, the labels need to be normalized, as well. In our case, $f_{\text{off.}}$ is scaled to the range of [-1,1] as in (13) to generate true label Y^{CFO} for the CNN.

$$Y^{\text{CFO}} = \frac{f_{\text{off.}}}{\Delta f/2} \quad (13)$$

The function $F^{\text{CFO}}_{\text{aug.}}(.)$ returns the augmented L-LTF signal, $X^{\text{CFO}}_{\text{aug.}}(t)$, along with its corresponding normalized true label, Y^{CFO}, as in (14).

$$X^{\text{CFO}}_{\text{aug.}}(t) \ , \ Y^{\text{CFO}} = F^{\text{CFO}}_{\text{aug.}}(X^{\text{CFO}}_{\text{RMS}}(t)) \quad (14)$$

CNN Modeling. Recall that real and imaginary components of $X^{\text{CFO}}_{\text{aug.}}(t)$s are separated to form inputs with dimensions $(\mathcal{L}, 2)$ which are fed to the training function along with their corresponding true labels, Y^{CFO}s. We design PRONTO coarse CFO estimator to learn to map each augmented L-LTF signal, $X^{\text{CFO}}_{\text{aug.}}(t)$, to a true CFO value, Y^{CFO}, as in (15) where δ is *tanh*, $F^{\text{CFO}}_{\theta}(.)$ denotes the regressor CNN, and $\hat{Y}^{\text{CFO}}$ represents the predicted coarse CFO.

$$\hat{Y}^{\text{CFO}} = \delta\left(F^{\text{CFO}}_{\theta}(X^{\text{CFO}}_{\text{aug.}}(t))\right), \qquad F^{\text{CFO}}_{\theta} : \mathbb{R}^{\mathcal{L}\times 2} \mapsto \mathbb{R}^1 \quad (15)$$

We train the CFO estimator CNN with batches of $X^{\text{CFO}}_{\text{aug.}}(t)$ signals. The *mean squared error* loss is calculated between

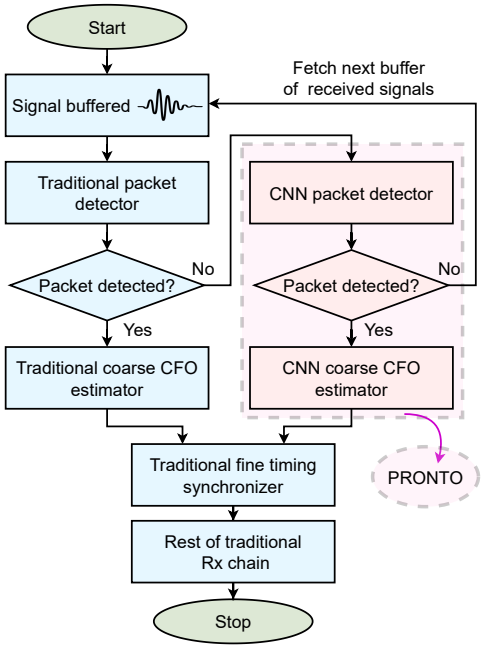

Fig. 6: Generalized decision flow for making PRONTO compatible with standard IEEE 802.11 waveforms. The system first looks for the L-STF in the received signal. If no L-STF is detected, it sends the signal to PRONTO packet detector CNN for L-LTF search.

Y^{CFO} and $\hat{Y}^{\text{CFO}}$ for each batch. We train the CNN for several hundreds of epochs with millions of CFO variations introduced in different L-LTF signals.

In the test phase, $X(t)$ signals only need to be normalized using (6).

After the test phase, the actual predicted coarse CFO, $\hat{f}_{\text{off}}$, is calculated as in (16), where $\hat{Y}^{\text{CFO}}$ is the CNN prediction in range [-1,1].

$$\hat{f}_{\text{off}} = \hat{Y}^{\text{CFO}} \times \Delta f/2 \quad (16)$$

D. System Overview

We propose an adaptive decision flow to make PRONTO compatible with standard IEEE 802.11 waveforms (i.e., containing both L-STF and L-LTF in the preamble), as well as the proposed modified version of this waveform (containing no L-STF in the preamble), as shown in Fig. 6. First the received signal is buffered and sent to the traditional packet detector block. If the latter detects the packet, it means that the L-STF pattern exists in the received signal, and the signal

goes through the traditional processing chain. If the traditional packet detector cannot detect a packet, there is a chance that the buffered signal is a modified waveform that starts with L-LTF. Therefore, we send the buffered signal to PRONTO packet detector to search for L-LTF pattern. If the L-LTF pattern is detected, we proceed with the PRONTO coarse CFO estimator. After coarse CFO estimation, the packet goes through the rest of the traditional receiver chain. If PRONTO packet detector CNN cannot detect the L-LTF pattern, it goes up to fetch the next buffered set of I/Q samples of the received signal. The whole process iterates again starting with the traditional packet detector.

IV. Datasets and Deep Learning Setup

A. Dataset Description

We use two OTA WiFi datasets to evaluate PRONTO.

1) Oracle Dataset: We use a publicly available[1] OTA WiFi dataset [36] that is referred to as *Oracle* in the rest of this paper. As described in the metadata files of this dataset, signals are collected in an indoor environment from 16 different USRP X310 software defined radios (SDRs) that transmit IEEE 802.11 Non-HT frames (see Fig. 2) with OFDM modulation, QPSK 3/4 in 2.4 GHz frequency with 5 MHz bandwidth. The distance between the transmitter and receiver varies between 2 ft and 62 ft with steps of 6 ft (11 different distances). The wireless channel is continuously sampled using a receiver USRP B210. We set the parameters defined in Section III, using the dataset properties, as shown in Table I.

2) Arena Dataset: To show how pre-trained PRONTO CNNs perform in new environments, we use the dataset used in [37], [38], which we refer to as *Arena* in the rest of this paper. As the name suggests, this dataset is collected in Arena [39], an indoor lab environment with X310 SDR arrays on the ceiling. A pair of X310 SDRs are used to collect ∼152k IEEE 802.11a packets with 16QAM modulation at 10 MHz bandwidth. The L-LTF parameters of this dataset are the same as Oracle dataset described in Table I, except for the sampling frequency (f_s=10 MHz) and the sub-carrier frequency spacing (Δf=156.5 kHz). As the SDR locations are fixed, transmission power is intentionally varied during data collection to emulate different SNRs.

