Neural Transmitter with Co-Optimized Digital Front End Functions

Master's thesis in Embedded Electronic System Design

Noopura Parvathi Attupurath

Department of Microtechnology and Nanoscience
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Neural Transmitter with Co-Optimized Digital Front End Functions

Noopura Parvathi Attupurath

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2026

Neural Transmitter with Co-Optimized Digital Front End Functions
Noopura Parvathi Attupurath

Supervisor: Lars Svensson, Microtechnology and Nanoscience
Company advisor: Himanshu Gaur, Ericsson AB
Examiner: Per Larsson Edefors, Microtechnology and Nanoscience

Master's Thesis 2026
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Neural Transmitter with Co-Optimized Digital Front End Functions
Noopura Parvathi Attupurath
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

In order to increase Power Amplifier (PA) linearity and efficiency in the transmission chain wireless communication systems, Digital Front-End (DFE) functions such as Crest Factor Reduction (CFR), Hard Limiter (HL) and Digital Pre-Distortion (DPD) are typically built and adjusted independently. Despite the effectiveness of this modular strategy, it does result in a high hardware footprint. The use of Neural Networks (NNs) to simultaneously model and co-optimize DFE functions for wireless communication systems with memory-effect-exhibiting PAs is examined in this thesis.

As a reference, a functional hardware-compatible legacy DFE with CFR, HL and DPD was developed, which was used to evaluate the NN-based DFE solutions. The NN models were compressed using pruning, projection, and quantization methods to allow deployment in contexts with limited resources. To compare the hardware cost of these solutions, bit-exact models were developed. The results demonstrate that, although at a much greater hardware cost, uncompressed NN-based DFE models give better performance than the legacy DFE. Implementation complexity is significantly reduced by compression, though the compression techniques investigated also give significant performance loss. The quantization-based compression gives the best implementation complexity reduction, though the projection-based compression offers a better performance-complexity trade-off than quantization.

Everything being considered, the findings show the promise of NN-based co-optimization of DFE functions while emphasizing the necessity of better compression methods for realistic implementation.

Keywords: Digital Front-End (DFE), Digital Predistortion (DPD), Crest Factor Reduction (CFR), Hard Limiter (HL), Neural Networks (NN), Power Amplifiers (PA), Co-optimization, Neural Network Compression, Quantization-Aware Training (QAT), Wireless Communications.

Acknowledgements

I would like to thank the following people for their support during the completion of my thesis.
I would like to thank my examiner, Per Larsson Edefors and my university supervisor, Lars Svensson for their valuable feedback and for securing the academic quality of this work.
Special thanks to my industrial supervisor Himanshu Gaur and my manager Erik Garmland both at Ericsson Lund for their daily guidance, technical insights and continuous encouragement during this work.
Special thanks to my thesis partner, Astrid Hofvander from Lund University. Our collaboration, shared discussions, and mutual support greatly enriched this research endeavor.
Finally, I would like to thank my parents and family for their constant support, patience and belief in me throughout my educational journey.

Noopura Parvathi Attupurath, Gothenburg, September 2026

Ethical Considerations and Use of Generative AI

Generative artificial intelligence (AI), especially OpenAI's ChatGPT, was used as an assistive tool in preparing this thesis. Its usage was limited to editing and checking the grammar of the language, improving sentence structure and readability and helping in clear presentation of some sections of the report. The experimental results were not generated by generative AI and the technical and scientific work of the authors was not replaced by generative AI. The methodology, implementation, simulations, measurements, data analysis, charts, tables and conclusions in this thesis are based on the authors' own work.

The AI-assisted content has all been reviewed and verified by the authors, who take full responsibility for the accuracy, originality and scientific content of the thesis.

Contents

Abbreviations

ACPR	Adjacent Channel Power Ratio
ACLR	Adjacent Channel Leakage Ratio
AdaRound	Adaptive Rounding
AFE	Analog Front End
AI	Artificial Intelligence
AILC	Augmented Iterative Learning Control
AM/AM	Amplitude-to-Amplitude
AM/PM	Amplitude-to-Phase
ASIC	Application-Specific Integrated Circuit
BiLSTM	Bidirectional Long Short-Term Memory
BRAM	Block Random Access Memory
CAF	Clip and Filter
CCDF	Complementary Cumulative Distribution Function
CFR	Crest Factor Reduction
CORDIC	Coordinate Rotation Digital Computer
CP	Cyclic Prefix
dB	decibel
dBc	decibels relative to the carrier
DFE	Digital Front End
DFT	Discrete Fourier Transform
DLA	Direct Learning Architecture
DNN	Deep Neural Network
DPD	Digital Pre-Distortion
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EVM	Error Vector Magnitude
FF	Flip-Flop
FFp	Flip-Flop proxy
FFT	Fast Fourier Transform
FIR	Finite-duration Impulse Response
FLOP	Floating Point Operation
FP16	16-bit Floating-Point Half-Precision
FP32	32-bit Floating-Point Single-Precision
FP64	64-bit Floating-Point Double-Precision
FPGA	Field-Programmable Gate Array
FT	Fourier Transform
GMP	Generalized Memory Polynomial
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HC	Hard Clipping
HDL	Hardware Descriptive Language
HL	Hard Limiter
I	In-phase
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
ILA	Indirect Learning Architecture
INT4	4-bit Integer
INT8	8-bit Integer
ISI	Inter-Symbol Interference
LMS	Least-Mean-Square
LS	Least Square
LSB	Least Significant Bit
LSTM	Long Short-Term Memory
LTE	Long-Term Evolution

LUT	Look-Up Table
MAC	Multiply-Accumulate Operation
MCFR	Model-Based Crest Factor Reduction
ML	Machine Learning
MP	Memory Polynomial
MSE	Mean Squared Error
MSLAE	Mean Squared Logarithmic Absolute Error
NMSE	Normalized Mean Squared Error
NN	Neural Network
NR	New Radio
NTK	Neural Tangent Kernel
OBD	Optimal Brain Damage
OFDM	Orthogonal Frequency Division Multiplexing
PA	Power Amplifier
PAPR	Peak-to-Average Power Ratio
PCA	Principal Component Analysis
PCCFR	Peak Cancellation Crest Factor Reduction
PTQ	Post-Training Quantization
PTS	Partial Transmit Sequence
PQN	Pseudo-Quantization Noise
Q	Quadrature
QAM	Quadrature Amplitude Modulation
QAT	Quantization-Aware Training
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RF	Radio Frequency
RRC	Root-Raised Cosine
RTL	Register-Transfer Level
SDR	Signal-to-Distortion Power Ratio
SLM	Selective Mapping
SRAM	Static Random Access Memory
SVD	Singular Value Decomposition
SynFlow	Synaptic Flow
TPU	Tensor Processing Unit
VHDL	Very High-Speed Integrated Circuit Hardware Description Language

1

Introduction

1.1 Background

The Power Amplifier (PA) is a hardware component used to amplify the power of Radio Frequency (RF) signals such that they can be transmitted with sufficient power in wireless communication systems. Due to its construction this amplification is not linear, which distorts the signal to be transmitted, and it may exhibit memory effects in which the PA output depends on current and past inputs [1]. The PA also consumes significant power, wherefore it is desired to operate it as efficiently as possible. However, for the operating points for which the PA is most efficient, the PA is also severely nonlinear [2]. Modern wireless communication systems rely on complex modulation schemes and wide bandwidths to achieve high data rates. These signals exhibit a high Peak-to-Average Power Ratio (PAPR), which requires the PA, the most power-hungry component in the transmitter, to operate with a large back-off to avoid nonlinear distortion [3]. This significantly reduces power efficiency [4], trading off between efficiency and linearity. To enable the PA to operate with high linearity and high efficiency, a so called Digital Front-End (DFE) is applied, which consists of components such as Digital Pre-Distortion (DPD), Crest Factor Reduction (CFR) and Hard Limiter (HL). The DFE in practice is implemented on hardware, either on Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs), which are resource constrained environments. As such the DFE must balance performance and resource usage, considering things such as efficient algorithm implementations for hardware and the precision of number representation.

DPD is a widely adopted technique to linearize PAs by predistorting the signal prior to amplification. This is done by applying an inverse nonlinear function before the PA, commonly by using Memory Polynomials (MPs) or Generalized Memory Polynomials (GMPs) which accounts for the PA nonlinearity and memory effects [5]. Traditionally these are implemented computationally efficient using Look-Up Tables (LUTs), but as bandwidth increases so does the model complexity [6]. CFR techniques, such as Peak Cancellation CFR (PCCFR), are used prior to DPD to reduce PAPR, but they introduce in-band distortion and out-of-band spectral re-growth. There are CFR techniques which minimize these impairments, however they incur high computational complexity [7]. The HL may be applied after the CFR to further reduce the PAPR of the signal prior to DPD [8]. Traditionally CFR and DPD are optimized separately and each DFE component is applied one after

the other to the signal, which results in a high hardware footprint in a resource constrained environment.

Neural Networks (NNs) are Machine Learning (ML) models that can be used to model complex input-output relationships [9], with the universal approximation theorem demonstrating that they are able to approximate any nonlinear function [10]. Due to their great adaptability and ability to jointly compensate for multiple impairments, recent research has explored NN-based DPD solutions [11], as well as other techniques in wireless communication [1]. There has been a number of attempts to model CFR and DPD individually or jointly using NNs, such as in [12], [1] and [13]. In general such attempts do not consider PA memory effects, and those that do generally do not consider the complexity of the solution. NNs are computationally expensive models, so in order to use NN-based solution to replace the DFE or parts of the DFE such as the DPD, there is a need to either impose size restrictions on the model which may affect their ability to learn the desired input-output relationship, or apply compression techniques to a larger, well-performing NN to reduce the model size while retaining good performance.

1.2 Purpose and Goals

The main purpose of this work is to investigate NN-based solutions to co-optimize the components of the DFE to address the fundamental trade-off between linearity and power efficiency in PAs by:

- Establishing a reference baseline using a conventional DFE, with independently optimized CFR and DPD models and HL in MATLAB/Simulink implemented in a hardware compatible manner. Specifically, the CFR is implemented using PCCFR and the DPD modeled using MP implemented with a dual polyphase architecture using LUTs.
- Looking into a paradigm shift toward unified NN-based solutions that can optimize both CFR and DPD at the same time, resulting in either better linearization or less complex hardware, optimally both.
- Setting up both techniques for hardware implementation through fixed-point conversion followed by Hardware Description Language (HDL) code generation from which hardware resource usage can be estimated.
- Looking specifically at NN-based co-optimized DFEs for PAs exhibiting memory effects, which has not been touched upon much in literature.

The specific technical goals are:

1. Develop functional models and compare performance:
   (a) Develop a functional reference MATLAB/Simulink-based DPD model, referred to as the legacy DPD, by implementing an established algorithm in a hardware compatible manner, specifically MP implemented using dual polyphase architecture with LUTs.
   (b) Develop a functional reference MATLAB/Simulink-based DFE model, referred to as the legacy DFE, consisting of separate CFR, HL and DPD

blocks, implementing established algorithms in a hardware compatible manner. Specifically, clipping with peak cancellation for CFR, and MP implemented using dual polyphase architecture with LUTs for DPD.

(c) Model the Analog Front-End and PAs with memory considerations using high-fidelity MATLAB Simulink RF models.

(d) Design and implement a compressed NN-based DPD model.

(e) Compare the performance of the compressed NN-based DPD against the hardware compatible legacy DPD using standard RF metrics (Error Vector Magnitude (EVM), Adjacent Channel Power Ratio (ACPR), PAPR).

(f) Design and implement a compressed NN-based co-optimized DFE model.

(g) Compare the performance of the compressed NN-based DFE against the hardware compatible legacy CFR, HL and DPD blocks using standard RF metrics (EVM, ACPR, PAPR) performing end-to-end performance evaluation against the legacy DFE chain.

2. Develop bit-exact models and compare hardware cost:

   (a) Set up the NN-based solutions and legacy solution for hardware implementation through fixed point conversion followed by HDL code generation from which the implementation feasibility can be assessed and the hardware resource usage can be estimated.

   (b) Compare the estimated hardware resource usage between the NN and legacy approaches.

1.3 Thesis Outline

In Chapter 2 all theory in regards to the reference system, the NN solution and the comparison of the two in terms of performance and hardware resource usage is outlined. More specifically, the theory behind each component in the reference system is explained, as well as the theory of NNs in general and NNs in wireless communication. Moreover, the theory of setting up the solutions for hardware implementation is explained.

In Chapter 3 it is explained how all components in the reference system and and NN solution were designed and implemented. Furthermore, it is explained how both solutions were set up for hardware implementation in practice.

In Chapter 4 the practical implementation of the reference system is presented, including the signal generation, PCCFR, hard limiter, digital pre-distortion, and power amplifier. The signal, power-amplifier, and performance-limit specifications are then defined. Finally, the implementation of the NN based solution is described, including the joint and co-optimized digital front-end architecture and the applied neural-network compression methods.

In Chapter 5 experimental results comparing the hardware compatible legacy and NN solutions in terms of performance and hardware resource usage is presented

and discussed. The experiments are done by running computer simulations on the reference system and the NN solution with varying parameters.

In the final Chapter 6 a concluding summary is given with the conclusions of the project in terms of how well the results meet the goals and purposes. Furthermore, areas open for further investigations and potential future work directions are presented.

1.4 Work Distribution

The thesis has been made in collaboration between Noopura Parvathi Attupurath for Chalmers University of Technology and Astrid Hofvander for Lund University and has also been published by Lund University in [14]. During the thesis work Noopura has given more attention to the design and implementation of the CFR as well as setting the legacy solution and NN solution up for hardware implementation, and Astrid has given more attention to the design and implementation of the HL, DPD and NN solution as well as the signal generation and PA implementation.

In practice this means that the introduction, results and conclusion chapters (1, 5 and 6) have been written by both Noopura and Astrid, and that the different parts in the background and methodology chapters (2 and 3) have been written by whoever focused on that area of investigation. Specifically, Noopura is the main author of following Sections:

- 2.2 (Hardware-Efficient Number Representation),
- 2.4.2 (Crest Factor Reduction) & 2.5 (HDL Code Generation from Fixed-Point Models),
- 2.6.2 (Basics of Compression: Pruning, Projection and Quantization),
- 3.1 (Software),
- 3.2 (Hardware Evaluations),
- 3.4 (Resource Usage Estimation)
- 4.1.2 (PCCFR Implementation) & 3.3 (Bit-Exact Conversion and Hardware Code Generation),
- 4.3.2 (Neural Network compression)

and Astrid is the main author of Sections:

- 2.3 (Algorithms),
- 2.4.1 (Signal), 2.4.3 (Hard Limiter), 2.4.4 (Digital Pre-Distortion) & 2.4.5 (Power Amplifier),
- 2.6 (Neural Networks) & 2.6.1 (Model Digital Front End using Neural Networks),
- 4.1.1 (Signal Generation), 4.1.3 (Hard Limiter Implementation), 4.1.4 (Digital Pre-Distortion Implementation) & 4.1.5 (Power Amplifier Implementation),
- 4.3.1 (Joint and Co-optimized Digital Front End Neural Network implementation),

and Section 2.1 (Performance Measurements) was written jointly.

2

Background

This chapter introduces the theoretical foundations and reference-system components required for the thesis. It first presents the performance measurements and number representations used in the evaluations, followed by the relevant algorithms, reference DFE components, hardware-code-generation concepts, and NN compression methods.

2.1 Performance Measurements

To evaluate the performance of a system for a signal, the measurements described in the subsequent subsections are used. Here, system refers to a DFE model and a PA, and signal refers to a representation of information to be transmitted through the system. A detailed explanation of the system and signal used for this thesis can be found in Section 2.4.

2.1.1 Peak-To-Average Power Ratio

The PAPR of a discrete signal x with length N is defined as

$$PAPR = 10 \cdot \log_{10}\left(\frac{P_{peak}}{P_{avg}}\right), \tag{2.1}$$

given in the unit decibel (dB), where $P(n)$ is the power of signal sample n given by

$$P(n) = |x(n)|^2, \tag{2.2}$$

and P_{peak} is the maximum power and P_{avg} is the average power of the signal given by

$$P_{peak} = \max_n P(n), \tag{2.3}$$

and

$$P_{avg} = \frac{1}{N}\sum_{n=1}^{N} P(n), \tag{2.4}$$

where N signal length, i.e. the number of signal samples. PAPR is an important measurement used to evaluate the performance of the DFE components. Good DFE components should result in output signals with PAPR values below set thresholds.

2.1.2 Complementary Cumulative Distribution Function

The Complementary Cumulative Distribution Function (CCDF) of instantaneous power of a discrete signal x with length N is defined as the probability that the power of a signal sample would result in a PAPR above a set threshold $PAPR_{lim}$, which is given by

$$CCDF(PAPR_{lim}, P(n)) = \mathcal{P}\left(10 \cdot \log_{10}\left(\frac{P(n)}{P_{avg}}\right) > PAPR_{lim}\right), \tag{2.5}$$

where $\mathcal{P}(\cdot)$ is the probability, $P(n)$ is the signal power given by Equation (2.2), P_{avg} is the average power of the signal given by Equation (2.4) and $PAPR_{lim}$ is the set PAPR limit. CCDF is an important measurement used to evaluate the CFR performance. A good CFR component should shift the CCDF curve to the left and downward, indicating that fewer signal samples have powers resulting in PAPR above the set PAPR limit.

2.1.3 Error Vector Magnitude

The EVM of a system (or a single component if so desired) is defined as

$$EVM(x_{in}, x_{out}) = \sqrt{\frac{\mathbb{E}(|x_{out} - x_{in}|^2)}{\mathbb{E}(|x_{in}|^2)}} = \sqrt{\frac{\frac{1}{N}\sum_{n=1}^{N}|x_{out}(n) - x_{in}(n)|^2}{\frac{1}{N}\sum_{n=1}^{N}|x_{in}(n)|^2}}, \tag{2.6}$$

where x_{in} is the discrete system input signal and x_{out} is the discrete system output signal in the RF domain, both of length N. It is often converted into dB as

$$EVM_{dB}(x_{in}, x_{out}) = 20\log_{10}(EVM(x_{in}, x_{out})/100). \tag{2.7}$$

EVM is an important measurement used to measure the amount of in-band distortion of the signal and is used to evaluate the total end-to-end system performance in terms of PA linearization. A good DFE should result in a system with low EVM, i.e. the error between the input and output signal should be small.

2.1.4 Normalized Mean Squared Error

The Normalized Mean Squared Error (NMSE) between model estimate Y and target T is calculated as

$$NMSE(Y, T) = \frac{\mathbb{E}(|Y - T|^2)}{\mathbb{E}(|T|)} = \frac{\frac{1}{N}\sum_{n=1}^{N}|Y_n - T_n|^2}{\frac{1}{N}\sum_{n=1}^{N}|T_n|^2}. \tag{2.8}$$

It can be converted into dB as

$$NMSE_{dB}(Y, T) = 10\log_{10}(NMSE(Y, T)). \tag{2.9}$$

In the case where Y and T are replaced with x_{in} and x_{out} of a system, the NMSE becomes the square of the EVM of the system and as such NMSE can be used instead of EVM to evaluate the linearization performance of the system.

2.1.5 Adjacent Channel Power Ratio

The ACPR, also known as Adjacent Channel Leakage Ratio (ACLR), is defined as the ratio between the power transmitted into an adjacent channel and the power transmitted into the main channel and is calculated as

$$ACPR = \frac{\text{power in the adjacent channel}}{\text{power in the main channel}}. \tag{2.10}$$

It is often converted into decibels relative to the carrier (dBc) as

$$ACPR_{dBc} = 10\log_{10}(ACPR). \tag{2.11}$$

The channel of a signal is the frequencies occupied by the signal, where the main channel is the desired occupied frequencies of the signal and the adjacent channels of a signal are the frequencies occupied by the signal that are the main channels of other signal. ACPR is an important measurement used to measure the amount of out-of-band spectral regrowth and is used to evaluate the total end-to-end system performance in terms of power leakage. A good DFE should result in a system with low ACPR, i.e. minimal power should be transmitted into the main channels of other signals. In this thesis the ACPR of a signal will be reported for the left and right adjacent channels of the signal.

2.2 Hardware-Efficient Number Representation

Developing hardware-efficient DPD and CFR systems requires an understanding of fixed-point and floating-point number representation, in regards to their precision and hardware resource usage. The theoretical foundations of both representation systems, their mathematical formulations, and the important factors to take into account when switching from floating-point reference models to bit-exact fixed-point hardware implementations are covered in this part.

2.2.1 Number Representation, Advantages and Limitations

Digital Number Representation

Numbers are generally represented using binary words, which comprise of sequences of fixed bits (1 and 0). The datatype definition determines whether numbers will be represented as either fixed point or floating point representation [15]. This is very important in design cases such as CFR and DPD where the number precision will affect the performance of the system.

Fixed-Point Arithmetic

The three essential characteristics of fixed point numbers are signedness, binary point position, and word length (number of bits). Both integer and fractional values can be represented according to the binary point position, which efficiently scales the number. The scaling equation in [15] can be used to express the real world value

($\tilde{V}$) for a generalized fixed-point number. For unsigned fixed-point numbers, the equation becomes

$$\tilde{V} = S \cdot \left[\sum_{i=0}^{w_s-1} b_i 2^i \right] + B, \tag{2.12}$$

and for signed (two's complement) fixed-point numbers the equation becomes

$$\tilde{V} = S \cdot \left[-b_{w_s-1} 2^{w_s-1} + \sum_{i=0}^{w_s-2} b_i 2^i \right] + B, \tag{2.13}$$

where b_i represents the i-th binary digit, w_s is the word size in bits, S is the slope (scaling factor) and B is the bias. The slope S is typically expressed as

$$S = F \cdot 2^E,$$

where F is the slope adjustment factor (in the range $[1.0, 2.0)$) and E is the fixed power-of-two exponent [15]. This representation allows for precise control over numerical range and resolution, which is essential for hardware-efficient implementations.

Fixed Point Hardware Advantages

For DPD and PCCFR implementations, fixed-point hardware has a number of strong benefits:

Size and Power Consumption Because fixed-point logic circuits do not require exponent handling, normalization, or rounding logic, they are far simpler than their IEEE floating-point equivalents. As a result, the chip area and power consumption are reduced. It is crucial to remember that floating-point multipliers internally incorporate fixed-point multipliers; exponent management, alignment, and exception handling are the main causes of IEEE floating-point's extra complexity. Compared to complete IEEE implementations, non-IEEE floating-point formats that reduce or eliminate these characteristics can be much simpler [16].

Memory Usage and Speed Real-time processing of high-bandwidth signals is made possible by fixed-point computations, which usually use less memory and run more quickly. However, it is crucial to understand that the operand resolution (bit-width) significantly influences the temporal performance of both fixed-point and floating-point arithmetic. Regardless of whether the representation is fixed-point or floating-point, wider data channels result in longer critical paths because of increased carry propagation and more sophisticated logic [17]. According to recent research, pipelining can greatly enhance floating-point units' timing performance; nevertheless, the effectiveness of these methods is highly dependent on the resolution used [17].

Cost Effectiveness Fixed-point hardware is a less expensive alternative for mass produced goods.

Floating Point Arithmetic

Scientific notation, represented as f, allows floating-point values to bypass the dynamic range constraints of fixed-point representation. It is expressed as:

$$f \cdot 2^e,$$

where e is the exponent and f is the fraction (mantissa) [18]. The most commonly used floating-point formats were originally established under IEEE Standard 754 [18].However, in order to reduce hardware complexity while maintaining dynamic range, AI developers are increasingly disregarding IEEE conformance in favor of proprietary formats that exclude or simplify costly features like rounding modes and normality support [16].

The traditional IEEE formats are summarized below:

- 16-bit Floating Point Half-Precision (FP16): 1 sign bit, 5 exponent bits, 10 fraction bits
- 32-bit Floating-Point Single-Precision (FP32): 1 sign bit, 8 exponent bits, 23 fraction bits
- 64-bit Floating-Point Double-Precision (FP64): 1 sign bit, 11 exponent bits, 52 fraction bits

Scientific notation, represented as f, allows floating-point values to bypass the dynamic range constraints of fixed-point representation. It is expressed as

$$\text{value} = (-1)^s \cdot 2^{(e-\text{bias})} \cdot (1.f)$$

where e is the exponent and f is the fraction (mantissa) [18]. The exponent is stored in biased form. The actual value of the exponent is obtained by subtracting a fixed value, called the bias, from the stored exponent. This bias approach simplifies comparison operations in hardware and removes the need for a separate sign bit for the exponent. The bias is 127 for single precision (FP32) and 1023 for double precision (FP64). It also supports exceptional arithmetic conditions through denormalized numbers, $\pm\infty$ (infinity), and NaN (not-a-number), providing robust handling of underflow and overflow conditions.

Floating Point Advantages and Limitations

More numerical flexibility and easier algorithm creation are made possible by floating-point arithmetic's large dynamic range and lack of explicit scaling control. Because of these benefits, floating-point formats are ideal for applications that need a large representational range and accuracy.

However, these benefits come with notable trade-offs. Floating-point units require more complex hardware, which uses more power, greater memory needs which depends on total bitwidth and slower performance when compared to fixed-point implementations. Because greater precision formats demand broader data channels, more logic resources, and higher power consumption, designers must carefully weigh numerical precision and dynamic range against hardware implementation costs.

2.2.2 Number representation considerations

Bit-Exact(Fixed-Point) Implementation for DFE systems

A crucial stage in the creation of hardware-efficient DPD and PCCFR systems is the translation of reference models from floating-point to bit-exact fixed-point implementations. This transformation needs careful consideration of quantization effects, overflow handling, and numerical precision trade-offs [15]. This transformation is here called fixed-point conversion.

Quantization effects

When a real-world value is represented in fixed-point format, quantization introduces an inaccuracy bounded by half the Least Significant Bit (LSB) value [15], where the LSB size is determined by selected word and fraction length. The fractional accuracy of an FP32 integer can be maintained with enough precision, demonstrating that quantization error depends on the selected precision rather than the representation format itself. EVM and ACPR in DPD systems can be impacted by quantization errors.

Overflow and Saturation

Fixed-point systems need to handle overflow on their own, either by limiting the value to the maximum allowed or by wrapping around to the other side, unlike floating-point systems which provide a range large enough to avoid it. This means that the range needs to be carefully checked during the conversion process.

Numerical Precision Trade-offs

The process of converting involves finding the right balance between how long each word is, which affects how precise the numbers are, how many fractions are used, which determines how fine the details can be, and the range, which needs to cover all the possible changes in the signal without it becoming too loud or distorted [15]. Fixed-point arithmetic is usually better for signal processing in embedded systems because it is cheaper and lets you use different sizes for numbers in the hardware.

Scaled Doubles as a Transition Tool

Scaled doubles use a mix method where values are stored as floating-point doubles but also keep track of the fixed-point scaling details [15]. They act as a middle stage in the conversion process, helping to spot overflow issues, analyze precision, and smoothly move parts of the system while keeping the whole system working properly.

2.3 Algorithms

Many common coefficient identification methods for MP and GMP models, often used to implement conventional DPDs, rely on either the Least Squares (LS) or the

Least-Mean-Square (LMS) algorithm. Both algorithms are explained in detail in the subsequent sections.

2.3.1 Least Squares

The LS method is an algorithm to fit a mathematical model $t \mapsto \varphi(t; \mathbf{x})$ to given data (t_i, b_i) with $i = 1, \ldots, m$, where the vector $\mathbf{x} = (x_1, \ \ldots, \ x_n)$ contains the $n < m$ parameters of φ. This is done by solving the LS problem, which is to minimize

$$f(\mathbf{x}) = \sum_{i=1}^{m} (\varphi(t_i; \mathbf{x}) - b_i)^2 = \sum_{i=1}^{m} r_i(\mathbf{x})^2 = \mathbf{r}(\mathbf{x})^T \mathbf{r}(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|^2,$$

where $r_i(\mathbf{x}) = \varphi(t_i; \mathbf{x}) - b_i$ with $i = 1, \ldots m$ are the residuals and $\mathbf{r}(\mathbf{x}) = (r_1(\mathbf{x}), \ \ldots, \ r_m(\mathbf{x}))$ the residual vector. If φ is linear in the parameters $\mathbf{x}$, the function can be written as

$$\varphi(t; \mathbf{x}) = x_1\alpha_1(t) + \cdots + x_n\alpha_n(t) = \sum_{j=1}^{n} \alpha_j(t) x_j = \alpha(t)^T \mathbf{x},$$

where $\alpha_j(t)$ with $j = 1, \ldots, n$ are the basis functions and $\alpha(t) = (\alpha_1(t) \ \ldots \ \alpha_n(t))$ is the vector of basis functions for t. The basis functions may be any nonlinear functions [19].

From this one can form the overdetermined system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where $\mathbf{A}$ has full rank, row i of the matrix $\mathbf{A}$ contains $\alpha(t_i)^T$, and row i of the vector $\mathbf{b}$ contains b_i. This gives an alternative formulation of the residual vector as $\mathbf{r}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \mathbf{b}$, and the LS problem can thus be reformulated as to minimize

$$f(\mathbf{x}) = \|\mathbf{r}(\mathbf{x})\|^2 = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 = \frac{1}{2}\mathbf{x}^T(2\mathbf{A}^T\mathbf{A})\mathbf{x} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}.$$

As this is a quadratic function with a positive definitive Hessian $2\mathbf{A}^T\mathbf{A}$, the minima is therefore obtained from

$$\mathbf{0} = \nabla f(\mathbf{x}) = 2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b},$$

equivalent to the normal equations

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b},$$

from which $\mathbf{x}$ can be uniquely determined given that $\mathbf{A}$ has full rank [19].

This derivation was made for a real-valued problem, [5] gives that for complex valued problems the LS solution that minimizes $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$ is instead given by the solution to the set of linear equations

$$\mathbf{A}^H\mathbf{A}\mathbf{x} = \mathbf{A}^H\mathbf{b}.$$

Here H denotes the Hermitian transpose given by $A^H = (A^*)^T = (A^T)^*$, also known as the complex conjugate transpose, where $*$ denotes the complex conjugate.

2.3.2 Least-Mean-Square Algorithm

The LMS algorithm is a stochastic gradient descent method that can be used for adaptive filtering. Here, a filter refers to an algorithm which processes a signal to give it desired properties. An adaptive filter is a filter with adjustable coefficients that can adapt to unknown or varying signal characteristics. The LMS algorithm can be used to update the filter coefficients of an adaptive filter on a sample-by-sample basis [20].

Let x be a possibly complex signal, to be passed trough a Finite-duration Impulse Response (FIR) filter of length M, for which the desired filter output is d. The actual filter output for the nth sample is computed as

$$\hat{d}(n) = \sum_{k=0}^{M-1} h_k x(n-k),$$

where $h_m \in C$, $m = 0, 1, \ldots, M-1$ are the filter coefficients and $n \in N$. This gives the estimation error $e(n) = d(n) - \hat{d}(n)$, and the Mean Squared Error (MSE) of the estimation error as a function of the filter coefficients becomes

$$\begin{aligned}
\varepsilon_M(\mathbf{h}) = \mathbb{E}[|e(n)|^2] &= \mathbb{E}\left[\left|d(n) - \sum_{k=0}^{M-1} h_k x(n-k)\right|^2\right] = \\
&= \mathbb{E}\left[|d(n)|^2 - 2\mathrm{Re}\left(\sum_{l=0}^{M-1} h_l^* d(n) x^*(n-l)\right) + \sum_{l=0}^{M-1}\sum_{k=0}^{M-1} h_l^* h_k x^*(n-l)x(n-k)\right] = \\
&= \mathbb{E}[|d(n)|^2] - 2\mathrm{Re}\left(\sum_{l=0}^{M-1} h_l^* \mathbb{E}[d(n)x^*(n-l)]\right) + \sum_{l=0}^{M-1}\sum_{k=0}^{M-1} h_l^* h_k \mathbb{E}[x^*(n-l)x(n-k)],
\end{aligned}$$

where $\mathbb{E}[\cdot]$ is the expected value and $\mathbf{h} = (h_0 \ \ldots \ h_{M-1})^T$ is the coefficient vector. The minima of this quadratic function can be found as the solution from the linear system of equations

$$\sum_{k=0}^{M-1} h_k \mathbb{E}[x(n-k)x^*(n-l)] = \mathbb{E}[d(n)x^*(n-l)], \ l = 0, 1, \ldots, M-1,$$

equivalent to the matrix formulation

$$\mathbb{E}_{\mathbf{xx}}\mathbf{h} = \mathbb{E}_{d\mathbf{x}},$$

where $\mathbb{E}_{\mathbf{xx}} = \mathbb{E}[\mathbf{x}^*(n)\mathbf{x}(n)^T]$ and $\mathbb{E}_{d\mathbf{x}} = \mathbb{E}[d(n)\mathbf{x}^*(n)]$, and $\mathbf{x}(n) = (x(n) \ x(n-1) \ \ldots \ x(n-(M-1)))^T$. From this system of equations the optimal parameters may be found iteratively by the use of a gradient method [21].

Assuming $\mathbb{E}_{\mathbf{xx}}$ and $\mathbb{E}_{d\mathbf{x}}$ are known, the filter coefficients can be updated using the gradient method steepest-descent according to

$$\mathbf{h}^{(n+1)} = \mathbf{h}^{(n)} - \frac{1}{2}\mu^{(n)}\mathbf{g}^{(n)}, \tag{2.14}$$

where $\mathbf{h}^{(n)}$ are the filter coefficients and $\mu^{(n)}$ is the step size of the nth iteration, and

$$\mathbf{g}^{(n)} = \nabla_{\mathbf{h}^{(n)}}\varepsilon_M = 2(\mathbb{E}_{\mathbf{xx}}\mathbf{h}^{(n)} - \mathbb{E}_{d\mathbf{x}}) \tag{2.15}$$

is the gradient of the MSE ε_M with respect to the coefficients at the nth iteration. If $\mathbb{E}_{\mathbf{xx}}$ and $\mathbb{E}_{d\mathbf{x}}$ are unknown, a stochastic steepest-descent algorithm can be used instead, replacing the gradient $\mathbf{g}^{(n)}$ with a gradient estimate $\hat{\mathbf{g}}^{(n)}$ in (2.14) [21].

The computation of the gradient in (2.15) may according to [20] be rewritten as

$$\mathbf{g}^{(n)} = -2\mathbb{E}[e(n)\mathbf{x}^*(n)], \tag{2.16}$$

from which an estimate may be obtained as

$$\hat{\mathbf{g}}^{(n)} = -2e(n)\mathbf{x}^*(n). \tag{2.17}$$

Furthermore, [20] states that in adaptive filtering it is common practice to use a constant update step size as it is easy to implement on hardware and software, and is appropriate when handling time-variant sequences, for which $\mu^{(n)}$ is replaced with μ in (2.14).

The LMS algorithm for updating the coefficients $\mathbf{h}^{(n)} = (h_0,\ h_1,\ \ldots,\ h_{M-1})^T$ of a filter is thus given by

$$\mathbf{h}^{(n+1)} = \mathbf{h}^{(n)} + \mu e(n)\mathbf{x}^*(n), \tag{2.18}$$

where $e(n) = d(n) - \hat{d}(n)$ is the error between the desired filter output and actual filter output, and $\mathbf{x}(n) = (x(n),\ x(n-1),\ \ldots,\ x(n-(M-1)))^T$ [20].

2.4 Reference System

The wireless communications system considered in this thesis is illustrated in Figure 2.1 and consists of a block chain with the blocks signal generator, CFR, HL, DPD, PA and antenna, where the CFR, HL and DPD are part of the DFE, and the PA and antenna are part of the AFE.

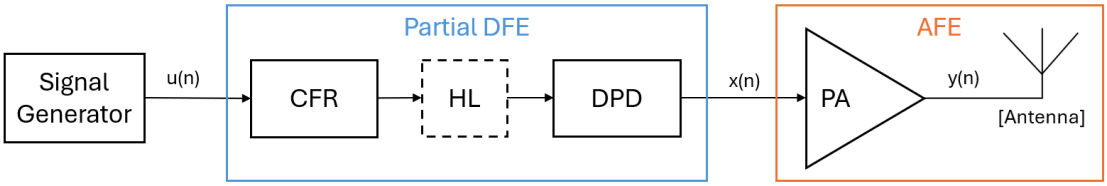


Figure 2.1: Block diagram of system.

As can be seen from the figure, the signal passes through the CFR, HL, DPD and PA, and is then transmitted through the antenna. To transmit a signal with sufficient power, the signal power is amplified using a PA. The PA is most efficient for large input powers for which it exhibits nonlinear behavior and compression, distorting the signal. To mitigate the nonlinear behavior while retaining the PA efficiency, the blocks of the DFE are applied. The signal $u(n)$ is passed through the DFE, where it first passes through the CFR and HL which lowers the PAPR of the signal, after which the output is passed through the DPD which predistorts the signal such that the PA output becomes linear. The DFE output $x(n)$ is then

passed through the PA which amplifies the signal power and the PA output $y(n)$ is then ready to be transmitted. The components in the DFE are implemented and optimized individually and as such the joint DFE may not be optimal as individual components may interfere with each other. A more detailed explanation of each of the blocks in the transmitter chain can be found in the subsequent sections.

2.4.1 Signal

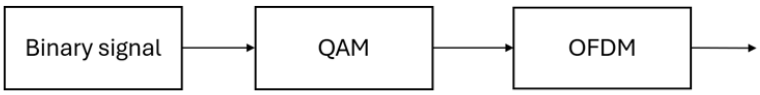


Figure 2.2: Schematic diagram of signal generation.

A signal is a modulated representation of information to be sent and received, which occupies a range of frequencies referred to as the bandwidth or channel of the signal [22]. In-band distortion of the signal refers to distortion of the signal at frequencies within the signal channel and out-of-band spectral regrowth refers to distortion of the signal outside the signal channel, generally in terms of adjacent channel leakage. In this thesis Orthogonal Frequency Division Multiplexing (OFDM) signals with binary signal modulated using Quadrature Amplitude Modulation (QAM) are used. A schematic diagram of the construction of such a signal can be seen in Figure 2.2. A more detailed explanation of binary signals, QAM and OFDM, as well as an example of such a signal, can be found in the subsequent sections.

Binary Signal

A binary signal is a binary representation of the information, or data, to be communicated through a communication system. In this thesis a wireless communication system is considered. The binary signal can also be referred to as a bitstream as it can be seen as a stream of bits, or binary digits.

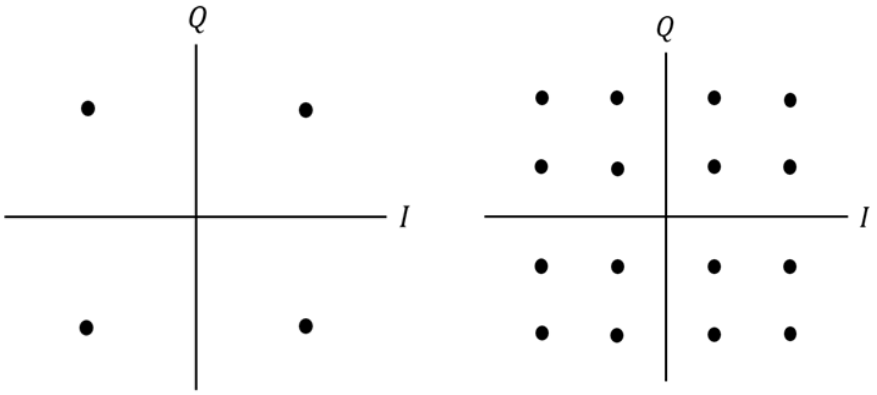
Quadrature Amplitude Modulation

QAM is a widely used digital modulation technique. QAM modulates a binary signal by converting it to a multilevel digital signal and modifies amplitude and phase of the signal simultaneously [22]. The QAM technique allows higher data rates, i.e. bits transmitted per time interval, within the same bandwidth compared to for the binary signal, giving higher spectral efficiency. Spectral efficiency is a measurement of how much data can be transmitted per time unit for a given bandwidth, where a high data rate corresponds to good spectral efficiency. M-QAM refers to a QAM with M symbols, where a symbol is a unique combination of phase and amplitude for a sample of a modulated signal [23].

The simplest form of QAM has 4 unique symbols and transmits $2 = \log_2(4)$ bits per time interval. It is commonly known as Quadrature Phase Shift Keying (QPSK).

The technique divides the serial bitstream and converts it into two parallel bitstreams containing every other bit, the In-phase (I) and Quadrature (Q) bitstreams (the parallel bitstreams each process one bit per time interval, i.e. half the data rate compared to the serial bitstream). The bitstreams are applied to mixers along with a local oscillator, that is shifted 90' for the Q-bitstream, after which the outputs are added giving the modulated signal [23]. The QPSK signal can be illustrated using a constellation diagram, where each symbol represents a phase amplitude pair [22]. Such a constellation diagram can be seen in Figure 2.3a.

QAM can be modified to process a higher number of bits per time interval such as the 16-QAM, in which 16 refers to the number of distinct symbols, and transmits $4 = \log_2(16)$ bits per time interval giving 16 distinct symbols [22]. A constellation diagram of a signal modulated using 16-QAM can be seen in Figure 2.3b.



(a) QPSK constellation diagram **(b)** 16-QAM constellation diagram

Figure 2.3: Constellation diagrams of signals modulated using QPSK and 16-QAM respectively.

Other higher modulation order QAMs are the 64- and 256-QAM, which transmit 6 and 8 bits per time interval respectively, as well as the 1024-QAM [23]. Higher level QAMs are more spectrally efficient, but also more sensitive to noise as the noise makes it harder to differentiate between symbols [22].

Orthogonal Frequency Division Multiplexing

OFDM is a widely used modulation method, used in many wireless communication system [22], and the waveform used in 4G and 5G is based on OFDM [24], where 4G is synonymous with Long-Term Evolution (LTE) and 5G is synonymous with New Radio (NR) [23]. OFDM is what is known as a broadband modulation method which means that the signal is spread over a wide range of frequencies, i.e. a broad band of frequencies [23]. It is a modulation method with high data rates and high spectral efficiency, as well as high resistance to noise [22]. On the other hand, OFDM is a modulation method resulting in signals with high PAPR [25]. Often a signal modulated using OFDM is referred to as an OFDM signal.

In OFDM the signal is divided into multiple subcarriers to which the Inverse Fast Fourier Transform (IFFT) is applied, converting the signal from the RF domain to the time domain. Applying the IFFT assures orthogonality between the subcarriers as the sinusoids of the Fourier Transform (FT) are orthogonal [25]. The Discrete Fourier Transform (DFT) and more specifically the Inverse Discrete Fourier Transform (IDFT) of length N are given by

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-i2\pi kn/N},\ k = 0, 1, \ldots, N-1 \tag{2.19}$$

and

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{i2\pi kn/N},\ n = 0, 1, \ldots, N-1 \tag{2.20}$$

respectively, where $X(k)$ denotes the signal in the frequency domain and $x(n)$ denotes the signal in the time domain, respectively. The Fast Fourier Transform (FFT) and IFFT are computationally efficient implementations of the DFT and IDFT respectively, and reduce the number of complex multiplications from N^2 to $N \log_2 N$ when N is a power of 2. In the case where the sequence length is not a power of 2, the sequence can be extended by padding it with zeros to get the length as a power of 2 [21].

OFDM signals have strong spectral side-lobes, as such a guard band is reserved between adjacent channels to reduce adjacent channel interference [26]. In practice this means inserting a number of unused subcarriers at the channel edges, called guard band carriers, which corresponds to zero padding the subcarrier sequence before applying the IFFT when generating the OFDM signal.

OFDM signals used in many practical applications are oversampled, which means that the sample frequency is greater than the Nyquist rate [27]. The Nyquist rate is the minimal sample frequency of an analog signal that allows the signal to be reconstructed exactly, and is given by

$$F_N = 2 \cdot F_{max},$$

where F_{max} is the highest frequency contained in the analog signal [28]. An OFDM signal is generally oversampled by applying an IFFT of greater length during the signal modulation and zero pad any unused subcarriers. To oversample the signal with oversampling rate L, and IFFT of length $L \cdot N$ would be applied [27].

Inter-Symbol Interference (ISI) refers to distortion being introduced to the transmitted symbol caused by the previously transmitted symbol, which may occur if the OFDM symbol length is greater than the channel length. An OFDM system with N subcarriers will have a bit rate N times lower and a period N times higher compared to a single carrier system, which for large N will cause the OFDM symbol length to be greater than the channel length causing multipath distortion. To prevent ISI from multipath distortion, a guard interval is inserted between symbols, commonly a Cyclic Prefix (CP). Applying a CP to a symbol means that the end of the symbol is added at the beginning of the symbol, where a longer CP is more efficient at preventing ISI but also results in greater loss of power and requires additional bandwidth [25].

Example

Let us consider an OFDM signal modulated using 256-QAM with bandwidth 100 MHz, generated in accordance with the method described in Section 4.1.1 with the specifications and attributes specified in Tables 4.1 and 4.2 in Section 4.2.1. The envelope, constellation diagram and spectrum of an example of such a signal can be found in Figures 2.4, 2.5 and 2.6 respectively.

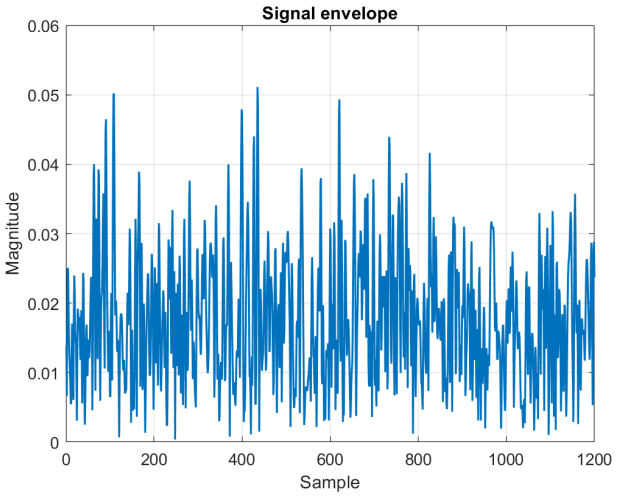


Figure 2.4: Envelope of first 1200 signal samples.

Figure 2.4 illustrates the envelope of the first 1200 samples of the signal. The envelope of a signal is the signal magnitude, and is often used to illustrate complex signals.

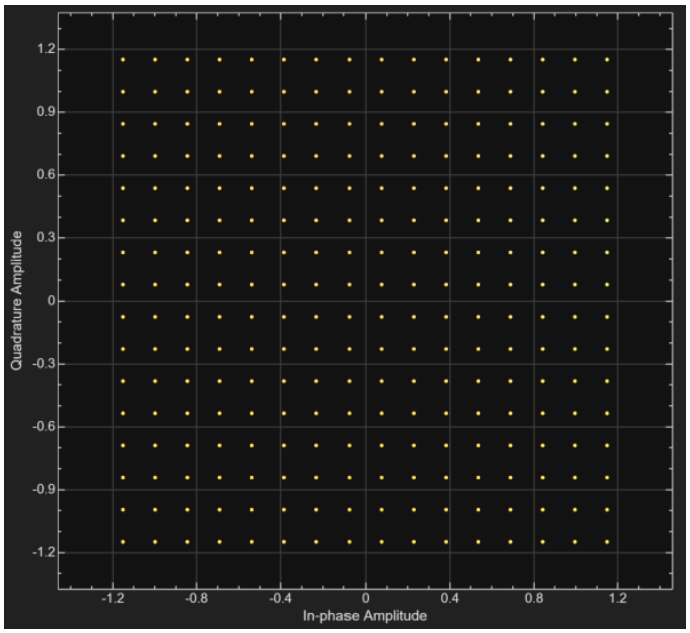


Figure 2.5: Constellation diagram of signal.

In Figure 2.5 the constellation diagram of the signal is illustrated. The signal was modulated using 256-QAM and as such the constellation diagram has 256 unique symbols ordered in a 16×16 grid with the I and Q components on the axes.

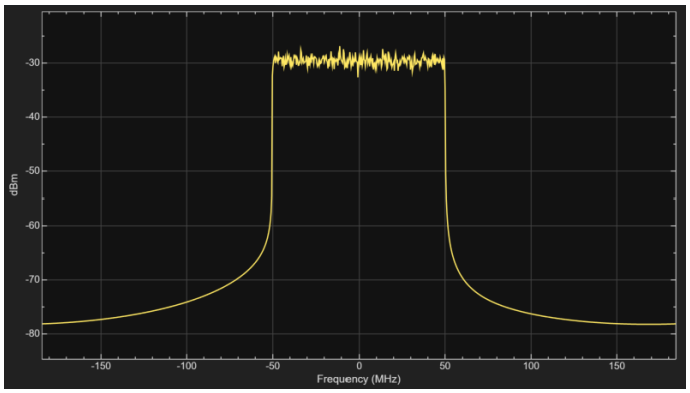


Figure 2.6: Spectrum of signal.

Figure 2.6 displays the spectrum of the signal. From the figure it can be seen that the signal mainly occupies the frequencies in the range -50 to 50 MHz giving the main channel bandwidth 100 MHz. From the figure it can also be seen that there is some signal leakage into adjacent channels.

2.4.2 Crest Factor Reduction

CFR is a signal processing technique used in modern wireless systems to reduce the high PAPR inherent in multi-carrier schemes like OFDM [7]. High PAPR forces PAs to operate with large back-off from saturation, reducing power efficiency [29]. CFR reduces peaks prior to amplification, enabling the PA to operate closer to saturation while maintaining linearity, at the cost of introduced distortion (spectral regrowth and increased EVM). The fundamental trade-off lies between PAPR reduction and signal fidelity, with achievable reduction depending on the standard's spectral mask and EVM requirements.

CFR techniques can be broadly categorized as distortion-based or distortionless. The main approaches are summarized in Table 2.1, with key methods discussed in the subsequent sections.

CFR Technique	**Principle**	**PAPR Reduction**	**EVM Impact**	**ACPR Impact**	**Complexity**	**Key Trade-off / Limitation**
Hard Clipping (HC)	Truncates samples above threshold	High	Moderate–High	Severe degradation	Very Low	Sharp transitions cause broadband spectral leakage.
Clip & Filter (CAF)	Clips and filters correction signal	Moderate–High	Moderate	Good (filter-dependent)	Low–Moderate	Peak regrowth requires iterations; filter length vs. EVM trade-off.
Constrained Clipping	Separately enforces in-band EVM and out-of-band spectral masks	Moderate	Controlled (threshold-based)	Excellent	Moderate	Requires per-subcarrier power budgets and standard-specific tuning.
Peak Cancellation (PCCFR)	Subtracts spectrally-shaped pulses at peak locations	High (∼3.75 dB)	Low (∼7.6%)	Excellent (∼86.6 dBc)	Moderate	Pulse design is critical; interpolation may be required for 1× sampling.
Distortionless (PTS/SLM)	Selects the best representation among multiple transforms	Moderate	None (no distortion)	None (spectrally clean)	High	Requires side information and multiple IFFT operations.

Table 2.1: Comparison of common CFR techniques

Hard Clipping

Hard Clipping (HC) is the most elementary method, truncating samples exceeding a threshold $A_{\max}$:

$$x_{\mathrm{HC}}(n) = \begin{cases} x(n), & |x(n)| \le A_{\max}, \\ A_{\max} e^{i\angle x(n)}, & |x(n)| > A_{\max}, \end{cases} \tag{2.21}$$

where $\angle x(n)$ is the angular argument of the sample $x(n)$. While simple, it introduces sharp time-domain transitions that generate broadband spectral leakage, severely degrading the ACPR [30].

Clip and Filter (CAF)

Clip and Filter (CAF) improves on hard clipping by filtering the difference (correction) signal:

$$\begin{aligned} c(n) &= x(n) - x_{\mathrm{HC}}(n), \\ x_{\mathrm{CAF}}(n) &= x(n) - \mathrm{filter}(c(n)) \,, \end{aligned} \tag{2.22}$$

which smooths sharp edges and reduces spectral regrowth (the expansion of signal bandwidth due to nonlinear distortion). Multiple iterations, known as Iterative CAF, are often needed to mitigate peak regrowth, where new peaks appear after filtering. Longer filters improve ACPR at the expense of higher EVM [29].

Constrained Clipping

Constrained Clipping separately enforces in-band EVM limits (distortion within the allocated channel bandwidth) and out-of-band spectral masks (regulatory limits

on emissions outside the channel). Out-of-band subcarriers are clipped based on individual power budgets:

$$\tilde{X}_k = \begin{cases} \tilde{X}_k, & |\tilde{X}_k|^2 \leq P_k, \\ \sqrt{P_k}\, e^{j\angle \tilde{X}_k}, & |\tilde{X}_k|^2 > P_k, \end{cases} \qquad k \in \mathcal{O}, \tag{2.23}$$

where P_k is the power budget of subcarrier k and $\mathcal{O}$ denotes the set of out-of-band subcarriers [31].

Peak Cancellation CFR (PCCFR) PCCFR subtracts spectrally shaped cancellation pulses at peak locations rather than HC:

$$x_{\text{out}}(n) = x_{\text{in}}(n) - \sum_{k=1}^{K} \alpha_k\, p(n - n_k), \tag{2.24}$$

where α_k is the scaling factor and $p(n)$ is the cancellation pulse. The pulse is designed to match the signal's frequency content, minimizing out-of-band interference. Enhanced variants improve peak detection and reduce computational complexity [29].

Distortionless Methods

Distortionless techniques, such as Partial Transmit Sequence (PTS) and Selective Mapping (SLM), reduce PAPR without introducing EVM degradation or spectral regrowth by generating multiple equivalent signal representations and selecting the one with the lowest PAPR. However, they require side-information transmission to the receiver and incur high computational complexity due to multiple IFFT operations [7].

2.4.3 Hard Limiter

The purpose of the HL is to further reduce the PAPR of the signal after the CFR, by reducing any remaining peaks. The HC CFR method defined by Equation (2.21) is according to [32] sometimes also referred to as hard limiting. In many papers, such as [7], [29] and [32], either a HC or another CFR method is used. Other papers such as [8] suggest using some CFR method and a separate model HL using HC to reduce the magnitude of any remaining samples whose magnitude exceeds the clipping threshold of the CFR. This is also done in many practical applications such as in [33], where a separate CFR and HL are used in the reference design to reduce the PAPR of the signal.

2.4.4 Digital Pre-Distortion

DPD background

The purpose of the DPD is to linearize the PA to achieve high linearity and high efficiency, by allowing the PA to operate near saturation [2]. The DPD does this

by predistorting the signal to compensate for PA nonlinearities and memory effects, approachning a linear relationship between DPD input and PA output, i.e.

$$y(n) = f_{PA}(f_{DPD}(x(n))) = G \cdot x(n),$$

where f_{PA} is the PA, f_{DPD} is the DPD and G is slope of the linear curve referred to as the linear gain of the system [34]. A PA with high linearity refers to a PA for which there is a close to linear relationship between the system input and system output.

