Envisioning Physical Layer Flexibility Through the Power of Machine-Learning

Michael Petry
Telecom Processing Germany
Airbus Defence and Space GmbH
Munich, Germany
michael.petry@airbus.com

Andreas Koch
Telecom Processing Germany
Airbus Defence and Space GmbH
Munich, Germany
andreas.c.koch@airbus.com

Martin Werner
Professorship of Big Geospatial Data Management
Technical University of Munich
Munich, Germany
martin.werner@tum.de

***Abstract*—This paper presents the vision of an adaptive radio frequency (RF) communication signal processing pipeline solely composed of machine learning domain operations, aiming to provide a fully hardware-accelerated alternative to dedicated RF chips. A hybrid architecture, comprising elements of classic signal processing and learnable algorithms trained in an end-to-end manner, is proposed, that is compatible with contemporary ML hardware accelerators. The RF-ML pipeline, including speed-up optimization modifications, are explained in detail, followed by a brief description summarizing the deployment workflow of the end-to-end system on a pair of AI-enabled space-grade FPGAs. Finally, a bit-error-rate performance study of the simulated system as well as a HW-deployed setup including software-defined radios (SDR) validates the concept, followed by a detailed throughput benchmark over multiple AI-accelerator hardware configurations. Finally, we raise questions regarding practical implementation, such as receiver synchronization, restrictions of the ML-accelerator feature space, and weight quantization, which are discussed at the end of this paper.**

***Index Terms*—signal processing, physical layer, machine learning, hardware acceleration, fpga, software-defined radio**

I. Introduction

Today's requirements on communication systems might not be the same as tomorrow's. This is best understood by the example of the 5G NR mobile communication standard. Starting from its initial version (Rel. 15) in 2017, new iterations are continuously passed that comprise changes/extensions to the physical layer. With Release 18 expected to pass at the end of 2023, it is clear, that at this rate of updates the broad majority of users cannot keep up with state-of-the-art equipment, hence, being "outdated" has become the new normal. This lack of technological advancement in the communications field can be traced back to hardware, specifically to specialized RF-chips or -ASICs not being upgradable w.r.t. physical layer modifications, making devices obsolete on a yearly basis. This conflicts heavily with the idea of sustainability. We envision to change this by combining the idea of a re-programmable physical layer hardware with the current mega-trend of machine learning and its available hardware accelerators. By not only exploiting the idea of augmenting the RF-pipeline with AI-algorithms, but also using the available AI-hardware accelerators for power-efficient and high-speed computation, we propose a system design that can combines various potential RF performance gains over traditional systems as shown in [1]–[4] with physical layer flexibility and adaptability unknown to compact mobile devices.

This paper builds on the idea of a learned communications system as proposed by O'Shea et al. [1] in 2017, which sparked the interest of applying machine learning in the RF communications field. Follow-up works primarily focused on extending the system's capabilities, e.g., bit-wise autoencoder [3], multiple antenna system [5], multi-carrier modulation schemes such as OFDM [6], neural equalization [7], graph neural network-based channel decoding [8], etc. However, all systems were either proposed in simulation or realized on impractical GPU-accelerated systems to fulfil computational demands, leaving the idea of AI-based communication practically infeasible for resource restricted platforms. The main contributions of this paper are the following: We aim at closing the gap between computer simulations and practical deployment by realizing a practically feasible non-GPU based hardware setup that builds on a power-efficient FPGA-based AI accelerator, but can be generalized to any compatible AI accelerator. Furthermore, we demonstrate continuous communication with our autoencoder implementation.

The rest of this paper is structured as follows: Section II details the designed RF-ML pipeline including the end-to-end training strategy for its learnable components. Section III briefly introduces the AI-optimized FPGA platform and its workflow including necessary modifications for transforming the generic NN model to a hardware compatible form. Validation of the proposed system, including a detailed BER- and throughput-performance benchmark, by using software-defined radios as RF-frontend to the FPGA platform is detailed in Section IV. Lastly, Section V concludes this work and discusses discovered challenges for future work.

II. System model

This work assumes a single-input single-output (SISO) system model with single-carrier modulation. The objective of the system is to transmit a *continuous* stream of binary data unidirectional from the sender to the receiver through a RF communications channel. This involves performing common signal

This work is supported by the German Aerospace Center (DLR) under project Machine Learning on Telecommunications Satellites (MaLeTeSa) with Grant number 50 YB 2103.

