RICE UNIVERSITY

On the Design of Reconfigurable Edge Devices for RF Fingerprint Identification (RED-RFFI) for IoT Systems

By

Thomas Keller

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE

Joseph Cavallaro

Joseph Cavallaro (Chair)
Professor of Electrical and Computer
Engineering
and Computer Science

Peter Varman

Peter Varman
Professor of Electrical and Computer
Engineering
and Computer Science

Edward Knightly

Edward Knightly
Sheafor-Lindsay Professor of Electrical and
Computer Engineering
and Computer Science

HOUSTON, TEXAS

August 2023

RICE UNIVERSITY

On the Design of Reconfigurable Edge Devices for RF Fingerprint Identification (RED-RFFI) for IoT Systems

by

Thomas Keller

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Joseph R. Cavallaro, Chair
Professor of Electrical and Computer Engineering and Computer Science

Peter Varman
Professor of Electrical and Computer Engineering and Computer Science

Edward Knightly
Sheafor-Lindsay Professor of Electrical and Computer Engineering and Computer Science

Houston, Texas

August, 2023

ABSTRACT

On the Design of Reconfigurable Edge Devices for RF Fingerprint Identification (RED-RFFI) for IoT Systems

by

Thomas Keller

Radio Frequency Fingerprint Identification (RFFI) classifies wireless transmitters by the signal distortions from their unique hardware impairments. RFFI capable receivers can authenticate insecure transmissions without the sender's cooperation, making them well suited for notoriously vulnerable IoT devices. Neural networks have dominated recent RFFI implementations but are prohibitively inflexible for practical use, requiring bespoke models for different transmission schemes and complete retraining for any change in authenticated devices. This along with the high computational and energy requirements for neural network training makes RFFI unfeasible for edge deployment: a primary use case of IoT.

To remedy this, we propose the Reconfigurable Edge Device for Radio Frequency Fingerprint Identification (RED-RFFI), a novel FPGA inference framework for RFFI using a programmable Deep-Learning Processing Unit (DPU) to analyze variable length signals for a mutable list of authenticated devices. This approach is uniquely capable of operating on the edge without relying on a high-performance computer for iterative FPGA redesign. Using the Xilinx Vitis AI inference development platform, we implement a state-of-the-art Transformer-based model analyzing LoRa signals as a test case.

Acknowledgments

I sincerely thank Dr. Joseph Cavallaro for his guidance, support, and insight throughout my thesis work. I would also like to thank Dr. Edward Knightly and Dr. Peter Varman for serving on my thesis committee, as well as for their thoughtful comments and suggestions. A huge thank you to Dr. Junqing Zhang and Dr. Guanxiong Shen for their inspiring work and continuous support. Thanks to the many undergraduate students who assisted in my work throughout my thesis in big and small ways: Ithzel Toscano, Emily Namm, Justin So, Jacob Flynn, Matthew Karazincir, and Ben Wilson. I thank Duke Trammell for his technical and emotional support throughout my graduate work. Finally yet foremost, I thank my family for their unyielding support, motivation, and love.

Contents

Illustrations

Tables

Chapter 1

Introduction

1.1 Motivation

Internet of things (IoT) devices are widely criticized for their lack of network security, which can create vulnerabilities in otherwise well-secured wireless networks. A study found 70% of IoT devices have security vulnerabilities, with each device having 25 unique flaws on average [2]. These flaws have not gone unnoticed by malicious actors; cyberattacks targeting IoT devices increased by 300% in 2019 alone [3]. This is a well established issue in the field, but the low-power and low-cost design requirements of IoT devices leave very few resources for security features, keeping it an endemic problem [4]. The magnitude of this security concern is amplified significantly by the continuous proliferation of IoT; there were 11.28 billion actively connected IoT devices in 2021, projected to more than double to 29.42 billion by 2030 [5].

The body of research exploring innovative security techniques for low-resource devices is significant [6, 7, 8]. However, existing IoT devices face a variety of unique technical hurdles preventing them from receiving security updates, and attempts to address this discrepancy through standardization have had a limited impact [9, 10]. Therefore, any new security solutions that require cooperation from the IoT device or manufacturer for deployment can only have a limited impact on the billions of existing IoT nodes. An ideal solution to this dilemma would be a "passive authentication" method, wherein insecure transmissions from an IoT device can be validated unilaterally by the receiver.

Radio Frequency Fingerprint Identification (RFFI) is an emerging non-cryptographic wireless authentication technology that is well suited to the restrictive requirements of IoT security. RFFI analyzes normal operation transmissions from network devices and identifies their source based on the hardware impairments of the transmitter. All devices, even theoretically identical instances of the same design, have manufacturing variances that create unique features in their signals that can be extracted, analyzed, and classified. These features are intrinsic to the hardware of the individual device and are therefore difficult to spoof.

RFFI was proposed as early as 1996 [11], but has taken time to reach maturity and faces further challenges before it could be feasible for wide-spread use. Traditional implementations of automatic modulation classification (AMC) require intensive feature engineering tailored for the specific use case. These are able to reach high accuracy in a laboratory setting, but fail to generalize well. As with many classification problems, Deep Neural Networks (DNNs) have begun to dominate RFFI feature extraction and classification in recent years, favored for their tolerance of noisy signals, adaptability to environmental changes, and relative ease of implementation [12].

However, machine learning is very computationally and power intensive. Running a state-of-the-art RFFI neural network at speeds to match IoT transmission standards requires powerful hardware with high parallelism, such as graphics processing units (GPUs) which consume hundreds of watts of power [13]. This is orders of magnitude more than edge IoT devices, putting RFFI neural networks at direct odds with the resource constrained environments driving the need for passive authentication. Field-Programmable Gate Arrays (FPGAs) are a viable alternative to meet this use case due to their programmable logic (PL), an array of configurable logic blocks.

Although each gate in the PL is significantly slower and lower power than in a GPU, their programmable nature allows the fundamental architecture of the FPGA to be customized for the specific operations used in the target neural neural network. This improves throughput significantly, matching or exceeding a GPU's performance at a fraction of the power.

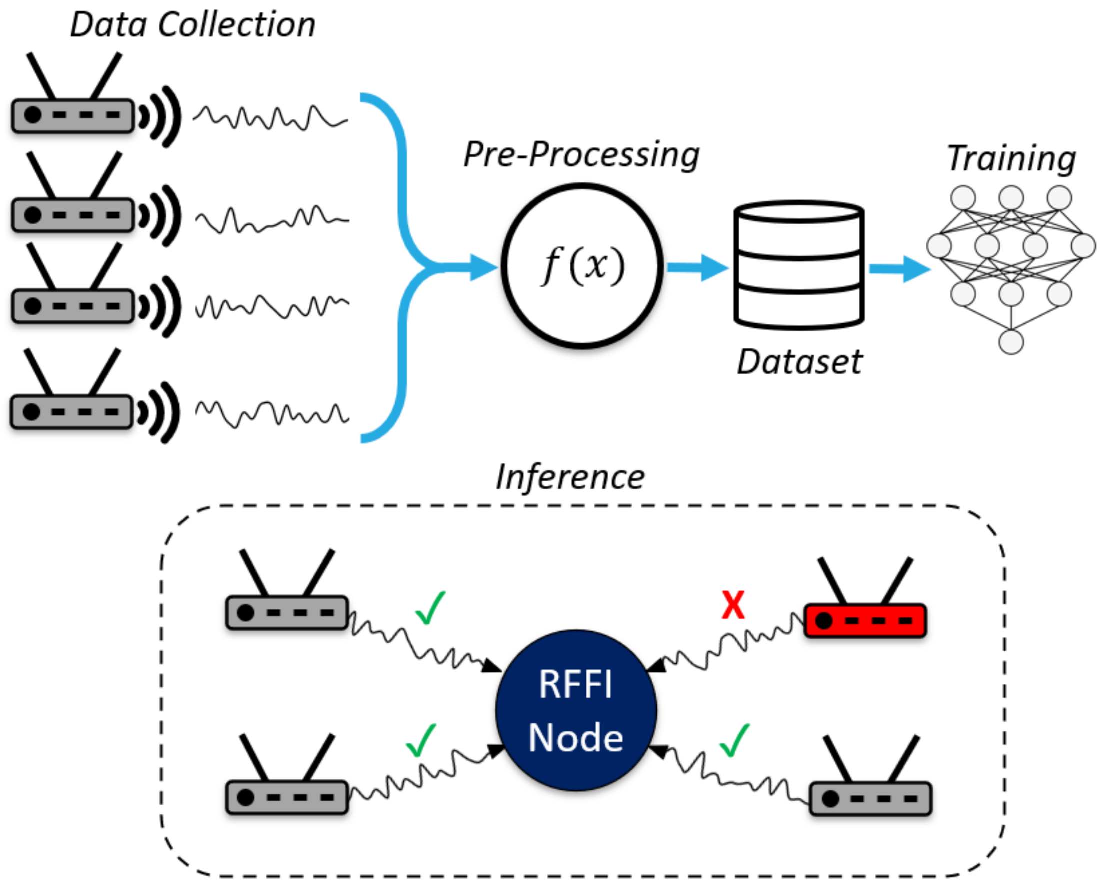

Figure 1.1 : Neural network workflow for RFFI

Custom FPGA accelerators are therefore an ideal solution to perform neural network inference on the edge alongside IoT devices, and in recent years there have been numerous papers designing bespoke FPGA neural network accelerators for RF fingerprinting [14, 15, 16, 17, 18]. These designs achieve state-of-the-art accuracy and power efficiency but are not versatile enough for real-world applications of RF fingerprint identification. RFFI networks are trained on samples collected from the unique devices one wishes to identify, and FPGA accelerators are then designed to fit that exact model. Any change to the neural network architecture necessitates that the accelerator be modified and reimplemented into the system. This process requires expertise in hardware design, Verilog/VHDL hardware level programming and verification, and an understanding of the underlying model.

Enrolling a new device is one of the most salient needs that would require retraining. The diversity of IoT devices and variety of network configurations across the consumer, commercial, and industrial sectors is vast [19]. Furthermore, IoT devices require relatively frequent replacement due to intentionally short lifespans and cheap, easily breakable materials [20]. These two considerations make model adaptation a necessity for long-term use of RFFI neural networks, but current accelerators are not designed to facilitate it. Many accelerators are intolerant of other real-world considerations as well, such as varying data rates or communication protocols.

The appeal of RFFI is as a passive authentication framework: a lightweight, unobtrusive layer of security and validation for an insecure network. Therefore, despite the best in class performance of some embedded DNN accelerators for RFFI, they remain impractical without the ability to be easily configured for the shifting requirements of IoT.

1.2 Contributions

We present the Reconfigurable Edge Device for Radio Frequency Fingerprint Identification (RED-RFFI), an FPGA neural network acceleration framework that is novel for RFFI and uniquely adaptable to the variability of IoT devices and networks. The core of RED-RFFI is a Deep-Learning Processor Unit (DPU): a programmable engine optimized for neural network operations. Implementing the DPU into the PL of the FPGA allows us to redefine layers at runtime with parameterizable dimensionality and weights. The DPU operations are scheduled along with static compute units created in Vitis HLS, a tool for synthesizing C++ code into RTL designs. This hybrid approach enables RED-RFFI to utilize the efficiency of highly optimized PL with some of the flexibility of a more traditional processor-based system.

These layers are configured and trained in accessible frameworks such as PyTorch and TensorFlow, empowering users to add authenticated devices using transfer learning or change communication protocol without hardware design expertise. The resulting model is then quantized, optimized, and compiled for the DPU using Vitis AI. We implement a Transformer-based model analyzing LoRa devices as a test case, achieving $> 99.8\%$ accuracy at 24dB for spreading factors 7-9. Quantized to 8-bit integer, RED-RFFI achieves $> 99.4\%$ accuracy at 24dB under the same test conditions with $< 1\mu s$ of latency per timeseries sample, enabaling real-time LoRa RF fingeprinting. To the author's knowledge, this is the first RF fingerprinting neural network accelerator that can enroll new devices without reconfiguring the programmable logic of the FPGA and the first Transformer-based RFFI accelerator.

This work has been selected for presentation at the 57th Asilomar Conference on Signals, Systems, and Computers in October 2023.

1.3 Thesis Overview

Chapter 2 details the background literature that RED-RFFI builds upon and explores related works to establish the capabilities and shortcomings of state-of-the-art RFFI accelerators. Chapter 3 establishes our design goals and examines the theory behind the techniques and algorithms we employ. Chapter 4 describes the means, methodology, and experimental results from the iterative design process that lead to the final design for RED-RFFI. Chapter 5 examines the final accelerator and characterizes its capabilities with regard to our design goals and limitations. Chapter 6 concludes the thesis and proposes future work to expand and refine RED-RFFI.

Chapter 2

Background and Related Work

Our work builds on the RFFI algorithms and models developed primarily through our ongoing collaboration with Professor Junqing Zhang and Guanxiong Shen at the University of Liverpool in [1, 21, 22, 23]. This chapter covers the key algorithms, techniques, and findings from their work that were utilized in the development of RED-RFFI. This section also explores the body of work for RFFI accelerators on FPGAs to define what key elements of adaptability are missing from the state-of-the-art. Finally, we explore the FPGA DNN acceleration design frameworks and transfer learning techniques used in our work.

2.1 Signal Preprocessing

Intelligent preprocessing is crucial for disentangling the transmitter's hardware fingerprint from the received signal, enhancing RED-RFFI's feature extraction and significantly improving classification accuracy. This is particularly important to combat poor performance for low SNR signals: a key constraint of RFFI as a technology.

2.1.1 LoRa and Basic Processing

We explore LoRa signals as a test case, however the techniques listed can be adapted to other protocols. LoRa (standing for Long Range) is a popular transmission protocol for IoT due to its low transmission power, long range, and resistance to attenuation [24]. Modulated with chirp spread spectrum (CSS), all LoRa signals begin with a preamble of repeated upchirps, given in baseband as:

$$x'(t) = Ae^{j(-\pi Bt + \pi \frac{B}{T})} \text{ for } (0 \leq t \leq T), \tag{2.1}$$

such that A and B denote signal amplitude and bandwidth, respectively. T denotes the duration of a LoRa symbol, given as:

$$T = \frac{2^{SF}}{B}, \tag{2.2}$$

such that SF denotes *spreading factor* [25]. With fixed bandwidths of 125 kHz, 250 kHz, and 500 kHz, the spreading factor controls data-length, ranging from SF 7 to SF 12. Lower spreading factors are used in more ideal conditions, driving a higher data rate. Higher spreading factors are utilized when transmitting over long distances or in a more noisy channel. Additionally, when LoRa signals are being controlled by the LoRaWAN network protocol, end-nodes can use an adaptive data rate to optimize spreading factor dynamically. A length-versatile RFFI model is required to accommodate this feature.

