Cyclo-AMC: Automatic Modulation Classification on Versal utilising Cyclostationary Features

Ruilin Wu*, Carol Jingyi Li†, Wei Zhang†, Xueyuan Liu*, Philip H.W. Leong*
* School of Electrical and Computer Engineering, The University of Sydney, NSW, Australia
† Reconfigurable Computing Systems Lab, The Hong Kong University of Science and Technology, Hong Kong
{ruilin.wu, xueyuan.liu, philip.leong}@sydney.edu.au, {eecarol, eeweiz}@ust.hk

***Abstract*—Recently, it has been shown that cyclostationary features can greatly improve generalization of deep neural network (DNN) automatic modulation classification (AMC) systems. Unfortunately, its computation has a high computational cost, limiting its application in real-time settings. To address these issues, we present Cyclo-AMC, the first field-programmable gate array (FPGA)-based custom AMC accelerator that combines cyclostationary features with a DNN. Cyclo-AMC runs entirely on an AMD Versal VEK280 AI Engine-Machine Learning (AIE-ML) device and integrates an area-efficient implementation of the FFT accumulation method (FAM) algorithm to extract cyclostationary features, with a compact ResNet-18 model. Cyclo-AMC achieves a $32\times$ reduction in energy consumption over an NVIDIA GeForce RTX 3090 graphics processing unit (GPU). On the CSPB.ML.2022 dataset, Cyclo-AMC achieves 93.8% classification accuracy compared to 81.7% for FINN-A.**

***Index Terms*—Automatic modulation classification, Versal, FPGA, Cyclostationary, Deep Learning, Signal Classification.**

I. Introduction

Recent developments in wireless systems have significantly increased spectrum utilization and the variety of signal formats; a fundamental building block of cognitive radio is the ability to detect unused channels, allowing them to be utilized dynamically [1]. In a non-cooperative setting, blind modulation classification can be utilized for this purpose based on two requirements. Firstly, accurate detection and classification of signals is necessary, and we refer to the problem as automatic modulation classification (AMC). Secondly, this must be done under strict real-time constraints to ensure high utilization of radio channels.

In 2017, O'Shea et. al. [2] classified raw in-phase and quadrature (I/Q) radio samples using a deep neural networks (DNNs) to significantly improve AMC accuracy over conventional approaches. Since then, numerous publications have achieved further improvements using different deep learning techniques, but mostly based on directly learning features from I/Q inputs [3]–[21]. A promising new approach to improve generalization is to first transform the I/Q inputs to cyclostationary features before applying a DNN [22]. This is the approach utilized in the present work.

Unfortunately, extracting cyclostationary features is computationally intensive and often requires hardware acceleration for real-time performance [23]. Graphics processing units (GPUs) are commonly used but have high power dissipation, which limits their practical deployment on edge nodes. Furthermore, the interface between the radio and GPU introduces significant latency [24] and edge devices often have small on-chip memories, limiting the resolution of cyclostationary features and the size of the DNN that can be implemented [25]. This is addressed through field-programmable gate array (FPGA)-based acceleration using a reduced feature representation and knowledge distillation [26] to substantially reduce computational and storage requirements.

To enhance robustness and meet real-time requirements, we introduce Cyclo-AMC, an open-source and reproducible[1] software AMC algorithm, along with an AMD/Xilinx Versal AI Engine-Machine Learning (AIE-ML) VEK280 design. Our novel contributions are summarized as follows:

- Cyclo-AMC is the first single-FPGA AMC system that combines cyclostationary feature extraction with a DNN model. In particular, we integrate a FFT accumulation method (FAM)-based spectral correlation density (SCD) estimator [27], [28] and a lightweight ResNet [29] classifier on a VEK280 device.
- To minimize the memory footprint of our design, we utilize knowledge distillation in the form of a teacher-student model [26]. It achieves an order of magnitude reduction in model size (6.40M vs. 0.59M parameters) with minimal loss in accuracy, allowing us to store all model parameters and intermediate values on the FPGA, resulting in significant performance improvement.
- To the best of our knowledge, our results achieve the highest reported accuracy for an FPGA-based implementation on the CSPB.ML.2022 dataset [30].

The remainder of this paper is structured as follows. Section II reviews the cyclostationary analysis, the FAM algorithm, AMC, and summarizes the Versal architecture. Section III details the algorithmic aspects of Cyclo-AMC and Section IV describes its hardware implementations. Section V presents the experimental results, and Section VI concludes the paper.

II. Background

A process is cyclostationary if its statistics vary periodically over time [31]. The SCD, also known as the cyclic spectral density or spectral correlation function, captures the complete set of spectral cross-correlations and is a common way to characterize cyclostationarity.

[1]The implementation for data preprocessing and model architecture in PyTorch is available at https://github.com/ruilin-wu/Cyclo-AMC.

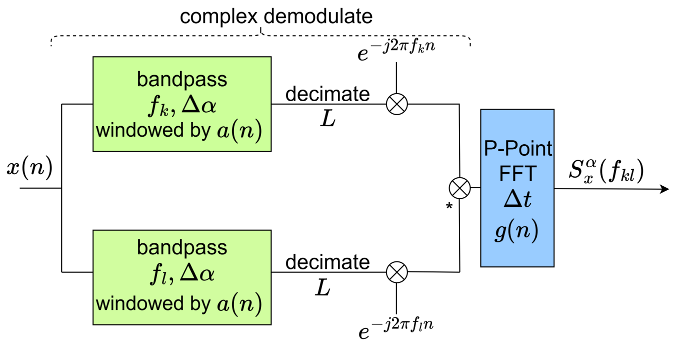

Fig. 1: Dataflow for FAM.

A. *Spectral Correlation Density and the FAM Technique*

The description of the SCD function below follows that of Roberts et. al. [27], [28]. The discrete-time *complex demodulate* of a continuous-time complex-valued signal $x(t)$ at frequency f is

$$X_T(n, f) = \sum_{r=-N/2}^{N/2} a(r)x(n-r)e^{-i2\pi f(n-r)T_s} \tag{1}$$

where $a(r)$ is a length $T = NT_s$ second windowing function, T_s is the sampling period, and N is the number of samples. Complex demodulates are low pass sequences with bandwidths $\Delta f \approx 1/T$. For inputs $x(n)$ and $y(n)$ of length N samples, we correlate demodulates $X_T(n, f_1)$ and $Y_T(n, f_2)$ separated by α_0 ($f_1 = f_0 + \alpha_0/2$, $f_2 = f_0 - \alpha_0/2$) over the time window $\Delta t = NT_s$ using a complex multiplier followed by a low pass filter (LPF) with bandwidth approximately $1/\Delta t$. Thus, the SCD function is given by

$$S_{xy_T}^{\alpha_0}(n, f_0)_{\Delta t} = \sum_r X_T(r, f_1)Y_T^*(r, f_2)g(n-r) \tag{2}$$

where the * operator is a complex conjugate and $g(n)$ is a length $\Delta t = NT_s$ windowing function. For the special case of auto-correlation studied in this paper, $y(n)$ is a time-delayed value of $x(n)$, i.e., $y(n) = x(n+d)$ where d is the delay.

Fig. 1 illustrates the signal flow for the FAM method, in which decimation and the fast Fourier transform (FFT) are applied to reduce the computational requirements of Eq. (2) [27]. We summarize the approach below and refer readers to references [28], [32] for a detailed derivation, with implementation guidance in [33].

