Neural Network-Based OFDM Receiver for Resource Constrained IoT Devices

Nasim Soltani, Hai Cheng, Mauro Belgiovine, Yanyu Li, Haoqing Li, Bahar Azari, Salvatore D'Oro, Tales Imbiriba, Tommaso Melodia, Pau Closas, Yanzhi Wang, Deniz Erdogmus, and Kaushik Chowdhury

Abstract

Orthogonal Frequency Division Multiplexing (OFDM)-based waveforms are used for communication links in many current and emerging Internet of Things (IoT) applications, including the latest WiFi standards. For such OFDM-based transceivers, many core physical layer functions related to channel estimation, demapping, and decoding are implemented for specific choices of channel types and modulation schemes, among others. To decouple hard-wired choices from the receiver chain and thereby enhance the flexibility of IoT deployment in many novel scenarios without changing the underlying hardware, we explore a novel, modular Machine Learning (ML)-based receiver chain design. Here, ML blocks replace the individual processing blocks of an OFDM receiver, and we specifically describe this swapping for the legacy channel estimation, symbol demapping, and decoding blocks with Neural Networks (NNs). A unique aspect of this modular design is providing flexible allocation of processing functions to the legacy or ML blocks, allowing them to interchangeably coexist. Furthermore, we study the implementation cost-benefits of the proposed NNs in resource-constrained IoT devices through pruning and quantization, as well as emulation of these compressed NNs within Field Programmable Gate Arrays (FPGAs). Our evaluations demonstrate that the proposed modular NN-based receiver improves bit error rate of the traditional non-ML receiver by averagely 61 percent and 10 percent for the simulated and over-the-air datasets, respectively. We further show complexity-performance tradeoffs by presenting computational complexity comparisons between the traditional algorithms and the proposed compressed NNs.

Introduction

The Internet of Things (IoT) paradigm will enable exciting applications, such as remote surgery, autonomous cars, augmented/virtual reality, all of which demand faster processing, higher data rates, and more reliable communications beyond what is realizable today. To meet such demands, most modern WiFi standards deployed in commercial consumer IoT devices today, including IEEE 802.11a/g/n/ac and other emerging standards use Orthogonal Frequency Division Multiplexing (OFDM).

The typical OFDM receiver consists of several signal processing blocks that detect and synchronize the packet, estimate the channel and equalize the payload to overcome channel-induced distortions, and finally extract useful bit representations through demapping and error correction (decoding). An overview of the traditional processing blocks is shown in Fig. 1 (bottom). Indeed, hand-engineered processing steps (e.g., using custom-designed packet preambles or fixed modulation schemes) offer limited opportunities for on-the-fly adaptation. Moreover, the wireless environment is too complex to be modeled accurately, and constraining the choice of the processing blocks to only one of several candidates may lower the performance. On the other hand, Neural Networks (NNs) provide an adaptable and noise-resilient solution for many physical layer processing tasks, such as modulation classification [1] and RF fingerprinting [2, 3], that improve the performance of their traditional counterparts. Similarly, in the domain of receiver design, NNs can offer a closed-form and flexible solution by learning to imitate previous channel estimations, symbol demappings, and decodings, instead of explicitly realizing the mathematical form

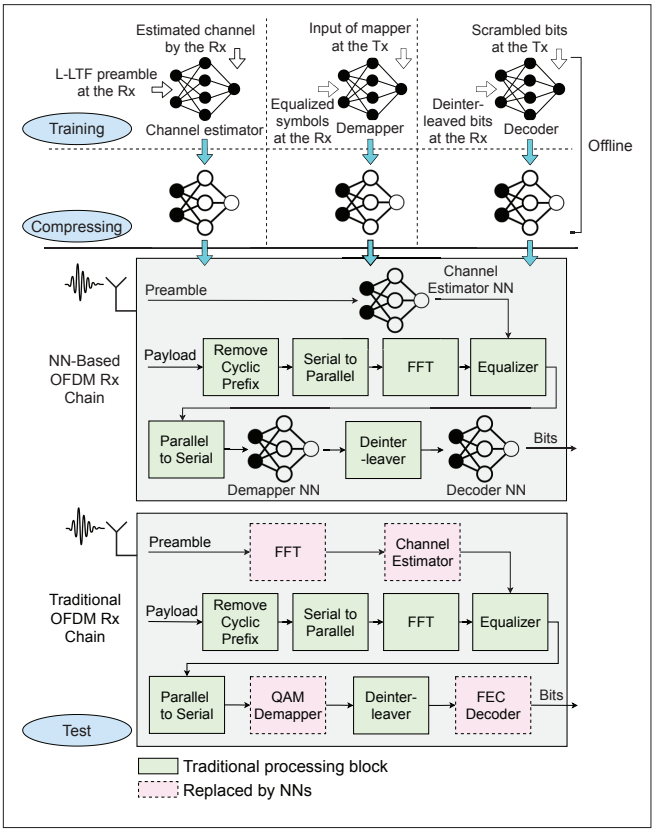

FIGURE 1. An overview of the proposed Neural Network (NN)-based receiver. The channel estimator, demapper, and error corrector (decoder) blocks are substituted with individual NNs. The NNs are trained and compressed offline, for online FPGA deployment.

The authors are with Northeastern University, USA.

Hai Cheng, Mauro Belgiovine, Yanyu Li, and Haoqing Li have equally contributed to this article.

Digital Object Identifier: 10.1109/IOTM.001.2200051

2576-3180/22/$25.00 © 2022 IEEE

of these processing blocks.