It is common to model the DPD using simplified Volterra series based models [35]. The Volterra series describes the most general form of nonlinearity with memory considerations and is in discrete time is given by

$$f_V(x(n)) = \sum_{k=1}^{K} f_k(x(n)),\ f_k(x(n)) = \sum_{m_1=0}^{M-1} \dots \sum_{m_k=0}^{M-1} h_k(m_1, \dots, m_k) \prod_{l=1}^{k} x(n - m_l). \tag{2.25}$$

It is a sum of multidimensional convolutions capable of describing nonlinearities and memory effects, where M is the memory depth and h_k is called the Volterra kernel [5]. This is a highly complex model, not feasible to implement in a resource constrained environment, due to which DPD is commonly modeled using a simplified Volterra based model instead.

One simplified Volterra series based model is the MP model given by

$$f_{MP}(x(n)) = \sum_{k=0}^{K} \sum_{m=0}^{M} a_{km} x(n-m)|x(n-m)|^k, \tag{2.26}$$

where a_{km} are the coefficients and $|x(n)|$ is referred to as the envelope of $x(n)$. The parameters K and M are referred to as the envelope order and the memory depth respectively [36]. Sometimes K may also be referred to as the polynomial order or the nonlinearity order as it controls the order of nonlinearities of the MP model. In the MP model, the coefficients a_{km} replaces the Volterra kernels $h_k(m_1, \dots M_k)$, a single sum replaces the multiple summations over memory and the terms $x(n-m)|x(n-m)|^k$ replaces the product. The MP model can capture the nonlinearities of the PA by using a polynomial to model the inverse of the nonlinear gain-compensated PA, and uses delayed samples of the input to account for the memory effects of the PA.

Another simplified Volterra series based model is the GMP model, which generalizes the MP model by including additional leading and lagging cross-terms given by $x(n-m)|x(n-m-l)|^k$ and $x(n-m)|x(n-m+l)|^k$ respectively. The GMP model is able to capture more complex dynamic behavior of the PA compared to the MP model, but is also more computationally complex. Other examples of simplified Volterra series based models used for DPD modeling are Wiener, Hammerstein, Wiener-Hammerstein and parallel Wiener [5]. In this thesis, the MP model is used to model DPD.

The identification of the coefficients of models derived from the Volterra series can be solved as a linear problem, by employing the LS method [2]. The LS method is

described in detail in Section 2.3.1, and in the following it will be explained how the method can be used to identify DPD model coefficients.

The DPD model is linear in parameters and can be written in matrix form as

$$f(x(n)) = \phi_u^T \beta,$$

where ϕ_u is the vector of basis functions for the input $\mathbf{u}(n) = (u(n)\, u(n-1)\, \ldots u(n-M))$ with memory depth M, and β are the coefficients. In the case where the DPD is modeled using a MP, the basis functions will be $u(n-m)|u(n-m)|^k$ and the corresponding coefficients a_{km} for $m = 0, 1, \ldots, M$ and $k = 0, 1, \ldots, K$. From this the system of equations

$$\mathbf{x} = \Phi_u \beta \tag{2.27}$$

can be formed, where row i of the matrix Φ_u consists of the vector of basis functions for the input $\mathbf{u}(i)$ corresponding to the output $x(i)$, and β are the coefficients.

The coefficients β can be identified using Direct Learning Architecture (DLA) or Indirect Learning Architecture (ILA) [2]. A schematic block diagram of both identification methods can be seen in Figure 2.7.

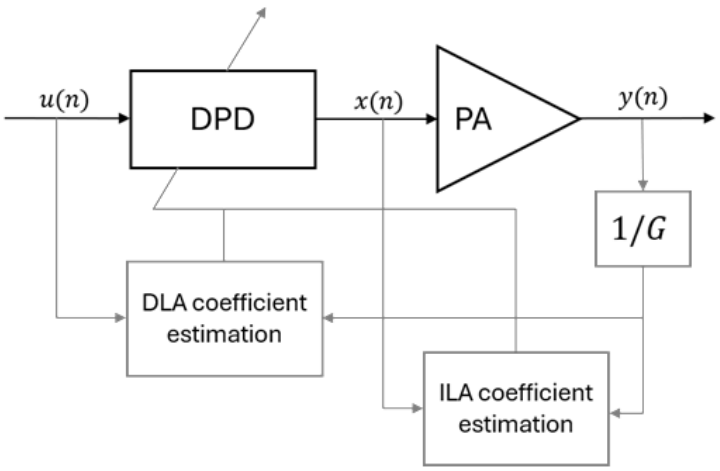


Figure 2.7: Block diagram of DPD adaptation with DLA and ILA coefficient estimation.

The gray arrows in the figure to the DLA and ILA coefficient estimation blocks represents the flow of information in regards to the DLA and ILA coefficient estimation of the DPD model. And the gray arrows in the figure going from the DLA and ILA coefficient estimation blocks and through the DPD represents the update of the DPD model coefficients. As can be seen from the figure, DLA uses DPD input $u(n)$ and gain-compensated PA output $y(n)/G$ to identify the coefficients, whereas ILA uses PA input $x(n)$ and gain-compensated PA output $y(n)/G$ to identify the coefficients.

For DLA the coefficients β in 2.27 are estimated iteratively according to

$$\beta_i = \beta_{i-1} - \mu \Delta\beta, \tag{2.28}$$

where $\mu < 1$ is the damping factor and $\Delta\beta$ is computed using the DPD input and gain-compensated PA output by minimizing $\|\varepsilon - \Phi_u \Delta\beta\|^2$ with $\varepsilon = \mathbf{y}/G - \mathbf{u}$ [2]. This has the solution

$$\Delta\beta = (\Phi_u^H \Phi_u)^{-1} \Phi_u^H \varepsilon. \tag{2.29}$$

and is computed using the LS method for complex valued problems described in Section 2.3.1.

For ILA the coefficients are estimated using the PA input and gain-compensated PA output to model the inverse of the PA with gain compensation and use those coefficients for the DPD model. The system of equations for the inverse of the gain-compensated PA is given by

$$\mathbf{x} = \Phi_{y/G} \beta,$$

where $\Phi_{y/G}$ is constructed in the same manner as in (2.27) but using the gain-compensated PA output instead of the DPD inputs to construct the basis functions. The coefficient identification is done by minimizing $\|\mathbf{x} - \Phi_{y/G}\beta\|^2$ [2]. This has the solution

$$\beta = (\Phi_{y/G}^H \Phi_{y/G})^{-1} \Phi_{y/G} \mathbf{x}.$$

and is again computed using the LS method for complex valued problems described in Section 2.3.1. The ILA method presented in [2] can be replaced by an iterative method according to (2.28) by instead minimizing $\|\varepsilon - \Phi_{y/G}\Delta\beta\|^2$ with $\varepsilon = \mathbf{x} - \Phi_{y/G}\beta$, giving the LS solution

$$\Delta\beta = (\Phi_{y/G}^H \Phi_{y/G})^{-1} \Phi_{y/G} \varepsilon \tag{2.30}$$

equivalent to the iterative coefficient update for the inverse modeling of a PA described in [5] with added PA output gain compensation.

For this thesis the DLA and ILA coefficient estimation will refer to the iterative methods according to Equation (2.28), where $\Delta\beta$ is computed according to Equations (2.29) and (2.30) for DLA and ILA respectively.

Non-polyphase and polyphase DPD feed-forward architecture

When implementing the DPD on hardware, there are a few things to consider. One such thing is whether to use non-polyphase or polyphase DPD feed-forward architecture. Here, non-polyphase and polyphase feed-forward architectures refers to whether to process the input signal sample stream serially or to divide it into multiple signal sample streams. This will be explained in detail for a DPD modeled using MP.

For hardware implementation it is practical to rewrite the MP model as

$$\begin{aligned} x(n) = f_{DPD}(u(n)) &= \sum_{m=0}^{M} u(n-m) \left(\sum_{k=0}^{K} a_{km} |u(n-m)|^k \right) \\ &= \sum_{m=0}^{M} u(n-m) h_m(|u(n-m)|), \end{aligned} \tag{2.31}$$

where

$$h_m(x) = \sum_{k=0}^{K} a_{km} x^k \tag{2.32}$$

are referred to as the coefficients of the architecture [36]. This model is illustrated as a schematic block diagram of hardware implementation using a single feed-forward path in Figure 2.8. In the figure the blocks labeled Z^{-1} refers to a delay in the stream of signal samples, the blocks marked with X to a multiplication and the block marked $\sum$ to a summation.

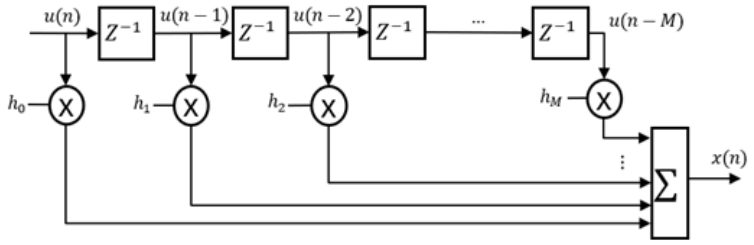


Figure 2.8: Schematic diagram of non-polyphase feed-forward path.

The timing of sequential operations in a digital system is dictated by a clock signal whose frequency is called the clock frequency. To ensure correct timing of all sequential operations in the digital system there is a minimum clock signal period, and hence a maximum clock frequency of the system [37]. An important aspect of the hardware implementation of the DPD is the required clock frequency. A wider signal bandwidth means a higher DPD sample frequency and thus a higher DPD clock frequency. To prevent the required DPD clock rate to exceed the hardware maximum clock frequency, it is possible to divide the single feed-forward path in Figure 2.8 into multiple parallel feed-forward paths running at a lower clock rate. The case with a single feed-forward path is called a non-polyphase DPD feed-forward architecture and the case with multiple parallel feed-forward paths is called a polyphase DPD feed-forward architecture [36].

Consider the polyphase DPD case with dual feed-forward paths. The input will pass through a deserializer which separates the samples into two separate feed-forward delay chains with even and odd samples, after which they are recombined into the DPD output signal after passing through a serializer. The two feed-forward chains will run at half clock rate as such reducing the clock rate of the DPD, and as each delay chain contains every other sample there will be some information exchange between the two feed-forward chains [36]. A schematic block diagram showing dual feed-forward delay chain parallelization can be seen in Figure 2.9 and a schematic block diagram of a dual polyphase DPD feed-forward architecture can be seen in Figure 2.10.

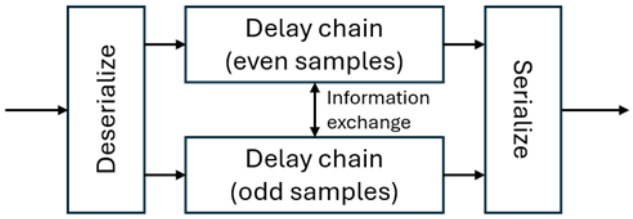


Figure 2.9: Schematic block diagram of dual polyphase delay chain parallelization.

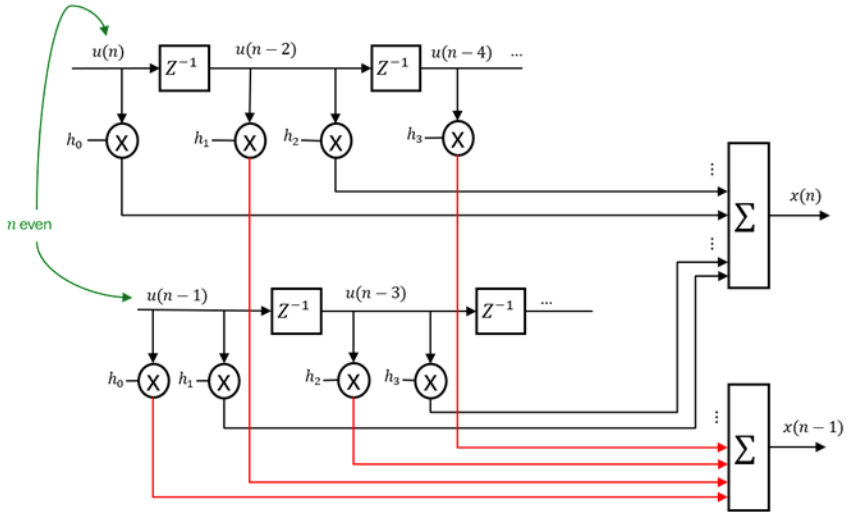


Figure 2.10: Schematic block diagram of dual polyphase feed-forward path.

In Figure 2.10 two parallel feed-forward chains can be seen where the upper one processes all even samples and the lower one processes all odd samples. Both chains has the same coefficients h_k and at the summations in the end there is some information exchange between the two chains. Mathematically this is equivalent to,

$$x(n) = u(n)\cdot h_0 + u(n-1)\cdot h_1 + u(n-2)\cdot h_2 + u(n-3)\cdot h_3 + \dots,$$
$$x(n-1) = u(n-1)\cdot h_0 + u((n-1)-1)\cdot h_1 + u((n-1)-2)\cdot h_2 + u((n-1)-3)\cdot h_3 + \dots,$$

where black means that the term is from the upper chain and blue means that the term is from the lower chain.

As can be seen when comparing the non-polyphase and dual polyphase DPD feed-forward architectures in Figures 2.8 and 2.10, the amount of required resources for the dual polyphase case becomes about twice the amount of required resources for the non-polyphase case, as the the dual polyphase case uses duplicates of the coefficients h_k and the summation blocks, as well as requiring deserialization and serialization blocks. So using a dual polyphase DPD feed-forward architecture means decreasing the clock rate by half but also doubling the amount of required resources compared to using a non-polyphase DPD feed-forward architecture.

For both the non-polyphase and polyphase DPD feed-forward architectures, the coefficients h_m can be computed either through the use of direct calculations with Digital Signal Processors (DSPs) or through the use of LUTs [36]. These two methods will be explained in the subsequent sections.

Direct calculations

Using direct calculations to compute the coefficients, h_m, means computing the coefficient values $h_m(x)$ according to Equation (2.32) in real time each time they are used in the DPD feed-forward path, generally this computation is done with DSPs. To make the computations more efficient for hardware implementation, the coefficient equation is rewritten as

$$h(x) = \sum_{k=0}^{K} a_{km} x^k = a_{0,m} + x \cdot (a_{1,m} + x \cdot (a_{2,m} + x \cdot (\dots (a_{K-1,m} + x \cdot a_K)))) \quad (2.33)$$

which can be implemented using hardware basic unit operation addition and multiplication [36]. This is efficient as only K multiplications and K additions are required in total compared to the k multiplications needed to compute each term $a_{km}x^k$ in the sum adding up to $K!$ multiplications and K additions.

The identification/update of the coefficients a_{km} can be done as described in the DPD background section about the coefficient identification by using the LS method with DLA or ILA iteratively in accordance with Equation (2.28) where $\Delta\beta$ is computed according to Equation (2.29) for DLA and Equation (2.30) for ILA.

Another way to update the coefficients a_{km} is by using the LMS method as described in Section 2.3.2, again with either DLA or ILA. For DLA the DPD input, $u(n)$, and gain-compensated PA output, $y(n)/G$, is used to estimate the coefficients, where the desired gain-compensated PA output is $d(n) = u(n)$, the estimated, or actual, gain-compensated PA output is $\hat{d}(n) = y(n)/G$ and the update input is $u(n-m)^k$ for coefficient a_{km}. This gives the coefficient update of a_{km} using DLA as

$$a_{km}^{\text{new}} = a_{km}^{\text{old}} + \mu e(n) u^*(n-m)^k, \quad (2.34)$$

where $e(n) = u(n) - y(n)/G$. For ILA, the gain-compensated PA output, $y(n)/G$, is used as input and PA input, $x(n)$ as output when computing the coefficients. This corresponds to the desired gain-compensated output $d(n) = x(n)$, estimated PA output $\hat{d}(n) = \hat{x}(n)$, where $\hat{x}(n) = f_{DPD}(y(n)/G)$ and the update input is $y(n)/G$ for coefficient a_{km}. This gives the coefficient update of a_{km} using ILA as

$$a_{km}^{\text{new}} = a_{km}^{\text{old}} + \mu e(n) y^*(n-m)^k / G, \quad (2.35)$$

where $e(n) = x(n) - \hat{x}(n)$.

Look-Up Table

A LUT is a structure which replaces repeated calculations by mapping input values to output values according to precomputed calculations made for a selected set of

inputs. This structure can be implemented both on software and as a hardware component, and has the purpose of saving time in exchange for additional memory requirements [38]. When using a LUT to replace a computation $y = f(\mathbf{x})$, the expected range of the inputs $\mathbf{x}$ is divided into bins, where each bin is given an address i, sometimes referred to as index, and an output y_i which approximates the output for inputs mapped to that address or index. How good this approximation is depends on how finely the inputs are divided into the bins, how the bins are distributed and if any interpolation strategies are used.

LUTs are a popular technique used to implement DPD [39], and are widely used [6], as it can replace computationally expensive run-time calculations with array indexing operations [40]. When using LUTs to implement DPD on hardware the number of required multipliers is reduced as less multiplications are needed. As such there is more flexibility in regards to the LUT polynomial order and the coefficient identification has low latency. On the other hand the number of required memory hardware components also increases [36]. When implementing DPD using LUTs, [41] highlights the need to consider LUT architecture, LUT sizing as well as LUT indexing and spacing. Another thing that should be considered according to [39] is the interpolation strategy used.

A common low complexity LUT architecture used to implement DPD is a 1D LUT with complex values that uses either the input power or input magnitude as LUT input [41] [36]. Other options include using a 2D LUT with complex values and the real and imaginary part of the input as LUT input, and using two 1D LUTs mapping the magnitude and phase distortion separately. The size of the LUT, i.e. the number of LUT entries, is directly correlated with the performance. A larger LUT size means better performance, but also slower convergence when computing the LUT entries [41]. The performance when using a small LUT size can be improved be using an interpolation technique [39], or by using a well chosen companding function. A companding function is a function which divides the spacing of input levels into different bins, each corresponding to a LUT index. The spacing can be uniform of non-uniform, where non-uniform spacing means that more memory can be converted to regions of interest [41].

Based on the LUT input a corresponding address/index is generated according to the LUT spacing and the best matching LUT input level, and the corresponding output level is returned [36]. The case when also applying interpolation is a bit more complex and will not be explored further in this thesis. A schematic block diagram of a 1D LUT with complex values and using the input magnitude to space the LUT and computing the index/ address can be seen in Figure 2.11.

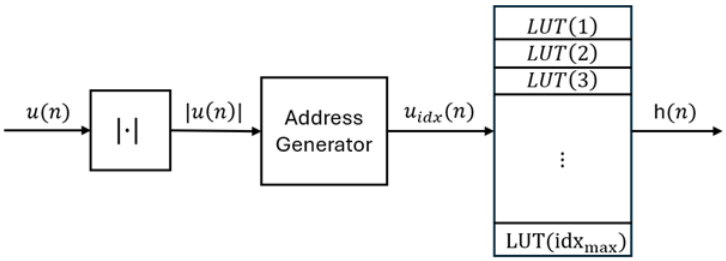


Figure 2.11: Schematic block diagram of a 1D LUT with complex values using input magnitude to compute the address/index.

There are two main ways of computing the LUT values of the DPD: external adaptation and runtime adaptation. For external adaptation, data is collected by an external computer such as a computer that identifies the DPD model coefficients using the LS method from which the LUT values are computed and updated. For runtime adaptation, the LUT values are updated iteratively by the system at runtime, typically using the LMS algorithm [41].

When applying external adaptation, the LS method can be used to compute the model coefficients in the same way as described in the DPD background section regarding coefficient identification, where either a DLA or ILA can be used to identify the coefficients. This corresponds to iteratively updating the coefficients according to Equation (2.28) with $\Delta\beta$ computed according to Equation (2.29) for DLA and (2.30) for ILA. Once the coefficients are identified, the LUT values can be computed according to (2.32).

When applying real time adaptation, the LMS method can be used to update the LUT values iteratively. The method is described in detail in Section (2.3.2). Again either DLA or ILA can be used. For DLA, the DPD input, $u(n)$, and gain-compensated PA output, $y(n)/G$, is used to estimate the coefficients. The desired gain-compensated PA output is $d(n) = u(n)$, the estimated, or actual, gain-compensated PA output is $\hat{d}(n) = y(n)/G$ and the update input is $u(n-m)$ for LUT m. This gives the LMS update of LUT m using DLA as

$$h_m^{\text{new}}(|u(n-m)|) = h_m^{\text{old}}(|u(n-m)|) + \mu e(n)u^*(n-m), \tag{2.36}$$

where $e(n) = u(n) - y(n)/G$. For ILA, the gain-compensated PA output, $y(n)/G$, is used as input and PA input, $x(n)$ as output when computing the coefficients. This corresponds to the desired gain-compensated output $d(n) = x(n)$, estimated PA output $\hat{d}(n) = \hat{x}(n)$, where $\hat{x}(n) = f_{DPD}(y(n)/G)$ and the update input is $y(n)/G$ for LUT m. This gives the LMS update of LUT m using ILA as

$$h_m^{\text{new}}(|y(n-m)/G|) = h_m^{\text{old}}(|y(n-m)/G|) + \mu e(n)y^*(n-m)/G, \tag{2.37}$$

where $e(n) = x(n) - \hat{x}(n)$.

An important hardware implementation note is that there exists hardware LUTs with dual input and output ports meaning that one LUT can be shared between

two parallel DPD feed-forward delay chains [36]. This means that the doubling of resources between polyphase and non-polyphase DPD feed-forward architectures mentioned above is not necessarily true when using LUTs to compute the coefficients. There will still be an increase in resource usage, but not as high.

2.4.5 Power Amplifier

PA background

The purpose of the PA is to amplify the magnitude of a signal. It is a hardware component part of the AFE made from transistors exhibiting nonlinear characteristics for high input powers leading to signal distortions [1]. The PA is a power hungry component so it should be run as efficiently as possible to save resources. The PA is most efficient near its saturation zone, but has severely nonlinear characteristics for operating points in this region [2]. Ideally, the PA would amplify the input signal linearly until saturation giving minimal signal distortions, however it exhibits nonlinear characteristics which are explained in greater detail in the next section.

PAs may also exhibit memory effects and/or be subject to time varying effects. The PA exhibiting memory effects means that the PA output depends on current and past inputs, which happens when the PA characteristics is frequency dependent. The PA being subject to time varying effects means that outside factors such as temperature, electrical variations and aging of the PA hardware component may affect the PA characteristics over time [1]. Most PAs implemented on hardware exhibits memory effects and are subject to time varying effects.

As previously explained, the DFE is applied prior to the PA, to address the trade-off between PA efficiency and linearity, enabling the PA to run with high efficiency and high linearity. The CFR is applied to reduce the PAPR of the input signal, which minimizes the required amount of signal back-off and improves PA efficiency, and the DPD is popular technique used to linearize the PA, accounting for both PA nonlinearities and memory effects [1].

PA characteristics

As stated, the PA exhibits nonlinear characteristics for high input powers which introduces signal distortions. To illustrate this the Amplitude-to-Amplitude (AM/AM) and Amplitude-to-Phase (AM/PM) distortions of a PA applied to a signal x scaled such that $|x| \in [0, 1]$ has been plotted in Figure 2.12. The PA is modeled with the modified Rapp model, which is to be described in the PA model section, and has the parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$, the same parameters used in [1] from which the model is taken, and the signal used is the example signal from Section 2.4.1, which is an OFDM signal modulated using 256-QAM with bandwidth 100 MHz.

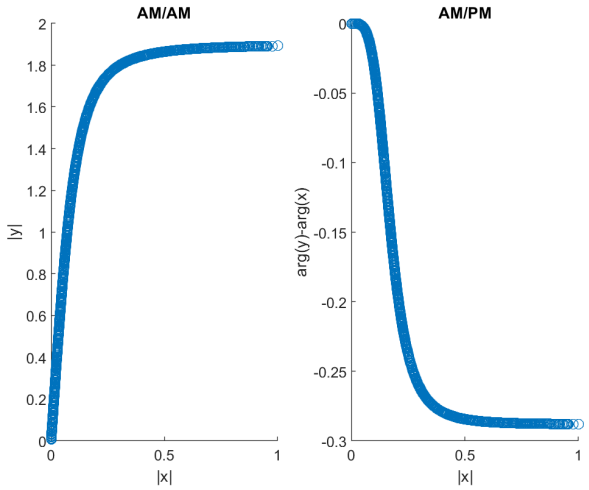


Figure 2.12: Amplitude and phase distortions of memoryless PA modeled using modified Rapp model with parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$ for input signal scaled such that $|x| \in [0, 1]$.

When studying the AM/AM diagram in Figure 2.12 it can be seen that the PA behaves linearly for small input powers, the so called linear region. The slope of the linear region is referred to as the linear gain of the PA, denoted G. From the AM/AM diagram of the figure it can also be seen that there is PA compression for middle input power and saturation for large input powers, the so called compression and saturation regions. As can be seen from the figure the input signal pushes the PA far into the saturation region.

In Figure 2.13 the AM/AM and AM/PM diagrams are instead plotted for the same PA model applied to the same signal scaled such that $|x| \in [0, 0.3]$ and both the input signal and PA output signal re-scaled to the original magnitude range of the input signal.

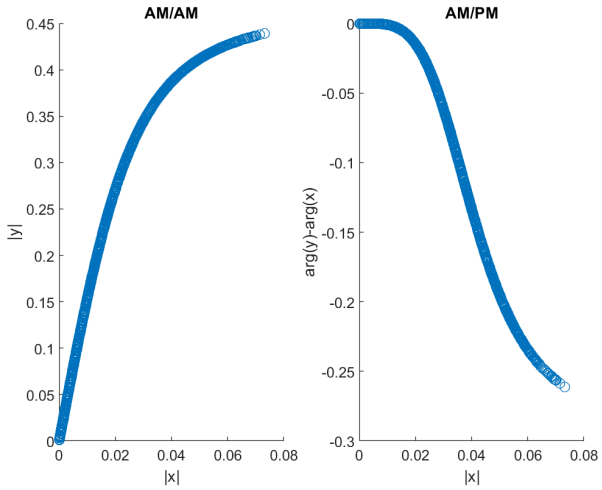


Figure 2.13: Amplitude and phase distortions of a memoryless PA modeled using the modified Rapp model with parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$, applied to example OFDM signal modulated using 256-QAM with bandwidth 100 MHz.

From the figure it can be seen that the input signal pushes the PA into the compression region without reaching the saturation region. The effects of the signal distortions can be seen in Figures 2.14, 2.15 and 2.16, which shows the envelope, constellation diagram and spectrum of the input and output signals of the PA respectively.

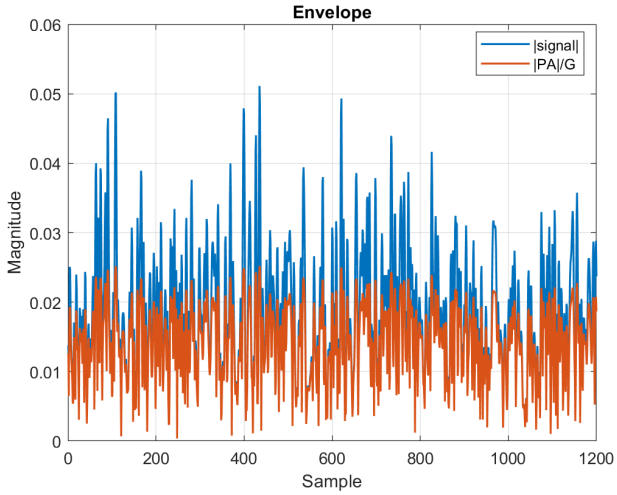


Figure 2.14: Envelope of first 1200 samples of a memoryless PA modeled using modified Rapp model with parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$, applied to example OFDM signal modulated using 256-QAM with bandwidth 100 MHz.

Figure 2.14 displays the envelope of the first 1200 samples of the PA input signal and gain-compensated output signal. From the figure it can be seen that the gain-

compensated PA output signal has a lower magnitude than the input signal for higher input magnitudes, due to the PA compression for the higher input signal samples.

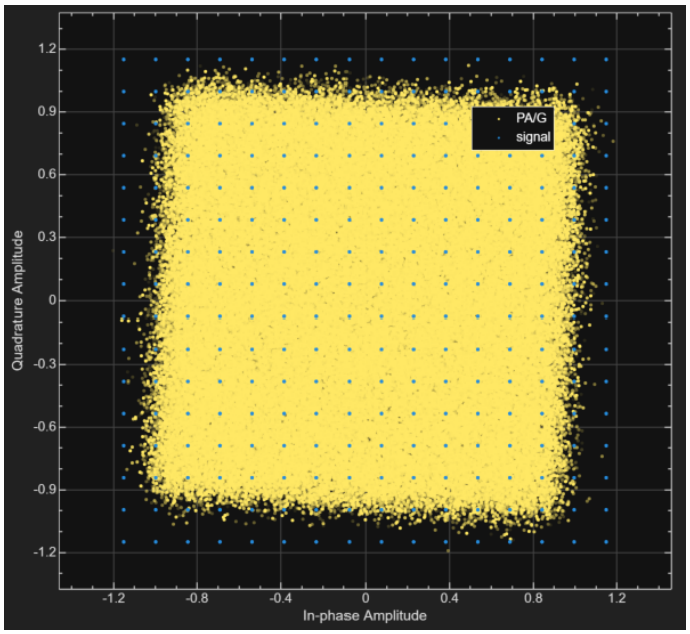


Figure 2.15: Constellation diagram of a memoryless PA modeled using modified Rapp model with parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$, applied to example OFDM signal modulated using 256-QAM with bandwidth 100 MHz.

The constellation diagram of the input signal and gain-compensated PA output signal can be seen in Figure 2.15. From the figure it can be seen that the PA output signal doesn't match the input signal constellation very well. The constellation diagram of the PA output signal is slightly rotated, the magnitude is somewhat compressed and overall much noisier compared to the constellation diagram of the input signal. These distortions makes it very difficult to identify the symbols at the receiver.

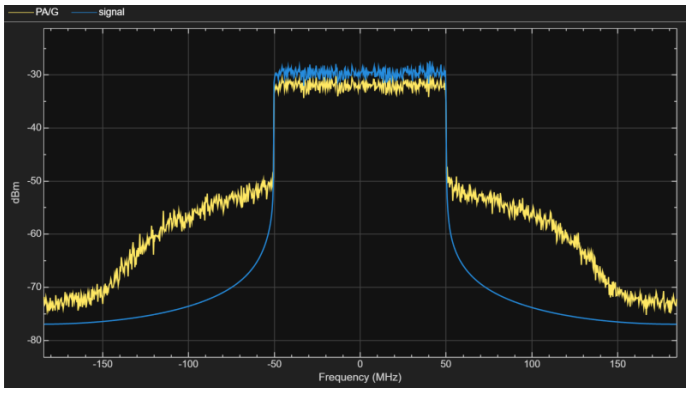


Figure 2.16: Spectrum of a memoryless PA modeled using modified Rapp model with parameters $G = 16$, $V_{sat} = 1.9$, $p = 1.1$, $A = -345$, $B = 0.17$ and $q = 4$, applied to example OFDM signal modulated using 256-QAM with bandwidth 100 MHz.

Similar problems can be seen in regards to the spectrum of the input signal and gain-compensated PA output signal in Figure 2.16. The PA output signal spectrum has a lower magnitude compared to the input signal spectrum as well as much higher leakage into adjacent channels.

PA Models

PA models characterizes the nonlinear distortions of the PA input-output relation and may consider memory effects and/or time varying effects [1]. Modeling a PA without considering memory effects is easier than modeling a PA exhibiting memory effects, and designing the DFE for such a model is easier than for a PA model exhibiting memory effects. On the other hand, modeling PA without memory effects is less realistic compared to modeling a PA with memory effects as PAs generally exhibit memory effects. In this thesis, experiments are over shorter periods for which time-varying effects are negligible and as such PA models considering time-varying effects will not be investigated.

Let us first consider modeling a memoryless PA, i.e. a PA which only exhibits nonlinear distortions. In polar coordinates, the PA input and output can be expressed as

$$x(n) = |x(n)| \cdot \exp(i \cdot \arg(x(n))) \text{ and } y(n) = |y(n)| \cdot \exp(i \cdot \arg(y(n)))$$

where

$$|y(n)| = f_\rho(|x(n)|) \text{ and } \arg(y(n)) = f_\Phi(|x(n)|) + \arg(x(n))$$

characterizes the AM/AM and AM/PM distortion curves of the PA [34]. One memoryless PA model is the Saleh model given in [34] for which the amplitude and phase distortions are characterized by

$$f_\rho(r) = \frac{\alpha_\rho r}{1 + \beta_\rho r^2} \text{ and } f_\Phi(r) = \frac{\alpha_\Phi r^2}{1 + \beta_\Phi r^2}, \tag{2.38}$$

where the coefficients α_ρ and β_ρ determine the nonlinearity degree of the amplitude distortion, and the coefficients α_Φ and β_Φ determine the nonlinearity degree of the

phase distortion. Another memoryless PA model is a modification of the Rapp model, here referred to as the modified Rapp model, given in [1] for which the amplitude and phase distortions are characterized by

$$f_\rho(r) = \frac{Gr}{(1 + |Gr/V_{\text{sat}}|^{2p})^{0.5p}} \text{ and } f_\Phi(r) = \frac{Ar^q}{1 + (r/B)^q}. \tag{2.39}$$

In the model the coefficient G is the linear gain of the PA and V_{sat} the PA output saturation level. The coefficient p determines the amount of nonlinearity of the amplitude characteristics of the PA and the coefficients A, B and q determine the phase distortions of the PA.

To model a PA with memory effects, one can use Volterra series based models such as MP or GMP. A direct method for identifying the inverse input output relation of the PA using the LS method is

$$\beta = (\Phi_y^H \Phi_y)^{-1} \Phi_y^H \mathbf{x} \tag{2.40}$$

and a corresponding iterative method is

$$\beta_{i+1} = \beta_i + \mu(\Phi_y^H \Phi_y)^{-1} \Phi_y^H \epsilon \tag{2.41}$$

with $\epsilon = \mathbf{x} - \hat{\mathbf{x}}$, where $\hat{\mathbf{x}} = \Phi_y \beta$ is the inverse model, Φ_y is the basis functions for the input $\mathbf{y}$ and β are the model coefficients [5]. An analogous iterative method for identifying the input-output relation of the PA using the LS method can be derived by replacing the model with $\hat{y} = \Phi_x \beta$, the error with $\epsilon = y - \hat{y}$, and the basis functions for the input y with the basis functions Φ_x for the input x. This gives a direct method for identifying the PA input-output relationship using the LS method as

$$\beta = (\Phi_x^H \Phi_x)^{-1} \Phi_x^H \mathbf{y} \tag{2.42}$$

and a corresponding iterative method as

$$\beta_{i+1} = \beta_i + \mu(\Phi_x^H \Phi_x)^{-1} \Phi_x^H \epsilon \tag{2.43}$$

with $\epsilon = \mathbf{y} - \hat{\mathbf{y}}$, where $\hat{\mathbf{y}} = \Phi_x \beta$ is the model, Φ_x is the basis functions for the input $\mathbf{x}$ and β are the model coefficients.

2.5 HDL Code Generation from Fixed-Point Models

The transition from fixed-point algorithmic models to synthesizable Register-Transfer Level (RTL) hardware description is a critical step in creating hardware-efficient systems. To bridge this gap, automated HDL code generating techniques are incorporated in modern design flows which save development time greatly and minimize human translation errors [42].

MathWorks' HDL Coder offers a thorough framework for producing synthesizable VHDL and Verilog code from Simulink models and MATLAB functions ready for

hardware implementation [43] [15]. The tool generates clear, traceable, and well-documented code [42] while maintaining the original structure of the model by converting fixed-point descriptions into RTL code. The generation process follows a structured workflow: HDL optimization uses pipelining, resource sharing, and RAM mapping to optimize the generated code for area, speed, or power [42]; model preparation involves identifying fixed-point data types using Fixed-Point Designer, specifying word lengths, fraction lengths, and scaling factors to achieve required accuracy while minimizing hardware resources [15]; and verification automatically creates co-simulation models and testbenches for precise RTL verification against the original Simulink model [42] [43]. Integration with Xilinx FPGA enables the combination of HDL Coder workflows with Xilinx System Generator for DSP, where Xilinx blocks offer optimized implementations on Xilinx FPGAs [43].

A number of factors must be carefully considered in order to achieve bit-exact implementation. Simulink's Fixed-Point Tool iteratively analyzes signal ranges and suggests best configurations [15]. Converting from floating-point to fixed-point introduces quantization effects that must be controlled by choosing proper word length and fractional length. Unlike floating-point systems, fixed-point implementations require explicit overflow protection using saturation logic or bit growth control. HDL Coder provides overflow management modes that convert directly to hardware [44]. HDL Coders can insert pipeline registers and define clock frequencies to fulfill timing closure requirements [42] [43], while RTL creation must follow target hardware timing constraints.

Co-simulation of Simulink models with external HDL simulators such as Cadence Xcelium or ModelSim [42] [43] is made possible by the HDL Verifier toolbox. This method has several advantages: third-party IP blocks and legacy HDL code can be integrated for full system-level validation [42] [43]; testbenches created for algorithmic models can be reused throughout the design flow for consistent verification; cycle-by-cycle comparison between the reference model and RTL implementation enables quick discrepancy detection. After co-simulation, a netlist optimized for the target FPGA or ASICs [42] [43] is created by synthesizing the RTL code using Cadence Genus or Xilinx Vivado. Important information on timing performance, power consumption, and area usage is provided by synthesis [42]. A developed edge detector uses just 6.57% of the logic area of its handwritten equivalent [42], demonstrating that coder-generated designs achieve comparable or better area efficiency than hand-coded HDL.

Verification models allow for overflow detection, precision analysis, and the gradual migration of subsystems while preserving fixed-point scaling information and removing quantization effects [15]. Block absence or additional register insertion [42] demonstrate the optimizations of these models, which remove dependence on MATLAB workspace variables and variant subsystems.

2.6 Neural Networks

An NN is a ML model that can be used to model nonlinear input-output relationships. This is done by building an architecture with several layers of connected

units with non-linear activation functions applied to linear regression models [9]. In this thesis only so-called feed-forward NNs will be considered, which is when the directionality of all the NN connections is forward [10]. Feed-forward DNNs are important ML tools, for which the universal approximation theorem gives that any non-linear function can be approximated arbitrarily well [10].

There are four main aspects to deep learning using NNs: data, architecture design, learning, and inference. In short, the data represents the input-output relationship to be replicated, the data has to be of high quality, and the amount of data must be sufficient for the NN to replicate said input-output relationship well. The NN architecture must be designed to consider the type of problem to be solved, the available data, and possibly some prior knowledge of the sought after input-output relationship, and is used to map the desired relationship between the input and output. The learning phase consists of calibrating the parameters of the NN to map the desired input-output relationship as well as possible, the so-called training. Finally, inference is using the trained NN to make predictions on new data [1]. A more detailed explanation of each of these aspects can be found in the subsequent sections, but first the universal approximation theorem will be explained in greater detail.

The Universal Approximation Theorem

The universal approximation theorem states that a feed-forward NN with at least one hidden layer that is sufficiently large (i.e. sufficient amount of hidden units), has a linear output layer and an activation function fulfilling certain conditions, can approximate any continuous function on a closed and bounded set between two finite dimensional spaces arbitrarily well [10]. The conditions on the activation function in the universal approximation theorem is the following; it can be any locally bounded piece-wise continuous function that is not a polynomial [45]. An important note is that no upper bound on the number of hidden units are given, meaning that the the hidden layer may be infeasibly large. In many cases this can be rectified by using several layers in a deeper NN. Another important note is that the universal approximation theorem guarantees that the NN can *represent* any input-output relationship, not that a training algorithm will be able to *learn* said relationship [10].

Data

As explained above, a sufficient amount of high quality data is needed for a NN to replicate a desired input-output relationship well. Here, high quality data refers to data which accurately represents the input-output relationship to be replicated. It should be complete, covering all cases of interest, and consistent as well as having an appropriate format.

It is common to divide the data into training, validation and test sets. The training set is the data used to train the model, the validation set is data used to validate the model performance during training, and the test set is data used to test the performance of the model after training is completed. Validation and test data are

not used to train the model [9]. According to [46], it is also common to apply transformations to the input variables in a pre-processing step with the hope that the input-output relationship will be easier to recognize. In this case, the same transformations have to be applied to the training, validation and test sets, as well as during inference.

Architecture Design

As stated the NN architecture is used to map the desired input-output relationship, such that said relationship can be learned during the learning phase by calibrating the parameters of the NN. As such the architecture must be designed with care so that the input-output relationship can be properly learned.

A NN gives an estimate of a nonlinear input-output relationship, which may be denoted $\hat{y} = f_\theta(\mathbf{x})$, where f is a function parameterized by the parameter vector θ. The function uses several layers of stacked linear regression models with learnable parameters and nonlinear activation functions [9]. In this case the NN is described to have a single output; NNs with multiple outputs can also be handled, [46].

The linear regression model is given by $\hat{y} = \mathbf{x}\mathbf{W}^T + b$, where $\mathbf{W} = (W_1 \ \dots \ W_p)^T$ are the weights, $\mathbf{x} = (x_1 \ \dots \ x_p)^T$ are the inputs and b is the bias. This model can be generalized to describe nonlinear input-output relationships by applying a nonlinear activation function according to $\hat{y} = h(\mathbf{x}\mathbf{W}^T + b)$ [9]. One common activation function is the sigmoid function [46], sometimes referred to as the logistic function; another common activation function is the Rectified Linear Unit (ReLU) function [9]. The sigmoid function and the ReLU function are given by

$$\begin{aligned} h_{\text{sigmoid}}(z) &= \frac{1}{1+e^{-z}}, \\ h_{\text{ReLU}}(z) &= \max(0, z). \end{aligned} \tag{2.44}$$

The model can be generalized further by stacking sequential layers of parallel generalized linear regression models to construct a network to map more complex nonlinear input-output relationships, creating a NN. If the NN has many layers it is referred to as a Deep Neural Network (DNN) [9]. A mathematical description of a DNN with L layers and a single output is given in [9] from which a corresponding mathematical description of a NN with multiple outputs can be derived as

$$\begin{aligned} \mathbf{q}^{(0)} &= \mathbf{x}, \\ \mathbf{q}^{(1)} &= h(\mathbf{z}^{(1)}), \ \mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{q}^{(0)} + \mathbf{b}^{(1)}, \\ &\vdots \\ \mathbf{q}^{(l)} &= h(\mathbf{z}^{(l)}), \ \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}, \\ &\vdots \\ \mathbf{q}^{(L-1)} &= h(\mathbf{z}^{(L-1)}), \ \mathbf{z}^{(L-1)} = \mathbf{W}^{(L-1)}\mathbf{q}^{()L-2)} + \mathbf{b}^{(L-1)}, \\ \mathbf{z}^{(L)} &= \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \\ \hat{\mathbf{y}} &= \mathbf{z}^{(L)}, \end{aligned} \tag{2.45}$$

where $\mathbf{W}^{(l)}$ is the weight matrix and $\mathbf{b}^{(l)}$ is the bias vector of layer l given by

$$\mathbf{W}^{(l)} = \begin{pmatrix} W_{11}^{(l)} & \dots & W_{1U_{l-1}}^{(l)} \\ \vdots & \ddots & \vdots \\ W_{U_l 1}^{(l)} & \dots & W_{U_l U_{l-1}}^{(l)} \end{pmatrix} \text{ and } \mathbf{b}^{(l)} = \begin{pmatrix} b_1^{(l)} \\ \vdots \\ b_{U_l}^{(l)} \end{pmatrix}. \tag{2.46}$$

A visualization of this can be found in Figure 2.17, where it is referred to as a fully connected feed-forward NN, as there is a weight between each neuron, or unit, in two consecutive layers and information is only propagated forward in the NN.

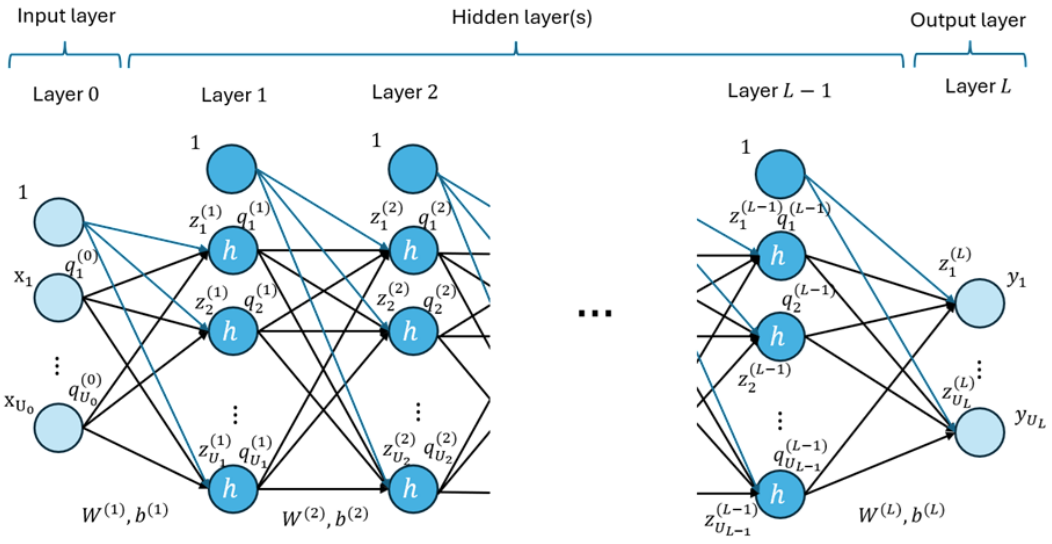


Figure 2.17: Schematic diagram of a fully connected feed-forward NN.

The layer with the inputs is called the input layer, the layer with the outputs the output, and all layers in between with hidden units the hidden layers. The connections from the top unit denoted with a one between layer $l-1$ and layer l represents the biases $\mathbf{b}^{(l)}$ and all other connections between the two layers represents the weights $\mathbf{W}^{(l)}$. It is important that the hidden layers each contain more hidden units than the inputs and outputs as there may otherwise be a loss of information. It is also important that the activation functions are included and nonlinear, as the NN could otherwise be replaced by an equivalent NN without hidden units. On the other hand a NN does not need to be fully connected, it may also be sparse, i.e. there does not need to be weights in between all units in two consecutive layers [46].

The NN architecture may be further generalized by including a skip connection [46]. A skip connection is a direct connection between non-consecutive layers, which helps accelerate training as well as improving final results, and allow for deeper NNs [47].

Learning

As stated the learning phase consists of calibrating the NN parameters θ, consisting of the weights $\mathbf{W}$ and biases $\mathbf{b}$ of the NN, so that the NN maps the desired input-

output relationship as accurately as possible. This is often referred to as training the NN.

When training NNs the goal is to find optimal parameters such that the error between the NN output and the desired output is minimal, which can be done by solving the optimization problem

$$\hat{\theta} = \arg\min_{\theta} J(\theta), \tag{2.47}$$

where the cost function

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}_i; \theta)$$

is the mean of the loss function $L(\cdot)$ evaluated at the training targets y_i and NN outputs $\hat{y}_i$ with $i = 1, 2, \ldots, N$ using the parameters θ which is a vector consisting of all NN parameters [9].

In [9] squared loss is recommended as loss function for a regression problem using a single output, this is defined as

$$L_{\mathrm{SE}}(y, \hat{y}) = (\hat{y} - y)^2.$$

Using squared error loss makes the cost function $J(\cdot)$ the MSE cost function. In other literature, such as [1], the cost function is often referred to as the loss function, which is used also for cases with multiple NN outputs, in which the mean is also taken over the outputs.

The solution of the optimization problem in (2.47) is typically found by using a gradient based search given by

$$\theta_{t+1} = \theta_t - \mu \nabla_\theta J(\theta_t), \ t = 0, 1, 2, \ldots \tag{2.48}$$

where θ_0 is initialized in some manner and $\hat{\theta}$ is taken as the final θ_t after termination according to some termination criterion [9].

Training a NN is in general a non-convex problem, for which gradient descent is not guaranteed to find a solution for an arbitrary non-convex function. However, in practice it is common for gradient descent to find a local minimum of a training loss function for a deep learning problem [48]. In general this solution is not unique as multiple parameter configurations can yield the same outputs, such as permutating the units and corresponding weights in a layer [49].

There are three main kinds of gradient descent algorithms; batch gradient descent methods in which the entire dataset is used to compute the optimization update, stochastic gradient descent methods in which a single sample is used to compute the optimization update, and mini-batch stochastic gradient descent methods (sometimes referred to as simply stochastic gradient descent methods) in which a random subset called a mini-batch of training samples are used to compute the optimization update [10]. When N is large it is computationally expensive to use batch gradient descent methods as the gradient is computed exactly using all training samples, instead its common to use stochastic gradient descent methods as they only approximate the gradient using much fewer samples [9]. Iterating through all training data

during learning is referred to as training the network for one epoch. In general the learning phase consists of multiple epochs.

For deep learning problems, the number of parameters are generally large, so to efficiently compute the gradients the backpropagation algorithm is applied. The algorithm uses the chain rule and reuses partial derivatives to reduce the number of calculations. The forward propagation is given by

$$\begin{cases} \mathbf{q}^{(0)} = \mathbf{x}, \\ \mathbf{q}^{(l)} = h(\mathbf{z}^{(l)}),\ \mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)},\ l = 1, 2, \ldots, L-1, \\ \mathbf{z}^{(L)} = \mathbf{W}^{(L)}\mathbf{q}^{(L-1)} + \mathbf{b}^{(L)}, \end{cases} \tag{2.49}$$

for a fully connected NN without a skip connection, where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ with $l = 0, 1, \ldots, L$ are the weights and biases in the parameter vector θ. The gradients of the cost function $J(\cdot)$ with respect to $\mathbf{z}^{(l)}$ and $\mathbf{q}^{(l)}$ for a single training sample are given by the following recursion for $l = 1, \ldots, L-1$:

$$\begin{aligned} \nabla_{\mathbf{z}^{(l)}} J(\theta) &= \begin{pmatrix} \partial J(\theta)/\partial z_1^{(l)} \\ \vdots \\ \partial J(\theta)/\partial z_{U_l}^{(l)} \end{pmatrix} = \nabla_{\mathbf{q}^{(l)}} J(\theta) \cdot h'(\mathbf{z}^{(l)}) \\ \nabla_{\mathbf{q}^{(l)}} J(\theta) &= \begin{pmatrix} \partial J(\theta)/\partial q_1^{(l)} \\ \vdots \\ \partial J(\theta)/\partial q_{U_l}^{(l)} \end{pmatrix} = \mathbf{W}^{(l+1)} \nabla_{\mathbf{z}^{(l+1)}} J(\theta), \end{aligned} \tag{2.50}$$

where $\cdot$ denotes element-wise multiplication and $h'(\cdot)$ is the derivative of the activation function. The gradient of the cost function with respect to the outputs can be computed directly as

$$\nabla_{\mathbf{z}^{(L)}} J(\theta) = \frac{\partial J(\theta)}{\partial \mathbf{z}^{(L)}}. \tag{2.51}$$

Using these gradients the gradients of the cost function $J(\cdot)$ with respect to the parameters can be computed as

$$\begin{aligned} \nabla_{\mathbf{W}^{(l)}} J(\theta) &= \begin{pmatrix} \partial J(\theta)/\partial W_{11}^{(l)} & \ldots & \partial J(\theta)/\partial W_{1U_{l-1}} \\ \vdots & \ddots & \vdots \\ \partial J(\theta)/\partial W_{U_l 1}^{(l)} & \ldots & \partial J(\theta)/\partial W_{U_l U_{l-1}} \end{pmatrix} = \nabla_{\mathbf{z}^{(l)}} J(\theta) \mathbf{q}^{(l-1)T} \\ \nabla_{\mathbf{b}^{(l)}} J(\theta) &= \begin{pmatrix} \partial J(\theta)/\partial b_1^{(l)} \\ \vdots \\ \partial J(\theta)/\partial b_{U_l}^{(l)} \end{pmatrix} = \nabla_{\mathbf{z}^{(l)}} J(\theta), \end{aligned} \tag{2.52}$$

from which the gradient of the cost function $J(\cdot)$ with respect to the parameters $\nabla_\theta J(\theta)$ can be computed [9].

The backpropagation algorithm described in [9] can be extended to also include skip connections. Let us assume the NN described above has a single weighted skip connection between layer m and layer n with $m < n$. For clarity the weights

and biases in this part will be denoted as the following; $\mathbf{W}^{(l-1\to l)}$ for the weights between two consecutive layers $l-1$ and l (otherwise denoted $\mathbf{W}^{(l)}$) and $\mathbf{W}^{(m\to n)}$ for the weights of the skip connection between layers m and n. Looking at the forward propagation, the skip connection does not affect the forward propagation of layer m, and the forward propagation of layer n becomes

$$\mathbf{q}^{(n)} = h(\mathbf{z}^{(n)}),\ \mathbf{z}^{(n)} = \mathbf{W}^{(n-1\to n)}\mathbf{q}^{(n-1)} + \mathbf{b}^{(n)} + \mathbf{W}^{(m\to n)}\mathbf{q}^{(m)}. \tag{2.53}$$

Looking at the backpropagation, the skip connection does not affect the backpropagation of layer n, and the backpropagation of layer m becomes

$$\begin{aligned}\nabla_{\mathbf{z}^{(m)}} J(\theta) &= \nabla_{\mathbf{q}^{(m)}} J(\theta) \cdot h'(\mathbf{z}^{(m)}),\\ \nabla_{\mathbf{q}^{(m)}} J(\theta) &= \mathbf{W}^{(m\to m+1)}\nabla_{\mathbf{z}^{(m+1)}} J(\theta) + \mathbf{W}^{(m\to n)}\nabla_{\mathbf{z}^{(n)}} J(\theta),\end{aligned} \tag{2.54}$$

giving

$$\begin{aligned}\nabla_{\mathbf{W}^{(m\to m+1)}} J(\theta) &= \nabla_{\mathbf{z}^{(m+1)}} J(\theta)\mathbf{q}^{(m)T},\\ \nabla_{\mathbf{W}^{(m\to n)}} J(\theta) &= \nabla_{\mathbf{z}^{(n)}} J(\theta)\mathbf{q}^{(m)T},\\ \nabla_{\mathbf{b}^{(m)}} J(\theta) &= \nabla_{\mathbf{z}^{(m)}} J(\theta).\end{aligned} \tag{2.55}$$

A final aspect regarding the backpropagation algorithm, by stacking N_b training samples in matrices according to

$$\mathbf{Q} = \begin{pmatrix}\mathbf{q}_1^T\\ \vdots\\ \mathbf{q}_{N_b}^T\end{pmatrix},\ \mathbf{Z} = \begin{pmatrix}\mathbf{z}_1^T\\ \vdots\\ \mathbf{z}_{N_b}^T\end{pmatrix},\ \nabla_{\mathbf{Q}} J(\theta) = \begin{pmatrix}\nabla_{\mathbf{q}_1} J(\theta)^T\\ \vdots\\ \nabla_{\mathbf{q}_{N_b}} J(\theta)^T\end{pmatrix} \text{ and } \nabla_{\mathbf{Z}} J(\theta) = \begin{pmatrix}\nabla_{\mathbf{z}_1} J(\theta)^T\\ \vdots\\ \nabla_{\mathbf{z}_{N_b}} J(\theta)^T\end{pmatrix} \tag{2.56}$$

the backpropagation algorithm can be applied to the entire mini-batch at once [9].

