NN-Defined Modulator: Reconfigurable and Portable Software Modulator on IoT Gateways

Jiazhao Wang and Wenchao Jiang, *Singapore University of Technology and Design;* Ruofeng Liu, *University of Minnesota;* Bin Hu, *University of Southern California;* Demin Gao, *Nanjing Forestry University;* Shuai Wang, *Southeast University*

https://www.usenix.org/conference/nsdi24/presentation/wang-jiazhao

This paper is included in the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation.

April 16–18, 2024 • Santa Clara, CA, USA

978-1-939133-39-7

Open access to the Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

NN-Defined Modulator: Reconfigurable and Portable Software Modulator on IoT Gateways

Jiazhao Wang[1], Wenchao Jiang[1], Ruofeng Liu[2], Bin Hu[3], Demin Gao[4], and Shuai Wang[5]

[1]*Singapore University of Technology and Design,* [2]*University of Minnesota,* [3]*University of Southern California,* [4]*Nanjing Forestry University,* [5]*Southeast University*

Abstract

A physical-layer modulator is a vital component for an IoT gateway to map the symbols to signals. However, due to the soldered hardware chipsets on the gateway's motherboards or the diverse toolkits on different platforms for the software radio, the existing solutions either have limited extensibility or are platform-specific. Such limitation is hard to ignore when modulation schemes and hardware platforms have become extremely diverse. This paper presents a new paradigm of using neural networks as an abstraction layer for physical layer modulators in IoT gateway devices, referred to as NN-defined modulators. Our approach addresses the challenges of extensibility and portability for multiple technologies on various hardware platforms. The proposed NN-defined modulator uses a model-driven methodology rooted in solid mathematical foundations while having native support for hardware acceleration and portability to heterogeneous platforms. We conduct the evaluation of NN-defined modulators on different platforms, including Nvidia Jetson Nano and Raspberry Pi. Evaluations demonstrate that our NN-defined modulator effectively operates as conventional modulators and provides significant efficiency gains (up to 4.7× on Nvidia Jetson Nano and 1.1× on Raspberry Pi), indicating high portability. Furthermore, we show the real-world applications using our NN-defined modulators to generate ZigBee and WiFi packets, which are compliant with commodity TI CC2650 (ZigBee) and Intel AX201 (WiFi NIC), respectively.

1 Introduction

In recent years, we have observed the swift progression of the Internet of Things (IoT), transitioning from theoretical concepts to tangible reality. IoT's objective is to connect many devices, such as sensors and actuators, globally via various Physical (PHY) layer technologies. These technologies are tailored to suit IoT connections based on factors like throughput, power consumption, and coverage area. For instance, IEEE 802.15.4 [19] is specifically designed for short-range, low-rate IoT connections, while NB-IoT [7] is intended for broader, low-power IoT connections. The IoT gateway functions as a central hub, establishing wireless communication links with IoT devices and bridging them to the rest of the Internet. Within PHY of IoT communication (Figure 1a), the modulator plays a vital role in generating signals for data transfer to transmit over the air.

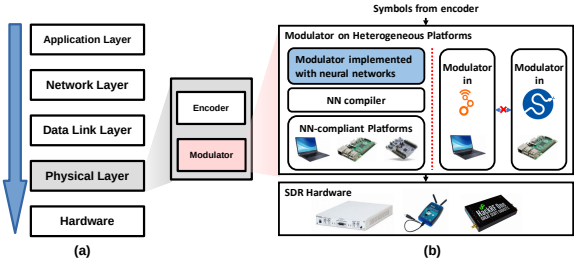


Figure 1: (a) Simplified layered model for IoT protocol stack. Our design is focused on the Physical layer. (b) Left: deployment of the modulator based on Neural Network. Right: deployment of the modulator based on SDR toolkits.

Therefore, it is essential for the gateway to be flexible, allowing it to support a variety of wireless technologies used by IoT devices and even accommodate emerging technologies for future readiness. However, numerous existing gateway designs [5, 21, 27, 43] employ hardware-based solutions, where wireless technologies are integrated into dedicated chipsets, which are either soldered onto the gateway's motherboard or connected through extension ports. Such hardware-based solutions offer limited adaptability, as their functions are fixed upon manufacturing. Software Defined Radio (SDR) is introduced as a flexible alternative for IoT gateways to address these limitations. Users can implement both current and emerging wireless transceivers as software, surpassing the extensibility of hardware-based solutions.

Despite the advantage of flexibility, SDR-based gateway design comes with several drawbacks. SDR-based designs consist of the radio frequency (RF) front-end and the computing device serving as the host device for the software radio application. The software radio application requires using signal processing toolkits or libraries, like GNURadio [4]

and SciPy [15]. Owing to the variety of development tools and host platforms, transferring the same functionality to a new platform demands a considerable learning curve and extensive effort in software development. Meanwhile, software radio comes at the cost of efficiency loss as we shift the signal processing from specialized hardware to general software systems. Many researchers and developers intend to optimize the software radio with the capability of hardware acceleration [26, 37, 38]. However, these works are targeted at specific platforms and require a considerable learning curve and extensive effort during development. Given the diversity of platforms and toolkits, deploying a highly efficient software modulator on multiple platforms becomes challenging.

To address these issues, we propose to develop a novel framework that facilitates the design of software transmitters on a variety of IoT gateways using neural networks. This approach would maintain the flexibility needed to accommodate a wide range of transmitters while enhancing portability and efficiency on different platforms. Our work is motivated by several interesting observations: **i)** the neural network module is widely supported across diverse hardware platforms due to the flourishing AI technologies, and **ii)** our research demonstrates that signal processing blocks within a modulator can be equivalently implemented using neural network models. These insights have led to the development of our innovative neural network-defined modulator [1], which offers a flexible and portable design for IoT gateways. The complete architecture of the design is depicted in Figure 1b. The proposed neural network-defined modulator functions are implemented by a unified neural network framework that can take advantage of accelerators across various platforms. In essence, the unified neural network framework operates as an abstraction layer for modulation tasks across heterogeneous platforms.

In summary, the original contributions of this paper are listed as follows:

- Conceptually, we propose an NN-defined physical layer modulator, which achieves high flexibility and extensibility to support multiple modulation schemes, and portability and efficiency on heterogeneous platforms.
- Technically, we adopt a model-driven approach to build the NN-defined modulators. The structure and parameters in the NN-defined modulators are rooted in a solid mathematics foundation from the modulation model.
- Experimentally, we deploy the NN-defined modulators on multiple hardware platforms (e.g., Nvidia Jetson Nano, Raspberry Pi) with extensive evaluations. We also employ our NN-defined modulators into the workflow of the IoT gateway to generate protocol-compliant signals, including ZigBee and WiFi.

[1]The code for reproduction is available at the anonymous repository: https://github.com/Repo4Sub/NSDI2024

2 Motivation

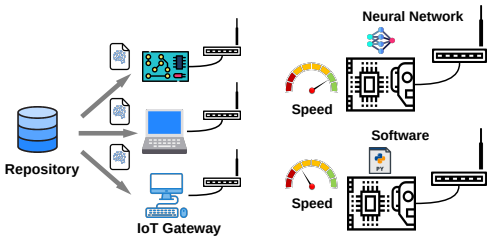
2.1 Problem Statement

Modern IoT gateways strive to offer adaptable transmitters to address the ever-evolving landscape of IoT connectivity technologies. Present IoT gateway solutions can be classified into hardware-based gateways and those based on software-defined radio (SDR). Hardware-based solutions, as the term implies, combine numerous chips/modules tailored for various connectivity technologies on a single board [5, 21, 27, 43]. While these hardware chips/modules exhibit merits such as cost-effectiveness and efficiency, they are limited by their lack of adaptability. This limitation stems from the fixed nature of technologies within the chips/modules and the restricted capacity for users to alter connectivity technologies. Gateway platforms can take the form of diverse devices, including personal computers, edge servers, and, increasingly, embedded computers, which can function as host devices for SDR. Consequently, it is possible to develop software radio for distinct IoT technologies, surpassing hardware-based solutions in terms of flexibility. Nevertheless, the multitude of development tools and deployment platforms for software radio can impede portability. For instance, GNU Radio establishes the signal processing blocks required for software radio construction, yet porting GNU Radio to embedded computers for IoT gateways is challenging, necessitating recompilation for target devices [28]. Moreover, platform/toolkit-specific implementations often depend on optimized designs that exploit acceleration capabilities, potentially resulting in efficiency loss when transferring SDR-based solutions to new platforms. For example, cuSignal [20] is a GPU-accelerated signal processing library, exclusively designed for devices equipped with NVIDIA GPUs, thus not providing a universal solution.