B. L-LTF Extraction

For extracting L-LTF signals from each dataset, we find packet offsets using the traditional packet detection method explained in Section II-A and implemented in `MATLAB` function `wlanPacketDetect` with β=0.8. The packet offset in the standard IEEE 802.11 waveform is the offset from the beginning of the L-STF. We add an offset of 160 (L-STF length) to it and extract the next 160-sample sequence as coarsely time-synchronized L-LTF. To calculate the noise power, $P_{N(t)}$, in each L-LTF signal that is needed for the packet detection problem as shown in (5), we use `MATLAB` function `helperNoiseEstimate`. To obtain inherent coarse CFO, f, used in (10), we record coarse CFOs estimated using the L-STF with the traditional coarse CFO estimation method described in Section II-B. This step is implemented in `MATLAB` function `wlanCoarseCFOEstimate`. We extract ∼663k and ∼152k L-LTF signals from the recorded sequences in Oracle and Arena datasets, respectively, along with their corresponding noise power, $P_{N(t)}$, and their coarse CFOs, fs. These signals form our training, validation, and test sets that are used in packet detection and coarse CFO estimation problems.

Parameter	Notation	Value
Length of FFT (Length of one L-LTF training symbol)	$\mathcal{N}$	64
Full length of L-LTF	$\mathcal{L}$	160
Start index of L-LTF in the original waveform	l	160
Output layer size for classification	$\text{len}(\mathbb{I}^{\text{PD}})$	98
Output layer size for regression	$\text{len}(\mathbb{I}^{\text{CFO}})$	1
Sampling frequency	f_s	5 MHz
Sub-carrier frequency spacing	Δf	78.125 kHz

TABLE I: Parameters for 5 MHz WiFi signals [31] in the Oracle dataset [36].

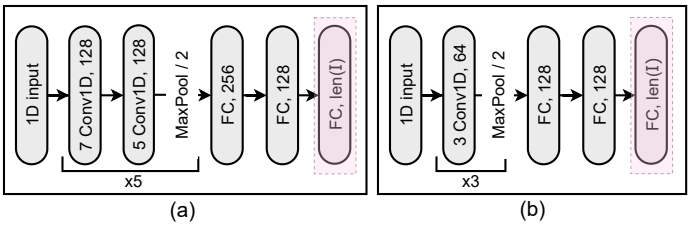

Fig. 7: (a) PRONTO-L with ∼1M parameters, and (b) PRONTO-S with ∼200k parameters, used for packet detection and coarse CFO estimation. The highlighted last layer has size 98 with *Softmax* activation for packet detection (classification), and size 1 with *tanh* activation for coarse CFO estimation (regression).

C. Deep Learning Setup

Data Augmentation. For the deep learning framework[2], we use Keras [40]. We use a data generator class inherited from `keras.utils.Sequence`, a special class from Keras libraries that gives us the possibility of loading the data to the memory batch-by-batch. This helps in efficiently performing the described data augmentation steps during packet detection and coarse CFO estimation.

CNN Input Dimensions. For both problems, we configure the input size of the CNN as $\mathcal{L}$ = 160, which is the size of an L-LTF signal. We separate real and imaginary parts in the L-LTF signal to form inputs of size (160, 2) with I/Q separated in the last dimension as separate channels.

CNN Architectures. We compare the performance of two 1D forward CNN architectures, (i) a large CNN called PRONTO-L with 13 layers and ∼1M parameters, and (ii) a smaller CNN called PRONTO-S with 6 layers and ∼200k parameters. Both architectures are combinations of 1D convolutional layers,

[1]https://genesys-lab.org/oracle

[2]https://github.com/nasimsoltani/PRONTO

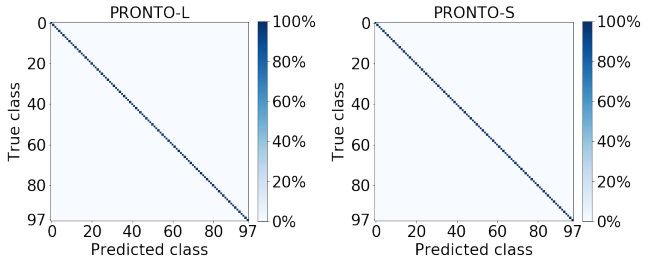

Fig. 8: Confusion matrices for packet detection using L-LTF, with PRONTO-L and PRONTO-S architectures. Both architectures yield 100% accuracy, therefore, for packet detection we choose PRONTO-S that has 1/5$^{\text{th}}$ parameters compared to PRONTO-L.

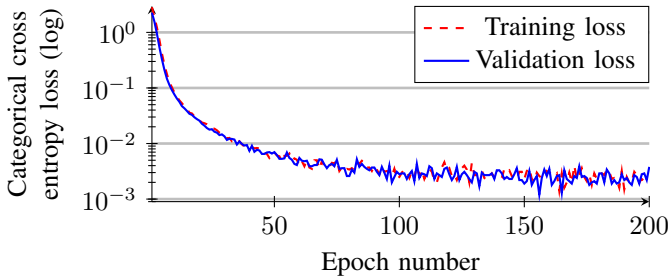

Fig. 9: Training and validation loss in packet detection using PRONTO-S.

1D MaxPooling layers, and fully connected (FC) layers. The details of PRONTO-L and PRONTO-S architectures are shown in Fig. 7. In this figure, the convolution layers are characterized with two numbers. The first number is the convolution kernel (filter) size, and the second number indicates the number of kernels in the layer. For example, *5 Conv1D, 128* means a convolution layer with 128 kernels of size 7 each. The FC layers are characterized with one value that is the output size of the FC layer. The MaxPooling layers have window size of 2 with stride 2.

CNN Output Dimensions. The final layer of CNNs in Fig. 7 is an FC layer whose size and activation function changes depending on the studied problem. For packet detection that is considered a classification problem with 98 classes, the final layer is of size 98 with *Softmax* activation. For CFO estimation that is considered a regression problem, the final layer has only one neuron with *tanh* activation function.

The PRONTO-L and PRONTO-S architectures shown in Fig. 7, with the aforementioned input and output dimensions have ∼1 million and ∼200 thousand parameters, respectively.

V. Evaluations

In this section, we validate PRONTO in terms of both packet detection and coarse CFO estimation performance using the OTA datasets described in Section IV. We use Oracle dataset to evaluate PRONTO training/test performance as the primary step. We further show how PRONTO trained in this initial environment (Oracle dataset) performs in a new environment (Arena dataset). Moreover, we present computational complexity and run-time of PRONTO on 4 different platforms and compare our results with the closest related work to PRONTO [27]–[29].