Gradient descent algorithms are common to use when training NNs, it is also possible to use momentum algorithms, in which the parameter update also depends on a moving average of previous gradients. Another possibility is to use adaptive learning algorithms such as AdaGrad, RMSProp, or Adam [10]. Adam is an efficient stochastic optimization method, requiring only first order gradients and has a low memory requirement, that can be used to train NNs. The name comes from adaptive momentum and the algorithm computes individual adaptive learning rates for different parameters for the estimates of the first and second momentum of the gradients, combining the advantages of AdaGrad and RMSProp [50].

As the cost function is generally non-convex, the training is sensitive to the initial parameter values. Typically all parameters are initialized as small random values. Furthermore, all initial biases of hidden units with the ReLU activation function are typically set as positive [9].

The vanishing gradient problem is when the parameter gradient of the NN tends to zero [51], resulting in the corresponding parameters to stop updating as the update increment tends to zero. The vanishing gradient problem occurs for DNNs with activation functions whose derivatives have magnitudes less than one giving a contraction for each layer of the backpropagation [51]. Using skip connections

mitigate this behavior allowing for the training of deeper NNs [47], the reason is that the skip connections bypass layers and as such creates a sort of path in the backpropagation with fewer multiplications with the contracting derivative of the activation function. Another way of avoiding the vanishing gradient problem is to use the ReLU activation function, which not only gives good results but also prevents the vanishing gradient problem [52], as the derivative of the ReLU function is one for inputs greater than zero.

Overfitting of a ML model, such as a NN, is when the model makes accurate predictions for the training dataset, but performs poorly on new unseen data. One way of preventing overfitting is to regularly validate the model performance on a validation dataset and stop training once the performance on the training and validation datasets starts diverging. Another way is to apply regularization, or increase the amount of training data [9].

Regularization may be applied to regulate the size of the NN parameters and keep them from getting too large. NN regularization means that a penalty is added to the cost function giving the regularized cost function

$$\hat{J}(\theta) = J(\theta) + \alpha\Omega(\theta), \tag{2.57}$$

where $J(\theta, \mathcal{D})$ is the original cost function, $\alpha \in [0, \infty)$ is a parameter weighing the penalty term $\Omega(\theta)$. The penalty term is a function which penalizes large parameter values. It is common to use norms for the penalty term and to assume all biases are negligible, i.e only consider the weights. Choosing

$$\Omega(\theta) = \frac{1}{2}\|\mathbf{W}\|_2^2 \tag{2.58}$$

gives L2-Regularization, also called ridge regression or Tikhonov regularization, and choosing

$$\Omega(\theta) = \|\mathbf{W}\|_1 \tag{2.59}$$

gives L1-Regularization [10].

Inference

As previously explained, inference is using the trained NN to make predictions on new data. This is fairly self explanatory, the input of interest is forward propagated through the trained NN with parameters calibrated to the training data in the learning phase.

2.6.1 Model Digital Front End using Neural Networks

Real- and complex-valued Neural Networks

In Artificial Intelligence (AI) and ML applications using NNs it is most common to use real valued NNs, i.e. NNs where all inputs, outputs and parameters have real values [53]. However the signals used for wireless communication in this thesis are complex, meaning that either a real valued representation of the complex valued

signal, e.g. use real and imaginary part of a Cartesian representation as inputs and outputs, or use a complex-valued NN. During this thesis work only real-valued NNs have been considered with real valued signal representations.

Neural Network Digital Pre-Distortion in literature

There have been several previous attempts to model DPD using NNs. These attempts generally train the NNs using methods comparable with the ILA and DLA coefficient identification methods for DPD models, and will here be referred to as training the NNs with ILA or DLA. When training a NN modeling DPD with ILA, the PA output $y(n)$ is used as NN input during training and the PA input $x(n)$ is used as NN target during training. The loss is then computed between the NN output $f_{DPD}(y(n))$ and the target $x(n)$. When training a NN modeling DPD with DLA, the signal $u(n)$ is used as input during training with the corresponding target $G \cdot u(n)$, i.e. the desired linearized PA output. The loss is then computed between the NN output propagated through a model of the PA $f_{PA}(f_{DPD}(u(n)))$ and the desired PA output $G \cdot u(n)$. An important note is that the DLA training requires that a derivable model of the PA is extracted before training.

In [54] and [55] fully connected NNs with Cartesian and polar coordinate signal representations respectively, are used to model DPD with ILA and MSE loss. Another paper that uses ILA to train a NN solution to model DPD is [1], in which two smaller NNs are used to correct the amplitude and phase distortions of the PA separately, making use of the amplitude and phase argument of polar coordinates to represent the signals. Here too, the MSE loss function was used. In [56] a fully connected NN with a skip connection between the input and output layer, and Cartesian signal representation is used to model DPD with DLA and the MSE loss function. For the DLA an NN of the gain-compensated PA is used when computing the loss with the target $x(n)$. A similar NN architecture also trained using DLA is found in [57] and [58]. Here fully connected NNs are used with skip connections between the input and output layers and four hidden layers with 64, 32, 16 and 32 hidden units respectively. Furthermore the Mean Squared Logarithmic Absolute Error (MSLAE) loss function is used (see [57] or [58]) and for the DLA a conventional PA model is used with the target $G \cdot x(n)$. To lower the NN complexity [57] suggests applying different NN pruning and quantization techniques. The NNs used to model DPD described here only use the current signal sample as NN input, and the NNs are as such unable to account for memory effects and are thus unsuitable to linearize PAs exhibiting memory effects.

One paper that does consider memory effects when modeling DPD using NNs is [59] in which a fully connected NN is used with Cartesian signal representation and the input layer consists of current and past signal samples according to

$$Re(in(n - m_1)) \text{ for } m_1 = 0, 1, \ldots, M_1 \text{ and } Im(in(n - m_2)) \text{ for } m_2 = 0, 1, \ldots, M_2,$$

where $in(\cdot)$ is the NN input signal, n is the current sample, and M_1 and M_2 are the maximum number of sample delays used for the real and imaginary part of the input signal respectively. The NN has a skip connection between the current input sample and the output layer, and it is trained using ILA with the MSE loss function.

Something similar is proposed in the MATLAB documentation in [60] and [61] where the input layer consists of real-valued Cartesian representations of the inputs used for MP models, given by

$$Re(in(n)),\ Im(in(n)) \text{ and } |in(n)|^k \text{ for } m = 0, 1, \ldots, M \text{ and } k = 1, \ldots, K,$$

where M is the memory depth and K is the nonlinearity order of the NN input signal. To model DPD a fully connected NN architecture is used, which is trained using ILA with mse loss. When modeling PA it is also suggested to use Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM) or Gated Recurrent Unit (GRU) NNs. The documentation also suggests preprocessing the data by multiplying it with the standard deviation of the training input signal to normalize it. The NNs used to model DPD described here does consider memory effects and can thus be used to linearize PAs with memory effects, however increasing the number of inputs also increases the NN complexity.

Neural Network Crest Factor Reduction in literature

There have been several previous attempts at modeling CFR using NNs, though it is often referred to as PAPR reduction using NNs in the literature. In [12] a fully connected NN with Cartesian signal representation is used to model the input-output relationship of a computationally complex CFR technique. Similar things are done in [62] and [63], where instead a fully connected NN and a linear regression model were applied to the modulation and demodulation of the signal respectively, to model the input-output relationship of computationally complex CFR techniques. Here the goal is to reduce the computational complexity, while giving similar performance as the respective conventional CFR techniques. The performance of the NN will be limited to that of the conventional technique it is trained to replace.

Neural Network joint Crest Factor Reduction and Digital Pre-Distortion in literature

There have been some previous attempts at jointly modeling CFR and DPD using NNs. In [57] and [58] it is suggested to use a joint CFR and DPD loss function to extend the NN DPD model to also model CFR. The joint loss would then be given by

$$L_{\text{joint}} = \gamma L_{\text{CFR}} + (1 - \gamma) L_{\text{DPD}},$$

where L_{CFR} would optimize the CFR metrics, i.e. PAPR, and L_{DPD} would be the MSLAE loss with DLA. However, investigations were only made for $\gamma = 0$, i.e. only DPD was modeled.

A similar method was used in [13], in which both CFR and DPD was modeled using a joint loss function. In the paper CFR is not explicitly mentioned, it is instead referred to as PAPR reduction. The method uses multiple NNs for signal modulation and demodulation as well as for modeling the DPD. The joint system with signal modulation, DPD and PA is optimized in terms of PAPR reduction, PA linearization and accurate symbol recognition at the receiver, by using a joint loss

function according to

$$L_{\text{joint}} = \alpha L_{\text{PAPR}} + \beta L_{\text{lin}} + \gamma L_{\text{acc}}.$$

The PAPR loss function L_{PAPR} is given by the NMSE between the actual PAPR of the DPD input and PAPR limit, the PA linearization loss function L_{lin} is given by the NMSE between the DPD input and PA output, and the accuracy loss function L_{acc} is given by the cross-entropy between the transmitter input and receiver output. This method does consider both CFR and DPD functionality without modeling specific conventional techniques, but also has high complexity as it uses multiple NNs.

In [64] a single fully connected NN is used to jointly model CFR and DPD. Cartesian representations of the signal is used for the inputs and outputs and memory is incorporated by including delayed samples of the input signal to the NN input as $in_{\text{re}}(n-m)$ and $in_{\text{im}}(n-m)$ for $m = 0, 1, \ldots, M$, similar to how memory is incorporated into the NN DPD model in [59]. An Augmented Iterative Learning Control (AILC) algorithm is used to obtain training data for a joint CFR and DPD model, from which the NN is trained to learn the input output relationship on the data. The AILC algorithm works by iteratively computing the error $e_k(n) = y_d(n) - y_k(n)$, where y_d is the desired system output and y_k is the actual system output of iteration k, and using the error to update the model output x_k. For this specific case, the desired system output is computed by applying the CFR technique CAF to the input signal and multiplying with the linear gain, and the actual system output is computed by applying the PA to the joint CFR and DPD model output. This method has lower complexity compared to the one in [13] as only one NN is used, but the performance is limited to the outcome of the AILC.

2.6.2 Basics of Compression: Pruning, Projection and Quantization

DNNs are well known for being extremely computationally and storage-intensive models, despite their remarkable success in a variety of application domains. For instance, to process a single image, the widely used VGG-16 architecture in [65] needs more than 500 MB of RAM and more than 15 billion Floating Point Operations (FLOPS). For hardware solutions, this high computational requirement presents numerous challenges, particularly in limited environments like embedded systems and edge devices. Because of this model compression has emerged as a key method for successfully enabling hardware implementations. With an emphasis on RTL synthesis, this section explain 3 main categories of compression schemes: Pruning, Projection and Quantization.

Pruning

Pruning is based on the fundamental realization that DNNs are extremely over-parameterized, which has been confirmed by tests conducted on numerous models and tasks. Over-parameterization is characterized by a large number of connections that serve mainly to optimize the NN during training and are not necessary for

producing outputs [66]. The fundamental idea behind pruning is that such redundant connections can be identified and removed from the model without negatively impacting its functionality.

Parameter pruning operates on the principle that DNNs are significantly over-parameterized, containing numerous connections that are either redundant or unimportant for the inference task,[67]. The theoretical basis for pruning traces back to the Optimal Brain Damage (OBD) framework proposed by LeCun, Denker, and Solla in [66], which provides a rigorous mathematical approach for identifying parameters whose removal minimally affects the loss function.

Consider a NN with parameter vector

$$\Theta = \{\theta_1, \theta_2, \ldots, \theta_N\}$$

that has been trained to minimize a loss function $L(\Theta)$. The change in loss resulting from perturbing a parameter θ_i can be expressed using the Taylor expansion:

$$\Delta L = \sum_i g_i \, \delta\theta_i + \frac{1}{2} \sum_i h_{ii} (\delta\theta_i)^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \, \delta\theta_i \, \delta\theta_j + O(\|\delta\Theta\|^3),$$

where

$$g_i = \frac{\partial L}{\partial \theta_i} \quad \text{and} \quad h_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}$$

are the gradients and the elements of the Hessian matrix, respectively.

At a local minimum, the first-order terms vanish ($g_i = 0$), and under the diagonal approximation where off-diagonal Hessian terms are neglected ($h_{ij} = 0$ for $i \neq j$), the saliency of parameter θ_i—defined as the increase in loss when that parameter is set to zero—is

$$S_i = \Delta L_i \approx \frac{1}{2} h_{ii} \theta_i^2.$$

Parameters with small saliency values can be removed with minimal impact on the loss function. This theoretical framework underpins modern pruning algorithms, though practical implementations employ various approximations to address computational constraints.

The following three stage pipeline is typically used in an iterative manner during the trimming process:

- Train: A highly specified over-fit model is trained until it reaches convergence.
- Pruning: Eliminating parameters according to a saliency metric
- Retraining the acquired sparse model to improve performance is known as fine-tuning One-shot pruning and training-time pruning are further potential strategies

A few common pruning techniques are: magnitude-based pruning, structured pruning, and Synaptic Flow (SynFlow) pruning. The subsequent subsections offer a thorough explanation of each of these techniques:

1. Magnitude based Pruning

   Magnitude-based pruning represents the simplest and most widely adopted pruning criterion, operating on the principle that weights with small absolute values contribute proportionally little to the NN's output and can therefore be removed without significantly affecting performance [68]. For a fully-connected layer computing

   $$\mathbf{y} = W\mathbf{x},$$

   the contribution of an individual weight W_{ij} to the output y_i is

   $$W_{ij}x_j.$$

   Under the assumption that input activations x_j are bounded and statistically well-behaved, weights with smaller magnitudes indeed produce smaller expected contributions.

   From a theoretical perspective, magnitude-based pruning can be derived as an approximation to the OBD framework under specific conditions. When the loss function is approximately quadratic and the NN has been trained to convergence with batch normalization, the diagonal Hessian elements h_{ii} are often approximately constant across parameters. In this case, the saliency becomes

   $$S_i \propto \theta_i^2,$$

   directly justifying magnitude-based pruning [69]. Additionally, from a Bayesian perspective, magnitude pruning corresponds to imposing a sparsity-inducing Laplace prior on the weights, leading to ℓ_1 regularization that explicitly promotes zero-valued parameters [70].

   The fundamental limitation of magnitude pruning lies in its independence assumption, treating each weight in isolation without considering statistical dependencies between weights. Multiple weights may jointly encode the same information, and removing one may be compensated by others, suggesting that the "true" importance of a weight is not uniquely determined by its magnitude alone [71].

2. Structured Pruning

   Structured pruning generalizes the pruning from individual weights to entire architectural groups such as filters, layers or neurons (channels) [72]. The main motivation is that taking out the structural units instead of the individual weights leads to hardware-friendly sparsity patterns: structured pruning yields dense, lower dimensional matrices that are well-suited to Multiply-Accumulate Operations (MACs) arrays and standard systolic arrays instead of irregularly placed zero weights.

   For a convolutional layer with weight tensor

   $$W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K_h \times K_w},$$

   structured pruning can operate at multiple granularities. Channel pruning removes entire input or output channels, reducing C_{in} or C_{out}. The importance

of channel j can be quantified as the expected contribution to the layer output:

$$S_j^{(\text{out})} = \mathbb{E}_x \left[\left\| \sum_{i=1}^{C_{\text{in}}} W_{j,i,:,:} * X_{i,:,:} \right\|_F^2 \right],$$

where $*$ denotes convolution and X is the input feature map [73]. Eliminate channels that have low predicted output magnitudes. In filter pruning, the Frobenius norm of filter weights is usually employed as a saliency metric, resulting in the removal of entire filters and is similar to output channel pruning.

In terms of optimization, structured pruning essentially boils down to a combinatorial subset selection problem, which is generally NP-hard. This leads to the development of effective approximations, e.g. group LASSO regularization and LASSO-based selection, which promote structured sparsity patterns [74]. Structured pruning approaches have theoretical bounds on the reconstruction error when the weight matrix has fast decay of singular values.

3. Synaptic Flow Pruning

Tanaka et al.'s Synaptic Flow (SynFlow) pruning in [75] is a paradigm from earlier pruning techniques because it does not rely on gradients or weight magnitudes. The fundamental concept of SynFlow is that the process enters a feedback loop when magnitude-based pruning is applied since the remaining parameters typically have bigger magnitudes to compensate for lost connections. The SynFlow method is predicated on the idea of preserving the total quantity of data transmitted through the NN, which is determined by multiplying the sum of the weights by their influence on the NN's output. When each weight w_{ij} is multiplied by an individual parameter α_{ij}, the influence of that specific weight on the output is captured by the derivative of the output with respect to α_{ij} evaluated at $\alpha_{ij} = 1$:

$$\left. \frac{\partial f(\boldsymbol{\alpha} \odot W)}{\partial \alpha_{ij}} \right|_{\alpha_{ij}=1},$$

where $f(\cdot)$ denotes the NN output and $\odot$ represents element-wise multiplication.

The SynFlow saliency for weight w_{ij} is defined as

$$S_{ij} = \left| w_{ij} \cdot \frac{\partial L}{\partial w_{ij}} \right|,$$

where L is the loss function defined as the sum of NN outputs for dummy inputs. This formulation simplifies to the sum of all paths passing through the weight, and it has the notable property that the sum of saliencies across any layer remains constant, enabling automatic layer-balanced pruning [76].

SynFlow's theoretical justification is based on its connection to the Neural Tangent Kernel (NTK), which explains how infinitely wide NNs are trained [77]. Retaining weight entries with high SynFlow saliency means keeping directions in the parameter space that are essential for NTK and, consequently,

NN training. Crucially, because SynFlow uses constant dummy input vectors and averages their output values as the loss function, it does not require training samples.

Projection: Principal Component Analysis

One linear method for lowering the number of dimensions is Principal Component Analysis (PCA), which transforms a set of potentially correlated variables into a set of linearly uncorrelated variables called principal components [78]. PCA compresses NNs by reducing weight matrices to a low-dimensional space that eliminates additional data while capturing the most variation.

Consider a weight matrix

$$W \in \mathbb{R}^{m \times n}$$

representing the parameters of a fully-connected layer mapping n-dimensional inputs to m-dimensional outputs. The goal of PCA-based projection is to find two matrices

$$U \in \mathbb{R}^{m \times r} \quad \text{and} \quad V \in \mathbb{R}^{r \times n}$$

such that

$$W \approx UV,$$

with $r \ll \min(m, n)$, minimizing the Frobenius-norm reconstruction error:

$$\min_{U,V} \|W - UV\|_F^2.$$

The optimal solution is given by the Eckart–Young–Mirsky theorem, which states that the best rank-r approximation of W is obtained via its Singular Value Decomposition (SVD) [79],

$$W = \tilde{U}\Sigma\tilde{V}^T,$$

where $\tilde{U} \in \mathbb{R}^{m \times m}$ and $\tilde{V} \in \mathbb{R}^{n \times n}$ are orthogonal matrices containing the left and right singular vectors, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing singular values

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0.$$

The rank-r approximation is

$$W^{(r)} = U\Sigma_r V^T,$$

where U contains the first r columns of $\tilde{U}$, Σ_r contains the first r singular values, and V contains the first r rows of $\tilde{V}^T$.

The reconstruction error is precisely the sum of squared discarded singular values:

$$\|W - W^{(r)}\|_F^2 = \sum_{i=r+1}^{\min(m,n)} \sigma_i^2.$$

The singular values σ_i represent the amount of variance captured by each principal component. The fraction of variance preserved by retaining r components is

$$\frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{\min(m,n)} \sigma_i^2},$$

providing a principled method for selecting the compression rank. In the information-theoretic approach, a PCA projection is the best linear compression in the sense that it preserves as much variance as possible, achieved by minimizing the MSE for a fixed rank constraint. The reconstruction minimizes the predicted squared reconstruction error, under the assumption that the data distribution is Gaussian, and the projection matrix Q maps the original n-dimensional inputs to an r-dimensional subspace ($r < n$) [80].

In NNs, PCA is applied layer by layer by factorizing the weight matrix of each layer separately into two smaller matrices. This factorized structure reduces the computational complexity and parameter storage by replacing one large matrix multiplication with two smaller matrix multiplications.

$$\text{Compression ratio} = \frac{r(m+n)}{mn}$$

For convolutional layers, the 4D weight tensor

$$W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K_h \times K_w}$$

can be reshaped into a 2D matrix of dimensions

$$C_{\text{out}} \times (C_{\text{in}} \cdot K_h \cdot K_w)$$

before applying SVD, effectively decomposing the convolution into a pointwise convolution followed by a spatial convolution [81].

Quantization

Quantization is the process of mapping a continuous (or high-precision) set of values to a discrete (or low-precision) representation. Quantization is an approach to compress NNs, which reduces the numerical precision of weights/activations from the standard FP32 format to lower-bit representations such as binary, INT8, or 4-bit integer (INT4) [82].

The quantization function

$$Q : \mathbb{R} \to \mathcal{Q}$$

maps real-valued numbers to a finite set

$$\mathcal{Q} = \{q_1, q_2, \ldots, q_k\}$$

where $k = 2^b$ for b-bit quantization.

Uniform quantization divides the input range $[\beta, \alpha]$ into 2^b equally spaced intervals with quantization parameters:

- **Scale factor:**

$$\Delta = \frac{\alpha - \beta}{2^b - 1}$$

- **Zero point:**

$$z = \text{round}\left(-\frac{\beta}{\Delta}\right)$$

The quantization and dequantization operations are:

$$Q(x) = \text{clip}\left(\left\lfloor \frac{x}{\Delta} \right\rceil + z,\ 0,\ 2^b - 1\right)$$

$$\tilde{x} = \Delta \cdot \Big(Q(x) - z\Big)$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer, and $\text{clip}(\cdot)$ constrains the value to the representable range $[0, 2^b - 1]$ [83]. Rounding error, which for uniform quantization follows a uniform distribution

$$\mathcal{U}\left(-\frac{\Delta}{2}, \frac{\Delta}{2}\right)$$

with variance

$$\frac{\Delta^2}{12},$$

and clipping error, which happens when input values fall outside the representable range and are clipped to the nearest endpoint, make up the total quantization error. The overall quantization distortion is measured by the MSE:

$$\text{MSE} = \int_{\beta}^{\alpha} (x - Q(x))^2\, p(x)\, dx + \int_{-\infty}^{\beta} (x - \beta)^2\, p(x)\, dx + \int_{\alpha}^{\infty} (x - \alpha)^2\, p(x)\, dx$$

In the following, a few different aspects to consider when applying quantization is explained:

- Quantization Granularity and Approaches:

  The accuracy and hardware complexity are greatly affected by the definition of the degree of quantization parameter. Per-tensor quantization is hardware-friendly, but it does not work well when the distributions of weights vary a lot across channels, because it uses a single scale factor for the whole weight tensor [84]. Per-channel quantization preserves relative magnitudes better at the expense of increased hardware resources by allowing different scale factors for each output channel. This is a tradeoff between quantization precision and hardware simplicity.

  The best values of clipping (β^*, α^*) are statistically those that minimize the MSE. This is usually done by assuming a distribution for $p(x)$, e.g., Gaussian or Laplacian, and then solving numerically.

  The selection of symmetric quantization (zero point at zero) or affine (asymmetric) quantization has consequences for hardware implementation. Symmetric quantization simplifies the arithmetic logic and does not require an addition after the multiplication, but may waste dynamic range for asymmetric distributions [85].

- Quantization-Aware Training vs. Post-Training Quantization:

  Post-Training Quantization (PTQ) relies on a small representative dataset to calibrate the quantization parameters after the model is trained. The procedure computes scale and zero point values, quantizes weights and activations and collects statistics (min, max and percentiles) for each tensor. PTQ

is computationally efficient as it can be performed without retraining, but distribution mismatch may lead to a large reduction in accuracy for low-bit quantization below 8 bits [86].

Quantization-Aware Training (QAT) as explained in [87] simulates quantization during training with "fake quantization" operators that perform the quantization function in the forward pass, and apply the straight-through estimator (STE) to approximate the gradients through the non-differentiable rounding operation:

$$\frac{\partial Q_{\text{fake}}(x)}{\partial x} \approx 1$$

In particular for sub-8 bit quantization, this allows the model to adjust its weight distribution to minimize the quantization error, and thus usually achieves much higher accuracy than PTQ. More advanced PTQ methods, such as Adaptive Rounding (AdaRound), consider the effect of rounding on the overall loss function and optimize the rounding decisions to minimize the task loss, rather than the local MSE, resulting in better performance [88].

- Implications for Hardware Implementation:

  By completely transforming the arithmetic core design, INT8 quantization provides significant advantages for hardware implementation. An INT8 multiplier only needs an 8×8-bit integer multiplication with a 16-bit product, whereas an FP32 multiplier requires 24-bit mantissa multiplication, 8-bit exponent addition, normalization, rounding, and special case handling [89]. Since FP16 uses a 10-bit mantissa multiplier, which is comparable in complexity to INT8's 8-bit core, comparing INT8 with FP16 is more relevant. Still, rounding, normalization, and exponent handling continue to contribute overhead to FP16 [**?**]. This represents a substantial reduction in arithmetic complexity. With area reductions of about 10×, INT8's energy per MAC (0.2–0.5 pJ) is significantly lower than that of FP32 (10–20 pJ).

  Quantization also offers equally important benefits for memory systems: bitwidth reduction from 32 to 8 leads to a 4× reduction in external memory transfers, on-chip Static Random Access Memory (SRAM) footprint, and memory bandwidth requirements. However, partial products accumulate to larger values, requiring 24–32 bits of accumulator precision to prevent overflow in typical deep learning inference workloads, even for 8-bit weights and activations. This requires the hardware designer to carefully consider saturation logic and the bitwidth of the accumulator.

3

Methodology

3.1 Software

In order to facilitate a smooth transition from algorithmic exploration to RTL code generation, the software implementation and hardware deployment for this design followed a model-based design workflow utilizing the MathWorks platform. Double-precision floating-point models were built in MATLAB and Simulink to evaluate the system's functional behavior during the initial development of the key algorithms.

The floating-point Simulink model underwent a bit-exact fixed-point conversion in order to prepare the design for hardware implementation. After that, the model was reorganized to include HDL compatible blocks, which allowed for hardware resource estimation from the HDL Coder report. The HDL Coder App and its integrated Workflow Advisor, which directed the process through model validation, test bench construction, and synthesis optimization, were used to generate the real RTL code. This automated method ensured that the Verilog or VHDL code produced was bit-exact.

Matlab scripts were used to apply several compression strategies, including as pruning, projection, and QAT, for the NN compression of the design in order to minimize the computational complexity and memory footprint of the model. The Deep Network Quantizer App in Matlab was used to quantize the NN parameters following projection. It modeled lower-precision inference to find the best scaling factors and zero-points while minimizing any loss in model accuracy. Hardware resource estimates were computed using approximate hardware metrics to assess the effectiveness of various compression and quantization configurations, allowing for useful comparisons between different designs. In order to achieve a balanced trade-off between performances, these tools and techniques worked together to ensure that both the signal processing pathways and the deep learning parameters were effectively mapped to the target hardware.

3.2 Hardware Evaluations

3.2.1 Justification for Hardware Performance Metrics

The multidimensional resource requirements of NN inference on FPGA platforms were carefully captured in the hardware performance evaluation approach used in

this study. The specific limitations of programmable logic devices, the necessity to assess the cumulative impacts of pruning, projection, quantization approaches, and basic hardware design principles all had a role in the selection of each metric. This section gives a thorough explanation of the measures that were selected and shows why other methods would not be enough to achieve our analytical goals.

The Underlying Concern with Hardware Performance Assessment

A basic problem in evaluating NN architecture's hardware efficiency is that a model's performance characteristics have an unbreakable connection to the particular hardware platform, toolchain, and implementation decisions. During the algorithmic research stage, direct hardware implementation which includes RTL design, synthesis, placement and routing and physical testing requires a significant amount of time [90]. This is especially noticeable when comparing different compression methods, since each variation would require distinct hardware implementation, synthesis and characterization cycles that may take weeks or months to complete.

We used analytical performance models to approximate hardware behavior without requiring full implementations due to the lengthy delays in hardware availability and the short time for experimentation. Despite being approximations, these models allow for quick design-space exploration and offer repeatable comparisons between various structures and methods. In order to balance analytical tractability and prediction accuracy, the metrics used in this study were chosen to offer significant hardware estimates while still being measurable from high-level NN descriptions.

3.2.2 Hardware Metrics Estimation Methodology

Power consumption and latency for the proposed NN-based solution are the main optimization goals of this thesis. In a perfect development environment, each die would display distinct power characteristics based on its fabrication process, operating voltage, and switching activity, and these metrics would be immediately measured by deploying the design onto actual hardware. However, an alternative estimation method is needed because real hardware prototypes are unavailable throughout the development process.

Hardware metrics serve as substitute indicators that enable quantitative evaluation of resource usage, which in turn informs power and latency predictions. Depending on the design paradigm, legacy hardware implementations versus NN-based models, different methods are used to calculate these metrics. The NN model uses a more straightforward and architecture-aware computation process, while the legacy approach depends on post-synthesis resource reports. The ensuing subsections provide more details on both approaches.

Legacy Hardware Metric Calculation

For legacy designs, hardware metrics are derived from the high-level resource report generated by the HDL Coder after the HDL code generation process. A MAC capacity metric is produced by normalizing different resource kinds, such as multipliers,

adders, and registers, into equivalent units. The following is the definition of the procedure:

1. INT8 MAC Primitive Definition: One 8-bit multiplier, one 8-bit adder, and one 8-bit register are combined to form a single INT8 MAC operation:

$$1\text{ MAC} = 1\times(8\times 8\text{ Multiplier}) + 1\times(\text{8-bit Adder}) + 1\times(\text{8-bit Register}) \quad (3.1)$$

2. Multiplier Normalization: All multipliers present in the design are normalized to 8×8 equivalent multipliers. For each multiplier instance with input bit-widths Width_1 and Width_2, the equivalent count is computed as:

$$\text{MultEq} = \sum \left(\text{count} \times \frac{\text{Width}_1 \times \text{Width}_2}{64} \right) \quad (3.2)$$

3. Adder Normalization: Eight-bit equivalent adders are used to normalize all adders. For every instance of an adder with bit-width. The corresponding count for width is:

$$\text{AddEq} = \sum \left(\text{count} \times \frac{\text{Width}}{8} \right) \quad (3.3)$$

4. Register Normalization: Every register is converted to comparable 8-bit registers. The total number of 1-bit registers is divided by 8 to find the comparable count:

$$\text{RegEq} = \frac{\text{Total 1-bit Registers}}{8} \quad (3.4)$$

5. Strict MAC Capacity ($\text{MAC}_{\text{strict}}$): The maximum number of full INT8 MAC structures that can be created, constrained by the most limited resource among multipliers, adders, and registers, is represented by the stringent MAC capacity:

$$\text{MAC}_{\text{strict}} = \min(\text{MultEq},\ \text{AddEq},\ \text{RegEq}) \quad (3.5)$$

6. Leftover Resources: The "leftover" resources are calculated as the difference between the total normalized resources and those used by the stringent MAC structures after resources have been allocated for the strict MAC capacity:

$$\text{Leftover Mult} = \text{MultEq} - \text{MAC}_{\text{strict}} \quad (3.6)$$
$$\text{Leftover Add} = \text{AddEq} - \text{MAC}_{\text{strict}} \quad (3.7)$$
$$\text{Leftover Reg} = \text{RegEq} - \text{MAC}_{\text{strict}} \quad (3.8)$$

These remaining resources are multipliers, adders, and registers that are not included in full MAC units but can still be used for additional computations or enhance processing power.

7. Resource Pool MAC Capacity (MAC_{pool}): By averaging the available normalized resources, the resource pool MAC capacity offers a more flexible estimate that permits the inclusion of leftover multipliers, adders, and registers that might not form entire MAC structures but can nevertheless increase computing throughput:

$$\text{MAC}_{\text{pool}} = \frac{\text{MultEq} + \text{AddEq} + \text{RegEq}}{3} \tag{3.9}$$

These metrics MAC pool, leftover resource numbers, and MAC strict offer complementary viewpoints on hardware capability. Leftover resources show the remaining capability for auxiliary operations, MAC pool offers an optimistic upper bound on total computational resources, and MAC strict represents a conservative lower constraint on entire MAC units.

Neural Network Hardware Metric Calculation

The NN hardware metrics are calculated using a more straightforward methodology that directly corresponds with the operational features of neural network inference, in contrast to the legacy approach. The metric calculation concentrates on measuring the actual workload rather than standardizing different resource types because structured MAC operations across convolutional and fully connected layers naturally dominate NN computations.

The network architecture factors, such as kernel dimensions, stride, input feature map sizes, number of channels, and batch size, directly determine the NN hardware metrics. This strategy avoids the normalization complications that come with the old approach and does away with the requirement for post-synthesis resource reports. The NN hardware metrics used in this thesis are described in depth in the following subsections.

1. Multiply-Accumulate Operations:

   $80 - 90\%$ of all arithmetic operations in modern deep learning models are MAC, which serve as the computational basis of NN hardware accelerators [91]. This metric is generally applicable across many NN topologies since all convolution, fully linked layer, and matrix multiplication ultimately reduce to sequences of MACs.

   The use of DSP slices in FPGA implementations is directly correlated with each MAC operation in a NN accelerator. With a roughly linear relationship between MAC operations and execution time in throughput-optimized architectures, Zhang et al. showed that MAC count is the main factor influencing inference delay [92]. According to Chen et al., more than 90% of the energy used in NN inference comes from MAC operations [93].

   A depth-wise separable convolution may need many MAC operations with few parameters, but a sparse NN may have many parameters with few MACs. The number of parameters alone does not reveal how parameters are used in calculation. In a similar vein, layer depth disregards the wide range of

computational intensity among various layer types. The efficiency of compression strategies is revealed by normalizing MACs in relation to the baseline; trimming can reduce MACs by $50 - 80\%$ without sacrificing accuracy.

The computational complexity is measured by MACs, which are calculated as:

$$\text{TotalMACs} = \sum_{i=1}^{L} \text{MACs}_i \tag{3.10}$$

where L stands for the total number of layers in the NN. For each convolutional or fully connected layer, the number of MACs is calculated as:

$$\begin{aligned} \text{MACs}_i = {} & \text{Input Channels} \times \text{Kernel Height} \times \text{Kernel Width} \\ & \times \text{ Output Channels} \times \text{Output Height} \times \text{Output Width}. \end{aligned} \tag{3.11}$$

2. Memory Footprint:

   The most limited resource in FPGA-based NN accelerators is on-chip memory, which is frequently more restrictive than computing resources. A resource imbalance occurs when memory capacity restricts deployment before computational capacity since modern FPGA architectures usually offer $2 - 4\times$ more DSP slices than Block Random Access Memory (BRAM) blocks.

   Regardless of computational efficiency, a NN that effectively uses DSPs yet surpasses BRAM capacity cannot be constructed. Furthermore, Horowitz showed that accessing data from Dynamic Random Access Memory (DRAM) uses about $700\times$ more energy than arithmetic operations [88]. This indicates that memory access dominates energy use. In most NN architectures, especially convolutional NNs with extensive filter banks, the memory footprint is dominated by parameter storage requirements, which are captured by the memory metric [94].

   The architectural difference between on-chip BRAM and off-chip DRAM, which has a fundamental impact on performance and viability, is ignored by the total model size in MB. NN architecture cannot be used to determine counting memory operations because it depends on the implementation. The basic limitation on on-chip weight storage is captured by parameter-based memory, which offers a consistent baseline.

   The total memory used is determined based on how much space the parameters take up:

$$\text{TotalMemory} = \sum_{i=1}^{L} (\text{Parameters}_i \times \text{Bitwidth}_i) \tag{3.12}$$

   where Parameters_i denotes the number of learnable parameters in layer i, and Bitwidth_i represents the numerical precision used for storage.

3. Compute Cost

   The super-linear effect of numerical precision on hardware resource usage is addressed by the compute cost metric. Multiplier complexity in digital circuit design increases quadratically with bitwidth; an n-bit multiplication requires roughly n^2 logic gates [95]. Custom accelerators, FPGAs, and ASIC all have this underlying link.

   About 64 logic gates are needed for an 8-bit multiplier, 256 for a 16-bit, and 1024 for a 32-bit multiplier, a 16× improvement in precision from 8-bit to 32-bit. According to Sze et al., 8-bit operations use about 1/16 the energy of 32-bit operations, demonstrating that energy per MAC scales as Bitwidth^2 [96]. This compound effect is captured by the compute cost measure

   $$\left(\text{TotalMACs} \times \text{Bitwidth}^2\right),$$

   which reveals the actual hardware benefits of quantization that would not be apparent if simply MAC count were taken into account.

   The usual precision used to train NN is FP32. On the other hand, lower precision formats like 8-bit Integer (INT8) and FP16 are being used more often for inference since they allow for faster computation while keeping acceptable accuracy and need less memory and energy. Computational volume cannot be captured by bitwidth alone. The benefits of quantization are underestimated by linear bitwidth weighting. Beyond analytical modeling, precise hardware characterisation is necessary for direct area or power measurement. In a single, comprehensible value, the compute cost measure offers a theoretically supported estimate that captures both aspects of hardware efficiency.

   The total cost for computation, considering both the math operations and how precisely numbers are handled, is written as:

   $$\text{ComputeCost} = \text{TotalMACs} \times \text{Bitwidth}^2. \tag{3.13}$$

   This measurement shows how much energy and space are needed for MACs in hardware, because the number of bits used has a big effect on how much of a circuit is used. Studies show that switching from 32-bit to 8-bit precision can make computations up to 4 times quicker than using FP32 [97]. Moreover, the bitwidth squared scaling shows how the complexity of multiplication grows quickly with higher precision; a 32-bit multiplier needs about 16 times more resources compared to an 8-bit multiplier.

4. Flip-Flop Proxy

   Sequential logic resource requirements, which may become a constraint in FPGA implementations, are addressed by the Flip-Flop proxy (FFp) metric. Flip-Flops (FFs) perform crucial tasks such pipelining, control logic, synchronization, and intermediate storage, while DSPs manage arithmetic and BRAM stores weights.

   In a multiply-accumulate datapath, each pipeline stage needs registers proportionate to bitwidth; deep pipelines, which are necessary for high operating

frequencies, use a large number of FFs [98]. Even when DSPs and BRAM are underutilized, Guo et al. showed that FFp usage frequently becomes the limiting factor during placement and routing in [99]. The FFp for a model with 1 million parameters at 8-bit precision is 8 million, whereas the FFp for a model with 32-bit parameters is 32 million, indicating the extra register resources needed.

Pipeline register overhead is ignored by DSP count. Intermediate storage and synchronization registers are not taken into consideration by BRAM measurement. Estimates of high-level synthesis vary on the tool. Early hardware feasibility assessment is made possible by the FFp's repeatable, architecture-independent estimation.

A substitute for sequential logic resource utilization

$$\mathrm{FF}_{\mathrm{proxy}} = \mathrm{TotalParameters} \times \mathrm{Bitwidth}. \tag{3.14}$$

This estimates the register requirements for storing intermediate results and parameters in hardware.

5. Arithmetic Intensity

   An optimization technique is largely determined by whether a NN is compute-bound or memory-bound, which is determined by its arithmetic intensity.

   Memory bandwidth is limited for workloads with low arithmetic intensity (<1–10 ops/byte); if memory cannot feed data quickly enough, increasing computation capabilities is useless [100]. Workloads with high arithmetic intensity (>100 ops/byte) are compute-limited; if compute resources are saturated, more memory bandwidth is squandered. Quantization and projection reduce data movement for memory-bound models, but pruning reduces MAC count for compute-bound models [101].

   Any access to memory, such as reading model weights from storage, loading input data, or displaying intermediate computation results, is referred to as a memory operation. Because they require data movement rather than arithmetic processing, these operations differ from computational operations (MACs). Memory requirements are not disclosed by MACs alone. Computational demand cannot be determined just by memory footprint. Compression strategy selection is guided by the arithmetic intensity metric, which offers direct insight into the performance bottleneck.

   The ratio of arithmetic operations to memory operations, which determines whether a workload is compute-bound or memory-bound:

$$\mathrm{ArithmeticIntensity} = \frac{\mathrm{TotalMemory}}{\mathrm{TotalMACs}}. \tag{3.15}$$

   This measure shows how effective processing is in relation to memory bandwidth needs. While low arithmetic intensity operators could be memory-bound, workloads with high arithmetic intensity (such as more than 100 operations per byte) benefit from compute accelerators [102] which is a specialized

hardware, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), or FPGA-based NN accelerators, that is made to execute mathematical operations effectively. These devices are particularly helpful for workloads with high arithmetic intensity, when the specific hardware investment is justified by the computing demands.

3.3 Bit-Exact Conversion and Hardware Code Generation

This section describes the methodical process used to produce synthesizable HDL code and convert the floating-point PCCFR and LUT DPD reference models to bit-exact fixed-point implementations.

1. Floating Point to Bit-Exact conversion:

   In order to determine signal ranges and crucial routes where precision is most vulnerable to quantization effects, the conversion procedure starts with a detailed examination of the floating-point reference models. The best fixed-point data types are then iteratively proposed using Simulink's Fixed-Point Tool, which analyzes bit consumption statistics to find overflows and underflows [15]. The program suggests data types that minimize hardware resource consumption and satisfy error tolerance requirements through iterative simulations with different word length and fraction length configurations. After optimization, all floating-point signals are swapped out for fixed-point data types (like ufix and sfix), with each signal's slope ($S = F \cdot 2^E$) and bias (B) set appropriately [15]. At this point, rounding and overflow handling modes (saturation or wrapping) are also provided. To guarantee numerical equivalency within allowable tolerances, the resulting fixed-point model is subsequently verified against the floating-point golden reference using extensive testbenches [43].

2. Model Preparation for HDL code generation:

   After the fixed-point model has been verified, the design is ready for hardware implementation by making sure that every subsystem makes use of HDL-compatible blocks from the DSP HDL Toolbox and Simulink block library. The data types for every block are carefully checked to make sure all signals are set up as integer or fixed-point types appropriate for hardware implementation. Every MATLAB function block is examined to make sure it uses fixed-point arithmetic and works with HDL generation. In order to handle important routes, pipeline registers are introduced when necessary. The model is organized hierarchically, with subsystems that match to RTL entity structures.

   All workspace blocks and simulation-specific elements (such stimulus sources, scopes, and visualization blocks) are commented out or eliminated from the design before HDL code is generated. This is due to the fact that only the fundamental inputs, outputs, variables, and functions needed for hardware implementation are included in the HDL code, which is generated just for the

core functionality and design architecture. The target device, clock frequency, and optimization choices including resource sharing, pipelining, and RAM mapping are then specified in order to configure the HDL Coder. Language version (VHDL or Verilog), coding style, and naming conventions are chosen.

3. HDL Code Generation and Resource Estimation:

   The HDL Workflow Advisor is used to generate HDL code, which results in intelligible, hierarchical, and well-documented VHDL or Verilog code for every subsystem. Workspace-dependent or simulation-only features are absent from the resulting code, which only contains hardware-relevant components such as inputs, outputs, internal variables, and functions. Following code generation, estimated hardware usage metrics are extracted by analyzing the high-level resource report generated by the HDL Coder code generation report. Estimates for the registers, multipliers, LUTs, and BRAMs needed for the implementation are among these measurements. In order to compare the resource requirements of NN-based solutions, the resource estimates derived from the code generation report are manually computed and tabulated. Without requiring complete synthesis or physical implementation, this comparison makes it possible to evaluate the fixed-point CFR, HL and DPD implementation's hardware efficiency in contrast to other methods, offering insightful information for design choices.

3.4 Resource Usage Estimation

In order to achieve a fair and consistent comparison of hardware complexity, the resource usage of both the legacy reference system and the suggested neural network architectures was estimated using a combination of automated toolchain reports from specially created Simulink models. The methodology was carefully developed to ensure that the comparison between the neural network solutions and the conventional signal processing chain was based on consistent, tool-derived hardware metrics, and all estimations were carried out for a target FPGA implementation.

To match the accuracy of the subsequent neural network implementations, the reference design for the legacy system was implemented in Simulink utilizing HDL-compatible blocks and floating-point data types [15]. The model was processed using the HDL Coder app in Simulink to produce RTL code in VHDL or Verilog. Key metrics, such as multipliers, adders and subtractors, registers, and total memory bits, were collected from a high-level resource consumption report that HDL Coder internally generated throughout this process. Higher-level abstractions like the number of MACs per sample, the total memory footprint in terms of registers and BRAMs, and LUT estimates for control and routing logic [97] [91], were then derived from these raw numbers, giving a reliable baseline for the hardware cost of the legacy method.

The compression and quantization procedures for the neural network alternatives were carried out completely in MATLAB, independently of the Simulink environment. A methodical compression workflow was applied to each network variant

with varying memory polynomial orders. This process started with pruning to eliminate unnecessary weights while maintaining accuracy, then projected onto a lower-dimensional space to further minimize storage needs, and finally quantized using the Deep Network Quantizer app. Crucially, only the inner hidden layers underwent this compression process; the input and output layers were purposefully retained at higher precision to preserve overall accuracy and prevent deterioration of the numerical interface with the rest of the system. To ensure that the final quantized models were reliable for hardware deployment, the networks were retrained using QAT after the initial quantization to recover any accuracy loss brought on by decreased precision [82]. The quantization details obtained in MATLAB were then used to calculate the resource usage of the compressed inner layers. MAC operations were calculated by counting the number of multiplications and additions in the quantized inner layers based on layer dimensions and activation functions; memory requirements were estimated by multiplying the bitwidth per parameter by the total number of quantized weight parameters and biases [96], and LUT and register estimates were derived from the calculated hardware mapping of the quantized operations using standard FPGA resource models for fixed-point arithmetic.

To construct the NN inputs and collect the NN outputs, a customized Simulink architecture was made, which was used to compute the MAC operations and resource usage of the data handling in order to precisely capture their hardware cost. A distinct Simulink model with delay lines for the necessary number of taps, the corresponding MATLAB Function block or an analogous HDL-compatible subsystem carrying out the summation, and interface blocks to match the system's overall data flow was built for each memory depth.

The HDL Coder workflow was then used to process these Simulink models, producing RTL and related resource reports for the NN data handling. The resource counts from the MATLAB-based compression analysis for the NN were then combined with the reported resource usage, which included adders, registers, and routing LUTs, to determine the total resource estimate for each NN variant. The HDL Coder workflow was then used to process these Simulink models, producing RTL and related resource reports for the NN data handling. The resource counts from the MATLAB-based compression analysis for the NN were then combined with the reported resource usage, which included adders, registers, and routing LUTs, to determine the total resource estimate for each neural network variant.

4

Implementation

4.1 Reference System Implementation

4.1.1 Signal Generation

The OFDM signal was generated according to the schematic diagram in Figure 2.2, i.e. QAM followed by OFDM modulation was applied to a binary signal. First the binary signal is generated, which for the purpose of this thesis is done by randomly generating a sequence of zeros and ones with length L. Then M-QAM is applied to the signal using a built-in MATLAB function. In this case the M refers to the number of symbols of the modulation, which corresponds to processing $\log_2(M)$ bits per time interval. As such the length of the modulated signal is reduced by a factor $\log_2(M)$. Finally, OFDM is applied to the M-QAM modulated signal. This is done by first dividing the modulated signal into several channels, one per OFDM symbol N_{sym}, each channel having the same length as the number of active subcarriers N_a, and is followed by applying a guard band, IFFT with oversampling and a CP. Everything except the channel division is done using a built-in MATLAB function.

The sample length used for FFT and IFFT N_{fft} is a power of 2, as such the sum of the number of active subcarriers and the number of guard band carriers N_{gbc} should be a power of two. To get the same number of guard band carriers on both sides of the active subcarriers, the number of guard band carriers on each side is calculated according to

$$N_{gbc} = \frac{1}{2}(N_{fft} - N_a), \tag{4.1}$$

where the number of active subcarriers is even, and the number of samples for the FFT and IFFT is chosen as the next power of two after the number of active subcarriers. The signal is oversampled with oversampling rate osr by applying further zero padding and applying IFFT of length $osr \cdot N_{fft}$.

As the OFDM modulation requires $N_{sym} \cdot N_a$ modulated signal samples and M-QAM reduces the number of samples with a factor $\log_2(M)$, the original binary signal should have length according to

$$L = N_a \cdot N_{sym} \cdot \log_2(M). \tag{4.2}$$

Furthermore, the length of the final OFDM signal, i.e. the number of samples Ns can be calculated according to

$$Ns = (N_{fft} + N_{cp}) \cdot osr \cdot N_{sym}. \tag{4.3}$$

The bandwidth Bw of such a signal can be computed as

$$Bw = N_a \cdot scs, \tag{4.4}$$

i.e. as the number of active subcarriers multiplied with the subcarrier spacing, and the sample rate Fs can be computed as

$$Fs = N_{fft} \cdot scs \cdot osr. \tag{4.5}$$

4.1.2 PCCFR Implementation

The general architecture and design of the PCCFR engine, which was created for high-performance wireless communication systems, are presented in this section. By lowering the PAPR of complex baseband signals, the PCCFR engine increases power amplifier efficiency and lowers out-of-band emissions. Bit-exact fixed-point arithmetic is used in the design's modular, cascaded architecture, which is suited for FPGA-based implementation. This section gives a high-level summary of the system architecture and the features of each main sub-block.

System Architecture Overview

Three similar processing stages referred to as ClipStage1, ClipStage2, and ClipStage3 connected in series form up the cascaded architecture used by the PCCFR which can be seen in Figure 4.1. Iterative peak reduction is made possible by this modular method, in which leftover peaks that were not completely eliminated by the preceding stage are further reduced in each subsequent stage. The selection of three phases offers the good balance between processing latency, hardware resource use, and PAPR reduction performance.

By processing the input signal through a number of functional blocks that function in a precisely scheduled sequence, each ClipStage creates a full peak cancellation loop. A fully pipelined control architecture oversees the entire system, guaranteeing correct synchronization between the data stream and control signals across all processing phases.

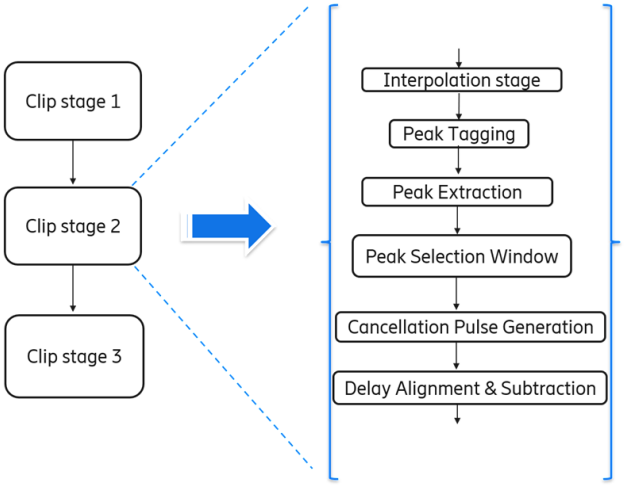


Figure 4.1: Flowchart of PCCFR implementation.

To reduce the hardware resource usage and processing latency, the number of clip stages can be reduced to two or even one clip stages, trading of the PCCFR performance. In the architecture, each clip stage cancel peaks remaining after the previous clip stage. By removing one or more clip stages the PAPR reduction performance will degrade and other performance metrics such as ACPR and EVM may also be affected.

Clipstage Architecture Overview

Peak detection and cancellation are carried out consecutively by the five main functional blocks that make up each ClipStage. A schematic block diagram for the implementation of clip stage n of a PCCFR engine, displaying the main processing blocks, can be seen in Figure 4.2, and the purpose and operation of each block is briefly described in the ensuing subsections.

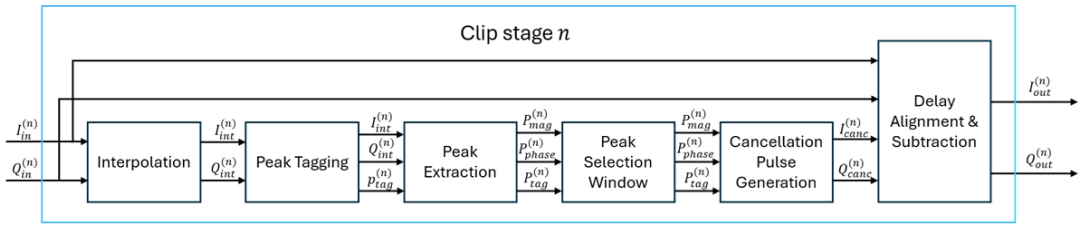


Figure 4.2: Schematic block diagram of PCCFR clip stage n implementation.

1. Interpolation Stage:

   By raising the signal sample rate, the interpolation stage improves peak detection accuracy. The sampling rate is essentially doubled by upsampling the I and Q components by a factor of two. The signals go via an interpolation filter after upsampling, which limits the signal bandwidth to avoid aliasing when reconstructing the continuous-time signal. In order to enable effective interpolation while preserving signal integrity, the filter is constructed as a half-band FIR filter with symmetric coefficients.

   In order to provide a higher-resolution representation of the signal envelope and for more accurate peak localization, the interpolation stage generates oversampled and filtered signal components. In order to effectively detect and cancel out peaks that arise between the original sample points, this step is crucial.

2. Peak Detection and Extraction:

   Samples that exceed a configurable clipping threshold are identified by the peak detection and extraction block, which also extracts the features of detected peaks. After receiving the interpolated I/Q signals, the block transforms the Cartesian I/Q samples into polar coordinates (magnitude and phase) using the Coordinate Rotation Digital Computer (CORDIC) method.

   Any sample that exceeds the clipping threshold is considered as a component of a peak that needs to be canceled. The current magnitude is compared to the

threshold. A complex peak tracking algorithm tracks each peak's evolution and determines its maximum magnitude, matching phase, and timing information while preserving state information across samples. The block outputs the peak properties, such as the excess magnitude, phase angle, index location, and delay information, when a peak ends.

3. Peak Selection Window:

   The crucial task of choosing the finest peak within a time window when several peaks occur nearby is carried out by the peak selection window block. This selection is crucial to avoid over-cancellation and signal distortion caused by the cancellation pulse generator being overloaded by several closely spaced peaks.

   Within a fixed-duration search window, the block implements a finite state machine that methodically assesses incoming peaks to identify which one represents the most important event that has to be canceled. When the window closes, the block reports the chosen peak parameters after tracking the peak with the highest magnitude during the window.

4. Cancellation Pulse Generator:

   The most advanced part of the CFR engine is the cancellation pulse generator, which creates cancellation pulses that correspond to the spectrum properties of the signal. The block keeps track of a queue of concurrently active pulses and combines the contributions of each active pulse to produce the composite cancellation signal.

   In order to guarantee that the energy of the cancellation pulses stays contained within the signal bandwidth, the cancellation pulse shape is pre-calculated and stored in a lookup table. The block avoids over-cancellation by performing inter-pulse interference compensation to account for overlapping pulses when a new peak is identified. For every active pulse, the block converts polar coordinates (magnitude and phase) back to Cartesian coordinates (I/Q components) using the CORDIC method. These contributions are then added up to create the final cancellation signal.