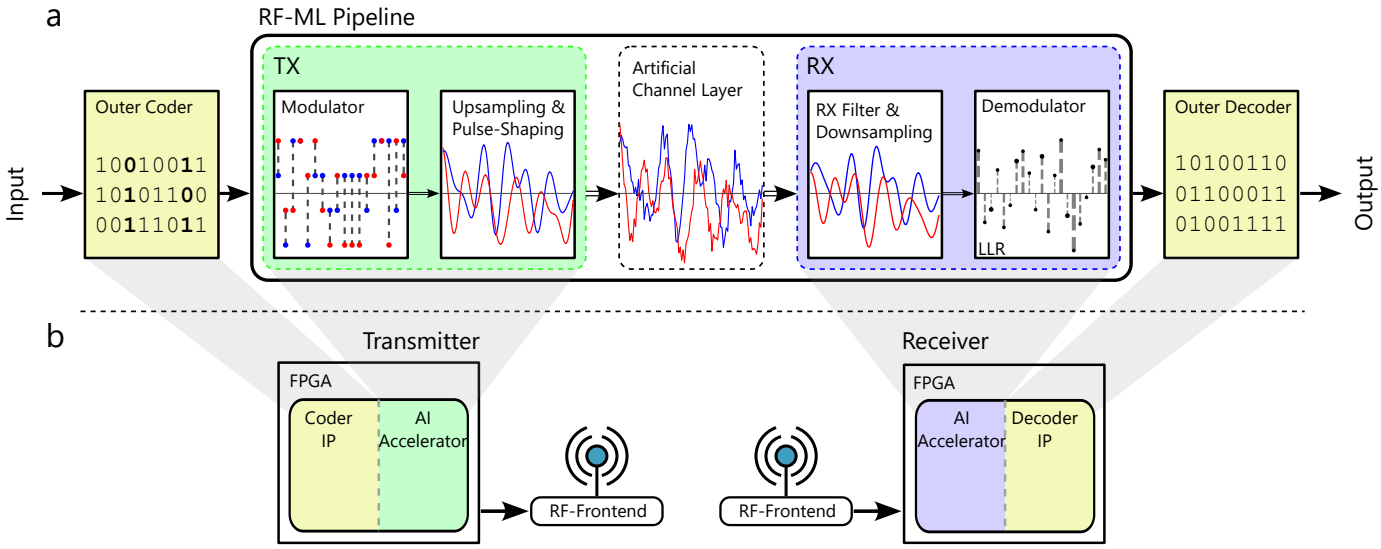

Fig. 1. System architecture of an AI-based hardware-accelerated (FPGA-based) RF-communication system. (a) Autoencoder-based NN implementation; (b) HW-accelerated deployment strategy. Dataflow from left to right. Transmitter-side: Binary data is encoded by an outer encoder implemented as soft-IP (yellow); encoded data is processed by HW-accelerated Transmiter-NN (green); output is propagated to RF-Frontend. Receiver-side: RF-Frontend sends received I/Q-samples to HW-accelerated Receiver-NN (purple); demapped LLRs are decoded by outer decoder implemented as soft-IP (yellow).

processing steps such as Coding, Modulation, Up-sampling, and Pulse-Shaping to generate the physical-layer baseband RF signal on the transmitter side, as well as their respective inverse operations to retrieve the transmitted information from the received signal on the receiver side.[1] Since this work envisions to perform all prior mentioned steps fully within the ML-domain, those steps are framed in the so-called "RF-ML Pipeline" as depicted in Fig. 1, which is a single neural network comprising different layers performing those tasks. We note, that coding and decoding is not implemented within the neural network as it suffers of the *curse of dimensionality* as discussed in [8]. Instead, it is implemented as pre- and post-processing steps by utilizing an Intellectual Property (IP) block[2] deployed on the Programmable Logic (PL). We differentiate between trained, i.e., (De-)Modulation, and non-trained, mathematical, i.e., Up-sampling and Filtering, components. The system is implemented using a bit interleaved coded modulation (BICM) auto-encoder, hence training is performed in an end-to-end (E2E) manner by minimizing the total binary cross entropy between input- and output-bits as follows, resulting in a Bit-Error-Rate (BER) minimization [3]:

$$L(\theta_M, \theta_D) := \sum_{j=1}^{m} \mathsf{E}[H(p_{\theta_M}(b_j|y), \tilde{p}_{\theta_D}(h_j|y))] \quad (1)$$

$$= \sum_{j=1}^{m} \mathsf{E}_{y,b_j}[\ \log\ p_{\theta_M}(b_j|y)] \quad (2)$$

[1]Note: For our hardware experiment we assume a synchronized system by hard-wiring the local oscillators of both transmitting and receiving RF-Frontends, therefore synchronization procedures are out of scope for this work.