A. Packet Detection Evaluation

We shuffle Oracle L-LTF dataset and partition it into 70%, 10%, and 20% portions to form the training, validation, and test sets, respectively. We train the CNNs PRONTO-L and PRONTO-S shown in Fig. 7, on the training set. We stop training when validation accuracy does not improve during 100 consecutive epochs. We test the trained model on the test set, and for each signal in the latter, we calculate the predicted class using (8). The accuracy over the test set is calculated by dividing the number of correctly predicted signals by the total number of signals in the test set. We observe that both PRONTO-L and PRONTO-S, each with 98 classes defined in Fig. 4, can detect packets with 100% accuracy using only L-LTF signals. Fig. 8 shows the confusion matrices of PRONTO-L and PRONTO-S for these 98 classes with all outcomes represented on the diagonal.

As both CNNs show 100% accuracy, for reducing computational complexity we choose PRONTO-S as PRONTO packet detector CNN.

Fig. 9 demonstrates training and validation loss over epochs for PRONTO-S architecture. As expected, validation loss closely follows the training loss, which shows the network is not overfitting. The variations in the plot are due to random augmentation of both training and validation batches, which exposes the CNN to different data in every epoch. PRONTO-L loss plot follows the same trend.

B. CFO Estimation Evaluation

For the CFO estimation problem, we sort Oracle dataset in ascending order with respect to their CFO labels, fs. We pick the first ∼50% portion for training and validation and the second ∼50% for test, so that the training and validation sets contain negative CFOs, whereas the test set contains positive CFOs. In this way, the training and test sets emulate the realistic situation where training set CFO values are in a limited range, but we wish to predict CFOs of other ranges during test, as well. For the sake of extensive evaluation, we artificially inject CFOs in the range of $[0, \Delta f/2]$ to the test set signals.

CFO Augmentation Impact. To show the effect of CFO augmentation step, we first train PRONTO-L with the determined training set without CFO augmentation in the training pipeline. We compare predicted CFO values $\hat{Y}^{\text{CFO}}$ at the output of the CNN with true CFO labels Y^{CFO} and calculate mean absolute error (MAE) for each SNR level (distance) as in (17), where M is the number of test L-LTF signals in each SNR.

$$\text{MAE} = \frac{1}{M}\sum_{m=0}^{M-1} |Y_m^{\text{CFO}} - \hat{Y}_m^{\text{CFO}}| \tag{17}$$

The MAE for the aforementioned case for different SNR levels is shown in Fig. 10 under the legend "PRONTO-L, w/o CFO aug.". In a follow-up experiment, we train and test PRONTO-L with the same training and test sets, this time with CFO augmentation in the training pipeline, same as in Fig. 5.

Distance	Traditional receiver (all MATLAB)			PRONTO-based receiver		
(ft)	Coarse CFO (Hz)	Fine CFO (Hz)	**Coarse+Fine (Hz)**	Coarse CFO (PRONTO) (Hz)	Fine CFO (MATLAB) (Hz)	**Coarse+Fine (Hz)**
2	31186.47413	1.672545869	**31188.14668**	31073.47413 ↓	114.6725459 ↑	**31188.14668**
8	27935.04853	-173.4179159	**27761.63061**	28056.04853 ↑	-294.4179159 ↓	**27761.63061**
14	24935.6091	-24.97061088	**24910.63849**	25076.6091 ↑	-165.9706109 ↓	**24910.63849**
20	22454.2147	118.8058465	**22573.02055**	22331.2147 ↓	241.8058465 ↑	**22573.02055**
26	26613.99321	-2.242109413	**26611.7511**	26773.99321 ↑	-162.2421094 ↓	**26611.7511**
32	25127.22223	-153.0018763	**24974.22035**	24901.22223 ↓	72.9981237 ↑	**24974.22035**
38	23959.62959	86.35591791	**24045.9855**	23641.62959 ↓	404.3559179 ↑	**24045.9855**
44	24733.93078	-196.1821002	**24537.74868**	24444.93078 ↓	92.81789983 ↑	**24537.74868**
50	24282.74508	153.5915397	**24436.33662**	24541.74508 ↑	-105.4084603 ↓	**24436.33662**
56	27072.75937	-37.8109977	**27034.94837**	27739.75937 ↑	-704.8109977 ↓	**27034.94837**
62	32227.95237	-5318.056399	**26909.89597**	31128.95237 ↓	-4219.056399 ↑	**26909.89597**

TABLE II: Comparison of CFO estimation by traditional and PRONTO-based receivers, evaluated on the OTA WiFi dataset [36]. We observe that despite the small error in 'Coarse CFO (PRONTO)', 'Coarse+Fine' (bold column) in both cases of MATLAB and PRONTO lead to the exact same values (given until 5 decimal places). The reason is that MATLAB fine CFO estimator ('Fine CFO (MATLAB)') that follows 'Coarse CFO (PRONTO)', estimates the residual CFO that remains after the execution of PRONTO, as a part of fine CFO. Hence, no BER degradation occurs in the PRONTO-based receiver in comparison to the traditional receiver.

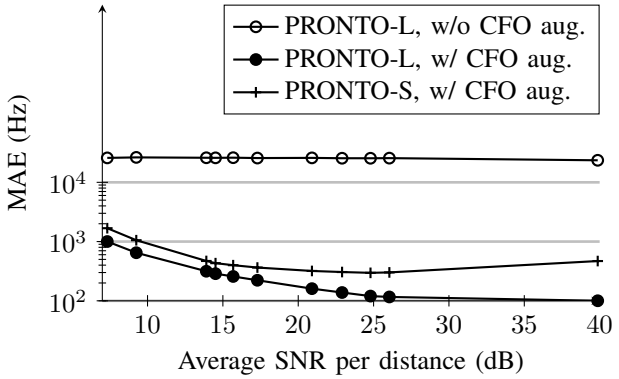

Fig. 10: Mean absolute error (MAE) between coarse CFOs predicted by CNNs using L-LTF signals and by MATLAB using L-STF signals, with and without CFO augmentation.

We calculate the MAE for this case for different SNR levels and show it under the legend "PRONTO-L w/ CFO aug." in Fig. 10. We see that the augmentation block improves the MAE results by up to ∼20000 Hz. This large improvement is achieved by exposing the neural network to different random CFOs that are not present in the training set and are artificially injected using the CFO augmentation block during training.

CNN Architecture Impact. In addition to CFO augmentation, the choice of CNN architecture also plays a key role. We train and test PRONTO-S in Fig. 7 with CFO augmentation block in the training pipeline. The results for different SNRs are shown as "PRONTO-S w/ CFO aug." in Fig. 10. We observe that the best CFO estimation (lowest MAE) is achieved with a comparatively deeper CNN (PRONTO-L), when CFO augmentation is included in the training pipeline.