2.2 Opportunities

Our design is inspired by the extensive integration of AI frameworks and hardware across diverse computing platforms, which can serve as IoT gateways. Hardware manufacturers continuously enhance their devices to facilitate neural network deployments, incorporating specialized instruction sets [18] and distinct hardware accelerators [9], along with programming libraries that capitalize on these features. Concurrently, nearly all mainstream machine learning frameworks, such as Tensorflow [16] and PyTorch [13], endeavor to function across various operating systems and hardware architectures. Additionally, these frameworks encapsulate low-level acceleration libraries, promoting developers to speed up the execution of neural network models.

By constructing transmitters as neural network modules with widespread support, we can attain not only extensibility for an array of technologies but also portability and efficiency on platforms compatible with neural networks. For one thing, a gateway device can always update its supported modulation schemes by retrieving the corresponding neural network implementation from the repository server (Figure 2a). Si-



(a) (b)

Figure 2: (a) Different devices can retrieve various neural network models to update modulation schemes. (b) Neural network modulators can be accelerated to achieve better efficiency compared with software modulators.

multaneously, the neural-network-defined modulators are expected to achieve superior efficiency compared to traditional software modulators (Figure 2b), which are blessed by the advantages of the hardware accelerators.

2.3 Challenges

The primary technical challenge involves integrating signal processing blocks into neural network models. One direct method is to utilize general-purpose neural network models, such as fully-connected (FC) layers, as in the literature [46, 58]. Nevertheless, we contend that this approach has two principal disadvantages compared to the traditional digital modulation model. The operational mechanism of a general-purpose machine learning model is often perceived as a black-box approach [31], which raises concerns about its reliability.

To illustrate this, we present a straightforward example of modulators based on general-purpose neural networks. We develop an FC-based neural network model to modulate OFDM symbols and train it using the dataset gathered from the standard 64-S.C. (subcarrier) OFDM modulator. The FC-based OFDM modulator converges to a Mean Squared Error (MSE) loss of 1.5×10^{-6} for the training set, signifying that the generated signals from training symbols closely resemble the corresponding training signals. However, it fails to modulate new OFDM symbols from the test set. The produced signal samples are depicted in Figure 3. The output from the FC-based modulator substantially deviates from the standard signals. Although this is a simple case study, we can deduce from these results that the neural network ought to be meticulously designed and executed to achieve modulation tasks.

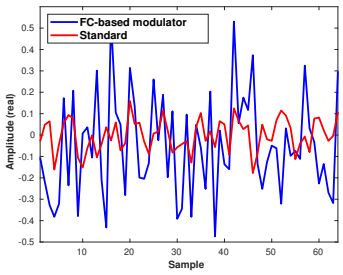


Figure 3: Waveform (real part) comparison of FC-based modulator and standard 64-S.C. OFDM modulator.

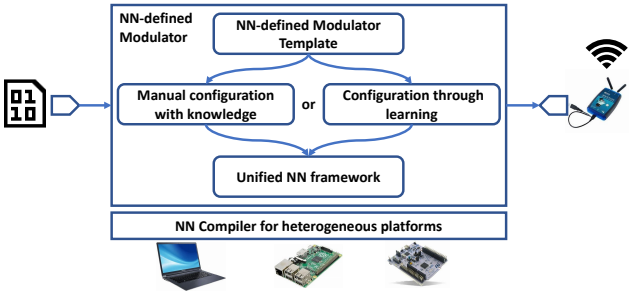


Figure 4: Architecture of NN-defined modulator.

Our research advocates for a model-driven strategy that integrates domain-specific knowledge of modulation techniques into the neural network design process. Instead of employing general-purpose neural network models or creating tailored neural network layers, our objective is to interpret fundamental neural network layers using domain-specific expertise regarding modulation schemes. By assembling the modulator with these neural network layers, which are comprehensively supported and efficiently implemented across diverse frameworks and platforms, we accomplish an interpretable, lightweight, and efficient neural network-based implementation for software modulators.

The architecture of the proposed Neural-Network-Defined (NN-defined) modulator is depicted in Figure 4. As in the figure, a modulator template (Section 3) rooted in solid mathematical foundations can be configured to implement specific modulation schemes either manually as in Section 4, or in a learning manner as in Section 5. Next, the NN-defined modulator will be transformed into a unified NN framework capable of executing across heterogeneous platforms (Section 6). The unified NN framework can be deployed onto various platforms and incorporated into the transmission pipeline (Section 7).

3 Template of NN-defined Modulator

In this section, we discuss how to use a model-driven methodology to construct neural networks for modulation tasks based on the underlying digital modulation models.

3.1 Mathematical Foundation of Digital Modulation

In wireless communication, a transmitter uses a modulator to convert symbols to signals before transmitting them to the air. The modulation process is usually analyzed through the Signal Space Analysis [29, 48], which is widely adopted in modeling amplitude/phase modulation techniques or named as linear modulation [29], including pulse amplitude modulation (PAM), phase-shift keying (PSK), and quadrature amplitude modulation (QAM). And the concepts are also applicable in modeling multicarrier modulation schemes, like OFDM.

Based on this method, a signal $S_i(t)$ modulated from a symbol s_i is considered a linear combination of the set of

basis functions. The synthesis process is given as

$$S_i(t) = \sum_{j=1}^{N} s_{ij}\phi_j(t) \tag{1}$$

where $\phi_j(t) \in \{\phi(t)\}^N$ is the j-th function in the set of N basis functions and s_{ij} is the j-th elements of the N-dimensional vector representation of the input symbol s_i. The basis functions and format of the symbol can be diverse and determine the different modulation schemes.

3.2 Modulator Template via Neural Network

After modeling the modulation process, we start to fit such a mathematical model into the neural network design to construct the proposed NN-defined modulator template.

3.2.1 Discrete-time Modulation Model

To accommodate the model into the neural networks, we first derive the discrete-time representation of the general model. The N-dimensional symbol vector s_i will be modulated to a series of signal samples as:

$$S_i[n] = \sum_{j=1}^{N} s_{ij}\phi_j[n] \tag{2}$$

where $\phi_j[n]$ is the discrete-time form of the basis functions. The symbols are processed sequentially, and the signal samples are concatenated into the final modulated signals for the whole symbol sequence, given as:

$$S[n] = \sum_i S_i[n - iL] \tag{3}$$

where L is the number of *samples per symbol*, meaning that L samples represent one symbol in the final modulated signals.

To extend Equation (2) to a complex I/Q signal, we have:

$$\begin{aligned} &S_I[n] + jS_Q[n] = \mathrm{Re}\{S_i[n]\} + j\mathrm{Im}\{S_i[n]\} \\ &= \sum_{j=1}^{N} [\mathrm{Re}\{s_{ij}\} + j\mathrm{Im}\{s_{ij}\}][\mathrm{Re}\{\phi_j[n]\} + j\mathrm{Im}\{\phi_j[n]\}] \\ &= \sum_{j=1}^{N} \mathrm{Re}\{s_{ij}\}\mathrm{Re}\{\phi_j[n]\} - \sum_{j=1}^{N} \mathrm{Im}\{s_{ij}\}\mathrm{Im}\{\phi_j[n]\} \\ &+ j(\sum_{j=1}^{N} \mathrm{Re}\{s_{ij}\}\mathrm{Im}\{\phi_j[n]\} + \sum_{j=1}^{N} \mathrm{Im}\{s_{ij}\}\mathrm{Re}\{\phi_j[n]\}) \end{aligned} \tag{4}$$

where S_I is the In-Phase signal and S_Q is the Quadrature. We can observe Equation (4) is composed of multiple $\sum_{j=1}^{N} s_{ij}\phi_j[n]$ patterns.

3.2.2 Basics of Transposed Convolutional Layer

Then we convert the $\sum_{j=1}^{N} s_{ij}\phi_j[n]$ pattern to a neural network. We find the *transposed convolutional layer* is a mathematically equivalent implementation. We first introduce the basic computation of a transposed convolutional layer in Figure 5.

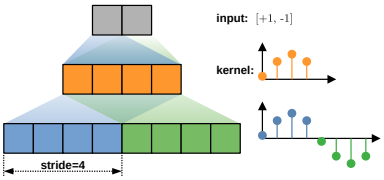


Figure 5: Diagram of the basic operation of the transposed convolutional layer.