1) Decimation: In FAM, the complex demodulate is decimated by the factor L. Reducing the window size of $a(r)$ from N to N_P, Eq. (1) becomes:

$$X_T(pL, f_m) = \underbrace{[\sum_{k=0}^{N_P-1} \underbrace{a(d-k)x(pL-d+k)}_{x(n)\text{ windowed by }a(n)} e^{-i2\pi mk/N_P}]}_{N_P\text{-point FFT}} \underbrace{e^{-i2\pi mpL/N_P}}_{\text{Down Conversion}}, \tag{3}$$

where $p = \{0, 1..., P-1\}$, $d = N_P/2 - 1, r = d - k, f_m = mf_s/N_P$, $f_s = 1/T_s$, and $-N_P/2 < m < N_P/2$ [32]. Thus, in Eq. (3), the input is windowed via $a(n)$, then passed through a N_P-Point FFT. A phase shift is introduced to compensate for the down conversion from N to N_P samples.

2) FAM method: Taking Eq. (2), and substituting $X_T = Y_T$ to compute at the frequency $f_{kl} = (f_k + f_l)/2$ in P segments (Eq. (3)), Eq. (2) becomes

$$S_{xy_T}^{\alpha_0}(pL, f_{kl})_{\Delta t} = \sum_r X_T(rL, f_k)X_T^*(rL, f_l)g_d(p-r) \tag{4}$$

where $p = \{0, 1, ..., P-1\}$ and $g_d(r) = g(rL)$. Now the cycle frequency parameter has been redefined to $\alpha_0 = f_l - f_k + \epsilon$, as the $\epsilon = \Delta f$ is the introduced frequency shift.

Introducing $\epsilon = q\Delta\alpha$ ($\Delta\alpha = f_s/P$ and $q = \frac{\Delta f}{\Delta\alpha}$) to Eq. (4) and substituting $a_{kl} = f_k - f_l$, $f_0 = f_{kl} = (f_k + f_l)/2$, and $\alpha_0 = a_{kl} + q\Delta\alpha$ [32], the following equation for FAM is obtained

$$S_x^{a_{kl}+\Delta\alpha}(pL, f_{kl})_{\Delta t} = \underbrace{\sum_r \underbrace{\underbrace{X_T(rL, f_k)\, X_T^*(rL, f_l)}_{\text{Conjugate Multiplication}}\, g_d(p-r)}_{\text{windowed by }g(n)}\, e^{-i2\pi rq/P}}_{P\text{-point FFT}}. \tag{5}$$

FAM implementations of the AMD ZCU111 [33], [34] and AMD Versal VCK5000 [35] have been proposed but none have integrated it with a DNN.

B. *Deep-Learning in AMC and Dataset Strategy*

1) DNN-Based AMC: Early work by O'Shea et. al. revealed that a DNN trained directly on I/Q samples can outperform cumulant detectors under additive white Gaussian noise (AWGN) [2]. Over-the-air trials confirmed that deeper DNNs retain accuracy in realistic channels [36]. However, it was later revealed that this approach may not generalize well because the I/Q waveform implicitly embeds fading, DC bias, and frequency drift. Exhaustive training would require an impractically large dataset [37], [38]. To address this problem, cyclostationary representations that are invariant to many channel effects were proposed. John et. al. augmented capsule networks (CAPs) with cyclostationary features [39], however it is computationally expensive and poorly suited for implementations on FPGAs.

Cyclo-AMC adopts the FAM for the SCD estimation and couples it with a lightweight ResNet. The SCD representation emphasizes periodic correlations that are insensitive to carrier offset and narrowband interference, while the residual network excels at spatial pattern extraction. FPGA implementations of SCD estimators have previously been proposed [33]–[35], as has a statistics-based AMC employing cyclostationary features with FPGA and Tegra K1 GPU acceleration [40].

2) Datasets: The DeepSig RadioML datasets (RADIOML 2016.04C, 2016.10A and 2018.01A) [41] have been widely used for AMC research [4]–[21], [42]. In particular, the FINN-A network [43] representing a state-of-the-art implementation was trained on the RadioML 2018 dataset [41] and achieved an accuracy of 94.1% at 30 dB.

Unfortunately, the DeepSig website [41] itself says that there are several known errata and do not appear to recommend the use of these data. Moreover, issues regarding which parameters were varied, the length of the I/Q inputs, and the mix of modulation types available have been raised [44]. Therefore, we advocate the use of the CSPB.ML.2022 dataset [30] that has addressed these issues.

The CSPB.ML.2022 dataset [30] consists of 4,000 files, each containing 32,768 I/Q samples across eight modulation schemes: BPSK, QPSK, 8PSK, $\pi/4$–DQPSK, 16QAM, 64QAM, 256QAM, and MSK. The physical parameters (symbol period, carrier offset, excess bandwidth, up/down-sampling factors, in-band SNR, and noise spectral density) remain fixed for each eight-file cycle.

C. Versal Architecture

The Versal AI Edge VEK280 board integrates a scalar processing subsystem(central processing unit (CPU)), programmable logic (PL), and a high-density AIE-ML array into a single heterogeneous device. The AIE-ML processor can run up to 1.3 GHz [45]. A single AIE-ML tile supports 512 INT4 operations per cycle at 1 GHz and is equipped with 64 kB of local data memory. Each memory tile has 512 kB of non-blocking local SRAM and supports autonomous DMA and hardware mutual exclusion, enabling data streaming between modules without traversing DRAM. There are up to 304 AIE-ML tiles and 76 memory tiles on the VEK280 board, which are suitable for FAM and lightweight on-chip DNN inference, eliminating transfers between host devices and unnecessary PL transfers [46]. Each AIE-ML tile of the VC2802 device is connected north-south and east-west via a fully bidirectional 512 bit cascade bus. The compute core of the AIE-ML tile is a VLIW SIMD processor with 16 KB of program memory and 64 KB of data memory organized into eight 8 KB banks, twice the native capacity of the original AI Engine (AIE) tile. Each AIE-ML tile achieves up to 128 bfloat16 multiply-accumulate (MAC) operations per cycle [46].

D. Knowledge Distillation

Hinton et al. [26] first formalized knowledge distillation as a model-compression technique that transfers the behavior of a large, accurate Teacher Network to a compact Student Network. Instead of learning only from hard one-hot labels, the student is trained to mimic the soft targets.

The distillation loss is shown in Eq. (6).

$$\mathcal{L} = (1-\alpha)\,\mathcal{L}_{\text{CE}} + \alpha T^2\,\text{KL}\big(p_s(T)\,\|\,p_t(T)\big) \tag{6}$$

where $p_t(T)$ and $p_s(T)$ are the teacher and student logits passed through a softmax divided by a temperature $T > 1$; $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy with ground-truth labels; $\alpha \in [0, 1]$ balances hard-label supervision and imitation of soft targets.

TABLE I: Network Topology of Teacher/Student Network (tuple values represent the parameter for (**Teacher**,**Student**)).