Scatter Plots For Coarse CFO Estimation. In Fig. 11, scatter plots of "True CFO" versus "Predicted CFO" for each distance are demonstrated. The prediction results are achieved using PRONTO-L trained with CFO augmentation block in the training pipeline. We observe that for higher SNRs, we see the presence of a strong diagonal. As the SNR decreases, there is increased deviation from the diagonal line, which shows an increase in error. This is consistent with our observations in Fig. 10.

Coarse CFO Estimation MATLAB vs. PRONTO. To demonstrate PRONTO's ability to estimate coarse CFO compared with traditional methods, we feed the coarsely-time synchronized L-LTF signals in the test set to MATLAB function `wlanFineCFOEstimate`. This function is designed to estimate CFO using L-LTF signals through the traditional methods, and is used in fine frequency synchronization in the traditional receiver.

We collect predictions on test set signals from PRONTO-L and then calculate the percentage of error (PoE) in each SNR (distance) for both MATLAB calculations and PRONTO-L predictions, using (18).

$$\text{PoE} = \frac{1}{M}\sum_{m=0}^{M-1}\frac{|Y_m^{\text{CFO}} - \hat{Y}_m^{\text{CFO}}|}{|Y_m^{\text{CFO}}|} \tag{18}$$

As demonstrated in Fig. 12 the MATLAB function yields very large errors of upto ∼2000% when fed with coarsely time-synchronized L-LTFs, as it fails to estimate the coarse CFO on L-LTF signals. In contrast, PRONTO-L can estimate coarse CFO on L-LTF signals with errors as small as 3%.

Coarse CFO Estimation Error Impact on BER. In Fig. 10, we demonstrate that our CFO estimator CNN (without L-STF) performs within an error of ∼300 Hz averaged over all SNRs (distances) in reference to the traditional coarse CFO estimator (that uses L-STF). The question is: *How much BER degradation is caused by this small error in PRONTO's coarse CFO estimator?*

To answer this question, we measure the BER over the dataset when the receiver works with the legacy coarse CFO estimator, as well as with the coarse CFO estimator CNN, PRONTO-L. We observe that the two BERs are exactly the same. Therefore, the answer to the above question is *PRONTO causes no BER degradation*.

We explain below as to why there is no BER degradation, despite some error in coarse CFO estimation. The fine CFO

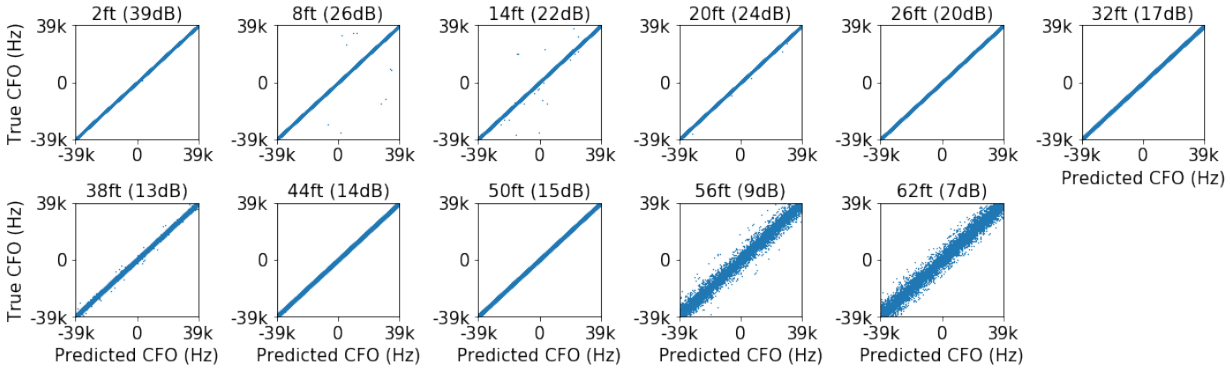

Fig. 11: Scatter plots showing the true CFO versus the predicted CFO for PRONTO CFO estimator in 11 different distances (SNRs). The predicted CFO shows a larger divergence from the true CFO in lower SNR regions.

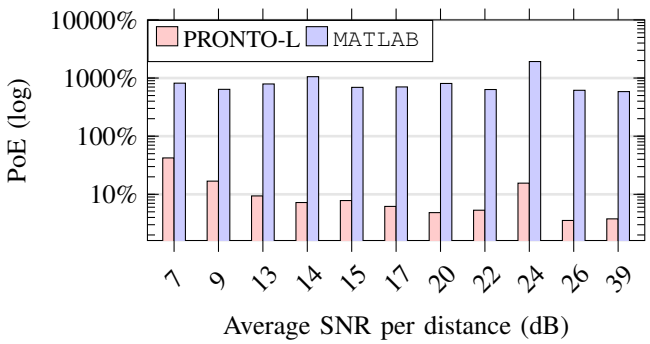

Fig. 12: Percentage of error (PoE) when MATLAB estimates CFO on coarsely time-synchronized L-LTF signals compared to PRONTO-L estimations.

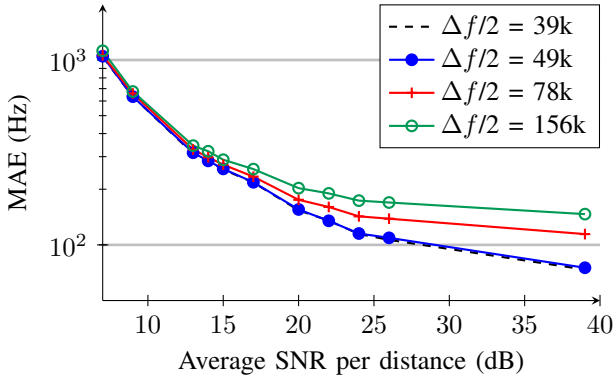

Fig. 13: Mean absolute error (MAE) between coarse CFOs predicted by CNNs and the imposed CFO when different values are chosen for Δf of CFO augmentation block.

estimation function, which is implemented through the legacy approach is shown in Fig. 1(b). This function tunes the fine CFO to compensate for the small errors in coarse CFO. Table II shows 11 test set packets with randomly injected CFOs for each distance from 2 ft to 62 ft. We observe that the traditional receiver (columns 2,3,4) estimates coarse and fine CFOs for each packet that leads to a total CFO (Coarse+Fine). In the PRONTO-based receiver (columns 5,6,7), the up/down arrows under the Coarse CFO (PRONTO) (column 5) show that the coarse CFO estimated by PRONTO increases/decreases in reference to coarse CFO of the traditional receiver (column 2). However, the fine CFO estimator that comes after coarse CFO estimation by PRONTO (column 6) compensates in the opposite direction. In this way, the total CFO (Coarse+Fine) for the traditional receiver (column 4) and PRONTO-based receiver (column 7) are calculated to be exactly equal to each other.