- **Real-time Computation.** The flexibility and performance improvement arising from including ML blocks within the receiver processing chain comes at a computational cost. Although NN computations are simple multiplications and additions, NNs are overall compute-intensive, and thus deploying them might cause delays that impact time-sensitive applications. In the wireless receiver, delays even as small as a few microseconds might prevent the processing chain to remain synchronized with the streaming data, which causes incorrect decoding. Thus, methods to achieve real-time NN computations are critical.
- **Proposed Modular Design.** While a single NN that captures the entire receiver processing chain simplifies the OFDM receiver, the routing of data, and NN inference execution, there are several drawbacks of such a monolithic design. First, this becomes a "black-box" approach that yields no insights on the performance of the intermediate functional steps. Thus, if say the *demapper* is under-performing in a given situation, there is no way for the designer to know this. Second, the receiver may be frequently deployed in new wireless environments, which necessitates re-training of the entire NN with massive volumes of data each time. For example, introducing a new modulation scheme or coding rate to the original waveform renders the entire prior training inadmissible. These considerations motivate us to pursue a model-driven design, with the goal of maintaining full compatibility with the classical processing chain. Thus, any individual classical processing block could be swapped with its ML block counterpart in a way that is transparent to the rest of the receiver chain. This fully modular approach distinguishes our work from other recent work that use NN-based solutions for cyclic prefix free [4], DFT free [5] or pilotless communications [6] in OFDM systems.

In this article, we propose an NN-based end-to-end OFDM receiver. Our scheme is composed of channel estimator, demapper, and error corrector (decoder) NNs with totally ∼3.1M parameters, cascaded to build a complete receiver, as shown in Fig. 1 (top). Using both simulated and Over-The-Air (OTA) datasets, we show that these NNs perform better than their non-ML counterparts. Moreover, we propose Block Column Row (BCR) pruning and Mixed Scheme Quantization (MSQ) to compress these models without accuracy loss for Field Programmable Gate Arrays (FPGA) deployment, and present FPGA results. We further calculate computational complexity in terms of floating-point operations (FLOPs) for the traditional and NN algorithms and provide comparisons of the two. Our prototype design can be implemented in small form-factor FPGAs that may be present in IoT devices, or it can be used to design a custom chip for NN-based OFDM receivers for specific IoT applications.

Related Work

We categorize the previous work on modeling the OFDM receiver chain into

- Data-driven approaches, where the NN models are developed using data, without domain knowledge being involved,
- Model-driven approaches, where domain knowledge is used in the design of NN-based receivers

In [7], a data-driven approach is proposed that directly predicts the transmitted symbols without explicit channel estimation, by a single deep NN. This NN is trained to minimize the difference between its output and the transmitted data. In [5], the authors eliminate the DFT in the OFDM receiver using a complex-valued deep NN. They train their channel equalizer based on a frozen pre-trained basic receiver.

In contrast, in model-driven design, domain knowledge is adopted to design separate NNs for each function in the receiver chain [8]. Reference [4] proposes an AI-aided OFDM chain for a cyclic prefix-free system, where channel estimation and signal detection are done by separate NNs. The authors integrate the orthogonal approximate message passing algorithm with the signal detection NN. The authors in [6] propose end-to-end learning with the purpose of eliminating orthogonal pilots in OFDM symbols by jointly optimizing parts of the transmitter and the receiver. percent, which results in no bit error rate deterioration.

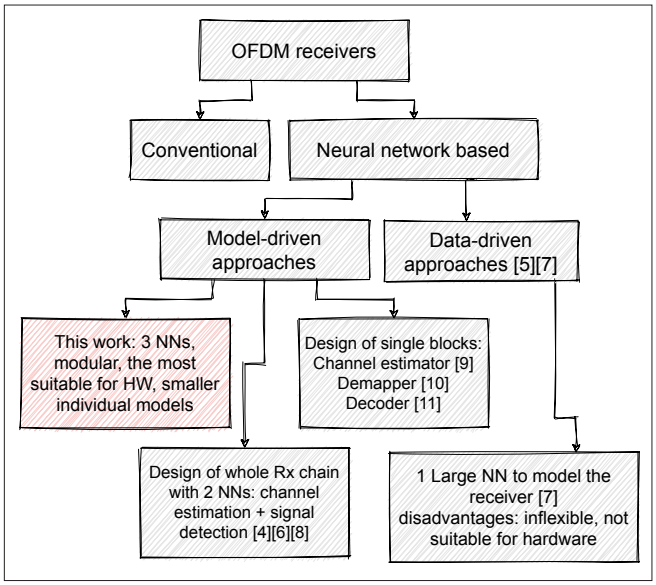

FIGURE 2. Related work summary for OFDM receivers.

There are additional work on replacing individual blocks in the traditional receiver with NNs. For example, [9] proposes a multi-layer perceptron NN for estimating the channel in massive MIMO systems for 5G and beyond. Authors in [10] propose a soft-demapper using a fully-connected NN. Aside from the OFDM receiver context, authors in [11] propose a recurrent neural network to decode different coding schemes, including the convolutional codes in the Viterbi algorithm [12].

A summary of the aforementioned categories is shown in Fig. 2. Despite the large body of literature that propose NNs for performing different tasks in OFDM receivers, to the best of our knowledge, prior work do not propose a fully-modular NN design for OFDM receivers, which is customized for implementation on resource-constrained hardware.

Model-Driven NN-Based OFDM Receiver

In this section, we describe channel estimation, demapping, and decoding tasks in the OFDM receiver, through both the classical and NN methods. These NNs are individually trained and then cascaded to build the complete receiver. Finally, we describe our pruning and quantization methods to compress these NNs for FPGA implementation.

Channel Estimation

Classical Approach: After packet detection, the OFDM wireless receiver estimates the Channel State Information (CSI) to compensate for channel-induced distortions in the received signal. This estimation is done in the frequency domain, for different OFDM formats in WiFi 802.11a/g/n/ac, by leveraging the Legacy Long Training Field (L-LTF) in the preamble.