The number of upchirps in a preamble varies, but is commonly set to eight, as we did in our testing. With this information and Equation 2.2 we can calculate the maximum end-to-end latency of our accelerator required to achieve real-time RF fingerprinting for each SF: 2.048 ms for SF 7, 4.096 ms for SF 8, 8.192 ms for SF 9, and so on, doubling for each higher spreading factor. When divided by the amount of samples in each preamble, all spreading factors require a processing latency of under 1 μs per sample in order to be received and processed continuously, in real-time. This metric is crucial to evaluate the viability of RED-RFFI, since passive authentication must not interrupt an authenticated device's normal operation.

Upon receiving a signal, we correlated it with an ideal preamble to synchronize and align it to the desired data. We then performed two rounds of Carrier Frequency Offset (CFO) estimation and compensation as described in [22] to improve system stability with regards to temperature variation. Finally, we normalized the signal using the root mean square to improve stability with regards to distance.

2.1.2 Data Augmentation

During training, input data is augmented with additive white Gaussian noise (AWGN) to improve robustness against channel variations and low SNR scenarios. We performed a variation of this technique called "online augmentation" wherein a mini-batch of randomly selected samples are augmented at the same time and with the same parameters. This was shown to outperform standard, individual sample augmentation in [21], improving accuracy by as much as 50% over baseline at low SNR.

2.1.3 Channel Independent Spectrogram

Modulation schemes such as LoRa exhibit time-frequency characteristics, which could be more clearly articulated in a spectrogram than in time-series. Additionally, one of the most troublesome sources of signal disruption is the wireless channel, as naive attempts to remove the channel effect can diminish the hardware fingerprint. We address both of these issues by converting received preambles into a channel independent spectrogram (CIS). This is achieved by first taking the discrete short-time Fourier transform (STFT) of the complex preamble signal s, given as:

$$S_{k,m} = \sum_{n=0}^{N-1} s[n]w[n - mR]e^{-j2\pi \frac{k}{N} n} \tag{2.3}$$

$$\text{for } k = 1, 2, ..., N \text{ and } m = 1, 2, ..., M,$$

such that N is the length of window function $w[n]$ for each timestep n, and R is the hop size. The resulting $S_{k,m}$ is an element of the complex matrix $S \in \mathbb{C}^{M,N}$. For a LoRa preamble with U upchirps, the number of columns M is given as:

$$M = \frac{U * \frac{2^{SF}}{B} * \frac{1}{T_S} - N}{R} + 1. \tag{2.4}$$

The parameters for the STFT should be tailored to the modulation scheme in use to best express its time-frequency characteristics. For the LoRa preamble, we match the window size N to the length of an individual upchirp, as determined by the spreading factor. This results such that each FFT captures exactly one upchirp, as seen in Figures 2.1 and 2.2. Although this conversion to the frequency domain clarifies the hardware features from modulation, it also accentuates phase noise. To correct this, we take the amplitude of the complex matrix S, moving forward with $|S|$. This moves our data from complex to real, which is also important to facilitate eventual quantization for the FPGA.

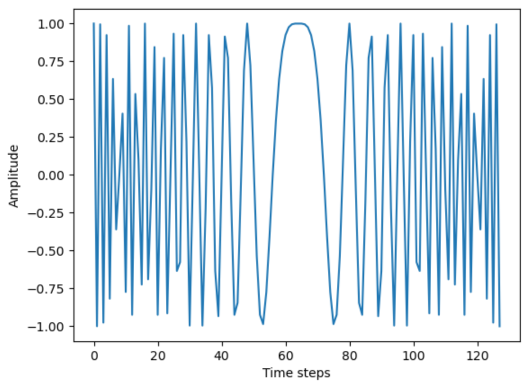

Figure 2.1 : Single LoRa upchirp

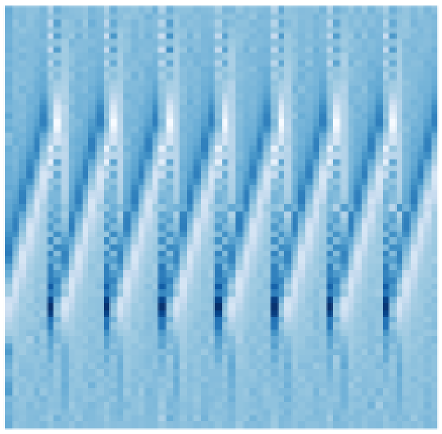

Figure 2.2 : CIS of LoRa preamble

In theory, provided that we sample with a small time gap (128 μs in paper [23]), it is plausible to assume that the channel does not change meaningfully between timesteps. Therefore, we can divide each column m by column $m-1$ to eliminate the channel effect while reducing the row size to $M-1$. The resulting data is given as:

$$Q = \left[\frac{|S_{:,2}|}{|S_{:,1}|} \; \frac{|S_{:,3}|}{|S_{:,2}|} \; \cdots \; \frac{|S_{:,M}|}{|S_{:,M-1}|} \right]. \tag{2.5}$$

Finally, the result in converted to dB scale:

$$\widetilde{Q} = 10 \log_{10}(|Q|^2). \tag{2.6}$$

This completes the conversion of the received data to a channel independent spectrogram $\widetilde{Q}$ and is ready to be fed into the neural network for fingerprinting.

2.2 Neural Network Selection

In order to maximize the performance and adaptability of RED-RFFI, it is important to thoughtfully select which model to accelerate. To this end, we considered four neural network architectures that were evaluated in with the same dataset and preprocessing for fair comparison [1]. The selected models are all length-versatile, as this is necessary for variable-length transmission protocols such as LoRa, Wi-Fi, and heterogeneous protocol networks. The models tested were a flatten-free convolutional neural network (CNN), long short-term memory (LSTM) network, gated recurrent unit (GRU) network, and Transformer. Table 2.1 shows the results of these tests evaluated for inference latency, classification accuracy at varying SNR conditions, and model size measured in trainable parameters. All four models show very similar accuracy and latency, with the Transformer taking a slight edge in latency. The largest discrepancy is in size; the Transformer far outclasses the others with almost half the parameters of the next smallest model. The size of the model is crucial for high throughput when deployed in a DPU based accelerator, as described in Section 2.5.2. Considering this and its latency advantage, the Transformer [26] was the clear choice as the baseline model for our accelerator.

Table 2.1 : Length-versatile model performance [1]

Neural Network	Average Inference Time (ms)	Average Accuracy			Parameters
		Low SNR (0dB)	Medium SNR (20dB)	High SNR (40dB)	
Flatten-Free CNN	33.53	20.41%	94.93%	99.99%	675,594
LSTM	32.63	24.05%	95.50%	99.99%	856,586
GRU	31.80	22.93%	94.90%	99.97%	644,618
Transformer	30.93	21.86%	95.11%	99.97%	348,938

Transformers have proven to be very effective for natural language processing (NLP) and image processing tasks [26, 27], making them well suited for spectrogram analysis. The most defining component of the Transformer is the multi-head attention (MHA) layer which projects the input spectrogram upon itself to quantify the relative relationship between different features in the signal. The resulting attention value is added to the original signal through a residual connection and normalized. This process is then repeated, but with a simple feed-forward layer instead of MHA.

The Transformer model we wish to accelerate consists of two encoder layers, a global average pooling layer to remove the time series dimension, and a softmax classifier as shown in Figure 2.3. Multi-head attention is the most computationally complex part of the model and therefore the most important consideration for acceleration. This is further explored in Section 3.3.

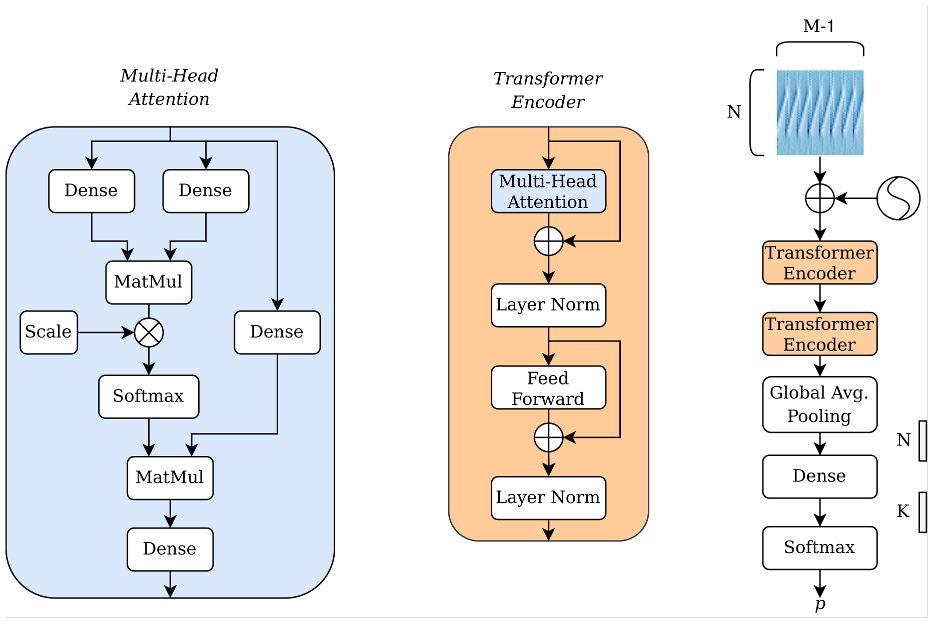

Figure 2.3 : Baseline Transformer RFFI architecture

2.3 Feasibility of Prior Work

We identified five works implementing RFFI neural network FPGA accelerators. We analyze them to establish what barriers exist to achieving viability for long-term deployment. Comparing RFFI accelerator performance is challenging, because state-of-the-art accuracy and acceptable maximum latency can vary significantly depending on communication protocol, bandwidth, and data rates. Therefore, we do not thoroughly analyze their quantitative performance in this work, but rather their qualitative feasibility for prolonged use.

All evaluated designs classify a fixed number of devices within a closed set on a

Table 2.2 : Existing RFFI DNN accelerators.

Model	Protocol	Energy	Latency	FPGA Platform
[14] ResNet50-1D	802.11b Wi-Fi	14W peak	1.36 ms	ZCU104
[15] Spiking NN	LEO-MIMO	48.23 mJ/image	27.14 ms	Virtex 7
[16] Photonic-RNN	ZigBee	23.38 mJ/image	0.219 ms	PYNQ-Z1
[17] Bayesian NN	Unmodulated	0.548 μJ/image	0.576 ms	ZCU102
[18] VGG-16	802.11a Wi-Fi	20W peak	5.4 sec	PYNQ-Z2

Xilinx FPGA System on Chip (SoC). Enrolling additional devices requires retraining the network with additional data from the new transmitter. The accelerators are designed for inference only, so training must be done externally. This could theoretically be performed on the CPU of the SoC, but is inefficient due to low clock speeds and limited DRAM. Furthermore, for any changes to the model's dimensionality or weights, the hardware design must be updated for the new model and then synthesized, implemented, and deployed to the FPGA from an external device. Accelerators [16] and [17] were designed with Vitis HLS and [18] with FINN (explored in Section 2.5.1) respectively. These tools allow for more quick iteration of the design without manually writing RTL, but still require re-implementation of the hardware and knowledge of hardware design. Training samples could be sent to a remote datacenter for inference as proposed in [23], which could also be used to generate the new board image for the FPGA and deployed to the accelerator remotely. The viability of this is explored in Section 3.1.

Many transmission protocols send signals of variable length depending on shifting network conditions, such as LoRa and Wi-Fi. Simple DNN models like [17] can only process inputs of a fixed-length, and are thus only suited for laboratory experiments. CNN accelerators [14] and [18] are fixed input as well, but use a sliding window to cut inputs into fixed length slices. This slicing technique accommodates longer signals, but decreases performance [1]. Accelerators [15] and [16] are fully length-versatile.

2.4 Transfer Learning

Transfer learning (TL) is the technique of leveraging a model trained on one domain to more rapidly train a new model on a related domain with a smaller dataset. There are a variety of novel approaches to transfer learning [28, 29, 30], but all generally revolve around some combination of freezing weights from the baseline model or using the baseline weights to directly influence the training of new layers.

Transfer learning has shown to be effective at accelerating the training of RFFI models to adapt to new environments and potentially even new devices [31, 32]. Two methods that were found to be effective for this are the *TL Shallow-layer-frozen* (TL-SLF) method and the *TL Convolutional-layer-frozen* (TL-CLF) method, both developed for a ConvMixer network [31].

TL-SLF freezes some amount of the most shallow layers and trains the remaining layers from the baseline checkpoint. This can be directly translated to our Transformer network by freezing the weights of the first encoder. In TL-CLF, all convolutional layers are frozen, while point-wise layers are fine-tuned from the baseline checkpoint. This has synergy with the LightConv technique described in Section 3.3.1. We sought to implement these techniques in our accelerator system to expedite device enrollment as described in Section 3.4.1.

2.5 FPGA DNN Accelerator Automation Frameworks

The programmable logic of FPGAs allows for hand-crafted DNN accelerator designs to achieve desirable throughput and energy efficiency. This has driven the creation of several development frameworks to expedite and automate this process. These tools can be split into two categories: dataflow-style and overlay-style [33]. We explored a framework of each kind (*FINN* and *Vitis-AI*) to determine which workflow would best enable adaptable deployment for RFFI on the edge. Both frameworks begin with either quantization-aware training (QAT) or post-training quantization (PTQ), because FPGAs compute significantly more efficiently with fixed-point operations instead of floating-point operations. This section describes the key differences between each toolchain, while Section 3.2 analyzes their suitability for RED-RFFI.

2.5.1 FINN

Dataflow-style accelerators create a pipeline of fixed-function computation units with model weights held in the on-chip buffers of the FPGA. This is the approach of all RFFI accelerators listed in Section 2.3. FINN is an open-source, experimental framework from Xilinx Research Labs for automated design space exploration and implementation of DNN accelerators on FPGAs [34, 35]. It utilizes three primary components: Brevitas, the FINN Compiler, and PYNQ. This produces an accelerator design as depicted in Figure 2.4.

Brevitas is a Python library for the QAT of neural networks, with support for PTQ. FINN focuses on aggressive quantization such as 3-bit weights and binary neural networks (BNNs), but Brevitas is highly configurable with 8-bit integer and mixed weight quantization, as well as support for custom quantization schemes. The FINN Compiler then applies limited dataflow optimizations, streamlining layers and folding weights into the fast, on-chip block RAM (BRAM) as possible before compiling the design into register-transfer level (RTL). The optimized model is then implemented to the FPGA using *PYNQ*, a framework that deploys embedded systems to the Programmable Logic in an "overlay" that is controlled using Python and Jupyter Notebooks on the Xilinx ZYNQ SoC ARM core processor.