5. Delay Alignment and Subtraction Stage:

   Each ClipStage's last stage subtracts the generated cancellation pulse from the delayed original signal to carry out the real peak cancellation. To ensure that the cancellation pulse perfectly coincides with the peak it is meant to cancel, the original I/Q signals are routed via a configurable delay line that matches the overall latency through the interpolation, peak detection, peak selection, and pulse production blocks.

   In order to eliminate the excess magnitude from the peak and bring it down to the clipping threshold level, the subtraction operation is carried out separately for the I and Q components. The resulting signals are forwarded to the subsequent stage in the cascade and constitute the peak-reduced output of the ClipStage.

4.1.3 Hard Limiter Implementation

In this thesis a CFR with a separate HL is used. As perviosuly explained the HL is implemented using the HC model defined in Equation (2.21). The model applied to a single sample can thus be described as

$$u_{HL} = \begin{cases} u(n), & |u(n)| \leq A_{ct}, \\ A_{ct} \cdot e^{i\angle u(n)}, & |u(n)| > A_{ct}, \end{cases} \tag{4.6}$$

where $\angle u(n)$ is the angular argument of the signal sample $u(n)$, and A_{ct} is the same clipping threshold used by the CFR model.

4.1.4 Digital Pre-Distortion Implementation

The reference DPD model used in this thesis is a MP model implemented with a LUT based dual polyphase DPD feed-forward architecture. The LUT architecture used is 1D LUTs with complex coefficient values and addresses are computed using the input magnitude, the LUT spacing is linear and no interpolation is used. Specifically, the address generator computes the LUT index as

$$idx(m) = \text{round}\left((idx_{max} - 1) \cdot \frac{m - m_{min}}{m_{max} - m_{min}}\right),$$

where idx_{max} is the maximum index of the LUT, and m_{min} and m_{max} are the expected minimum and maximum magnitudes of the type of signal used respectively. For any input magnitude outside the expected range, the address generator will map the magnitude to LUT index 1 or idx_{max} for magnitudes lower than the expected minimum or higher than the expected maximum respectively. The LUT values are updated using real time adaption with the LMS method using DLA according to Equation (2.36), and initialized with unit gain. Initializing with unit gain means that LUT 0, corresponding to memory depth zero or the current sample, is initialized with all entries one and all other LUTs are initialized with all entries zero, meaning that before the first update $x(n) = f_{\text{DPD}}(u(n)) = u(n)$.

For simplicity, let us first consider the non-polyphase case. In Figure 4.3 a schematic diagram of the top level implementation of a non-polyphase LUT DPD and LUT update logic can be seen.

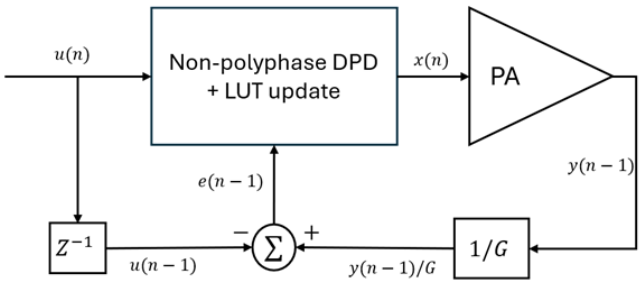


Figure 4.3: Schematic block diagram of non-polyphase LUT DPD and LUT update logic.

A schematic block diagram of the non-polyphase LUT DPD feed-forward path can be seen in Figure 4.4. In the upper part of the figure the same non-polyphase DPD feed-forward path from Figure 2.8 can be seen. The lower part of the figure shows a delay chain with the LUT addresses $u_{idx}(n-k)$ which are computed by passing the input through the magnitude calculation block marked '$|\cdot|$' and the address generator block. The delay chain with the LUT addresses is passed through the different LUTs giving the corresponding coefficient value outputs $h_k(n)$.

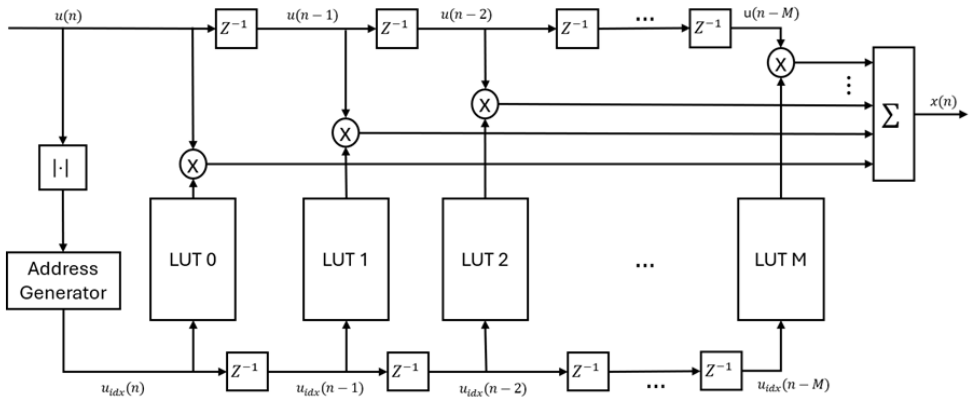


Figure 4.4: Schematic block diagram of a non-polyphase LUT DPD feed-forward path.

A schematic block diagram of the use and update of LUT k for a non-polyphase LUT DPD can be seen in Figure 4.5. To the right in the figure is LUT k as seen in the feed-forward path with address input delayed by k samples $u_{idx}(n-k)$, coefficient value output $h_k(n)$, and the constant 'addresses' block which handles the possible addresses of the LUT. To the left in the figure is a LUT update block which handles the DLA LMS update of the LUT with address input delayed by $k+1$ samples $u_{idx}((n-1)-k)$, input delayed by $k+1$ samples $u((n-1)-k)$, error delayed by 1 sample $e(n-1)$, and constants LUT_k^0 containing the initial LUT values and μ the update step size. The update blocks compute the update according to Equation (4.7).

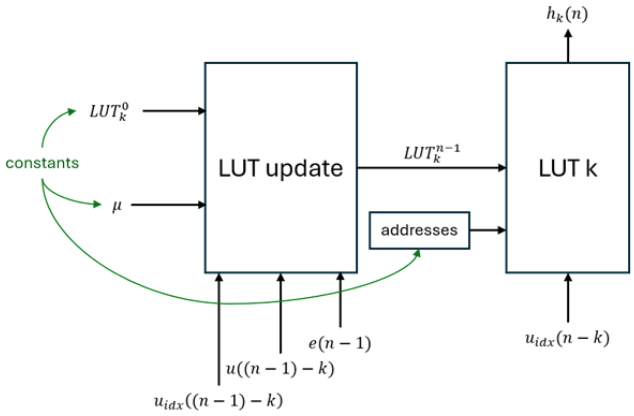


Figure 4.5: Schematic block diagram of non-polyphase LUT DPD update implementation.

$$LUT_k^{\text{new}}(|u(n-k)|) = LUT_k^{\text{old}} - \mu u^*(n-k)e(n),\ e(n) = y(n)/G - u(n) \tag{4.7}$$

Moving on to polyphase case, in Figure 4.6 a schematic diagram of the top level implementation of a non-polyphase LUT DPD and LUT update logic can be seen.

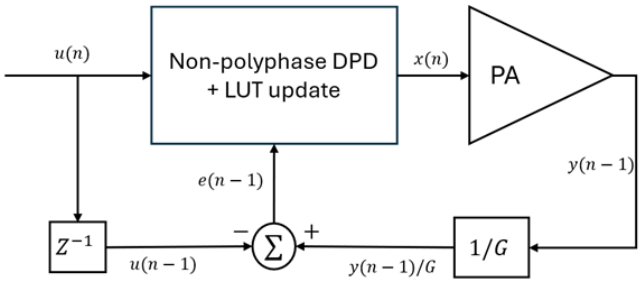


Figure 4.6: Schematic block diagram of a polyphase LUT DPD and LUT update logic.

A schematic block diagram of a dual polyphase LUT DPD feed-forward architecture can be seen in Figure 4.7. In the upper and lower part of the figure the dual feed-forward delay chains can be seen with the even and odd samples and the information exchange between them can be seen to the right, same as in Figure 2.10. To the left in the figure dual magnitude calculation blocks and address generation blocks can be seen and in the middle part of the figure dual address delay chains can be seen. The delays of the address delay chains match the corresponding input delay chains and the delayed addresses are passed through the corresponding shared LUTs giving the corresponding LUT coefficient values. The red lines in the figure indicate the signal paths and are used only for visual differentiation.

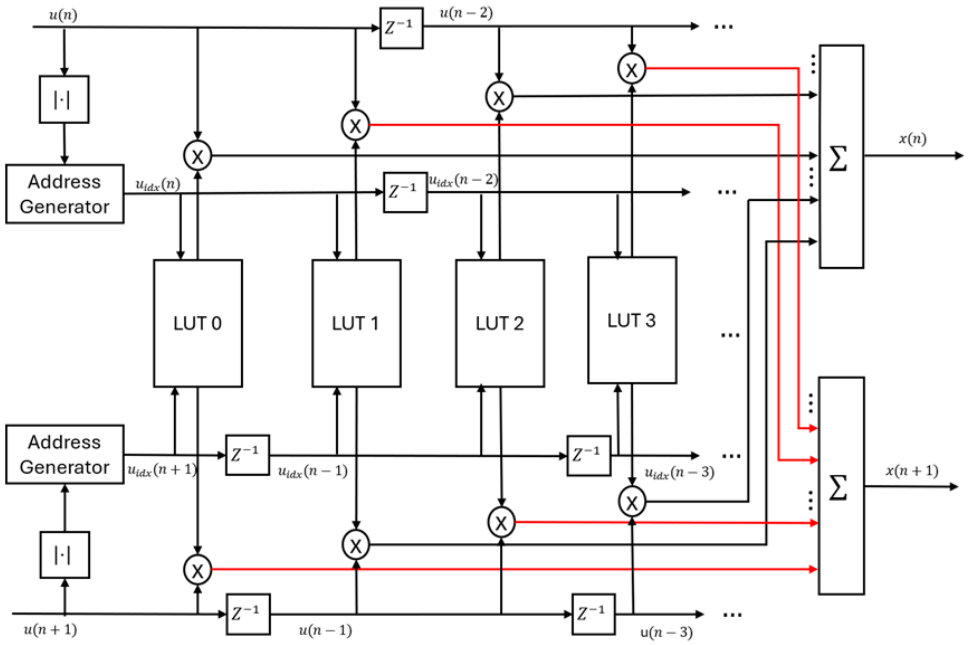


Figure 4.7: Schematic block diagram of dual polyphase LUT DPD feed-forward path.

For the dual polyphase case the DLA LMS update looks slightly different for even and odd numbered LUTs. A schematic block diagram of the use and update of even numbered LUT k for a dual polyphase LUT DPD can be seen in Figure 4.8. To the right in the figure is LUT k which has the address input delayed k samples $u_{idx}(n-k)$ and coefficient value output $h_k(n)$ for the even (upper) delay chain, and the address input delayed k samples $u_{idx}((n+1)-k)$ and coefficient value output $h_k(n+1)$ for the odd (lower) delay chain. The LUT also has the constant 'addresses' block which handles the potential addresses of the LUT. To the left in the figure is the LUT update block which handles the DLA LMS update of the LUT. The LUT update block has the following inputs: has the address input delayed $k+2$ samples $u_{idx}((n-2)-k)$, the input delayed $k+2$ samples $u_{idx}((n-2)-k)$ and coefficient value output delayed by 2 samples $h_k(n-2)$ for the even (upper) delay chain, and the address input delayed $k+2$ samples $u_{idx}((n-1)-k)$, input delayed $k+2$ samples $u((n-1)-k)$ and coefficient value output delayed by 2 samples $h_k(n+1)$ for the odd (lower) delay chain. Furthermore it has the constants LUT_k^0 and μ which are again the initial LUT values and update step size, and outputs the updated LUT values. The update blocks computes the update according to Equation (4.8), where the vector multiplication at the end is element-wise.

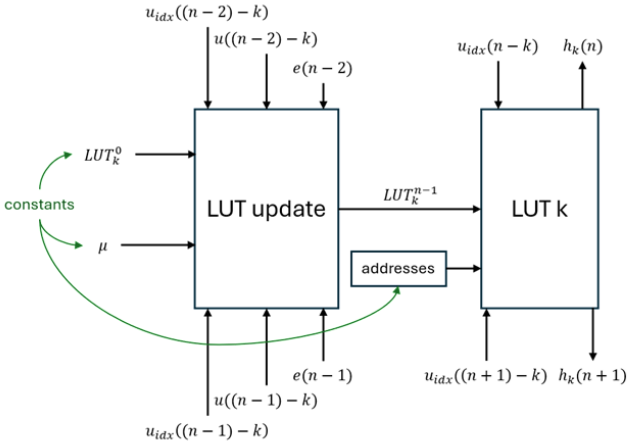


Figure 4.8: Schematic block diagram of polyphase LUT DPD update implementation (k even).

$$LUT_k^{\text{new}}\left(\begin{vmatrix} u(n-k) \\ u((n+1)-k) \end{vmatrix}\right) = LUT_k^{\text{old}}\left(\begin{vmatrix} u(n-k) \\ u((n+1)-k) \end{vmatrix}\right) - \mu \begin{pmatrix} u^*(n-k) \\ u^*((n+1)-k) \end{pmatrix} \begin{pmatrix} e(n) \\ e(n+1) \end{pmatrix} \quad (4.8)$$

A schematic block diagram of the use and update of odd numbered LUT k for a dual polyphase LUT DPD can be seen in Figure 4.9. As can be seen from the figure it is very similar to Figure 4.8 with the only difference being that the inputs and outputs from and to the even (upper) and odd (lower) delay chains have switched place. The update blocks computes the update according to (4.9), which also is very similar to Equation (4.8) with the only difference being that the two rows of the equation has switched place, again the vector multiplication is element-wise.

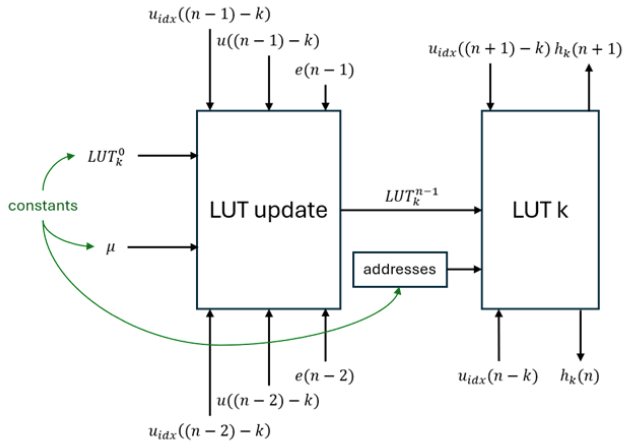


Figure 4.9: Schematic block diagram of polyphase LUT DPD update implementation (k odd).

$$LUT_k^{\text{new}}\left(\begin{vmatrix} u((n+1)-k) \\ u(n-k) \end{vmatrix}\right) = LUT_k^{\text{old}}\left(\begin{vmatrix} u((n+1)-k) \\ u(n-k) \end{vmatrix}\right) - \mu \begin{pmatrix} u^*((n+1)-k) \\ u^*(n-k) \end{pmatrix} \begin{pmatrix} e(n+1) \\ e(n) \end{pmatrix} \quad (4.9)$$

The update of the LUT values for implementations of both the non-polyphase and dual polyphase LUT DPD can be turned of. This might be desirable in case of performance evaluation or in case ACPR starts degrading.

4.1.5 Power Amplifier Implementation

The PA used in this thesis for the reference system is an MP modeled after data of a signal passed through a PA. The data used to extract the PA model was taken from [103], which was collected from an NXP Airfast LDMOS Doherty PA with operating frequency 3.6-3.8 GHz and 29 dB gain using a 100 MHz OFDM test signal.

When looking at the actual data it can be noted that $N = 48384$ sets of PA inputs and outputs are provided and that the linear gain of the data is $G = 23.3998$ corresponding to 27.3842 dB. It can also be noted that the magnitude of the input signal is in the range $|x(n)| \in [0, 2.4001]$ and that the output signal is in the range $|y(n)| \in [0, 48.8576]$, as well as the signal is oversampled with a factor 7.

Furthermore, the characteristics of the PA shows a lot of signal distortions, but not much in terms of signal compression. Typically, a hardware PA has more compression. As such a modified version of the data is used with additional compression introduced artificially, to get a more realistic PA model. This modification is done by multiplying the PA output signal y with a compressing function according to

$$y_{\text{mod}}(n) = y(n) \cdot f(n). \quad (4.10)$$

The compressing function is given by

$$f(n) = \frac{1}{1 + \frac{|x(n)|}{3 \cdot \max_n |x(n)|}}, \quad (4.11)$$

where x is the input signal, and takes values from 1 to 0.75, providing up to 25% additional compression for larger input magnitudes. The modification of the data only affects the amplitude of the data, meaning that the phase distortions of the modified and original PA data as well as the linear gain is the same. The compression of the data gives PA saturation $V_{sat} = 37.5758$. A visualization of the amplitude and phase distortions, as well as the spectrum of the original and modified PA output data can be seen in Figures 4.10 and 4.11.

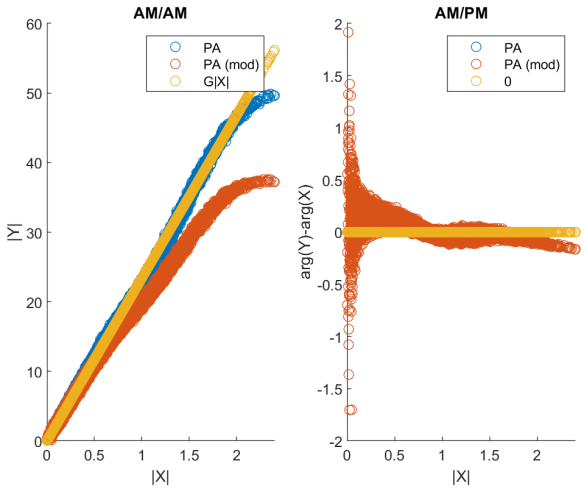


Figure 4.10: Amplitude and phase distortions for the original and modified PA data.

In Figure 4.10 it can be seen from the AM/AM diagram that the modified PA data has more compression of the amplitude for higher input amplitudes than for the original PA data. In the AM/PM diagram of the figure the phase distortions for the original PA data can not be seen since the phase is not affected by the modifications for the modified PA data and the phase distortions of the original and modified PA data coincides perfectly.

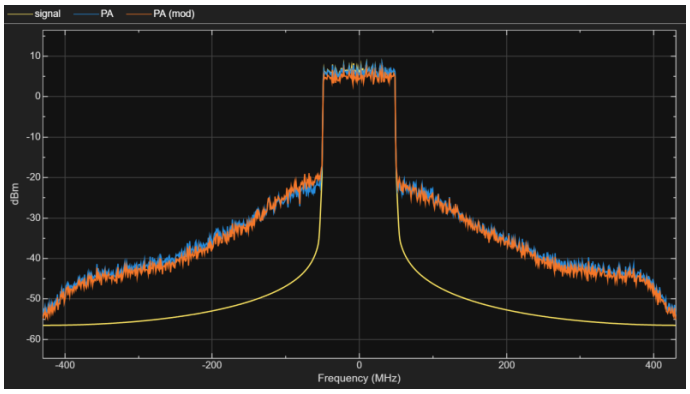


Figure 4.11: Signal spectrum for the PA input signal, and original and modified PA output signal.

From Figure 4.11 it can be seen that the modified PA data has a similar signal spectrum as the original PA data.

The MP model is given in Equation (2.26) and identification of the PA MP model coefficients are done in accordance with the direct PA model identification method described in (2.42) using the modified PA data. Using this method models of different nonlinearity orders and memory depths have been extracted. If a PA model is applied to an input signal with a different magnitude range than the signal used

for model identification, the signal is scaled to have the same range before passing through the PA and then scaled back.

A couple of MP PA models are extracted with nonlinearity order $K = 9$ and varying memory depth M. For cases where the PA model input is outside the input magnitude range it was extracted for, the PA model behaves erratically. For this reason a limiter was applied which suppresses large model output values for input magnitudes outside the range the model was extracted for.

4.2 Specifications

4.2.1 Signal Specifications

All signals used in this thesis, to compute the LUT updates of the DPD model, for training the NN solutions and for evaluating the performance of the legacy and NN solution, are 100 MHz signals modulated using 256-QAM. The specifications used for the signal generation can be found in Table 4.1.

Parameter	**Value**
FFT length (N_{fft})	2048
Number of active sub-carriers (N_a)	1668
CP length (N_{cp})	72
Oversampling rate (osr)	3
Number of OFDM symbols (N_{sym})	100
QAM order (M)	256
Sub-carrier spacing (scs)	60 kHz

Table 4.1: Signal specifications

An example of a signal generated in accordance with these specifications can be found in the Section 2.4.1 and the attributes of such a signal can be seen in Table 4.2. From the table it can be seen that the binary sample length is 1334400, specifying the amount of information that can be transmitted with one signal. It can also be seen that the number of signal samples is 636000 and the sample rate is approximately 300 MHz, meaning that in order to transmit a signal with these specifications, the system must process the entire signal with this sample rate.

Parameter	**Value**
Binary signal length (L)	1334400
Number of samples (Ns)	636000
Bandwidth (Bw)	~ 100 MHz
Sample rate (Fs)	~ 300 MHz
Expected maximum amplitude	0.085
Expected standard deviation	$0.0199 - 0.02$

Table 4.2: Signal attributes

A number of signals were pre-generated and used to update the LUT values of the LUT DPD and for training the NNs, such that the same signals were used for updates and trainings of all legacy system simulations and NNs. All evaluations were made using the same pre-generated signal, which was not used for the update of LUT values or NN trainings.

4.2.2 PA Specifications

As explained previously in Section 4.1.5 a couple of MP PA models were extracted from the modified data, with nonlinearity order $K = 9$ and varying memory depth M. The models were extracted with memory depths $M = 0$, 2, 4 and 6, and will be referred to as MP0, MP2, MP4 and MP6 PA models respectively. These models have been scaled to the input signal magnitude range $[0, 0.085]$ as this is the expected signal magnitude range of the signals used. The linear gain of the models are $G = 23.3998$ and model saturation becomes $V_{sat} = 1.24$.

An important note is that as previously mentioned the data from which the models were extracted consisted of $N = 48384$ sets of PA inputs and outputs, which is not a lot when compared to the $N = 636000$ signal samples of the signals used in this thesis. Furthermore the data has been modified prior to model extraction for additional compression and the model was then scaled to account for the different input signal magnitude ranges of the data and the signals used for this thesis. As such the models may not be that accurate.

The legacy and NN solution has been applied to and optimized for all four MP PA extracted. The results presented here have been restricted to the PA model with the most memory taps, MP6, as this is the most complex PA model extracted and as such the most challenging PA model to handle for the legacy and NN DFE solutions.

A visualization of the MP6 PA model applied to the evaluation signal can be seen in Figures 4.12, 4.13, 4.15 and 4.14, where the AM/AM and AM/PM distortions, envelope, constellation diagram and spectrum diagram of the model applied to the evaluation signal can be seen respectively.

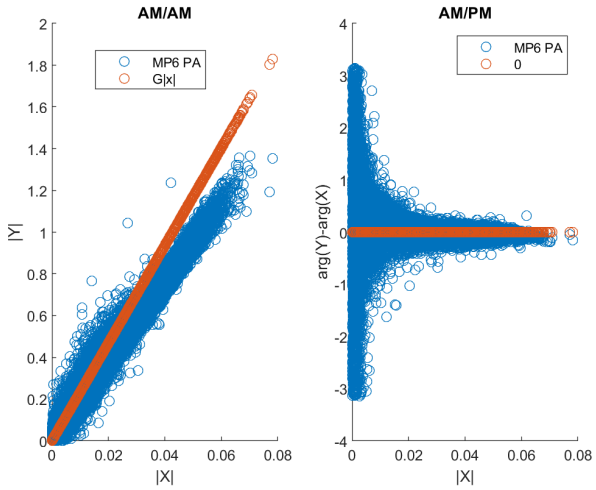


Figure 4.12: Visualization of AM/AM and AM/PM distortions for the MP6 PA model applied to the evaluation signal.

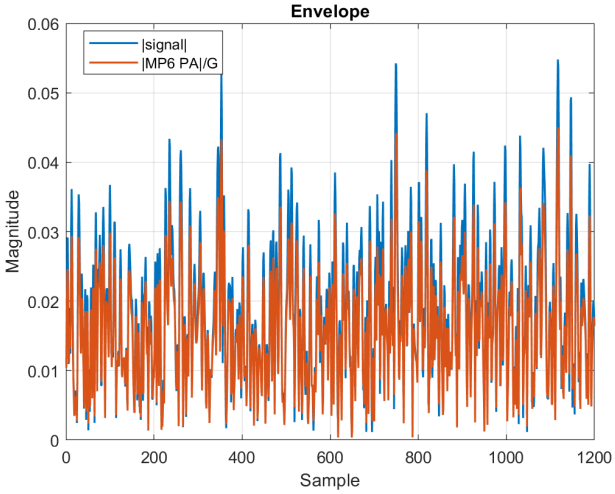


Figure 4.13: Visualization of the envelope of the first 1200 samples for the MP6 PA model applied to the evaluation signal.

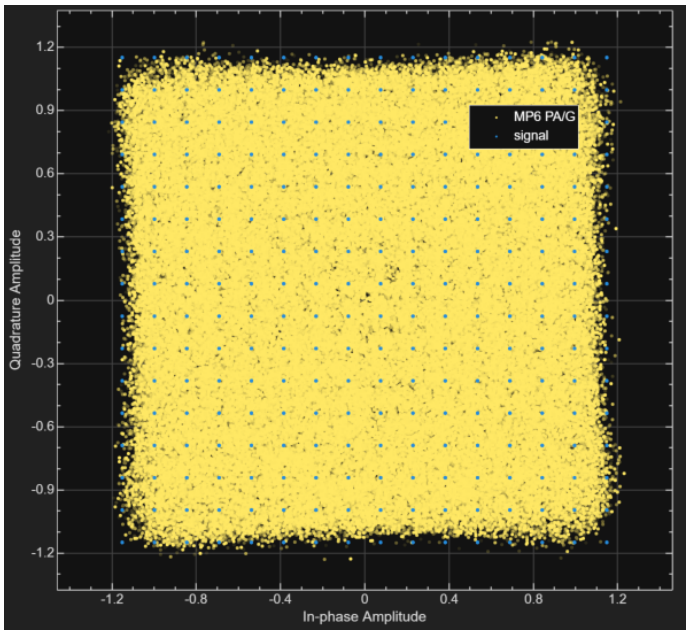


Figure 4.14: Visualization of the constellation diagram for the MP6 PA model applied to the evaluation signal.

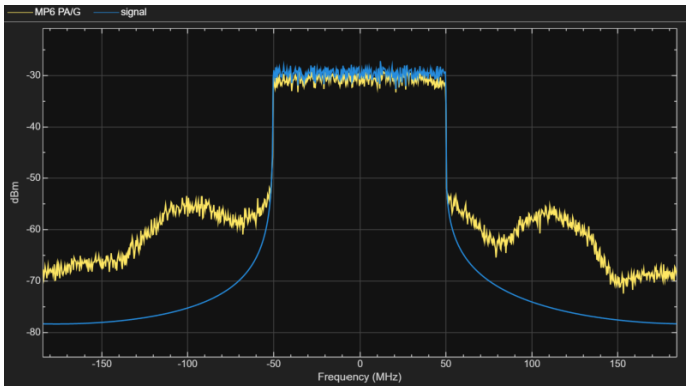


Figure 4.15: Visualization of the spectrum for the MP6 PA model applied to the evaluation signal.

The performance of the MP6 PA model evaluated on the evaluation signal can be found in Table 4.3, along with the signal performance prior to amplification.

Stage	ACPR (dBc)		EVM (dB)	PAPR (dB)
signal	−41.893	− 42.148	$-\infty$	11.863
MP6 PA	−27.994	− 29.530	−15.830	10.460

Table 4.3: Performance of evaluation signal prior to amplification and of PA MP6 model applied to evaluation signal.

4.2.3 Limit Specifications

Investigations regarding PAPR reduction using the legacy CFR and HL as well as NN solution considering joint DFE have been conducted for the PAPR thresholds 9, 8.5, 8, 7.5, 7 and 6.5 dB. Greater PAPR reductions pose a greater challenge as the risk for introducing signal distortions increase with the PAPR reduction, showcasing the CFR performance, and it is also sought after for greater PA efficiency. The results are presented for each PAPR limit when CFR, but in depth analysis and comparison will be limited to the PAPR threshold 6.5 dB, as this is the greatest PAPR reduction investigated and hence the most challenging.

4.3 Neural Network Solution Implementation

4.3.1 Joint and Co-optimized Digital Front End Neural Network implementation

Data

The data used to train, validate and test the joint co-optimized CFR and DPD NN solution is the same signals used for the reference system. For the training of the NN, one or multiple signals are used and one signal each is used for the validation and testing of the NN. All signals used for a specific NN are generated using the same signal specifications.

As NN input the signals are used and for NN output target the same signals multiplied with the linear gain is used (DLA is used for training, see the learning section). As a preprocessing step the data is scaled by dividing with the standard deviation of one of the signals used for the training data, in order to normalize the data, same as in [60] and [61]. If the PA is applied during the training phase the data is scaled back by multiplying with the standard deviation before propagating through the PA to then be re-scaled.

Architecture

The NN architecture used in this thesis to jointly model the CFR and DPD is a real-valued fully connected NN with the ReLU activation function and a weighted skip-connection between the input and output layer. The NN has four hidden layers with 64, 32, 16 and 32 hidden units respectively, and a Cartesian signal representation is used for the inputs and outputs. The number of hidden layers and hidden units per layers used in the architecture here is the same as in [57] and [58]. For the input layer the envelope of the input signal is given on top of the real and imaginary parts of the signal, and to incorporate memory M delayed samples of the input signal are also given. An illustration of the NN architecture can be seen in Figure 4.16.

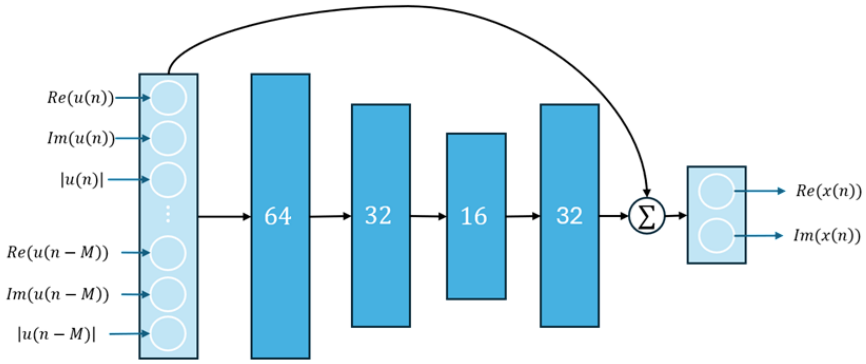


Figure 4.16: Schematic diagram of the NN architecture used in this thesis.

In some attempts to model CFR or jointly model CFR and DPD using NNs, multiple NNs were used during the modulation and demodulation of the signal and before the PA. However in most attempts to model DPD or jointly model CFR and DPD using NNs, a single NN was used. As such a single NN was used here as it is common across the literature and decrease the complexity. The dual NN solution in [1] has low complexity. But it is also designed specifically to model DPDs for memoryless PAs and was therefore not investigated further. The architecture here also has a skip connection between the input and output layers which was used in several previous attempts.

Across the literature both Cartesian and polar coordinate signal representations are used. It appears to be somewhat more common to use Cartesian signal representations, especially in solutions that account for memory effects when modeling DPD and solutions that jointly model CFR and DPD. As such a Cartesian signal representation is used here. To account for memory effects a mixture of the inputs used in the paper [59] and the MATLAB documentation [60] and [61] was used, in which the same memory depth was used for all terms and the envelope $|u(n-m)|$ was included but no higher order nonlinearities $|u(n-m)|^k$.

A mathematical description of the NN forward propagation is given by

$$\begin{aligned} \mathbf{q}^{(0)} &= \mathbf{u}, \\ \mathbf{q}^{(l)} &= h_{ReLU}(\mathbf{W}^{(l)}\mathbf{q}^{(l-1)} + \mathbf{b}^{(l)}) \text{ for } l = 1, 2, 3 \text{ and } 4, \\ \mathbf{q}^{(5)} &= \mathbf{W}^{(5)}\mathbf{q}^{(4)} + \mathbf{b}^{(5)} + \mathbf{W}^{(0\to5)}\mathbf{q}^{(0)}, \\ \mathbf{x} &= \mathbf{q}^{(5)}, \end{aligned} \tag{4.12}$$

where the input $\mathbf{u}$ and output $\mathbf{x}$ are given by

$$\begin{aligned} \mathbf{u}(n) = (&Re(u(n))\ Im(u(n))\ |u(n)|,\ \ldots, \\ &Re(u(n-m))\ Im(u(n-m))\ |u(n-m)|,\ \ldots, \\ &Re(u(n-M))\ Im(u(n-M))\ |u(n-M)|)^T \\ \mathbf{x}(n) = &(Re(x(n))\ Im(x(n)))^T. \end{aligned} \tag{4.13}$$

Let $W^{(x)}$ and $b^{(x)}$ denote the number of weights and biases of $\mathbf{W}^{(x)}$ and $\mathbf{b}^{(x)}$ respectively. The degrees of freedom for the NN are given by the total number of trainable

parameters of the NN which is

$$\begin{aligned} DoF &= W^{(1)}{+}b^{(1)}{+}W^{(2)}{+}b^{(2)}{+}W^{(3)}{+}b^{(3)}{+}W^{(4)}{+}b^{(4)}{+}W^{(5)}{+}b^{(5)}{+}W^{(0\to5)} = \\ &= 64{\cdot}3(M{+}1){+}64{+}32{\cdot}64{+}32{+}16{\cdot}32{+}16{+}32{\cdot}16{+}32{+}2{\cdot}32{+}2{+}2{\cdot}3(M{+}1) = \\ &= 3480 + 198M, \end{aligned} \tag{4.14}$$

as there are $3(M+1)$ inputs for memory depth M. For simplicity the NN output as a function of the input will henceforth be written as

$$x = f_{\mathrm{DPD}}(u(n)). \tag{4.15}$$

Learning

The training of the NN is done using the adaptive learning algorithm Adam with a joint loss function for optimizing the CFR and DPD aspects simultaneously, as suggested in [57] and [58], and implemented in [13]. The joint loss function is given by

$$L_{\mathrm{joint}} = w_{\mathrm{Lin}}L_{\mathrm{Lin}} + w_{\mathrm{OOB}}L_{\mathrm{OOB}} + w_{\mathrm{PAPR}}L_{\mathrm{PAPR}} + w_{\mathrm{Reg}}L_{\mathrm{Reg}}, \tag{4.16}$$

where w_{Lin}, w_{OOB}, w_{PAPR} and w_{Reg} are the weights of the respective loss functions. In the following an in depth explanation of each of the loss functions are given:

- Linear loss:

$$\begin{aligned} L_{\mathrm{Lin}} &= \sqrt{NMSE(f_{\mathrm{PA}}(f_{\mathrm{DPD}}(U), G \cdot U)} \\ &= \sqrt{\frac{\frac{1}{N}\sum_{n=1}^{N}|f_{\mathrm{PA}}(f_{\mathrm{DPD}}(u(n))) - G \cdot u(n)|^2}{\frac{1}{N}\sum_{n=1}^{N}|G \cdot u(n)|^2}} \end{aligned} \tag{4.17}$$

  The linear loss function uses DLA to evaluate how well the NN linearizes the PA. It is the square root of the NMSE function between the NN output propagated through a MP model of the PA and the desired output. The loss function is closely related to the computation of the EVM, which measures the difference between the system input and output, and is used to optimize the PA linearization of the NN in terms of EVM performance.

- Out-of-Band loss:

$$L_{\mathrm{OOB}} = \sqrt{\frac{\sum_{i\in\mathrm{oobMask}} P(i)}{\sum_{i\in\mathrm{mainMask}} P(i)}}, \text{ where } P(n) = (f_{\mathrm{PA}}(f_{\mathrm{DPD}}(u(n))))^2 \tag{4.18}$$

  The out-of-band loss function uses DLA to evaluate how much of the system output power leaks outside the main channel, where the system output is computed by by propagating the NN output through a MP model of the PA. It is closely related to the computation of the PAPR, with the exception that the total out-of-band power is divided by the total main power and not just the power of an adjacent channel, furthermore the square root is applied. The loss function is used to optimize the PA linearization in terms of ACPR performance.

- PAPR loss:

$$L_{\text{PAPR}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} \text{excess}(n)^2}, \text{ where excess}(n) = \max(0, P(n) - P_{\text{lim}}), \\ P(n) = (f_{\text{DPD}}(u(n)))^2 \ \& \ P_{\text{lim}} = P_{\text{avg}} \cdot 10^{PAPR_{\text{lim}}/10} \tag{4.19}$$

  The PAPR loss function evaluates how well the NN output power adheres to a set PAPR limit. It is defined as the square root of the mean squared power excess, i.e. the power above the power limit corresponding to the set PAPR limit computed by inverting the PAPR computation. The loss function is used to optimize the PAPR reduction of the NN according to the desired threshold, under the assumption that the PAPR of the mini-batch will accurately represent the PAPR of the signal given a large enough mini-batch.

- Regularization loss:

$$L_{\text{Reg}} = \frac{1}{N}\sum_{n=1}^{N} |f_{\text{DPD}}(u(n))|^2 \tag{4.20}$$

  The regularization loss is meant to in a way act as a regularization term similar to L2-Regularization, however instead of the weights the NN output is used. The purpose of this loss function is to prevent the NN output from growing to large giving erratic system behavior.

Inference

The trained NN can be used to make predictions on new data using the forward propagation in (4.12) as part of the DFE replacing the conventional CFR and DPD components. However, in order to used the trained NN for a hardware implementation of the system, NN compression must be applied. Furthermore, when using the compressed NN as part of the system for simulations, a Simulink model is used to construct the inputs and collect the outputs for the compressed NN. To construct the input for a NN with memory depth M, the input signal is split into its real, imaginary and envelope components with M delay blocks, and the real and imaginary parts of the NN are collected and combined. This is illustrated in Figure 4.17.

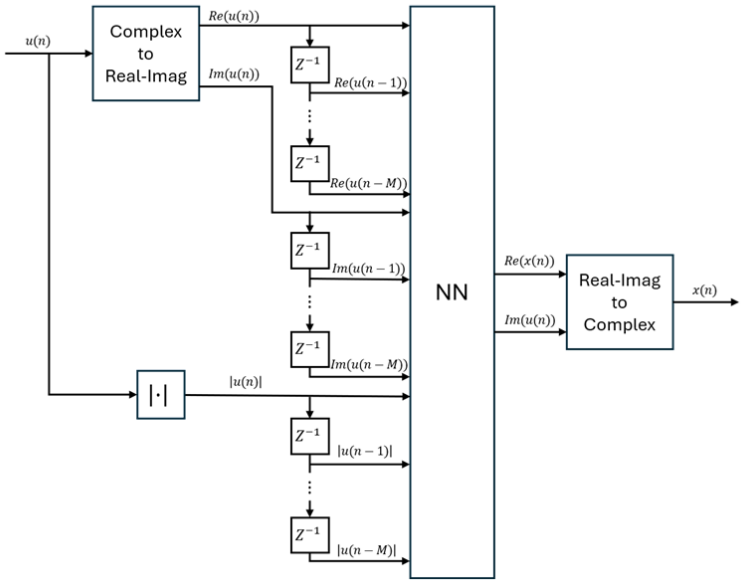


Figure 4.17: Schematic diagram of the construction and collection of input and output data for a NN with memory depth M during inference.

4.3.2 Neural Network compression

DNN's increasing size and computational requirements pose serious obstacles to their deployment in resource-constrained environments like mobile devices, embedded systems, and real-time communication systems, even as they continue to achieve state-of-the-art performance across a variety of applications [88]. NN compression is a collection of methods that aim to maintain acceptable performance levels while reducing model complexity by parameter minimization, numerical accuracy reduction, or NN architectural reformulation [81]. Reducing memory footprint, increasing inference speed, lowering power consumption, and enabling deployment on edge hardware without significant accuracy deterioration are the main goals [96]. In this work, we examined several compression techniques in three primary categories: quantization (which reduces numerical precision), projection (which reduces dimensions by linear transformations), and pruning (which eliminates unnecessary parameters). The effectiveness of each method is assessed in the context of NN-based DPD for PA linearization, and each offers unique trade-offs between compression ratio and performance preservation.

1. Pruning

   One of the most basic methods for NN compression is pruning, which is based on the idea that many parameters in over-parameterized NNs contribute very little to the final output and can therefore be removed [104]. The basic idea is to assess the significance of parameters using a variety of criteria, eliminate those that are considered unimportant, and then fine-tune to restore any lost performance [101]. The main control parameter is the sparsity level, also known as the pruning ratio, which normally ranges from 30% to 90% depending

on the NN architecture design and application needs . In this work, three different pruning approaches were investigated, each with special benefits and varying granularities of operation.

- Structural Pruning: Because structural pruning creates regular, dense weight matrices, it is especially well suited for hardware acceleration because it approaches compression at the neuron or filter level [73]. By calculating the L2 norm across all incoming connections to each output neuron, the approach determines significance scores for each neuron. The significance score for neuron i in a weight matrix W, where each row represents an output neuron, is calculated as $\sqrt{W_i^{12}} = \sqrt{\sum_j W_{ij}}$. Only neurons with scores over this threshold are kept after the quantile function is used to determine the threshold corresponding to the goal sparsity level; all other neurons are zeroed out. In order to enable effective implementation on GPUs and specialized accelerators that profit from dense matrix operations, this method successfully decreases the NN's width while maintaining regular structure.
- Magnitude-based Pruning: This is the simplest and most used pruning method, which eliminates weights with the lowest absolute values at the individual parameter level. After evaluating each weight separately using the importance score $S = |W_{ij}|$, the approach globally ranks all parameters and applies a binary mask $M_{ij} = 1$ if $|W_{ij}| > \tau$ and 0 otherwise, where τ is the threshold at the sparsity percentile. Because it may choose remove any weight while preserving the overall NN structure, this method is computationally economical and usually maintains model accuracy better than structural pruning at similar sparsity levels [71]. Nevertheless, it may eliminate weights that, although minor on their own, collectively make a significant contribution to the NN's functionality and produce erratic sparsity patterns that might not fully utilize hardware optimizations.
- SynFlow Pruning: This evaluates parameter relevance based on gradient flow sensitivity rather than absolute values, which overcomes the basic drawbacks of magnitude-based pruning [75]. This method is especially useful because magnitude-based pruning requires pre-training before pruning can be successful because it fails at initiation when all weights are small and random. In order to calculate significance scores, SynFlow uses a dummy all-ones input to compute gradients with respect to the sum of all outputs. These gradients are then combined with the weights themselves to compute important scores as $S = \nabla L(W) \odot W$ with $\odot$ denoting element-wise multiplication. Pruning proceeds by eliminating the weights with the lowest scores after taking the absolute values to guarantee positive scores. This technique allows for efficient pruning without the need for training data or pre-training of the model [75], and maintains the whole gradient flow across the NN, eliminating layer collapse where an entire layer could be totally pruned.

  To enable the NN to adjust to the new parameter configuration, the

pruned NN is fine-tuned for 15 epochs using the same hyperparameters as during training after applying any pruning technique [104]. Because pruning invariably eliminates some informative weights and fine-tuning allows the remaining parameters to make up for the lost functionality, this recovery phase is crucial. Throughout this process, the validation performance is tracked to guarantee convergence, and common metrics like ACPR, EVM and PAPR are used to assess the final pruned NNs.

2. Projection

   In contrast to pruning, the projection technique uses PCA to accomplish compression through dimensional reduction. This approach finds the most relevant directions in the parameter space and projects the weight matrices onto a lower-dimensional subspace that captures maximal variance, as opposed to eliminating specific parameters or neurons [79]. The process begins by extracting neuron activations from the training data: for each fully connected layer, the responses of neurons to a representative subset of input samples are collected. PCA is then performed on these activation patterns to determine the principal components that explain the majority of variance, expressed as the eigenvectors of the covariance matrix

$$C = \frac{1}{N} \sum_i (x_i - \mu)(x_i - \mu)^{\mathrm{T}},$$

   where x_i represents activation vectors and μ is their mean [80]. The weight matrix W of each layer is projected onto the top k principal components using

$$W_{\text{projected}} = W \times V_k,$$

   where V_k contains the k most significant eigenvectors sorted by their corresponding eigenvalues [81].

   As a result, the effective parameter count is reduced from $d^2 \to d \times k$, where k is usually used to attain a parameter reduction of roughly 50%. In contrast to pruning, the projection offers a continuous, smooth compression that can preserve information more gracefully [72]. It also successfully removes redundant dimensions that contribute little to the overall output variation. Following projection, the compressed NN is adjusted for 15 epochs using the same hyperparameters as during training in order to adjust to the transformed parameter space. The final outcomes are then saved for comparison with pruning methods.

3. Post-Projection Processing: Equalization and Pseudo-Quantization Noise (PQN)

   The compressed networks go through extra processing after the projection stage in order to further optimize them for hardware deployment. To guaranty consistency across layer dimensions, the `equalizeLayers` function is initially applied to the projected network. This fixes any dimensional discrepancies created during projection and gets the network ready for further quantization.

The network goes through a lightweight quantization-aware training process employing PQN insertion before quantization. In contrast to full QAT, which simulates accurate fixed-point operations by inserting fake quantization nodes across the network, PQN mimics the statistical characteristics of quantization errors by adding artificial noise to weights. The quantization step size for each weight matrix is computed using the dynamic range of the weights as $\Delta = 2^{\lfloor \log_2(\max |W|) \rfloor - 6}$, where the exponent denotes the allocation of 6 bits for the fractional part in a fixed-point representation. After that, uniform noise $n \sim \mathcal{U}(-\Delta/2, \Delta/2)$ is produced and added to the weights as $W_{\text{noisy}} = W + n$. This pseudo-noise simulates the quantization-induced rounding error, which for uniform quantization has a uniform distribution with variance $\Delta^2/12$ and zero mean.

While gradients are calculated with respect to the initial weights using the straight-through estimator approach, where the noise injection is viewed as having zero gradient contribution, training continues using the noisy weights throughout the forward pass. Without the computational expense of complete QAT, which necessitates altering the network topology with fake quantization nodes, this method enables the network to learn weight distributions that are naturally resistant to quantization errors. The Adam optimizer is used to train the PQN for five epochs at a learning rate of 1×10^{-6}.

The Deep Learning Quantizer (`dlquantizer`) from the Deep Learning Toolbox is used to quantize the network after PQN training. The quantizer is set up with the exponent scheme set to "Histogram" and the execution environment set to "FPGA" in order to find the best scaling factors depending on the value distribution. Prior to applying quantization, the network is calibrated using a representative calibration dataset. The quantizer then determines the proper scaling factors and zero points by analyzing the dynamic ranges of weights and activations across all layers. Following calibration, the final quantized network is created by applying the `quantize` function, which maps weights and activations from floating-point to fixed-point representation. To guaranty proper application to the intended layers, the quantization details are validated using `quantizationDetails`.

The complete process structural pruning with projection, magnitude pruning with projection, and SynFlow pruning with projection is applied separately to each of the pruned and projected networks. To measure the effect of compression and quantization on the network's performance, each quantized network's performance is assessed using ACPR, EVM, and PAPR measures.

4. Quantization

   As a compression technique that functions orthogonally to both pruning and projection, quantization lowers model complexity by reducing the numerical precision of weights and activations [82]. On hardware that supports lower-precision arithmetic, quantization allows for faster inference by reducing the amount of bits needed to hold each parameter rather than eliminating them. The technique determines the best quantization parameters by analyzing the weight distributions across all layers using the Deep Net Quantizer tool [15].

To determine scaling factors and zero-points that minimize quantization error, the range of weight values for each layer is analyzed.

The quantized representation for each weight is obtained through

$$q = \text{round}\left(\frac{W}{\text{scale}} + \text{zero\_point}\right),$$

where

$$\text{scale} = \frac{\max - \min}{2^b - 1}$$

for b-bit quantization, and

$$\text{zero\_point} = \text{round}\left(-\frac{\min}{\text{scale}}\right)$$

to handle asymmetric distributions [84]. This mapping projects the continuous floating-point values to a discrete fixed-point grid, with the number of levels determined by the bitwidth (typically 8 bits, providing 256 discrete levels). The quantized values are stored in integer format, achieving a 4× reduction in memory compared to FP32 representation (assuming 32-bit to 8-bit conversion). While quantization inevitably introduces approximation errors that may degrade prediction accuracy, it significantly improves inference speed and enables deployment on edge devices with limited memory and power budgets [96].

5

Results and Analysis

5.1 Legacy System Results

5.1.1 System Specifications

The CFR used for the legacy DFE is a PCCFR implemented as described in Section 4.1.2. Simulations have been carried out for PCCFRs with 1, 2 or 3 clip stages, denoted as CFR (1), CFR (2) and CFR (3) respectively. The HL used for the legacy DFE is implemented in accordance with Equation (4.6) and the DPD used for the legacy DFE is a dual polyphase LUT DPD implemented as described in Section 4.1.4. Simulations have been carried out using 4 LUTs, corresponding to memory depth $M = 3$, with LUT size 64 and LUT update step size $\mu = 1$. All evaluations were done after updating the LUTs using 5 signals, unless the LUT update was stopped prematurely due to ACPR degradation.

All components of the DFE were implemented with the same bitwidth using a signed fixed-point number representation with word length 24 and fraction length 20. In other words, 1 bit is dedicated to the sign of the value, 3 bits are dedicated to the integer part of the value and 20 bits are dedicated to the fractional part of the value. This bitwidth was selected because it was the lowest bitwidth for which all three component gave good performance.

The simulations have been carried out for the legacy DFE applied to the MP6 PA model for the cases with the DFE consisting only of DPD, and with the DFE consisting of CFR, HL and DPD, where the CFR has 1, 2 or 3 clip stages. Detailed results of these simulations in terms of performance and estimated hardware resource usage can be found in Section 5.1.2, and a summary and discussion of this can be found in Section 5.1.3.

5.1.2 Detailed Results for Legacy DFE

DFE consisting only of DPD

Detailed results for the system performance when applying the legacy DFE consisting only of DPD prior to the PA can be found in Table 5.1. The performance metrics ACPR, EVM and PAPR are reported at each stage of the system, i.e. the signal prior to modifications, after applying DPD and after applying PA.

Stage	ACPR (dBc)		EVM (dB)	PAPR (dB)
signal	−41.893	− 42.148	$-\infty$	11.863
DPD+PA	−27.919	− 26.833	−26.784	12.274

Table 5.1: System performance when only applying legacy DPD prior to the PA. Results are reported for the signal prior to modification and after PA.

A visualization of system performance can be found in figures 5.1a, 5.2a, 5.3a and 5.4a. The figures display the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the legacy DFE solution, when the DFE consists of only DPD.

Detailed results for the estimated system hardware resource usage of the legacy DFE when it consists of only DPD can be found in Table 5.2.

Component	Mult_{Eq}	Add_{Eq}	Reg_{Eq}	$\text{MAC}_{\text{strict}}$	MAC_{pool}	Mult_{Rem}	Add_{Rem}	Reg_{Rem}
DPD	737.53	1618.38	4540.75	737.53	2298.89	0	880.84	3803.22

Table 5.2: Estimation of legacy DFE hardware resource usage, for DFE consisting of only DPD.

DFE consisting of CFR with 1 clip stage, HL and DPD

Results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.3, where the legacy DFE consists of CFR with 1 clip stage, HL and DPD, for various PAPR thresholds. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

Threshold	ACPR (dBc)		EVM (dB)	PAPR (dB) (after HL)	PAPR (dB) (after DPD)
6.5 dB	−32.463	− 32.104	−23.885	6.6093	8.0185
7 dB	−31.676	− 31.161	−25.064	7.0679	8.4289
7.5 dB	−30.853	− 30.106	−25.921	7.5398	8.8480
8 dB	−29.921	− 28.995	−26.427	8.0217	9.4337
8.5 dB	−29.175	− 28.027	−25.905	8.5112	9.6724
9 dB	−28.907	− 28.130	−25.984	9.0049	9.2916

Table 5.3: System performance when the applying legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 1 clip stage for different PAPR thresholds. ACPR and EVM are evaluated after the PA and the PAPR is evaluated after the HL and after the DPD.

Detailed results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.4, where the legacy DFE consists of CFR with 1 clip stage, HL and DPD, and the PAPR threshold is 6.5 dB. The performance metrics ACPR, EVM and PAPR are reported at each stage of the system, i.e. the signal prior to modifications, after applying CFR, after applying HL, after applying DPD and after applying PA.

Stage	ACPR (dBc)		EVM (dB)	PAPR (dB)
signal	−41.893	− 41.148	$-\infty$	11.8630
CFR(1)	−41.637	− 41.891	−26.205	8.8492
CFR(1)+HL	−41.400	− 41.624	−26.140	6.6093
CFR(1)+HL+DPD+PA	−32.463	− 32.104	−23.885	7.6346

Table 5.4: System performance when applying the legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 1 clip stage and the PAPR threshold is 6.5 dB. Results are reported for the signal prior to modification, after CFR, after HL and after PA.

A visualization of system performance for PAPR threshold 6.5 dB can be found in figures 5.1b, 5.2b, 5.3b and 5.4b. The figures display the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the legacy DFE solution, when the DFE consists of CFR with 1 clip stage, HL and DPD.

Detailed results for the estimated system hardware resource usage of the legacy DFE when it consists of CFR with 1 clip stage, HL and DPD can be found in Table 5.5.

Component	$\mathbf{Mult_{Eq}}$	$\mathbf{Add_{Eq}}$	$\mathbf{Reg_{Eq}}$	$\mathbf{MAC_{strict}}$	$\mathbf{MAC_{pool}}$	$\mathbf{Mult_{Rem}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$
CFR (1)	276.750	4642.38	1057	276.75	1992.040	0	4365.630	780.25
HL	48.750	1364.38	0	0	471.040	48.75	1364.380	0
DPD	737.530	1618.38	4540.75	737.53	2298.890	0	880.840	3803.22
Combined	1063.03	7625.14	5597.75	1063.03	4761.97	0	6562.11	4534.72

Table 5.5: Estimation of legacy DFE hardware resource usage, for DFE consisting CFR with 1 clip stage, HL and DPD.

DFE consisting of CFR with 2 clip stages, HL and DPD

Results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.6, where the legacy DFE consists of CFR with 2 clip stages, HL and DPD, for various PAPR thresholds. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

Threshold	ACPR (dBc)		EVM (dB)	PAPR (dB) (after HL)	PAPR (dB) (after DPD)
6.5 dB	−32.615	− 32.171	−23.762	6.6134	8.0155
7 dB	−31.763	− 31.206	−24.985	7.0697	8.4226
7.5 dB	−30.912	− 30.156	−25.875	7.5406	8.7492
8 dB	−29.962	− 29.041	−26.400	8.0220	9.3262
8.5 dB	−29.157	− 28.002	−25.897	8.5112	9.6753
9 dB	−28.911	− 28.153	−25.981	9.0051	9.2787

Table 5.6: System performance when applying the legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 2 clip stages for different PAPR thresholds. ACPR and EVM are evaluated after the PA and the PAPR is evaluated after the HL and after the DPD.