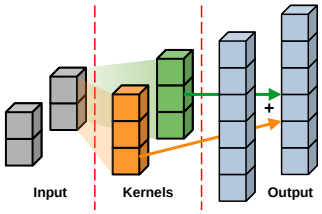


Figure 6: Diagram of the operation of multi-channel transposed convolutional layer.

The 1-D transposed convolutional layer has only 1 input channel and 1 output channel. The elements in input sequence $[+1,-1]$ are multiplied by a *kernel*. The multiplication results are mapped to the output. The step between each multiplication result is determined by the *stride* parameter.

The transposed convolutional layer supports multiple input channels and multiple output channels [13, 16]. In Figure 6, we visualize the operation of the transposed convolutional layer with multiple input and output channels (both are 2 in this figure). As illustrated here, each input channel will *convolve* with a set of 2 kernels, and the results are combined to generate one output channel. The calculation process of the transposed convolutional layer is the same as in one channel of Equation 4, *if the kernel is set to the same as the real/imaginary parts of the basis functions, i.e.,* $\mathrm{Re}\{\phi_j[n]\}$ *and* $\mathrm{Im}\{\phi_j[n]\}$*, and the stride is set to the samples per symbol, i.e., L as in Equation 3.*

3.2.3 NN-defined Modulator Template

With the transposed convolutional layer, we can express the whole modulation process in Equation (4) as an NN-defined template modulator in Figure 7. The input channel comprises real and imaginary parts of the symbol vectors, which form two *groups* of the transposed convolutional layer. The kernels of the transposed convolutional layer are determined by the basis functions. After that, a linear (fully-connected) layer is added to merge the four-channel outputs to generate the real and imaginary parts of the modulated signals. Its weight are set as $[+1,0,0,-1]$ and $[0,+1,+1,0]$ according to the coefficients in Equation (4) as shown in Figure 7.

Thus, we begin with the mathematical foundation of the modulation process and derive a generalized modulation model. Subsequently, we show how to adapt the general model within our template for the NN-defined modulator. The universal template comprises a transposed convolutional layer followed by a fully-connected layer. By meticulously con-

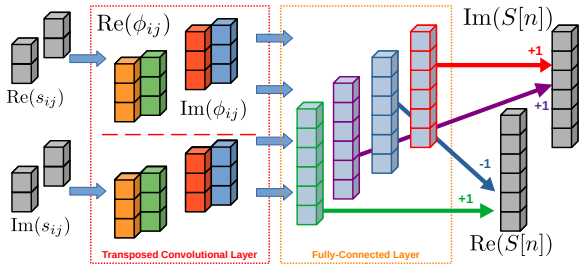


Figure 7: Diagram of the template of NN-defined modulator. 0-weight connections are omitted in the fully-connected layer.

figuring the template for the NN-defined modulator, we can accomplish a range of modulation schemes.

4 Instances of NN-defined Modulator Template

With the NN-defined modulator template, we further study how to generate specific modulation schemes.

4.1 Common NN-defined Modulators

4.1.1 Single Carrier Amplitude/Phase Modulation

For amplitude/phase modulation on the single carrier, the information is carried by the modulated signal's amplitude and/or phase. The most general case is quadrature amplitude modulation (QAM). For example, ZigBee [24] adopts Offset-Phase-Shift Keying (O-QPSK) as its modulation scheme, which is a variant of QPSK or 4-QAM scheme.

The QAM symbols are represented in a complex scalar as $s_k = \mathrm{Re}\{s_k\} + j\mathrm{Im}\{s_k\}$. The symbols pass a real-valued pulse-shaping filter to generate signals. Similar to equation (4), we represent the I/Q signals as in

$$\begin{aligned} S_I[n] &= \mathrm{Re}\{S[n]\} = \sum_k \mathrm{Re}\{s_k\} p[n-kL] \\ S_Q[n] &= \mathrm{Im}\{S[n]\} = \sum_k \mathrm{Im}\{s_k\} p[n-kL] \end{aligned} \tag{5}$$

where $p[n]$ represents the pulse-shaping filter, and L is the number of samples per symbol.

Based on Equation (4) and (5), when applying the NN-defined modulator template, we can configure the kernels of the transposed convolutional layer to be the values of shaping filter $p[n]$. This also implies the potential simplification of the template. If the shaping filter is real-valued, we can omit two channels of the transposed convolutional layer that correspond to the imaginary parts. We can also discard the fully-connected layer in the template because the output from the remaining 2 output channels from the transposed convolutional layer directly forms the desired modulated signals. For better illustration, an NN-defined QPSK modulator with a half-sine wave shaping filter is depicted in Figure 8. The output from the transposed convolutional layer is I/Q signals.

4.1.2 Multicarrier Modulation

We also extend our design for multicarrier modulation schemes, more specifically, the widely used OFDM scheme.

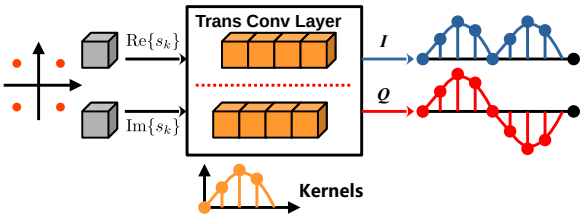


Figure 8: Diagram of a simplified NN-defined QPSK modulator with half-sine wave shaping filter.

We consider a N-S.C. OFDM modulator as an example, of which the input symbol vector consists of N elements, $s_0, s_1, \cdots, s_{N-1}$, that correspond to the components in the frequency domain. Thus, to get the signal samples $S[n]$, they are transformed to the time domain by performing an inverse Discrete Fourier Transform (IDFT) on the input N elements, given as

$$S[n] = \sum_{i=0}^{N-1} s_i e^{j2\pi ni/N},\ 0 \le n \le N-1. \tag{6}$$

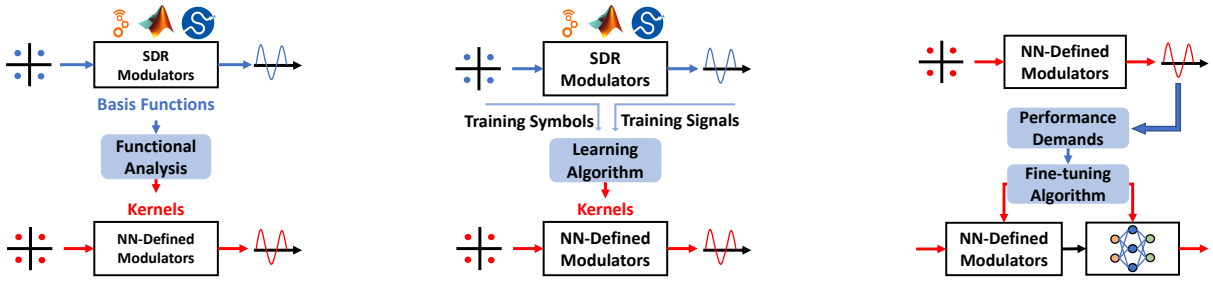
The transformation can be interpreted as mapping the complex symbol vector $\mathbf{s} = [s_0, s_1, \cdots, s_{N-1}]$ of N dimensions to signal $S[0], \cdots, S[N-1]$ with the basis functions set $\phi_i[n]$, which consists of N functions in total, like $\phi_i[n] = e^{j(2\pi ni/N)}$.

The OFDM scheme is consistent with the general case as in Equation (4). As discussed in Section 3, the input to the NN-defined modulator consists of real and imaginary elements from the complex symbol vectors. They are divided into 2 groups at the transposed convolutional layer. For each group, the kernels are set based on the real and imaginary parts of $e^{j2\pi ni/N}$. Then, the four-channel output from the transposed convolutional layer is fed into the fully-connected layer to generate the final In-phase and Quadrature signals.

4.2 Protocol-specified NN-defined Modulators

IoT protocol modulators may incorporate additional operations to enhance system reliability. For instance, ZigBee adopts an offset operation to the QPSK modulator by shifting the quadrature signals by half a symbol duration. OFDM systems used in WiFi adopt cyclic-prefix to improve robustness against multipath effects. Concurrently, some IoT protocols introduce intricate frame structures containing various fields for signaling. For example, WiFi frames typically encompass different signal fields. Although all these fields utilize the OFDM modulator, they may require different operations.