The L-LTF consists of 2 identical OFDM symbols containing pilot information, known at the receiver. The Least Square (LS) channel estimation is used to estimate the channel coefficients of each sub-carrier. In the traditional receiver, after OFDM demodulation, LS method divides the received pilot sequence element-wise by the known pilot sequence. The final CSI is the average of the two different CSI vectors obtained by performing the LS estimation on the two OFDM symbols in the L-LTF.

Proposed NN Architecture: In contrast to the classical channel estimator whose input is frequency-domain L-LTF, our pro-

posed channel estimator NN processes the time-domain L-LTF. This obviates the need for the FFT operation during demodulation, as shown in Fig. 1, which is beneficial as FFT becomes expensive for large number of subcarriers. As the estimated channel is calculated independently for each sub-carrier, we use fully-connected layers to design the channel estimator NN. Our NN consists of two Multi-Layer Perceptron (MLP) architectures that estimate the real and imaginary parts of CSI independently. Each MLP has an input size of 160 with 2 hidden layers, of 512 and 256 neurons, respectively. The hidden layers have ReLU activation. Without loss of generality, we use WiFi 802.11a-compliant waveform that has 52 channel coefficients. Thus, the channel estimator output layer also is of size 52, with linear activation for the regression task. We add a dropout layer with drop probability 15 percent between the first and the second hidden layers to avoid overfitting. Each MLP has 227k parameters giving ∼450k parameters cumulatively for the complete architecture.

Training Process: We create the training dataset by generating 10k standard compliant transmissions and passing them through a simulated `wlanTGnChannel` in MATLAB. These are packets distorted by the wireless channel without Additive White Gaussian Noise (AWGN). We collect these packets at the receiver-side, and create a set of time-domain L-LTF OFDM symbols denoted as $X^{(a)}_{\text{train}}$s. We estimate the channel with LS method and save the set of 52 CSI coefficients (one for each subcarrier) as $Y^{(a)}_{\text{train}}$s. Similar to the noise model introduced within the data augmentation pipeline in [2], we dynamically add different levels of AWGN to $X^{(a)}_{\text{train}}$s, before feeding them to the NN. This added noise simulates SNRs within the range of 0 to 30 dB with steps of 5 dB. Since $Y^{(a)}_{\text{train}}$s are estimated from noiseless preambles, this dynamic addition of noise helps the NN learn to generalize and associate noisy $X^{(a)}_{\text{train}}$s with noiseless $Y^{(a)}_{\text{train}}$s, which improves channel estimator NN performance over its traditional counterpart.

We use Mean Squared Error (MSE) loss function, computed between the output of the MLP and its corresponding ground truth CSI computed through LS. In order to make our model able to process signals received at different power levels, we use Root Mean Square (RMS) normalization on the signal to bring it to a nominal signal power of 1W (i.e., 0 dBW).

Demapping

Classical Approach: After estimating the wireless channel, we equalize the demodulated payload, through dividing the frequency-domain payload by the frequency-domain estimated channel. The task of the demapper block is to map complex equalized symbols to a sequence of either soft or hard bits in soft-demapping and hard-demapping, respectively. The length of the output bit sequence is proportional to the modulation order. For example, in 16QAM, the transmitter-side *mapper* relates every four bits to one complex symbol. Consequently, in the receiver, the demapper demaps each equalized symbol to four bits.

Proposed NN Architecture: We follow the traditional demapper concept, and propose an NN that generates a sequence of bits for each equalized symbol. Without losing generality, we design an example demapper that demaps symbols from 16QAM. Consequently, our NN has an input size of 2 that represent In-phase (I) and Quadrature (Q) parts of one equalized symbol. Since the demapper NN is supposed to work on one equalized symbol at a time and the symbols are independent from each other, we use a fully-connected NN. Unlike convolutional networks with window size > 1, our approach operates on each equalized sample separately. The output layer has size 4, where each neuron represents one bit. Since each output bit must be set to either "0" or "1", and multiple bits can be "1" simultaneously, we consider demapping as a multi-label classification problem, and choose *Sigmoid* activation for the last layer. To determine the number of layers, we search the design space of fully-connected NNs with different number of layers and different number of neurons in each layer. We find out that the smallest NN for the 16QAM demapper is a fully-connected NN with 2 layers, with output sizes of 20 and 4, respectively. This model has only 144 total parameters.

Unlike the demapper NN where each equalized sample is mapped to a bit sequence independently, decoding from a convolutional code requires processing of a sequence of inputs.

Training Process: The best demapping performance is achieved when the training set contains only low SNR packets. We train the demapper NN with equalized symbols, as $X^{(b)}_{\text{train}}$s, from 16k packets with SNR 2 dB. For the labels ($Y^{(b)}_{\text{train}}$s), we use inputs of the mapper at the transmitter-side. In this way, the demapper learns to demap the equalized symbols to the original, non-distorted bits at the transmitter-side.

During inference, the NN receives the equalized symbols. Since we use Sigmoid activation in the last layer that produces outputs in the range of [0,1], we consider the probability of firing each neuron as the probability of corresponding bit being "1". The soft-bits (i.e., Log Likelihood Ratios (LLRs)) are calculated by taking the logarithm of probability of the output being "0" divided by the probability of the output being "1". Using this relationship, one LLR value is calculated for each output neuron. These LLR values are scaled by a combination of CSI and noise variance, calculated by the equalizer. Next, the scaled LLRs are de-interleaved and provided to the Forward Error Correction (FEC) decoder to compute the actual transmitted bits, as shown in Fig. 1.