Detailed results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.7, where the legacy DFE consists of CFR with 2 clip stages, HL and DPD, and the PAPR threshold is 6.5 dB. The performance metrics ACPR, EVM and PAPR are reported at each stage of the system, i.e. the signal prior to modifications, after applying CFR, after applying HL, after applying DPD and after applying PA.

Stage	**ACPR (dBc)**		**EVM (dB)**	**PAPR (dB)**
signal	−41.893	− 41.148	$-\infty$	11.8630
CFR(2)	−41.610	− 41.867	−25.947	7.7271
CFR(2)+HL	−41.605	− 41.861	−25.941	6.6134
CFR(2)+HL+DPD+PA	−32.615	− 31.171	−23.762	7.5764

Table 5.7: System performance when applying the legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 2 clip stages and the PAPR threshold is 6.5 dB. Results are reported for the signal prior to modification, after CFR, after HL and after PA.

A visualization of system performance PAPR threshold 6.5 dB can be found in subfigures 5.1c, 5.2c, 5.3c and 5.4c. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the legacy DFE solution, when the DFE consists of CFR with 2 clip stages, HL and DPD.

Detailed results for the estimated system hardware resource usage of the legacy DFE when it consists of CFR with 2 clip stages, HL and DPD can be found in Table 5.8.

Component	$\mathbf{Mult_{Eq}}$	$\mathbf{Add_{Eq}}$	$\mathbf{Reg_{Eq}}$	$\mathbf{MAC_{strict}}$	$\mathbf{MAC_{pool}}$	$\mathbf{Mult_{Rem}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$
CFR (2)	553.500	9284.750	2114	553.50	3984.080	0	8731.250	1560.50
HL	48.750	1364.380	0	0	471.040	48.75	1364.380	0
DPD	737.530	1618.380	4540.75	737.53	2298.890	0	880.840	3803.22
Combined	1339.781	12267.5	6654.75	1339.781	6754.010	0	10927.719	5314.969

Table 5.8: Estimation of legacy DFE hardware resource usage, for DFE consisting CFR with 2 clip stages, HL and DPD.

DFE consisting of CFR with 3 clip stages, HL and DPD

Results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.9, where the legacy DFE consists of CFR with 3 clip stages, HL and DPD, for various PAPR thresholds. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

Threshold	ACPR (dBc)		EVM (dB)	PAPR (dB) (after HL)	PAPR (dB) (after DPD)
6.5 dB	−32.617	− 32.169	−23.749	6.6138	8.0158
7 dB	−31.767	− 31.206	−24.978	7.0700	8.4217
7.5 dB	−30.916	− 30.158	−25.872	7.5407	8.7489
8 dB	−29.980	− 29.061	−26.400	8.0220	9.3220
8.5 dB	−29.152	− 27.997	−25.896	8.5112	9.6604
9 dB	−28.914	− 28.159	−25.981	9.0051	9.2755

Table 5.9: System performance when applying the legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 3 clip stages for different PAPR thresholds. ACPR and EVM are evaluated after the PA, and the PAPR is evaluated after the HL and after the DPD.

Detailed results for the system performance when applying the legacy DFE prior to the PA can be found in Table 5.10, where the legacy DFE consists of CFR with 3 clip stages, HL and DPD, and the PAPR threshold is 6.5 dB. The performance metrics ACPR, EVM and PAPR are reported at each stage of the system, i.e. the signal prior to modifications, after applying CFR, after applying HL, after applying DPD and after applying PA.

Stage	ACPR (dBc)		EVM (dB)	PAPR (dB)
signal	−41.893	− 41.148	$-\infty$	11.8630
CFR(3)	−41.606	− 41.862	−25.923	6.9844
CFR(3)+HL	−41.605	− 41.860	−25.922	6.6138
CFR(3)+HL+DPD+PA	−32.617	− 32.169	−23.749	7.5779

Table 5.10: System performance when applying the legacy DFE consisting of CFR, HL and DPD prior to the PA, where the CFR has 3 clip stages and the PAPR threshold is 6.5 dB. Results are reported for the signal prior to modification, after CFR, after HL and after PA.

A visualization of system performance PAPR threshold 6.5 dB can be found in subfigures 5.1d, 5.2d, 5.3d and 5.4d. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the legacy DFE solution, when the DFE consists of CFR with 3 clip stages, HL and DPD.

Detailed results for the estimated system hardware resource usage of the legacy DFE when it consists of CFR with 3 clip stages, HL and DPD can be found in Table 5.11.

Component	$\mathbf{Mult_{Eq}}$	$\mathbf{Add_{Eq}}$	$\mathbf{Reg_{Eq}}$	$\mathbf{MAC_{strict}}$	$\mathbf{MAC_{pool}}$	$\mathbf{Mult_{Rem}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$
CFR (3)	830.250	13927.13	3171	830.25	5976.130	0	13096.880	2340.75
HL	48.750	1364.38	0	0	471.040	48.75	1364.380	0
DPD	737.530	1618.38	4540.75	737.53	2298.890	0	880.840	3803.22
Combined	1616.53	16909.89	7711.75	1616.53	8746.06	0	15293.36	6095.22

Table 5.11: Estimation of legacy DFE hardware resource usage, for DFE consisting CFR with 3 clip stages, HL and DPD.

5.1.3 Summary and discussion of legacy system results

From Tables 5.3, 5.6 and 5.9 it can be seen that the performance of the legacy DFE consisting of CFR, HL and DPD for different PAPR thresholds is rather similar when the CFR has 1, 2 or 3 clip stages. The ACPR generally improves with lower PAPR thresholds and the EVM generally degrades for lower PAPR thresholds. The PAPR evaluated before DPD follows the PAPR threshold and degrades after the application of DPD. In general, the ACPR and PAPR performance marginally improves and EVM performance marginally degrades when the number of clip stages for the CFR increases. The PAPR reduction and EVM degradation when applying CFR and HL is within expectation. Though the ACPR was not expected to improve as much when lowering the PAPR threshold.

A summary of the performance from when applying the legacy DFE consisting of only DPD and when applying the legacy DFE consisting of CFR, HL and DPD where the CFR has 1, 2 or 3 clip stages, for PAPR threshold 6.5 dB can be found in Table 5.12. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

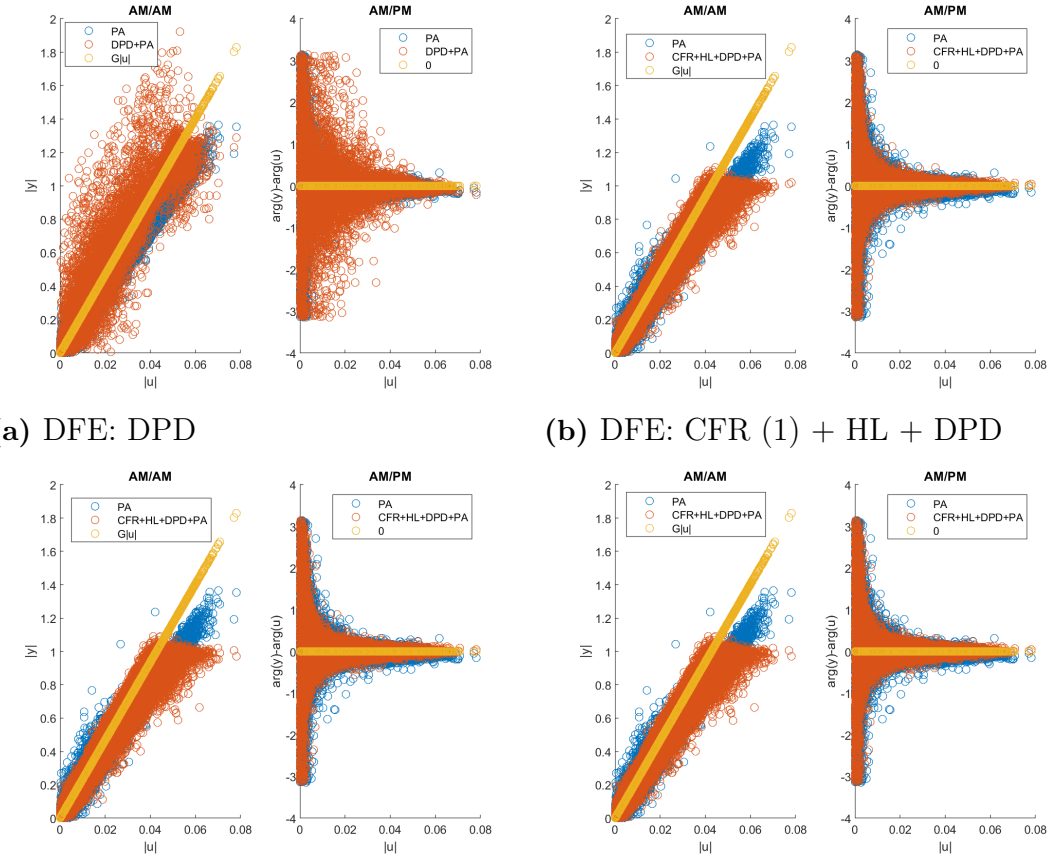
DFE Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
DPD	−27.919	− 26.833	−26.784	10.1980
CFR(1)+HL+DPD	−32.463	− 32.104	−23.885	8.0185
CFR(2)+HL+DPD	−32.615	− 32.171	−23.762	8.0155
CFR(3)+HL+DPD	−32.617	− 32.169	−23.749	8.0158
PA (without DFE)	−27.994	− 29.530	−15.830	11.8630

Table 5.12: Summary of system performance when only DPD is applied prior to PA, and when the DFE components CFR, HL and DPD are applied prior to PA, where the CFR has 1, 2 or 3 clip stages and the PAPR threshold is 6.5 dB. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

From the table, it can be seen that the ACPR and PAPR performance improves and the EVM performance degrades for the legacy DFE consisting of CFR, HL and DPD compared to for the DFE consisting only of DPD. When comparing the performance when applying the legacy DFE models to not applying a DFE model prior to the PA, it can be seen that the EVM performance when applying any of the legacy DFE models outperforms applying no DFE prior to the PA. It can also be seen that the ACPR performance when applying the legacy DFE consisting of CFR, HL and DPD is better than applying no DFE prior to the PA, while applying

the legacy DFE consisting only of DPD gives worse performance than applying no DFE prior to the PA in terms of ACPR. The degradation in ACPR performance when applying the DFE consisting only of DPD prior to the PA is unexpected, as applying a DFE prior to the PA is expected to improve all performance metrics. Finally it can be seen that applying the legacy DFE with CFR and HL lowers the PAPR compared to applying the legacy DFE without CFR and HL or applying no DFE prior to the PA.

A visualization of the results in Table 5.12 can be found in Figures 5.1, 5.2, 5.3 and 5.4. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the legacy DFE solutions.



(a) DFE: DPD

(b) DFE: CFR (1) + HL + DPD

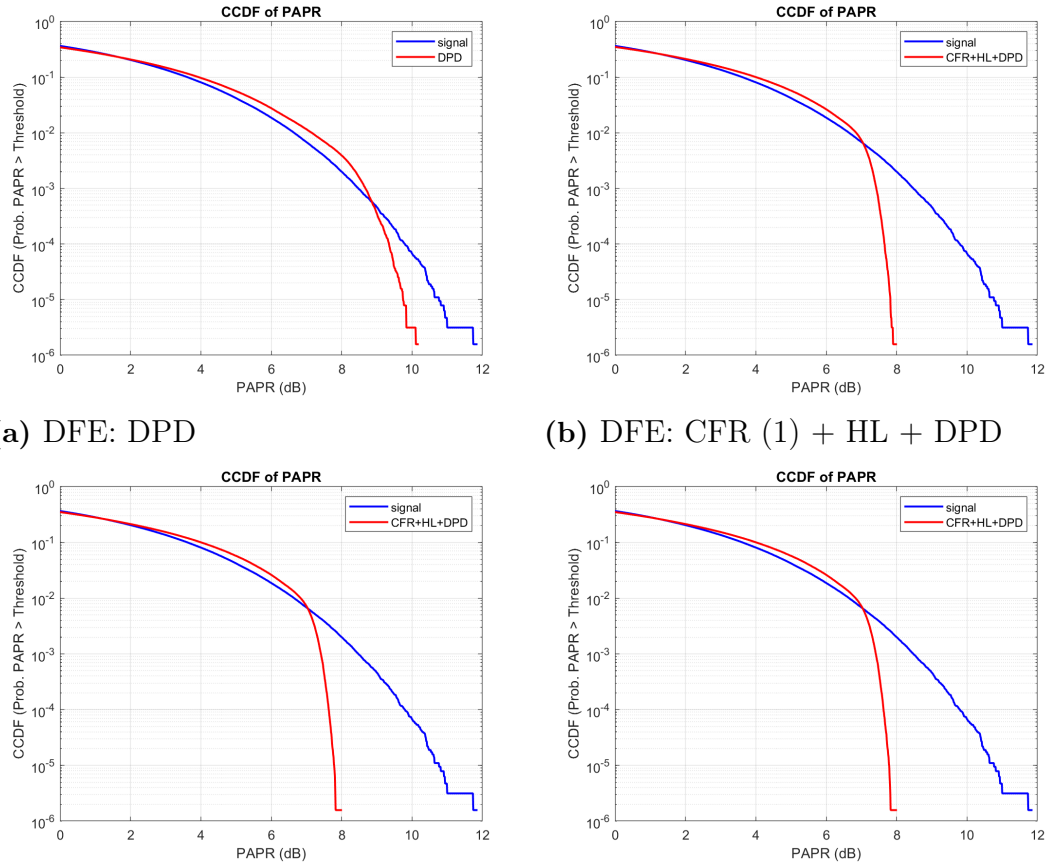
(c) DFE: CFR (2) + HL + DPD

(d) DFE: CFR (3) + HL + DPD

Figure 5.1: Visualization of the AM/AM and AM/PM distortions for the MP6 PA model, and the combined legacy DFE models and MP6 PA model applied to the evaluation signal. For DFEs with CFR and HL the PAPR threshold is 6.5 dB. Magnifications of the subfigures can be found in Appendix A, specifically in Figures A.1, A.2, A.3 and A.4.

When looking at the AM/AM and AM/PM distortions for the legacy DFE consisting only of DPD, it can be seen that the combined DPD and PA has rather sprawling amplitude and phase distortions that deviate a lot from the ideal linear gain amplitude and nonexistent phase distortions. Furthermore, it appears that applying the

legacy DPD prior to the PA gives more amplitude and phase distortions compared to applying no DPD prior to the PA. When instead looking at the AM/AM and AM/PM distortions for the combined legacy DFEs consisting of CFR, HL and DPD, and PA, it can be seen that the amplitude and phase distortions are very similar regardless of the number of clip stages for the CFR. It can also be seen that these distortions are much more concentrated around the ideal linear gain in amplitude and nonexistent phase distortions compared to those for the legacy DFE consisting only of DPD. Furthermore applying the legacy DFEs consisting of CFR, HL and DPD prior to the PA appears to give better result than applying no DFE prior to the PA. On the other hand, introducing CFR and HL to the legacy DFE constricts the amplitude of the system output.



(a) DFE: DPD

(b) DFE: CFR (1) + HL + DPD

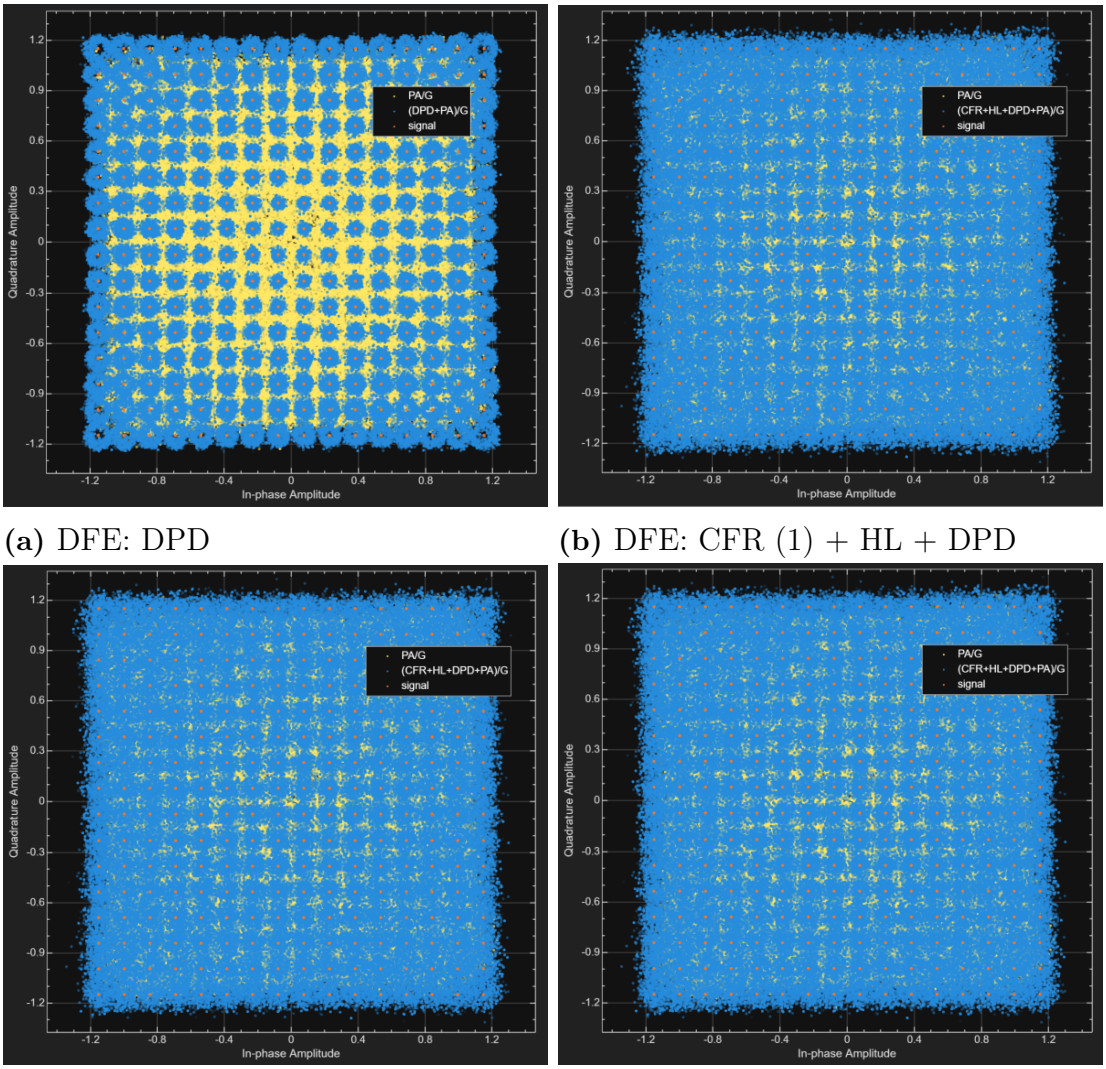
(c) DFE: CFR (2) + HL + DPD

(d) DFE: CFR (3) + HL + DPD

Figure 5.2: Visualization of the CCDF curves for the legacy DFE models applied to the evaluation signal. For DFEs with CFR and HL the PAPR threshold is 6.5 dB. Magnifications of the subfigures can be found in Appendix A, specifically in Figures A.5, A.6, A.7 and A.8.

When looking at the CCDF curve for the legacy DFE consisting only of DPD, it can be seen that the curve follows the CCDF curve of the evaluation signal rather well with only a slight decline in probability for PAPR thresholds over 9 dB and a slight reduction in PAPR. When instead looking at the CCDF curves for the legacy

DFEs consisting of CFR, HL and DPD, it can be seen that the curves are also rather similar regardless of the number of clip stages for the CFR. The CCDF curves show a clear PAPR reduction with a steep decline in probability for PAPR thresholds over 7.5 dB.



(a) DFE: DPD

(b) DFE: CFR (1) + HL + DPD

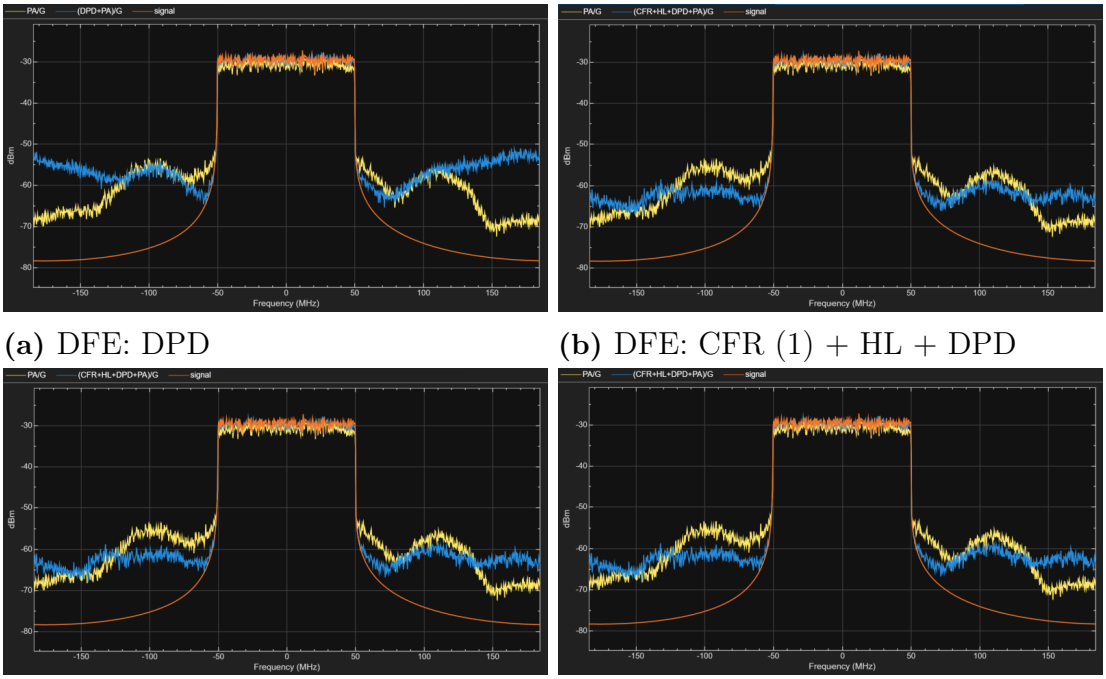
(c) DFE: CFR (2) + HL + DPD

(d) DFE: CFR (3) + HL + DPD

Figure 5.3: Visualization of the constellation diagrams for the MP6 PA model, and the combined legacy DFE models and MP6 PA model applied to the evaluation signal. For DFEs with CFR and HL the PAPR threshold is 6.5 dB. Magnifications of the subfigures can be found in Appendix A, specifically in Figures A.9, A.10, A.11 and A.12.

When looking at the constellation diagram for the legacy DFE consisting only of DPD applied prior to the PA, it can be seen that the clusters are centered and somewhat concentrated around the ideal undistorted constellation of the evaluation signal, making it mostly possible to identify the the symbols of the evaluation signal from the PA output. Compared to applying no DFE prior to the PA, it appears to be significantly easier to identify the signal symbols. When instead looking at the

constellation diagrams of the legacy DFE consisting of CFR, HL and DPD applied prior to the PA, it can be seen that these diagrams are also very similar regardless of the number of clip stages for the CFR. It can also be seen that the clusters are more centered around the signal symbols compared to applying no DFE prior to the PA, but also less concentrated compared to when applying the legacy DFE consisting only of DPD. Hence, it is easier to distinguish between the symbols when applying the DFE consisting of CFR, HL and DPD prior to the PA than when applying no DFE prior to the PA, but more difficult than when applying the DFE consisting only of DPD prior to the PA.



(a) DFE: DPD

(b) DFE: CFR (1) + HL + DPD

(c) DFE: CFR (2) + HL + DPD

(d) DFE: CFR (3) + HL + DPD

Figure 5.4: Visualization of the spectrums for the MP6 PA model, and the combined legacy DFE models and MP6 PA model applied to the evaluation signal. For DFEs with CFR and HL the PAPR threshold is 6.5 dB. Magnifications of the subfigures can be found in Appendix A, specifically in Figures A.13, A.14, A.15 and A.16.

When looking at the spectrum for the legacy DFE consisting only of DPD applied prior to the PA, it can be seen that there is more adjacent channel leakage for the PA output compared to when applying no DFE prior to the PA. When instead looking at the spectrums for the legacy DFEs consisting of CFR, HL and DPD applied prior to the PA, it can be seen that the spectrums are very similar, again regardless of the number of clip stages for the CFR. It can also be seen that there is less adjacent channel leakage compared to when applying no DFE prior to the PA, and in extention less adjacent channel leakage compared to when applying the legacy DFE consisting only of DPD prior to the PA.

A summary of the estimated hardware resource usage of the legacy DFE when it

consists only of DPD and when it consists of CFR, HL and DPD where the CFR has 1, 2 or 3 clip stages can be found in Table 5.13.

DFE Model	$\mathbf{Mult_{Eq}}$	$\mathbf{Add_{Eq}}$	$\mathbf{Reg_{Eq}}$	$\mathbf{MAC_{strict}}$	$\mathbf{MAC_{pool}}$	$\mathbf{Mult_{Rem}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$
DPD	737.530	1618.38	4540.75	737.530	2298.890	0	880.840	3803.220
CFR(1)+HL+DPD	1063.030	7625.14	5597.75	1063.030	4761.970	0	6562.110	4534.720
CFR(2)+HL+DPD	1339.781	12267.50	6654.75	1339.781	6754.010	0	10927.719	5314.969
CFR(3)+HL+DPD	1616.530	16909.89	7711.75	1616.530	8746.060	0	15293.360	6095.220

Table 5.13: Summary of system hardware

From the table, it can be seen that the hardware metrics increases when the DFE consists of more components and when the CFR consists of more clip stages, which is all within expectation. It can also be seen that the MultEq is lower than the AddEq and RegEq, making the MACStrict equal to the MultEq and the MultRem equal to zero.

5.2 Neural Network Solution Results

5.2.1 Network Specifications

Several NNs have been trained and compressed, both to model the DFE consisting of only DPD and to jointly co-optimize the components of the DFE consisting of CFR and DPD. The NN trainings have been carried out for the MP6 PA model, where the memory depths of these NNs match either the memory depth of the PA model, i.e. $M = 6$, or the memory depth of the legacy DPD model, i.e. $M = 3$.

The hyperparameters used to train the initial uncompressed NN solutions can be found in Table 5.14. In the table, M is the memory depth of the NN and Ns is the number of signal samples for a single signal.

Hyperparameter	Value
Input layer size	$3 \cdot (M + 1)$
Output layer size	2
Hidden layer sizes	[64, 32, 16, 3]
Number of epochs	30
Learning rate (μ)	0.01
Mini-batch size	12720 ($Ns/50$)
Number of signals	10
Loss function	joint loss
Weights DPD	[1, 0.4, 0, 0.02]
Weights DFE	[1, 0.4, 0.25, 0.02]

Table 5.14: Hyperparameters used for training the initial uncompressed NN solution.

The loss function "joint loss" refers to the weighted joint loss function defined in Equation (4.16). The PA model used during training for the linear (L_{Lin}) and out-

of-band (L_{OOB}) loss functions defined by Equations (4.17) and (4.18), is the same one the network has been trained for, i.e. the MP6 PA model. Weights for DPD and DFE refer to the weights w_{Lin}, w_{OOB}, w_{PAPR} and w_{Reg} of the joint loss function, used when training the NNs modeling the DFE consisting of only DPD and the DFE consisting of co-optimized CFR and DPD respectively.

The NNs are trained using FP32 number representation. After training the initial floating point NNs, the compression techniques listed in Table 5.15 are applied, after which the compressed NNs are fine-tuned with additional training to recover as much loss in performance as possible. For this additional training, the same learning rate, mini-batch size, number of signals, loss function and weights used to train the initial NNs are used, and the number of training epochs used for each compression technique are specified in Table 5.15.

Category	Technique	Additional Epochs
Pruning	Structural, Magnitude-based & SynFlow	15
Projection	PCA	15
Quantization	QAT	0

Table 5.15: Compression techniques investigated with number of additional training epochs to fine-tune and recover performance after compression.

The NNs are compressed by first applying one of the pruning techniques specified in the table followed by fine-tuning, after which projection is the pruned NN is applied to the pruned NN and the projected NN is then again fine-tuned. Finally, the projected NN is quantized using QAT after which no additional fine tuning is applied. The compressed NNs will be referred to as the name of the pruning technique followed by the last stage of the compression.

The memory depths of the trained NNs used to model the DFE are either $M = 3$ or $M = 6$. An estimation of the hardware resource usage for the construction and collection of the inputs and outputs of the NN for these memory depths can be found in Table 5.16.

Memory depth	Mult_{Eq}	Add_{Eq}	Reg_{Eq}	$\text{MAC}_{\text{strict}}$	MAC_{pool}	Mult_{Rem}	Add_{Rem}	Reg_{Rem}
M3	131.67	205.12	1780.5	131.67	705.77	0	73.453	1648.8
M6	131.67	370.12	3033.8	131.67	1178.50	0	238.450	2902.1

Table 5.16: Estimation of hardware resource usage for construction of NN inputs and collection of NN outputs.

5.2.2 Results NN-based DPD

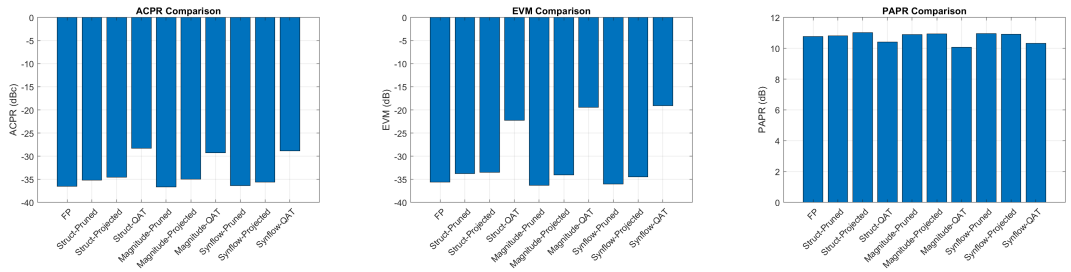
NN DPD M6

Detailed results for the system performance when applying the NN DFE, with memory depth $M = 6$ modeling only DPD, prior to the PA can be found in Table 5.17 and Figure 5.5. The performance metrics ACPR, EVM and PAPR are reported

for the uncompressed floating point NN as well as for each compression technique, where the ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

NN DPD M6 Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
Floating Point	−36.508	− 36.691	−35.584	10.769
Struct-Pruned	−35.151	− 35.824	−33.789	10.811
Struct-Projected	−34.566	− 35.345	−33.528	11.010
Struct-QAT	−28.327	− 25.794	−22.267	10.412
Magnitude-Pruned	−36.663	− 36.905	−36.257	10.885
Magnitude-Projected	−34.957	− 35.589	−34.044	10.945
Magnitude-QAT	−29.281	− 27.149	−19.445	10.062
SynFlow-Pruned	−36.380	− 36.791	−36.044	10.956
SynFlow-Projected	−35.602	− 35.958	−34.469	10.908
SynFlow-QAT	−28.875	− 27.643	−19.071	10.330

Table 5.17: System performance when only applying NN DPD prior to PA, where the NN DPD has memory depth $M = 6$. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.



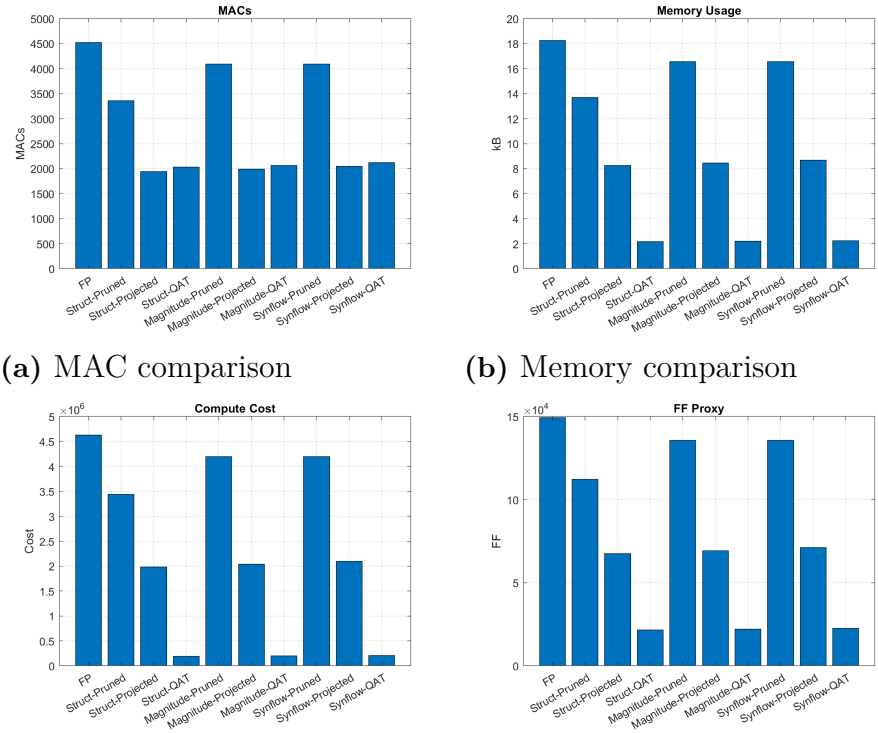
(a) ACPR comparison **(b)** EVM comparison **(c)** PAPR comparison

Figure 5.5: Visualization of the ACPR, EVM and PAPR performance of the compressed NN DPD models with memory depth $M = 6$. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.1, B.2, and B.3.

In Table 5.18 and Figure 5.6, detailed results for the estimated system hardware resource usage of the NN DFE, with memory depth $M = 6$ modeling only DPD, can be found. The hardware measurements are reported for the uncompressed floating point NN as well as for each compression technique.

NN DPD M6 Model	MAC	Params	TotalMemory	ComputeCost	$\mathbf{FF_{proxy}}$
Floating Point	4522	4670	18.2420e+3	4.6305e+6	1.4944e+5
Struct-Pruned	3359	3507	13.6990e+6	3.4396e+6	1.1222e+5
Struct-Projected	1939	2113	8.2539e+3	1.9855e+6	67616
Struct-QAT	2034	2208	2.1563e+3	1.9504e+5	21621
Magnitude-Pruned	4096	4244	16.5780e+3	4.1943e+6	1.3581e+5
Magnitude-Projected	1991	2165	8.4570e+3	2.0388e+6	69280
Magnitude-QAT	2065	2239	2.1865e+3	2.0076e+5	22077
SynFlow-Pruned	4096	4244	16.5780e+3	4.1943e+6	1.3581e+5
SynFlow-Projected	2048	2222	8.6797e+3	2.0972+6	71104
SynFlow-QAT	2119	2293	2.2393e+3	2.0439e+5	22520

Table 5.18: Estimation of NN DFE hardware resource usage, for NN DFE with memory depth $M = 6$ consisting of only DPD.



(a) MAC comparison **(b)** Memory comparison

(c) Compute cost comparison **(d)** FFp comparison

Figure 5.6: Visualization of the MAC, total memory, compute cost and FFp for the compressed NN DPD models with memory depth $M = 6$. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.4, B.5, B.6 and B.7.

As can be seen from the tables and figures, the NN compressed using magnitude-based pruning has the best overall performance in terms of ACPR and EVM. The NN with the lowest estimated hardware resource usage is the one compressed using the structured QAT. This NN is also one of the compressed NNs with the worst performance, along with the other NNs compressed with QAT. The NNs compressed

with projection have a much better performance significantly closer to that of the uncompressed floating point NN, while still having a significant hardware cost reduction, although not as good as the ones for the NNs compressed with QAT.

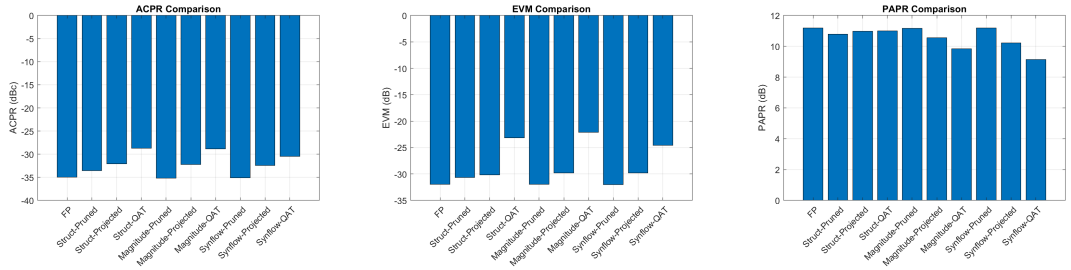
A visualization of the system performance for the NNs with the best performance and lowest estimated hardware resource usage respectively can be found in figures 5.13(a,b), 5.14(a,b), 5.15(a,b) and 5.16(a,b). The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the NN DFE solution with memory depth $M = 6$ modeling only DPD.

NN DPD M3

Detailed results for the system performance when applying the NN DFE, with memory depth $M = 3$ modeling only DPD, prior to the PA can be found in Table 5.19 and Figure 5.7. The performance metrics ACPR, EVM and PAPR are reported for the uncompressed floating point NN as well as for each compression technique, where the ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

NN DPD M3 Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
Floating Point	−34.975	− 35.860	−32.939	11.1820
Struct-Pruned	−33.560	− 34.588	−30.656	10.7910
Struct-Projected	−32.125	− 33.482	−30.182	10.9760
Struct-QAT	−28.720	− 27.836	−23.169	10.9910
Magnitude-Pruned	−35.181	− 36.235	−31.959	11.1620
Magnitude-Projected	−32.258	− 33.207	−29.785	10.5570
Magnitude-QAT	−28.866	− 27.019	−22.118	9.8396
SynFlow-Pruned	−35.145	− 36.246	−32.042	11.1800
SynFlow-Projected	−32.434	− 33.345	−29.772	10.2270
SynFlow-QAT	−30.443	− 28.995	−24.548	9.1365

Table 5.19: System performance when only applying NN DPD prior to PA, where the NN DPD has memory depth $M = 3$. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.



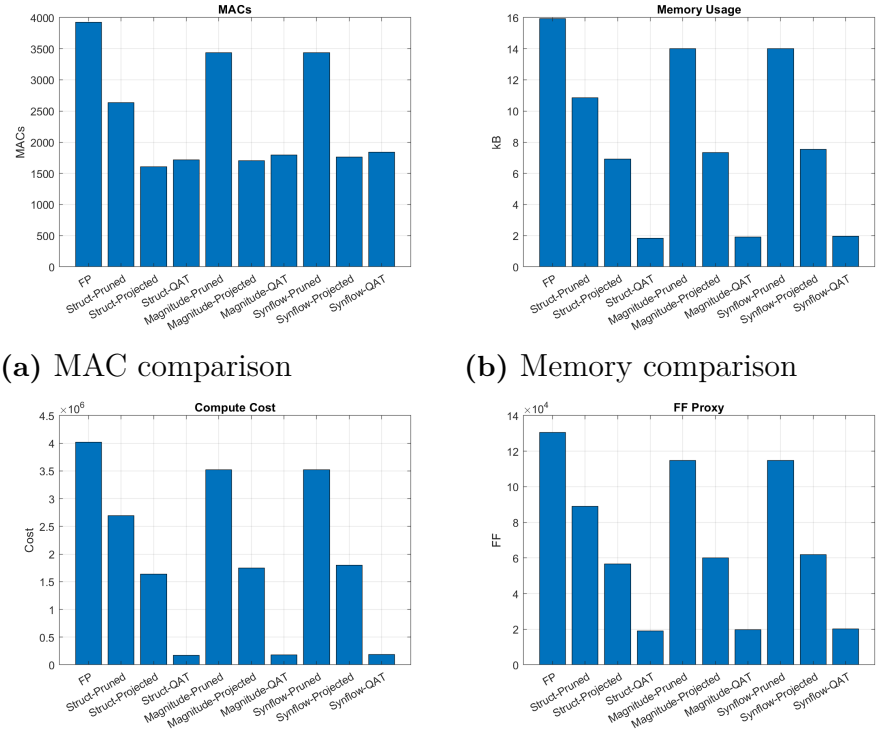
(a) ACPR comparison **(b)** EVM comparison **(c)** PAPR comparison

Figure 5.7: Visualization of the ACPR, EVM and PAPR performance of the compressed NN DPD models with memory depth $M = 3$. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.8, B.9, and B.10.

In Table 5.20 and Figure 5.8, detailed results for the estimated system hardware resource usage of the NN DFE, with memory depth $M = 3$ modeling only DPD, can be found. The hardware measurements are reported for the uncompressed floating point NN as well as for each compression technique.

NN DPD M3 Model	MAC	Params	TotalMemory	ComputeCost	$\mathbf{FF_{proxy}}$
Floating Point	3928	4076	15.9220e+3	4.0223e+6	1.3043e+5
Struct-Pruned	2633	2781	10.8630e+3	2.6962e+6	88992
Struct-Projected	1603	1774	6.9297e+3	1.6415e+6	56768
Struct-QAT	1715	1886	1.8418e+3	1.7361e+5	18976
Magnitude-Pruned	3438	3586	14.0080e+3	3.5205e+6	1.1475e+5
Magnitude-Projected	1706	1877	7.3320e+3	1.7469e+6	60064
Magnitude-QAT	1792	1963	1.9170e+3	1.82446e+5	20192
SynFlow-Pruned	3438	3586	14.0080e+3	3.5205e+6	1.1475e+5
SynFlow-Projected	1760	1931	7.5320e+3	1.8022e+6	61792
SynFlow-QAT	1840	2011	1.9639e+3	1.855e+5	20192

Table 5.20: Estimation of NN DFE hardware resource usage, for NN DFE with memory depth $M = 3$ consisting of only DPD.



(a) MAC comparison **(b)** Memory comparison

(c) Compute cost comparison **(d)** FFp comparison

Figure 5.8: Visualization of the MAC, total memory, compute cost and FFp for the compressed NN DPD models with memory depth $M = 3$. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.11, B.12, B.13 and B.14.

As can be seen from the tables and figures, the NN compressed using SynFlow

pruning has the best overall performance in terms of ACPR and EVM, and is closely followed by the NN compressed using magnitude-based pruning. The NN with the lowest estimated hardware resource usage is the one compressed using the structured QAT. This NN is also one of the compressed NNs with the worst performance, along with the other NNs compressed with QAT. The NNs compressed with projection have a much better performance significantly closer to that of the uncompressed floating point NN, while still having a significant hardware cost reduction, although not as good as the ones for the NNs compressed with QAT.

A visualization of the system performance for the NNs with the best performance and lowest estimated hardware resource usage respectively can be found in subfigures 5.13(c,d), 5.14(c,d), 5.15(c,d) and 5.16(c,d). The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the NN DFE solution with memory depth $M = 3$ modeling only DPD.

5.2.3 Results NN CFR and DPD

NN DFE M6

Results for the system performance when applying the uncompressed floating point NN DFE, with memory depth $M = 6$ jointly modeling CFR and DPD, prior to the PA can be found in Table 5.21, for various PAPR thresholds. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

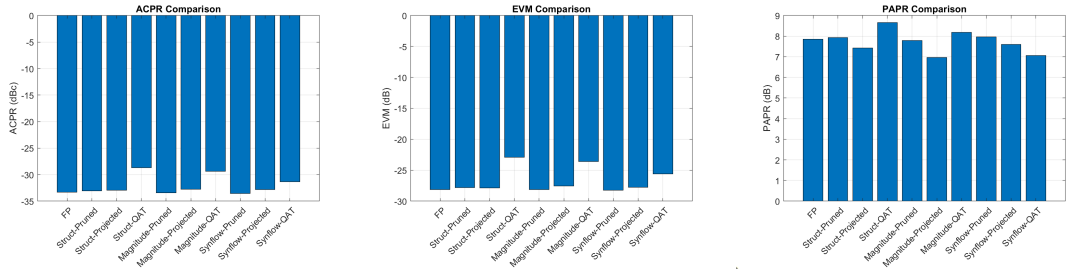
Threshold	ACPR (dBc)		EVM (dB)	PAPR (dB)
6.5 dB	−33.307	− 34.007	−28.149	7.8456
7 dB	−34.700	− 35.347	−29.768	7.7677
7.5 dB	−35.425	− 35.800	−31.918	8.0599
8 dB	−35.594	− 35.894	−32.493	8.7178
8.5 dB	−35.748	− 35.992	−33.980	9.4872
9 dB	−36.052	− 36.295	−33.975	10.4060

Table 5.21: System performance when applying the uncompressed NN DFE, with memory depth $M = 6$ jointly modeling CFR and DPD, prior to the PA, for different PAPR thresholds. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

Detailed results for the system performance when applying the NN DFE, with memory depth $M = 6$ jointly modeling CFR and DPD, prior to the PA can be found in Table 5.22 and Figure 5.9 for PAPR threshold 6.5 dB. The performance metrics ACPR, EVM and PAPR are reported for the uncompressed floating point NN as well as for each compression technique, where the ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

NN DFE M6 Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
Floating Point	−33.307	− 34.007	−28.149	7.8456
Struct-Pruned	−33.053	− 33.353	−27.818	7.9277
Struct-Projected	−32.942	− 33.408	−27.887	7.4237
Struct-QAT	−28.688	− 27.332	−22.893	8.6560
Magnitude-Pruned	−33.420	− 34.159	−28.129	7.7828
Magnitude-Projected	−32.727	− 33.175	−27.524	6.9709
Magnitude-QAT	−29.367	− 28.166	−23.608	8.1883
SynFlow-Pruned	−33.574	− 34.369	−28.208	7.9611
SynFlow-Projected	−32.826	− 33.672	−27.754	7.5950
SynFlow-QAT	−31.337	− 30.879	−25.594	7.0643

Table 5.22: System performance when only applying NN DFE jointly modeling CFR and DPD prior to PA, where the NN has memory depth $M = 6$ for PAPR threshold 6.5 dB. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.



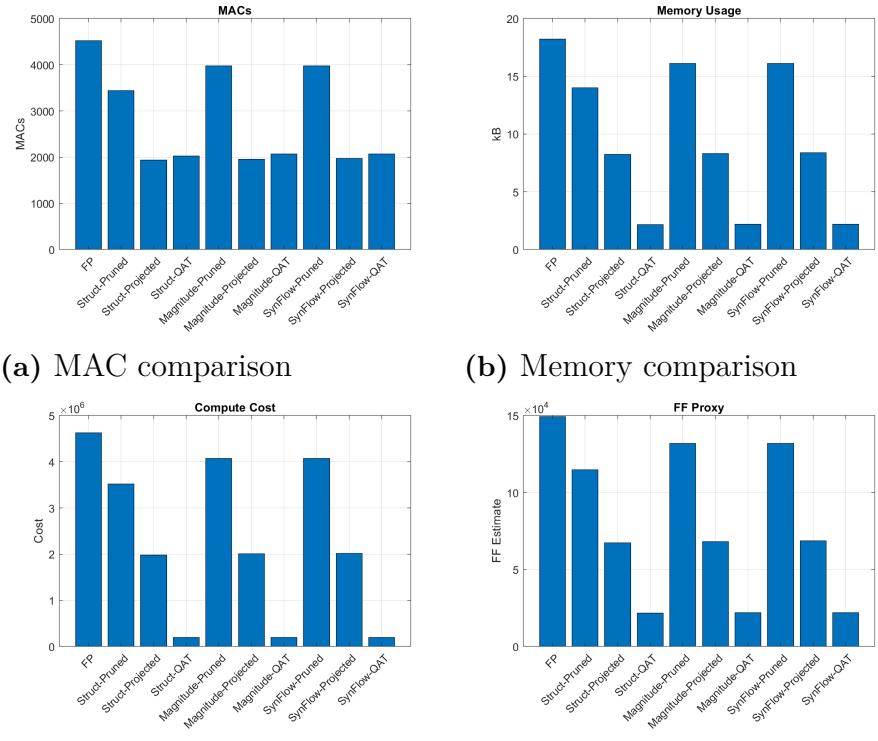
(a) ACPR comparison **(b)** EVM comparison **(c)** PAPR comparison

Figure 5.9: Visualization of the ACPR, EVM and PAPR performance of the compressed NN DFE modeling co-optimized CFR and DPD with memory depth $M = 6$ for PAPR threshold 6.5 dB. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.1, C.2, and C.3.

In Table 5.23 and Figure 5.10, detailed results for the estimated system hardware resource usage of the NN DFE, with memory depth $M = 6$ jointly modeling CFR and DPD for PAPR threshold 6.5 dB, can be found. The hardware measurements are reported for the uncompressed floating point NN as well as for each compression technique.

NN DFE M6 Model	MAC	Params	TotalMemory	ComputeCost	$\mathbf{FF_{proxy}}$
Floating Point	4522	4670	18.2420e+3	4.6305e+6	1.4944e+5
Struct-Pruned	3439	3587	14.0120e+3	3.5215e+6	1.1478e+5
Struct-Projected	1934	2108	8.2344e+3	1.9804e+6	67456
Struct-QAT	2029	2203	2.1514e+3	1.9606e+5	22116
Magnitude-Pruned	3977	4125	16.1130e+3	4.0724e+6	1.32e+5
Magnitude-Projected	1958	2132	8.3281e+3	2.0050e+6	68224
Magnitude-QAT	2069	2243	2.1924e+3	2.0115e+5	22136
SynFlow-Pruned	3976	4124	16.1090e+3	4.0714e+6	1.3197e+5
SynFlow-Projected	1973	2147	8.3867e+3	2.0204e+6	68224
SynFlow-QAT	2071	2245	2.1924e+3	2.0135e+5	56322

Table 5.23: Estimation of NN DFE hardware resource usage, for NN DFE with memory depth $M = 6$ jointly modeling CFR and DPD with PAPR threshold 6.5 dB.



(a) MAC comparison **(b)** Memory comparison

(c) Compute cost comparison **(d)** FFp comparison

Figure 5.10: Visualization of the MAC, total memory, compute cost and FFp for the compressed NN DFE modeling co-optimized CFR and DPD with memory depth $M = 6$ for PAPR threshold 6.5 dB. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.4, C.5, C.6 and C.7.

As can be seen from the tables and figures, the compressed NN with the overall best performance in terms of ACPR, EVM and PAPR is the one compressed using the structured projection. The NN with the lowest estimated hardware resource usage is the one compressed using the structured QAT. This NN is also one of the

compressed NNs with the worst performance, along with the other NNs compressed with QAT. The NNs compressed with projection have a much better performance significantly closer to that of the uncompressed floating point NN, while still having a significant hardware cost reduction, although not as good as the ones for the NNs compressed with QAT.

A visualization of the system performance for the compressed NNs with the best performance and lowest estimated hardware resource usage respectively for PAPR threshold 6.5 dB can be found in subfigures 5.17(a,b), 5.18(a,b), 5.19(a,b) and 5.20(a,b). The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the NN DFE solution with memory depth $M = 6$ jointly modeling CFR and DPD.

NN DFE M3

Results for the system performance when applying the uncompressed floating point NN DFE, with memory depth $M = 3$ jointly modeling CFR and DPD, prior to the PA can be found in Table 5.24, for various PAPR thresholds. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

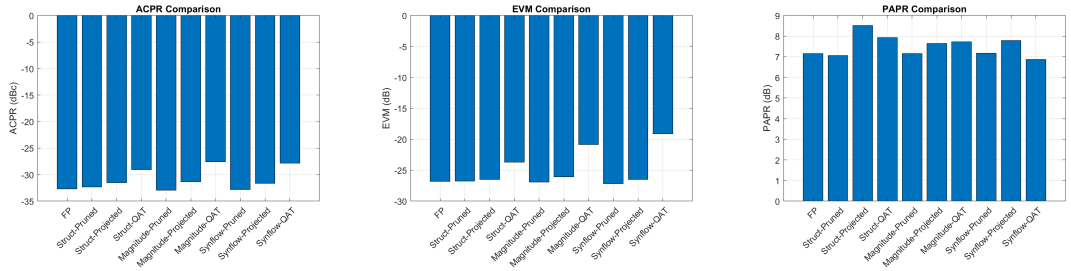
Threshold (dB)	**ACPR (dBc)**		**EVM (dB)**	**PAPR (dB)**
6.5 dB	−32.720	− 33.410	−26.816	7.1556
7 dB	−33.429	− 34.215	−28.371	7.6169
7.5 dB	−33.861	− 34.359	−29.329	8.7765
8 dB	−34.045	− 34.368	−30.167	9.4550
8.5 dB	−34.147	− 34.717	−30.027	9.0663
9 dB	−34.398	− 34.992	−30.723	9.9657

Table 5.24: System performance when applying the uncompressed NN DFE, with memory depth $M = 3$ jointly modeling CFR and DPD, prior to the PA, for different PAPR thresholds. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

Detailed results for the system performance when applying the NN DFE, with memory depth $M = 3$ jointly modeling CFR and DPD, prior to the PA can be found in Table 5.25 and Figure 5.11 for PAPR threshold 6.5 dB. The performance metrics ACPR, EVM and PAPR are reported for the uncompressed floating point NN as well as for each compression technique, where the ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

NN DFE M3 Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
Floating Point	−32.720	− 33.410	−26.816	7.1556
Struct-Pruned	−32.303	− 33.126	−26.753	7.0661
Struct-Projected	−31.549	− 31.850	−26.492	8.5136
Struct-QAT	−29.043	− 28.162	−23.700	7.9305
Magnitude-Pruned	−32.961	− 33.897	−26.932	7.1522
Magnitude-Projected	−31.365	− 31.304	−26.093	7.6476
Magnitude-QAT	−27.602	− 25.990	−20.859	7.7311
SynFlow-Pruned	−32.846	− 33.972	−27.206	7.1796
SynFlow-Projected	−31.635	− 32.299	−26.516	7.7840
SynFlow-QAT	−27.811	− 25.721	−19.088	6.8662

Table 5.25: System performance when only applying NN DFE jointly modeling CFR and DPD prior to PA, where the NN has memory depth $M = 3$ for PAPR threshold 6.5 dB. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.