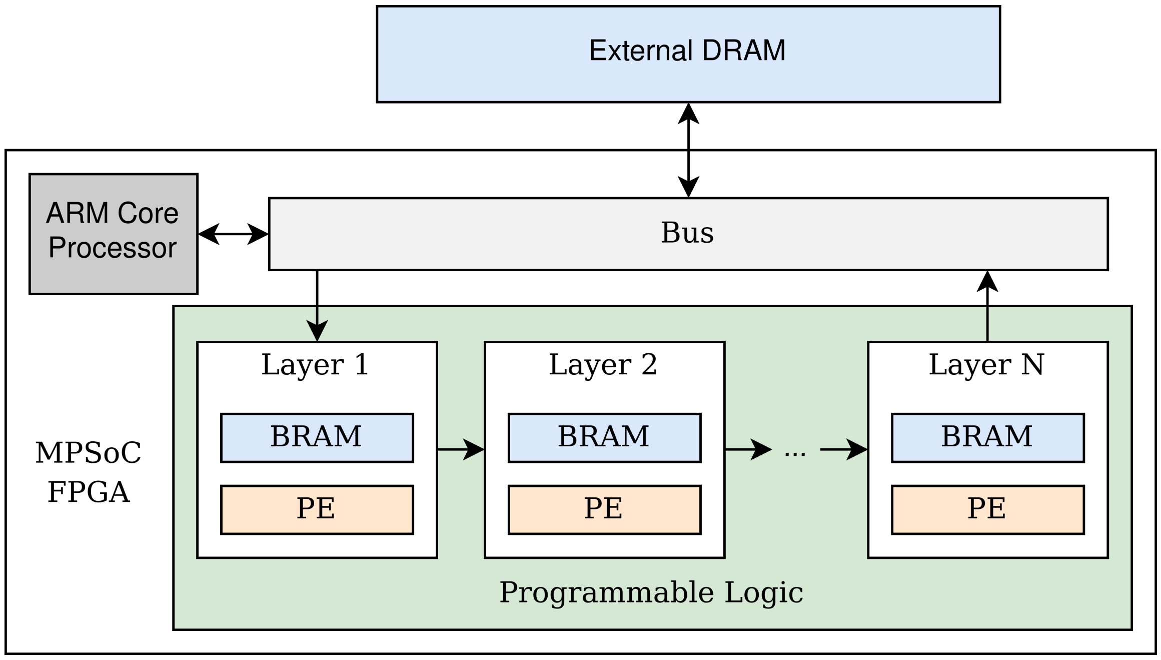

Figure 2.4 : Dataflow-style accelerator (FINN)

Being an experimental framework, only a handful of layers and activations from PyTorch are implemented by default. Additional functions and layers must be implemented in Brevitas for quantization and in the FINN Compiler to be properly handled and translated to RTL.

2.5.2 Vitis AI

The overlay-style approach centers on the Deep-Learning Processing unit: a heterogeneous architecture forming a CPU+FPGA configurable computation engine. The DPU is programmed and controlled using the Vitis AI software framework.

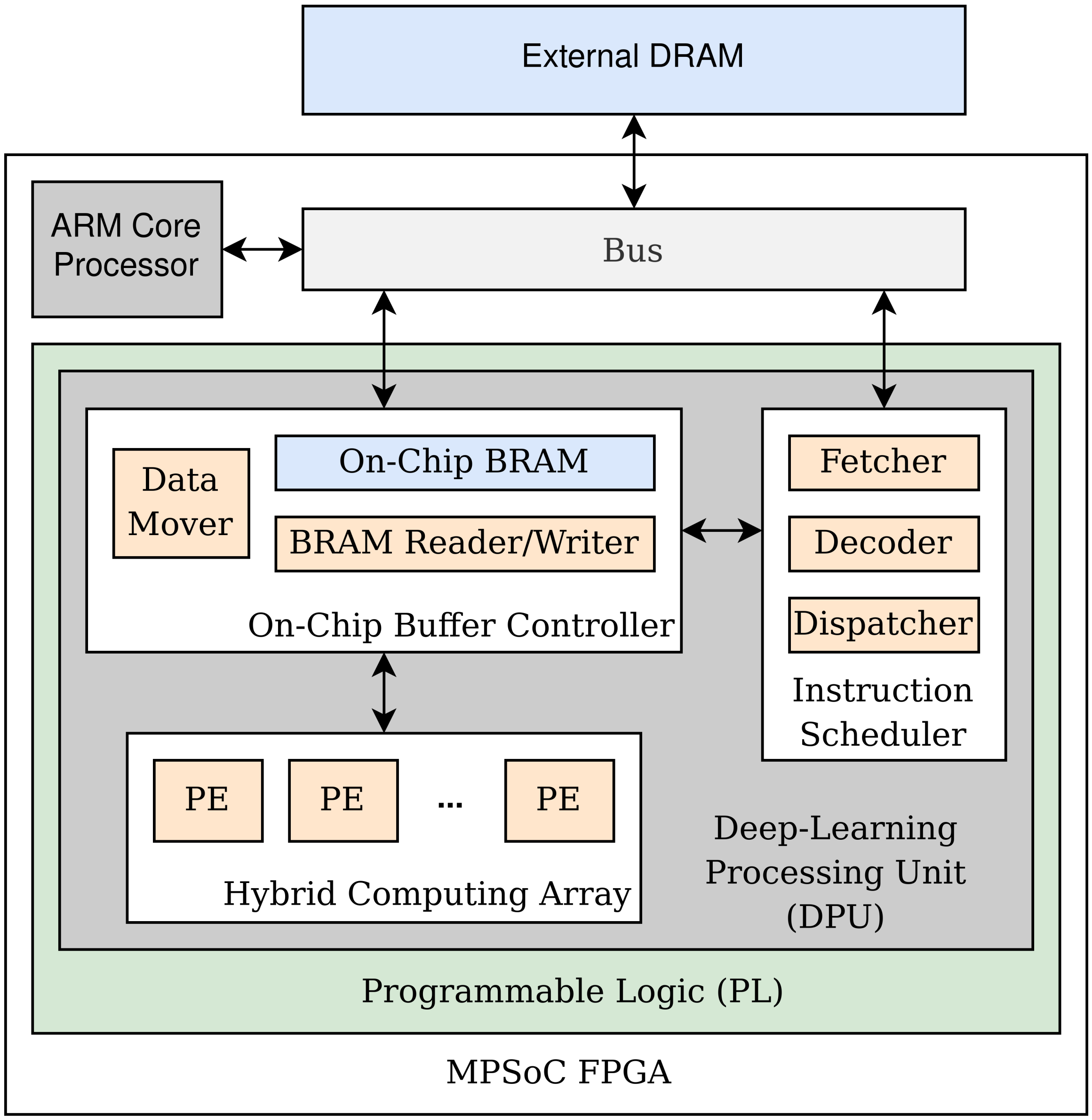

Figure 2.5 : Overlay-style accelerator (Vitis AI)

Deep-Learning Processing Unit

The design ethos of the DPU is in between a dataflow-style FPGA accelerator and a GPU. The processing elements in the hybrid computing array are highly optimized for tasks common to machine learning, similar to dataflow-style accelerators. However, the datapath is dynamically controlled by an instruction scheduler, which is more similar to a CPU or GPU.

The DPU consists of three primary components as depicted in Figure 2.5: the on-chip buffer controller, instruction scheduler, and hybrid computing array. For a given neural network layer, instructions and weights are loaded into DRAM along with the current hidden state of the model. The weights and input data are then loaded to on-chip BRAM for computation. Instructions are sent to the instruction scheduler, which selects a processing element. The hybrid computing array contains a diverse array of processing elements to handle different types of neural networks layers. However, this DPU was designed specifically for CNNs, so the convolution engine takes up a significant portion of the DPU's resources. The full list of possible computations is listed in the user guide [36]. The programmable logic for a convolutional layer is depicted in Figure 2.6 as described in the patent filed for the generic DPU design [37]. Multiple instances of this can be deployed in parallel to compute multiple kernels concurrently.

The output data from the processing element is then streamed back to BRAM as the new hidden state of the model. The hidden state is then either sent into another PE for the next layer, or returned to DRAM for use by the CPU. The dataflow for this process is depicted in Figure 2.7. There are multiple DPU designs optimized for different FPGA platforms. The Zynq UltraScale+ MPSoC FPGA uses the DPUCZDX8G [36] IP block, which is the focus of our work. Up to four DPU cores

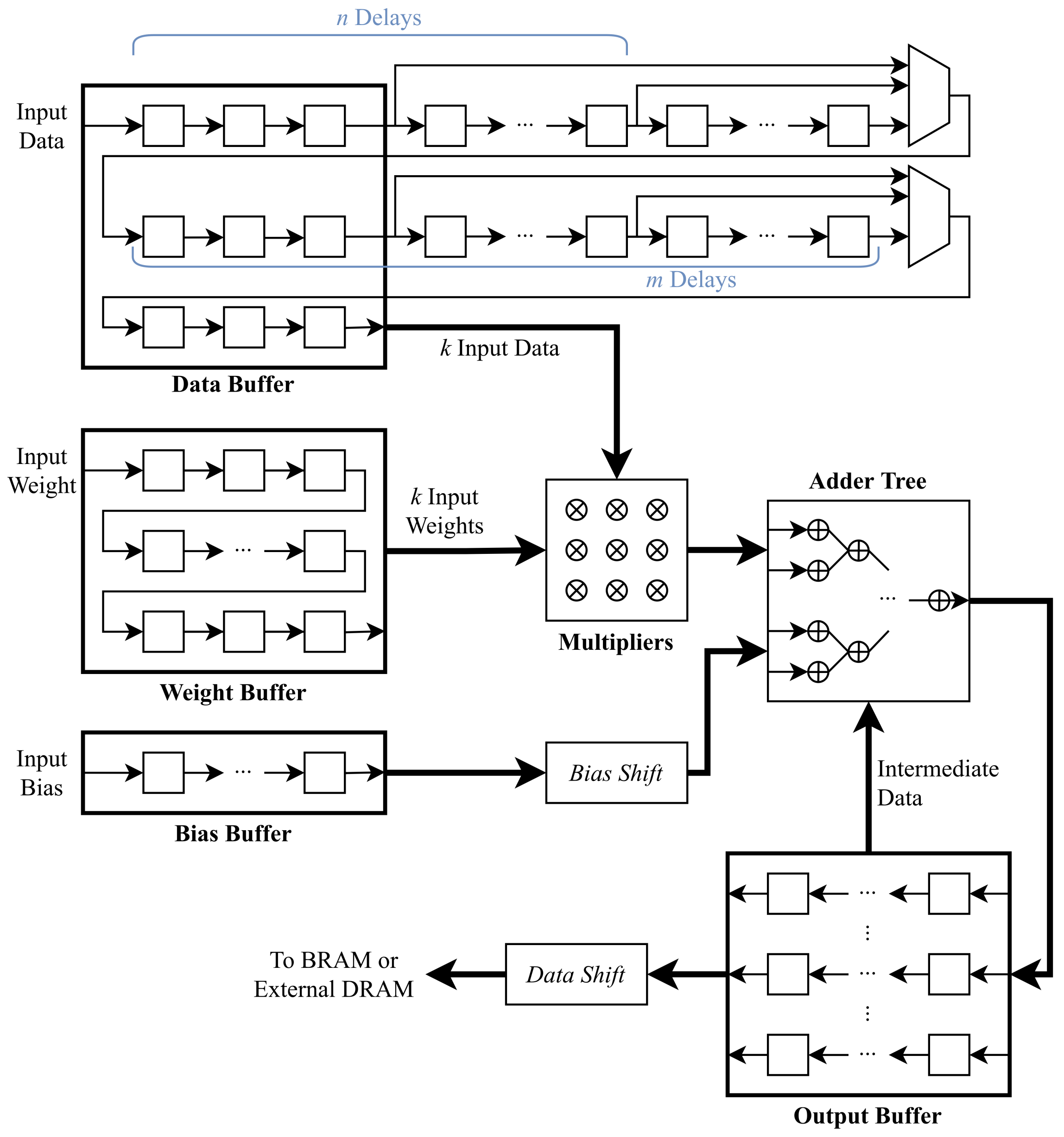

Figure 2.6 : Block diagram for convolutional layer programmable logic

can be implemented at once for additional parallel compute throughput. The size and power draw of each DPU can be configured in the initial deployment in Verilog. Larger versions of the DPU enable higher levels of pixel and input channel parallelism for the convolution engine as depicted in Figure 2.6, increasing peak operations per cycle. Some, but not all of the other processing elements scale up in size as well.

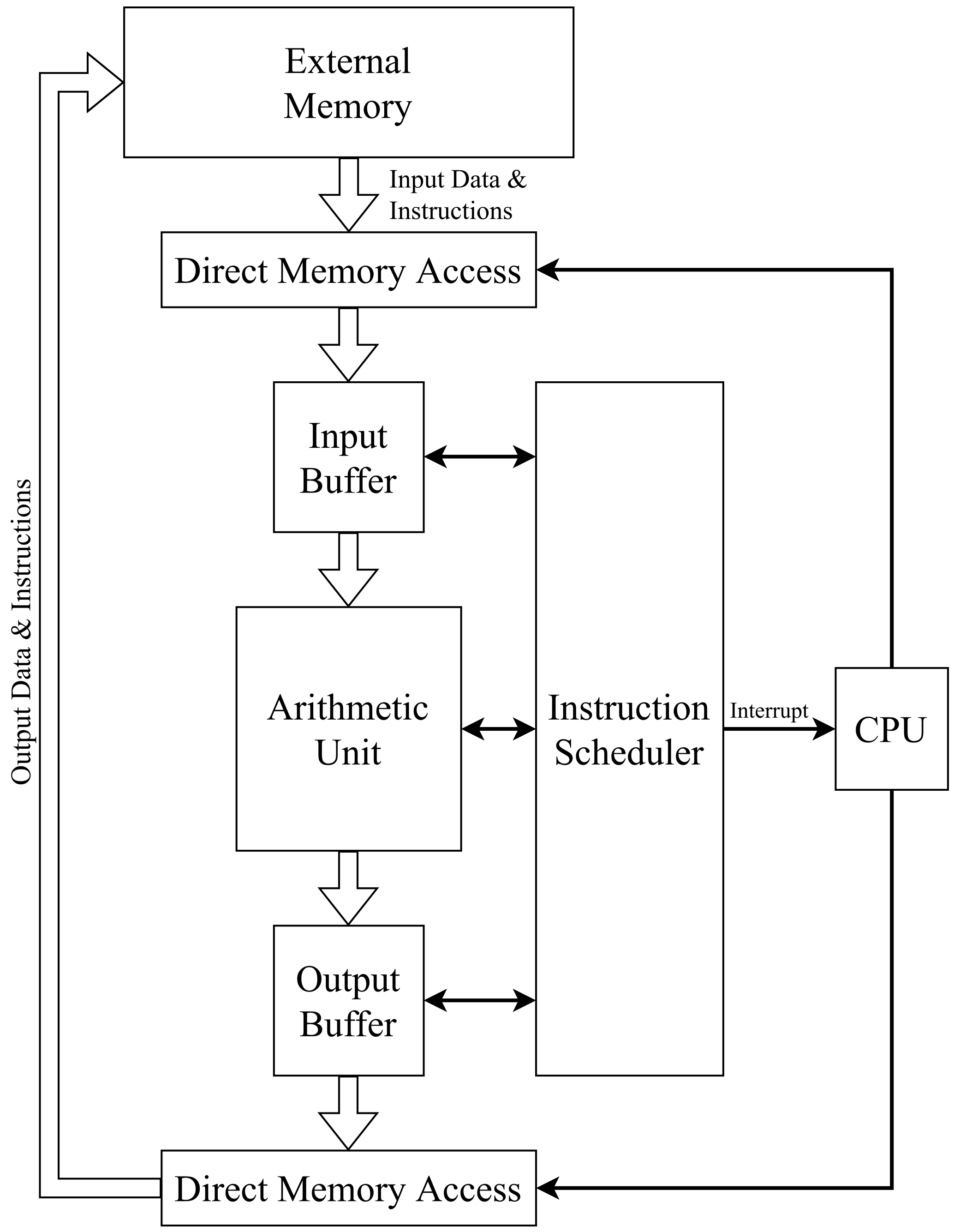

Figure 2.7 : Dataflow of DPU operations

Software Workflow

The Vitis AI workflow converts neural networks defined in PyTorch or TensorFlow into instructions for the DPU in the form of an *xmodel* file. The Vitis AI workflow consists of a quantizer, optimizer, compiler, and runtime. The AI Quantizer performs PTQ and has limited support for QAT, calibrating the dynamic range to the input dataset. The array of quantizable layers in Vitis AI is much wider than in FINN, although there are limits to maximum data size based on the initial configuration of the processing elements. The AI Optimizer can then perform optional iterative pruning to reduce model size and computational complexity while fine-tuning for minimal impact on accuracy. The AI Compiler then converts the quantized model to an *xmodel* file which contains instructions for the DPU, model weights, and biases. This file is saved on the FPGA and loaded at runtime to define the model to run.