Stage	**Layer type**	**#Filters**	**Kernel / Stride**
Input	–	1	– / –
Stem	Conv + BN + ReLU	(64,24)	$(7,3)\times(7,3)$ / 1
	MaxPool	–	3×3 / 2
Stage 1	$(3,2)\times$BasicBlk	(64,24)	3×3 / 1
Stage 2	$(3,2)\times$BasicBlk	(128,48)	3×3 / 2
Stage 3	$(3,2)\times$BasicBlk	(192,72)	3×3 / 2
Stage 4	$(3,2)\times$BasicBlk	(256,96)	3×3 / 2
Head	GAP	–	1×1 / –
	FC (Soft-max)	8	– / –

III. Algorithmic Optimizations

In this section, we introduce the algorithmic optimizations in Cyclo-AMC. In Section III-A, we describe how to sparsify the original SCD matrix. We then introduce the knowledge distilled DNN architecture in Section III-B.

A. SCD Extraction using Customized FAM

Similarly to reference [28], we adopted the FAM parameter values $N = 512$, $N_P = 64$, $L = N_P/4 = 16$, and $P = N/L = 32$. Calculating the full SCD matrix ($P/2\times N_P^2$ samples) is expensive both in terms of storage and run-time so we adopt a simplified spectral correlation representation that (i) retains modulation-related information, (ii) reduces the amount of data by a factor of $\frac{P}{2}$, and (iii) matches the memory and compute envelope of a lightweight ResNet.

This is done using a subset of the FAM output. Starting from Eq. (5), let $q \in \{-\frac{P}{4}, \ldots, \frac{P}{4} - 1\}$ and $f_k, f_l \in \{-\frac{N_P}{2}, \ldots, \frac{N_P}{2}\}$. The indices (q, k_1, k_2) are mapped to the normalized frequency f and cyclic frequency α via

$$f = \frac{f_k + f_l}{2N_P}, \qquad \alpha = \frac{f_k - f_l}{N_P} + \frac{q}{PL}, \tag{7}$$

with $-0.5 \le f \le 0.5$ and $-1 \le \alpha \le 1$ when $f_s = 1$. Setting $q = 0$ retains the entire power spectral density (PSD) slice ($\alpha = 0$) and a thin band of spectral correlation functions (SCFs) whereby Eq. (7) simplifies to

$$f = \frac{f_k + f_l}{2N_P}, \qquad \alpha = \frac{f_k - f_l}{N_P}. \tag{8}$$

As a result, the original three-dimensional tensor of size $\frac{P}{2}\times(N_P \times N_P)$ collapses to a two-dimensional SCD matrix $S_x^\alpha(f) \in \mathbb{C}^{N_P\times N_P}$, thereby lowering storage and computation costs.

Our customized FAM reduces the final SCD output by a factor of 256 compared to previous implementations [33]–[35], allowing efficient use of the on-chip resources. Section V-B demonstrates that the extracted $\alpha = 0$ slice preserves modulation-related information relevant to classification.

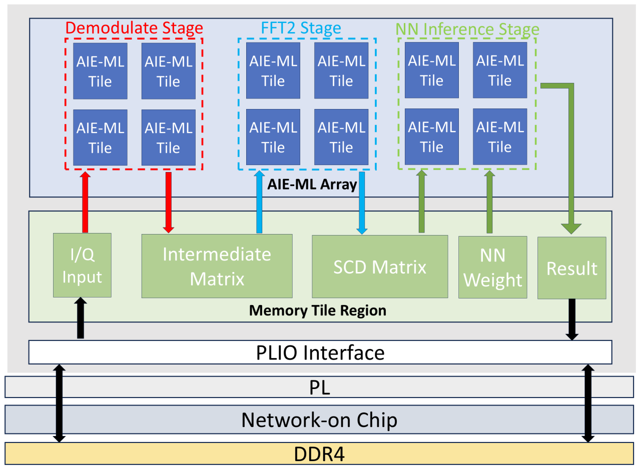

Fig. 2: Cyclo-AMC architecture overview.

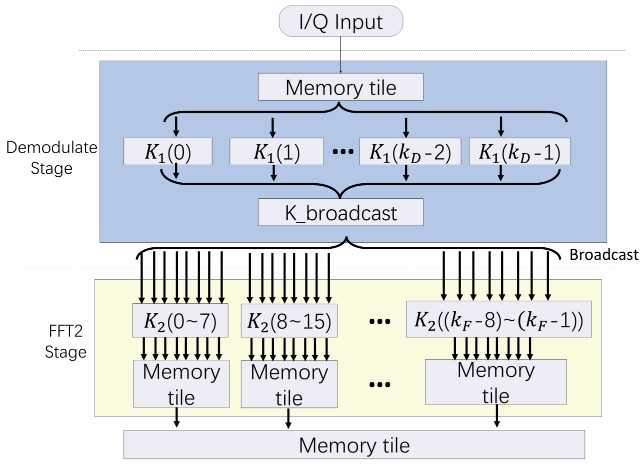

Fig. 3: FAM implementation in AIE-ML.

B. Knowledge Distillation in DNN

1) Teacher–Student Network Pair: Both Teacher and Student are implementations of ResNet, as detailed in Table I. The Teacher, `SCDResNet64`, has 6.40 M parameters and `SCDResNet64_S` has 0.59 M. Both accept single-channel 64×64 spectral correlation slices and share four residual stages. The Teacher Network uses a 7×7 convolutional layer, each followed by three BasicBlk structures, while the Student Network uses a 3×3 convolutional layer, each followed by two BasicBlk structures. A final fully connected layer is used to map eight modulation classes.

Given an input feature map $\mathbf{x} \in \mathbb{R}^{C_{\text{in}}\times H\times W}$, BasicBlk computes

$$\mathbf{y} = \mathrm{ReLU}\Big(\mathrm{BN}_2\big(\mathrm{Conv}_2(\mathrm{ReLU}(\mathrm{BN}_1(\mathrm{Conv}_1(\mathbf{x}))))\big) + \mathrm{Residual}(\mathbf{x})\Big) \quad (9)$$

This module follows the typical dual-convolution ResNet design: 3×3 convolution-batch normalization-ReLU units are stacked twice, where the first convolution adopts stride $s \in \{1, 2\}$ for optional downsampling. When $s > 1$ or the input/output channel widths are different ($C_{\text{in}} \neq C_{\text{out}}$), the skip connection is upgraded from identity to 1×1 convolution-BN projection. The final ReLU activation is the summed feature map. This compact unit serves as the main part in the Teacher and Student Networks.

IV. FPGA Implementation

This section details the implementation of the FAM and neural network inference stages. Section IV-B covers the hardware mapping of the FAM. Section IV-C describes the implementation of the Student Network inference.

A. System Overview

Fig. 2 illustrates how Cyclo-AMC is partitioned across the AIE-ML array. The Demodulate Stage frames the incoming I/Q samples, applies a Hamming window, performs a N_P-point FFT, and executes down-conversion; the resulting matrix is buffered in a memory tile. The FFT2 Stage performs $N_P \times N_P$ times conjugate multiplication and P-point FFT to generate a reshaped SCD matrix. Finally, the Neural Network Inference Stage executes ResNet-based inference on the reshaped SCD matrix, and its final classification result is returned to the host. All data are kept in the memory tiles at the bottom of the AIE-ML array and direct memory access (DMA) is used for transfers between AIE-ML and the memory tiles.

B. FAM Implementation

The end-to-end FAM datapath is divided into two pipeline stages, as illustrated in Fig. 3. Alg. 1 introduces the pseudocode of the FAM part implemented on AIE-ML.