Error Correction

Classical Approach: The role of the forward error correction (FEC) decoder block is to convert the de-interleaved bits to an error-corrected sequence of bits, as shown in Fig. 1. The relationship between the input size and the output size of the decoder depends on the coding rate C_{rate}, with the output size being C_{rate} times the input size. Similar to demapping, conventional decoding can be done in two ways of soft-decoding and hard-decoding, depending on the decoder inputs being whether soft-bits or hard-bits, respectively. In the traditional decoder, the Viterbi [12] algorithm is used to decode the correct sequence of bits.

Proposed NN Architecture: Unlike the demapper NN where each equalized sample is mapped to a bit sequence independently, decoding from a convolutional code requires processing of a sequence of inputs. For this reason, instead of a simple fully-connected network, the decoder uses Recurrent Neural Networks (RNNs), which are designed to process sequential data. To have the input of the decoder NN between 0 and 1, we convert the de-interleaved LLRs to the probability of a bit being equal to "1". This probability is achieved by inverting the LLR formula, therefore, probability of each soft-bit (LLR) being "1" equals $1/(1 + e^{LLR})$.

Due to the nature of RNNs, the input size of our proposed decoder NN can vary up to l, which corresponds to the size of de-interleaved LLR vector for the payload. The decoder RNN architecture consists of a recurrent part and a fully-connected part stacked together. The recurrent part has 3 Gated Recurrent Unit (GRU) layers, each with 256 units. The fully-connected part has 2 dense layers. The hidden layer has 16 neurons with ReLU activation, and the output layer has 1 neuron. Since, same as the demapper, the decoder produces bit sequences, the output layer has Sigmoid activation with size C_{rate} times the size of the LLR vector. The decoder NN has ∼2.7M total parameters.

Training Process: Similar to demapping, the best decoding performance is achieved when the training set contains only low SNR packets. Therefore, we use 16k packets in SNR 2 dB to train the decoder. As shown in Fig. 1, we train the decoder NN with outputs of de-interleaver at the receiver-side as $X^{(c)}_{train}$s and the scrambled bits at the transmitter-side as $Y^{(c)}_{train}$s. Using transmitter-side scrambled bits as $Y^{(c)}_{train}$s helps the NN learn to map noisy inputs to undistorted original bits, which boosts the decoder performance.

During inference, the decoder NN generates the probability of each bit being "1". To yield the final bit sequence, these probabilities are mapped to bit "1" if they are greater than 0.5, and to bit "0" otherwise.

So far, we described how NNs can substitute classical signal processing blocks in the OFDM receiver chain. Next, we describe how the NNs are compressed during training for FPGA deployment.

FPGA Implementation

The NNs described in this article span a range from small (144 parameters) to large (up to ~2.7M parameters). Direct deployment of such components on resource-constrained hardware (i.e., FPGA) is not possible. To compress these architectures suitably for FPGA deployment, we use two approaches:

1. Making the network sparser by reducing the number of weights (pruning)
2. Restricting the weights to be represented by a small number of bits (quantization)

The latter not only reduces the memory needed to store each weight, but also fits the limitations of a fixed-point hardware such as FPGA. Moreover, by implementing different quantization methods, we can control which FPGA resources will be used for weight multiplication. Both pruning and quantization methods happen along training and the resulting model has quantized weights where many are set to zero (are pruned).

1. *Pruning:* Two popular types of pruning are *structured pruning* and *irregular pruning*. Structured pruning is a coarse-grained pruning approach that removes the whole filter or channel in an NN layer, and is the best for hardware acceleration. However, structured pruning adversely affects the accuracy. In contrast, irregular pruning is a fine-grained pruning approach that sets the weights with small magnitudes to zero, preserves the accuracy, but does not attain acceleration on most hardware platforms. To solve this issue, we propose a Block-based Column Row (BCR) pruning scheme that serves as the universal fine-grained structured pruning. This pruning method can prune convolutional and fully-connected layers. As shown in Fig. 3, in BCR pruning, each weight matrix is divided into multiple blocks, and row and column pruning is applied to each block separately. We employ ADMM-based pruning [13] to determine the row/column pruning ratio automatically. The described fine-grained BCR pruning can significantly outperform the traditional coarse-grained structured pruning, and can provide larger acceleration compared to the more flexible irregular pruning.
2. *Quantization:* Weight quantization is another method for compressing the NN for FPGA implementation. Two popular methods of quantization are *fixed-point quantization* and *power-of-two quantization*. Fixed-point quantization is a naive quantization method that prepares the NN to run on a fixed-point hardware such as FPGA. Power-of-two quantization converts weight multiplications to simple bit shifts. In this way, weight multiplications, that are typically implemented on the specialized hardware blocks called *DSP48s* inside FPGAs, will be implemented on the Look-Up Tables (LUTs) as simple bit shifts. Based on these two schemes, we propose a Mixed Scheme Quantization (MSQ) approach that applies fixed-point quantization and power-of-two quantization on different rows of the weight matrix. There are two major motivations for using MSQ: First, different rows of the weight matrix have different distributions. Since power-of-two quantization has higher resolution around the center, it is best to be applied to the rows with a lower variance. In contrast, fixed-point quantization is suitable for rows with near uniform weight distribution, that have higher variance. Second, by using a specific type of quantization for each row, a mix of FPGA resources (DSP48s and LUTs) are used, which balances the resource utilization for NN weight multiplications. We observe that MSQ maintains the accuracy of the two single methods, due to being able to accommodate to different weight distributions.