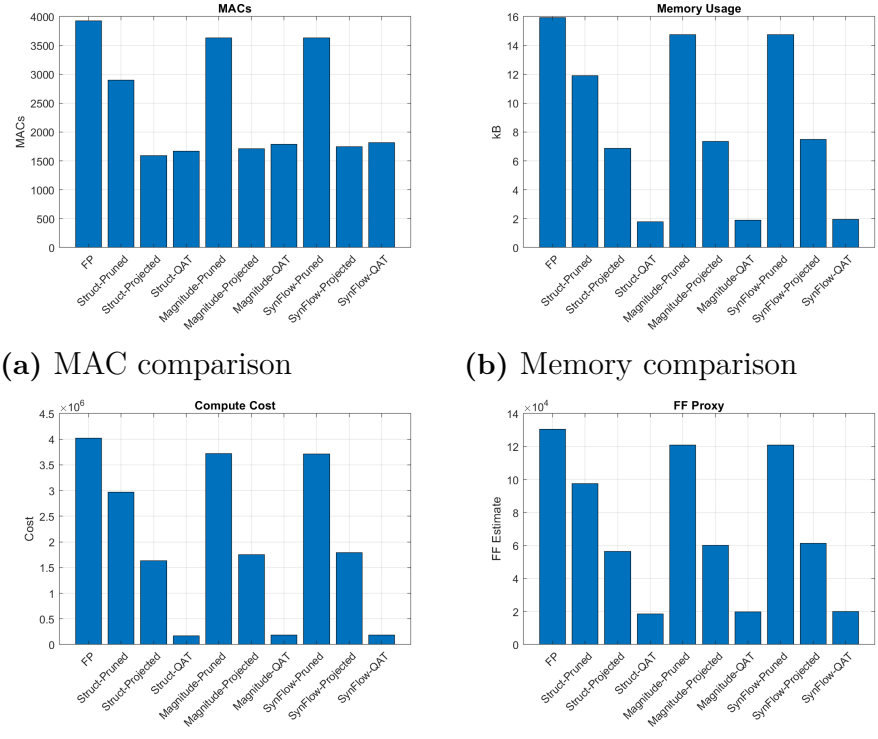
(a) ACPR comparison **(b)** EVM comparison **(c)** PAPR comparison

Figure 5.11: Visualization of the ACPR, EVM and PAPR performance of the compressed NN DFE modeling co-optimized CFR and DPD with memory depth $M = 3$ for PAPR threshold 6.5 dB. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.8, C.9, and C.10.

In Table 5.26 and Figure 5.12, detailed results for the estimated system hardware resource usage of the NN DFE for PAPR threshold 6.5 dB, with memory depth $M = 3$ modeling only DPD, can be found. The hardware measurements are reported for the uncompressed floating point NN as well as for each compression technique.

NN DFE M3 Model	MAC	Params	TotalMemory	ComputeCost	$\mathbf{FF_{proxy}}$
Floating Point	3928	4076	15.9220e+3	4.0223e+6	1.3043e+5
Struct-Pruned	2901	3049	11.9100e+3	2.9706e+6	97568
Struct-Projected	1592	1763	6.8867e+3	1.6302e+6	56416
Struct-QAT	1669	1840	1.7969e+3	1.7067e+5	18606
Magnitude-Pruned	3630	3778	14.7580e+3	3.7171e+6	1.209e+5
Magnitude-Projected	1709	1880	7.3438e+3	1.7500e+6	60160
Magnitude-QAT	1787	1958	1.9121e+3	1.8196e+5	19758
SynFlow-Pruned	3629	3777	14.7540e+3	3.7161e+6	1.2086e+5
SynFlow-Projected	1747	1918	7.4922e+3	1.7889e+6	61376
SynFlow-QAT	1820	1991	1.9443e+3	1.8424e+5	20032

Table 5.26: Estimation of NN DFE hardware resource usage, for NN DFE with memory depth $M = 3$ jointly modeling CFR and DPD with PAPR threshold 6.5 dB.



(a) MAC comparison **(b)** Memory comparison

(c) Compute cost comparison **(d)** FFp comparison

Figure 5.12: Visualization of the MAC, total memory, compute cost and FFp for the compressed NN DFE modeling co-optimized CFR and DPD with memory depth $M = 3$ for PAPR threshold 6.5 dB. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.11, C.12, C.13 and C.14.

As can be seen from the tables and figures, the compressed NN with the overall best performance in terms of ACPR, EVM and PAPR is the one compressed using structured pruning. The NN with the lowest estimated hardware resource usage is the one compressed using the structured QAT. This NN is also one of the compressed

NNs with the worst performance, along with the other NNs compressed with QAT. The NNs compressed with projection have a much better performance significantly closer to that of the uncompressed floating point NN, while still having a significant hardware cost reduction, although not as good as the ones for the NNs compressed with QAT.

A visualization of the system performance for the compressed NNs with the best performance and lowest estimated hardware resource usage respectively for PAPR threshold 6.5 dB can be found in subfigures 5.17(c,d), 5.18(c,d), 5.19(c,d) and 5.20(c,d. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums for the NN DFE solution with memory depth $M = 3$ jointly modeling CFR and DPD.

5.3 Results Summary

This section summarizes a comprehensive comparison and summary of the main performance and the estimated hardware-resource results of NN-based solutions and their respective legacy implementations. The results are presented separately for the DPD only solutions in Section 5.3.1 and the complete DFE solutions in Section 5.3.2, focusing on the models providing the best performance and the lowest estimated hardware usage.

5.3.1 Summary for Digital Pre-Distortion Solutions

A summary of the system performance when applying the NN DFE consisting only of DPD can be found in Table 5.27, where the performance of the compressed NNs with memory depth $M = 6$ and $M = 3$ with the best performance and lowest estimated hardware resource usage as well as their uncompressed floating point counterparts is reported. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

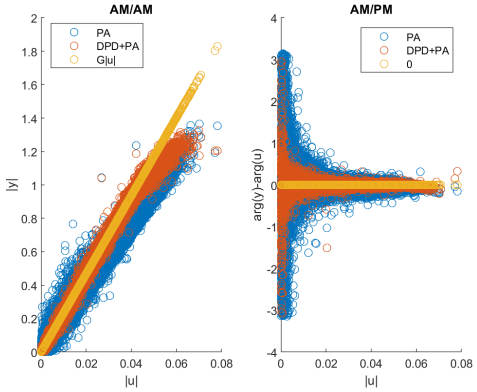
DPD Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
NN DPD M6 FP	−36.508	− 36.691	−35.584	10.769
NN DPD M3 FP	−34.975	− 35.860	−32.939	11.182
NN DPD M6 Magnitude-Pruned	−36.663	− 36.905	−36.257	10.885
NN DPD M6 Struct-QAT	−28.327	− 25.794	−22.267	10.412
NN DPD M3 SynFlow-Pruned	−35.145	− 36.246	−32.042	11.180
NN DPD M3 Struct-QAT	−28.720	− 27.836	−23.169	10.991
Legacy DPD	−27.919	− 26.833	−26.784	10.198
PA (without DPD)	−27.994	− 29.530	−15.830	11.863

Table 5.27: Summary of system performance for best performing compressed NNs when only applying NN DPD prior to PA. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

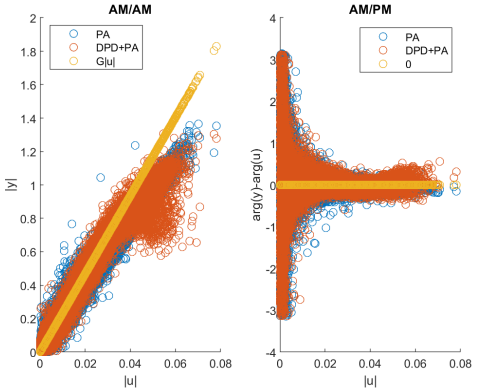
From the table it can be seen that the best performing compressed NN models, i.e. the NN compressed using magnitude-based pruning for memory depth $M = 6$ and

the NN compressed using SynFlow pruning for memory depth $M = 3$, outperforms the legacy DPD model for all performance metrics, both for memory depth $M = 6$ and memory depth $M = 3$. Furthermore, it can be seen that the best performing compressed NN for memory depth $M = 6$ has better performance than the best performing compressed NN for memory depth $M = 3$ as well as better performance than for its uncompressed floating point counterpart. The best performing compressed NN for memory depth $M = 3$ has better performance in terms of ACPR and slightly worse performance in terms of EVM compared to for its uncompressed floating point counterpart. It can also be seen that the NN DPDs with the lowest estimated hardware usage, i.e. the NNs compressed using structured QAT for both memory depth $M = 6$ and $M = 3$, doesn't perform as well as the legacy DPD in terms of EVM. Furthermore, it can be seen that the NN DPD with memory $M = 3$ compressed using structured QAT performs better than the legacy DPD, while the NN DPD with memory $M = 3$ compressed using structured QAT performs worse than the legacy DPD in terms of ACPR. Finally, it can be seen that all NN DPD solutions in the table gives better performance than applying no DPD model in terms of EVM and PAPR, and that the uncompressed floating point NN DPD solutions gives better performance than applying no DPD model in terms of ACPR, whereas the NN DPD solutions compressed using structured QAT gives worse performance than applying no DPD model in terms of ACPR. The performance of the uncompressed floating point NN models compared to that of the legacy DPD model was expected, however the application of the compression techniques had an unexpected amount of performance degradation. Other works such as [57] managed to reduce the NN complexity with lower performance loss.

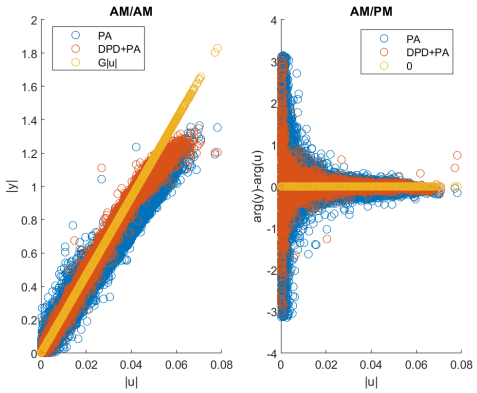
A visualization of the results of the compressed NN DFE models in Table 5.27 can be found in Figures 5.13, 5.14, 5.15 and 5.16. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums of the compressed NNs in the table.



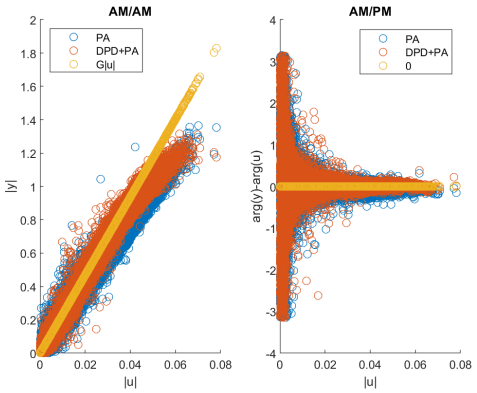
(a) NN DPD M6 best performance



(b) NN DPD M6 lowest hardware usage



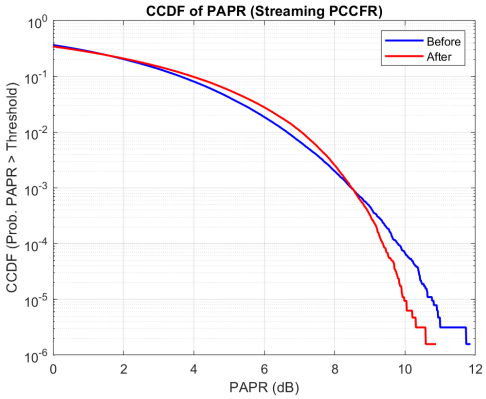
(c) NN DPD M3 best performance



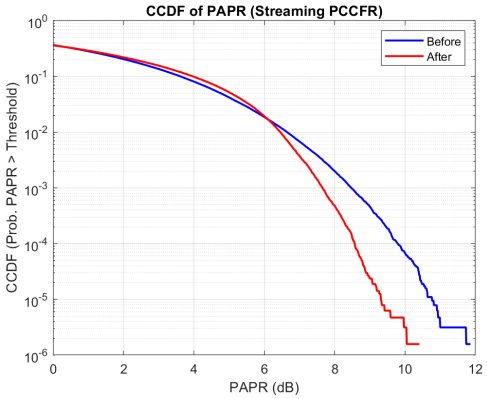
(d) NN DPD M3 lowest hardware usage

Figure 5.13: Visualization of the AM/AM and AM/PM distortions for the MP6 PA model, and the combined NN DPD models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.15, B.16, B.17 and B.18.

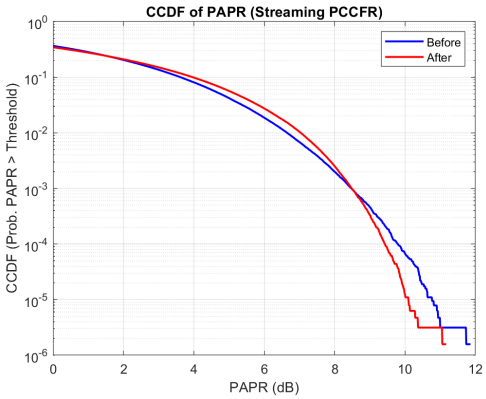
When looking at the AM/AM and AM/PM distortions for the compressed NN DPD models with the best performance for memory depths $M = 6$ and $M = 3$, it can be seen that the combined NN DPD models and PA results in rather good linearization performance. The output amplitudes are better centered around the ideal linear gain and the phase shifts are better centered around the ideal zero phase shift compared to when applying no DPD prior to the PA for both memory depths. The linearization performance for the NN DPD with memory depth $M = 6$ appears to be slightly better than that for the NN DPD with memory depth $M = 3$, though both NN DPD models have better PA linearization performance than the legacy DPD. When instead looking at the AM/AM and AM/PM distortions for the NN DPD models with the lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$, it can be seen that the PA linearization performance is not as good as that for the best performing NN DPD models, especially for the one with memory depth $M = 6$. It does however appear that the amount of amplitude and phase distortions for the NN DPD models with the lowest estimated hardware resource usage for both memory depths is smaller than for the legacy DPD model.



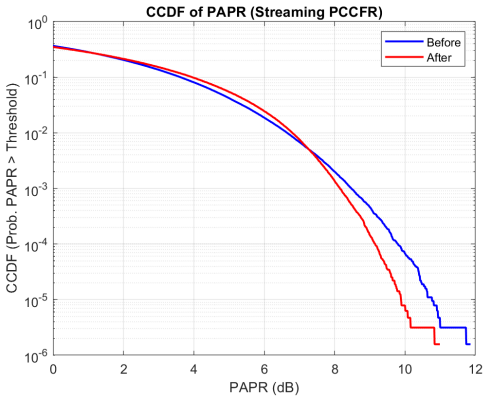
(a) NN DPD M6 best performance



(b) NN DPD M6 lowest hardware usage



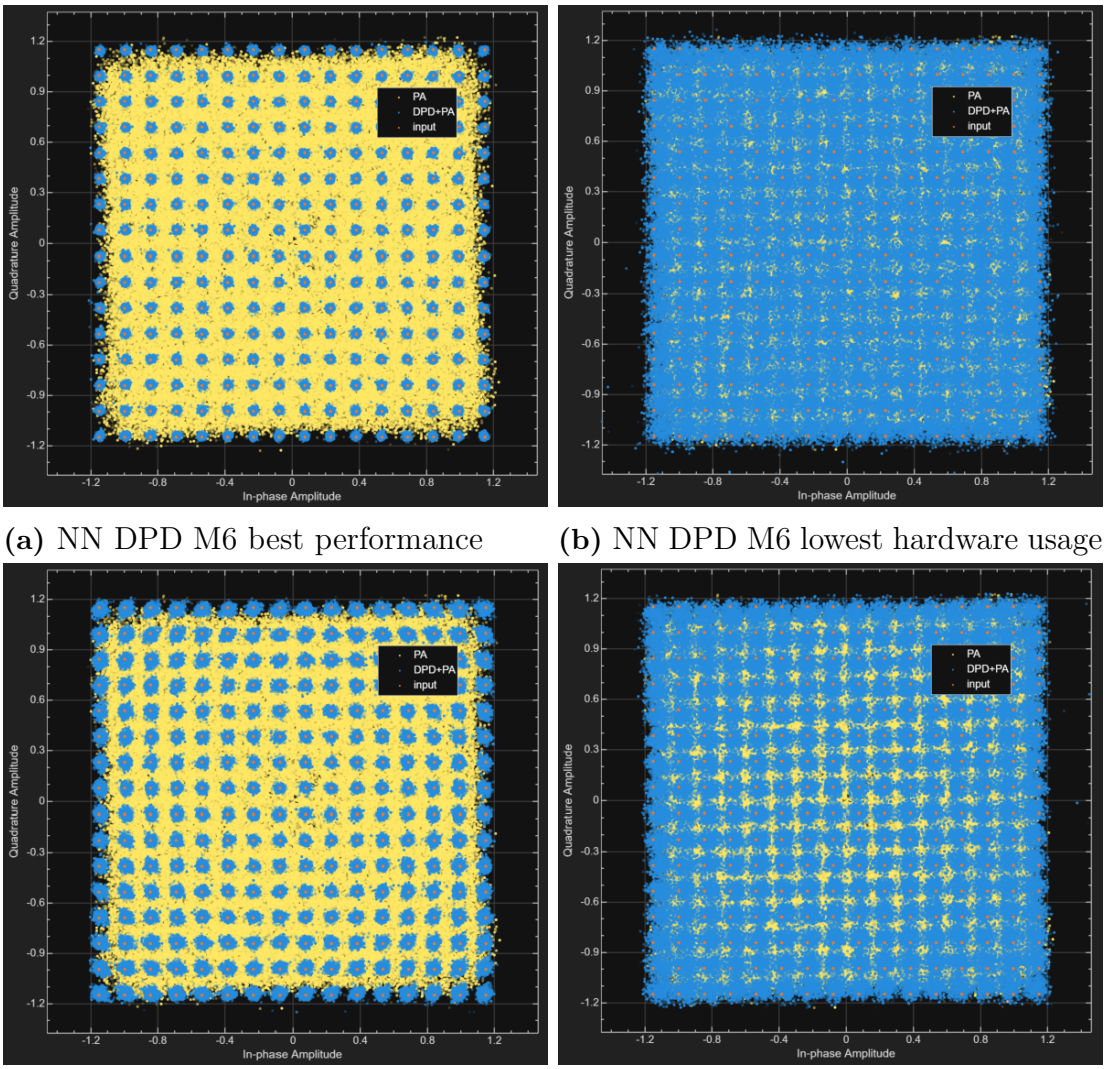
(c) NN DPD M3 best performance



(d) NN DPD M3 lowest hardware usage

Figure 5.14: Visualization of the CCDF curves for the NN DPD model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.19, B.20, B.21 and B.22.

When looking at the CCDF curves for the compressed NN DPD models with the best performance and the NN DPD models with the lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$, it can be seen that the curves of all models follows the CCDF curve of the evaluation signal rather well with only slight PAPR reductions and gentle declines in probability. For the best performing NN DPD models this decline starts for PAPR thresholds over 8.5 dB, and for the NN DPD models with the lowest estimated hardware resource usage this decline starts for PAPR thresholds over 6 and 7.5 dB for memory depths $M = 6$ and $M = 3$ respectively. The behavior of the CCDF curves for all NN DPD models is similar to that of the legacy DPD model.



(a) NN DPD M6 best performance

(b) NN DPD M6 lowest hardware usage

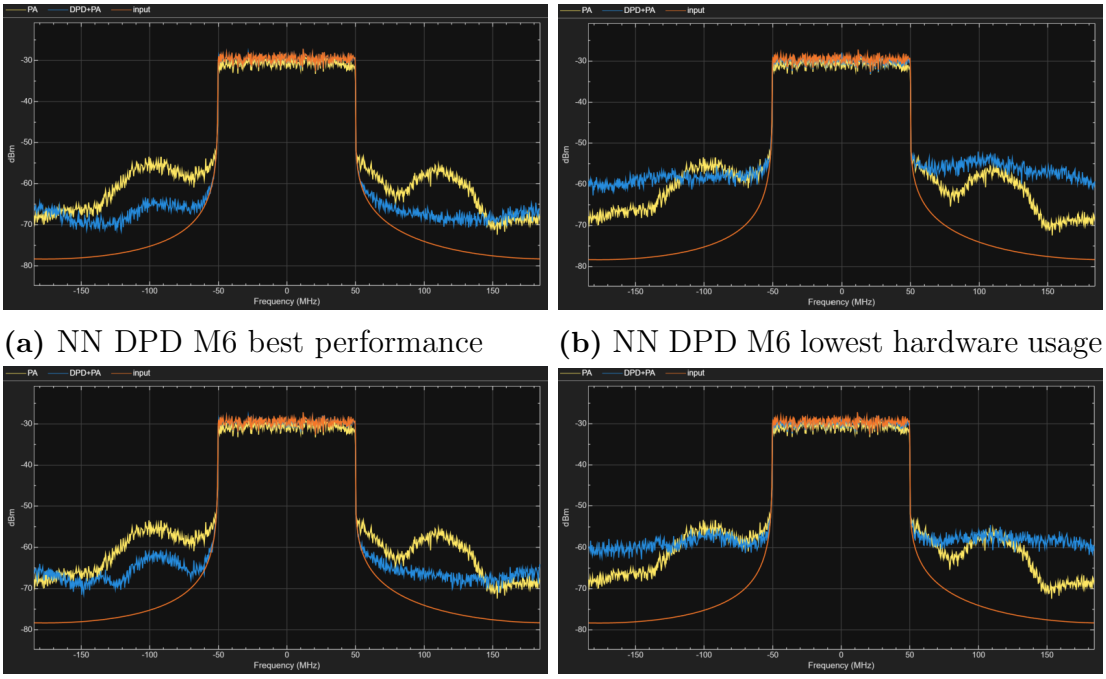
(c) NN DPD M3 best performance

(d) NN DPD M3 lowest hardware usage

Figure 5.15: Visualization of the constellation diagrams for the MP6 PA model, and the combined NN DPD models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.23, B.24, B.25 and B.26.

When looking at the constellation diagrams for the best performing compressed NN DPD models applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the clusters are well centered and concentrated around the ideal undistorted constellation of the evaluation signal, making it easy to identify the symbols of the evaluation signal from the output of the PA. It can also be seen that these constellations are less distorted compared to the one for the legacy DPD, and that the NN DPD with memory depth $M = 6$ has the least distorted constellation. When looking at the constellation diagrams for the NN DPD models with the lowest estimated hardware resource usage applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the clusters are well centered but far from as concentrated around the ideal undistorted constellation of the evaluation signal as for the best performing NN DPD models or the legacy DPD. It appears that the

constellation diagram for the NN DPD model with memory depth $M = 3$ is slightly less distorted than the one for the NN DPD model with memory depth $M = 6$, and it is easier to distinguish between the symbols compared to when applying no DPD prior to the PA, though not by much.



(a) NN DPD M6 best performance

(b) NN DPD M6 lowest hardware usage

(c) NN DPD M3 best performance

(d) NN DPD M3 lowest hardware usage

Figure 5.16: Visualization of the spectrums for the MP6 PA model, and the combined NN DPD models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix B, specifically in Figures B.27, B.28, B.29 and B.30.

When looking at the spectrums for the best performing compressed NN DPD models applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the adjacent channel leakage for both models is less than that of the legacy DPD model applied prior to the PA and that of when no DPD is applied prior to the PA. It can also be seen that the adjacent channel leakage is slightly less for the NN DPD model with memory depth $M = 6$ than for the NN DPD model with memory depth $M = 3$. When instead looking at the spectrums of the NN DPD models with the lowest estimated hardware resource usage applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that there is significantly more adjacent channel leakage than for the best performing NN DPD models. Furthermore, there appears to be about the same amount of adjacent channel leakage for the NN DPD models with the lowest hardware resource usage applied prior to the PA as for the legacy DPD model applied prior to the PA and more adjacent channel leakage than for no DPD applied prior to the PA.

A summary of the estimated hardware resource usage of the NN DFE consisting only of DPD can be found in Table 5.28. The table reports the estimated hardware

resource usage for the compressed NNs with memory depths $M = 6$ and $M = 3$ with the best performance and lowest estimated hardware resource usage.

DPD Model	$\mathbf{MAC/MAC_{strict}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$	$\mathbf{MAC_{pool}}$	TotalMemory	ComputeCost
NN DPD M6 Magnitude-Pruned	4096	–	–	4096	16.5780e+3	4.1943e+6
NN DPD M6 Struct-QAT	2034	–	–	2034	2.1563e+3	1.9504e+5
NN DPD M3 SynFlow-Pruned	3438	–	–	3438	14.0080e+3	3.5205e+6
NN DPD M3 Struct-QAT	1715	–	–	1715	1.8418e+3	1.7361e+5
Legacy DPD	737.53	880.84	3803.22	2298.89	4.4340e+3	4.2482e+5

Table 5.28: Estimation of NN DFE hardware resource usage, for best performing compressed NN DFE models consisting of only DPD.

From the table it can be seen that the floating point NNs and NNs compressed using structured QAT with memory depth $M = 3$ has a lower estimated hardware resource usage compared to the corresponding NNs with memory depth $M = 6$. It can also be seen that the hardware resource numbers in terms of MAC count for the compressed networks are less than half of their respective floating point NN numbers, but still higher than the MAC_{strict} count of the legacy DPD. At the same time, the legacy DPD has remaining adders and registers which are not part of the MAC_{strict} count and the MAC_{pool} count of the legacy DPD is higher than the MAC count of the compressed NNs. Furthermore, it can be seen from the table that the total memory and compute cost of the NNs compressed using structured QAT are significantly lower than for the NNs compressed using magnitude-based or SynFlow pruning, and less than half the total memory and compute cost of the legacy DPD model. Finally, it can be seen that the NN DPD models compressed using magnitude-based and SynFlow pruning for memory depths $M = 6$ and $M = 3$ respectively has a significantly larger estimated hardware resource usage than for the legacy DPD model for all metrics being compared. The achieved reduction in the hardware resource usage of the compressed NN models is within expectation. Apart from the hardware cost of the NNs there is also the small additional hardware cost of the handling of the data for the NNs.

5.3.2 Summary of Digital Front-End Solutions

From tables 5.21 and 5.24 it can be seen that the performance of the uncompressed floating point NN DFE solutions co-optimizing CFR and DPD with memory depths $M = 6$ and $M = 3$ differs a lot compared to the performance of the legacy DFE consisting of CFR, HL and DPD. The ACPR and EVM performance of the NN DFE solutions degrades with lower PAPR thresholds, whereas the ACPR performance improves and EVM performance degrades for the legacy DFE solutions with lower PAPR thresholds. Furthermore, the PAPR performance evaluated after the NN DFE models is better/closer to the threshold compared to the PAPR performance evaluated from the legacy DFE models. This is the expected performance for co-optimized NN DFE models for varying PAPR thresholds.

A summary of the system performance when applying the NN DFE jointly modeling CFR and DPD can be found in Table 5.29, where the performance of the compressed NNs with memory depth $M = 6$ and $M = 3$ with the best performance and lowest

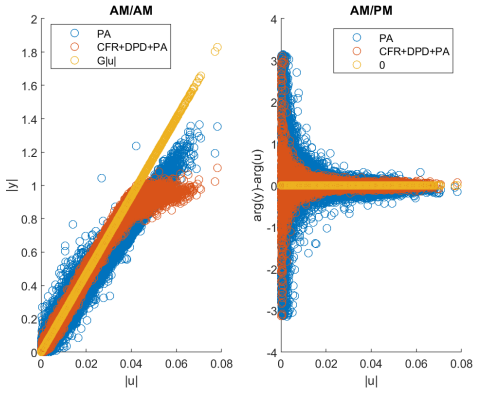
estimated hardware resource usage as well as their uncompressed floating point counterparts is reported for the PAPR threshold 6.5 dB. The ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

DFE Model	ACPR (dBc)		EVM (dB)	PAPR (dB)
NN DFE M6 FP	−33.307	− 34.007	−28.149	7.8456
NN DFE M3 FP	$-32.72j0$	− 33.410	−26.816	7.1556
NN DFE M6 Struct-Projected	−32.942	− 33.408	−27.887	7.4237
NN DFE M6 Struct-QAT	−28.688	− 27.332	−22.893	8.6560
NN DFE M3 Struct-Pruned	−32.303	− 33.126	−26.753	7.0661
NN DFE M3 Struct-QAT	−29.043	− 28.162	−23.700	7.9305
Legacy CFR(2)+HL+DPD	−32.615	− 32.171	−23.762	8.0155
PA (without DFE)	−27.994	− 29.530	−15.830	11.8630

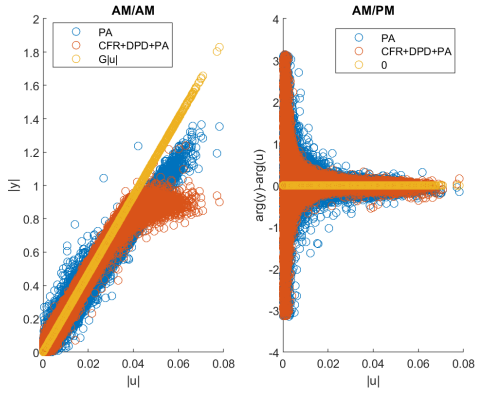
Table 5.29: Summary of system performance for best performing compressed NNs when applying NN DFE jointly modeling CFR and DPD prior to PA. ACPR and EVM are evaluated after the PA and the PAPR is evaluated prior to the PA.

From the table it can be seen that the compressed NN DFE models co-optimizing CFR and DPD with the best performance, i.e. the NN DFE compressed using structured projection for memory depth $M = 6$ and the NN DFE compressed using structured pruning for memory depth $M = 3$, outperforms the legacy DFE consisting of CFR, HL and DPD. Furthermore, applying the best performing compressed NN DFE models prior to the PA gives significantly better performance compared to applying no DFE prior to the PA. When comparing the performance of the uncompressed floating point NN DFE models to that of the best performing compressed NN DFE models, it can be seen that the floating point NN models has better performance than the best performing compressed NN models in terms of ACPR and EVM, but not in terms of PAPR for the respective memory depths. It can also be seen that the NN DFE models co-optimizing CFR and DPD with the lowest estimated hardware resource usage, i.e. the NN DFE models compressed using structured QAT for both memory depth $M = 6$ and $M = 3$, doesn't perform as well as the legacy DFE consisting of CFR, HL and DPD, both in terms of ACPR and EVM. The PAPR performance for the NN DFE compressed using structured QAT with memory depth $M = 3$ is slightly better than for the legacy DFE, while the NN DFE compressed using structured QAT with memory depth $M = 6$ has significantly worse performance than the legacy DFE in terms of PAPR. It is not surprising that the NN DFE compressed using structured QAT would perform worse than the legacy DFE, especially in terms of ACPR, considering that there is not much performance margin between the uncompressed floating point NN DFE models and the legacy DFE model. Finally, it can be seen that applying the NN DFE models with the lowest hardware usage prior to the PA gives slightly worse performance in terms of ACPR and significantly better performance in terms of EVM and PAPR compared to applying no DFE prior to the PA. The expectation was to achieve a larger performance margin in ACPR for the floating point and best performing NN DFE models. The EVM and PAPR performance improvement for the NN DFE are however within expectation. Finally, the application of the compression techniques had again an unexpected amount of performance degradation.

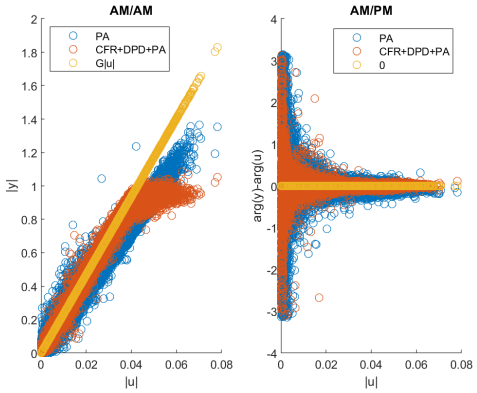
A visualization of the results of the compressed NN DFE models in Table 5.27 can be found in Figures 5.17, 5.18, 5.19 and 5.20. The figures displays the AM/AM and AM/PM distortions, the CCDF curves, the constellation diagrams and the spectrums of the compressed NNs in the table.



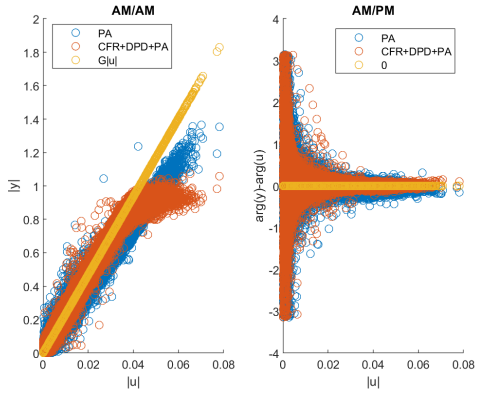
(a) NN DFE M6 best performance



(b) NN DFE M6 lowest hardware usage



(c) NN DFE M3 best performance

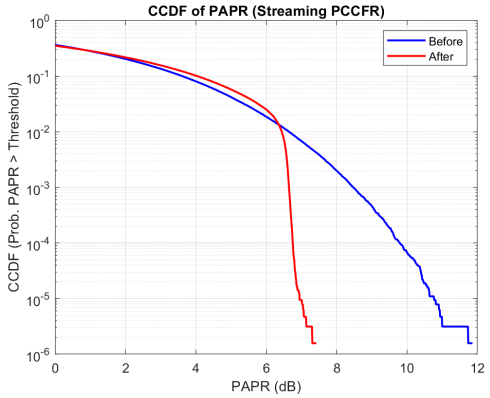


(d) NN DFE M3 lowest hardware usage

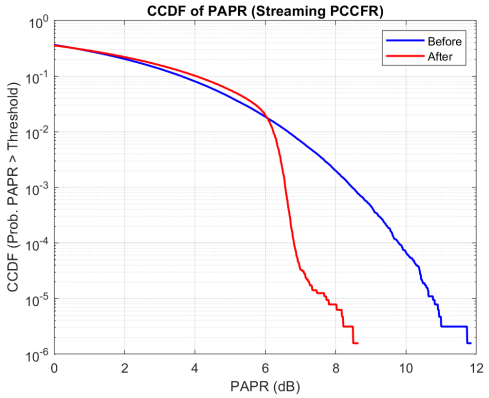
Figure 5.17: Visualization of the AM/AM and AM/PM distortions for the MP6 PA model, and the combined NN DFE jointly modeling CFR and DPD with PAPR threshold 6.5 dB models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.15, C.16, C.17 and C.18.

When looking at the AM/AM and AM/PM distortions for the compressed NN DFE models co-optimizing CFR and DPD with the best performance for memory depths $M = 6$ and $M = 3$, it can be seen that the combined NN DFE models and PA results in rather good linearization performance. The output amplitudes are better centered around the ideal linear gain and the phase shifts are better centered around the ideal zero phase shift compared to when applying no DFE prior to the PA for both memory depths. The linearization performance for the NN DFE with memory depth $M = 6$ appears to be slightly better than that for the NN DFE with memory depth $M = 3$, though both NN DFE models have better PA linearization performance than the legacy DFE models consisting of CFR, HL and DPD. When

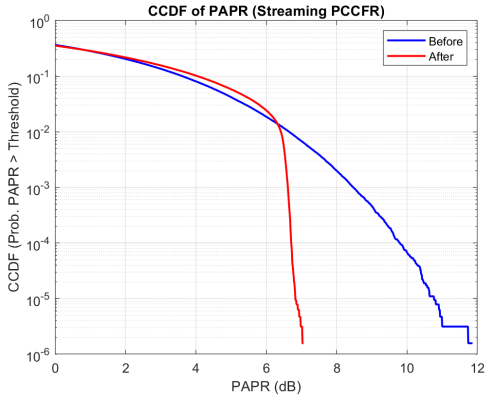
instead looking at the AM/AM and AM/PM distortions for the NN DFE models with the lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$, it can be seen that the PA linearization performance is not as good as that for the best performing NN DFE models. It does however appear that the amount of amplitude and phase distortions for the NN DFE models with the lowest estimated hardware resource usage for both memory depths is sightly better than that of the legacy DFE models consisting of CFR.



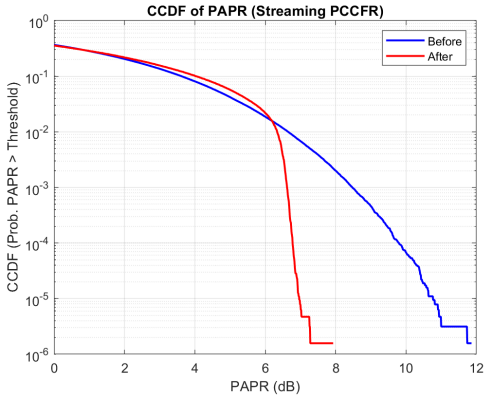
(a) NN DFE M6 best performance



(b) NN DFE M6 lowest hardware usage



(c) NN DFE M3 best performance

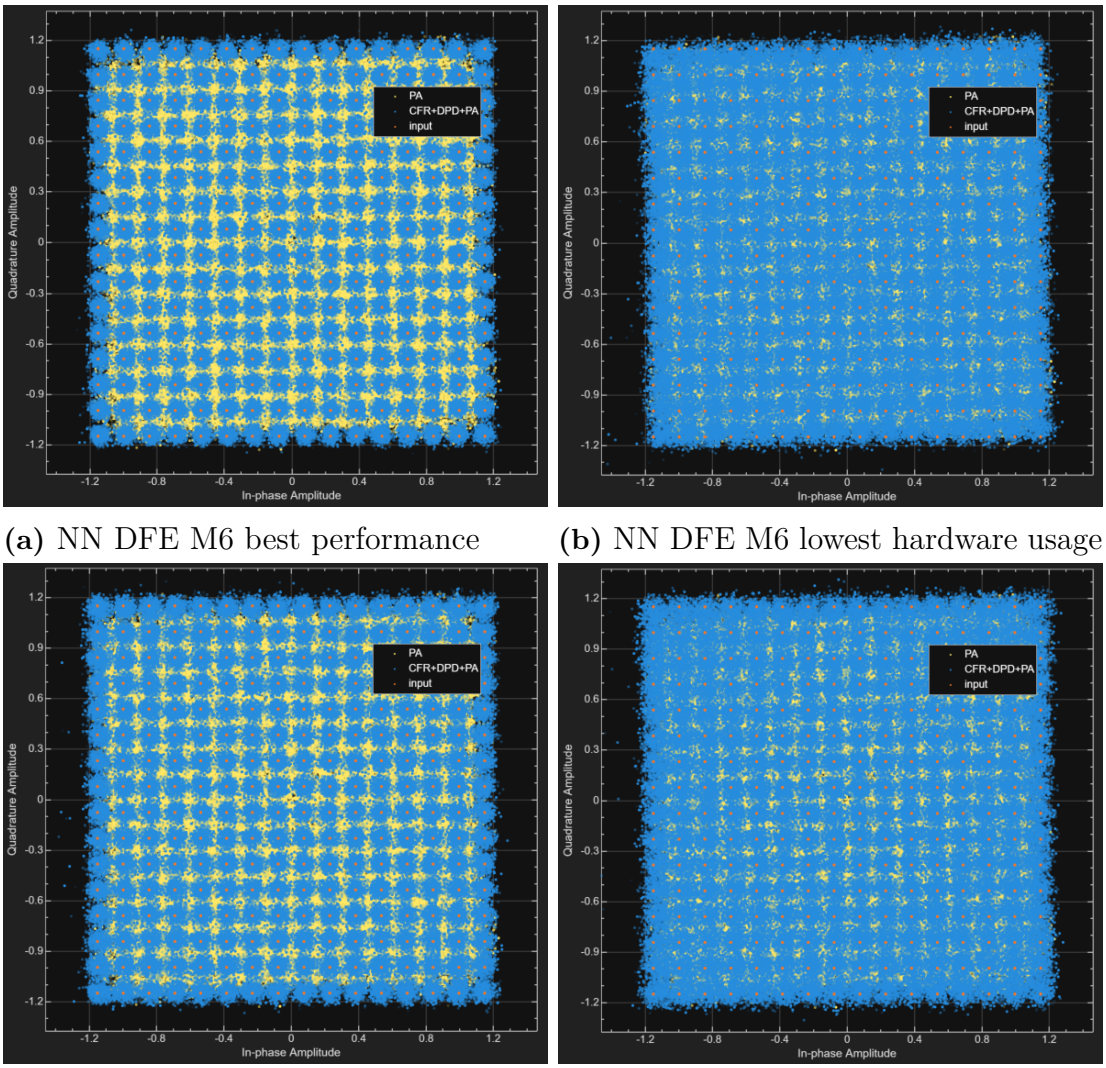


(d) NN DFE M3 lowest hardware usage

Figure 5.18: Visualization of the CCDF curves for the NN DFE jointly modeling CFR and DPD with PAPR threshold 6.5 dB model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.19, C.20, C.21 and C.22.

When looking at the CCDF curves for the compressed NN DFE models co-optimizing CFR and DPD with the best performance for memory depths $M = 6$ and $M = 3$, it can be seen that there is a steep decline in probability for PAPR thresholds over 6.5 dB. These CCDF curves are very similar to those for the legacy DFE consisting of CFR, HL and DPD, only the decline starts for thresholds 1 dB lower. When instead looking at the CCDF curves for the NN DFE models co-optimizing CFR and DPD with the lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$, it can be seen that there is also a decline

in probability, though not as steep as for the best performing NN DFE models. Furthermore, this decline starts earlier for PAPR thresholds over 6.5 dB.



(a) NN DFE M6 best performance

(b) NN DFE M6 lowest hardware usage

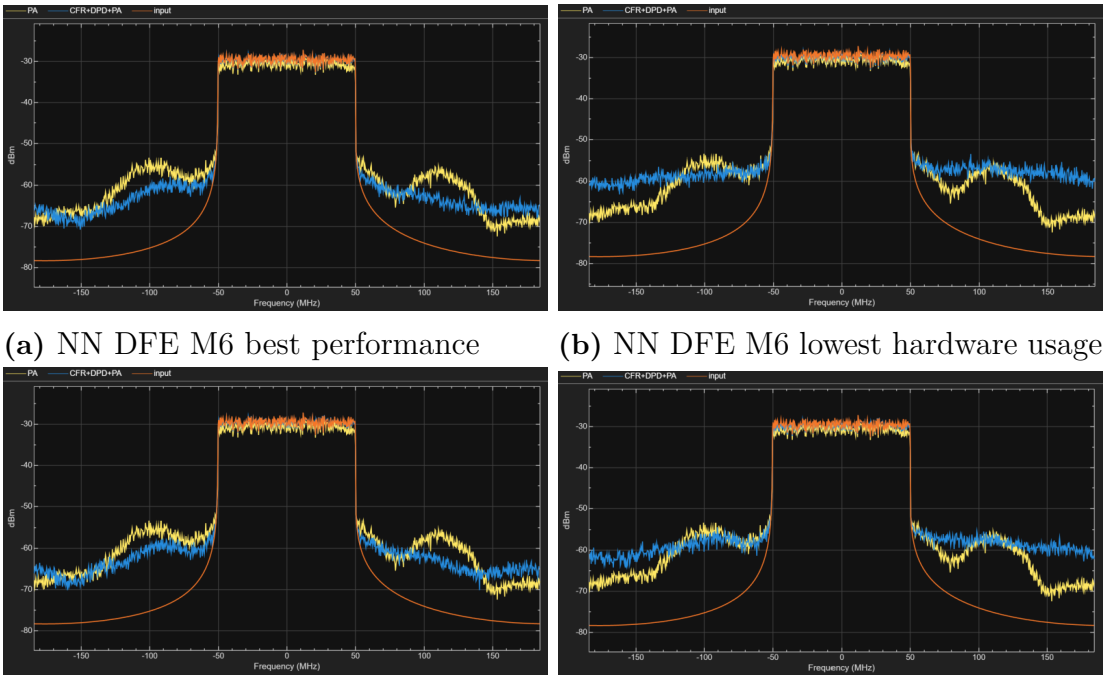
(c) NN DFE M3 best performance

(d) NN DFE M3 lowest hardware usage

Figure 5.19: Visualization of the constellation diagrams for the MP6 PA model, and the combined NN DFE jointly modeling CFR and DPD with PAPR threshold 6.5 dB models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.23, C.24, C.25 and C.26.

When looking at the constellation diagrams for the best performing compressed NN DFE models co-optimizing CFR and DPD applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the clusters are well centered and fairly concentrated around the ideal undistorted constellation of the evaluation signal, making it mostly possible to identify the symbols of the evaluation signal from the output of the PA. It can also be seen that these constellations are significantly less distorted compared to the ones for the legacy DFE consisting of CFR, HL and DPD, and that the NN DFE with memory depth $M = 6$ has the least distorted

constellation. When looking at the constellation diagrams for the NN DFE models with the lowest estimated hardware resource usage applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the clusters are well centered but far from as concentrated around the ideal undistorted constellation of the evaluation signal as for the best performing NN DFE models or the legacy DFE model consisting of CFR, HL and DPD. It appears that the constellation diagram for the NN DFE model with memory depth $M = 3$ is slightly less distorted than the one for the NN DPD model with memory depth $M = 6$, and it is marginally easier to distinguish between the symbols compared to when applying no DPD prior to the PA.



(a) NN DFE M6 best performance

(b) NN DFE M6 lowest hardware usage

(c) NN DFE M3 best performance

(d) NN DFE M3 lowest hardware usage

Figure 5.20: Visualization of the spectrums for the MP6 PA model, and the combined NN DFE jointly modeling CFR and DPD with PAPR threshold 6.5 dB models and MP6 PA model applied to the evaluation signal. Magnifications of the subfigures can be found in Appendix C, specifically in Figures C.27, C.28, C.29 and C.30.

When looking at the spectrums for the best performing compressed NN DFE models co-optimizing CFR and DPD applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that the adjacent channel leakage for both models is about the same or slightly more than that of the legacy DFE model consisting of CFR, HL and DPD applied prior to the PA and less than that of when no DFE is applied prior to the PA. It can also be seen that the adjacent channel leakage is marginally less for the NN DFE model with memory depth $M = 6$ than for the NN DFE model with memory depth $M = 3$. When instead looking at the spectrums of the NN DFE models with the lowest estimated hardware resource usage applied prior to the PA for memory depths $M = 6$ and $M = 3$, it can be seen that there is

significantly more adjacent channel leakage than for the best performing NN DFE models and for the legacy DFE model. Furthermore, there is more adjacent channel leakage for the NN DFE models with the lowest hardware resource usage applied prior to the PA than for no DFE applied prior to the PA.

A summary of the estimated hardware resource usage of the NN DFE jointly modeling CFR and DPD can be found in Table 5.30. The table reports the estimated hardware resource usage for the compressed NNs with memory depths $M = 6$ and $M = 3$ with the best performance and lowest estimated hardware resource usage for PAPR threshold 6.5 dB.

DFE Model	$\mathbf{MAC/MAC_{strict}}$	$\mathbf{Add_{Rem}}$	$\mathbf{Reg_{Rem}}$	$\mathbf{MAC_{pool}}$	TotalMemory	ComputeCost
NN DFE M6 Struct-Projected	1934	–	–	1934	8.2344e+3	1.9804e+6
NN DFE M6 Struct-QAT	2029	–	–	2029	2.1514e+3	1.9606e+5
NN DFE M3 Struct-Pruned	2901	–	–	2901	11.9100e+3	2.9706e+6
NN DFE M3 Struct-QAT	1669	–	–	1669	1.7969e+3	1.7067e+5
Legacy CFR(2)+HL+DPD	1339.78	10927.719	5314.969	6754.01	6.8750e+3	7.7167e+5

Table 5.30: Estimation of NN DFE hardware resource usage, for best performing compressed NN DFEs jointly modeling CFR and DPD with PAPR threshold 6.5 dB.

From the table it can be seen that the estimated hardware resource usage for the compressed NN DFE models co-optimizing CFR and DPD with the best performance and lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$ are rather similar to those for the corresponding DPD models in table 5.28. The estimated hardware resource usage of the best performing compressed NN DFE model with memory depth $M = 6$, i.e. the NN DFE compressed using structured projection, is comparable to that of the legacy DFE consisting of CFR, HL and DPD. The MAC count of the NN DFE model is a bit higher than the MAC_{strict} count of the legacy DFE model, though the legacy DFE also has remaining adders and registers, and the MAC_{pool} count of the legacy DFE is higher than the MAC count of the NN DFE model. Furthermore, the total memory and compute cost of the NN DFE model is higher than that of the legacy DFE model. The estimated hardware resource usage for the best performing NN DFE model with memory depth $M = 3$, i.e. the NN DFE compressed using structured pruning, is higher than that of the best performing NN DFE model with memory depth $M = 6$. As such the MAC count, total memory and compute cost of the NN DFE model is also higher than that of the legacy DFE model, though the MAC count of the NN DFE model is still lower than the MAC_{pool} count of the legacy DFE model. Finally, the estimated hardware resource usage for the NN DFE models with the lowest estimated hardware resource usage for memory depths $M = 6$ and $M = 3$, i.e. the NN DFEs compressed using structured QAT for those memory depths, are slightly lower than for the corresponding NN DPD models with the lowest estimated hardware resource usage, also compressed using structured QAT. On the other hand, the estimated hardware resource usage for the legacy DFE consisting of CFR, HL and DPD is higher compared to that for the legacy DFE consisting only of DPD. As such the estimated hardware resource usage for the NN DFE looks better in comparison. The achieved reduction in the hardware resource usage of the compressed NN models is

again within expectation. Apart from the hardware cost of the NNs there is again also the small additional hardware cost of the handling of the data for the NNs.

6

Conclusions and Future Work

This thesis investigated the use of NNs for modeling a co-optimized DFE for wireless communication systems with PAs exhibiting memory and non-linearity effects. To evaluate the system performance a functional reference DFE model was developed, referred to as the legacy DFE, which was implemented using established algorithms in a hardware compatible manner. First, the legacy DFE consisted only of DPD and was later expanded to also include CFR and HL. Then a NN architecture and loss function was investigated, and NN models were trained to model DPD, which were then compressed using the compression techniques pruning, projection and quantization. The performance of these NN DPD models was then compared to that of the legacy DPD in terms of ACPR and EVM. The NN loss function was then extended to add CFR and jointly co-optimize CFR and DPD. NN models were then trained to model the co-optimized DFE, which were also compressed using the same compression techniques. The performance of the NN DFE could then be compared to that of the legacy DFE consisting of CFR, HL and DPD, now in terms of ACPR, EVM and PAPR. This comparison was made for various PAPR thresholds, and more in depth for the threshold 6.5 dB. To evaluate the hardware cost of the solutions, bit-exact models were developed by setting up the NN-based and legacy solutions for hardware implementation first through fixed point conversion and then followed by HDL code generation. From this the hardware resource utilization of the solutions could be estimated and compared in terms of MAC count, memory footprint and computational complexity.

The legacy DFE model was implemented using PCCFR with 1, 2 or 3 clip stages for the CFR and MP with a dual polyphase architecture with LUTs for the DPD. The performance when applying the DFE consisting only of DPD prior to the PA was worse than applying no DFE prior to the PA. This performance was improved when also the CFR and HL were added which lowered the PAPR prior to the DPD, and the performance applying the legacy DFE consisting of CFR, HL and DPD prior to the PA is better than when applying no DFE prior to the PA. Furthermore, the performance when applying the legacy DFE consisting of CFR, HL and DPD was very similar regardless of the number of clip stages of the CFR, with only a marginal improvement when using more clip stages.

The floating point NN DFE models as well as the best performing compressed NN DFE models outperform the legacy DFE models, both when only DPD is modeled and when CFR and HL are considered. However, the hardware costs of the uncompressed floating point NN DFE models is much higher than that of the legacy

DFE models and the hardware cost for the best performing compressed NN DFE models is lower than that of their respective floating point NN models, but still higher than that of the legacy DFE models. When comparing the estimated hardware resource usage of the NN DPD models to that of the legacy DFE consisting only of DPD, it can be seen that the difference in hardware resource usage is quite large. When instead comparing the estimated hardware resource usage of the NN DFE co-optimizing CFR and DPD to that of the legacy DFE consisting of CFR, HL and DPD, it can be seen the gap in resource usage is much smaller compared to the case when only DPD was modeled. This makes the co-optimized NN solution competitive to the legacy solution.

The NN DFE models compressed using all three compression techniques discussed in this work ending at QAT has a much lower hardware cost compared to their respective initial floating point models. The memory footprint and computational complexity of these compressed NN models is even lower than for the legacy DFE models, though the MAC count is still higher for the compressed NN models than the MAC_{strict} count for the legacy solution. When taking the remaining adders and registers of the legacy DFE into account, the MAC count for the compressed NN DFE models is lower than the MAC_{pool} count for the legacy DFE models. This gives compatible hardware complexity for the compressed NN DFE models and the legacy DFE models. However, the hardware cost reduction comes at the price of some performance loss, in which the compressed NN DFE models performs worse than the corresponding legacy DFE models in terms of ACPR and EVM. For the NN DFE models modeling only DPD there is more performance margin between the NN solution and corresponding legacy solution than there is between the NN DFE co-optimizing CFR and DPD, and the legacy DFE consisting of CFR, HL and DPD for PAPR threshold 6.5 dB. However, both compressions results in NN DFE models with worse performance than their legacy counterparts when applying compression ending at QAT.

The NN DFE models compressed using the three compression techniques ending at projection has a much smaller loss of performance compared to those compressed with the three compression techniques ending at QAT. The hardware cost in terms of memory footprint and computational complexity is about four times larger than for the NN DFEs compressed using the three compression techniques ending at QAT, but are still comparable to those of the legacy DFE models. Furthermore the MAC count after the two compression stages are rather similar with the MAC count after the projection even being slightly smaller than after quantization. The NN models compressed using the three compression techniques ending at projection should as such also be considered when modeling the DFE using NNs.

The results demonstrates that co-optimizing the DFE using NNs shows promise. However, further investigations are required, regarding improving the performance of the uncompressed floating point NN DFE models, as well as improving the compression techniques to minimize the loss in performance, while retaining the reduction in hardware cost.

6.1 Future Work

Several opportunities exist for extending and improving the work presented in this thesis. To begin with, different means of further improving the performance of the uncompressed floating point NN models can be investigated. First, the weight distribution of the joint loss function can be refined to optimize the overall NN DFE performance, both for NN DFE models modeling only DPD and for NN DFE models jointly co-optimizing CFR and DPD. To enhance the joint optimization of ACPR, EVM and PAPR, the individual loss functions could also be further investigated and improved. In order to find operating points that maximize hardware efficiency while preserving acceptable communication performance, a more thorough analysis of the trade-off between performance margin and resource consumption is also necessary.

Future studies also need to look at more compression methods and a wider range of compression parameters. Using the available quantization tools, quantization was mostly restricted to FP32-to-INT8 conversion in this work. Higher compression ratios without sacrificing performance could be achieved by investigating other compression configurations, lower precision representations, and alternative quantization algorithms.

To evaluate the scalability of the suggested NN model framework, both for NN DFE models modeling only DPD and jointly co-optimizing CFR and DPD, in increasingly demanding communication scenarios, using signals with larger bandwidths and higher-order modulation schemes should be investigated. Furthermore, developing NN-based co-optimized CFR and DPD solutions that can handle signals with different parameters, such as shifting bandwidths, modulation formats, and PAPR characteristics, is another interesting point of investigation. Since conventional formulations frequently assume relatively stable signal statistics throughout the signal, such scenarios may necessitate the exploration of alternative PAPR-related loss functions.

Future research could also look into expanding the DFE functionality and working with the NN to jointly co-optimize more DFE functionalities. Lastly, the projected hardware metrics and resource usage results reported in this work could be validated by putting the suggested NN DFE architectures into practice on real hardware platforms. A more thorough evaluation of the practical feasibility of compressed NN DFE systems would involve comparing the measured hardware performance with simulation findings and testing the system with a real PA.