To address these additional operations, we draw inspiration from the inheritance feature in computer programming. The NN-defined modulators serve as the foundational component, and we attach operations to the temporal output from the base NN-defined modulator to generate the ultimate output signals. The attached processes are also achieved through operators supported by neural networks, allowing us to derive specialized NN-defined modulators for diverse protocols. We will discuss the protocol-specific NN-defined modulator in greater detail in Section 7.



(a) (b) (c)

Figure 9: Different approaches to configure kernels. (a) Manual setting with expert knowledge, (b) Learning from existing datasets, (c) Fine-tuning with other ML modules.

5 Modulator Kernel Configuration

From the previous sections, we designed a template for the general modulation model, where the kernels of the template can be derived for a specific modulation scheme. In this section, we will discuss how to use the NN-defined modulator template to learn from signals whose analytical expression is unknown or fine-tune the NN-defined modulator to compensate for hardware distortion in practical systems.

5.1 Manual Setting with Expert Knowledge

As shown in Figure 9a, for a modulation scheme with a known analytical expression, communication experts can take a direct way to derive the kernels of the transposed convolutional layer as discussed in Section 4. It is an efficient and accurate way to construct signals in the NN-defined modulator, similar to the conventional Software modulator in the SDR development.

5.2 Learning from Dataset

As shown in Figure 9b, for a signal with an unknown analytical expression or a non-expert developer, the kernels of the template can be derived by training the NN-defined modulator. For example, a non-expert developer who intends to shift an existing software radio to another platform can utilize the learning ability of the NN-defined modulator from the existing system to reconstruct the modulator, which will significantly ease the development complexity. One can treat it as a standard machine learning task to minimize the mean squared error. Thanks to the model-driven approach, the trained kernels imply a potential signal processing pipeline to mimic the target signal.

More specifically, the training input has the dimension of $[Batch\_size, 2 \times Symbol\_dimension, \ Sequence\_length]$, where $2 \times Symbol\_dimension$ indicates the input is represented using the real and imaginary parts. And the training output has the size of $[Batch\_size, Signal\_length, 2]$, where the 2 on the last dimension also indicates the real and imaginary parts of complex signals. There are $2 \times Symbol\_dimension$ kernels to train in total.

For demonstration, we apply the NN-defined modulator template to learn the 64-S.C. OFDM scheme. The NN-defined OFDM modulator is trained with the same training settings as the example FC-based modulator in Section 2 with the same dataset and training epochs. The training set contains 256 different OFDM symbol sequences, each of which represents 128 input complex symbols. The FC-based modulator is implemented with two fully-connected layers, with almost ~ 60000 trainable parameters in total. We calculate the mean squared error between the modulated signals from two kinds of modulators and the standard signals on the training and test sets, respectively. Both the FC-based modulator and our NN-defined modulator have tiny errors on the training set. Our NN-defined modulator outperforms the FC-based modulator significantly on the test set. We plot signals generated from our NN-defined and the FC-based modulator in Figure 10. As in the figure, our NN-defined modulator can modulate the symbols correctly, while the FC-based modulator fails. The NN-defined modulator has much fewer parameters to train compared with the FC-based modulator, and the parameters are physically meaningful, which ensures that our NN-defined modulator is more reliable.

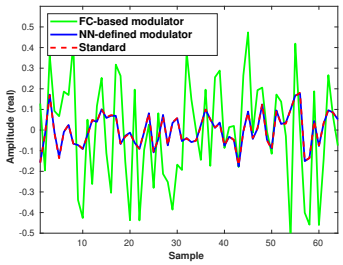


Figure 10: Waveform (In-phase) comparison of FC-based modulator, NN-defined modulator, and standard modulator for 64-S.C. OFDM scheme.

5.3 Fine-tuning for Better Performance

As shown in Figure 9c, the NN-defined modulator can be combined with extra AI/ML models to fine-tune to meet specific performance demands. During the fine-tuning procedure, the kernels of the NN-defined modulators and parameters within the appended AI/ML module are adjusted to fulfill the goals. The fine-tuning process is an open design because the performance demands and the AI/ML extra modules are diverse. For better illustration, we discuss combining the proposed NN-defined modulator with additional AI/ML modules to handle the hardware distortion in the transmitter systems.

The modulated signals are processed at the RF front-end in the transmitter systems to send over the air. Due to the characteristics of the circuits, there is some non-linearity in the RF front-end hardware, which will introduce distortion to the output signal compared with the ideal output. One efficient approach to reduce the distortion effect is to apply the predistortion process to the modulated signals before feeding into the RF front-end [52]. Here, we propose to use a neural network-based predistortion (NN-PD) module. Without loss of generality, we focus on the non-linearity introduced by the power amplifier.

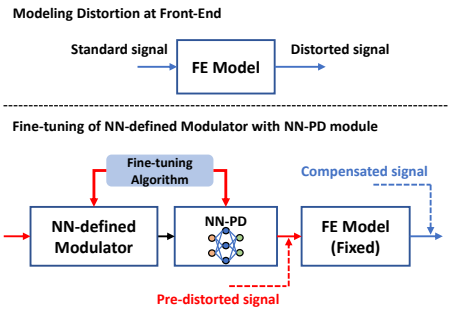


Figure 11: Diagram of Front-End model and NN-defined modulator with NN-PD module.

As illustrated in Figure 11, we first use a neural network, the front-end (FE) Model, to model the nonlinear behavior of the RF front-end. The FE model serves as the simulator of the RF front-end for the fine-tuning procedure. Next, we construct the NN-PD and insert it between the NN-defined modulator and the FE model. The predistorted signals from the NN-PD will pass the FE model, and the compensated signal is generated. The compensated signal is supposed to be as similar as possible to the ideal output signal. So, we set the training goal of our fine-tuning algorithm and tune the kernels in the NN-defined modulator and the parameters in NN-PD module while the parameters in the FE model are fixed. Once the fine-tuning procedure is finished, the NN-defined modulator and NN-PD module can generate predistorted signals, which can compensate for the non-linearity of the RF front-end.

As for verification, we compare the Bit Error Rate (BER) performance of QAM-modulated signals with predistortion and those without predistortion. The simulation is conducted in additive white Gaussian noise (AWGN) channel. We plotted the BER curves in Figure 12. The BER curve of the ideal signals is also visualized as the baseline. Furthermore, we conduct Error Vector Magnitude (EVM) test on the modulated signals. EVM can be evaluated by a percentage scale that reflects the deviation of the modulated and standard constellations. We measure the root mean squared EVM of the signals at different SNR levels. The results are illustrated in Table 1. When the SNR is low (SNR$<$ 0dB), the noise is dominant in such conditions, so all three signals suffer from noisy environments, resulting in high error rates and high EVM. However, when SNR is relatively high (SNR$>$ 0dB), the distortion effect of the RF front-end is more significant than the noise. Hence, the signals with predistortion perform much better than those without predistortion because the hardware distortion is reduced. However, the compensation is imperfect, so the error rates and EVMs of the predistorted signals are still slightly larger than the ideal signals. The above results indicate the great potential that the proposed NN-defined modulator can be integrated with other AI/ML modules and deliver better performance.

	SNR=-10dB	0dB	10dB
EVM of ideal signals	65.9%	31.2%	15.4%
w/ predistortion	66.6%	32.1%	15.7%
w/o predistortion	79.5%	33.4%	21.7%

Table 1: Root mean squared EVM of ideal modulated signals, signals with predistortion, and signals without predistortion.

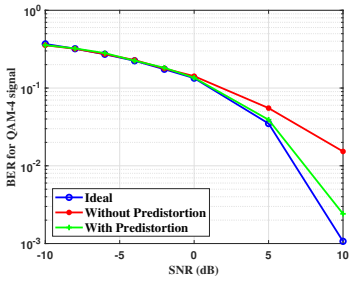


Figure 12: BER of NN-defined Modulator with NN-PD.

6 Modulator with Portability

We first highlight better portability of NN-defined modulators compared with conventional software radio systems by demonstrating the pipeline of the software modulator to generate signal samples. The software radio relies on some libraries which contain signal processing operations. We choose SciPy [15], a scientific computing library in Python, and GNURadio [4], the recognized software radio library, as case studies. Moreover, without loss of generality, we consider the QAM with Root Raised Cosine (RRC) pulse shaping filter as an example. It requires two major steps for modulation: upsampling and pulse-shaping filtering. We list the corresponding implementation for the GNURadio-based and the SciPy-based modulators in Table 2. As shown in the table, although the two kinds of implementations share the same pipeline, the functions used are quite different, which requires the developer to master new development tools for smooth conversion. Besides, we also notice that GNURadio provides several predefined shaping filter blocks, like `Root Raised Cosine Filter(rrc_fir)` for quick usage. In contrast, SciPy does not provide such predefined functions, so we need to configure the filter manually, which also increases the difficulty of porting.