[2]LDPC Encoder / Decoder: https://www.xilinx.com/products/intellectual-property/ef-di-ldpc-enc-dec.html

$p_{\theta_M}(b_j|y)$ denotes the probability of the j^{th} output-bit, which is obtained by applying the sigmoid function to the corresponding logit. An artificial channel layer functions as a bottleneck by restricting the amount of information conveyed with one channel use (symbol), forcing the system to adapt it's properties (here: symbol constellation).

Following the training of the system, the channel layer is removed and the NN is split into TX (green) and RX (purple) parts as depicted in Fig. 1, which are then individually deployed on the ML hardware accelerator on both devices. Concatenation of outer (de)-coding completes the RF pipeline.

The following subsection explains the NN realisation of these computation steps in detail:

A. Neural Network RF-ML Pipeline Implementation

Fig. 2 visualizes the E2E data-flow through the processing chain from input bits to RF-signal and backwards.

1) First, the (coded) input bit-stream is modulated by mapping sub-vectors b of length m to a complex baseband symbol c by using a Dense Neural Network (DNN), also known as fully-connected dense layers. Hence, the DNN implements the parametrized function $f_M(\theta_M, b)$ performing the mapping $f_M : f0, ..., 2^m \quad 1g\ 7!\ C$ with trainable parameters θ_M, resulting in modulation order m. In practice, two real-valued outputs are produced for one complex number. The required depth and inner size of the network depends on m and the constellation shape complexity. A shift from sequential to batch-wise modulation, e.g., to boost throughput, is achieved by formulating the DNN operations using equivalent convolutional (CNN) operations, which allows for spatial parallelism in two dimensions, resulting in a modulated output tensor $\mathbf{M}\ 2\ \mathsf{R}^{N_x \mathrm{x} N_y \mathrm{x} 2}$.

2) Up-sampling of the baseband symbols $\mathbf{M}$ with upsampling-factor u_{TX} in the dimension of sequential symbols (here: x-direction) is achieved by applying a transpose depth-wise 2D-convolution with one-hot kernel = $f1, 0, ..., 0g$ 2 $\mathsf{N}^{u_{\mathrm{TX}} \mathrm{x} 1}$ and stride u_{TX}, resulting in an output tensor $\mathbf{U}$ 2 $\mathsf{R}^{(N_x \mathrm{x} u_{\mathrm{TX}}) \mathrm{x} N_y \mathrm{x} 2}$.

3) Finally, transmit-pulse shaping is achieved by applying a depth-wise 2D convolution with "same"-padding and odd-length kernel k 2 $\mathsf{R}^{l \mathrm{x} 1}$ (filled with filter coefficients, e.g., root-raised-cosine) on $\mathbf{U}$, resulting in the I/Q sample tensor $\mathbf{O}$ 2 $\mathsf{R}^{(N_x \mathrm{x} u_{\mathrm{TX}}) \mathrm{x} N_y \mathrm{x} 2}$.[3] N_x and N_y can be considered hyper-parameters to the RF-ML-Pipeline, since they both effect the numerical results of the filtering operation and also the underlying degree of compute parallelization, hence, achievable throughput.

Lastly, the tensor is flattened in the direction of sequential samples and then fed to the RF-Frontend.

On the receiver side, in principle the same operations are performed in an inverse manner. RX processing works on a sampled signal tensor $\mathbf{R}$ 2 $\mathsf{R}^{(N_x \mathrm{x} u_{\mathrm{RX}}) \mathrm{x} N_y \mathrm{x} 2}$ using an up-sampling factor of u_{RX} in the receiving RF-Frontend.

1) First, the received samples will be filtered using a matched-filter by applying the same operation as in 3) in the transmitter-side on $\mathbf{R}$.

2) The filtered samples are down-sampled by factor u_{RX} by applying a depth-wise 2D-convolution with stride u_{RX} and one-hot kernel of length u_{RX}, with the 1 at the position corresponding to the symbol-centered sample.

3) Demodulation from a symbol-centered I/Q-sample s to a Log-Likelihood-Ratio (LLR) vector l of length m is performed using a DNN implementing the function $f_D(\theta_D, s)$, with $f_D : s$ 7! R^m. Similarly as in 1) in TX-side, a CNN is used to perform the same operation in parallel, resulting in a LLR tensor $\mathbf{L}$ 2 $\mathsf{R}^{N_x \mathrm{x} N_y \mathrm{x} m}$.

The LLR tensor is then flattened and fed to the decoder.