1) Demodulate Stage: The normalised I/Q stream $x(t)$ is first buffered in a memory tile and divided into P overlapping frames of length N_P with stride $L = N_P/4$ to form the decimated matrix $X_{\text{De}}[n, p] = x[pL+n] \in \mathbb{C}^{N_P\times P}$. To fully utilize the six MM2S ports of the tile, X_{De} is split column-wise into k_D sub-matrices $X^{(i)}_{\text{De}} \in \mathbb{C}^{N_P\times P/k_D}$, $i = 0 \dots k_D - 1$, which are streamed to k_D identical `Fam_stage1` tiles (denoted $K_1(0) \sim K_1(k_D - 1)$ in Fig. 3). Each $K_1(i)$ tile operates on its frame subset $\mathcal{J}_i = \{\, j \mid iP/k_D \leq j < (i+1)P/k_D\}$, performing Hamming windowing, a N_P-point FFT, and complex down-conversion, i.e.

$$X^{(i)}_T(f_m) = \big[X_T(jL, f_m)\big]_{j\in\mathcal{J}_i}, \qquad i = 0 \dots k_D - 1.$$

A `K_broadcast` tile then gathers the k_D buffers and reorders them into k_D contiguous P/k_D-column groups. The `K_broadcast` tile forwards the data to the FFT2 Stage, facilitating data delivery using its stream port by issuing two broadcasts to all `Fam_stage2` tiles:

- **Broadcast A (data delivery).** The `K_broadcast` tile streams to each `Fam_stage2` tile a block of N_P/k_F $\mathbf{X}_T(rL, f_{k_j})$ vectors.
- **Broadcast B (conjugate term delivery).** Broadcasts a conjugate vector $\mathbf{X}^*_T(rL, f_l)$ to all `Fam_stage2` tiles for multiplication with the inputs from Broadcast A.

Algorithm 1: AIE-ML-based FAM pipeline pseudocode.

```
Function Demodulate(iq_input):
    $X \leftarrow$ zeros($N_P, P$);
    for $k = 0$ to $P-1$ do
        $X(:, k+1) \leftarrow$ iq_input[$k \cdot L : k \cdot L + N_P - 1$];
    $Y \leftarrow$ zeros($N_P, P$);
    for $k = 0$ to $P-1$ do
        $data\_win \leftarrow$ Window($hamming[N_P], X(:, k)$);
        $data\_fft \leftarrow N_P$-Point FFT($data\_win$);
        $Y(:, k) \leftarrow$ Down_Conversion($data\_fft$, $k$);
    return $Y$;
Function FFT2(Y):
    for $m = 0$ to $N_P - 1$ do
        for $n = 0$ to $N_P - 1$ do
            $z_{abs} \leftarrow$
              $|P$-point FFT $(Y(:, m) \cdot \text{conj}(Y(:, n)))|^2$;
            write_to_output($z_{abs}[\frac{P}{4}]$);
```

Because AIE-ML tiles support multiple input buffer interfaces, only one `K_broadcast` tile is needed to aggregate the output of k_D `Fam_stage1` tiles [46]. We note that this scheme cannot be used in the AIE tile as it can only receive from two input interfaces [47].

2) FFT2 Stage: The FFT2 Stage performs conjugate multiplications followed by P-point FFTs. For the generalized setting with N_P frequency bins, we deploy k_F `Fam_stage2` tiles, denoted $K_2(0) \sim K_2(k_F-1)$, each operating on N_P/k_F adjacent bins:

$$f_{k_j} = \frac{N_P}{k_F} i + j, \quad j = 0, \ldots, \frac{N_P}{k_F} - 1, \quad i = 0, \ldots, k_F - 1.$$

Each $K_2(i)$ performs N_P^2/k_F times conjugate multiplications between the $\mathbf{X}_T(rL, f_{k_j})$ and the $\mathbf{X}_T^*(rL, f_l)$ (from the `K_broadcast` tile), followed by a P-point FFT to compute the corresponding slice of the SCD matrix.

As shown in Fig. 3, the k_F output streams are gathered by $k_F/8$ memory tiles (8 streams per tile) and merged into a single memory tile that stores the SCD matrix for the NN inference stage.

C. ResNet Implementation

In Cyclo-AMC, the entire neural network inference pipeline is embedded within the AIE-ML array. The network weights are streamed into the compute tiles through multiple programmable logic input/output (PLIO) interfaces, while the feature maps produced by each layer are retained in the memory tiles, thus reducing PLIO traffic. Every convolutional layer is statically fused with its subsequent batch normalization layer to minimize inference latency in our AIE-ML deployment [48].

Fig. 4 describes the mapping on-chip of the Cyclo-AMC inference pipeline in the AIE-ML array. Each colored dashed box contains a group of AIE-ML tiles assigned to one network stage, while the blue blocks labeled Mem Tile represent memory tiles that cache the feature maps produced by the preceding layer. The weights for every layer are streamed from the PL through dedicated PLIO interfaces (green arrows), allowing the compute tiles to operate without off-chip traffic. The input SCD matrix enters the AIE-ML array in stem and is forwarded layer by layer entirely within the AIE-ML array; only the weight input and the final classification output cross the PL interface.

1) Vectorised 3×3 Convolution with `mac_4x8_8x4()`: Xilinx AIE-ML application-level programming tools include the `mac_4x8_8x4()`, intrinsic which performs the matrix product: $\mathbf{M}_{4\times4} = \mathbf{X}_{4\times8}\mathbf{Y}_{8\times4}$, in a single clock cycle, yielding $4 \times 4 = 16$ MAC results from 32 input pixels and 32 weights. We exploit this capability to realize a 3×3 convolution as follows:

1) **Input blocking.** For four contiguous spatial positions $\{p, p+1, p+2, p+3\}$ in row r, we gather the corresponding pixels from eight input channels into $\mathbf{X}_{4\times8}$, each column contains one row of pixels, while the rows are mapped onto input channels and vector lanes.
2) **Weight blocking.** The 8×4 matrix $\mathbf{Y}$ stores kernel weights that connect the same eight input channels to four output channels for a single kernel tap (k_h, k_w).
3) **Nine-tap accumulation.** Repeating Step 1 and Step 2 for the nine kernel offsets $(k_h, k_w) \in \{0, 1, 2\}^2$ and summing the intermediate `mac_4x8_8x4()` outputs in a local accumulator produces the complete 3×3 convolution over the 4×4 output tile.
4) **Write back.** After all the taps are processed, all 16 results are streamed to the output feature map.

This scheme sustains the intrinsic's peak throughput (128 MAC/cycle) provided the DMA engines keep the 4×8 pixel stripes and 8×4 weight blocks double-buffered.

Alg. 2 shows how a single AIE-ML tile performs a 3×3 Conv2D using the intrinsic `mac_4x8_8x4()`.

2) Mapping of a single BasicBlk: Table I reveal that the majority of both Teacher and Student Networks are composed of BasicBlk units. Consequently, devising an efficient parallelization strategy for these blocks is important to accelerating inference. Packing the entire computation into a single AIE-ML tile is impractical for the following reasons:

1) **Insufficient on-tile memory.** Even for modest feature map sizes, the intermediate tensor produced by Conv1 cannot be stored in the 64 kB data memory of a tile [46].
2) **Bandwidth contention between main and residual paths.** After Conv1, the tile would need to fetch a second kernel (Conv2) and simultaneously stream the original input for bypass. The single stream ports cannot sustain the two disjoint dataflows at line rate.