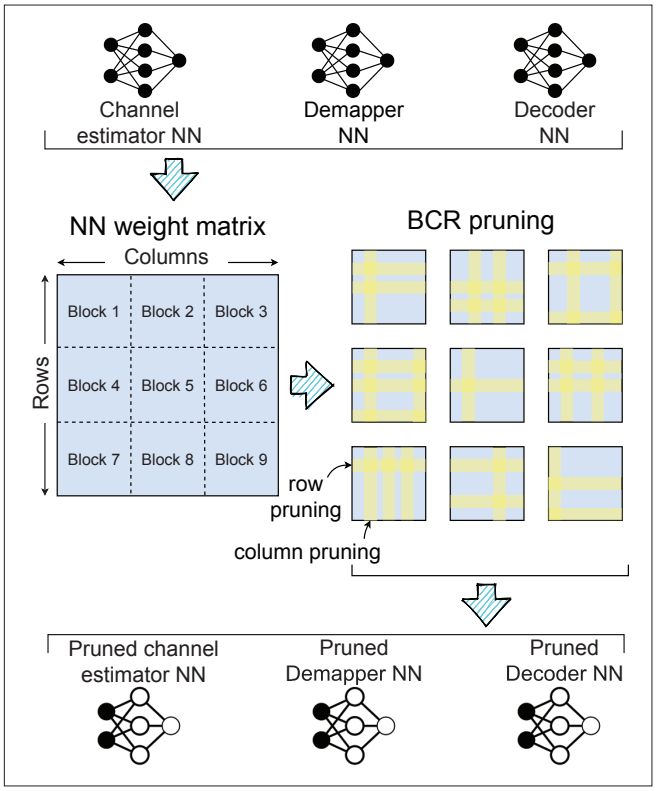

FIGURE 3. The proposed BCR pruning method is applied to all the fully-connected layers in the channel estimator, demapper, and decoder NNs.

Performance Evaluation

We use PyTorch on Nvidia RTX 2080Ti GPUs to train individual NNs. The metric we use for evaluating the proposed NN-based receiver is the Bit Error Rate (BER), that shows the rate of incorrectly recovered received bits after error correction. percent, after the signal passes through all the parts in the OFDM chain. We evaluate the individual performance of the NNs by inserting each NN in the OFDM receiver, while the rest of the processing is performed by the classical blocks. To evaluate the whole NN-based OFDM receiver, we cascade all the 3 NNs and measure the BER. For fair comparison, we further create a MATLAB baseline, which demonstrates the BER of a traditional MATLAB OFDM chain, without ML involved, as shown in Fig. 1 (bottom). In the MATLAB processing chain, for channel estimation, we use the standard compliant LS estimation for preamble-based frequency-domain channel estimation that is implemented in MATLAB function `wlanLLTFChannelEstimate`. For the demapper, we use soft-demapping through approximate LLR method implemented in MATLAB function `wlanConstellationDemap`. For the decoder, we use convolutional decoder (that decodes Binary Convolutional Coding (BCC)) implemented in MATLAB function `wlanBCCDecode`. For BER evaluation, we use simulated and OTA datasets that are briefly described below.

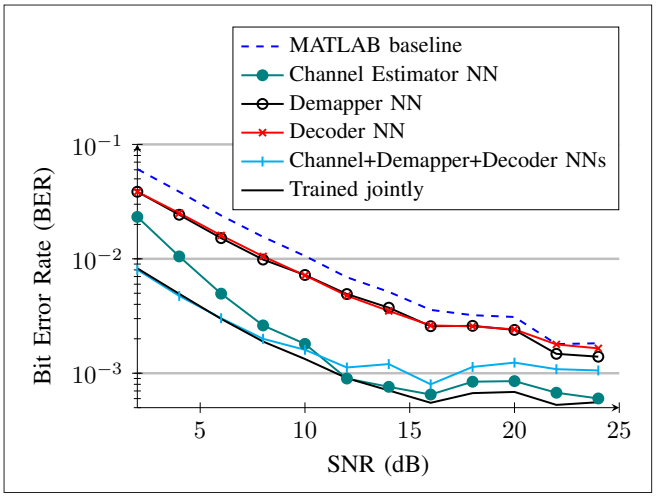

FIGURE 4. BER of simulated dataset achieved from NNs and MATLAB baseline. For the simulated dataset, individual Channel Estimator, Demapper, and Decoder NNs, trained separately, show 86 percent, 36 percent, and 36 percent average BER improvement over MATLAB baseline. The cascade of separately trained NNs provide 61 percent BER improvement over MATLAB baseline. If the NNs are trained jointly, BER improvement reaches 77 percent, compared to MATLAB baseline.

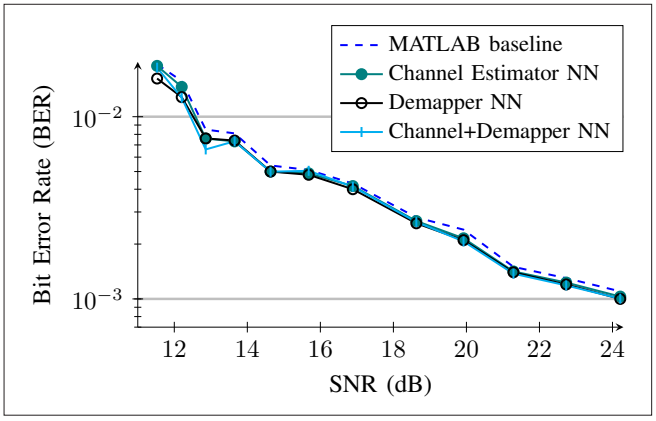

FIGURE 5. BER of the OTA dataset achieved from NNs and MATLAB baseline. For OTA dataset, Channel Estimator and Demapper NNs provide upto 12 percent and 20 percent BER improvement over MATLAB baseline. The cascade of these NNs yields an average 10 percent BER improvement over all SNRs.