Operations	GNURadio	SciPy
Upsampling	`interp_fir`	`scipy.interpolate`
Filtering	`rrc_fir`	`scipy.convovle`

Table 2: Operations for QAM modulator in different toolkits.

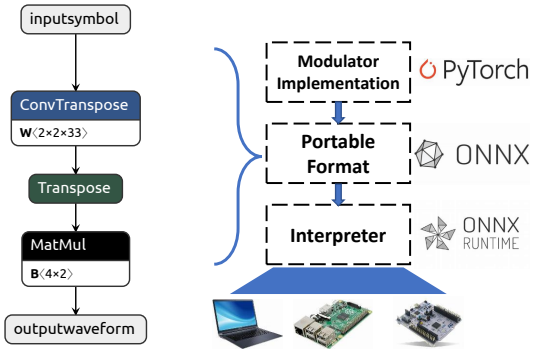
6.1 Framework-Independent NN-defined Modulators

Framework-dependent design implies that the NN-based modulators depend on unique functions or models provided by specific machine learning frameworks. Although machine learning frameworks offer some mathematical functions that can be employed to develop customized neural network layers for modulation tasks, as seen in NVIDIA Sionna [33] based on TensorFlow, the customized neural network modulator remains reliant on the development framework. To exemplify, we demonstrate the implementation details of the Sionna-based QAM modulator. Sionna employs the built-in operations and encapsulates them into the customized neural network layers, `Upsampling` and `Filter`, to emulate the functions as in the conventional pipeline. `Upsampling` layer applies `tf.pad` and some dimensional operations like `tf.expand_dims` to insert zeros between symbols, and `Filter` layer applies `tf.math.convolve` which takes the upsampled symbol sequences and filter taps as input to generate the modulated signals. We compare the availability of mathematical functions used in the Sionna modulator across other mainstream ML frameworks. The results are presented in Table 3. Although there are similar functions such as `pad` and `convolve`, the direct transition among different frameworks is still hard. Consequently, the framework-dependent modulator can be ported to platforms running the same framework, but it is challenging to deploy it on platforms operating with different frameworks.

	Tensorflow	PyTorch
NN-defined	`Conv1DTranspose`	`ConvTranspose1d`
	`Linear`	`Linear`
Sionna	`pad`	`pad+concatenate`
	`expand_dims`	`unsqueeze`
	`convolve`	`convolve`

Table 3: Original operations and converted ONNX operators in our NN-defined and Sionna QAM modulator.

Framework-independent design means our NN-based modulators are implemented by the share functions or models by various machine learning frameworks. Unlike NVIDIA Sionna, which constructs customized layers fro modulators, our NN-defined modulators utilize the fundamental neural network layers that are considered basic components of existing machine learning frameworks. More specifically, the transposed convolutional layer and the fully connected layer are generally supported by various frameworks [13, 16, 23]. Although the layer names vary (Table 3), they share the same functionalities, which ensures that the proposed NN-defined modulator can be a framework-independent design.



(a) (b)

Figure 13: (a) Example converted ONNX format of a QAM NN-defined modulator, (b) Diagram of development and deployment of NN-defined modulators.

We utilize the ONNX [10] as an intermediate framework to ensure the interoperability. ONNX is an open ecosystem for technology companies and research organizations to store and import neural network models onto different frameworks. ONNX defines a common set of operators that contains the fundamental layers of neural network models, including the transposed convolutional layer and the fully-connected layer used in our design. As a validation, we visualize the graph of the ONNX model of our NN-defined modulator template in Figure 13a. As depicted in the figure, the transposed convolutional layer operator is *ConvTranspose*, and the fully-connected layer is represented by the *MatMul* operator. Almost all mainstream machine learning frameworks support conversions between their native models and ONNX ones. It is also worth noting that porting customized neural network layers to ONNX models demands significant effort. Consequently, the custom layers in NVIDIA Sionna are challenging to convert to ONNX models, while our NN-defined modulator built upon fundamental layers exhibits better interoperability across different frameworks.

6.2 Seamless Acceleration

As previously illustrated, IoT gateway hardware platforms provide acceleration capabilities to expedite the execution of neural network models. The proposed NN-defined modulator is constructed based on fundamental neural network layers that are generally supported and well-optimized for execution on various hardware platforms. Therefore, the NN-based modulator can leverage these capabilities to enhance efficiency, speeding up the modulation process.

A typical development and deployment workflow for the proposed NN-defined modulators is depicted in Figure 13b. The prototype of the NN-defined modulators can be developed in mainstream machine learning frameworks, such as PyTorch. Then, NN-defined modulators are converted to a portable ONNX format for improved interoperability across different platforms. Deploying ONNX models requires a compiler, such as ONNX runtime [11] or Apache TVM [2]. Using

Figure 14: Prototype of NN-defined modulator. Left box: Nvidia Jetson Nano as the host device for the NN-defined modulator. Right box: ADI Pluto SDR as SDR hardware.

ONNX runtime as an example, it can utilize different accelerator backends. Numerous accelerator backends have been developed by the community. For instance, it can employ Nvidia GPU [8] on GPU-equipped systems, Arm ACL [3] for Arm SoC platforms, and OpenVINO [12] for Intel x86 platforms. Therefore, the NN-defined modulator can be seamlessly accelerated on various platforms.

7 Evaluation

7.1 Implementation

7.1.1 Framework and hardware

We design the NN-defined modulator in PyTorch [13] with `ConvTranspose1d` and `Linear` layers. Once the NN-defined modulators are ready for port, we convert the modulators into ONNX format. We port the ONNX NN-defined modulator to Nvidia Jetson Nano [9] and Raspberry Pi [14] for verification. Both devices support the ONNX runtime, and Jetson Nano is equipped with a GPU, which can be used to accelerate the execution of the ONNX NN-defined modulators.

Besides, we also implement an NN-defined modulator prototype. We connect the host (Nvidia Jetson Nano) running the NN-defined modulator with the SDR hardware (ADI Pluto SDR [1]) as shown in Figure 14. We use this prototype to transmit the modulated signals over the air.

7.1.2 Modulation schemes

Without loss of generality, we choose several typical schemes for 1) PAM-2 with the rectangular filter, 2) QPSK with the half-sine wave filter, 3) 16-QAM with RRC filter for amplitude/phase modulation, and 4) 64-S.C. OFDM scheme for multicarrier modulation. We use MATLAB Signal Processing Toolbox [6] to generate the symbols and the signals as for training sets. When evaluating the efficiency and portability, we select 16-QAM modulator with RRC filter as the example. The conventional SDR modulators for 16-QAM with RRC filter are implemented with signal processing libraries, GNU-Radio [4] on x86 laptop, and SciPy [15] on Nvidia Jetson Nano, as the baselines. For comparison, we also implement a 16-QAM modulator with RRC filter using Nvidia Sionna [33] on the x86 laptop for comparison.

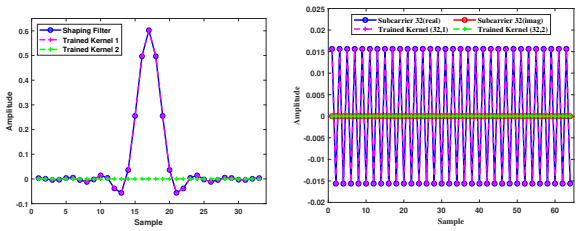
7.2 Signal Quality of NN-defined Modulator

7.2.1 Trained kernels in NN-defined modulators

As discussed in Section 5, the kernels within the NN-defined modulator can be trained with training sets. Here, we use 16-QAM with RRC filter and 64-S.C. OFDM scheme as examples to analyse the trained kernels.

For the 16-QAM scheme with RRC shaping filter, the input symbol is 1-dimensional, so there are 2 kernels trained. According to the analysis in Section 4, the trained kernels are supposed to be the real and imaginary parts of the shaping filter. We visualize trained kernels and the original RRC shaping filter in Figure 15a. One of the trained kernels is nearly identical to the original shaping filter. The other one is almost zero-valued, which is consistent with the zero-valued imaginary parts of the shaping filter.