For these reasons, we adopt the BasicBlk shown in the right half of Fig. 4. Conv1, the residual projection, and Conv2 are assigned to an independent group of AIE-ML tiles, while three memory tiles (Memory Tile 1–3) retain the intermediate feature maps locally. Alg. 3 shows how we implement the BasicBlk pipeline in AIE-ML by creating a tile cluster consisting of 9 AIE-ML tiles. Eq. (9) requires two 3×3 convolutions,

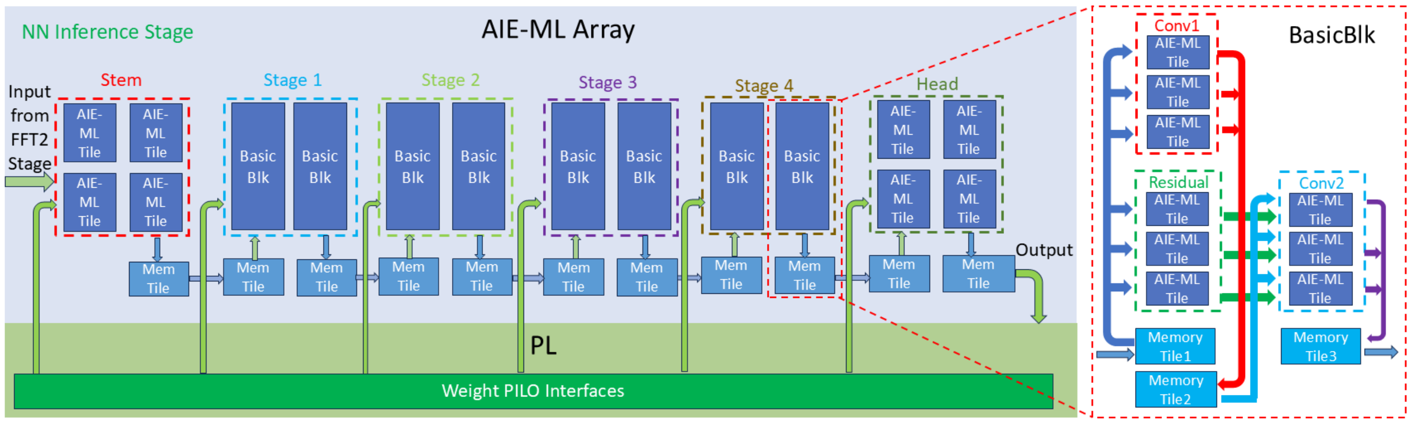

Fig. 4: Neural network implementation in AIE-ML.

Algorithm 2: AIE-ML 3×3 Conv2D Pipeline

```
Function ConvTile_3x3($X_{4\times8}$, $W_{8\times4}[3][3]$):
    $\mathbf{ACC}_{4\times4} \leftarrow \mathbf{0}$;
    for $k_h = 0$ to 2 do
        for $k_w = 0$ to 2 do
            $\mathbf{Y} \leftarrow W_{8\times4}[k_h][k_w]$;
            $\mathbf{ACC}_{4\times4}$ += mac_4x8_8x4( $X_{4\times8}$, $\mathbf{Y}$ );
    return ACC;
Function
ConvMap_3x3($x[C_I][H_I][W_I], w[C_O][C_I][3][3]$):
    for $o_0 = 0$ to $C_O - 1$ step 4 do
        for $r = 0$ to $H_O - 1$ do
            for $p = 0$ to $W_O - 1$ step 4 do
                for $c_0 = 0$ to $C_I - 1$ step 8 do
                    $X \leftarrow x[c_0 : c_0{+}7][r : r{+}2][p : p{+}3]$;
                    W_b $\leftarrow w[o_0 : o_0 + 3][c_0 :$
                      $c_0 + 7][*][*]$;
                    OUT += ConvTile_3x3(X, W_b);
                $y[o_0 : o_0{+}3][r][p : p{+}3] \leftarrow \mathrm{ReLU}(\mathbf{OUT})$;
    return y;
```

two batch normalization layers, two ReLU activations, and a residual bypass that must be added element-wise to the main path.

Since each memory tile provides only 6 MM2S ports [46]. To broadcast an input feature map to both the Conv1 and the residual kernels, we allocate 3 MM2S ports to them, respectively. Consequently, the number of output channels per BasicBlk is chosen to be a multiple of 24. This value is divisible by 3, enabling an even workload distribution across the three Conv1 (or Residual) tiles, and also divisible by 8, allowing each tile to utilize `mac_4x8_8x4()` to achieve peak vector load rate.

V. Results

We deploy the FAM and neural network designs described in Sec. III on the VEK280 platform, where the AIE-ML array operates at 1 GHz and the PL at 625 MHz. For reference, we

Algorithm 3: BasicBlk pipeline with 9 AIE-ML tiles

```
Function BasicBlk_Cluster(x):
    $f_{in} \leftarrow$ MemTile1.write();
    parallel for $g = 0$ to 2
      $\{f_g \leftarrow \mathrm{ReLU}\big(\texttt{Conv3x3}(f_{in},\, W_g^{(1)})\big)$;
    $r_g \leftarrow \texttt{Conv3x3}(f_{in},\, W_g^{(\mathrm{res})})\}$;
    MemTile2 $\leftarrow \{f_0, f_1, f_2\}$;
    $f_{in1} \leftarrow$ MemTile2.write();
    parallel for $g = 0$ to 2
      $\{y_g \leftarrow \mathrm{ReLU}\big(\texttt{Conv3x3}(f_{in1},\, W_g^{(2)}) + r_g\big)\}$;
    MemTile3 $\leftarrow \{y_0, y_1, y_2\}$;
    return MemTile3.write();
```

benchmark an x86 server with an Intel® Core i9-9900KF CPU and an NVIDIA GeForce RTX 3090 GPU, running at 3.60 GHz and 2.10 GHz under Ubuntu 22.04.4 LTS respectively.

A. Experimental Setup

1) Data Preparation: The Student Network is trained offline in PyTorch using the CSPB.ML.2022 dataset [30]. The features were extracted from the I/Q data using FAM to produce the (64×64) spectral correlation slices corresponding to Eq. (8). We use 20 batches (01–10 and 19–28) to form the training set, 4 batches (11–14) for the validation set and 4 batches (15–18) for the test set, yielding $N_{\text{train}} = 80,000$, $N_{\text{val}} = 16,000$ and $N_{\text{test}} = 16,000$ images. A global Z-score normalization [49] is applied to each SCD matrix: $\hat{x} = x\gamma + \beta$ with $\gamma = 6.93232 \times 10^{-5}$ and $\beta = -0.296705$.

2) Teacher Network training: We use stochastic gradient descent (SGD) (lr=0.1, momentum=0.9, weight-decay=10^{-4}) with a linear warm-up of 5 epochs, then cosine annealing to 10^{-8} over 500 epochs [50]. Cross-entropy loss includes 5 % label smoothing.

3) Knowledge Distillation Settings: The fixed teacher provides soft targets [26]. The student is optimized with SGD (lr=0.05, weight-decay=2×10^{-4}) under the same warm-up/cosine schedule. The settings in Eq. (6) are temperature $T = 4$ and mixing factor $\gamma = 0.7$ [51].