The Vitis AI Runtime (VART) is built on the Xilinx Runtime (XRT) and allows for easy control of the DPU through C++ or Python. The desired xmodel file is instantiated as a DPU runner thread for all DPU operations. Operations that aren't supported by the DPU are compiled for the CPU. However, this requires moving data off the DPU and to the CPU, and back again. These transactions add significant latency compared to a model that is kept entirely in the programmable logic.

As an alternative to CPU implementations for undefined layers, it is possible to create an HLS kernel and register it with the DPU for integration and use. This must be done during the initial deployment in Verilog and is not reconfigurable at runtime like the DPU, making HLS plugins prohibitively inflexible for operations with variable dimensions.

Chapter 3

Model and System Design

Accuracy, latency, adaptability, and security are the core design goals of an IoT RFFI accelerator as established in Chapter 2. In this chapter, we define the design constraints, acceleration techniques, and automation platforms that RED-RFFI adopts in pursuit of these goals.

3.1 Adaptability and Security

While the goals of accuracy and latency can be directly numerically compared, adaptability and security lack clearly defined metrics. We characterize these objectives as *maximizing the system's compatibility with IoT while minimizing its attack surface.* Considering the established capabilities of existing accelerators and the technical requirements for IoT RFFI, we propose the following two key design constraints in pursuit of these goals.

1. Do not reconfigure the programmable logic of the FPGA

Requiring updates to the core configuration of the FPGA fabric is an unrealistic burden for a network administrator. Area knowledge of hardware engineering is very uncommon, even in technology focused industries. According to the U.S. Bureau of Labor Statistics, there were 74,640 Hardware Engineers in 2022, compared to the over 1.5 million Software Engineers [38]. This is only a very rough indicator of a potential user's area of expertise, but is still demonstrative of the relevant disparity.

Software-engineer focused tool flows like Vitis HLS and PYNQ make alterations of the programmable logic more accessible to a potential user, but still require domain knowledge that makes enrollment prohibitively onerous.

Theoretically, this update process could be automated into a user interface suitable for the general public. However, adjustments and deployment of hardware designs must be performed on an external system that is capable of handling the computational complexity of synthesis and implementation. This is at odds with the resource constraints of edge deployment. An active Xilinx license is also required to generate FPGA designs through Vivado, further complicating the process.

2. Perform device enrollment and model updates locally

A potential solution to the burden posed by hardware redesign and deployment is sending training data and platform information to a remote server or high-powered compute unit freed from the resource constraints of edge computing. The server could then retrain the neural network and implement a new hardware design, remotely deploying the design and revised model weights to the accelerator.

This approach limits viable use cases, as it assumes an internet connection suitable for large file transfers. More importantly, transmitting this data over the network adds multiple attack vectors that a malicious actor could use to obtain the neural network or sufficient training data in order to reproduce it. Research on spoofing RFFI is limited by the difficulty of imitating the subtle hardware features of an authorized transmitter from the RFFI node's perspective. Existing methods take complex approaches, such as treating spoofing as an intractable non-convex optimization problem [39]. Direct access to the DNN used for classification gives a method of rapidly validating spoofing attempts, empowering attackers.

Therefore, in order to operate securely, any model updates due to device enrollment, transmission protocol change, environmental changes, or any other reason should be handled locally on the accelerator.

3.2 Framework Selection

The throughput of FINN generated accelerators significantly outpaces that of Vitis AI generated accelerators for the same model. A recent study implementing ResNet-8 on both frameworks found that FINN outperformed Vitis AI by 8.4x in latency, 3.0x in throughput, and 3.3x in power efficiency [33]. This performance disparity is largely due to a DRAM bottleneck in Vitis AI. DRAM is relied upon to transfer DPU instructions, inference data, weights and biases, intermediate feature maps, and output meta-data. Should these needs exceed DRAM bandwidth, system performance becomes bottlenecked, limiting parallelism. For very large models or more resource constrained FPGAs with less programmable logic, Vitis AI regains an edge thanks to its reuse of PEs enabling more operations than could be fit sequentially in the PL.

Examining adaptability, both frameworks can implement length-versatile models. FINN makes it easier to update the model on the accelerator when compared to traditional FPGA toolflows, but it still requires generating and deploying a new design to the programmable logic for any model changes. This necessitates an additional high-performance computer for synthesis and implementation, with the ramifications described in Section 3.1. Conversely, when given a new model, Vitis AI only needs to compile a new instruction set for the DPU which can be done on a low-power edge processor or potentially even the ARM core of the SoC. These instructions are then loaded into the BRAM at runtime for inference.

FINN's throughput advantage over Vitis AI is substantial, but IoT standards such as LoRa do not require exorbitant throughput for RFFI; it just needs to be fast enough to match the data rate. Therefore, provided our design can reach sufficient throughput for real-time LoRa RF fingerprinting, Vitis AI is the preferred workflow for RED-RFFI as it is uniquely capable of meeting our design goal for adaptability.

3.3 Model Architecture

The Transformer is a relatively new architecture compared to CNNs and are commonly utilized by larger models with more than 100 million parameters. For these reasons, some of the key components of multi-head attention are not expressly supported by Vitis AI for edge device SoCs. We explore those operations and how best to implement them or equivalent alternatives in a sufficiently flexible manner in this section. The performance of these approaches is tested and documented in Section 4.2.

3.3.1 Matrix Multiply and Lightweight Convolutions

Designed originally to accelerate CNNs, the majority of processing elements in the DPU are Convolution Engines. Matrix multiplies are converted to two-dimensional convolutions and mapped to these engines. Operations that convolve an input with a static, learned weight (such as Dense and Conv2D) function as expected, but convolution engine operations with two dynamic inputs fail to deploy to the DPU for MPSoC devices as of Vitis AI version 3.5. Although these operations can be quantized properly, enabling simulation, this prevents multi-head attention from operating on the DPU, as it requires two matrix multiplies with dynamic inputs as seen in Figure 2.3.

Two alternatives for self-attention in the Transformer are Lightweight and Dynamic Convolution [40]. *LightConv* is a form of Depthwise Convolution, wherein convolution is performed independently for each channel. For kernel size k and output dimension c, DepthwiseConv with weight $W \in \mathbb{R}^{c\times\ k}$ is defined as the following for each channel c, element i:

$$\text{DepthwiseConv}(X, W_{c,:}, i, c) = \sum_{j=1}^{k} W_{c,j} \dot{X}_{(i+j-\lceil \frac{k+1}{2} \rceil),c} \tag{3.1}$$

LightConv departs from this by softmax-normalizing the weights. This is an operation performed on the weights directly rather than the output like in self-attention. Therefore, softmax only needs to be done during training, which is further discussed in Section 3.3.3. For multi-head LightConv with d heads, the input of size $[T, E]$ is reshaped into $[d, T, h]$ such that $d = \frac{E}{h}$. Depthwise Convolution with kernel size k therefore has complexity $(E\times k)$. Optionally, by sharing weights between heads, this can be further reduced to $(h\times k)$.

The primary limitation of this approach is that LightConv lacks the theoretically unlimited context window of self-attention, reusing weights across time-steps. Infinite context is not necessary for good performance however, and preceding the LightConv with a Gated Linear Unit (GLU) allows it to represent sufficient context for parity performance with traditional Transformers. This structure utilizing the GLU was proposed by [40] but unnamed, so we refer to this as Multi-Head LightConv (MHL), illustrated in Figure 3.1.

MHL removes all two-input matrix multiplies from the Transformer encoder, circumventing the MatMul limitation of Vitis AI. *DynamicConv* expands on LightConv by dynamically updating the weights of the DepthwiseConv layer using a Linear layer for a small accuracy increase. However, this is a two-input convolution which experiences the same issues as two-input matrix multiply, and therefore is not desirable for the Vitis AI workflow at this time.

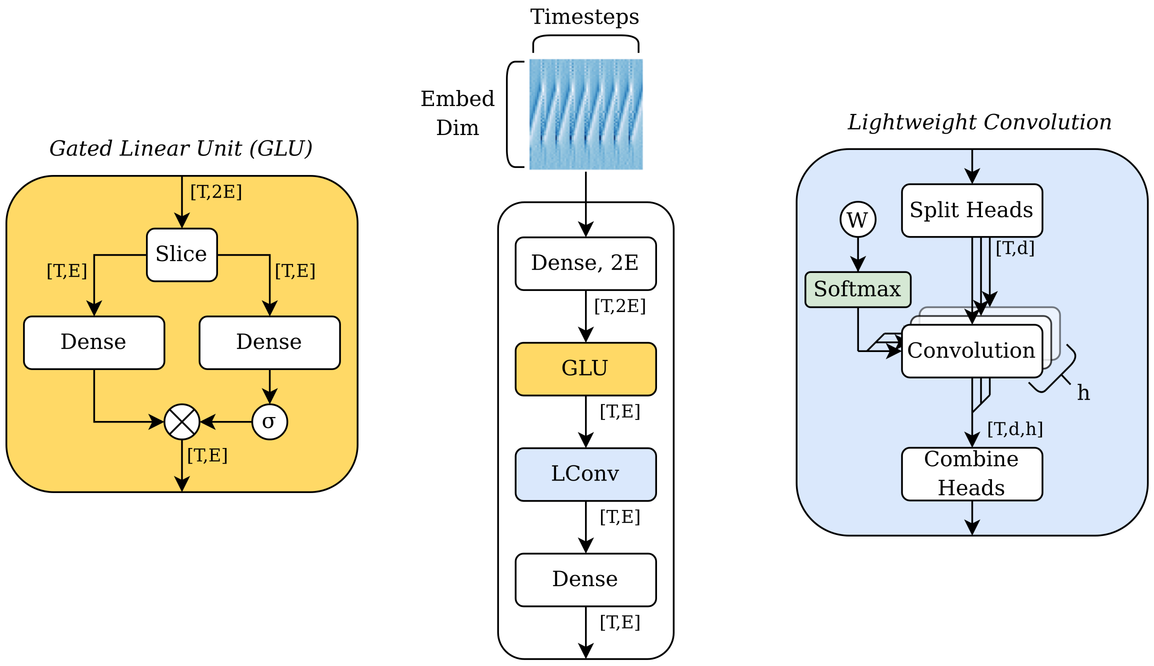

Figure 3.1 : Multi-Head LightConv for input spectrograms of size $[T, E]$

3.3.2 Positional Encoding

Positional encoding is the first step of the Transformer architecture. Often analyzing tokenized information, positional encoding injects information representing the relative position of each token (in the natural language processing use case, this is the order of words in a sentence). Transformers uses a static sinusoidal vector as opposed to a learned weight, as it is shown to give near identical performance [26]. We propose that the channel independent spectrogram preprocessing adds similar positional context, and thus this layer can be eliminated to reduce parameters and latency.

3.3.3 Softmax

The softmax operation converts an array of scalars into probabilities, given as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}. \tag{3.2}$$

Softmax is one of the primary computational bottlenecks for Transformers on FPGAs due to its quadratic complexity and use of exponential: an operation very poorly suited to programmable logic. Furthermore, Vitis AI implements Softmax on the CPU by default which adds significant latency as described in Section 2.5.2 from moving data off-and-on of the DPU. However, MHL only uses softmax during training, leaving only one softmax for inference: the classifier, the final layer. This removes all unnecessary CPU/DPU transactions, as the output of the DPU always goes to DRAM and the CPU after inference regardless of the final layer of the model.

3.3.4 Layer Norm

Layer Normalization [41] is applied after both the MHA and feed-forward components of the encoder and serves to project attention keys onto a single hyperplane and scale them to prevent the problem of "unselectable" keys [42]. For some input array x and stabilizing factor ϵ, it is given as:

$$\text{LayerNorm}(x) = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma * \beta, \tag{3.3}$$

for the learnable affine transform parameters γ and β which are established during training. Normalization is performed over the timestep dimension, meaning manual implementation using a Vitis HLS plugin would prevent the model from being length variable.

Batch Normalization is supported by the DPU and is very similar to LayerNorm in theory, save that it normalizes over the batch dimension. This theoretically allows us to implement LayerNorm simply by transposing data before and after, but this method has two limitations. First, BatchNorm doesn't have learnable parameters, resetting γ to 1 and β to 0 with each iteration. This can reduces performance. Second, transposing inputs is done on the ARM core CPU of the SoC, requiring more data transfers and increasing latency. However, we hypothesize that the normalization and limited dynamic range of the data from our pre-processing limits the need for scaling. We evaluate accuracy with and without LayerNorm and the transposed BatchNorm in Section 4.2.3.

3.4 Reconfigurability on the Edge

We propose the following system to satisfy our design goals for adaptability while achieving sufficient performance for live RFFI of LoRa transmissions on the edge.

3.4.1 Device Enrollment Through Transfer Learning

The re-training required to enroll a new device is not under the same latency constraints as inference, since it is a one-off operation. However, it is still desirable to minimize runtime. This is particularly true due to our requirement to perform retraining locally on the accelerator or an attached edge processor. There are a wide array of techniques that make neural network training on memory constrained devices possible [43], albeit prohibitively slow. The two largest components of enrollment runtime are data collection and model training. RFFI neural networks require thousands of training samples to achieve high accuracy, and with the limited throughput of IoT protocols such as high spreading factor LoRaWAN, data collection alone can take multiple hours.

Transfer learning can lower the number of training samples and epochs required to achieve a target accuracy considerably, shortening both data collection and training time, as demonstrated in previous works as described in Section 2.4. By strategically freezing and fine-tuning from a model checkpoint trained on n devices, an RFFI model can be retrained for $n + 1$ devices with a smaller dataset and/or less epochs. This expedites model retraining and fine-tuning on the edge to generate new configurations for RED-RFFI. However, this is not as effective when transferring between communication protocols [32].