B. Training

The auto-encoder system is trained using a variant of stochastic gradient descent (Adam). The training parameters are summarized in Tab. I.

TABLE I
TRAINING PARAMETERS

Parameters	Value
Loss	Binary Cross Entropy
Num. Epochs	25.000
Batch-Size	1000 bits
Training Algorithm	Adam
Learning rate	starting at 10^{-1} and decreasing by /5, /10, /100, /1000 every 5000 epochs
Training SNR	Fix per constellation
Weight Initialization	Glorot

[3]We want to note, that pulse-filtering using a 2D-convolution does not exactly replicate the original operation working on a 1D data array, since the convolution is not performed in one run, but on N_y slices of the data. This results in small dissimilarities in the neighborhood of every N_y samples whose consequences are discussed in chapter IV.

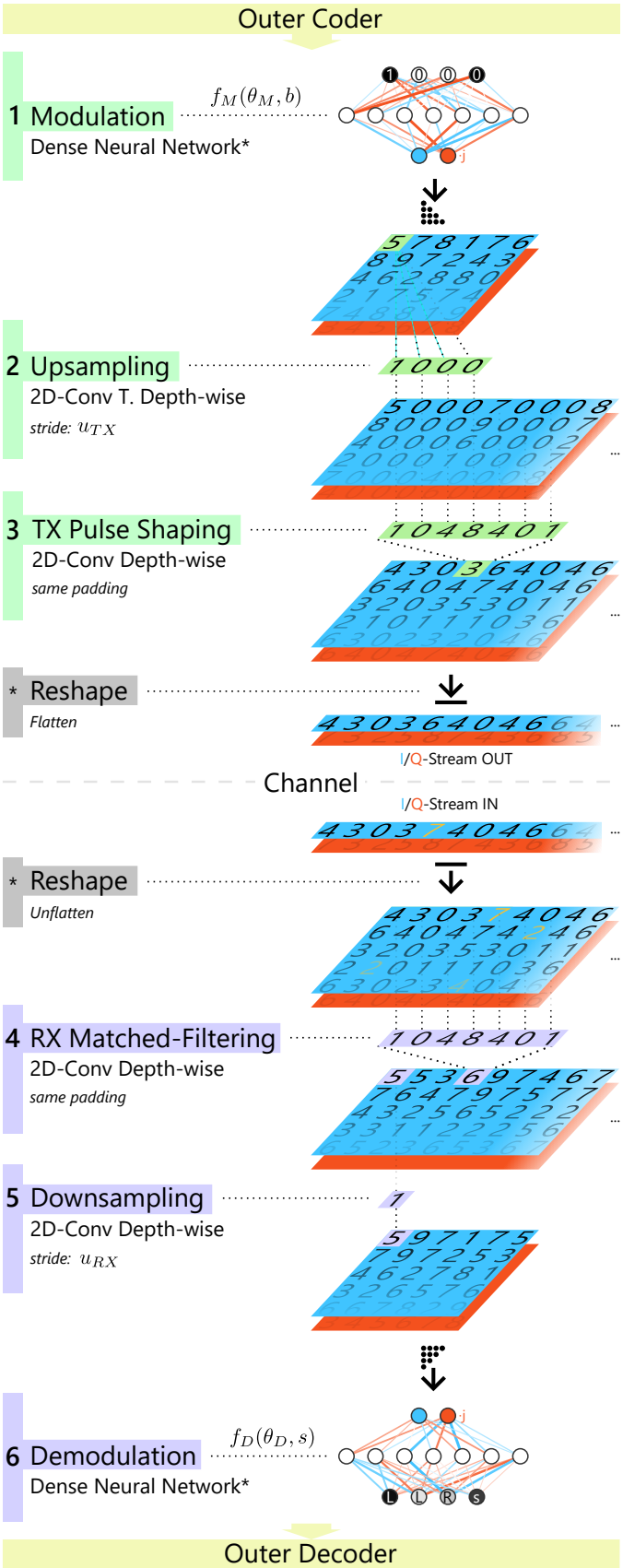

Fig. 2. Detailed implementation of the RF-ML-Pipeline NN.

Constellation energy normalization is performed after the modulator and is not inherently learned by the DNN, since no additional constraint (e.g., impact of energy on the loss) is applied. For long-term training this leads to an exploding output-range of the modulator. Similarly, symbol centering is applied outside the DNN, leading to a non-negligible offset to the DNN's output. These effects have to be taken into account for quantization and HW-model compatibility, as described in Sub-section III-D.