For the 64-S.C. OFDM scheme, there are 2×64 kernels trained. According to the analysis in Section 4, the kernels are supposed to be the real and imaginary parts of the subcarrier functions, i.e., $e^{j\frac{2\pi in}{64}}$. We also visualized a pair of the trained kernels in Figure 15b. And these two kernels are the same as real and imaginary parts of the standard subcarrier $e^{j\frac{2\pi\times 32n}{64}}$. The NN-defined OFDM modulators share the same conclusion that the trained kernels perfectly match the signal processing pipeline in conventional modulators.



(a) (b)

Figure 15: Trained kernels from NN-defined modulators for (a) QAM with RRC filter, (b) 64-S.C. OFDM.

7.2.2 Transmission performance in AWGN channel

We apply the trained NN-defined modulators to generate signals and pass the signals in the additive white Gaussian noise (AWGN) channel to verify the transmission performance. And we plot the Bit Error Rate (BER) curves in Figure 16. Meanwhile, The BER curves of the signals from standard modulators in MATLAB are also plotted as the baseline. As illustrated in the figure, the NN-defined modulators for the selected modulation schemes can modulate the symbols correctly so that the modulated signals can achieve the same error performance as standard modulators in AWGN channels.

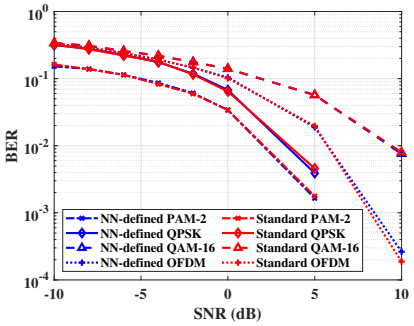


Figure 16: The BER performance of NN-defined modulator compared with standard modulators.

7.3 Efficiency and Portability

7.3.1 Efficiency improvement

To verify the efficiency improvement of the NN-defined modulator, we measure the running time of the conventional SDR QAM modulator, the NN-defined QAM modulator, and the Nvidia Sionna QAM modulator on an x86 laptop and compare these time recordings in Figure 17. All the QAM modulators modulate a batch consisting of 32 symbol sequences with 256 symbols.

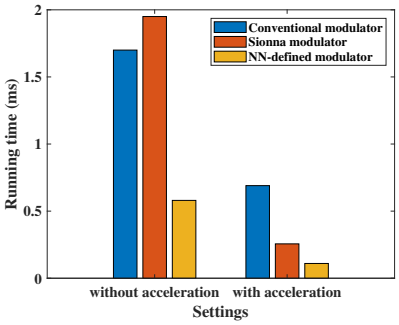


Figure 17: Running time of different implementations.

When all three modulators run without acceleration, it takes 0.58ms for our design to finish, which is much faster than the conventional (1.7ms) and Sionna modulator (1.9ms). Our NN-defined modulator applies the fundamental neural network layers that are well-optimized so that it performs better than the customized neural network layers implemented in Nvidia Sionna as well as the conventional SDR modulator.

The NN-defined and Sionna modulators support hardware acceleration thanks to the neural network implementation. We measure their running time with acceleration enabled. We also implement an accelerated QAM modulator with cuSignal [20]. The NN-defined modulator and Sionna modulator execute much faster than without acceleration. The running time of the NN-defined modulator is reduced to 0.059ms from 0.58ms, which is 28 times faster than the conventional SDR modulators without acceleration, even 10 times faster than the implementation using cuSignal. For the Sionna modulator, the running time is also reduced to 0.25ms. Both our NN-defined modulator and Sionna modulator run faster than the conventional modulator. These results prove that the NN-defined modulator can significantly improve efficiency compared with conventional SDR modulators.

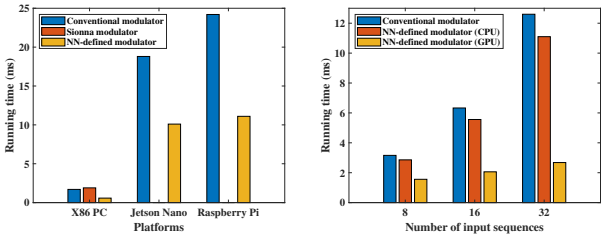
7.3.2 Portability

Porting among platforms. As aforementioned in Section 6, porting the conventional SDR implementations and Sionna-based one from one platform to another requires considerable effort. Here, we focus on the portability of our NN-defined modulator. Following the development diagram of the NN-defined modulators, we first implement the NN-defined QAM modulator in PyTorch and convert it into the ONNX model. We list the converted operations in the ONNX framework in Table 4. The transposed convolutional layer (`torch.ConvTranspose1d`) and the linear layer (`torch.Linear`) are widely supported so that they can be converted to the portable format.

Implementations	PyTorch layer	ONNX operator
NN-defined	`ConvTranspose1d`	`ConvTranspose`
	`Linear`	`MatMul`

Table 4: Original operations and converted ONNX operators in the NN-defined modulator.

Performance on different platforms. We now deploy the ONNX NN-defined QAM modulator on embedded computers such as Nvidia Jetson Nano and Raspberry Pi. Figure 18a illustrates the running time on different platforms. Sionna modulator fails to be ported because the customized layers are hard to be transformed into ONNX models. Although the running time of the NN-defined modulator on embedded systems is longer than that on the x86 laptop, we successfully port our NN-defined modulators to different platforms.



(a) (b)

Figure 18: (a) Running time on different platforms of x86 PC, Nvidia Jetson Nano, Raspberry Pi. (b) Acceleration evaluation on the target platform of Nvidia Jetson Nano.

The following evaluation demonstrates the acceleration capability of the target platform. We configure the ONNX NN-defined QAM modulator on Nvidia Jetson Nano to run with GPU acceleration as discussed in Section 6. We compare the running time of the conventional modulator and our NN-defined modulator modulate symbol batches of different sizes. The evaluation results are visualized in Figure 18. We can observe a considerable efficiency improvement compared with the conventional modulator as well as the CPU-only

NN-defined modulator. Moreover, the efficiency of our NN-defined modulator is still much better than the conventional modulator implemented with an accelerated signal processing library. More specifically, when the number of input symbol sequences is 32, the accelerated NN-defined modulator is 4.7 times faster than the conventional modulator and even 2.5 times faster than the accelerated modulator. These results showcase that we can easily run the NN-defined modulators on target platforms with acceleration capability.

7.4 Application in IoT Technologies

The proposed NN-defined modulators are employed to generate protocol-compliant signals, showcasing representative use cases in IoT gateways.

7.4.1 ZigBee-compliant Signals

ZigBee [24], developed based on IEEE 802.15.4 [19], utilizes Offset-QPSK, a variant of amplitude/phase modulation, as its modulation scheme. The diagram of the O-QPSK modulator is illustrated in Figure 19. The modulator input comes from a 4-QAM constellation, where the symbols are $\{\pm 1 \pm 1j\}$. The real and imaginary parts of the input symbols are processed separately to generate I/Q signals. The quadrature branch of signal samples is shifted by a delay to introduce the offset. As evident in the output waveform, the quadrature branch exhibits a slight lag.

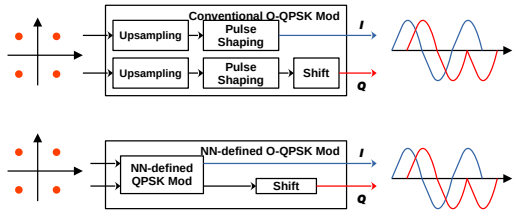
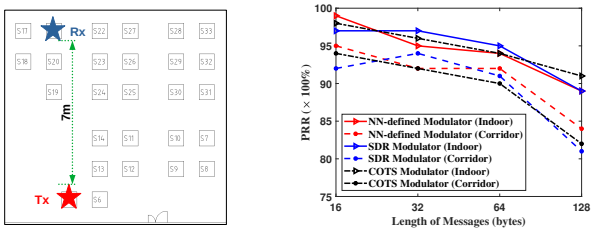


Figure 19: Diagram of conventional O-QPSK modulator (top) and the NN-defined O-QPSK modulator (bottom).

To construct an O-QPSK modulator for ZigBee protocol, we combine the NN-defined QPSK modulator with a shifting process to form the NN-defined O-QPSK modulator, as depicted in Figure 19. We generate symbols from messages following the specification and feed them into our NN-defined O-QPSK modulator. The modulated signals are sent over the air utilizing the prototype in Figure 14. We employ the TI CC2650 Kit [17] as the ZigBee receiver, which can parse the captured signals into messages.