Bibliography

[1] A. Falempin, "Adaptive pre-distortion for power amplifier linearization based on neural networks," Ph.D. dissertation, Universiteté Grenoble Alpes, 2022.

[2] S. Wang, M. Roger, and C. Lelandais-Perrault, "Impacts of crest factor reduction and digital predistortion on linearity and power efficiency of power amplifiers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 3, pp. 407–411, 2019. [Online]. Available: https://doi.org/10.1109/TCSII.2018.2855084

[3] X. Wang and J. Wang, "A combined cfr-dpd approach for rf power amplifier," *Proceedings of the 2020 International Conference on Microwave and Milimeter Wave Technology (ICMMT)*, 2022.

[4] E. Bala, L. Kazakevich, and R. Yang, "Techniques to improve power amplifier energy efficiency for 5g," *Proceedings of the 2014 1st International Conference on 5G for Ubiquitous Connectivity, 5GU 2014*, pp. 104–109, 2014. [Online]. Available: https://doi.org/10.4108/icst.5gu.2014.257998

[5] D. R. Morgan, Z. Ma, J. Kim, M. G. Zierdt, and J. Pastalan, "A generalized memory polynomial model for digital predistortion of rf power amplifiers," *IEEE Transactions on Signal Processing*, vol. 54, no. 10, pp. 3852–3860, 2006. [Online]. Available: https://doi.org/10.1109/TSP.2006.879264

[6] L. Sun, X. Hu, Z. Liu, K. Han, S. Zhang, and W. Wang, "A low complexity lut-based digital predistortion block with new pruning method," *IEEE Microwave and Wireless Components Letters*, vol. 32, no. 9, pp. 1131–1134, 2022. [Online]. Available: https://doi.org/10.1109/LMWC.2022.3163890

[7] C. Zhao, R. J. Baxley, G. T. Zhou, D. Boppana, and J. S. Kenney, "Constrained clipping for crest factor reduction in multiple-user ofdm," in *2007 IEEE Radio and Wireless Symposium*, 2007.

[8] G. Olsson, "Joint cfr and dpd optimization for 5g digital front ends," Master's thesis, Lund University, Faculty of Engineering (LTH), Sweden, 2023.

[9] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, mar 2026, ch. 2,4,5,6, (Accessed 2026-06-25). [Online]. Available: https://smlbook.org

[10] I. Goodfellow, Y. Bongio, and A. Courville, *Deep Learning*. MIT press, 2016, ch. 6-8, (Accessed 2026-06-25). [Online]. Available: http://www.deeplearningbook.org

[11] P. Jaraut, M. Helaoui, W. Chen, M. Rawat, N. Boulejfen, and F. M. Ghannouchi, "Review of the neural network based digital predistortion linearization of multi-band/mimo transmitters." Nanjing, China: 2021 IEEE MTT-S International Wireless Symposium (IWS), 2021, pp. 1–3. [Online]. Available: https://doi.org/10.1109/IWS52775.2021.9499466

[12] B. S. de C. da Silva, P. H. C. de Souza, and L. L. Mendes, "Neural network approaches for papr reduction in ofdm systems," in *2025 Workshop on Communication Networks and Power Systems (WCNPS)*. Brasilia, Brazil: IEEE, 2025, pp. 1–7.

[13] Q. Zhang, R. Han, C. Jiang, J. Wang, H. Chang, and F. Liu, "End-to-end joint optimization for papr reduction and digital predistortion based on neural network," *IEEE Microwave and Wireless Technology Letters*, vol. 35, no. 5, pp. 509–512, 2025. [Online]. Available: https://doi.org/10.1109/LMWT.2025.3546643

[14] A. Hofvander and N. A. Attupurath, "Neural transmitter with co-optimized digital front end functions," Master's thesis, Lund University, Faculty of Engineering (LTH), Sweden, 2026.

[15] The MathWorks, Inc., *Fixed-Point and Floating-Point Basics (Release 2026a)*, The MathWorks, Inc., Natick, Massachusetts, United States, 2026. [Online]. Available: https://www.mathworks.com

[16] P. Larsson-Edefors, "Energy-efficient computation of TensorFloat32 numbers on an FP32 multiplier," in *IFIP/IEEE 33rd Int. Conf. on Very Large Scale Integration (VLSI-SoC)*, 2025.

[17] P. Larsson-Edefors and E. Börjeson, "Efficacy of pipelining to reduce energy of floating-point adders and multipliers," in *IEEE 33rd Symp. Computer Arithmetic (ARITH)*, 2026, pp. 161–166.

[18] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019.

[19] S. Diehl, *Nonlinear Optimization: A Basic Course*, 2nd ed. Studentlitteratur, 2025, pp. 92–94.

[20] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing: Pearson New International Edition*, 4th ed. Pearson Education Limited, 2013, pp. 120–121,900,922–936.

[21] ——, *Digital Signal Processing: Pearson New International Edition*, 4th ed. Pearson Education Limited, 2013, pp. 471–472,567.

[22] A. Author, *Handbook of Serial Communications Interfaces: A Comprehensive Compendium of Serial Digital Input/output (I/o) Standards*. Newnes, 2016, ch. 63, pp. 235–243. [Online]. Available: https://doi.org/10.1016/B978-0-12-800629-0.00063-2

[23] A. Autor, *Electronics Explained: Fundamentals for Engineers, Technicians, and Makers*, 2nd ed. Newnes, 2018, ch. 8, pp. 195–216. [Online]. Available: https://doi.org/10.1016/B978-0-12-811641-8.00008-4

[24] P. von Butovitsch, H. Asplund, D. Astely, T. Chapman, M. Frenne, F. Ghasemzadeh, M. Hagström, B. Hogan, G. Jöngren, J. Karlsson, F. Kronestedt, and E. Larsson, *Advanced Antenna Systems for 5G Network Deployments: Bridgeing the Gap Between Theory and Practize*. Academic Press (Elsevir), 2020, ch. 5. [Online]. Available: https://doi.org/10.1016/B978-0-12-820046-9.00005-8

[25] M. Araújo and M. Melo, "Ofdm: Theory and applications," *The Journal (Institute of Telecommunications Professionals)*, vol. 4, no. 1, pp. 32–37, 2010. [Online]. Available: https://www.researchgate.net/publication/290774058_OFDM_Theory_and_applications

[26] J. Luo, A. Kortke, and W. Keusgen, "Guardband optimization for cellular systems applying raised cosine windowed ofdm," *Wireless Personal Communications*, vol. 78, pp. 1375–1390, 2014. [Online]. Available: https://doi.org/10.1007/s11277-014-1822-z

[27] H. Ali and F. Khan, "On the performance analysis of one tap equalizers in oversampled ofdm systems," *International Journal of Communications, Network and System Sciences*, vol. 11, no. 9, pp. 187–198, 2018.

[28] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing: Pearson New International Edition*, 4th ed. Pearson Education Limited, 2013, p. 28.

[29] J. Wenhao, J. Jie, and F. Yingying, "An enhanced peak cancellation algorithm for papr reduction with variable coefficient length and recoverable leak peak," *IEEE Access*, 2025.

[30] S. Wang, M. Roger, J. Sarrazin, and C. Lelandais-Perrault, "A joint crest factor reduction and digital predistortion for power amplifiers linearization based on clipping-and-bank-filtering," *IEEE Transactions on Microwave Theory and Techniques*, vol. 68, no. 4, pp. 1390–1402, 2020.

[31] C. Zhao, "Distortion-based crest factor reduction algorithms in multi-carrier transmission systems," Ph.D. dissertation, Georgia Institute of Technology, School of Information & Computer Science, Atlanta, GA, USA.

[32] F. Hoshino and T. Grosch, "Impacts of threshold clipping on bit error rate in ofdm-like systems," *Kennesaw Journal of Undergraduate Research*, vol. 5, no. 1, 2017, article nbr 2. [Online]. Available: https://doi.org/10.32727/25.2019.15

[33] I. Analog Devices, "Adrv904x dfe system level overview," (acessed 2026-07-30).

[34] M. Qing and Z. Luo, "A combined approach of dpd and cfr in f-ofdm system," *Wireless Personal Communications*, vol. 114, pp. 2681–2691, 2020. [Online]. Available: https://doi.org/10.1007/s11277-020-07497-7

[35] Y. Gau, C. Yu, and A. Zhu, "Power adaptive digital predistortion for wideband rf power amplifiers with dynamic power transmission," *IEEE Transactions on Microwave Theory and Techniques*, vol. 63, no. 11, pp. 3595–3607, 2015. [Online]. Available: https://doi.org/10.1109/TMTT.2015.2480739

[36] A. Corporation, "Designing polyphase dpd solutions with 28-nm fpgas," 2012, (Accessed 2026-06-

21). [Online]. Available: https://docs.altera.com/v/u/docs/650543/designing-polyphase-dpd-solutions-with-28-nm-fpgas-white-paper

[37] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*, 2nd ed. Elsevier Science, ch. 3, pp. 108–171. [Online]. Available: https://doi.org/10.1016/B978-0-12-394424-5.00003-3

[38] B. Inc, "What is a lookup table (lut)?" https://bltinc.com/2025/08/20/what-is-a-lookup-table-lut/, aug 2025, (Accessed 2026-06-18).

[39] X. Feng, Y. Wand, B. Feuvrie, A. S. Descamps, and Y. Ding, "Analysis on lut based digital predistortion using direct learning architecture for linearizing power amplifiers," *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, 2016, article nbr 132. [Online]. Available: https://doi.org/10.1186/s13638-016-0628-y

[40] C. Masterson, "Digital predistortion for rf communications: From equations to implementation," *Analog Dialogue*, vol. 56, no. 2, 2022, (Accessed 2026-06-21). [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/digital-predistortion-for-rf-communication.html

[41] P. Gilbert, "Multi look-up table digital predistortion for rf power amplifier linearization," Ph.D. dissertation, Universitat Politècnica de Catalunya, 2007.

[42] J. D. Concori and T. Hammarbäck, "A digital design flow - from concept to rtl description, using mathworks and cadence's tools," Master's Thesis, Department of Electrical and Information Technology, Faculty of Engineering, LTH, Lund University, Lund, Sweden, 2022. [Online]. Available: https://lup.lub.lu.se/student-papers/search/publication/9101845

[43] K. Kintali and Y. Gu, "Model-based design with simulink, hdl coder, and xilinx system generator for dsp," Tech. Rep.

[44] G. Kaur, N. Gupta, and M. Singh, "Fixed-point simulink designs for automatic hdl generation of binary dilation & erosion," in *CT Institute of Engineering, Management and Technology*, Jalandhar, India, 2014.

[45] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a polynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, 1993. [Online]. Available: https://doi.org/10.1016/S0893-6080(05)80131-5

[46] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006, ch. 5, (Accessed 2026-06-25). [Online]. Available: https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

[47] G. Xu, X. Wang, X. Wu, X. Leng, and Y. Xu, "Development of residual learning in deep neural networks for computer vision: A survey," *Engineering Applications of Artificial Intelligence*, vol. 142, 2025, article nbr 109890. [Online]. Available: https://doi.org/10.1016/j.engappai.2024.109890

[48] S. Chatterjee, "Convergence of gradient descent for deep neural networks," 2022, (last revised 2026-02-20). [Online]. Available: https://arxiv.org/abs/2203.16462

[49] B. Zhao, R. Walters, and R. Yu, "Symmetry in neural network parameter spaces," *ArXiv*, 2025, (last revised 2025-12-10). [Online]. Available: https://arxiv.org/abs/2506.13018

[50] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization." San Diego: 3rd International Conference for Learning Representations, 2015, pp. 1–15, (Revised 2017-01-30). [Online]. Available: https://doi.org/10.48550/arXiv.1412.6980

[51] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[52] A. F. Agarp, "Deep learning using rectified linear units (relu)," Manila, Philippines, 2018, (revised 2026-04-14). [Online]. Available: https://doi.org/10.48550/arXiv.1803.08375

[53] M. M. Hammad, "Comprehensive survey of complex-valued neural networks: Insights into backpropagation and activation functions," Egypt, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.19258

[54] R. Zayani, R. Bouallegue, and D. Roviras, "An adaptive neural network predistorter for non stationary hpa in ofdm systems." Poznan, Poland: 15th European Signal Processing Conference, 2007, pp. 1352–1356.

[55] R. Zayani, Y. Medjahdi, H. Bouhadda, H. Shaiek, D. Roviras, and R. Bouallegue, "Adaptive predistortion techniques for non-linearly amplified fbmc-oqam signals." Seoul, South Korea: 2014 IEEE 79th Vehicular Technology Conference (VTC Spring), 2014, pp. 1–5. [Online]. Available: https://doi.org/10.1109/VTCSpring.2014.7022810

[56] C. Tarver, L. Jiang, A. Sefidi, and J. R. Cavallaro, "Neural network dpd via backpropagation through a neural network model of the pa." Pacific Grove, CA, USA: 2019 53rd Asilomar Conference on Signals, Systems, and Computers, 2019. [Online]. Available: https://doi.org/10.1109/IEEECONF44664.2019.9048910

[57] G. Giachnakis, "Compressed machine learning models for digital front-end processing in wireless communication systems," Master's thesis, Kungliga Tekniska Högskolan, Sweden, 2025.

[58] S. Ghosh, "Machine learning based digital predistortion considering power amplifier variations," Master's thesis, Kungliga Tekniska Högskolan, Sweden, 2025.

[59] Y. Wu, U. Gustavsson, A. Graell i Amat, and H. Wymeersch, "Residual neural networks for digital predistortion." Taipei, Taiwan: GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/GLOBECOM42002.2020.9322327

[60] "Neural network for digital predistortion design-offline training," MathWorks, 2022, (Accessed 2026-07-06). [Online]. Available: https://se.mathworks.com/help/comm/ug/neural-network-for-digital-predistortion-design-offline-training.html

[61] "Power amplifier modeling using neural networks," MathWorks, 2024, (Accessed 2026-07-06). [Online]. Available: https://se.mathworks.com/help/comm/ug/power-amplifier-modeling-using-neural-networks.html

[62] A. V. Mayakannan, C. Arvind, G. Sasikala, B. Sathyasri, K. Srihari, and V. K. Shanmuganathan, "Neural network-based regression assisted papr reduction method for ofdm systems," *Sādhanā*, vol. 47, 2022, article nbr 242. [Online]. Available: https://doi.org/10.1007/s12046-022-02024-9

[63] A. Kumar and A. Nanthaamornphong, "Nn-pts: a neural network-assisted papr reduction technique for otfs with high-order modulation," *High Energy Density Physisc*, vol. 56, 2025, article nbr 101225. [Online]. Available: https://doi.org/10.1016/j.hedp.2025.101225

[64] S. Wang, M. Roger, J. Sarrazin, and C. Lelandais-Perrault, "Augmented iterative learning control for neural-network-based joint crest factor reduction and digital predistortion of power amplifiers," *IEEE Teansactions on Microwave Theory and Techniques*, vol. 68, no. 11, pp. 4835–4845, 2020. [Online]. Available: https://doi.org/10.1109/TMTT.2020.3011152

[65] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[66] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[67] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1506.02626

[68] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2017. [Online]. Available: https://arxiv.org/abs/1611.06440

[69] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1507.06149

[70] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, jan 1996. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

[71] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," 2019. [Online]. Available: https://arxiv.org/abs/1902.09574

[72] C. Zhang, C. Li, B. Guo, and N. Liao, "Neural network compression via low frequency preference," *Remote Sensing*, vol. 15, no. 12, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/12/3144

[73] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," 2017. [Online]. Available: https://arxiv.org/abs/1708.06519

[74] W. Wen, C. Wu, Y. Wang, and Y. Chen, "Learning structured sparsity in deep neural networks," p. 10, 08 2016.

[75] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020. [Online]. Available: https://arxiv.org/abs/2006.05467

[76] D. Kunin, J. Sagastuy-Brena, S. Ganguli, D. L. K. Yamins, and H. Tanaka, "Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics," 2021. [Online]. Available: https://arxiv.org/abs/2012.04728

[77] "Generalization in neural networks." [Online]. Available: https://arxiv.org/abs/1806.07572

[78] I. Jolliffe, *Principal component analysis (2nd edition)*. Berlin: Springer Verlag, 2002.

[79] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[80] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014. [Online]. Available: https://arxiv.org/abs/1405.3866

[81] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu, "Recent advances in efficient computation of deep convolutional neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1802.00939

[82] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017. [Online]. Available: https://arxiv.org/abs/1712.05877

[83] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018. [Online]. Available: https://arxiv.org/abs/1806.08342

[84] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," 2019. [Online]. Available: https://arxiv.org/abs/1906.04721

[85] R. Banner, Y. Nahshan, E. Hoffer, and D. Soudry, "Post-training 4-bit quantization of convolution networks for rapid-deployment," 2019. [Online]. Available: https://arxiv.org/abs/1810.05723

[86] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort, "Up or down? adaptive rounding for post-training quantization," 2020. [Online]. Available: https://arxiv.org/abs/2004.10568

[87] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *ArXiv*, vol. abs/1308.3432, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:18406556

[88] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[89] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," 2016. [Online]. Available: https://arxiv.org/abs/1602.01528

[90] B. Xu, A. Banerjee, and S. Gupta, "Hardware acceleration for neural networks: A comprehensive survey," 2026. [Online]. Available: https://arxiv.org/abs/2512.23914

[91] B. Khurshid, "High-performance cordic-based approximate mac architectures for fpga platforms," *Integration*, vol. 101, p. 102338, 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926024002025

[92] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, 2015, pp. 161–170.

[93] Y. Chen, T. Chen, Z. Xu *et al.*, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2016.

[94] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[95] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[96] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer, "Efficient processing of deep neural networks: A tutorial and survey," 2017. [Online]. Available: https://arxiv.org/abs/1703.09039

[97] H. Nair, P. Vellaisamy, T.-H. Lin, P. Wang, S. Blanton, and J. P. Shen, "Ozmac: An energy-efficient sparsity-exploiting multiply-accumulate-unit design for dl inference," 2024.

[98] J. Cong and J. Xu, "Simultaneous FPGA resource binding and placement with delay and power optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 1, pp. 45–58, 2010.

[99] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[100] N. Koilia and C. Kachris, "Hardware acceleration of llms: A comprehensive survey and comparison," 2024. [Online]. Available: https://arxiv.org/abs/2409.03384

[101] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2019. [Online]. Available: https://arxiv.org/abs/1810.05270

[102] T. Mohaidat and K. Khalil, "A survey on neural network hardware accelerators," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 8, pp. 3801–3822, 2024.

[103] "Power amplifier characterization," 2026, (Accessed 2026-07-10). [Online]. Available: https://se.mathworks.com/help/simrf/ug/power-amplifier-characterization.html

[104] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015, pp. 1135–1143.

Appendix

A Images for legacy solution results

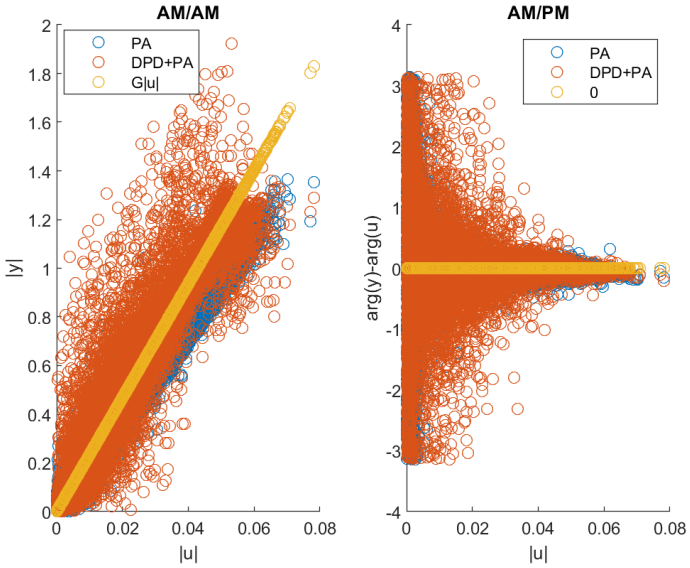


Figure A.1: Magnification of subfigure 5.1a, which visualizes the AM/AM and AM/PM distortions for the combined legacy DFE model consisting of only DPD and MP6 PA model, and the ideal PA model applied to the evaluation signal.

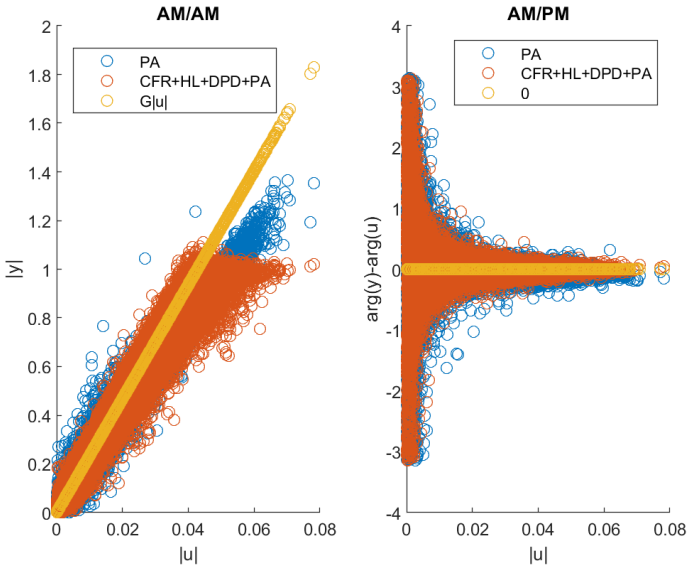


Figure A.2: Magnification of subfigure 5.1b, which visualizes the AM/AM and AM/PM distortions for the combined legacy DFE and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 1 clip stage, HL and DPD, and the PAPR threshold was 6.5 dB.

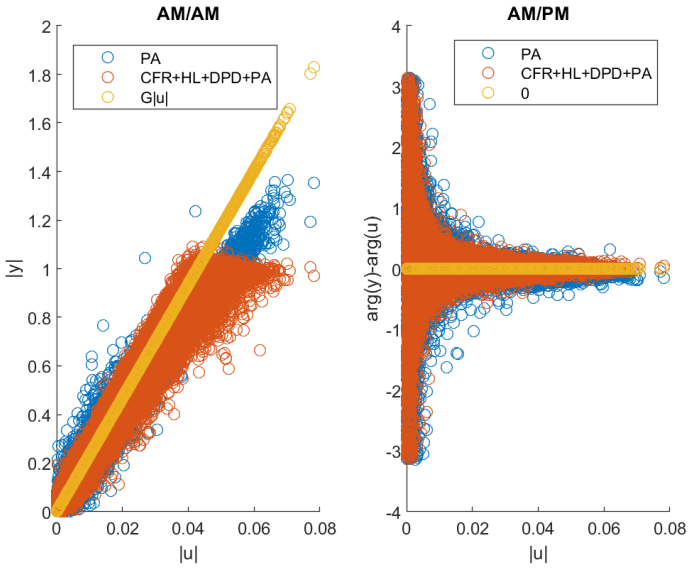


Figure A.3: Magnification of subfigure 5.1c, which visualizes the AM/AM and AM/PM distortions for the combined legacy DFE and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 2 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

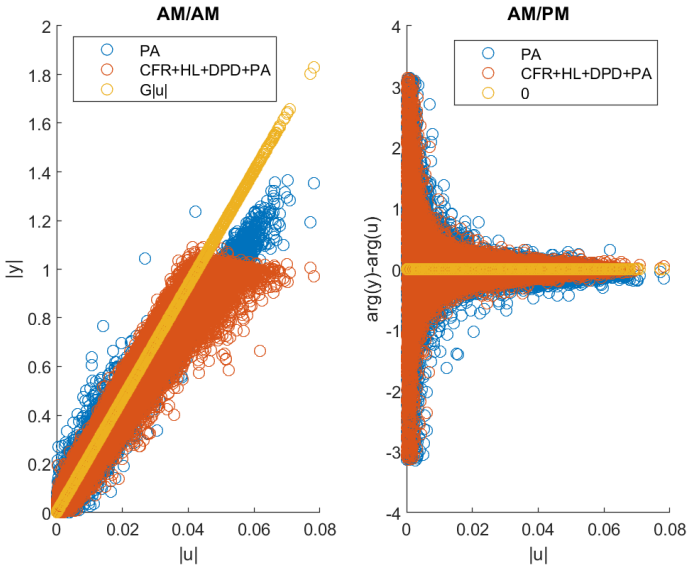


Figure A.4: Magnification of subfigure 5.1d, which visualizes the AM/AM and AM/PM distortions for the combined legacy DFE and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 3 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

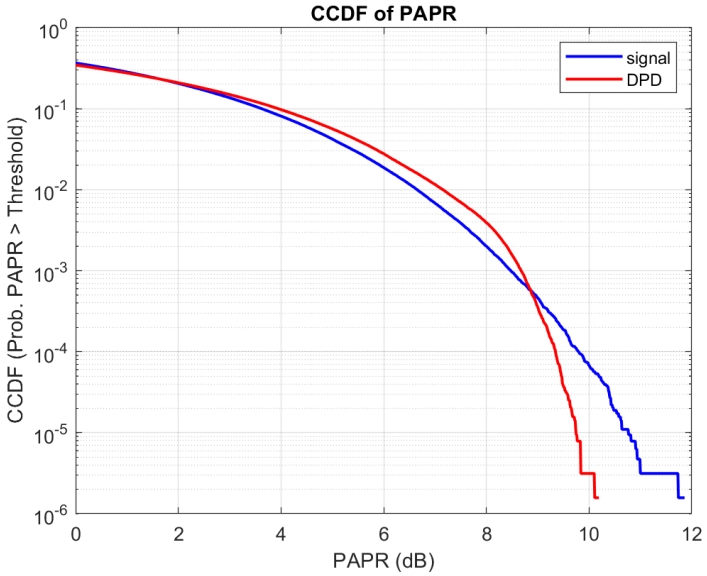


Figure A.5: Magnification of subfigure 5.2a, which visualizes the CCDF curve for the legacy DFE model consisting of only DPD applied to the evaluation signal.

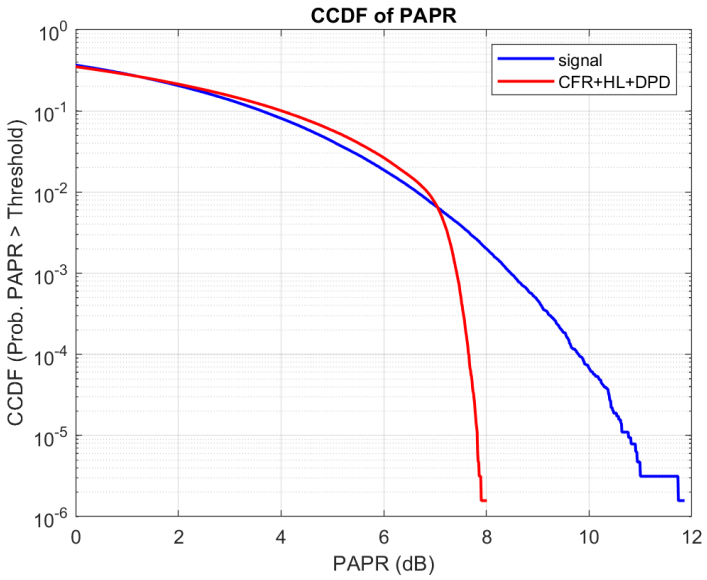


Figure A.6: Magnification of subfigure 5.2b, which visualizes the CCDF curve for the legacy DFE model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 1 clip stage, HL and DPD, and the PAPR threshold was 6.5 dB.

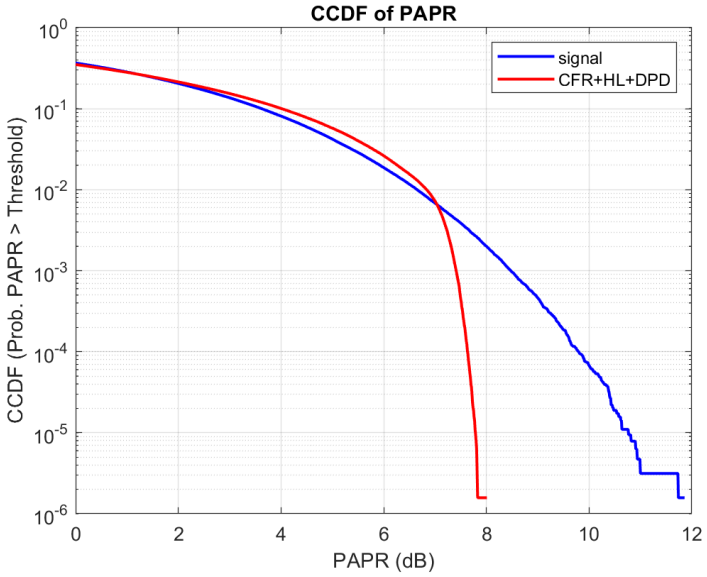


Figure A.7: Magnification of subfigure 5.2c, which visualizes the CCDF curve for the legacy DFE model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 2 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

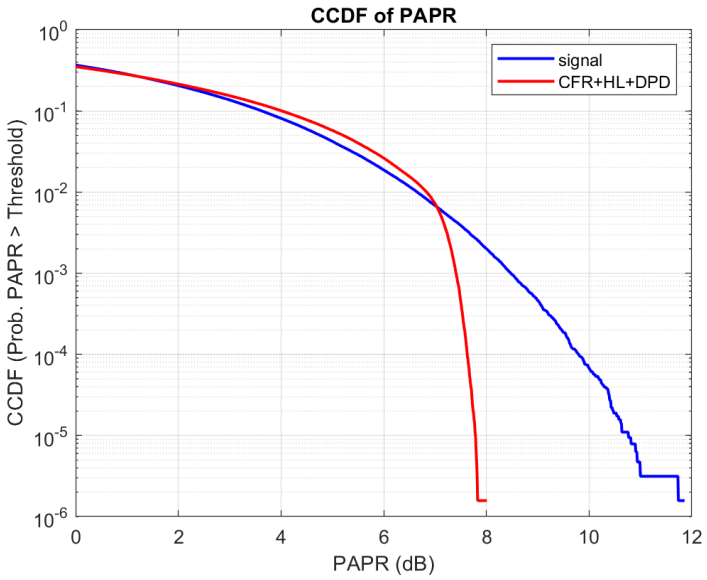


Figure A.8: Magnification of subfigure 5.2d, which visualizes the CCDF curve for the legacy DFE model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 3 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

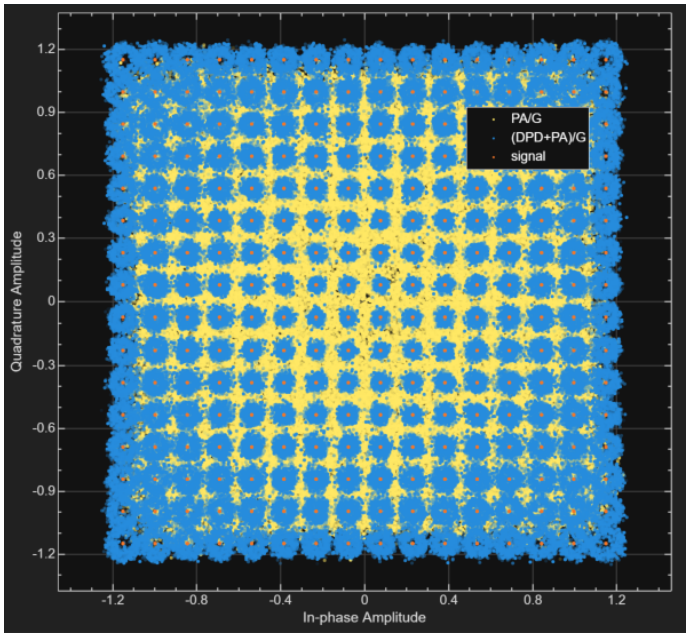


Figure A.9: Magnification of subfigure 5.3a, which visualizes the constellation diagrams for the MP6 PA model, the combined legacy DFE model consisting of only DPD and MP6 PA model, and the ideal PA model applied to the evaluation signal.

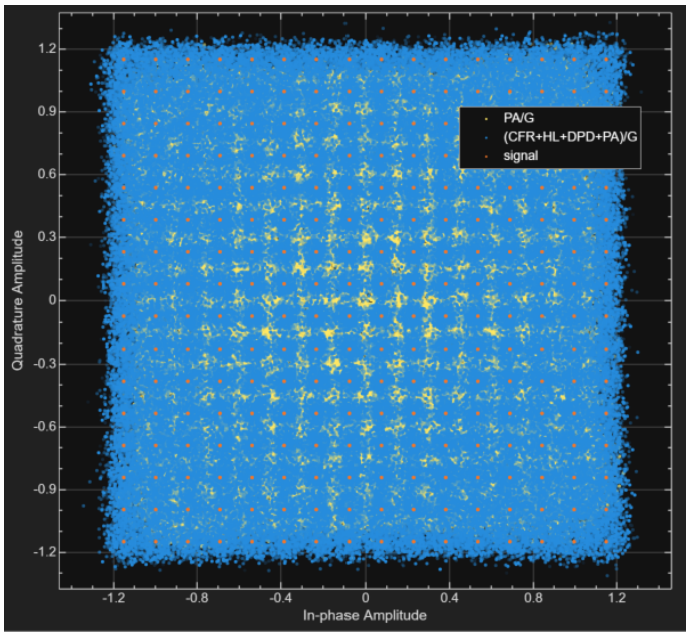


Figure A.10: Magnification of subfigure 5.3b, which visualizes the constellation diagrams for the MP6 PA model, the combined legacy DFE model and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 1 clip stage, HL and DPD, and the PAPR threshold was 6.5 dB.

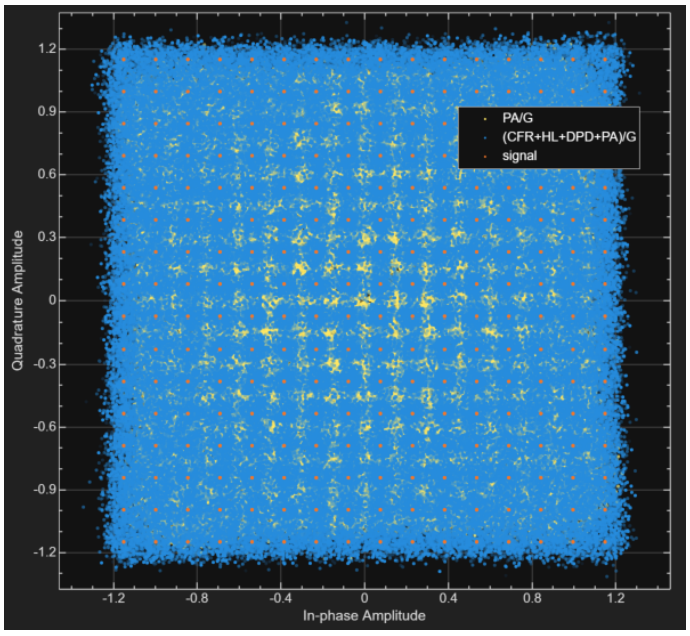


Figure A.11: Magnification of subfigure 5.3c, which visualizes the constellation diagrams for the MP6 PA model, the combined legacy DFE model and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 2 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

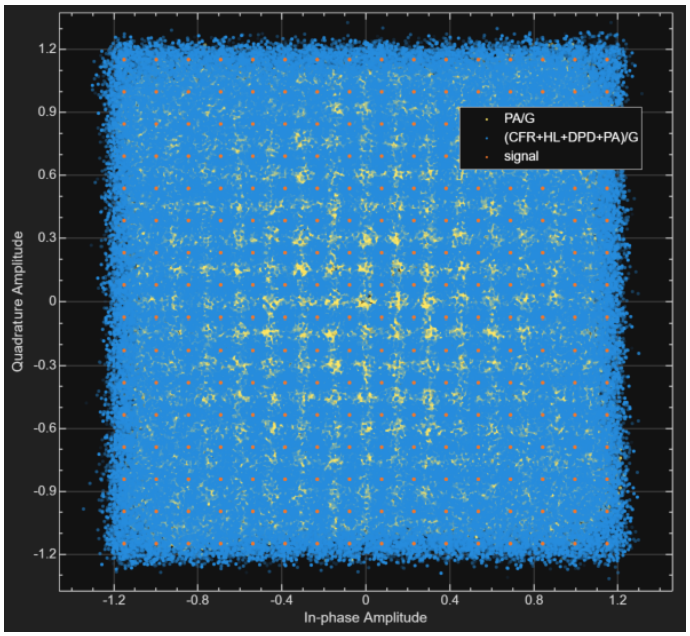


Figure A.12: Magnification of subfigure 5.3d, which visualizes the constellation diagrams for the MP6 PA model, the combined legacy DFE model and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 3 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

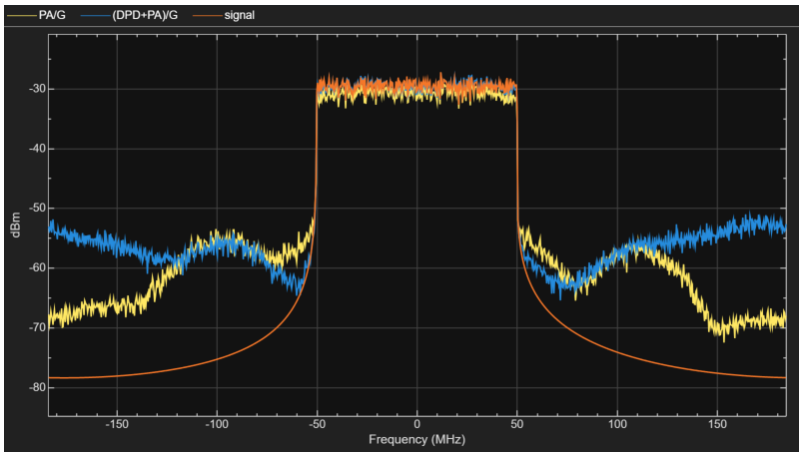


Figure A.13: Magnification of subfigure 5.4a, which visualizes the spectrums for the MP6 PA model, the combined legacy DFE model consistion of only DPD and MP6 PA model, and the ideal PA model applied to the evaluation signal.

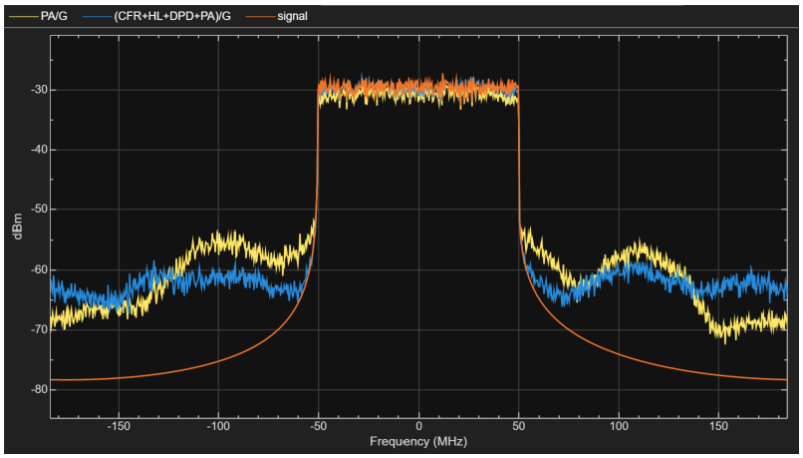


Figure A.14: Magnification of subfigure 5.4b, which visualizes the spectrums for the MP6 PA model, the combined legacy DFE models and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 1 clip stage, HL and DPD, and the PAPR threshold was 6.5 dB.

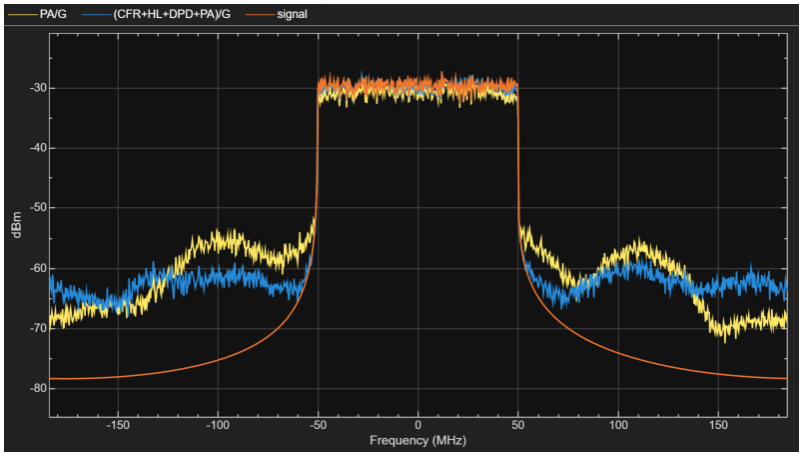


Figure A.15: Magnification of subfigure 5.4c, which visualizes the spectrums for the MP6 PA model, the combined legacy DFE models and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 2 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

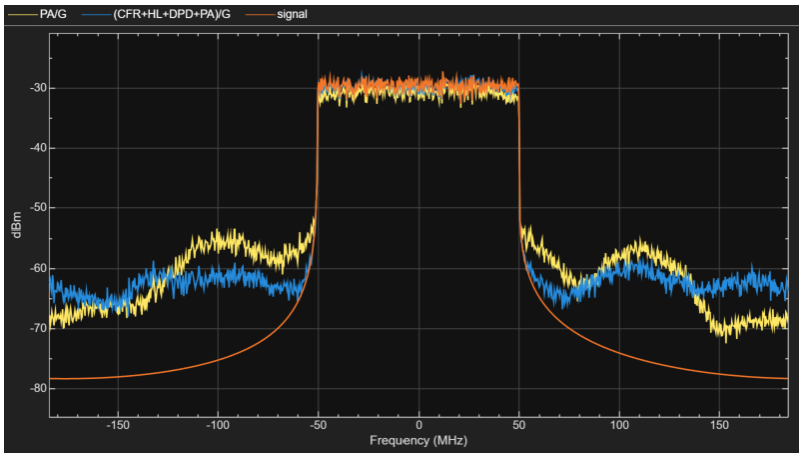


Figure A.16: Magnification of subfigure 5.4d, which visualizes the spectrums for the MP6 PA model, the combined legacy DFE models and MP6 PA model, and the ideal PA model applied to the evaluation signal. Here the legacy DFE model consisted of CFR with 3 clip stages, HL and DPD, and the PAPR threshold was 6.5 dB.

B Images for NN DPD solution results

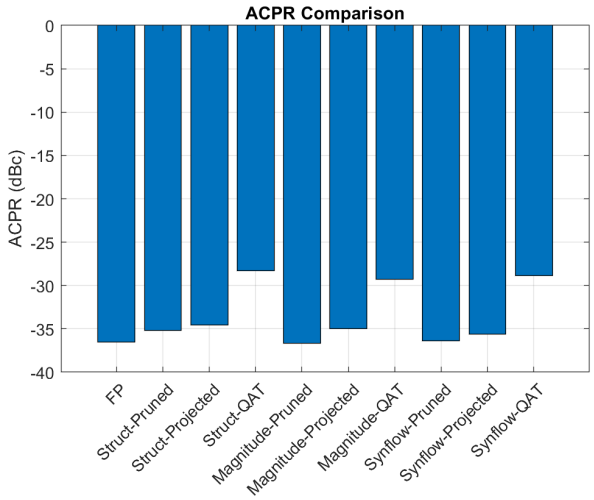


Figure B.1: Magnification of subfigure 5.5a, which visualizes the ACPR performance of the compressed NN DPD models with memory depth $M = 6$.

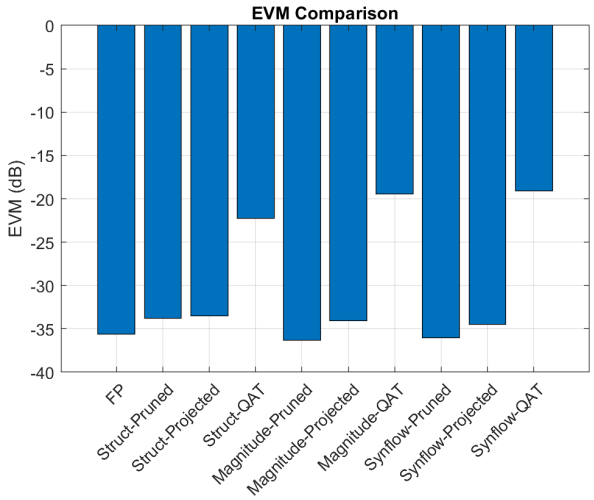


Figure B.2: Magnification of subfigure 5.5b, which visualizes the EVM performance of the compressed NN DPD models with memory depth $M = 6$.

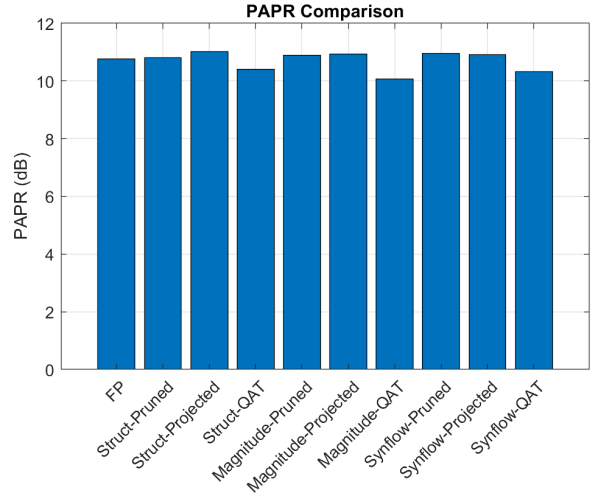


Figure B.3: Magnification of subfigure 5.5c, which visualizes the PAPR performance of the compressed NN DPD models with memory depth $M = 6$.

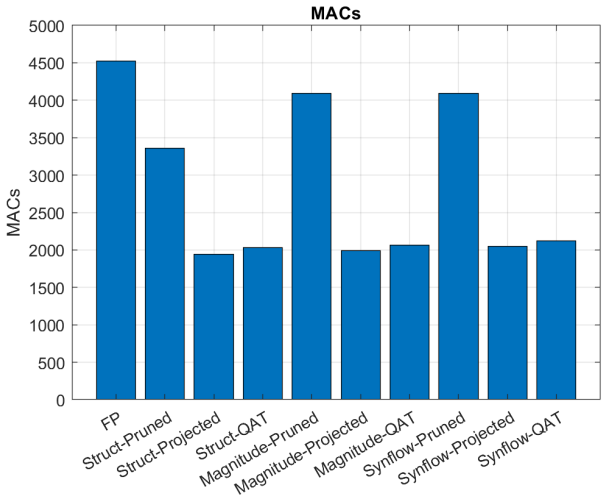


Figure B.4: Magnification of subfigure 5.6a, which visualizes the MAC count for the compressed NN DPD models with memory depth $M = 6$.

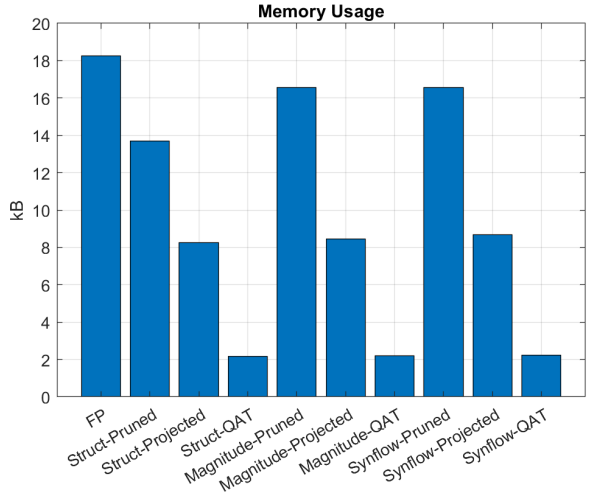


Figure B.5: Magnification of subfigure 5.6b, which visualizes the total memory for the compressed NN DPD models with memory depth $M = 6$.

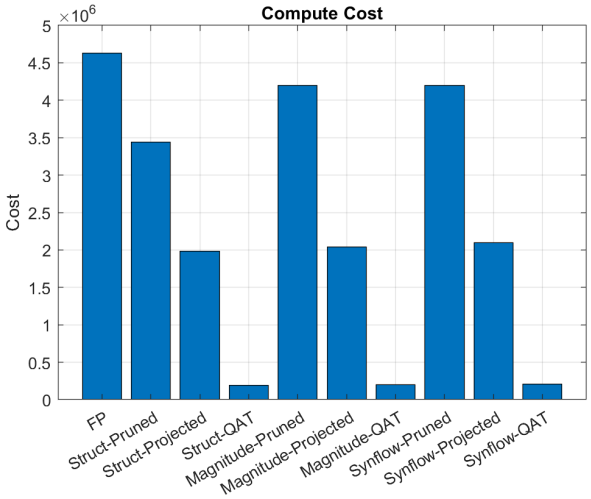


Figure B.6: Magnification of subfigure 5.6c, which visualizes the compute cost for the compressed NN DPD models with memory depth $M = 6$.

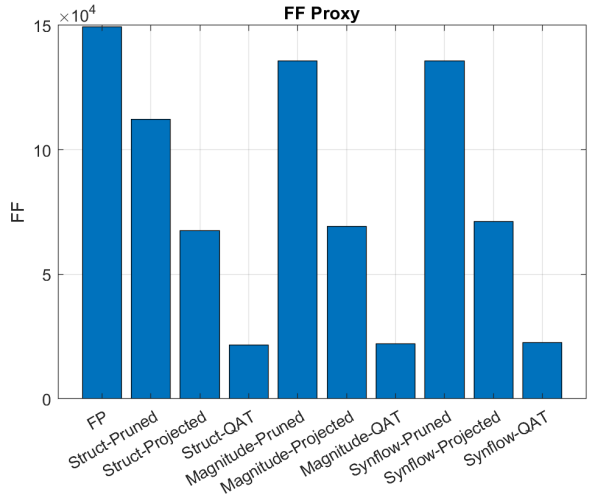


Figure B.7: Magnification of subfigure 5.6d, which visualizes the FFp for the compressed NN DPD models with memory depth $M = 6$.

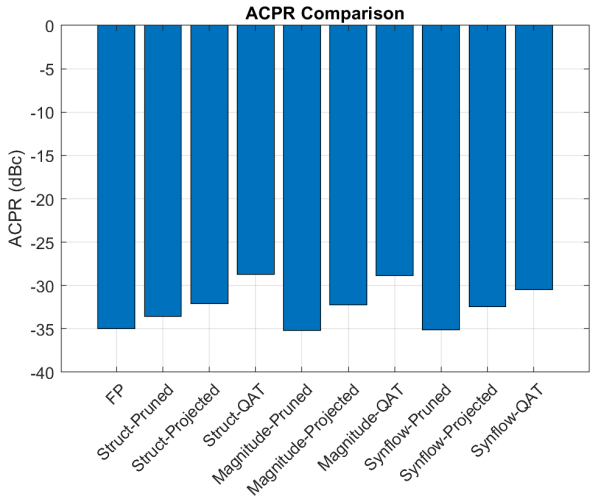


Figure B.8: Magnification of subfigure 5.7a, which visualizes the ACPR performance of the compressed NN DPD models with memory depth $M = 3$.

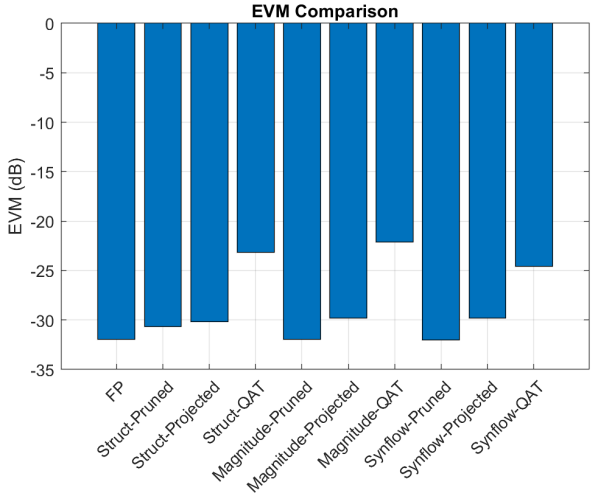


Figure B.9: Magnification of subfigure 5.7b, which visualizes the EVM performance of the compressed NN DPD models with memory depth $M = 3$.

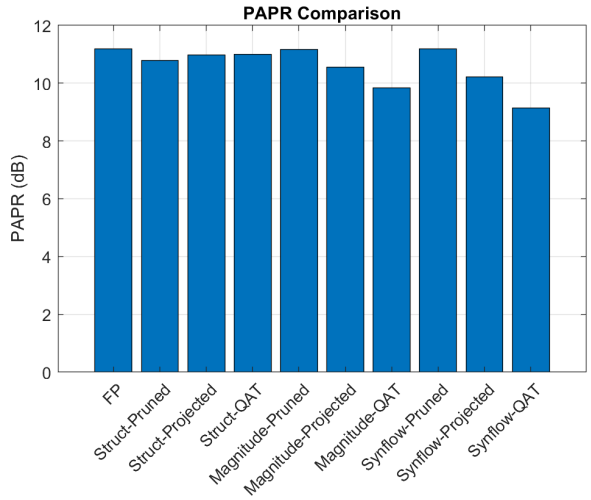


Figure B.10: Magnification of subfigure 5.7c, which visualizes the PAPR performance of the compressed NN DPD models with memory depth $M = 3$.

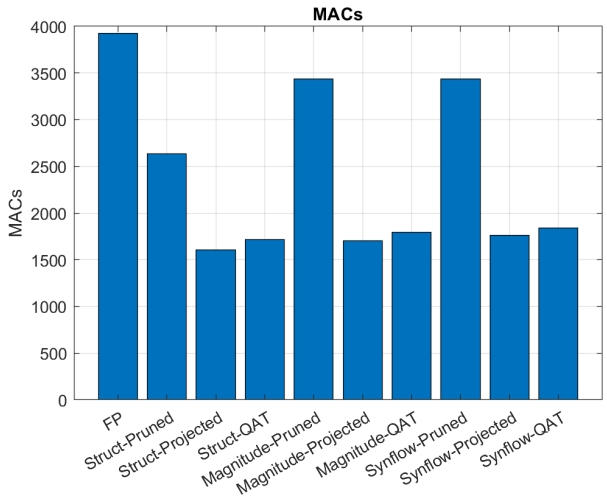


Figure B.11: Magnification of subfigure 5.8a, which visualizes the MAC count for the compressed NN DPD models with memory depth $M = 3$.

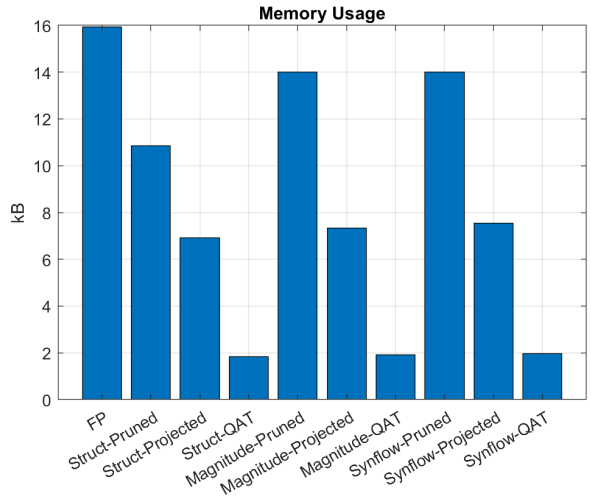


Figure B.12: Magnification of subfigure 5.8b, which visualizes the total memory for the compressed NN DPD models with memory depth $M = 3$.