III. Algorithm Deployment

This section provides an overview of the utilized AI-accelerated hardware platform and its deployment process on which the prior introduced RF-ML pipeline is later benchmarked. Remarks regarding compatibility issues and necessary modifications conclude this section.

A. AI-Accelerator overview: AMD-Xilinx Versal

The Versal platform of the manufacturer AMD-Xilinx presents a powerful series of embedded System-on-Chips (SoC) that target a multitude of different computational-demanding embedded tasks, with a particular focus on high-speed ML inference [9]. The Versal AI Core series is one of the state-of-the-art ML hardware accelerators, featuring a novel architecture by combining three different computing paradigms, i.e., processing system / CPU (PS), programmable logic (PL) / FPGA, and a third class of vector compute engines. In the context of ML, these compute engines are referred to as AI-Engines and carry the main computational workload. This work utilizes the VCK190 development board, which ships with 400 AI-Engines. Fig. 3 provides a coarse overview over its computing resources.

To exploit full capabilities of all computing resources in concert, the manufacturer provides a generic inference accelerator in the form of an IP block to be synthesised and deployed on the Versal, as well as a tool-chain to compile deep-learning models from standard formats such as HDF5 and ONNX to hardware-suitable instructions. The interplay of both tools and their deployment workflow is described in the following subsection.

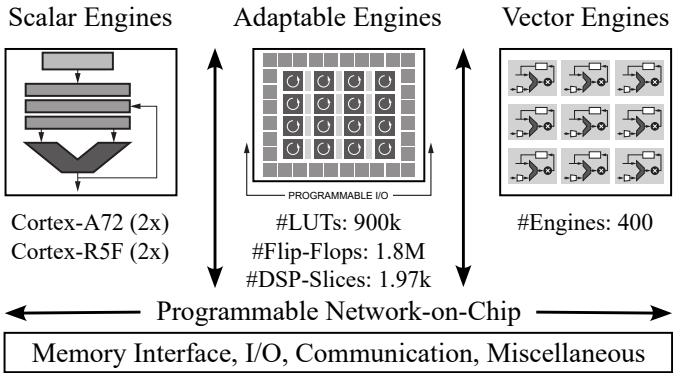

Fig. 3. Computing resources on the Versal platform, comprising scalar (CPU), adaptable (FPGA), and vector (AI-optimized) engines.

B. Generic Inference Accelerator

The generic inference accelerator, from now on also referred to as Deep-Learning Processor Unit (DPU), can be described as a microcoded co-processor with an instruction set optimized for NN inference. The IP is able to execute various ML-architectures dynamically by loading compiled model-files and updating the computation pipeline on PL and AI-Engines on-the-fly without any reprogramming. Configuration options exists that allow trading of its compute power (e.g., through the underlying level of parallelism) vs. hardware resource utilization. In this work, various configurations regarding hardware utilization of the FPGA, num. of AI-Engines, and batch parallelism are analyzed regarding achievable performance. Tab. II summarizes the considered configurations.

TABLE II
Analyzed DPU hardware utilization configurations.

Configuration	Freq.[a] [MHz]	FPGA [%]	AIE [#]	Batch [#]	Int-8 Perf [TOPS]
C32B1	325/1250	6%	32	1	10.24
C32B3	325/1250	15%	96	3	30.72
C32B5	325/1250	24%	160	5	51.20
C64B5[b]	325/1250	26%	320	5	102.4

[a]Frequency of PL / AI-Engines
[b]Maximum Performance Configuration

C. Deployment Workflow

The Deep-Learning Processing Unit expects a hardware-compatible model for inference. This subsection summarizes the steps for transforming a generic ML model to a hardware executable one using the tool-chain Vitis-AI. The deployment workflow can be divided into three categories.

a) Compatible ML Application: The DPU executes ML operations jointly on PL and AI-Engines in a highly efficient and hardware-tailored manner, hence only a subset of operations is supported [10]. To guarantee computation fully within the DPU, it is strictly necessary to ensure compatibility of the designed AI application to the DPU. Both operation type and parameters (e.g., kernel size, padding) are limited. However, unsupported features can also be executed on the PS, which in turn requires the exchange of intermediate data between DPU and PS, imposing a noticeable performance penalty to the application.

b) Model Quantization: DPU inference is based on 8-bit fixed-point computation, since FPGA-based systems benefit from a lower power-consumption, latency [11], and hardware complexity by realizing fixed-point units compared to floating-point units. Even though the AI Engines support floating-point based vector operations, fixed-point computation is more efficient due to their particular high Int-8 compute performance [12]. Vitis-AI performs 8-bit quantization of inputs, weights, biases, and activations after training using a power-of-two quantization scheme, a special form of quantization well-suited for FPGAs [13]. While this scheme is referred to as Post-training quantization (PTQ), it is also possible, though not used