3.4.2 System Design Summary

We implemented the RED-RFFI accelerator using the Xilinx ZCU111 evaluation board with the Zynq UltraScale+ RFSoC. We chose this board due to its integrated DAC, ADC, and other wireless components which facilitate future work performing inference on live collected samples.

RED-RFFI is deployed with either a pre-made model as previous works have assumed or a generic, baseline model trained on similar devices. Any unregistered devices are then enrolled at accelerated speed on the local CPU using transfer learning. Alternatively, a model can be trained fresh if sufficient time is provided. Once a final model is produced, it is quantized and calibrated to the dynamic range of the dataset using the Vitis AI Quantization Python library. The quantized model is then compiled to DPU instructions by the Vitis AI compiler in the form an xmodel file. This process is not particularly computationally expensive and can be performed on the local CPU. However, at this time the Vitis AI deployment is only compatible with x86 CPUs, while the Xilinx SoC uses an ARM core processor. This is primarily due to dependencies in the environment deployment process, so adapting it for ARM processors is likely feasible. However, this is beyond the scope of this current work, so we perform proof of concept with the LattePanda 3 Delta, a low-power, single-board computer (SBC) driven by the Intel Celeron N5105. We can use the SBC to update models through transfer learning and convert it to an xmodel. The xmodel DPU instructions are then sent to the FPGA over USB, as shown in Figure 3.2 which depicts our experimental setup.

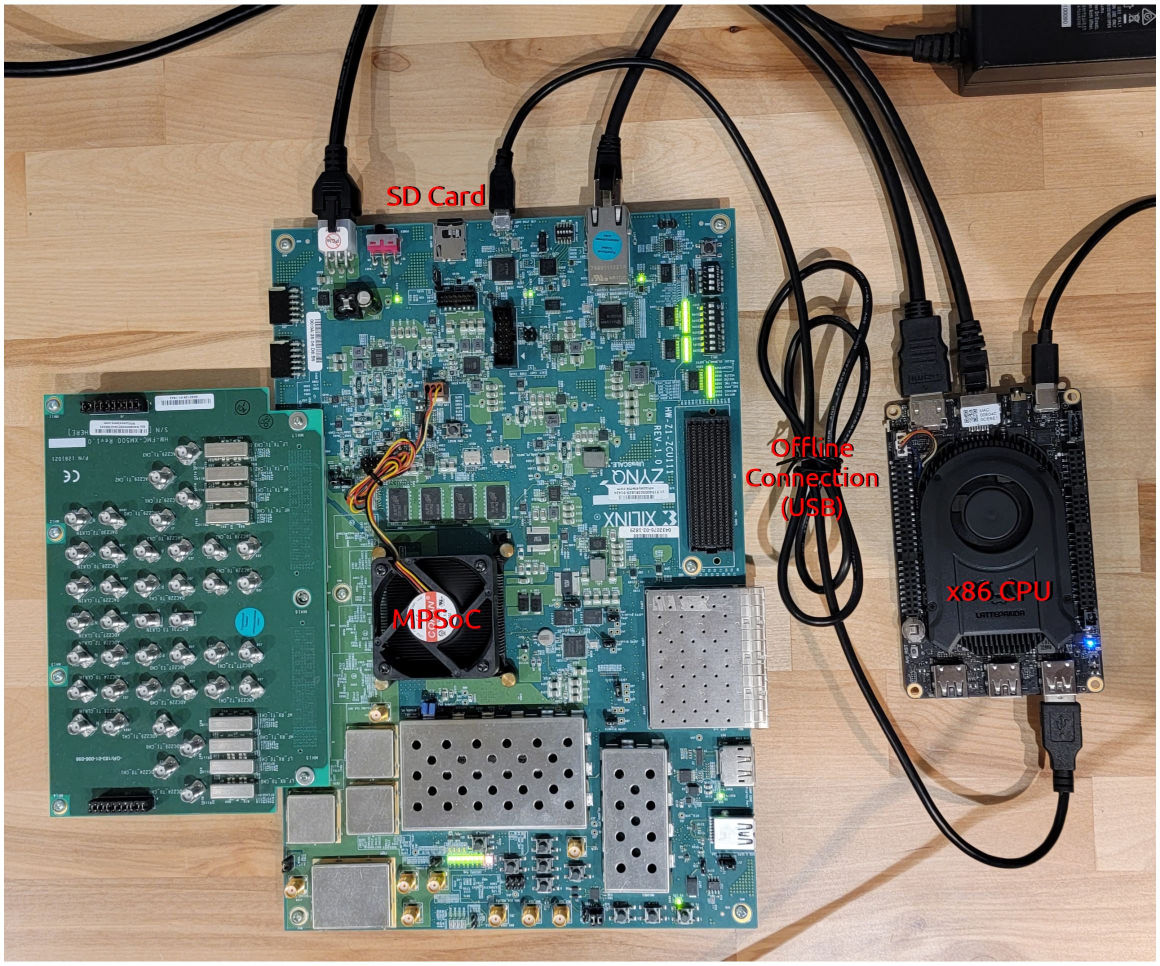

Figure 3.2 : Experimental setup for RED-RFFI

Once the xmodel is prepared, inference is performed on datasets stored on the SD card. However, future work will explore performing live inference on samples gathered via the ZCU111. Depending on the number of DPU cores deployed to the device (from 1-4) and the clock frequency of the DPUs, multiple threads can be run on the SoC CPU to perform inference on each core concurrently. This allows RED-RFFI to fingerprint multiple devices simultaneously or analyze multiple transmissions from the same transmitter to improve accuracy through multi-packet inference. However, either method necessarily increases latency regardless of parallelism, since RED-RFFI must wait to receive each preamble.

Through this workflow, RED-RFFI can operate fully independently. Figure 3.3 depicts this and our hardware platforms in a detailed block diagram. Blocks surrounded by a dotted red line are supported on the platform, but not utilized in this work. The accelerator can perform inference and device enrollment without transferring sensitive information off-device or requiring hardware design iteration, satisfying our design goals.

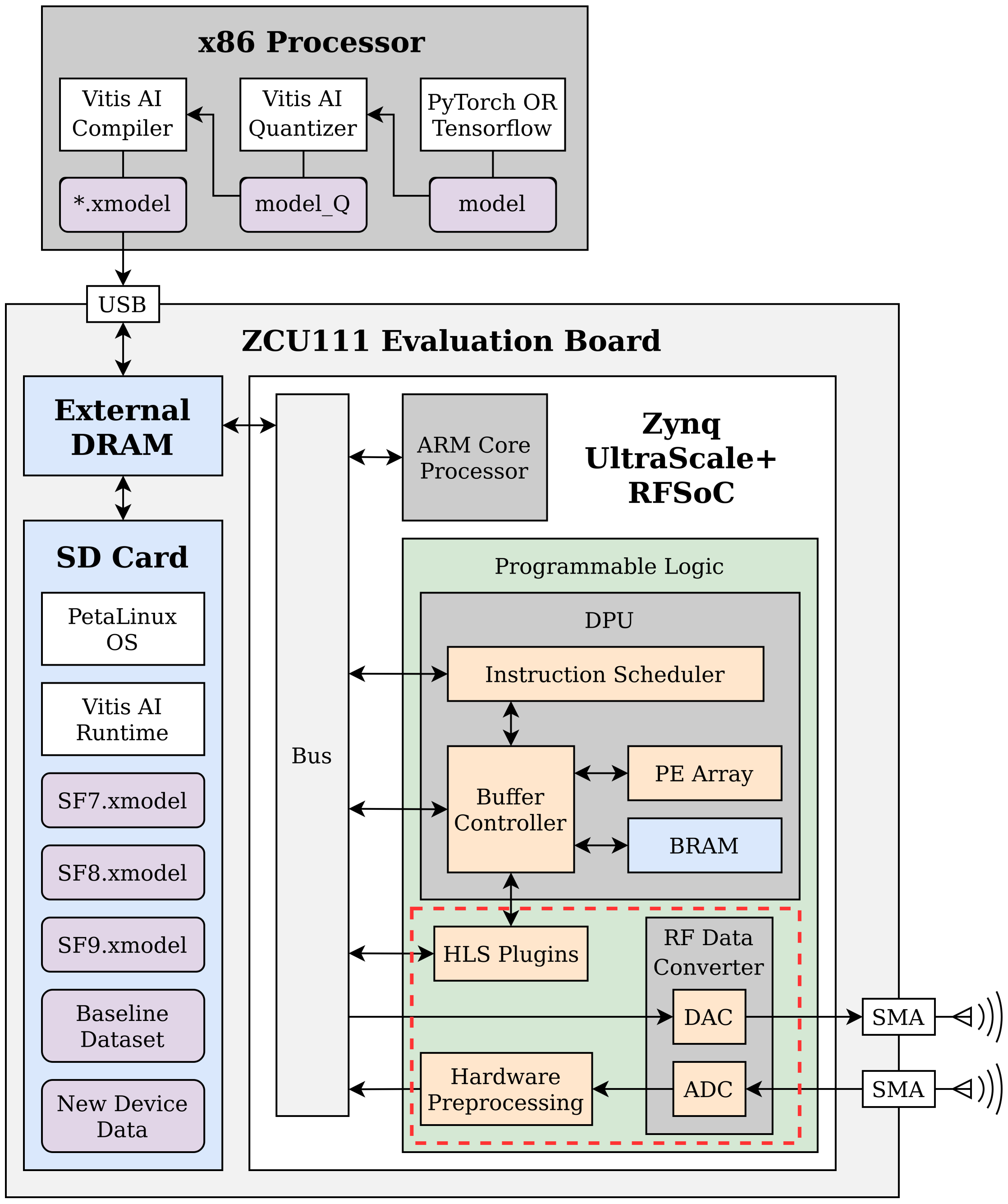

Figure 3.3 : Block design of RED-RFFI

Chapter 4

Experimentation and Implementation

With design constraints for security and adaptability established, we iterated our model within those confines to maximize accuracy and minimize latency on the accelerator. This chapter describes the results of that iterative process and the hardware platforms used in our work.

4.1 Dataset Selection and Creation

The LoRa RFFI data [44] used to develop the baseline Transformer architecture in [1] is the primary dataset used in this work, and henceforth referred to as the "Liverpool dataset." We chose this dataset primarily due to its robust representation of the LoRa transmission standard for RFFI. By utilizing transmissions from consumer LoRa devices rather than simulation data or manually manipulated power amplifiers, our test results are more representative of real-world use cases. In our tests using the dataset, we evaluated transmissions from 9 Pycom LoPy4 devices captured by a USRP N210 software-defined radio (SDR). Each device sent 3000 preambles for spreading factors 7, 8, and 9, transmitted at a bandwidth of 500 kHz. These are the three fastest spreading factors at the highest bandwidth in LoRa, and therefore the most difficult test case in terms of throughput.

However, transfer learning performs best when inferring from a large dataset, motivating us to collect additional data using the Skylark Faros SDRs, programmed

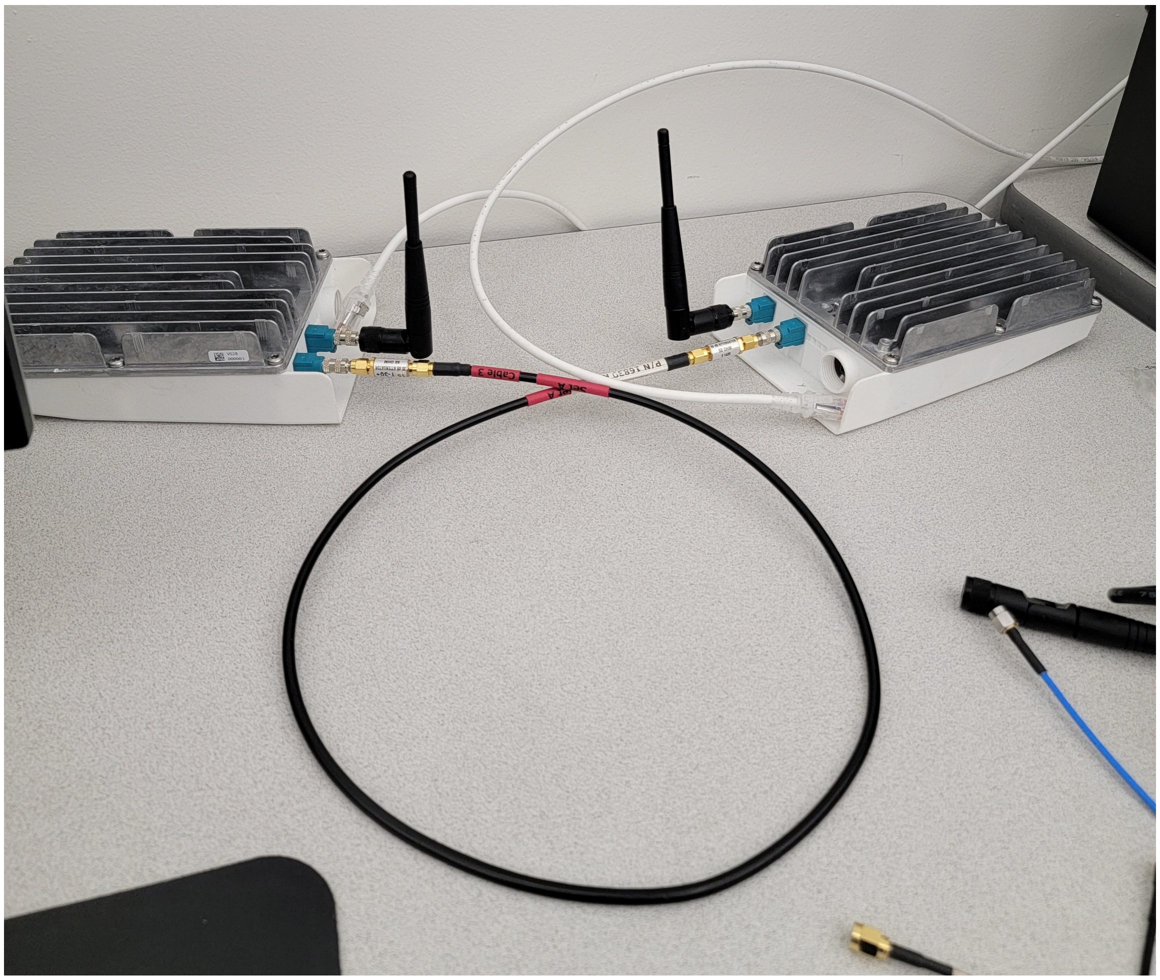

Figure 4.1 : Experimental setup for gathering the Skylark dataset

using SoapySDR. The second generation of the Skylark Iris SDR [45], this SDR is controlled by dual-core ARM A9 processors and a Xilinx Zynq XC7Z030 SoC FPGA. A Lime Microsystems LMS7002M field-programmable RF (FPRF) transceiver allows the center frequency to be tuned up to 3.8 GHz with 122.88 MHz SISO bandwidth. We used [46] to generate a LoRa preamble at various spreading factors, transmitting at the standard LoRa carrier frequency of 915 MHz and bandwidth of 250 kHz. We transmitted from 8 different SDRs and received at a sampling rate of 500 KHz (the

Nyquist frequency of the transmitting bandwidth). Additionally, we sampled at 5 MHz and 10 MHz in an attempt to capture potential transient features from the transmitter's power amplifier. Each dataset was collected over one day with 10,400 preambles from every device at each sampling rate, for a total of three datasets of 83,200 preambles. As this work focuses on adaptability rather than noise compensation, we transmitted over a wired coax connection with a 30 dB attenuator. This data is referred to as the "Skylark dataset" for the remainder of this work. For both datasets, tests at lower SNR were simulated using additive white Gaussian noise.