Coarse CFO Values Beyond Sub-carrier Spacing. The MATLAB function for coarse CFO estimation (i.e., `wlanCoarseCFOEstimate` [34]) can estimate coarse CFOs as large as twice the sub-carrier spacing. To show that PRONTO-L CFO estimator can perform similarly, we draw $f_{\text{off.}}$ from $U(-\frac{\Delta f}{2}, \frac{\Delta f}{2})$ as explained in Section III-C, and vary the ranges as [-39k, 39k] (same Δf as shown in Table I), [-49k, 49k], [-78k, 78k], and [-156k, 156k]. For each range, we train and test PRONTO-L CFO estimator with Δfs from the specific range and show MAE versus SNR in Fig. 13. We observe that MAE in higher SNRs slightly increases from 73 Hz (39k graph) to 146 Hz (156k graph), however, it remains smaller than the largest MAE for $\Delta f/2$=39k. From Table II, this CFO MAE is compensated by fine CFO estimation, and does not impact the BER. Therefore, the PRONTO-L is able to perform coarse CFO estimation of CFOs as large as twice the sub-carrier spacing similar to the MATLAB function.

C. PRONTO Performance in a New Environment

To evaluate packet detection in a new environment, we test the PRONTO-S architecture, pretrained on Oracle dataset in Section V-A, and we test it on packets from Arena dataset. The reason this test is possible is that although sampling frequency is different in Oracle (training) and Arena (test) datasets, the L-LTF length is equal to 160 I/Q samples for sampling frequencies between 5 to 20 MHz. Therefore, the number of classes for packet detection in Arena dataset is the same as Oracle dataset as shown in Fig. 4. We observe the same 100% classification accuracy for the Arena dataset without re-training the neural network.

To evaluate CFO estimation in a new environment, we train two PRONTO-L CNNs on the training portion of the Oracle dataset, one with Δf=78 kHz ($f_{\text{off.}} \sim U$(-39k, 39k)), and another with Δf=156 kHz ($f_{\text{off.}} \sim U$(-78k, 78k)). In this way, we set Δf for each CNN equal to the sub-carrier

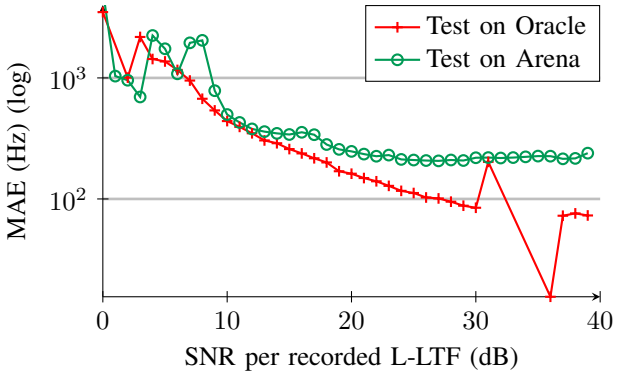

Fig. 14: Mean Absolute Error (MAE) for CFO estimation in test sets of Oracle and Arena (two different environments). For both tests, the same model trained on data from Oracle environment is used.

spacing in each of the 5 MHz and 10 MHz Oracle and Arena datasets, respectively. However, in both cases, we set f_s as the sampling rate of the training dataset, i.e., 5 MHz, and we train both CNNs on Oracle dataset. Since there is no distance concept in the Arena dataset, for a point to point comparison, we calculate the per-L-LTF frequency domain SNR using `MATLAB` function `helperNoiseEstimate`, as explained in Section IV-B, without averaging over distances. We test the trained models on their respective test datasets from Oracle (same environment as the training dataset) and Arena (different environment from the training dataset), and measure the MAE at each SNR for each test set. The average of MAEs shown in Fig. 14 are 247 Hz and 270 Hz for Oracle and Arena datasets, respectively. We observe that with changing the environment, coarse CFO error increases by 9%, however, the MAE is still in the range of Table II, and hence the residual CFO is compensated by fine CFO estimation process. Therefore, no compromise in BER happens if PRONTO is trained in an indoor environment, and is deployed in a new indoor environment.

D. Computation Cost

We evaluate PRONTO in terms of computation cost by counting floating point operations (FLOPs), similar to the state-of-the-art (SOTA) [16], [27]–[29], and run-time on 4 different commonly used platforms of desktop and edge CPUs and GPUs, as shown in Table III and listed below:

1) Desktop-CPU: AMD Ryzen Threadripper 2950X 16-Core Processor
2) Desktop-GPU: NVIDIA GeForce RTX 2080 GPU
3) Jetson TX2-CPU: Quad-Core ARM Cortex-A57
4) Jetson TX2-GPU: NVIDIA Pascal GPU

As explained in Sections V-A and V-B, in the PRONTO-based wireless receiver we use PRONTO-S architecture for packet detection and PRONTO-L for coarse CFO estimation (shown in Fig. 7).

FLOPs and CPU Run-Time. We count FLOPs in both `MATLAB` functions [41] and PRONTO CNNs for the two tasks of packet detection and coarse CFO estimation, and show them in the third row of Table III. As expected, PRONTO-L that is a larger CNN consists of more FLOPs compared to PRONTO-S (40.8M FLOPs for PRONTO-L versus 1.7M FLOPs for PRONTO-S). To have a fair comparison between traditional (`MATLAB`) and PRONTO algorithms run-time, we first run all the algorithms with *one* input (batch size=1) on the same platform of Desktop-CPU. To account for operating system interrupts and run-time fluctuations, we run `MATLAB` implementations for each algorithm of packet detection and coarse CFO estimation 100 times in a loop and measure 100 individual run-time durations. We calculate mean and standard deviation of these run-times and show them in the fourth row of Table III as *mean*$\pm$*standard deviation*. We repeat the same process of measuring run-time on CPU for PRONTO-S and PRONTO-L CNNs with batch size 1. We observe that PRONTO CNNs consist of much larger number of FLOPs compared to their traditional `MATLAB` counterparts. However, `MATLAB` packet detector function takes longer to run compared to PRONTO-S. The reason for this is that in the traditional `MATLAB` packet detection algorithm there are many comparison statements and branches that contribute to increasing the time, but are not counted as FLOPs.

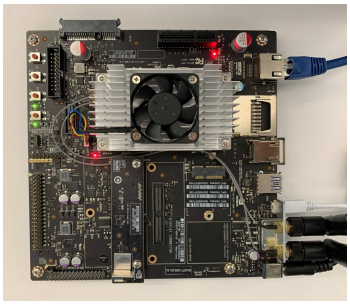

Fig. 15: NVIDIA Jetson TX2 used for edge implementation.