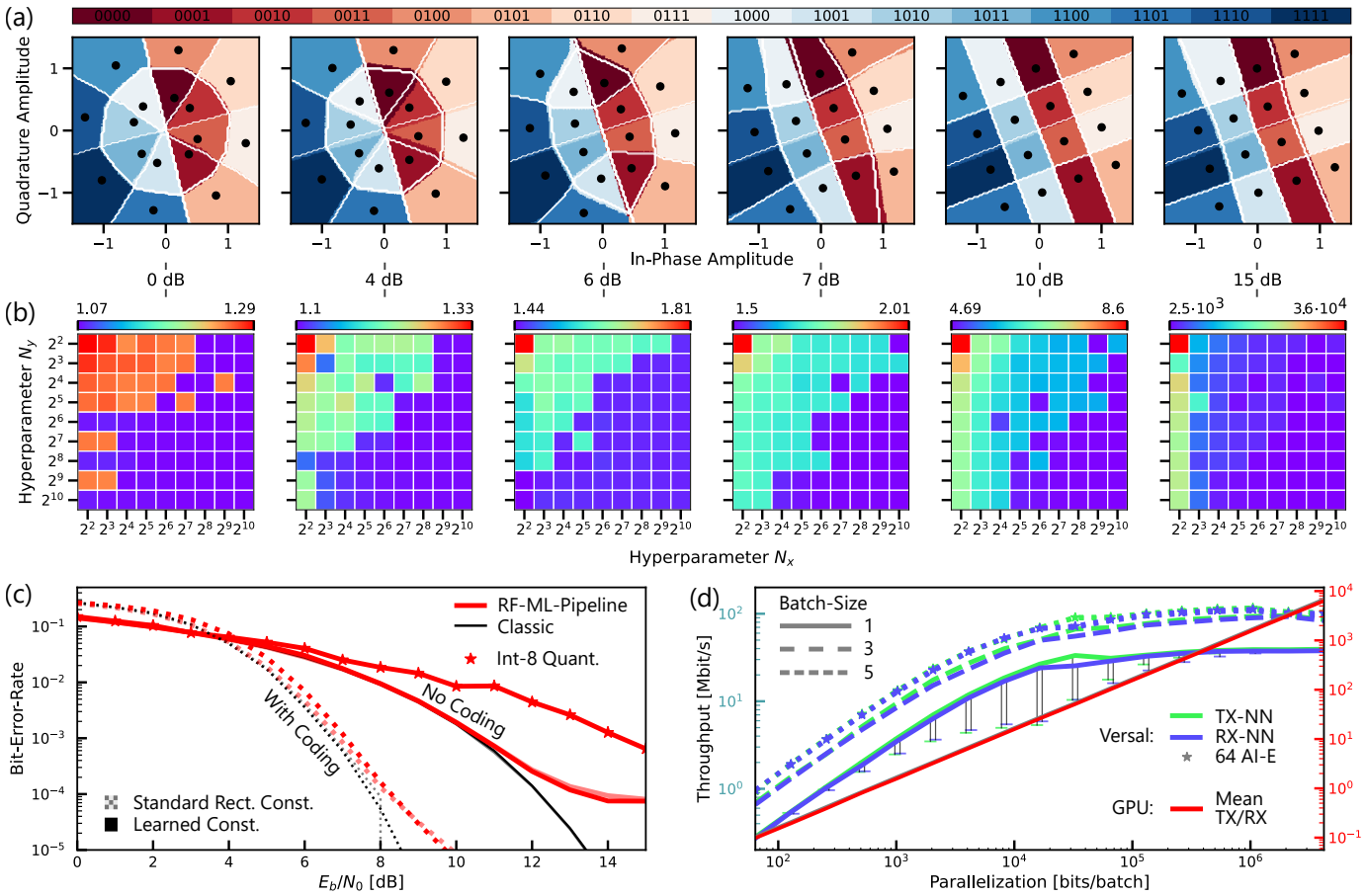

Fig. 4. BER- & Throughput-Performance of the proposed RF-ML-Pipeline. (a): Learned constellations (dotted points) and corresponding demapper decision regions for selected BERs for hardware quantized (shaded area) and Float-32 (white lines) computation. (b): BER ratio of uncoded HW-deployed RF-ML-pipeline vs. Float-32 "classic" computation for all power-of-two parallelization combinations in [2; 10]. (c): Simulated BER performance of coded (dotted) and un-coded (solid) system configurations with classic (black) or 256x256-RF-ML-Pipeline (red, thick) unquantized computation and learned (opaque) or standard rect. 16-QAM (semi-transparent) constellations. Line with star-marker denotes HW-measured BER performance. (d): Throughput analysis of RF-ML-Pipeline (blue: TX-NN, green: RX-NN, left y-axis) on Versal hardware platform for batch-sizes [1,3,5] vs. GPU throughput (red, right y-axis) for batch-size 1.