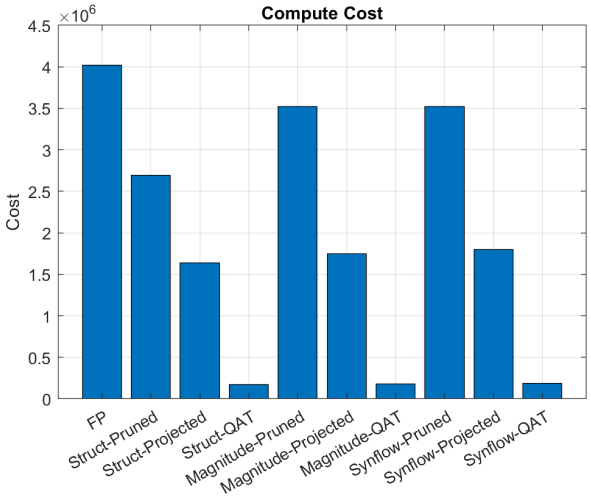


Figure B.13: Magnification of subfigure 5.8c, which visualizes the compute cost for the compressed NN DPD models with memory depth $M = 3$.

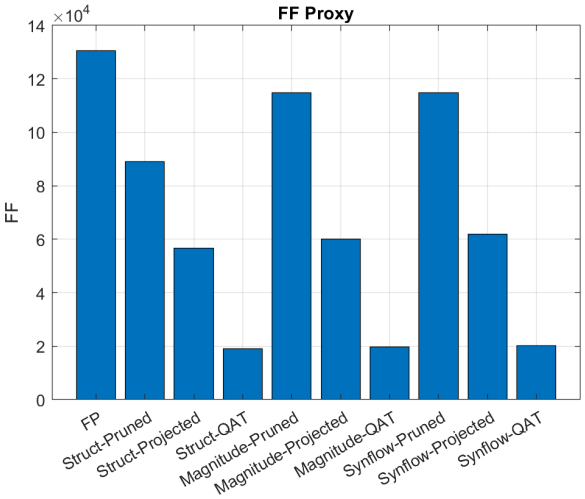


Figure B.14: Magnification of subfigure 5.8d, which visualizes the FFp for the compressed NN DPD models with memory depth $M = 3$.

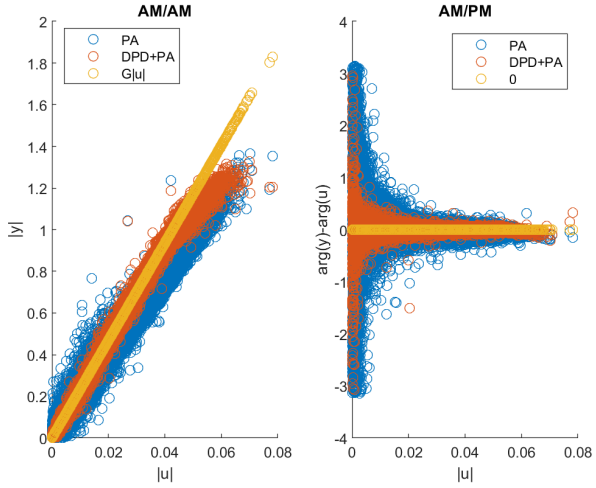


Figure B.15: Magnification of subfigure 5.13a, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 6$.

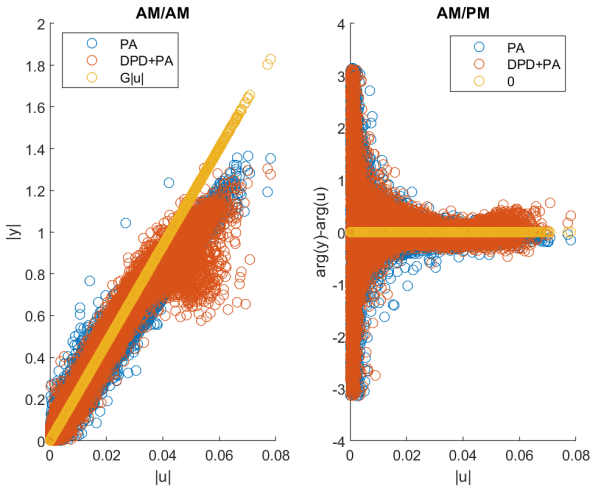


Figure B.16: Magnification of subfigure 5.13b, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$.

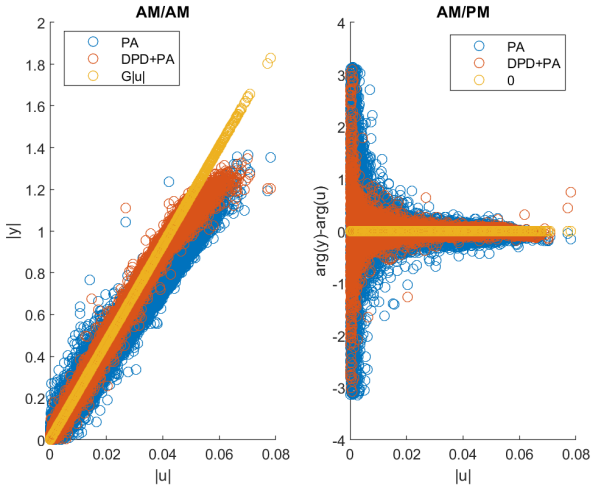


Figure B.17: Magnification of subfigure 5.13c, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 3$.

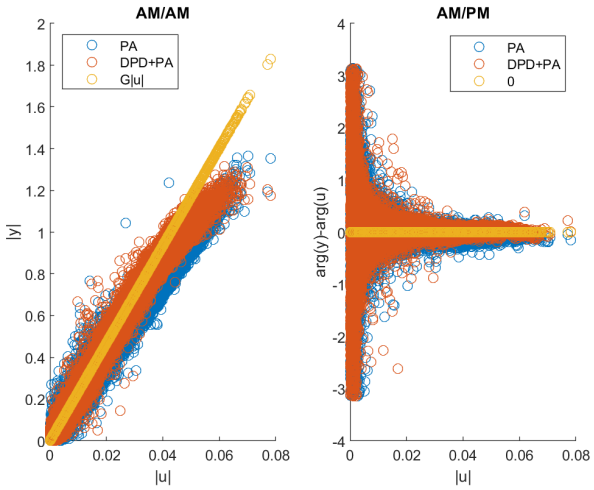


Figure B.18: Magnification of subfigure 5.13d, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$.

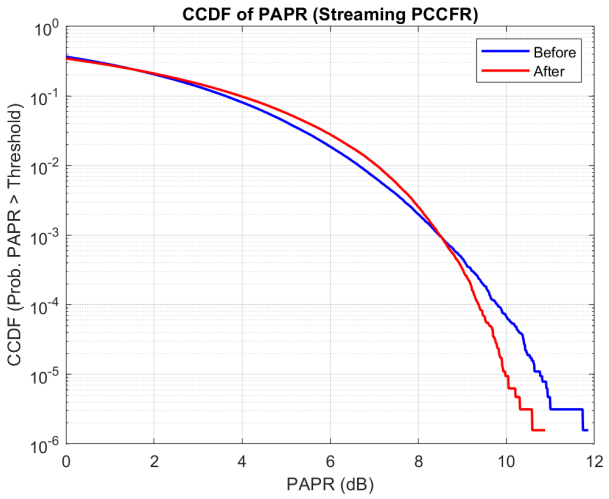


Figure B.19: Magnification of subfigure 5.14a, which visualizes the CCDF curves for the NN DPD model with the best performance for memory depth $M = 6$ applied to the evaluation signal.

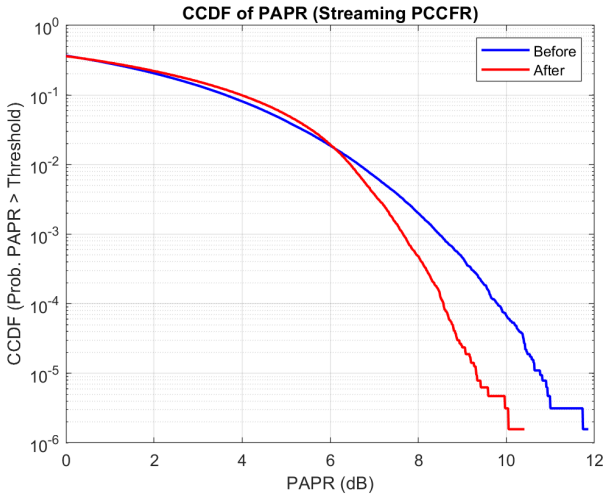


Figure B.20: Magnification of subfigure 5.14b, which visualizes the CCDF curves for the NN DPD model with the lowest estimated hardware resource utilization for memory depth $M = 6$ applied to the evaluation signal.

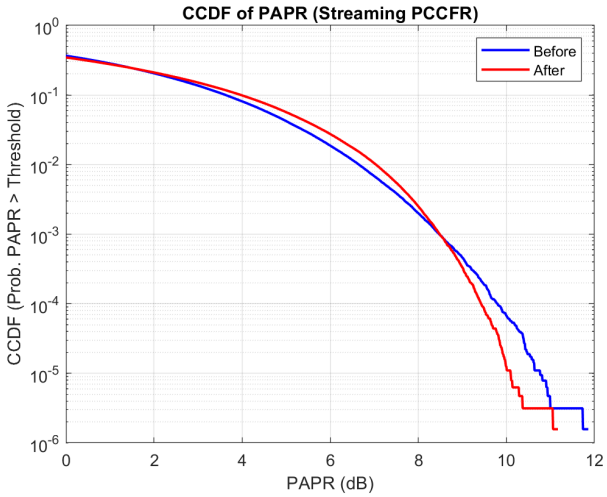


Figure B.21: Magnification of subfigure 5.14c, which visualizes the CCDF curves for the NN DPD model with the best performance for memory depth $M = 3$ applied to the evaluation signal.

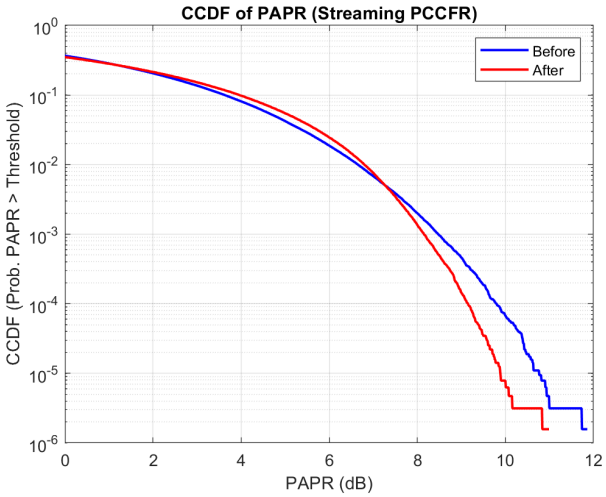


Figure B.22: Magnification of subfigure 5.14d, which visualizes the CCDF curves for the NN DPD model with the lowest estimated hardware resource utilization for memory depth $M = 3$ applied to the evaluation signal.

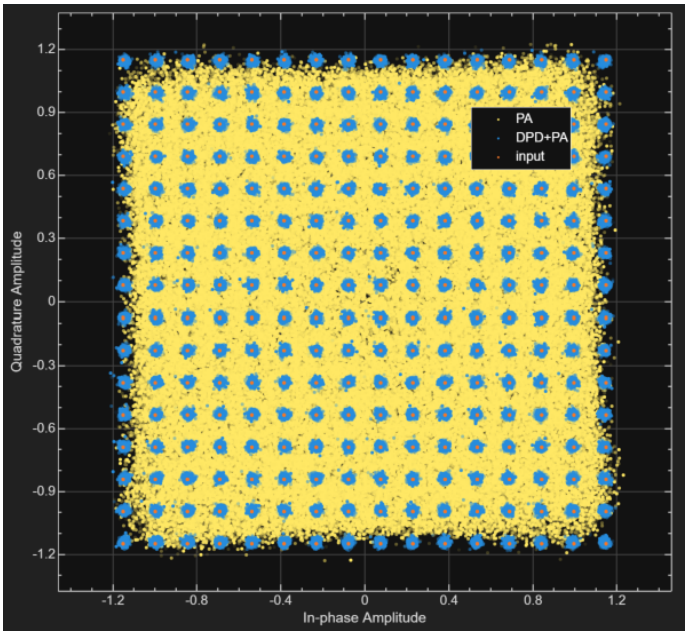


Figure B.23: Magnification of subfigure 5.15a, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 6$.

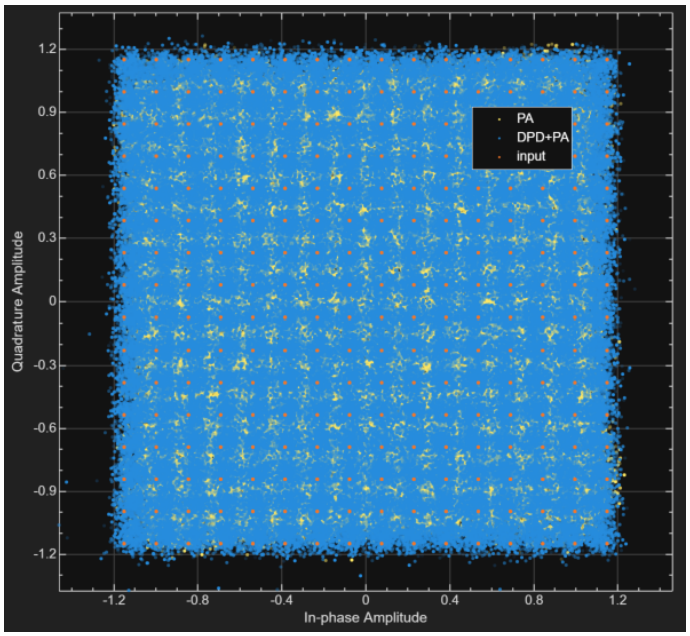


Figure B.24: Magnification of subfigure 5.15b, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$.

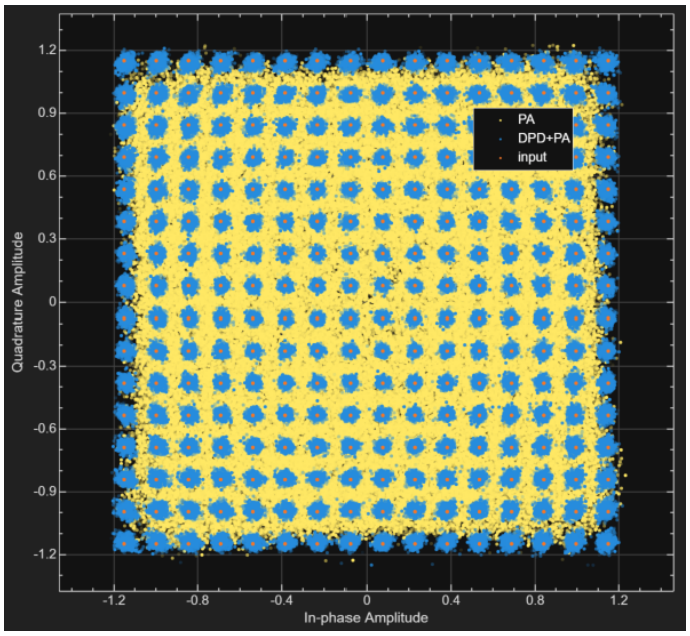


Figure B.25: Magnification of subfigure 5.15c, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 3$.

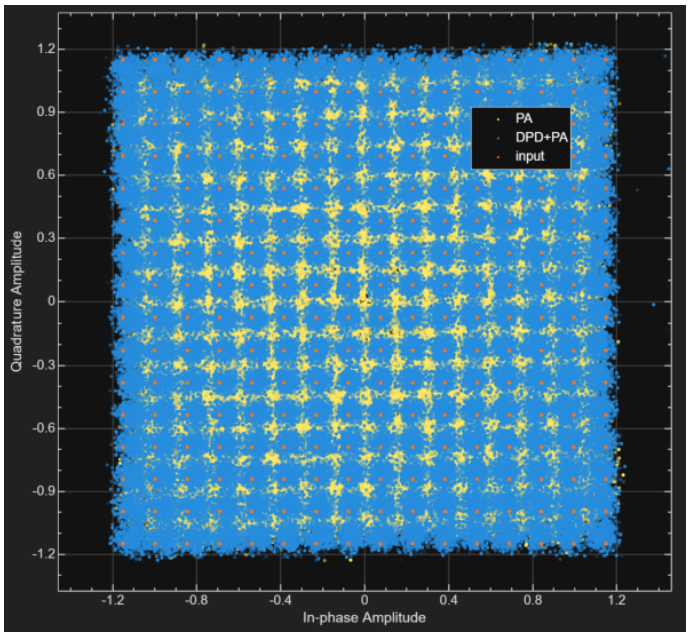


Figure B.26: Magnification of subfigure 5.15d, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$.

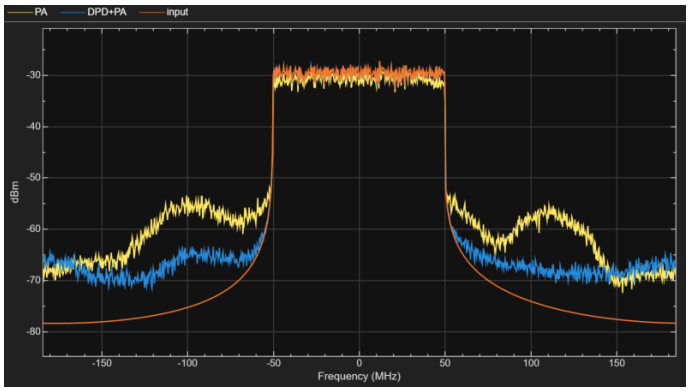


Figure B.27: Magnification of subfigure 5.16a, which visualizes the spectrums for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 6$.

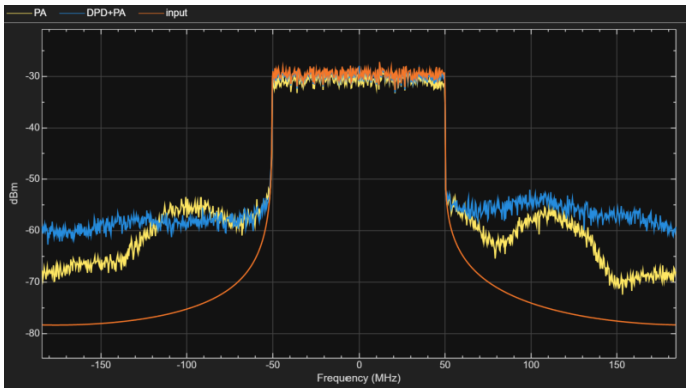


Figure B.28: Magnification of subfigure 5.16b, which visualizes the spectrums for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$.

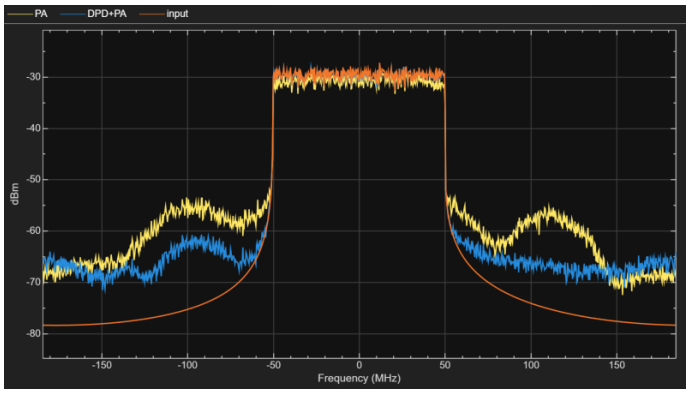


Figure B.29: Magnification of subfigure 5.16c, which visualizes the spectrums for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the best performance for memory depth $M = 3$.

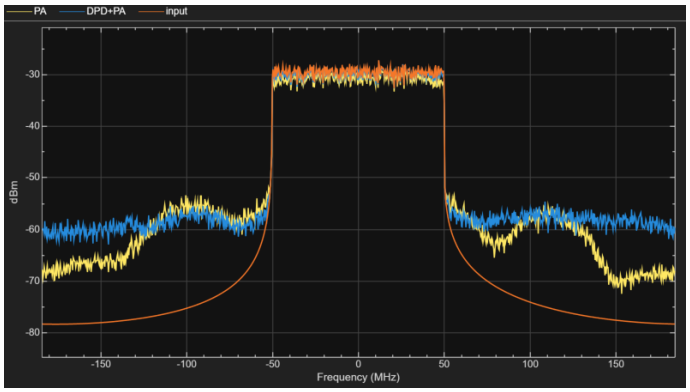


Figure B.30: Magnification of subfigure 5.16d, which visualizes the spectrums for the MP6 PA model, the combined NN DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DPD here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$.

C Images for co-optimized NN DFE solution results

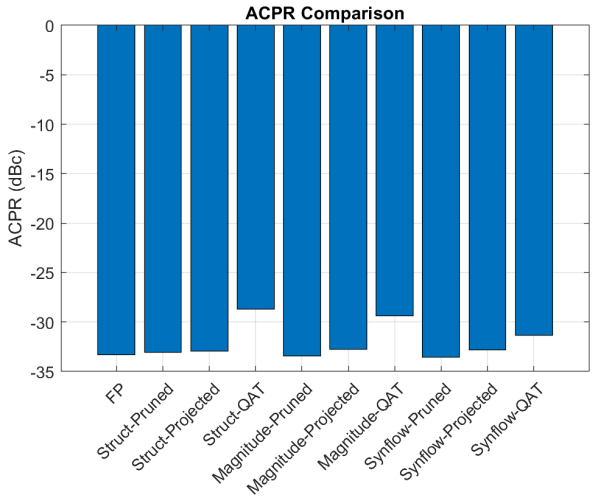


Figure C.1: Magnification of subfigure 5.9a, which visualizes the ACPR performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

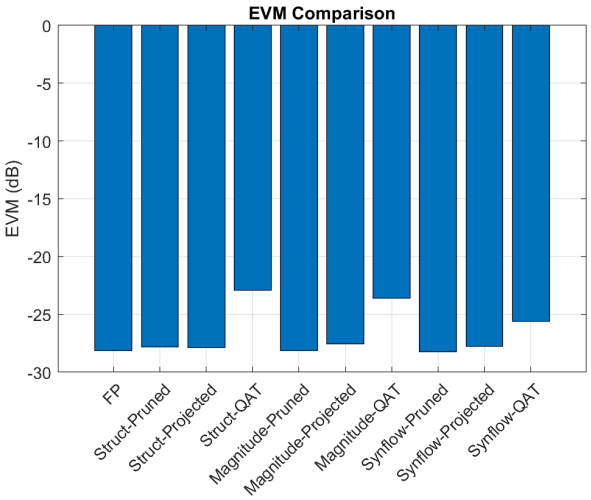


Figure C.2: Magnification of subfigure 5.9b, which visualizes the EVM performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

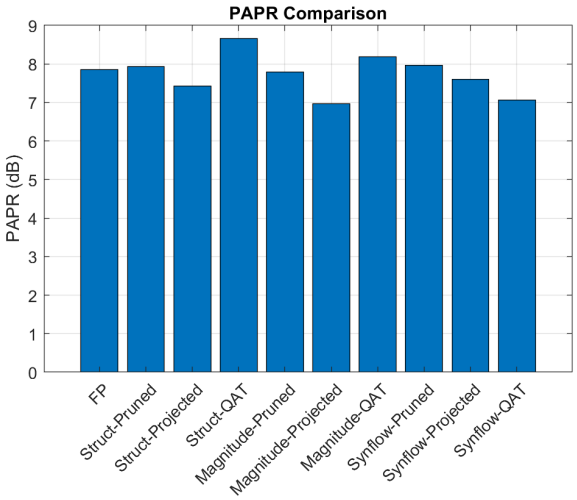


Figure C.3: Magnification of subfigure 5.9c, which visualizes the PAPR performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

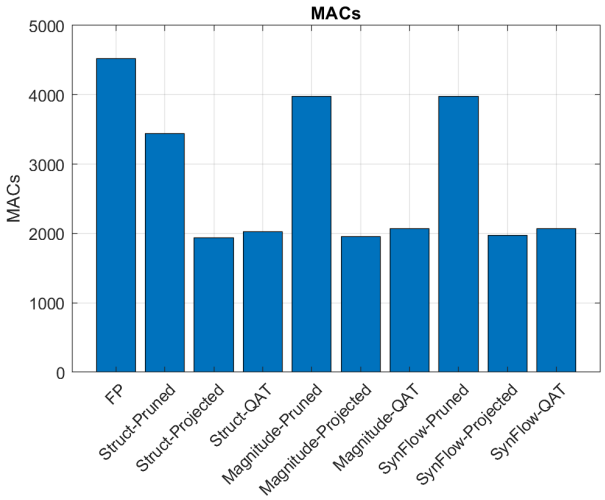


Figure C.4: Magnification of subfigure 5.10a, which visualizes the MAC count for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

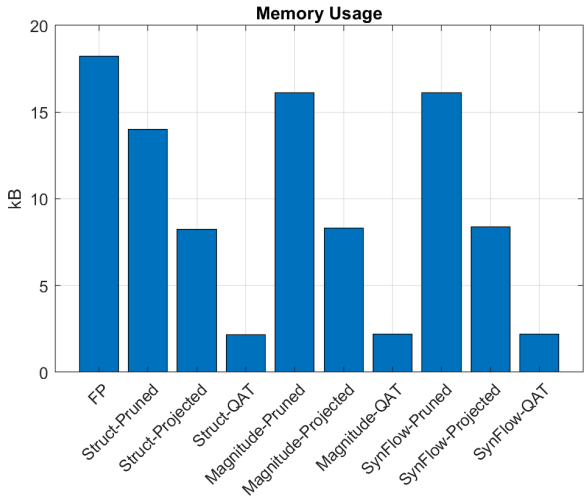


Figure C.5: Magnification of subfigure 5.10b, which visualizes the total memory for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

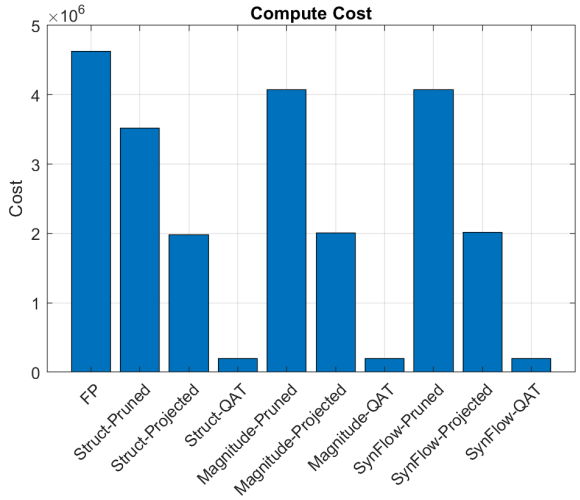


Figure C.6: Magnification of subfigure 5.10c, which visualizes the compute cost for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

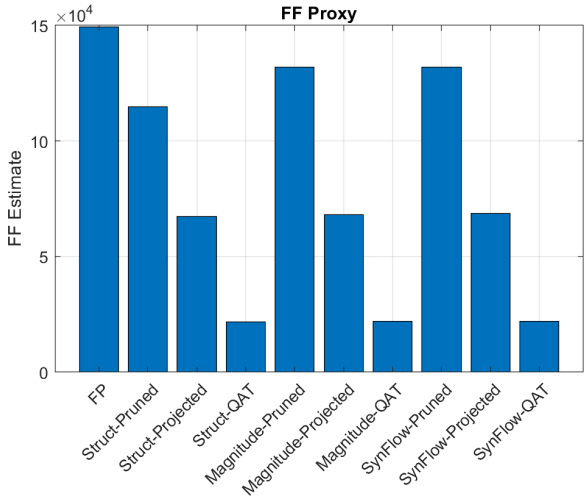


Figure C.7: Magnification of subfigure 5.10d, which visualizes the FFp for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 6$.

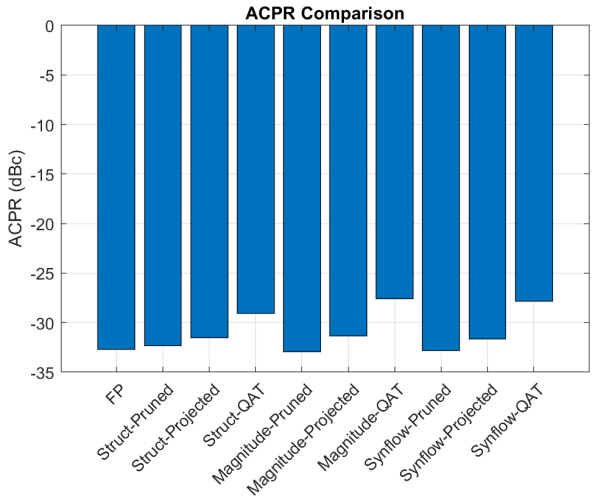


Figure C.8: Magnification of subfigure 5.11a, which visualizes the ACPR performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

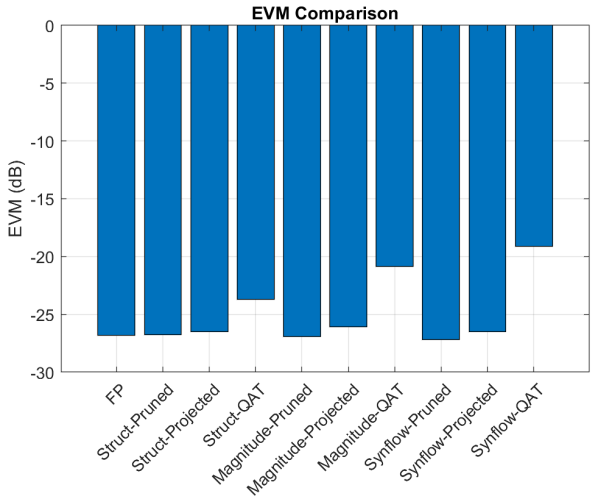


Figure C.9: Magnification of subfigure 5.11b, which visualizes the EVM performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

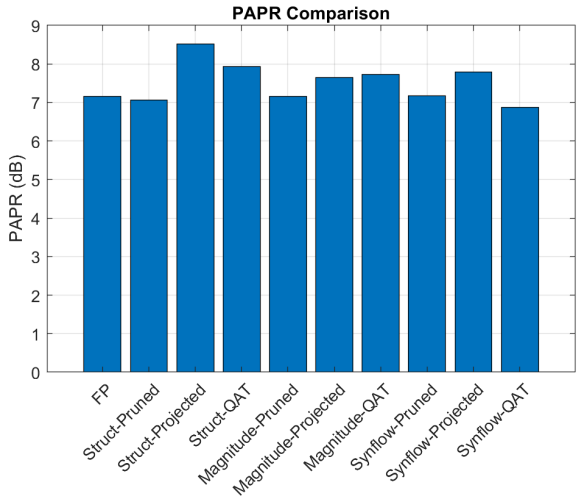


Figure C.10: Magnification of subfigure 5.11c, which visualizes the PAPR performance of the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

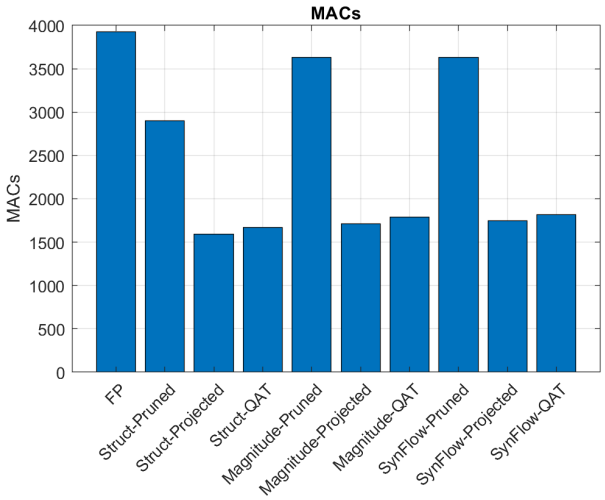


Figure C.11: Magnification of subfigure 5.12a, which visualizes the MAC count for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

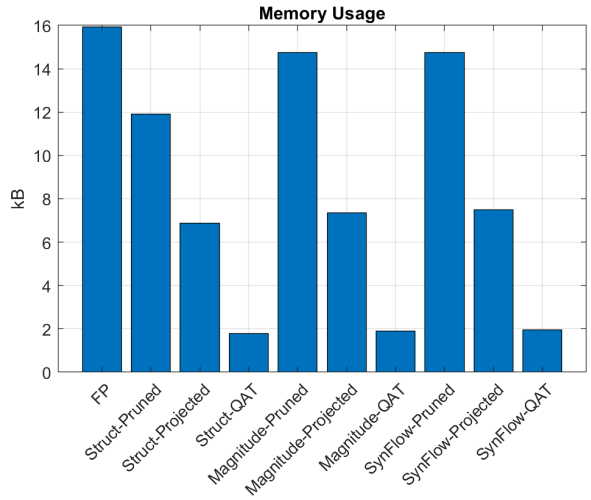


Figure C.12: Magnification of subfigure 5.12b, which visualizes the total memory for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

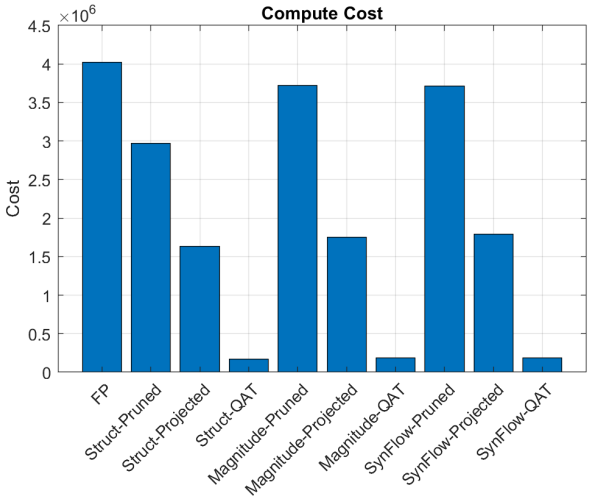


Figure C.13: Magnification of subfigure 5.12c, which visualizes the compute cost for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

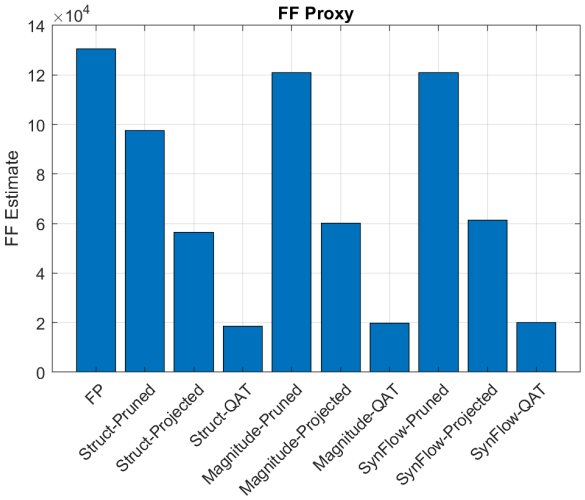


Figure C.14: Magnification of subfigure 5.12d, which visualizes the FFp for the compressed nn DFE modeling co-optimized CFR and DPD with memory depth $M = 3$.

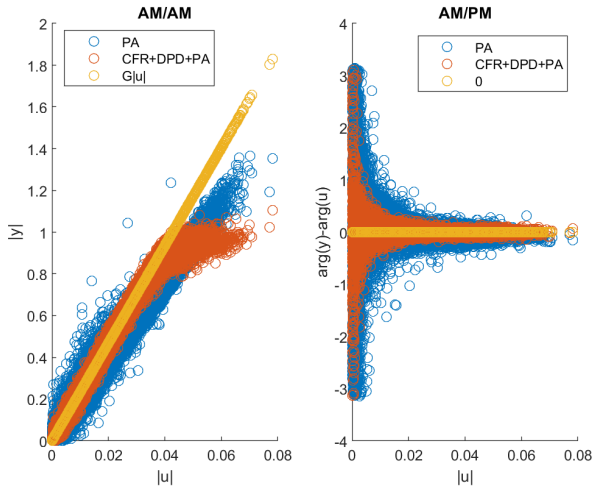


Figure C.15: Magnification of subfigure 5.17a, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the best performance for memory depth $M = 6$ and PAPR threshold 6.5 dB.

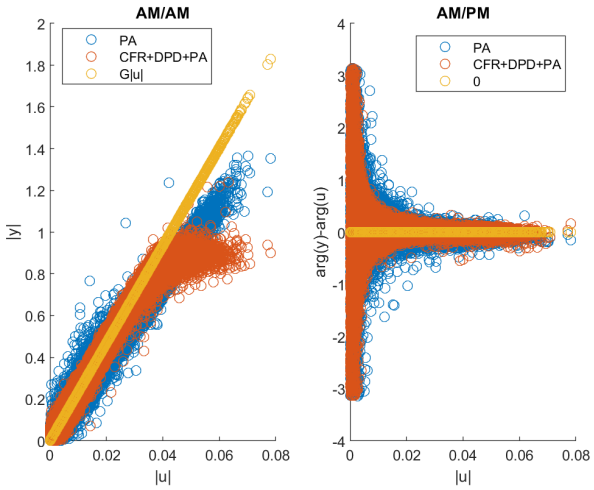


Figure C.16: Magnification of subfigure 5.17b, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$ and PAPR threshold 6.5 dB.

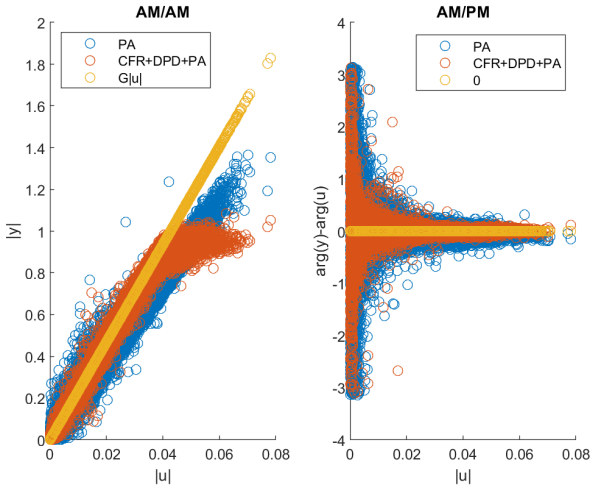


Figure C.17: Magnification of subfigure 5.17c, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFe here is the one with the best performance for memory depth $M = 3$ and PAPR threshold 6.5 dB.

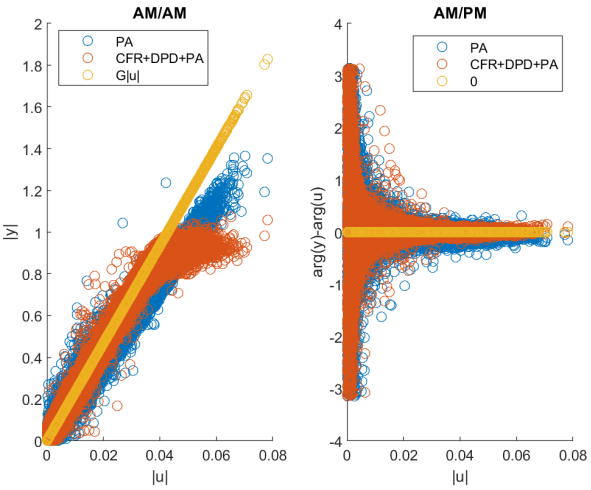


Figure C.18: Magnification of subfigure 5.17d, which visualizes the AM/AM and AM/PM distortions for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$ and PAPR threshold 6.5 dB.

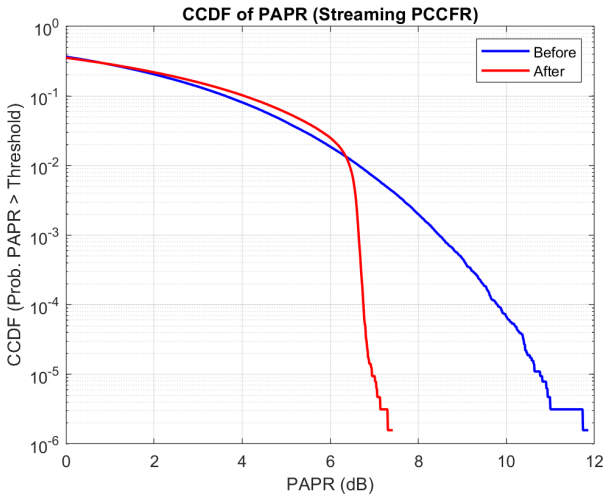


Figure C.19: Magnification of subfigure 5.18a, which visualizes the CCDF curves for the NN DFE jointly modeling CFR and DFE model with the best performance for memory depth $M = 6$ and PAPR threshold 6.5 dB applied to the evaluation signal.

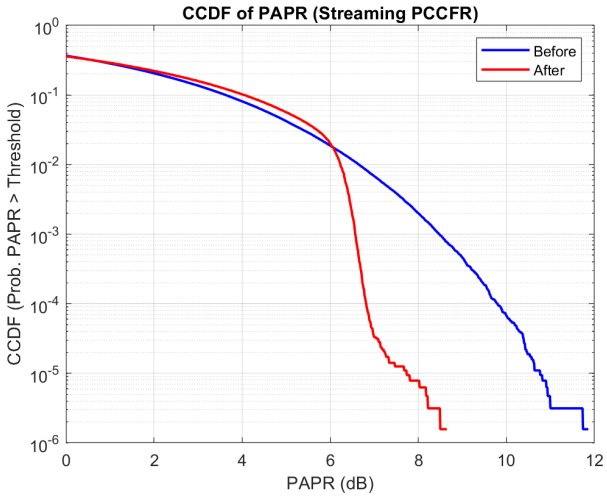


Figure C.20: Magnification of subfigure 5.18b, which visualizes the CCDF curves for the NN DFE jointly modeling CFR and DFE model with the lowest estimated hardware resource utilization for memory depth $M = 6$ and PAPR threshold 6.5 dB applied to the evaluation signal.

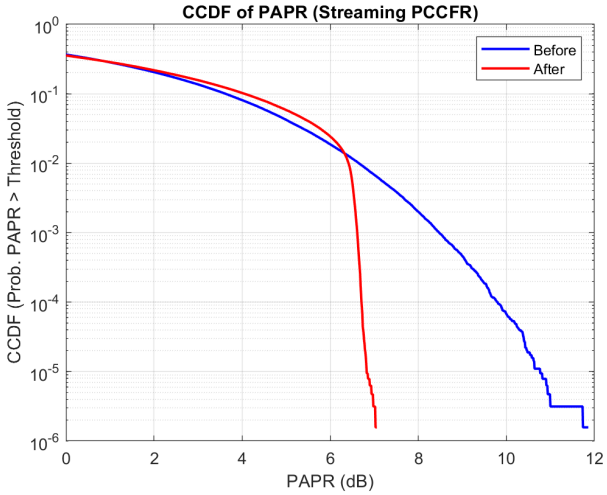


Figure C.21: Magnification of subfigure 5.18c, which visualizes the CCDF curves for the NN DFE jointly modeling CFR and DFE model with the best performance for memory depth $M = 3$ and PAPR threshold 6.5 dB applied to the evaluation signal.

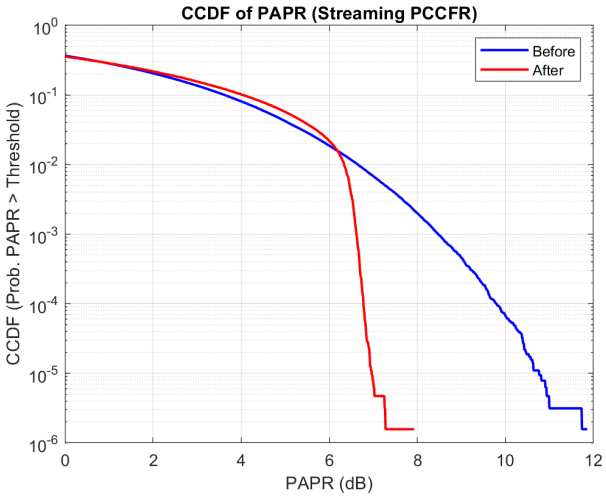


Figure C.22: Magnification of subfigure 5.18d, which visualizes the CCDF curves for the NN DFE jointly modeling CFR and DFE model with the lowest estimated hardware resource utilization for memory depth $M = 3$ and PAPR threshold 6.5 dB applied to the evaluation signal.

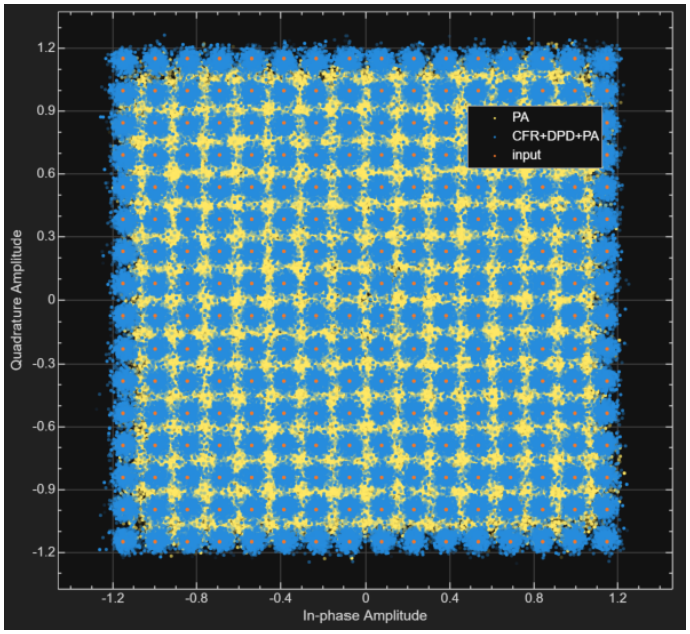


Figure C.23: Magnification of subfigure 5.19a, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the best performance for memory depth $M = 6$ and PAPR threshold 6.5 dB.

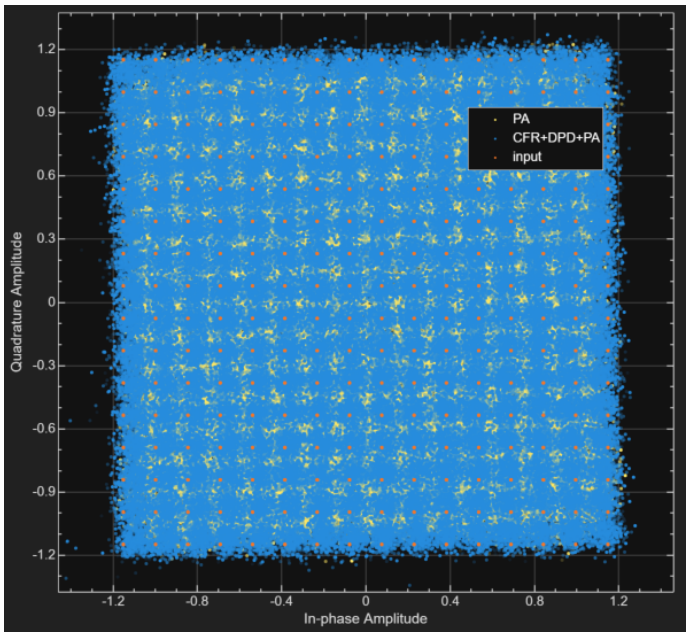


Figure C.24: Magnification of subfigure 5.19b, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$ and PAPR threshold 6.5 dB.

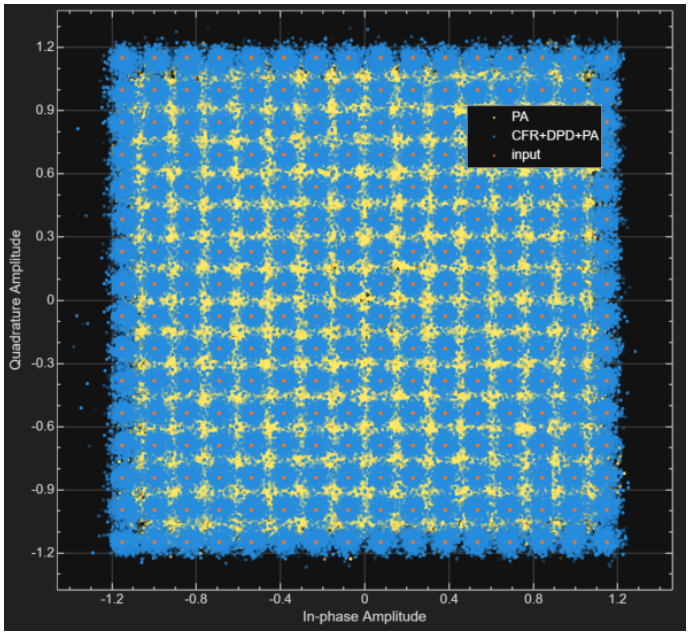


Figure C.25: Magnification of subfigure 5.19c, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the best performance for memory depth $M = 3$ and PAPR threshold 6.5 dB.

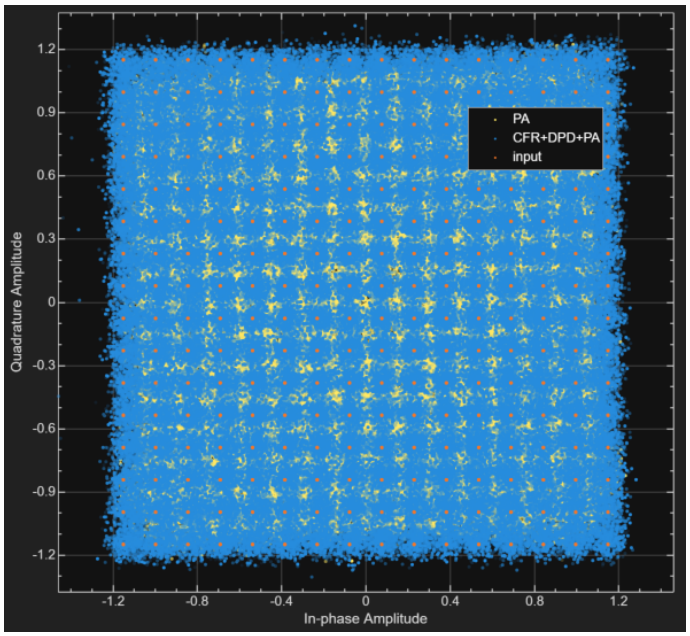


Figure C.26: Magnification of subfigure 5.19d, which visualizes the constellation diagrams for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$ and PAPR threshold 6.5 dB.

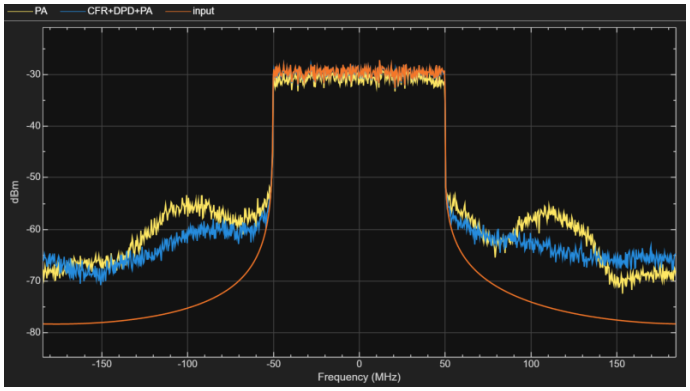


Figure C.27: Magnification of subfigure 5.20a, which visualizes the spectrums for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the best performance for memory depth $M = 6$ and PAPR threshold 6.5 dB.

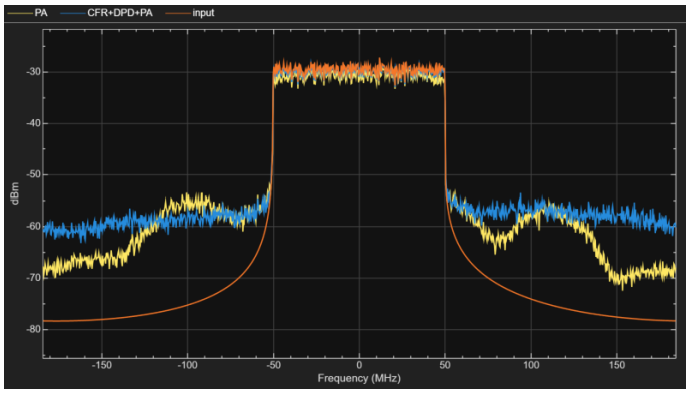


Figure C.28: Magnification of subfigure 5.20b, which visualizes the spectrums for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 6$ and PAPR threshold 6.5 dB.

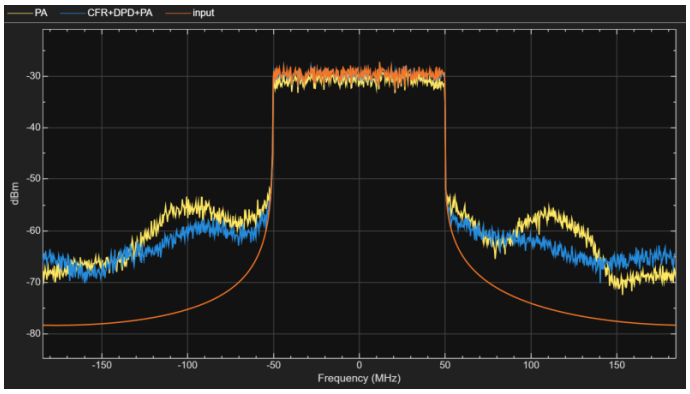


Figure C.29: Magnification of subfigure 5.20c, which visualizes the spectrums for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the best performance for memory depth $M = 3$ and PAPR threshold 6.5 dB.

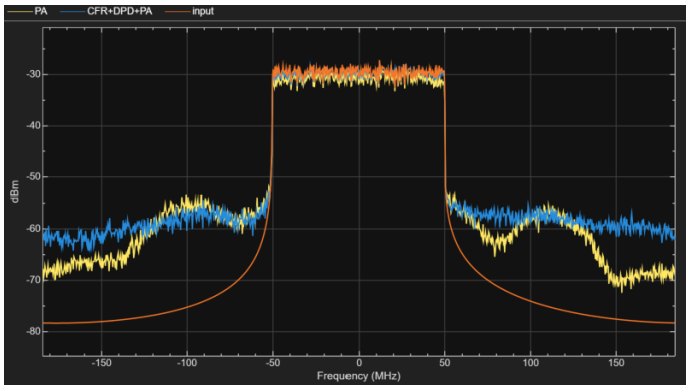


Figure C.30: Magnification of subfigure 5.20d, which visualizes the spectrums for the MP6 PA model, the combined NN DFE jointly modeling CFR and DPD model and MP6 PA model, and the ideal PA model applied to the evaluation signal. The NN DFE here is the one with the lowest estimated hardware resource utilization for memory depth $M = 3$ and PAPR threshold 6.5 dB.