CPU/GPU Comparison. We run PRONTO-S and PRONTO-L on Desktop-GPU and measure run-time for one input L-LTF (batch size=1) during the same process as CPU run-time measurements. We observe that in the case of PRONTO-L GPU implementation decreases CPU mean run-time by 40% (1.8 ms for GPU compared to 3.15 ms for CPU). However, in the case of PRONTO-S GPU implementation increases CPU mean run-time by 65% (1.9 ms for GPU versus 1.09 ms for CPU). The reason for this is that GPU hardware consists of a large number of processing cores that run with a slower clock compared to CPU clock. GPU implementation is only beneficial when the algorithms are realized with compute-intensive large matrix multiplications, which is the case for large neural networks. Smaller CNNs specially if run with only one input have limited opportunity for GPU parallelization compared to larger CNNs. In other words, in the case of smaller CNN (PRONTO-S) GPU resources are under-utilized, which leads to longer run-time compared to CPU implementation. For the same reason, PRONTO-S run-time on GPU is larger than that of PRONTO-L despite PRONTO-S being smaller than PRONTO-L.

Edge Implementation. In order to measure PRONTO run time on a power-efficient edge device, we run PRONTO-S and PRONTO-L on NVIDIA Jetson TX2, an embedded AI

Task		Packet Detection		Coarse CFO Estimation	
Method		MATLAB	PRONTO-S	MATLAB	PRONTO-L
Computational complexity (FLOPs)		2400	1.7M	417	40.8M
Run Time (ms) Batch size=1	Desktop-CPU	**30.8±0.62**	1.09±0.12	**0.05±0.02**	3.15±0.29
	Desktop-GPU	-	1.9±0.19	-	1.88±0.22
	Jetson TX2-CPU	-	4.91±0.54	-	9.88±0.38
	Jetson TX2-GPU	-	8.98±0.61	-	7.99±1.01
Run Time (ms) Batch size=128	Desktop-GPU	-	2.02±0.16 **(0.015 ms per input)**	-	3.34±0.27 **(0.026 ms per input)**
	Jetson TX2-GPU	-	17.63±0.89 **(0.137 ms per input)**	-	53.19±1.64 **(0.415 ms per input)**

TABLE III: Comparison of FLOPs and run-time in traditional (MATLAB) and PRONTO (CNN) algorithms for packet detection and coarse CFO estimation. Run-durations are measured 100 times on each of the 4 different platforms, and results are demonstrated as *mean±standard deviation* of these 100 time durations.

computing device shown in Fig. 15. We can access an ARM Cortex-A57 processor as well as NVIDIA Pascal GPU on the TX2. We run one input L-LTF (batch size=1) through PRONTO-S and PRONTO-L on the ARM core and the GPU of the TX2 and measure mean and standard deviation of the run-time same as before. In Table III we observe that TX2-GPU decreases PRONTO-L run-time but increases PRONTO-S run-time compared to the run-time on the TX2-CPU core (for the same reason explained earlier). As expected, the run-time of PRONTO on the TX2 is overall larger compared to the powerful desktop computer.

Large Batch Size Effect. We explore the effect of feeding in large batch size of input (batch size=128) on effective GPU parallelization and PRONTO run-time. We repeat the run-time measuring procedure in a loop that runs 100 times on Desktop-GPU and TX2-GPU, however, instead of just one input (batch size= 1) we feed in 128 inputs together (batch size= 128). These 128 inputs are parallelized on the GPU architecture instead of running sequentially. We observe that the average run-time for PRONTO-S and PRONTO-L equals 2.02 ms and 3.34 ms, respectively, on the Desktop-GPU. If the run-time periods are averaged over the number of inputs, we achieve 0.015 ms and 0.026 ms per input for PRONTO-S and PRONTO-L, respectively. Similarly, on TX2-GPU run-time per input decreases to 0.137 ms and 0.415 ms for PRONTO-S and PRONTO-L, respectively, compared to batch size 1 on the same platform.

Run-time Comparison with Commercial WiFi Card. For an example WiFi card with 5 MHz sampling rate, each L-LTF is received in 32 μs. Therefore, each of packet detection and coarse CFO estimation tasks must happen within 32 μs for the card to be able to receive streaming signals without packet loss. The reported GPU run-time for PRONTO (15 μs and 26 μs) shows promising results wherein with a custom chip design, a fully-pipelined WiFi card receiver can leverage PRONTO for packet synchronization without packet loss in higher sampling rates. As an example, the decoder neural network proposed in [37] with 4.45×10^{10} FLOPs that is bigger than both PRONTO-S and PRONTO-L (with 1.7×10^6 and 40.8×10^6 FLOPs, respectively) works in 100 MHz frequency on FPGAs. A custom designed chip can boost the working frequency of these NNs even further and provide PRONTO timing efficiency in high sampling rate WiFi protocols.

	Ninkovic *et al.* [28]	Ninkovic *et al.* [27]	Zhou *et al.* [29]	**PRONTO** (this paper)
Performed packet detection	✓	✓	-	✓
Performed CFO estimation	-	✓	✓	✓
Preamble field used	L-STF	L-STF	L-STF	**L-LTF**
Used over the air data	-	✓	-	✓
Studied BER degradation	-	-	-	✓
Bandwidth (MHz)	1	1	10	**5**
Packet detection FLOPs	200M	200M	-	**1.7M**
CFO estimation FLOPs	>8738	-	-	**40.8M**
Edge implementation	-	-	-	✓
Packet detection miss rate	0.7%	0.4%	-	**0%**
Preamble length reduction	0%	0%	0%	**upto 40%**

TABLE IV: Comparing PRONTO with the state-of-the-art.

E. Comparison with State-of-the-art (SOTA)

Finally, we demonstrate the superiority of PRONTO over the SOTA techniques by evaluating its performance on various metrics as presented in Table IV. As PRONTO is a first-of-its-kind that uses only L-LTF for both packet detection and coarse CFO estimation, we do not have a direct comparison point among the related work. However, we compare our results with the SOTA approaches which use deep learning for packet detection and coarse CFO estimation using the traditional inputs for these tasks (i.e., the L-STF signals). In the following, we list the SOTA-related work that we use for comparison which were also previously discussed in Section I.

1) Ninkovic *et al.* [28]: Proposes deep learning for estimating the packet (L-STF) start index as a regression problem on simulated data.
2) Ninkovic *et al.* [27]: Proposes deep learning for estimating the packet (L-STF) start index as well as CFO, both as regression problems, on OTA data.
3) Zhou *et al.* [29]: Proposes deep learning for detecting the class of CFO with L-STF of simulated data as a classification problem.