in this work, to perform Quantization-aware training (QAT), where the model is further trained by emulating inference-time quantization, creating a more robust model.

c) Model Compilation: Finally, the quantized NN is compiled into DPU-compatible instructions by invoking the Vitis AI Compiler. This generates a program consisting of quantized data and microcode for the DPU, which can be loaded from the file system into memory and transferred to the co-processor.

D. Model compatibility adaptation

The proposed RF-ML pipeline NN implementation as depicted in Fig. 2 is operation-wise fully compatible with the feature support of the DPU. However, limitations w.r.t. parameters, e.g., kernel dimensions of w, h 2 f1, ..., 15g, wxh 64 for 2D-convolutions, apply, which can result in degraded system performance (here: restricting filter taps to a maximum of 15). Circumventing these limitations can pose significant implementation challenges. In this work, the kernel length limitation has been overcome by splitting the original filter into multiple, smaller filters and concatenating the partial results Moreover, quantization strategies can require zero-centered outputs to exploit full Int-8 bit-width. Hence, the mapper DNN's output offset, as described in Sub-section II-B, is compensated by manually adjusting the bias of its last layer.

IV. System Validation and Performance Analysis

In this section we validate the functionality of the proposed RF-ML pipeline in simulation and on hardware through comparison to its "equivalent" classic implementation for various combinations of both hyper-parameters N_x and N_y. Additionally, the processing capabilities and implementation efficiency of the NN on the Versal platform w.r.t. throughput are benchmarked for a variety of hardware configurations and compared to a modern GPU-based setup.

To analyze the BER performance of multiple configurations of the RF-ML-Pipeline over a suitable signal-to-noise power range, we implement both AI-based and classic baseline systems with *modulation order* $m = 4$, wrap them inside a Low-density parity-check *(LDPC)* encoder-decoder pair with

code-rate $c_r = \frac{1}{2}$ and codeword length $c_l = 80$ if applicable, and train them to specific SNR values each. For pulse-shaping and matched-filtering a root-raised-cosine *(RRC)* filter with roll-off-factor $\beta = 0.22$ and odd length spanning 22 symbols is used with up-sampling factors $u_{\text{TX}} = u_{\text{RX}} = 2$. Mapper and demapper are 1-hidden layer DNNs with 128 neurons each. The classic systems are modeled using the Sionna framework [14]. The results are displayed in Fig. 4.

Fig. 4(a) visualizes the learned constellations (black dots) and the corresponding demapper decision regions (area: quantized; white line: unquantized) as a function of the SNR. It can be seen that the match of quantized and unquantized decision regions is good, however, differences exist that will result in misclassification, degrading quantized performance slightly.

Fig. 4(b) denotes the BER ratio of the uncoded HW-deployed RF-ML-Pipeline (with two Ettus X300 USRPs over a coaxial-cable channel) vs. simulated "classic" baseline as a function of the hyperparameters N_x and N_y and the SNR. While the HW-deployed performance is always worse than the simulated classic implementation, the ratio shows a strong dependency on the hyperparameters, decreasing approximately symmetrically with the increase of either one. This effect is caused by the data anomaly introduced by the "row-wise" filtering and concatenation operation in the TX- and RX-NN, introducing a systematic numerical error compared to classic filtering, leading to a decreased BER. This impact shows a high correlation with the SNR. Moreover, for high SNRs N_x has a stronger impact than N_y, which is simply caused by the fact that the numerical error can be eradicated by increasing the filter-row's length, i.e., N_x, to infinity.

Fig. 4(c) compares the BER-curves of *simulations* for various system configurations that can be divided in three categories: No-coding (solid) vs. coding (dotted), RF-ML-Pipeline computation ($N_x = N_y = 256$, red, thick) vs. classic computation (black), and learned mapping/demapping (opaque) vs. standard 16-QAM (semi-transparent). Firstly, the choice of constellation does not noticeable impact the performance, verifying the learned constellation's quality. Secondly, one can observe the RF-ML-Pipeline's BER degredation for high SNRs caused by the above described systematic error. Coding mitigates this issue only partially as visible for SNRs between 8-10 dB. This might be caused by the distribution of bit errors caused by the systematic numerical error, which is not uniform but exhibits spikes at a repeating rate determined by N_x. Lastly, the BER curve of the HW-deployed system is denoted by the star-marked line for comparison.