TABLE II: Execution time and resource usage ($k_D = 4$)

k_F	8	16	32	64
AIE-ML Tiles	13	21	37	69
Memory Tiles	4	6	6	10
Execution Time (ms)	0.668	0.352	0.186	0.109

TABLE III: Classification Accuracy on CSPB.ML.2022

Model	**Params (M)**	**Accuracy (%)**
Teacher Network (Ours)	6.40	97.2
Student Network (Ours)	0.59	93.8
Tiny Student Network (Ours)	0.15	83.6
FINN-A (Retrained) [43]	0.16	81.7
CC-trained CAP [52]	1.27	92.5

4) FAM Implementation Settings: In Section IV-B, we introduce the parameters k_D and k_F that affect the parallelism in the FAM pipeline. Since the MAC operations in the Demodulate Stage are significantly lower than in the FFT2 Stage, we allocate more AIE-ML tiles to the FFT2 Stage by setting $k_D = 4$. Given $N_P = 64$, the value of k_F must be a power of two no greater than 64. Table II compares execution time and resource usage for different k_F values. Empirically, we found that $k_F = 16$ achieves the best trade-off.

B. Accuracy

Table III summarizes the overall classification accuracy and model size of the networks implemented in PyTorch. Our three design models and FINN-A[2] were all retrained and then evaluated on the CSPB.ML.2022 dataset using the same data splits, i.e. FINN-A and our networks were trained and tested using identical data and methodologies. Our Student Network achieves an overall precision of 93.8% with only 0.59 M parameters: a $11\times$ reduction in parameters compared to the Teacher Network, while incurring a 3.4% drop in accuracy. Compared to FINN-A [43], our Student Network achieves a 12.1% improvement in accuracy with a modest increase in parameters. To ensure a fair comparison, we also developed a Tiny Student Network with only 0.15 M parameters - comparable to FINN-A's 0.16 M - but it still delivers a 1.9% gain in accuracy. Although the CC-trained CAP method [52] uses more parameters, it still lags behind our Student Network.

The Teacher's soft outputs show how different classes relate, helping the Student learn smoother decision boundaries and avoid overfitting to hard labels. This regularization lets the smaller model retain useful features and is more robust to noise and interference. To gain deeper insights into class-wise performance, Fig. 5 presents the normalized confusion matrices of both networks. The upper and lower parts of each cell are the classification results of the Student and Teacher Network implemented in PyTorch, respectively.

As illustrated in Fig. 5, BPSK records the lowest accuracy in both models (Student: 92.2%, Teacher: 96.0%), whereas MSK

[2]The source code for FINN-A is available at https://github.com/Xilinx/brevitas-radioml-challenge-21.

TABLE IV: Comparison of FINN-A and Cyclo-AMC Implementations.

Metric	**FINN-A [43]**	**Cyclo-AMC (Ours)**
Network	VGG10	ResNet
Parameters (M)	0.16	0.59
Accuracy (%)	81.7	93.8
Latency (μs)	11.7	1803

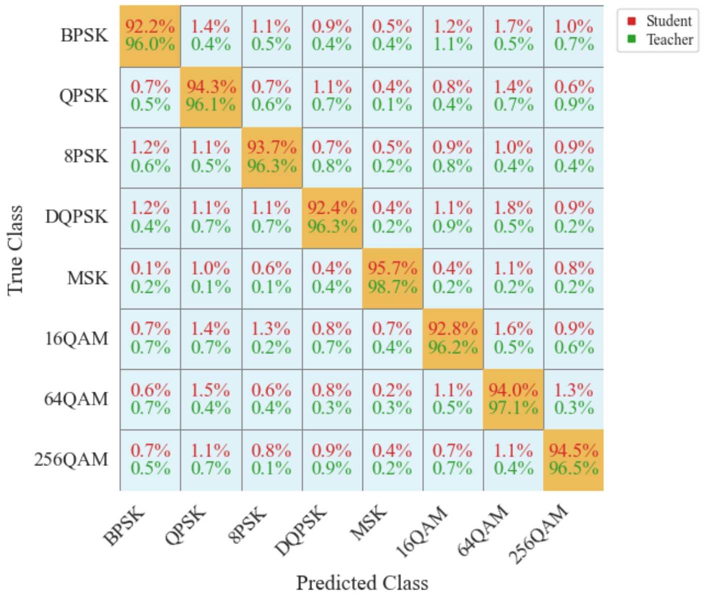

Fig. 5: Confusion matrix for Teacher and Student Networks.

attains the highest (Student: 95.7%, Teacher: 98.7%). The fact that this max-min pattern is preserved after training confirms that knowledge distillation enables the Teacher Network to transfer the difficulty of distinguishing modulation categories to the Student Network.

As shown in Table IV, we further compare Cyclo-AMC with FINN-A. FINN-A, based on a VGG10 network with 0.16 M parameters, achieves 81.7% accuracy on CSPB.ML.2022 with a latency of 11.7 μs. In contrast, Cyclo-AMC employs a larger ResNet-based student network with 0.59 M parameters, reaching 93.8% accuracy on CSPB.ML.2022 but with a longer latency of 1803 μs.

C. Utilization

Table V presents the utilization, latency, power composition, and energy efficiency of the proposed FAM and Student Network implementations on the AIE-ML array. We separately report FAM and Student Network utilization from `vitis_analyzer` to facilitate comparisons with prior work.

We measured power consumption using Xilinx Power Design Manager [53]. The Student Network consumes a total of 12.38 W, higher than FAM (2.45 W) due to deeper layers and increased I/O. Despite this, it achieves excellent energy efficiency (6.371 GFLOPs/W), slightly higher than FAM (5.854 GFLOPs/W).

TABLE V: Utilization comparison of FAM and Student Network implementations on AIE-ML

AIE-ML Implementations	Total AIE-ML Tiles	Memory Tiles	PLIOs	Latency (ms)	Throughput (GFLOPs)	AIE core P. (W)	Memory P. (W)	Power (W)	Energy (mJ)	Energy Eff. (GFLOPs/W)
FAM	21 (6.9%)	5 (6.6%)	2 (1.0%)	0.35	14.36	1.993	0.460	2.453	0.859	5.854
Student Network	79 (26.0%)	20 (26.3%)	79 (40.3%)	1.45	78.91	8.282	4.102	12.384	17.957	6.371
Total	100 (32.9%)	25 (32.9%)	81 (41.3%)	1.80	66.30	7.059	3.394	10.453	18.816	6.343

TABLE VI: End-to-End Inference Performance and Efficiency: AIE-ML vs. CPU and GPU Platforms

Platform	Inference Latency (ms)								Speedup	Throughput	Power	Energy Eff.
	FAM	Stem	Stage 1	Stage 2	Stage 3	Stage 4	Head	Total		(GFLOPs)	P. (W)	(GFLOPs/W)
CPU (i9-9900KF)	6.40	0.79	2.14	1.72	1.03	0.91	0.23	13.24	1	9.03	14	0.65
GPU (RTX 3090)	2.95	0.51	0.56	0.72	0.71	0.70	0.23	6.38	2.08	18.75	95	0.20
AIE-ML	0.35	0.14	0.60	0.39	0.23	0.08	0.01	1.80	7.36	66.30	10	6.34

TABLE VII: Comparison of the FAM in Cyclo-AMC with Prior FAM Implementations.