Our comparison outcomes are shown in Table IV and the highlights are described in the following.

Novelty and Superiority. Our novelty of *not* using L-STF for packet detection and coarse CFO estimation provides the possibility of removing the L-STF, which reduces WiFi preamble length by upto 40%.

Model Size and Complexity. We show that our packet detector CNN (PRONTO-S) is much lighter compared to the architectures used in the SOTA. We show 1.7M FLOPs for packet detection compared to 200M FLOPs presented in the SOTA [27].

Communication Quality. While we study communication quality (i.e., BER) in our proposed PRONTO-based receiver, none of the studied related work consider the effect of errors in their packet detection and coarse CFO estimation on the BER.

Edge Implementation. None of the studied related work present edge implementation and run-time results for deployment of packet detection and CFO estimation neural networks in wireless receivers. However, we present edge implementation results on different platforms for conventional blocks as well as PRONTO.

Packet Detection Miss Rate. From another perspective, [28] and [27] report 0.7% and 0.3% miss detection rate, respectively, for packet detection with input size 160 (same input size as PRONTO). However, our 100% packet detection accuracy shows that all the packets detected by the traditional `MATLAB`-based packet detection algorithm are detected by PRONTO too and no packets are missed (0% miss rate). We believe that our robust packet detection algorithm is achieved by two techniques. Firstly, we treat packet detection as a classification problem instead of the less accurate regression proposed by the SOTA [27], [28]. Secondly, we use a data augmentation block that emulates different shifts in the L-LTF, accompanied by periods of noise, that contribute to training a more robust packet detection CNN.

To the best of our knowledge, there is no related work on packet detection and CFO estimation using L-LTF signals to be used as a direct comparison point. However, we establish the efficacy of PRONTO by achieving same BER as the traditional method after eliminating upto 40% of the preamble.

VI. Conclusions

PRONTO reduces the packet preamble overhead by eliminating the need for the first field of the preamble (i.e., L-STF), while remaining compliant with legacy standard waveforms that contain the entire preamble. To decode packets without L-STF, PRONTO performs coarse time and frequency synchronizations using the next field of the preamble (i.e., the L-LTF). PRONTO consists of two CNN-based modules for packet detection (coarse time synchronization) and coarse CFO estimation, along with their corresponding augmentation steps for training robust CNNs. We evaluated PRONTO on two OTA WiFi datasets collected from a testbed of SDRs and showed that packet detector CNN yields 100% accuracy with a modest error as small as 3% in coarse CFO estimation. We demonstrate that this small error in coarse CFO estimation does not degrade the BER. Finally, by comparing to neural networks implemented on custom hardware in the existing literature, we envision that with properly pipelined customized hardware, PRONTO can keep up with WiFi transmissions with high sampling rates.

Acknowledgement

This work is supported by the US National Science Foundation (NSF) CNS-1923789 award.

References

[1] H. Shimodaira, G. K. Tran, K. Sakaguchi, and K. Araki, "Investigation on Millimeter-wave Spectrum for 5G," in *2015 IEEE conference on standards for communications and networking (CSCN)*, pp. 143–148, IEEE, 2015.

[2] E. G. Larsson, O. Edfors, F. Tufvesson, and T. L. Marzetta, "Massive MIMO for next generation wireless systems," *IEEE communications magazine*, vol. 52, no. 2, pp. 186–195, 2014.

[3] L. Lu, G. Y. Li, A. L. Swindlehurst, A. Ashikhmin, and R. Zhang, "An Overview of Massive MIMO: Benefits and Challenges," *IEEE journal of selected topics in signal processing*, vol. 8, no. 5, pp. 742–758, 2014.

[4] X. Liu, H. Zeng, N. Chand, and F. Effenberger, "Efficient Mobile Fronthaul via DSP-based Channel Aggregation," *Journal of Lightwave Technology*, vol. 34, no. 6, pp. 1556–1564, 2015.

[5] S. D'Oro, F. Restuccia, and T. Melodia, "Can you fix my neural network? real-time adaptive waveform synthesis for resilient wireless signal classification," *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021.

[6] S. Peng, H. Jiang, H. Wang, H. Alwageed, and Y.-D. Yao, "Modulation Classification using Convolutional Neural Network Based Deep Learning Model," in *2017 26th Wireless and Optical Communication Conference (WOCC)*, pp. 1–5, IEEE, 2017.

[7] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air Deep Learning Based Radio Signal Classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[8] E. Perenda, S. Rajendran, G. Bovet, S. Pollin, and M. Zheleva, "Learning the Unknown: Improving Modulation Classification Performance in Unseen Scenarios," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021.

[9] W. Xiong, L. Zhang, M. McNeil, P. Bogdanov, and M. Zheleva, "Exploiting Self-Similarity for Under-Determined MIMO Modulation Recognition," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 1201–1210, 2020.

[10] Y. Lin, H. Zhao, Y. Tu, S. Mao, and Z. Dou, "Threats of adversarial attacks in DNN-based modulation recognition," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp. 2469–2478, IEEE, 2020.

[11] N. Soltani, K. Sankhe, S. Ioannidis, D. Jaisinghani, and K. Chowdhury, "Spectrum Awareness at the Edge: Modulation Classification using Smartphones," in *2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp. 1–10, IEEE, 2019.

[12] G. Shen, J. Zhang, A. Marshall, and J. R. Cavallaro, "Towards Scalable and Channel-Robust Radio Frequency Fingerprint Identification for LoRa," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 774–787, 2022.

[13] G. Shen, J. Zhang, A. Marshall, L. Peng, and X. Wang, "Radio frequency fingerprint identification for LoRa using spectrogram and CNN," *IEEE INFOCOM-IEEE Conference on Computer Communications*, 2021.

[14] N. Soltani, K. Sankhe, J. Dy, S. Ioannidis, and K. Chowdhury, "More is Better: Data Augmentation for Channel-Resilient RF Fingerprinting," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 66–72, 2020.

[15] N. Soltani, G. Reus-Muns, B. Salehi, J. Dy, S. Ioannidis, and K. Chowdhury, "Rf fingerprinting unmanned aerial vehicles with non-standard transmitter waveforms," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 15518–15531, 2020.

[16] T. Jian, Y. Gong, Z. Zhan, R. Shi, N. Soltani, Z. Wang, J. G. Dy, K. R. Chowdhury, Y. Wang, and S. Ioannidis, "Radio Frequency Fingerprinting on the Edge," *IEEE Transactions on Mobile Computing*, 2021.