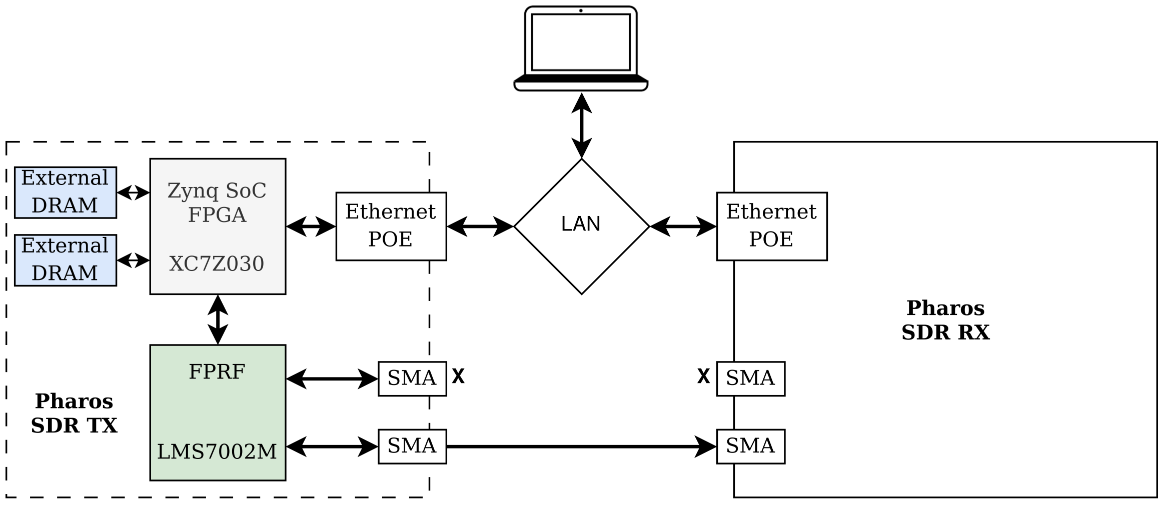

Figure 4.2 : Block diagram of the Skylark Pharos SDR

In gathering the Skylark dataset, we observed two issues. First, due to difficulties syncing transmission and reception sampling, our data experienced significant phase drift, with the in-phase component bleeding into the quadrature and vice versa. This was corrected by taking the norm of the signal for the purposes of detection and alignment. In inference, this is corrected through the normalization component of calculating the channel independent spectrogram. Second, the received signal was consistently inverted. This was trivial to correct, but took some time to detect due to the existing data malformation from phase drift. Initial evaluation of the Skylark dataset sees a remarkable increase in accuracy compared to the training and evaluation of the Liverpool dataset: 20-30 percentage points higher at lower SNRs, the most challenging test case. This is expected however, as wired connections are far more protected from sources of channel noise than wireless communication.

4.2 Model Architecture Optimization

In this section, we explore the efficacy and flexibility of the techniques and model alterations we propose in Section 3.3. We performed all tests on the Liverpool dataset with a training/testing/validation split of 7/2/1 for fair comparison against the original Transformer architecture. In experimentation, we trained using the Adam optimizer for 40 epochs with a learning rate of 0.001 that decreases as loss plateaus. In final model development, we trained with 30 epochs based on our observations of how the model's loss stabilized. We developed our model using an RTX 4090 for acceleration. We worked iteratively in both PyTorch and Tensorflow, as Vitis AI's layer conversion capabilities vary depending on the library used to develop the model. The final implementation and tests use Tensorflow Keras.

4.2.1 Attention

Although LightConv is far more lightweight than a full convolutional layer, using MHL instead of MHA increases the parameter count for a two-encoder Transformer model from 100,537 to 125,545. This increases inference latency and requires more BRAM on the FPGA, but it is necessary to utilize Vitis AI and the DPU. MHL consistently outperforms MHA in terms of accuracy, as seen in Figure 4.3. This accuracy gap is

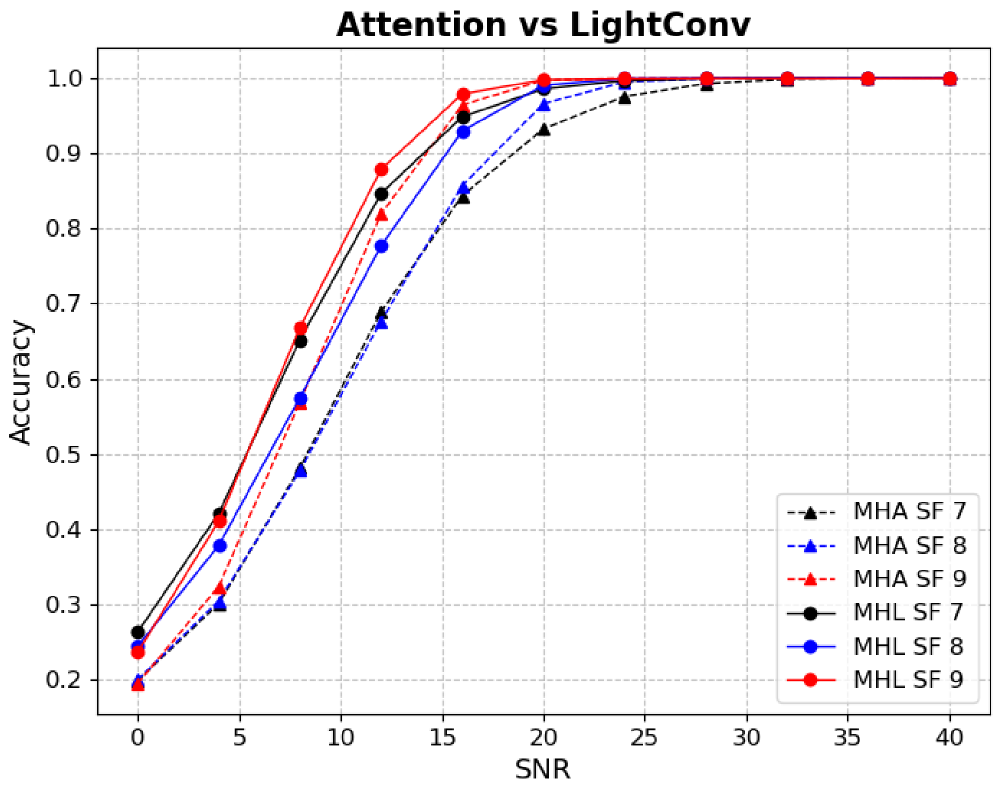

Figure 4.3 : Multi-head Attention vs Multi-Head LightConv Transformer

more significant at lower spreading factors, with HLS improving accuracy by up to 17 percentage points for SF 7 (35% increase) but only up to 11 percentage points for SF 8 (20% increase). This is due to MHA's unlimited context size capturing more information from longer signals.

4.2.2 Layer Streamlining

Removing the positional encoder had an impact of less than a 1 percentage point for any given test and actually produced better performance on average than with the encoder. Layer normalization had a more considerable effect on accuracy; removing the LayerNorm cost between 1-6 percentage points depending on the SNR. The transposed BatchNorm only regained a small amount of this difference (around 0.05 percentage points on average) as shown in Table 4.1; not enough to be a compelling option. Accuracy is very similar at SNRs greater than 24 for all scenarios tested.

Table 4.1 : Accuracy for layer streamlining experiments

SNR	PE & LayerNorm	LayerNorm	BatchNorm	Normless
0	26.30%	26.02%	23.44%	23.26%
4	42.11%	41.40%	37.84%	37.49%
8	65.19%	64.29%	60.07%	59.55%
12	84.65%	83.67%	80.48%	79.53%
16	94.85%	93.94%	92.33%	91.74%
20	98.56%	97.98%	97.62%	97.17%
24	99.65%	99.42%	99.35%	99.18%

LayerNorm is not supported by Vitis AI, so using it on the accelerator requires a CPU implementation as explained in Section 3.3.4. This adds two transactions to and from the CPU for every encoder layer, as well as the compute time for the LayerNorm on the ARM core. Initial tests indicated this would add latency on the scale of milliseconds, an unacceptable delay to achieve real-time LoRa RF fingerprinting. Once the model is finalized and deployed to the accelerator, we have little to no options to expedite CPU transaction latency. However, accuracy can be improved through multi-packet inference. Therefore, we chose to forgo a normalization layer.

4.2.3 Hyperparameter Tuning

The chosen model architecture has several parameters that can be varied to further improve performance. We explored the design space of these variables to finalize our model for acceleration.

Attention Heads

Our model utilizes an embedding dimension of 64 for our channel independent spectrogram, creating an input size of $[T, 64]$ for timesteps T which is defined by the spreading factor and the receiver's sampling rate. This input is split into h attention heads of size $[T, d]$ such that $d = \frac{64}{h}$, as depicted in the *Lightweight Convolution* block in Figure 3.1. When testing head numbers 4, 8, and 16, we found that 8 heads provides the best accuracy. More heads allow for the learning of more independent features, while a larger head dimension d allows for the localized context of LightConv to become more robust. We posit that 8 heads performs the best due to striking a balance between these competing interests.

Depth and Kernel Size

Many Transformer models see significant performance benefits from deepening: stacking more and more Transformer encoders. However, in our initial testing with the baseline Transformer model we found that accuracy either plateaued or degraded past two encoders. LightConv complicates testing model depth by introducing the parameter of kernel size. Viable kernel sizes for attention heads of length 8 are 1, 3, 5, and 7. We tested all permutations of our candidate kernel sizes of a descending shape (e.g. [7,5,3] but not [3,5,7]) up to 5 encoders deep. The best performing kernel sizes for each model depth are [7], [7,5], [7,3,3], [7,7,3,1], and [7,7,5,3,1], whose performance is shown in Tables 4.2 and 4.3.

Table 4.2 : Average accuracy of models with different depths

SNR (db)	1 Layer	2 Layers	3 Layers	4 Layers	5 Layers
0	22.73%	23.26%	25.27%	25.02%	23.78%
4	37.09%	37.49%	40.11%	40.03%	38.20%
8	59.22%	59.55%	62.61%	62.87%	60.25%
12	79.62%	79.53%	82.05%	83.04%	79.88%
16	91.60%	91.74%	92.84%	93.92%	91.45%
20	97.08%	97.17%	97.80%	98.47%	96.96%
24	99.06%	99.18%	99.48%	99.74%	99.12%

Table 4.3 : Peak accuracy of models with different depths

SNR (db)	1 Layer	2 Layers	3 Layers	4 Layers	5 Layers
0	24.09%	25.10%	26.43%	26.30%	26.34%
4	38.60%	40.19%	42.84%	42.25%	42.54%
8	61.31%	63.99%	67.17%	65.50%	66.15%
12	81.96%	83.46%	86.58%	84.53%	85.00%
16	93.70%	94.03%	96.28%	95.00%	94.89%
20	98.33%	98.29%	99.33%	98.99%	98.56%
24	99.73%	99.66%	99.93%	99.89%	99.65%

All models scored above 99.9% at high SNRs (above 24db). For low and medium SNRs, average accuracy across models increases with depth through 4 encoders deep, beginning to degrade at 5, as shown in Table 4.2. However, we observed a significant variance between different instances of the same model, as much as 3 percentage points of accuracy. Analyzing the single best performing model instance for each depth, we found that the 3 layer model performs the best, as seen in Table 4.3.

4.2.4 Quantization

Initial investigation into the viability of quantization performed in Brevitas indicated high resiliency to quantization, with the baseline Transformer implementation maintaining surprisingly high accuracy down to 4 and 2 bit data formats. However, the DPU is optimized for 8-bit integer computation, so we do not explore aggressive quantization in this work. Through Vitis AI's quantizer, we applied post-training quantization while calibrating the dynamic range for quantization with 1000 samples from the testing dataset for the relevant spreading factor. We then applied a coarse fine-tuning algorithm on the quantized model per Vitis AI's workflow.

We performed model training for the LightConv Transformer in single precision floating-point (FP32) and then quantized the parameters to 8-bit integer (Int8). Accuracy degradation varied with both model depth and spreading factor. The 2-layer model suffered the least accuracy loss, and its results are depicted in Figure 4.4. All spreading factors were resilient to accuracy loss in high SNR tests, but for low SNRs SF 9 lost up to 6 percentage points and SF 7 lost at most 2 percentage points.

4.3 Hardware Design Tools and Methodology

The design flow of RED-RFFI begins in Vivado, implementing the DPUCZDX8G IP block and Vitis HLS to design any custom logic. The hardware design is then synthesized and implemented to a file defining the programmable logic for the FPGA.

The software stack to control the system is implemented from this definition in the Vitis workflow, generating the device's file tree. This is then built into a custom PetaLinux image along with the Vitis AI Library and Vitis AI Runtime (VART). Finally, the design is deployed to the ZCU111 as a Vitis Platform. The Verilog component of this process alone takes more than 50 minutes on an Intel i9 processor, and building the PetaLinux image lasts between 30 minutes and 1 hour depending on how many optional features are included in the configuration. For a dataflow-style accelerator, this process needs to be repeated for any dimensionality changes. In contrast, loading an xmodel file for our overlay-style accelerator takes less than a second.

To run tests and benchmark performance, we utilized DPU-PYNQ to connect the Vitis AI Runtime with a PYNQ overlay. This library allows easy access to the accelerator through a Jupyter Notebooks interface over LAN. This may not be advisable for secure deployment, but was useful for our iterative testing. DPU-PYNQ requires an AXI interconnect to facilitate an AXI Verification IP. This adds latency to the design, but is also necessary for any potential HLS plugins which is how locally received LoRa packets from the RFSoC will be preprocesssed and routed to the DPU in future work. After this point, the hardware design flow does not need to be touched again by the user.