Metric	[33]	[34]	[35]	Cyclo-AMC
Platform	ZCU111	ZCU111	VCK5000	VEK280
AIE/AIE-ML tiles	—	—	137	21
Initiation Interval (ms)	0.26	0.164	0.63	0.35
Throughput (MS/s)	7.88	12.50	3.25	1.46
Computational Performance (GOPS)	60.40	460	189	14.36
Power (W)	12.50^{+}	35*	40*	2.453^{+}

* Full board power.
$^{+}$ Chip-only power.

Compared to FAM part, the Student Network uses more AIE-ML tiles, PLIOs, and memory tiles. FAM occupies only 21 AIE-ML tiles, enabling lightweight feature extraction with low memory usage and 0.35 ms latency. In contrast, the Student Network uses 79 tiles to achieve 78.91 GFLOPs throughput. As shown in Table V, FAM consumes only 0.859 mJ (4.6% of total energy), while the Student Network consumes 17.957 mJ (95.4%). Our customized FAM reduces computational complexity and power, while parallelization of the Student Network maintains high throughput.

D. Performance

We implemented both the CPU and GPU versions of Cyclo-AMC using PyTorch 2.5.1 with CUDA 12.1 support. Power consumption during inference was measured using a dedicated power station for the CPU and the `nvidia-smi` command-line tool for the GPU. Table VI compares end-to-end inference performance with Cyclo-AMC achieving a 7.36× speedup on CPU and 3.54× over GPU.

While GPU offers moderate throughput (18.75 GFLOPs) at the cost of high power consumption (95 W), Cyclo-AMC delivers a throughput of 66.30 GFLOPs with only 10 W of power, resulting in an energy efficiency of 6.34 GFLOPs/W. This represents almost 31.7× better energy efficiency than GPU and 9.8× better than the CPU.

Table VII presents a comparison between our implementation and prior FPGA-based designs. Prior FPGA-based works were based on the parameter configuration of $N = 2,048$, $N_P = 256$, $L = 64$, and $P = 32$. In contrast, Cyclo-AMC operates with a smaller configuration of $N = 512$, $N_P = 64$, $L = 16$, and $P = 32$. The most comparable work is [35], which employs 137 AIE tiles, while Cyclo-AMC implements the FAM using only 21 AIE-ML tiles.

The Cyclo-AMC implementation attains 66.30 GFLOPs on the VEK280. This corresponds to merely 0.26% of the approximately 25.6 TFLOPs peak achievable with 100 AIE-ML tiles operating at 1 GHz [46]. The low utilization arises because most of the AIE-ML tiles in Cyclo-AMC spend most of their time on data movement rather than on arithmetic operations, thereby limiting the arithmetic intensity delivered to the compute units. On RTX 3090, Cyclo-AMC reaches 18.75 GFLOPs, that is, only 0.05% of the theoretical GPU FP32 peak of 35.5 TFLOPs. The low operational intensity keeps the kernel below the compute roof, so the performance is limited by memory bandwidth. Consequently, both AIE-ML and GPU implementations are bandwidth-bound rather than compute-bound.

VI. Conclusion

We presented Cyclo-AMC, the first FPGA-based AMC system that integrates cyclostationary feature extraction with a compact DNN classifier entirely on the Versal AIE-ML platform. By combining FAM with ResNet, Cyclo-AMC achieves 93.8% classification accuracy on the CSPB.ML.2022 dataset while using only 0.59 M parameters. The system delivers 66.30 GFLOPs at 10 W, achieving 6.34 GFLOPs/W, a 31.7x improvement in energy efficiency over a high-end GPU, and 7.4x faster inference compared to a CPU. The proposed architecture lays the foundation for future integration with software-defined radio front ends to produce a fully embedded AMC system. Future work will investigate how complementary software-level techniques such as ensembles can be used to further enhance the accuracy of Cyclo-AMC.

References

[1] B. Wang and K. R. Liu, "Advances in cognitive radio networks: A survey," *IEEE Journal of Selected Topics in Signal Processing*, vol. 5, no. 1, pp. 5–23, 2011.

[2] T. O'Shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 4, pp. 563–575, 2017.

[3] A. Parmar, A. Chouhan, K. Captain, and J. Patel, "Deep multilevel architecture for automatic modulation classification," *Physical Communication*, vol. 64, p. 102361, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S187449072400079X

[4] Y. Dong, Y. Peng, M. Yang, S. Lu, and Q. Shi, "Signal transformer: Complex-valued attention and meta-learning for signal recognition," 06 2021.

[5] Y. TU, Y. LIN, H. ZHA, J. ZHANG, Y. WANG, G. GUI, and S. MAO, "Large-scale real-world radio signal recognition with deep learning," *Chinese Journal of Aeronautics*, vol. 35, no. 9, pp. 35–48, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1000936121002934

[6] H. Zhang, F. Zhou, Q. Wu, and C. Yuen, "FSOS-AMC: Few-shot open-set learning for automatic modulation classification over multipath fading channels," *IEEE Internet of Things Journal*, vol. 12, no. 12, pp. 18 718–18 731, 2025.

[7] X. Kang, H. mei Chen, G. Chen, K.-C. Chang, and T. M. Clemons, "Joint detection and classification of communication and radar signals in congested RF environments using YOLOv8," 2024. [Online]. Available: https://arxiv.org/abs/2406.00582

[8] J. Krzyston, R. Bhattacharjea, and A. Stark, "High-capacity complex convolutional neural networks for I/Q modulation classification," 2020. [Online]. Available: https://arxiv.org/abs/2010.10717

[9] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," 2016. [Online]. Available: https://arxiv.org/abs/1602.04105

[10] T. J. O'Shea, L. Pemula, D. Batra, and T. C. Clancy, "Radio transformer networks: Attention models for learning to synchronize in wireless systems," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, 2016, pp. 662–666.

[11] A. Faysal, M. Rostami, T. Boushine, R. G. Roshan, H. Wang, and N. Muralidhar, "DenoMAE2.0: Improving denoising masked autoencoders by classifying local patches," 2025. [Online]. Available: https://arxiv.org/abs/2502.18202

[12] D. Zhang, Y. Lu, Y. Li, W. Ding, B. Zhang, and J. Xiao, "Frequency learning attention networks based on deep learning for automatic modulation classification in wireless communication," *Pattern Recognition*, vol. 137, p. 109345, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320323000468

[13] C. Harper, M. Thornton, and E. Larson, "Automatic modulation classification with deep neural networks," 2023. [Online]. Available: https://arxiv.org/abs/2301.11773

[14] J. Cai, F. Gan, X. Cao, and W. Liu, "Signal modulation classification based on the transformer network," *IEEE Transactions on Cognitive Communications and Networking*, vol. 8, no. 3, pp. 1348–1357, 2022.

[15] X. Wu, S. Wei, and Y. Zhou, "Deep multi-scale representation learning with attention for automatic modulation classification," 2022. [Online]. Available: https://arxiv.org/abs/2209.03764

[16] Y. Chen, "Automatic modulation recognition algorithm based on phase transformation and ResCNN-BiLSTM," *Advances in Engineering Innovation*, vol. 16, pp. None–None, 04 2025.

[17] M. R. Ziemann and C. A. Metzler, "Adaptive LPD radar waveform design with generative deep learning," *IEEE Transactions on Radar Systems*, vol. 3, p. 417–429, 2025. [Online]. Available: http://dx.doi.org/10.1109/TRS.2025.3542283

[18] H. Xing, X. Zhang, S. Chang, J. Ren, Z. Zhang, J. Xu, and S. Cui, "Joint signal detection and automatic modulation classification via deep learning," *IEEE Transactions on Wireless Communications*, vol. 23, no. 11, pp. 17 129–17 142, 2024.