Finally, Fig. 4(d) shows throughput performance in MBit/s of the RF-ML-Pipeline implementation on the Versal AI-accelerator for multiple hardware configurations as a function of the number of bits processed per batch. Since multiple combinations of N_x and N_y can yield the same total parallelization, only the best performing value is chosen. The vertical lines for batch-size 1 configuration denote the respective throughput ranges for a given parallelization factor. A strong correlation to the degree of parallelization is observed. Both TX- and RX-NN perform similar, as they feature the same type of operations. RX-NN is slightly slower, which can be traced back to different data load and store speeds of the DPU. An increase in the batch-size is reflected on the throughput with a mean increase by factor 2.3 for $1 \; ! \; 3$ and 3.0 for $1 \; ! \; 5$. Surprisingly, increasing the AI-Engines from 32 to 64 for the batch-size 5 configuration only yields a 1.4% throughput gain. In general, throughput saturates for high parallelization values, topping out at 116 MBit/s. For comparison, GPU throughput performance of a NVidia V100 (AWS ml.p3 instance) for batch-size 1 is denoted by the red line, and exhibits a linear performance with a maximum of 6.8 GBit/s.

V. Conclusion and Future work

In this work we proposed and implemented our vision of a flexible, reconfigurable physical layer for RF communications through the use of generic AI hardware accelerators. The performance analysis w.r.t. BER shows promising results but hints improvements both for NN implementation and hardware-tailored deployment (e.g., quantization-aware training), indicating the necessity for follow-up work. Moreover, extending the RF-ML-Pipeline's capabilities, e.g., synchronization procedures, channel equalization, etc., is required for practical operation. Lastly, exploring the landscape of RF-targeted AI-accelerators, e.g., the Versal RF SoC, is subject of future work.

References

[1] 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.

[2] S. Dörner, S. Cammerer, J. Hoydis, and S. t. Brink, "Deep learning based communication over the air," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 132–143, 2018.

[3] S. Cammerer, F. A. Aoudia, S. Dörner, M. Stark, J. Hoydis, and S. ten Brink, "Trainable communication systems: Concepts and prototype," *IEEE Transactions on Communications*, vol. 68, no. 9, 2020.

[4] A. Felix, S. Cammerer, S. Dörner, J. Hoydis, and S. Ten Brink, "Ofdm-autoencoder for end-to-end learning of communications systems," in *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2018, pp. 1–5.

[5] S. Dörner, S. Rottacker, M. Gauger, and S. t. Brink, "Bit-wise autoencoder for multiple antenna systems," in *2021 17th International Symposium on Wireless Communication Systems (ISWCS)*, 2021.

[6] M. B. Fischer, S. Dörner, S. Cammerer, T. Shimizu, H. Lu, and S. T. Brink, "Adaptive neural network-based ofdm receivers," in *2022 IEEE 23rd International Workshop on Signal Processing Advances in Wireless Communication (SPAWC)*, 2022, pp. 1–5.

[7] W. Xu, Z. Zhong, Y. Be'ery, X. You, and C. Zhang, "Joint neural network equalizer and decoder," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, 2018, pp. 1–5.

[8] S. Cammerer, J. Hoydis, F. Aoudia, and A. Keller, "Graph neural networks for channel decoding," 12 2022, pp. 486–491.

[9] Advanced Micro Devices, Inc., "Versal architecture and product data sheet: Overview," https://docs.xilinx.com/v/u/en-US/ds950-versal-overview, 2022, [Online; accessed 25-May-2023].

[10] AMD-Xilinx, "Dpucvdx8g for versal acaps," https://docs.xilinx.com/r/en-US/pg389-dpucvdx8g/Introduction, 2022.

[11] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, and M. van Baalen, "A white paper on neural network quantization," 2021.

[12] A. Core and A. K. Gupta, "Architecture apocalypse dream architecture for deep learning inference and compute-versal ai core," 2020.

[13] D. Przewlocka-Rus, S. S. Sarwar, H. E. Sumbul, Y. Li, and B. D. Salvo, "Power-of-two quantization for low bitwidth and hardware compliant neural networks," 2022.

[14] J. Hoydis, S. Cammerer, F. Ait Aoudia, A. Vem, N. Binder, G. Marcus, and A. Keller, "Sionna: An open-source library for next-generation physical layer research," *arXiv preprint*, Mar. 2022.