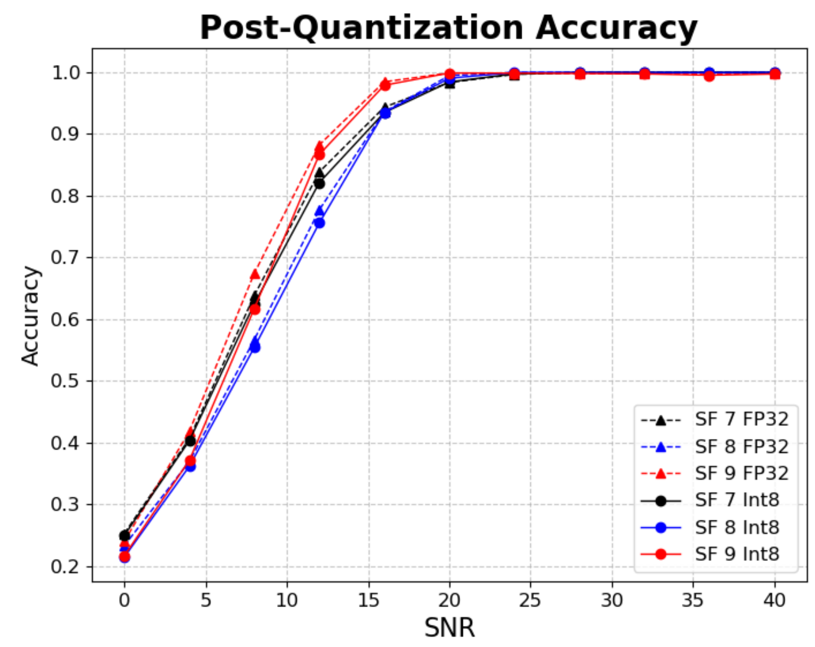

Figure 4.4 : Accuracy before and after quantization

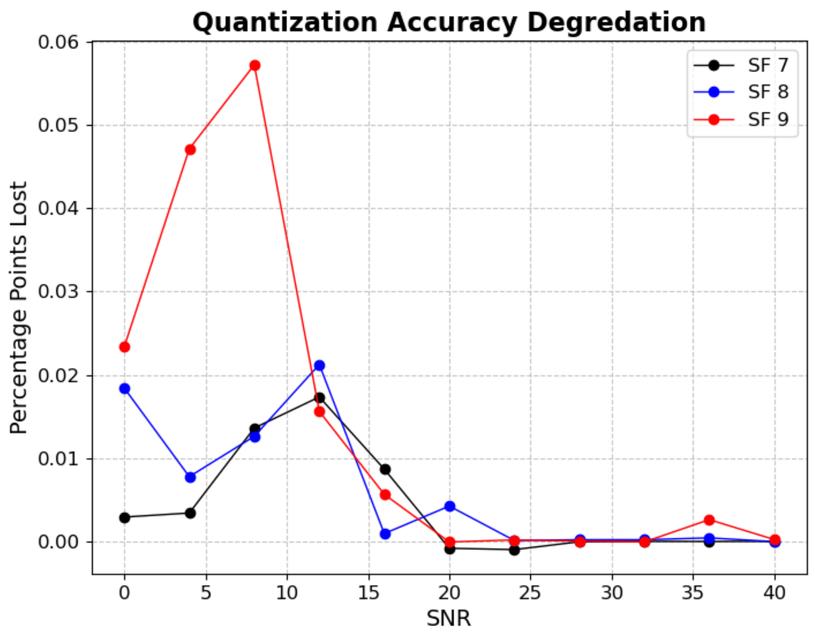

Figure 4.5 : Percentage points lost after quantization

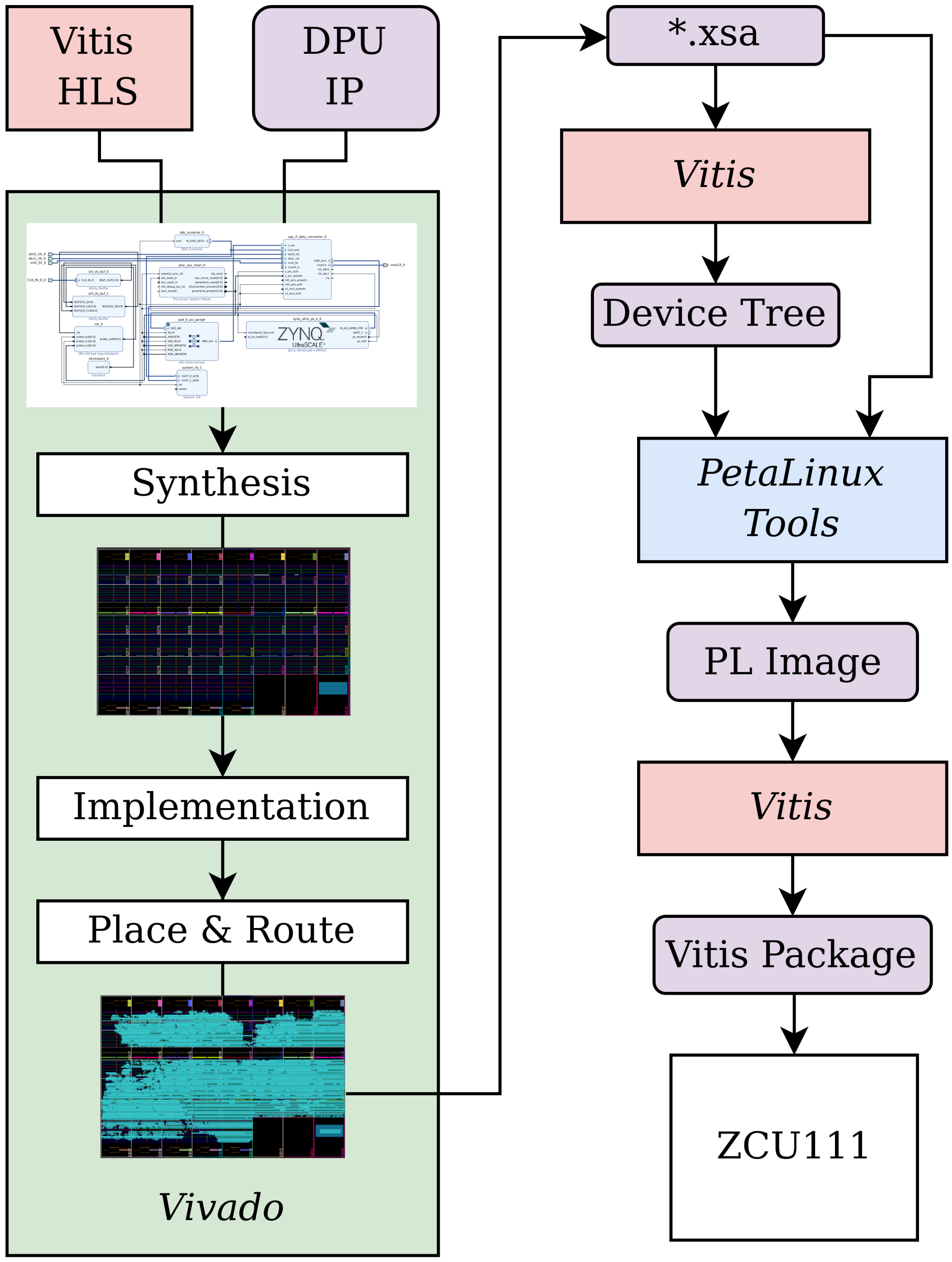

Figure 4.6 : Xilinx design tool workflow

Chapter 5

Accelerator Results and Analysis

5.1 FPGA Utilization

Minimizing utilization of the FPGA is valuable to reduce the power requirements of the device. The most plentiful resources on the MPSoC are lookup tables (LUT) and flip-flops (FF), which can be used together to implement Boolean logic, distributed RAM, and shift-registers. On-chip block RAM (BRAM) and LUT RAM are used for larger data storage, while phase-locked loops (PLL) and simple buffers (BUFG) are used for clocking. The final core components are the digital signal processing slices: generalized multiply-accumulate accelerators for DSP.

The DPU taxes the MPSoC's BRAM and DSP resources the most heavily. We examined configurations of 1, 2, and 3 DPU cores which have a maximum power draw of 10.03 W, 15.86 W, and 21.79 W respectively. Using the DPU-PYNQ framework adds a small additional overhead which we characterized separately. Post-implementation utilization reports for each design are in Table 5.1. The number of DPU cores for a given implementation should be chosen based on available power resources, the resource needs of any other IP blocks on the PL, and desired throughput. Less cores should be deployed for lower-resource scenarios and more cores for demanding use cases where power is more ample.

Our design must store model configurations for each desired communication protocol and datarate in the form of xmodel files. These range in size from 278 KB for a single-encoder model analyzing SF 7 to 2.8 MB for a three-encoder model analyzing SF 9. The primary form of storage for Xilinx SoCs is through flash SD cards, a cheap and plentiful form of storage.

Table 5.1 : Resource utilization for RED-RFFI on the RFSoC

System	LUT	LUTRAM	FF	BRAM	DSP	BUFG	PLL
1 DPU	60973	7283	107219	259	724	3	1
2 DPU	111666	14032	204753	514	1434	4	1
3 DPU	162175	20782	303652	769	2144	5	1
DPU-PYNQ	2376	410	2464	0	13	0	0
Capacity	425280	213600	850560	1080	4272	696	16

5.2 Latency and Throughput

As calculated in Section 2.1.1, to perform real-time RF fingerprinting of LoRa signals as they arrive, RED-RFFI must have an end-to-end latency of 2.048 ms, 4.096 ms, and 8.192 ms for processing SF 7, SF 8, and SF 9 signals respectively. This can also be calculated on a per-sample rate, in which case all spreading factors require a latency of 1 μs to fingerprint one sample on average.

To evaluate if RED-RFFI meets this standard, we performed continuous inference on the Liverpool test dataset, one sample at a time as if they were freshly arriving. The runtime required to process the entire dataset was then divided by the number of test cases to determine latency per signal. This value was then divided by the length of a signal of the same spreading factor to obtain average latency per sample. Our LightConv Transformer achieves varying accuracy depending on the depth of model used, as demonstrated in Section 4.2.3. However, more layers create more latency in the model, negatively impacting throughput. To thoroughly assess RED-RFFI's capabilities, we performed latency tests for model depths 1 to 3.

We measured both end-to-end latency and DPU latency for each model size and spreading factor. End-to-end latency is the most important measurement, representing the time to perform RF fingerprinting and classification on an input spectrogram. DPU latency only measures the runtime of the DPU. This still characterizes the vast majority of the accelerator, and is useful for evaluating how much latency could potentially be eliminated by future work replacing non-DPU operations such as the ARM core Softmax used for final classification. The results of these experiments are detailed in Table 5.2.

Table 5.2 : RED-RFFI latency for real-time LoRa fingerprinting

		Latency per Signal			Latency per Sample		
		SF 7	SF 8	SF 9	SF 7	SF 8	SF 9
End to End	1 Layer	1.004 ms	1.411 ms	2.248 ms	0.490 μs	0.344 μs	0.274 μs
	2 Layer	1.463 ms	2.238 ms	3.867 ms	0.714 μs	0.547 μs	0.472 μs
	3 Layer	1.912 ms	3.081 ms	5.484 ms	0.934 μs	0.752 μs	0.669 μs
DPU	1 Layer	0.648 ms	1.043 ms	1.891 ms	0.317 μs	0.255 μs	0.231 μs
	2 Layer	1.104 ms	1.876 ms	3.505 ms	0.539 μs	0.458 μs	0.428 μs
	3 Layer	1.553 ms	2.717 ms	5.131 ms	0.758 μs	0.663 μs	0.626 μs
LoRa	–	2.048 ms	4.096 ms	8.192 ms	1 μs	1 μs	1 μs

RED-RFFI performs within specification for real-time LoRa RF fingerprinting at the maximum datarate for models up to 3 layers deep. Examining the latency per sample, we see a downward trend with latency decreasing as spreading factor increases. This implies that models for the slower spreading factors that we did not test (SF 10-12) are likely to operate at sufficient speeds for real-time LoRA RFFI.

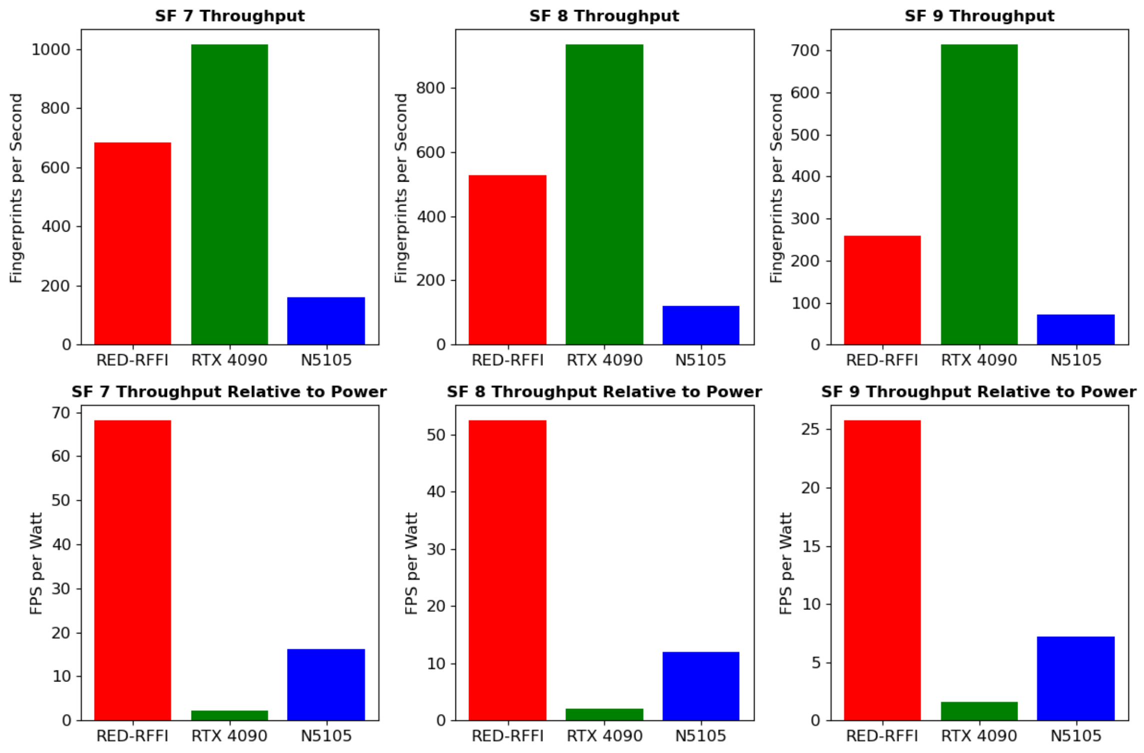

Figure 5.1 : Inference throughput per watt for different platforms

Figure 5.1 compares the throughput of RED-RFFI to the same 2-layer model running on the RTX 4090 used for training and the Intel N5105 representing edge CPUs. Performance is measured in fingerprints per second (FPS) for different spreading factors. Devices are compared in terms of raw throughput and FPS per watt, contextualizing the power requirements highly relevant for IoT deployment. RED-RFFI keeps a decisive lead of 3.5→4.2× throughput when compared to the N5105. Although the RTX 4090 out performs RED-RFFI due to its raw clock speed and resources, RED-RFFI outperforms the RTX 4090 by $16-30\times$ in terms of throughput per watt.

5.3 Accuracy and Adaptability

Although latency is acceptable through 3 layer models, we found the 2 layer model to have the best performance after quantization after quantization across all spreading factors by a thin margin, as noted in Section 4.2.4. We evaluated accuracy with this model on the 1 DPU design across all spreading factors. Some accuracy degredation from Int8 simulations occurred due to optimizations for the DPU performed in compilation, but performance still meets or exceeds the original MHA accuracy by up to 8 percentage points.