[19] A. Owfi, A. Abbasi, F. Afghah, J. Ashdown, and K. Turck, "Dynamic online modulation recognition using incremental learning," 2023. [Online]. Available: https://arxiv.org/abs/2312.04718

[20] V. Clerico, J. González-López, G. Agam, and J. Grajal, "LSTM framework for classification of radar and communications signals," in *2023 IEEE Radar Conference (RadarConf23)*, 2023, pp. 1–6.

[21] H. Zhang, L. Yuan, G. Wu, F. Zhou, and Q. Wu, "Automatic modulation classification using involution enabled residual networks," *IEEE Wireless Communications Letters*, vol. 10, no. 11, pp. 2417–2420, 2021.

[22] J. A. Snoap, D. C. Popescu, and C. M. Spooner, "Deep-learning-based classifier with custom feature-extraction layers for digitally modulated signals," *IEEE Transactions on Broadcasting*, vol. 70, no. 3, pp. 763–773, 2024.

[23] D. de Oliveira Rubiano, G. Korol, and A. C. S. Beck, "Adaptive inference for FPGA-Based 5G automatic modulation classification," in *Design and Architecture for Signal and Image Processing*, M. Chavarrías and A. Rodríguez, Eds. Cham: Springer Nature Switzerland, 2023, pp. 95–106.

[24] G. Vasiliadis, L. Koromilas, M. Polychronakis, and S. Ioannidis, "Design and implementation of a stateful network packet processing framework for GPUs," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 610–623, 2017.

[25] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google neural network models for edge devices: Analyzing and mitigating machine learning inference bottlenecks," 2021. [Online]. Available: https://arxiv.org/abs/2109.14320

[26] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[27] R. S. Roberts, W. A. Brown, and H. H. Loomis, "Computationally efficient algorithms for cyclic spectral analysis," *IEEE Signal Processing Magazine*, vol. 8, no. 2, pp. 38–49, 1991.

[28] W. A. Brown and H. H. Loomis, "Digital implementations of spectral correlation analyzers," *IEEE Transactions on Signal Processing*, vol. 41, no. 2, pp. 703–720, 1993.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[30] C. Spooner, "Data Sets for the Machine Learning Challenge," 2022, accessed: 16 May 2022. [Online]. Available: https://cyclostationary.blog/data-sets/

[31] W. A. Gardner, A. Napolitano, and L. Paura, "Cyclostationarity: Half a century of research," *Signal Processing*, vol. 86, no. 4, pp. 639–697, 2006.

[32] W. A. Gardner, *Cyclostationarity in communications and signal processing*. New York: IEEE Press, 1994.

[33] X. Li, D. L. Maskell, C. J. Li, P. H. W. Leong, and D. Boland, "A scalable systolic accelerator for estimation of the spectral correlation density function and its FPGA implementation," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3546181

[34] C. J. Li, X. Li, B. Lou, C. T. Jin, D. Boland, and P. H. W. Leong, "Fixed-point FPGA implementation of the FFT accumulation method for real-time cyclostationary analysis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, Jun. 2023. [Online]. Available: https://doi.org/10.1145/3567429

[35] C. J. Li, R. Wu, and P. H. W. Leong, "AMD Versal implementations of FAM and SSCA estimators," 2025. [Online]. Available: https://arxiv.org/abs/2506.18003

[36] T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 168–179, 2018.

[37] J. A. Latshaw, D. C. Popescu, J. A. Snoap, and C. M. Spooner, "Using capsule networks to classify digitally modulated signals with raw I/Q data," in *2022 14th International Conference on Communications (COMM)*. IEEE, Jun. 2022, p. 1–6. [Online]. Available: http://dx.doi.org/10.1109/COMM54429.2022.9817229

[38] J. A. Snoap, D. C. Popescu, and C. M. Spooner, "On deep learning classification of digitally modulated signals using raw I/Q data," in *2022 IEEE 19th Annual Consumer Communications &amp; Networking Conference (CCNC)*. IEEE, Jan. 2022, p. 441–444. [Online]. Available: http://dx.doi.org/10.1109/CCNC49033.2022.9700688

[39] J. A. Snoap, J. A. Latshaw, D. C. Popescu, and C. M. Spooner, "Robust classification of digitally modulated signals using capsule networks and cyclic cumulant features," in *MILCOM 2022 - 2022 IEEE Military Communications Conference (MILCOM)*, 2022, pp. 298–303.

[40] N. Bidyanta, G. Vanhoy, M. Hirzallah, A. Akoglu, and B. Ryu, "GPU and FPGA based architecture design for real-time signal classification," in *Proceedings of the Wireless Innovation Forum Conference on Wireless Communications Technologies and Software Defined Radio (WInnComm'15)*, San Diego, CA, USA, Mar. 24–26 2015, pp. 70–79.

[41] DeepSig, “Datasets,” 2018, accessed: May 16, 2022. [Online]. Available: https://www.deepsig.ai/datasets/

[42] D. Hou, L. Li, W. Lin, J. Liang, and Z. Han, “ClST: A convolutional transformer framework for automatic modulation recognition by knowledge distillation,” *IEEE Transactions on Wireless Communications*, vol. 23, no. 7, pp. 8013–8028, 2024.

[43] F. Jentzsch, Y. Umuroglu, A. Pappalardo, M. Blott, and M. Platzner, “Radioml meets FINN: Enabling future RF applications with FPGA streaming architectures,” *IEEE Micro*, vol. 42, no. 6, pp. 125–133, 2022.

[44] C. Spooner, “DeepSig’s 2018 Dataset,” 2020, accessed: May 16, 2022. [Online]. Available: https://cyclostationary.blog/2020/09/24/deepsigs-2018-data-set-2018-01-osc-0001_1024x2m-h5-tar-gz/

[45] AMD Xilinx Inc., “AMD AI Engine Technology,” 2025, accessed: Jul 28 2025. [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/ai-engine.html

[46] AMD Xilinx Inc, *Versal Adaptive SoC AIE-ML Architecture Manual (AM020)*, 1st ed., May 2024, accessed: May 14, 2025. [Online]. Available: https://docs.amd.com/r/en-US/am020-versal-aie-ml

[47] ——, *Versal Adaptive SoC AI Engine Architecture Manual (AM009)*, 1st ed., May 2023, accessed: May 14, 2025. [Online]. Available: https://docs.amd.com/r/en-US/am009-versal-ai-engine

[48] S. Ioffe and C. Szegedy, “Batch normalization: accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML’15. JMLR.org, 2015, p. 448–456.

[49] K. Pearson, “LIII. on lines and planes of closest fit to systems of points in space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[50] H. Robbins and S. Monro, “A Stochastic Approximation Method,” *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729586

[51] Y. Matsubara, “torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation,” in *International Workshop on Reproducible Research in Pattern Recognition*. Springer, 2021, pp. 24–44.

[52] J. A. Snoap, D. C. Popescu, J. A. Latshaw, and C. M. Spooner, “Deep-learning-based classification of digitally modulated signals using capsule networks and cyclic cumulants,” *Sensors*, vol. 23, no. 12, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/12/5735

[53] AMD Xilinx Inc, “Power Design Manager (PDM),” 2024, accessed: Oct. 2024. [Online]. Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/power-design-manager.html