[17] T.-J. Liang, W. Rave, and G. Fettweis, "On Preamble Length of OFDM-WLAN," in *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pp. 2291–2295, IEEE, 2007.

[18] Y. Wang, J. Oostveen, A. Filippi, and S. Wesemann, "A novel Preamble Scheme for Packet-based OFDM WLAN," in *2007 IEEE Wireless Communications and Networking Conference*, pp. 1481–1485, IEEE, 2007.
[19] M. Belgiovine, K. Sankhe, C. Bocanegra, D. Roy, and K. Chowdhury, "Deep Learning at the Edge for Channel Estimation in Beyond-5G Massive MIMO," *IEEE Wireless Communications Magazine*, pp. 1–7, 2021.
[20] R. Jiang, X. Wang, S. Cao, J. Zhao, and X. Li, "Deep neural networks for channel estimation in underwater acoustic OFDM systems," *IEEE access*, vol. 7, pp. 23579–23594, 2019.
[21] X. Zheng and V. K. Lau, "Online Deep Neural Networks for MmWave Massive MIMO Channel Estimation With Arbitrary Array Geometry," *IEEE Transactions on Signal Processing*, vol. 69, pp. 2010–2025, 2021.
[22] S. Liu, Z. Gao, J. Zhang, M. Di Renzo, and M.-S. Alouini, "Deep denoising neural network assisted compressive channel estimation for mmWave intelligent reflecting surfaces," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9223–9228, 2020.
[23] M. Schaedler, S. Calabrò, F. Pittalà, C. Bluemm, M. Kuschnerov, and S. Pachnicke, "Neural Network-Based Soft-Demapping for Nonlinear Channels," in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, pp. 1–3, IEEE, 2020.
[24] M. Schaedler, F. Pittalà, G. Böcherer, C. Bluemm, M. Kuschnerov, and S. Pachnicke, "Recurrent neural network soft-demapping for nonlinear isi in 800gbit/s dwdm coherent optical transmissions," in *2020 European Conference on Optical Communications (ECOC)*, pp. 1–4, IEEE, 2020.
[25] H. Kim, Y. Jiang, R. B. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication Algorithms via Deep Learning," in *International Conference on Learning Representations*, 2018.
[26] Y. He, M. Jiang, X. Ling, and C. Zhao, "A neural network aided approach for LDPC coded DCO-OFDM with clipping distortion," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2019.
[27] V. Ninkovic, A. Valka, D. Dumic, and D. Vukobratovic, "Deep Learning Based Packet Detection and Carrier Frequency Offset Estimation in IEEE 802.11 ah," *IEEE Access*, 2021.
[28] V. Ninkovic, D. Vukobratovic, A. Valka, and D. Dumic, "Preamble-Based Packet Detection in Wi-Fi: A Deep Learning Approach," in *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pp. 1–5, IEEE, 2020.
[29] M. Zhou, X. Huang, Z. Feng, and Y. Liu, "Coarse frequency offset estimation in MIMO systems using neural networks: A solution with higher compatibility," *IEEE Access*, vol. 7, pp. 121565–121573, 2019.
[30] F. A. Aoudia and J. Hoydis, "End-to-end Learning for OFDM: From Neural Receivers to Pilotless Communication," *IEEE Transactions on Wireless Communications*, 2021.
[31] T. Cooklev, *Wireless communication standards: A study of IEEE 802.11, 802.15, 802.16*. IEEE Standards Association, 2004.
[32] The MathWorks, Inc., "wlanPacketDetect to Estimate timing offset of OFDM packet." https://www.mathworks.com/help/wlan/ref/wlanpacketdetect.html, 2021.
[33] J. Terry and J. Heiskala, *OFDM Wireless LANs: A Theoretical and Practical Guide*. Sams publishing, 2002.
[34] The MathWorks, Inc., "wlanCoarseCFOEstimate to Perform coarse CFO estimation." https://www.mathworks.com/help/wlan/ref/wlancoarsecfoestimate.html, 2021.
[35] "IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, 2016.
[36] K. Sankhe, M. Belgiovine, F. Zhou, S. Riyaz, S. Ioannidis, and K. Chowdhury, "Oracle: Optimized Radio Classification Through Convolutional Neural Networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 370–378, IEEE, 2019.
[37] N. Soltani, H. Cheng, M. Belgiovine, Y. Li, H. Li, B. Azari, S. D'Oro, T. Imbiriba, T. Melodia, P. Closas, *et al.*, "Neural Network-Based OFDM Receiver for Resource Constrained IoT Devices," *IEEE Internet of Things Magazine*, vol. 5, no. 3, pp. 158–164, 2022.
[38] B. Azari, H. Cheng, N. Soltani, H. Li, Y. Li, M. Belgiovine, T. Imbiriba, S. D'Oro, T. Melodia, Y. Wang, *et al.*, "Automated Deep Learning-based Wide-band Receiver," *Computer Networks*, vol. 218, p. 109367, 2022.
[39] L. Bertizzolo, L. Bonati, E. Demirors, A. Al-shawabka, S. D'Oro, F. Restuccia, and T. Melodia, "Arena: A 64-antenna SDR-based ceiling grid testing platform for sub-6 GHz 5G-and-Beyond radio spectrum research," *Computer Networks*, vol. 181, p. 107436, 2020.
[40] F. Chollet *et al.*, "Keras." https://github.com/fchollet/keras, 2015.
[41] H. Qian, "Counting the Floating Point Operations (FLOPS)." https://www.mathworks.com/matlabcentral/fileexchange/50608-counting-the-floating-point-operations-flops, 2021.

Nasim Soltani is a PhD candidate at the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA. She started her PhD under supervision of Professor Kaushik Chowdhury in 2018. Her research area is ML applications for wireless systems. Her interests are within the physical layer of WiFi and cellular systems, specifically RF fingerprinting, neural network-based receiver design, signal classification, and secure communication.

Debashri Roy is currently an Associate Research Scientist at the Department of Electrical and Computer Engineering, Northeastern University. She received her MS (2018) and PhD (2020) degrees in Computer Science from University of Central Florida, USA. She was an experiential AI postdoctoral fellow at Northeastern University (2020-2021). Her research interests are in the areas of AI/ML enabled technologies in wireless communication, multimodal data fusion, network orchestration and nextG networks.

Kaushik Chowdhury is a Professor at Northeastern University, Boston, MA. He is presently a co-director of the Platforms for Advanced Wireless Research (PAWR) project office. His current research interests involve systems aspects of networked robotics, machine learning for agile spectrum sensing/access, wireless energy transfer, and large-scale experimental deployment of emerging wireless technologies.