Test Dataset Description

Simulated dataset: We use `WLAN` toolbox in MATLAB R2020a to create a simulated dataset by generating 192k packets, each containing a random sequence of bits. These packets are then modulated in accordance with IEEE 802.11a standard with Modulation Coding Scheme (MCS) 16QAM 1/2, before passing through a simulated `wlanTGnChannel` and an AWGN channel with desired SNR level. The SNR levels we use are between 2 dB and 24 dB with steps of 2 dB. With 192k total packets distributed among 12 SNR levels, we create a test set of 16k packets per SNR.

Over-The-Air (OTA) Dataset: We collect an OTA dataset in Arena [14], using one transmitter and one receiver. Similar to the simulated data, we generate random bit sequences, modulate them according to IEEE 802.11a standard, and transmit them via Software-Defined Radios (SDRs) placed in an overhead ceiling-mounted array. We repeat this process with different power levels to account for different SNRs, and collect ∼17k real-channel-distorted packets with SNRs between 11 to 24 dB, as our OTA test set.

Receiver Bit Error Rate Results

Figure 4 shows the BER yielded from classical MATLAB processing chain (MATLAB baseline). We also demonstrate the BER when only one of the channel estimator, demapper, or decoder blocks is replaced by the corresponding NN, as well as the end-to-end BER when all the classical blocks are replaced with NNs in the OFDM receiver chain.

We observe that channel estimator, demapper, and decoder NNs provide up to 86 percent, 36 percent, and 36 percent improvement in BER, respectively, compared to the MATLAB baseline. We observe that the cascade of the 3 NNs trained separately tagged as "Channel+Demapper+Decoder" in Fig. 4 shows 61 percent BER improvement compared to MATLAB baseline. We further perform another experiment where we train the 3 NNs jointly. We test the cascade of the trained NNs on the simulated data and show it as "Trained jointly" in Fig. 4, which improves MATLAB baseline BER by 77 percent.

Before testing the OFDM NN-based receiver on the OTA dataset, we need to re-train the channel estimator NN to learn the variations of the real wireless channel. However, the demapper and the decoder NNs do not need to be re-trained. This is because, as shown in Fig. 1, the channel effects are compensated by the equalizer before the data reaches the demapper and the decoder, and hence, demapper and decoder NNs perform independently of the wireless channel.

As explained earlier, we need data with very high SNR for training the channel estimator NN. Therefore, we collect additional OTA data of totally ∼35k packets between SNRs 25 and 37 dB, to re-train the channel estimator NN. Figure 5 shows the BER generated by processing the OTA data via the classical MATLAB processing chain (MATLAB baseline), as well as individual and cascaded NN BERs. For the OTA dataset, the channel estimator and the demapper NNs provide up to 12 percent and 20 percent BER improvement, respectively, over MATLAB baseline. This results in improved BER performance with the end-to-end NN-based receiver showing an average of 10 percent BER improvement compared to the MATLAB baseline.

FPGA Results

During the training phase that happens on GPUs, we apply the proposed BCR pruning to all the fully-connected and recurrent layers in the channel estimator, demapper, and decoder NNs. Then, we use the proposed MSQ to quantize the NN parameters. We run FPGA emulation to estimate FPGA resources for our models, which are shown in Table 1. We verify the speedup introduced by our compression methods on Ettus X310 SDRs with Xilinx Kintex7 T410 FPGA, by measuring the inference latency before and after NN compression. As shown in Table 1, the inference latency of the model decreases by 80 percent, 82 percent, and 81 percent after compression, for the channel estimator, demapper, and decoder NNs, respectively. We also measure the BER after compression and observe that compression increases the BER averagely by a negligible ratio of 3 percent, 1 percent, and 4 percent in the channel estimator, demapper and decoder NNs, respectively.

Computational Complexity

Finally, we compare the computational complexity of the proposed compressed NNs with the traditional counterparts, in terms of FLOPs.

Channel estimator: As explained earlier and shown in Fig. 1 (bottom), our channel estimator NN replaces both the FFT and the LS algorithms in the standard MATLAB pipeline. We estimate the traditional MATLAB channel estimation to have $\sim 4.4 \times 10^3$ FLOPs for our L-LTF length of 160 time-domain complex samples. By comparing this value to the compressed channel estimator NN FLOPs shown in the last row of Table 1, we observe that NN FLOPs are ∼50 × the traditional algorithm.

Demapper: The traditional demapping algorithm that we use in MATLAB is implemented through *approximate LLR* method.

NN block	Channel estimator	Demapper	Decoder
Model type	Dense	Dense	RNN+Dense
Layers (input size, output size)	Linear (160, 512) Linear (512, 256) Linear (256, 52)	Linear (2, 20) Linear (20, 4) (per-symbol)	bi-GRU ((2,256,3), 512) Linear (512, 16) Linear (16, 1)
Pruning Rate	2.0×	1.0×	2.0×
Weight bit-width	8	4	8
Overall size compression	8×	8×	8×
Working frequency	100 MHz	100 MHz	100 MHz
Non-compressed latency	1.67 *ms*	4.97 μs	210.04 μs
Compressed latency	0.33 *ms*	0.89 μs	38.19 μs
Non-compressed FLOPs	4.6×10^5	4.9×10^5	9.08×10^{10}
Compressed FLOPs	2.2×10^5	4.9×10^5	4.45×10^{10}

TABLE 1. Compression techniques, overall compression rate and FPGA speedup of the proposed NNs. FLOPs are reported for an L-LTF length of 160 time-domain samples in 5 MHz bandwidth as the Channel estimator NN, packet length of 4128 equalized symbols as the Demapper NN input, which yields 16512 soft-bits for the Decoder NN input.

We estimate the complexity of this algorithm to be 82 FLOPs per generated soft-bits for 16QAM, as this algorithm calculates the distance of each equalized symbol from all the known symbols in the constellation. The FLOPs add up to $\sim 1.3 \times 10^6$ for our packet length of 4128 equalized 16QAM symbols (4128 × 4 soft-bit). By comparing this value to the compressed demapper NN FLOPs shown in the last row of Table 1, we observe that NN FLOPs are ∼0.37× those of the traditional algorithm.