To demonstrate the unique flexibility of our design, we performed inference on SF 7 and SF 9 datasets simultaneously on separate DPU cores from the same Python interpreter. PYNQ-DPU does not directly support running multiple DPU subgraphs from the same Python kernel, but by instantiating DPU runners through the Vitis AI Runtime (VART) directly we bypass that restriction. Creating the threads in Python, they then operate asynchronously, resuming upon receiving output from their respective DPU. As each DPU operates independently, accuracy and the throughput of each DPU were unaffected.

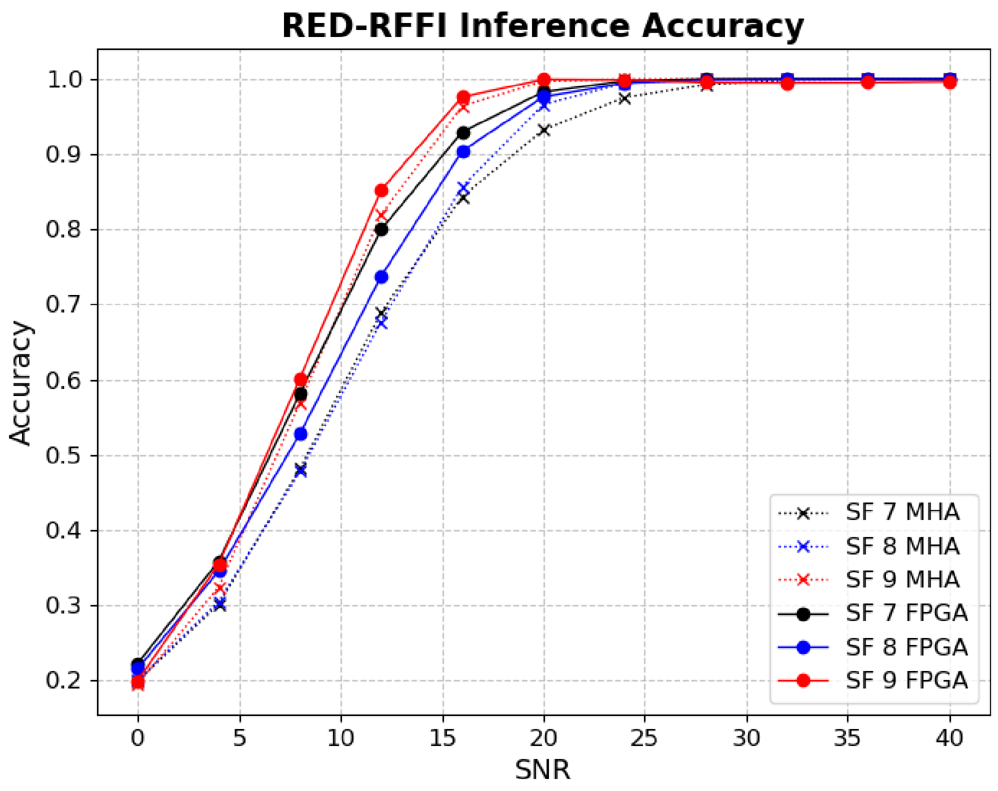

Figure 5.2 : RED-RFFI inference accuracy vs baseline MHA model

Chapter 6

Conclusion

In this thesis, we explored the feasibility of Radio Frequency Fingerprint Identification for authenticating IoT devices on the edge. We reviewed the state-of-the-art solutions and demonstrated key assumptions in previous works that prevent practical use in resource constrained scenarios. We detailed the design process of RED-RFFI: the first Transformer-based FPGA accelerator for RF fingerprinting. Its programmable DPU design makes it the only RFFI accelerator that can be dynamically reconfigured for changing devices, transmission protocols, and data rates. This uniquely allows it to meet the highly variable conditions and configurations of IoT networks on the edge, while still achieving best in class RFFI accuracy for LoRa.

Previous RFFI accelerators follow a dataflow-style design which generally achieves higher throughput than overlay-style accelerators [33]. However, this added performance isn't beneficial if the neural network doesn't reflect present network conditions. In our experimentation, RED-RFFI achieved sufficient throughput to perform real-time LoRa RF fingerprinting without this level of restrictive design. Additionally, we performed simultaneous inference on signals of different length and spreading factor on the same low-power device: another novel feat for RFFI accelerators.

6.1 Future Work

RED-RFFI is a platform for numerous avenues of future research. Our proposed future work on the accelerator consists of three components: refinement, expansion, and exploration.

Refinement

As demonstrated in Table 5.2, the few CPU-bound operations add significant latency to the design. An alternate version of the DPU can be configured to enable a hardware softmax, but this adds dimensionality restrictions. However, this could be potentially be resolved by dynamically padding output data to a maximum size. This is promising future work to further improve throughput.

Another potential measure to improve performance is implementing LayerNorm on the FPGA. The only difference between LayerNorm and BatchNorm are the weights and biases. If the BatchNorm engine in the DPU could be modified to initialize with trained weights, this would functionally implement LayerNorm on the DPU.

Expansion

Although RED-RFFI has the throughput to perform real-time LoRa RF fingerprinting, it currently lacks the ability to receive and preprocess live samples. The ZCU111 contains all the necessary components to perform this process, but requires integration with the DPU workflow. The preprocessing workflow is largely length independent and could be performed in the programmable logic of the FPGA, except for the STFT. Although FFTs are commonly accelerated using FPGAs, the window size of the STFT varies depending on the transmission scheme and signal bandwidth. Therefore, different HLS kernels would be required for each transmission scheme we wish to support.

Exploration

The ability to process signals with different lengths and properties is one of the primary features of RED-RFFI. However, our current implementation only uses this to analyze LoRa signals of different length. By training models for other transmission standards such as ZigBee and WiFi, our accelerator could perform fingerprinting in heterogeneous networks: another novel feat for RFFI accelerators.

Bibliography

[1] G. Shen, J. Zhang, A. Marshall, M. Valkama, and J. R. Cavallaro, “Toward Length-Versatile and Noise-Robust Radio Frequency Fingerprint Identification,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2355–2367, 2023.

[2] E. Bertino and N. Islam, “Botnets and Internet of Things Security,” *Computer*, vol. 50, no. 2, pp. 76–79, 2017.

[3] A. Arampatzis, “Top 10 Vulnerabilities that Make IoT Devices Insecure,” 2021. Last accessed 14 July 2023.

[4] W. Trappe, R. Howard, and R. S. Moore, “Low-Energy Security: Limits and Opportunities in the Internet of Things,” *IEEE Security Privacy*, vol. 13, no. 1, pp. 14–21, 2015.

[5] L. Vailshery, “Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2021, with forecasts from 2022 to 2030,” 2022. Last accessed 14 July 2023.

[6] J. Zhang, G. Li, A. Marshall, A. Hu, and L. Hanzo, “A New Frontier for IoT Security Emerging From Three Decades of Key Generation Relying on Wireless Channels,” *IEEE Access*, vol. 8, pp. 138406–138446, 2020.

[7] P. Verma and N. Bharot, “A Review on Security Trends and Solutions Against

Cyber Threats in Industry 4.0," in *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pp. 397–402, 2023.

[8] S. Bindra and A. Malik, "An Analysis Of Anomaly Detection Techniques for IoT Devices: A Review," in *2023 Third International Conference on Secure Cyber Computing and Communication (ICSCCC)*, pp. 275–280, 2023.

[9] S. A. Baho and J. Abawajy, "Analysis of Consumer IoT Device Vulnerability Quantification Frameworks," *Electronics*, vol. 12, no. 5, 2023.

[10] J. L. Hernández-Ramos, G. Baldini, S. N. Matheu, and A. Skarmeta, "Updating IoT devices: challenges and potential approaches," in *2020 Global Internet of Things Summit (GIoTS)*, pp. 1–5, 2020.

[11] J. Toonstra and W. Kinsner, "A Radio Transmitter Fingerprinting System ODO-1," in *Proceedings of 1996 Canadian Conference on Electrical and Computer Engineering*, vol. 1, pp. 60–63 vol.1, 1996.

[12] A. Jagannath, J. Jagannath, and P. S. P. V. Kumar, "A Comprehensive Survey on Radio Frequency (RF) Fingerprinting: Traditional Approaches, Deep Learning, and Open Challenges," *Comput. Netw.*, vol. 219, dec 2022.

[13] G. Shen, J. Zhang, A. Marshall, L. Peng, and X. Wang, "Radio Frequency Fingerprint Identification for LoRa Using Deep Learning," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 8, pp. 2604–2616, 2021.

[14] T. Jian, Y. Gong, Z. Zhan, R. Shi, N. Soltani, Z. Wang, J. Dy, K. Chowdhury, Y. Wang, and S. Ioannidis, "Radio Frequency Fingerprinting on the Edge," *IEEE Transactions on Mobile Computing*, vol. 21, no. 11, pp. 4078–4093, 2022.

[15] Q. Jiang and J. Sha, "RF Fingerprinting Identification Based on Spiking Neural Network for LEO–MIMO Systems," *IEEE Wireless Communications Letters*, vol. 12, no. 2, pp. 287–291, 2023.

[16] H.-T. Peng, J. C. Lederman, L. Xu, T. F. de Lima, C. Huang, B. J. Shastri, D. Rosenbluth, and P. R. Prucnal, "A Photonics-Inspired Compact Network: Toward Real-Time AI Processing in Communication Systems," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 28, no. 4: Mach. Learn. in Photon. Commun. and Meas. Syst., pp. 1–17, 2022.

[17] J. Xu, Y. Shen, E. Chen, and V. Chen, "Bayesian Neural Networks for Identification and Classification of Radio Frequency Transmitters Using Power Amplifiers' Nonlinearity Signatures," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 457–471, 2021.

[18] H. Zeng and Y. Xie, "Design and Implementation of a Radio Frequency Fingerprint Identification System," in *2022 4th International Conference on Intelligent Information Processing (IIP)*, pp. 42–46, 2022.

[19] C. Xenofontos, I. Zografopoulos, C. Konstantinou, A. Jolfaei, M. K. Khan, and K.-K. R. Choo, "Consumer, Commercial, and Industrial IoT (In)Security: Attack Taxonomy and Case Studies," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 199–221, 2022.

[20] M. Stead, P. Coulton, J. Lindley, and C. Coulton, *The Little Book of SUSTAINABILITY for the Internet of Things*. 02 2019.

[21] G. Shen, J. Zhang, A. Marshall, M. Valkama, and J. Cavallaro, "Radio Frequency Fingerprint Identification for Security in Low-Cost IoT Devices," in *2021 55th*

Asilomar Conference on Signals, Systems, and Computers, pp. 309–313, 2021.

[22] G. Shen, J. Zhang, A. Marshall, L. Peng, and X. Wang, "Radio Frequency Fingerprint Identification for LoRa Using Spectrogram and CNN," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021.

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

[24] S. Wesemeyer, I. Boureanu, Z. Smith, and H. Treharne, "Extensive Security Verification of the LoRaWAN Key-Establishment: Insecurities Patches," in *2020 IEEE European Symposium on Security and Privacy (EuroSP)*, pp. 425–444, 2020.

[25] S. Kim, H. Lee, and S. Jeon, "An Adaptive Spreading Factor Selection Scheme for a Single Channel LoRa Modem," *Sensors*, vol. 20, no. 4, 2020.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2017.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *CoRR*, vol. abs/2010.11929, 2020.

[28] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain Adaptation via Transfer Component Analysis," *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 199–210, 2011.

[29] M. Long, J. Wang, G. Ding, J. Sun, and P. S. Yu, "Transfer Feature Learning with Joint Distribution Adaptation," in *2013 IEEE International Conference on Computer Vision*, pp. 2200–2207, 2013.

[30] J. Wang, Y. Chen, S. Hao, W. Feng, and Z. Shen, "Balanced Distribution Adaptation for Transfer Learning," in *2017 IEEE International Conference on Data Mining (ICDM)*, (Los Alamitos, CA, USA), pp. 1129–1134, IEEE Computer Society, nov 2017.

[31] T. Tian, Y. Wang, H. Dong, Y. Peng, Y. Lin, G. Gui, and H. Gacanin, "Transfer Learning-Based Radio Frequency Fingerprint Identification Using ConvMixer Network," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, pp. 4722–4727, 2022.

[32] S. Kuzdeba, J. Robinson, and J. Carmack, "Transfer Learning with Radio Frequency Signals," in *2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC)*, pp. 1–9, 2021.

[33] F. Hamanaka, T. Odan, K. Kise, and T. V. Chu, "An Exploration of State-of-the-Art Automation Frameworks for FPGA-Based DNN Acceleration," *IEEE Access*, vol. 11, pp. 5701–5713, 2023.

[34] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, pp. 65–74, ACM, 2017.

[35] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An End-to-End Deep-Learning Framework

for Fast Exploration of Quantized Neural Networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.

[36] Xilinx, "DPUCZDX8G for Zynq UltraScale+ MPSoCs Product Guide (PG338)," 2023. https://docs.xilinx.com/r/en-US/pg338-dpu. Last accessed 22 July 2023.

[37] S. Yao and K. Guo, "Deep Processing Unit (DPU) for Implementing an Artificial Neural Network (ANN)," US Patent Application 2018/0046903 A1, Feb. 2018.

[38] U.S. Bureau of Labor Statistics, *Occupational Employment and Wage Statistics, May 2022*, 2022. https://www.bls.gov/oes/. Last accessed 22 July 2023.

[39] B. Liu, H. Zhang, Y. Wan, F. Zhou, Q. Wu, and D. W. K. Ng, "Robust Adversarial Attacks on Deep Learning-Based RF Fingerprint Identification," *IEEE Wireless Communications Letters*, vol. 12, no. 6, pp. 1037–1041, 2023.

[40] F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, and M. Auli, "Pay Less Attention with Lightweight and Dynamic Convolutions," 2019. https://arxiv.org/abs/2207.03341.

[41] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," 2016.

[42] S. Brody, U. Alon, and E. Yahav, "On the Expressivity Role of LayerNorm in Transformers' Attention: ACL 2023," *Findings of the Association for Computational Linguistics*, p. 14211–14221, 2023.

[43] N. S. Sohoni, C. R. Aberger, M. Leszczynski, J. Zhang, and C. Ré, "Low-Memory Neural Network Training: A Technical Report," 2022. Stanford University.

[44] G. Shen, J. Zhang, and A. Marshall, "LoRa RFFI dataset," 2022. https://dx.doi.org/10.21227/qqt4-kz19.

[45] R. Doost-Mohammady, O. Bejarano, L. Zhong, J. R. Cavallaro, E. Knightly, Z. M. Mao, W. W. Li, X. Chen, and A. Sabharwal, "RENEW: Programmable and Observable Massive MIMO Networks," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 1654–1658, 2018.

[46] J. Tapparel, O. Afisiadis, P. Mayoraz, A. Balatsoukas-Stimming, and A. Burg, "An Open-Source LoRa Physical Layer Prototype on GNU Radio," in *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pp. 1–5, 2020.