We generate ZigBee packets with varying message lengths and transmit 100 packets. At the receiver side, the received ZigBee packets without errors are recorded, and we calculate the packet reception ratio (PRR) in different settings, repeating the evaluation 5 times. We compare the performance with the SDR implementation using signal processing libraries. And we conduct the same experiments on commercial off-the-shelf (COTS) TI devices as a baseline. The evaluation is conducted indoors and outdoors. The settings of indoor environments are demonstrated in Figure 20a. As depicted in Figure 20b, the ZigBee signals generated by the NN-defined modulator can be successfully received by the commercial device, achieving performance comparable to the existing SDR implementation and commercial devices.



(a) (b)

Figure 20: (a) Evaluation settings in indoor environment. (b) Packet Reception Ratio of ZigBee packets modulated by NN-defined modulator and SDR modulator.

7.4.2 WiFi-compliant Signals

WiFi, which is also extensively utilized for IoT communication, typically employs the OFDM scheme. When implementing the NN-defined modulator for WiFi communication, the process becomes slightly more complex, as WiFi utilizes the CP-OFDM [48] modulator, and WiFi frames generally consist of signals generated from various fields.

Taking IEEE 802.11a/g as an example in Figure 21, WiFi frames comprise four fields: Short Training Field (STF), Long Training Field (LTF), Signaling Field (SIG), and Data Field (DATA). The STF and LTF primarily serve detection, synchronization, and channel estimation purposes at the receiver. The SIG contains information about the current frame, such as frame length and modulation and coding scheme information, while the DATA field carries the data.

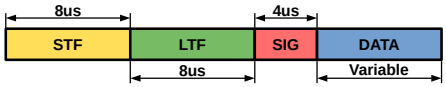


Figure 21: Fields in IEEE 802.11a/g frame.

Different fields need specific operations. The STF and LTF involve repeating the signals from the OFDM modulator, while the SIG and DATA require adding a cyclic prefix to the modulated signals by copying the ending parts of the OFDM signals to the front. Following the discussion in Section 4, we combine an NN-defined OFDM modulator with additional operations to add the cyclic prefix.

Four NN-defined modulators corresponding to the four fields in IEEE 802.11a/g WiFi frames are implemented. These modulators are then combined to create a single NN-defined WiFi modulator. The overall structure is illustrated in Figure 22. The NN-defined modulators for `STF`, `LTF`, `SIG`, and `DATA` fields collectively form the NN-defined WiFi modulator, allowing for a comprehensive modulation process that addresses the unique requirements of each field.

We generate beacon packet signals using the NN-defined WiFi modulator and transmit them over the air. A laptop is

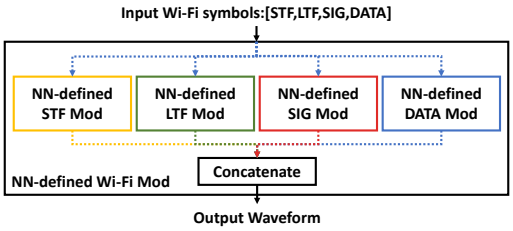


Figure 22: NN-defined Wi-Fi modulator.

used to sniff the beacon packets. We test beacon reception in an indoor environment at the 5GHz band, transmitting 100 beacon packets for 5 times. Figure 23 demonstrates that the laptop can successfully receive the beacon with an SSID of "NN-definedModulator", achieving a PRR at 96%.

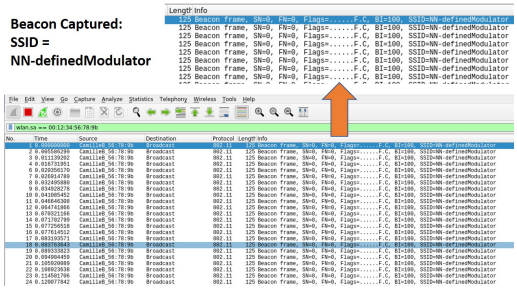


Figure 23: Reception of beacon signals generated by NN-defined Wi-Fi modulator.

Next, we extend our design to transmit data by generating data packets and passing the signals through simulated AWGN channels. We follow the standard process to detect and synchronize WiFi frames using STF signals, conduct channel estimation and equalization using LTF signals, and then demodulate and decode the SIG and DATA signals. In our evaluation, the symbols for a grayscale image data are generated using 16-QAM and 64-QAM. The results are listed in Figure 24. As shown in the figure, we can successfully reconstruct the transmitted images under different settings, further demonstrating the effectiveness and versatility of our NN-defined modulator design in practical applications.

(a) (b) (c)

Figure 24: (a) Original image of 256×256 pixels; Received images using (b) 16-QAM at SNR=10dB and (c) using 64-QAM at SNR=20dB.

8 Related Works

SDR solutions for IoT gateway: SDRs are proposed as universal gateways operating across technologies. Past work [22, 51, 59] has developed smart home gateways using the USRP radio with GNUradio support. [45] revisited the SDR-based IoT gateway for decoding collapsed packets. Other solutions employed the cross-technology communication technique as an alternate for IoT gateways [34–36, 39–42, 53–56].

Machine learning for communication system: Neural networks or machine learning has been extensively used in physical layer designs [25, 31, 44, 47, 49, 57, 61]. [46] introduced a method to learn an end-to-end communication system by interpreting it as an autoencoder [30]. In [60], a DNN model replaces all blocks in the conventional OFDM receiver. In [50], the researchers propose to replace processing blocks in the OFDM receiver with neural network models and deploy them on IoT devices.

Our work has innovations in two tiers, distinctive objectives, and different methodologies. Objective-wise, most literature views the neural network as an optimizer and seeks performance gains under complex conditions [61]. In contrast, we use the neural network as an abstraction layer for the portability of IoT gateway functionalities. Methodology-wise, the literature commonly adopts data-driven approaches that employ general-purpose neural networks [25, 49, 57]. We adopt a model-driven approach that designs the neural network-based modulators with reference to the mathematical models.

9 Discussion

It's worth pointing out that we only discuss the linear amplitude/phase modulation schemes in this paper. Other modulation schemes require further study, such as frequency modulation, also known as non-linear modulation. Following the similar idea, We can model the frequency modulation based on the phase changes and construct another NN-defined modulator template that can be used for the Gaussian frequency shift keying (GFSK) modulators used in Bluetooth [32]. Moreover, we intend to extend the application of the learning ability.We can further apply the neural network to learn to reduce the adjacent channel leakage ratio (ACLR) for single carrier scheme or to reduce the peak-average power ratio (PAPR) for OFDM scheme. We can also apply the NN-defined modulator to learn from noisy signal samples to reconstruct noiseless modulators. The model-driven approach can also be applied to the receiver design, including demodulation and decoding, which is an emerging topic in wireless communication.

10 Conclusion

In this paper, we present an NN-defined modulator template for various modulation schemes that can be converted to a unified NN framework for portable deployment for IoT gateway design. The proposed NN-defined modulator has the extensibility to achieve various modulation schemes for IoT connections, and the evaluation results show that they can perform well. The NN-defined modulator outperforms the existing SDR solutions in terms of portability and efficiency thanks to the wide support of the NN on heterogeneous computing platforms. Meanwhile, the NN-based implementation also enables our design with the learning ability, featuring the potential for intelligent communication systems.

Acknowledgments

We extend our heartfelt thanks to the reviewers for their rigorous and constructive feedback and the shepherding committee for their invaluable guidance. Special appreciation goes to our collaborator, Shuai Wang from George Mason University, whose diverse skills also significantly enriched this work. This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MOE-T2EP20221-0017), National Natural Science Foundation of China under Grant No.62272098 and The Future Network Scientific Research Fund Project (Grant No. FNSRFP-2021-YB-17). This research is also supported by the National Research Foundation, Singapore, and Infocomm Media Development Authority under its Future Communications Research & Development Programme.

References

[1] Adi adalm-pluto. https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html.

[2] Apache tvm. https://tvm.apache.org/.

[3] Arm compute library. https://www.arm.com/technologies/compute-library.

[4] Gnu radio. https://www.gnuradio.org/.

[5] Hardwario iot platforms. https://www.hardwario.com/.

[6] Matlab signal processing toolbox. https://www.mathworks.com/products/signal.html.

[7] Narrowband–internet of things (nb-iot). https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/.