Decoder: The traditional decoding algorithm that we use in MATLAB is implemented through Viterbi algorithm, as explained earlier. We calculate the decoder function complexity to be $\sim 5.6 \times 10^8$ FLOPs for our decoder input length of 16512 soft-bits. By comparing traditional algorithm FLOPs to the compressed decoder NN FLOPs shown in the last row of Table 1, we observe that NN FLOPs are ∼79× the traditional algorithm.

The total FLOPs for the three compressed NN blocks add up to 4.45×10^{10} which is compared with total FLOPs for the traditional algorithms, 5.61×10^8. We observe that FLOPs count of the proposed compressed NNs are overall 79 × the cumulative FLOPs for the traditional algorithms. This opens up new research topics to study the tradeoff and identify switching instances, as discussed next.

Open Research Challenges

1. **Processing granularity:** In our proposed NN-based scheme, there is a difference between processing granularity for different NNs. The channel estimator processes one packet at a time, however, the demapper granularity is one equalized sample. This granularity gap opens up opportunities for ways to parallelize the operations via an ensemble of demapper NNs that demap successive samples in parallel. This brings up interesting resource-performance planning and tradeoffs in the choice of FPGA size versus the possible speedup in time.
2. **Performance under different environments or configurations:** Our results show that properly compressed NN-based receiver provides better BER performance and has fewer FLOPs compared to the traditional MATLAB receiver. However, the NN-based receiver has its limitations. For example, different channel estimators, demappers, and decoders need to be trained for different environments, different modulation schemes, and different coding rates, respectively. These further impose larger memory requirements to store weights for multiple NNs for each block. Methods such as transfer learning and life-long learning with pruning [15] can be explored to reuse and share a portion of NN weights among different configurations, and reduce large memory requirements.
3. **Identifying switching instances between classical and NN blocks:** We have shown that the NN-based OFDM receiver provides better performance compared to the classical one, in a variety of circumstances. However, a purer NN-based receiver can consume more resources and power compared to the traditional receiver. Since our NN-based receiver is modular, the logic that determines which modules to introduce into the receiver chain and when, is a completely new area of research. As an example, this decision may be made at runtime, based on desired reception performance and available on-board resources.

Conclusion

In this article, we proposed a model-driven design for NN-based OFDM receivers. Our receiver chain consists of 3 NNs for channel estimation, symbol to bit demapping, and error correction decoding. The NNs were designed based on wireless domain knowledge, and trained independently with data acquired from different data parts in the traditional transmitter and receiver. The trained networks were then cascaded to compose the complete receiver chain. The proposed NN-based receiver was evaluated with both simulated and OTA datasets, and showed averagely 61 percent and 10 percent improvement in BER compared to the traditional solution, when tested with simulated and OTA datasets, respectively. We further proposed two methods of pruning and quantization to compress our NNs and prepare them for FPGA implementation. We also showed that despite the BER performance gain, the proposed compressed NNs FLOPs are ∼79× their traditional counterparts. This complexity-performance tradeoff opens up new research opportunities as discussed earlier.

Acknowledgement

This work is supported by DARPA SPiNN HR00112090055, DARPA LwLL SC1821301, NSF 1923789, and NSF 1845833 awards.

References

[1] N. Soltani *et al.*, "Spectrum Awareness at the edge: Modulation Classification Using Smartphones," *2019 IEEE Int'l. Symp. Dynamic Spectrum Access Networks (DySPAN)*, 2019, pp. 1–10.
[2] N. Soltani *et al.*, "More is Better: Data Augmentation for Channel-Resilient RF Fingerprinting," *IEEE Commun. Mag.*, vol. 58, no. 10, 2020, pp. 66–72.
[3] N. Soltani *et al.*, "RF Fingerprinting Unmanned Aerial Vehicles with Non-standard Transmitter Waveforms," *IEEE Trans. Vehic. Tech.*, 2020.
[4] J. Zhang *et al.*, "Artificial Intelligence-Aided Receiver for A CP-Free OFDM System: Design, Simulation, and Experimental Test," *IEEE Access*, vol. 7, 2019, pp. 58,901–14.
[5] Z. Zhao *et al.*, "Deep-Waveform: A Learned OFDM Receiver Based on Deep Complex Convolutional Networks," arXiv preprint arXiv:1810.07181, 2018.
[6] F. A. Aoudia and J. Hoydis, "End-to-End Learning for OFDM: From Neural Receivers to Pilotless Communication," *IEEE Trans. Wireless Commun.*, 2021.
[7] H. Ye *et al.*, "Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems," *IEEE Wireless Commun. Letters*, vol. 7, no. 1, 2017, pp. 114–17.
[8] X. Gao *et al.*, "ComNet: Combination of Deep Learning and Expert Knowledge in OFDM Receivers," *IEEE Commun. Letters*, vol. 22, no. 12, 2018, pp. 2627–30.
[9] M. Belgiovine *et al.*, "Deep Learning at the Edge for Channel Estimation in Beyond-5G Massive MIMO," *IEEE Wireless Commun.*, 2021, pp. 1–7.
[10] M. Schaedler *et al.*, "Neural Network-Based Soft-Demapping for Nonlinear Channels," *020 Optical Fiber Commun. Conf. and Exhibition (OFC)*, 2020, pp. 1–3.
[11] H. Kim *et al.*, "Communication Algorithms via Deep Learning," *Int'l. Conf. Learning Representations*, 2018.
[12] A. Viterbi, "Error Bounds for Convolutional Codes and An Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Info. Theory*, vol. 13, no. 2, 1967, pp. 260–69.

[13] A. Ren *et al.*, "Admm-NN: An Algorithm-Hardware Co-Design Framework of DNNs Using Alternating Direction Methods of Multipliers," *Proc. 24th Int'.l Conf. Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 925–38.
[14] L. Bertizzolo *et al.*, "Arena: A 64-Antenna SDR-Based Ceiling Grid Testing Platform for Sub-6 GHz 5G-and-Beyond Radio Spectrum Research," Computer Networks, vol. 181, 2020, p. 107436.
[15] Z. Wang *et al.*, "Learn-Prune-Share for Lifelong Learning," *2020 IEEE Int'l. Conf. Data Mining (ICDM)*, 2020, pp. 641–50.

Biographies

Nasim Soltani (soltani.n@northeastern.edu) is a Ph.D. candidate at the Institute for Wireless IoT at Northeastern University, advised by professor Chowdhury. Her area of interest is AI-aided algorithms for applications in the physical layer of wireless communications systems.

Hai Cheng (cheng.hai@northeastern.edu) is a Ph.D. candidate in Computer Engineering at the Institute for Wireless IoT at Northeastern University. He received his B.Eng degree in 2015 from Xidian University, China, and master degree in 2018 from ShanghaiTech University, China. His research interests include machine learning and optimization in wireless network systems.

Mauro Belgiovine (belgiovine.m@northeastern.edu) is pursuing his Ph.D. at the Electrical and Computer Engineering department at Northeastern University, under the guidance of Professor Kaushik Chowdhury. His current research interests involve deep learning, wireless communications, and heterogeneous computing.

Yanyu Li (li.yanyu@northeastern.edu) is a Ph.D. candidate at the Department of Electrical and Computer Engineering in Northeastern University, advised by Professor Yanzhi Wang. His research interests include deep learning, neural network architecture search, pruning and quantization.

Haoqing Li (li.haoq@northeastern.edu) is a Ph.D. candidate in Electrical and Computer Engineering at Northeastern University, Boston, MA. got his BS degree in Electrical Engineering from Wuhan University, China and MS degree in Electrical and Computer Engineering at Northeastern University, Boston, MA. His research interests include GNSS signal processing, anti-jamming technology and robust statistics.

Bahar Azari (azari@ece.neu.edu) is a Ph.D. student at the center for signal processing, imaging, reasoning, and learning (SPIRAL) of Northeastern University. She received her B.Sc. in electrical engineering from the Amirkabir University of Technology in 2011 and her M.Sc. in telecommunications engineering from Politecnico di Milano in 2014. Her research interests include applied signal processing and machine learning with expertise in deep generative models and latent variable models.

Salvatore D'Oro (s.doro@ece.neu.edu) is a Research Assistant Professor with the Institute for the Wireless IoT at Northeastern University, USA. He received his Ph.D. from the University of Catania in 2015. He serves on the technical program committee of IEEE INFOCOM and the Elsevier Computer Communications journal. His research interests include optimization and learning in NextG systems.

Tales Imbiriba (talesim@ece.neu.edu) received his Doctorate degree from the Department of Electrical Engineering (DEE) of the Federal University of Santa Catarina (UFSC), Florianópolis, Brazil, in 2016. He served as a Postdoctoral Researcher at the DEE–UFSC and is currently a Postdoctoral Researcher at the ECE dept. of the Northeastern University, Boston, MA, USA. His research interests include audio and image processing, pattern recognition, kernel methods, adaptive filtering, and Bayesian Inference.

Tommaso Melodia (t.melodia@ece.neu.edu) is a Professor at Northeastern University. He has been named William Lincoln Smith Professor in recognition of his significant research contributions and exceptional leadership in the field of electrical and computer engineering. He is the Director of the Institute for the Wireless IoT, and the Director of Research for the PAWR Project Office. He received his Ph.D. degree in Electrical and Computer Engineering from Georgia Institute of Technology in 2007. His research focuses on modeling, optimization, and experimental evaluation of wireless networked systems. He serves as Editor-in-Chief for Computer Networks.

Pau Closas (pau.closas@ece.neu.edu) is an Assistant Professor at Northeastern University, Boston, MA. He received the MS and Ph.D. degrees in Electrical Engineering from UPC in 2003 and 2009. He also holds a MS in Advanced Mathematics from UPC, 2014. His primary areas of interest include statistical signal processing, robust stochastic filtering, and machine learning, with applications to positioning systems and wireless communications. He is the recipient of the 2014 EURASIP Best Ph.D. Thesis Award, the 9th Duran Farell Award, the 2016 ION Early Achievements Award, and a 2019 NSF CAREER Award.

Yanzhi Wang (yanz.wang@northeastern.edu is currently an Assistant Professor at the Department of ECE at Northeastern University, Boston, MA. His research focuses on model compression and platform-specific acceleration of deep learning architectures, maintaining the highest model compression rates on representative DNNs. He received the U.S. Army Young Investigator Program Award (YIP), Massachusetts Acorn Innovation Award, Ming Hsieh Scholar Award, and other research awards from Google, MathWorks, etc. His recent research achievement, CoCoPIE, can achieve real-time performance on almost all deep learning applications using off-the-shelf mobile devices, outperforming competing frameworks by up to 180X acceleration.

Deniz Erdogmus [SM] (erdogmus@ece.neu.edu) is a Professor of ECE at Northeastern University, Boston, MA. He received his Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, in 2002. He held a postdoctoral position at the University of Florida, until 2004. His researches focus on statistical signal processing and machine learning with applications to contextual signal/image/data analysis with applications in cyber-human systems.

Kaushik Chowdhury [M'09, SM'15] (krc@ece.neu.edu) is a Professor at Northeastern University, Boston, MA. He received his Ph.D. degree from Georgia Institute of Technology in 2009. 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.