[8] Nvidia cuda toolkit.

[9] Nvidia jetson nano developer kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[10] Onnx: Open neural network exchange. https://onnx.ai.

[11] Onnx runtime. https://onnxruntime.ai/.

[12] Openvino. https://docs.openvino.ai/latest/home.html.

[13] Pytorch. https://pytorch.org/.

[14] Raspberry pi. https://www.raspberrypi.com/.

[15] Scipy. https://www.scipy.org/.

[16] Tensorflow. http://tensorflow.org/.

[17] Texas instruments launchxl-cc2650. https://www.ti.com/tool/LAUNCHXL-CC2650.

[18] Up edge computing kit. https://www.aaeon.com/en/c/up-edge-computing-kit.

[19] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, 2016.

[20] Cusignal. https://developer.nvidia.com/blog/accelerated-signal-processing-with-cusignal, 2023.

[21] Amiruddin Amiruddin, Anak Agung Putri Ratna, Ruki Harwahyu, and Riri Fitri Sari. Secure multi-protocol gateway for internet of things. In *2018 Wireless Telecommunications Symposium (WTS)*, pages 1–8, 2018.

[22] Carlos J Bernardos, Antonio De La Oliva, Pablo Serrano, Albert Banchs, Luis M Contreras, Hao Jin, and Juan Carlos Zúñiga. An architecture for software defined wireless networking. *IEEE wireless communications*, 21(3):52–61, 2014.

[23] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[24] Sinem Coleri Ergen. Zigbee/ieee 802.15. 4 summary. *UC Berkeley, September*, 10(17):11, 2004.

[25] Xuanxuan Gao, Shi Jin, Chao-Kai Wen, and Geoffrey Ye Li. Comnet: Combination of deep learning and expert knowledge in ofdm receivers. *IEEE Communications Letters*, 22(12):2627–2630, 2018.

[26] Georgios Georgis, Alexios Thanos, Marcin Filo, and Konstantinos Nikitopoulos. A dsp acceleration framework for software-defined radios on x86 64. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1648–1652. IEEE, 2020.

[27] Egidio Gioia, Pierluigi Passaro, and Matteo Petracca. Amber: An advanced gateway solution to support heterogeneous iot technologies. In *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5. IEEE, 2016.

[28] Gwenhael Goavec-Merou. Gnuradio running on embedded boards: porting to buildroot. *Proceedings of the GNU Radio Conference*, 2021.

[29] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[30] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[31] Hengtao He, Shi Jin, Chao-Kai Wen, Feifei Gao, Geoffrey Ye Li, and Zongben Xu. Model-driven deep learning for physical layer communications. *IEEE Wireless Communications*, 26(5):77–83, 2019.

[32] Robin Heydon and Nick Hunn. Bluetooth low energy. *CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx*, 2012.

[33] Jakob Hoydis, Sebastian Cammerer, Fayçal Ait Aoudia, Avinash Vem, Nikolaus Binder, Guillermo Marcus, and Alexander Keller. Sionna: An open-source library for next-generation physical layer research. *arXiv preprint arXiv:2203.11854*, 2022.

[34] Wenchao Jiang, Song Min Kim, Zhijun Li, and Tian He. Achieving receiver-side cross-technology communication with cross-decoding. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 639–652, 2018.

[35] Wenchao Jiang, Zhimeng Yin, Song Mim Kim, and Tian He. Transparent cross-technology communication over data traffic. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[36] Wenchao Jiang, Zhimeng Yin, Ruofeng Liu, Zhijun Li, Song Min Kim, and Tian He. Bluebee: a 10,000 x faster cross-technology communication via phy emulation. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, pages 1–13, 2017.

[37] Tarik Kazaz, Christophe Van Praet, Merima Kulin, Pieter Willemen, and Ingrid Moerman. Hardware accelerated sdr platform for adaptive air interfaces. *arXiv preprint arXiv:1705.00115*, 2017.

[38] Kaipeng Li, Bei Yin, Michael Wu, Joseph R Cavallaro, and Christoph Studer. Accelerating massive mimo uplink detection on gpu for sdr systems. In *2015 IEEE dallas circuits and systems conference (DCAS)*, pages 1–4. IEEE, 2015.

[39] Zhijun Li and Tian He. Webee: Physical-layer cross-technology communication via emulation. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 2–14, 2017.

[40] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. Lte2b: Time-domain cross-technology emulation under lte constraints. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 179–191, 2019.

[41] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. Xfi: Cross-technology iot data collection via commodity wifi. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2020.

[42] Ruofeng Liu, Zhimeng Yin, Wenchao Jiang, and Tian He. Wibeacon: Expanding ble location-based services via wifi. In *Proceedings of the 27th annual international conference on mobile computing and networking*, pages 83–96, 2021.

[43] Roberto Morabito, Riccardo Petrolo, Valeria Loscrì, and Nathalie Mitton. Legiot: A lightweight edge gateway for the internet of things. *Future Generation Computer Systems*, 81:1–15, 2018.

[44] Tianjie Mu, Xiaohui Chen, Li Chen, Huarui Yin, and Weidong Wang. An end-to-end block autoencoder for physical layer based on neural networks. *arXiv preprint arXiv:1906.06563*, 2019.

[45] Revathy Narayanan and Swarun Kumar. Revisiting software defined radios in the iot era. In *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, pages 43–49, 2018.

[46] Timothy O'shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

[47] Timothy O'shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

[48] John G. Proakis. *Digital Communications*. McGraw-Hill New York, 2007.

[49] Morteza Soltani, Wael Fatnassi, Ahmed Aboutaleb, Zouheir Rezki, Arup Bhuyan, and Paul Titus. Autoencoder-based optical wireless communications systems. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6. IEEE, 2018.

[50] Nasim Soltani, Hai Cheng, Mauro Belgiovine, Yanyu Li, Haoqing Li, Bahar Azari, Salvatore D'Oro, Tales Imbiriba, Tommaso Melodia, Pau Closas, et al. Neural network-based ofdm receiver for resource constrained iot devices. *arXiv preprint arXiv:2205.06159*, 2022.

[51] Manolis Surligas, Antonis Makrogiannakis, and Stefanos Papadakis. Empowering the iot heterogeneous wireless networking with software defined radio. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2015.

[52] Chance Tarver, Liwen Jiang, Aryan Sefidi, and Joseph R Cavallaro. Neural network dpd via backpropagation through a neural network model of the pa. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 358–362. IEEE, 2019.

[53] Shuai Wang, Jianlin Guo, Pu Wang, Kieran Parsons, Philip Orlik, Yukimasa Nagai, Takenori Sumi, and Parth Pathak. X-disco: Cross-technology neighbor discovery. In *2022 19th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 163–171. IEEE, 2022.

[54] Shuai Wang, Woojae Jeong, Jinhwan Jung, and Song Min Kim. X-mimo: Cross-technology multi-user mimo. In *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pages 218–231, 2020.

[55] Shuai Wang, Song Min Kim, and Tian He. Symbol-level cross-technology communication via payload encoding. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 500–510. IEEE, 2018.

[56] Shuai Wang, Zhimeng Yin, Zhijun Li, Yongrui Chen, Song Min Kim, and Tian He. Networking support for bidirectional cross-technology communication. *IEEE Transactions on Mobile Computing*, 20(1):204–216, 2019.

[57] Chao-Kai Wen, Wan-Ting Shih, and Shi Jin. Deep learning for massive mimo csi feedback. *IEEE Wireless Communications Letters*, 7(5):748–751, 2018.

[58] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. Power of deep learning for channel estimation and signal detection in ofdm systems. *IEEE Wireless Communications Letters*, 7(1):114–117, 2017.

[59] Chaorui Zhang, Peng Xie, Deyuan Li, Jiekai Zhang, and Rong Yu. Wireless home gateway: Software-defined radio architecture and applications. In *IET International Conference on Communication Technology and Application (ICCTA 2011)*. IET, 2011.

[60] Zhongyuan Zhao, Mehmet Can Vuran, Fujuan Guo, and Stephen D Scott. Deep-waveform: A learned ofdm receiver based on deep complex-valued convolutional networks. *IEEE Journal on Selected Areas in Communications*, 39(8):2407–2420, 2021.

[61] Banghua Zhu, Jintao Wang, Longzhuang He, and Jian Song. Joint transceiver optimization for wireless communication phy using neural network. *IEEE Journal on Selected Areas in Communications*, 37(6):1364–1373, 2019.