


Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informaticos

Master in Data Science

Master Thesis

Transformer DPD for Non-Linear Power Amplifiers in Multi Carrier Signals

Author: **Tommaso Baroni**

Madrid, June 2024

This Master Thesis has been deposited in ETSI Informáticos de la Universidad Politécnica de Madrid.

Master Thesis
Master in Data Science

Title: Transformer DPD for Non-Linear Power Amplifiers in Multi Carrier Signals

June 2024

Author: Tommaso Baroni
Defending: Master of Science in Data Science (EIT Digital Master School)

Supervisor
Emilio Serrano
Department of Artificial Intelligence
Universidad Politécnica de Madrid

Company Supervisor
Sergio Gutiérrez
DPD IP Program Manager
Ericsson AB

Host Company
Ericsson AB
Algorithm IP Department
Lund/Kista, Sweden

Resumen

Esta tesis investiga la aplicación de una Red Neuronal Transformadora en un sistema de pre distorsión digital (DPD) para contrarrestar las no linealidades de los amplificadores de potencia en escenarios de señales multicarrier. Basándose en trabajos anteriores [1] que probaron la arquitectura Transformador en una señal de un solo portador, demostrando su superioridad sobre otros modelos de última generación, este estudio extiende la exploración a entornos multicarrier más complejos y realistas. La investigación investiga y propone mejoras y modificaciones a la arquitectura Transformer y a las características de entrada utilizadas para entrenar la red, y evalúa sus capacidades en diversas condiciones de señal. El modelo propuesto se compara contra otros modelos de Neural Network, específicamente un Long Short-Term Memory Neural Network (LSTMNN), un Bidirectional LSTM Neural Network (BiLSTMNN) y un Bidirectional Gated Recurrent Unit Neural Network (BiGRUNN), conocidos por su eficacia en la modelización de DPD. Además, el modelo ha sido evaluado junto con los modelos DPD propietarios de Ericsson. Las métricas evaluativas incluyen el normalized mean square error (NMSE) y el adjacent channel leakage ratio (ACLR). El modelo basado en Transformer muestra un rendimiento superior cuando se compara con las otras redes neuronales, y también un rendimiento comparable al de los DPD de Ericsson. Sin embargo, la complejidad del modelo sigue siendo demasiado alta para la implementación práctica en FPGA. Este aspecto se identifica como un área significativa para futuras investigaciones y optimización, aunque queda fuera del alcance de esta tesis.

Abstract

This thesis investigates the application of the Transformer Neural Network for digital predistortion (DPD) to counteract the nonlinearities of power amplifiers in multi-carrier signal scenarios. Building upon previous work [1] that tested the Transformer architecture on a single-carrier signal, demonstrating its superiority over other state-of-the-art models, this study extends the exploration to more complex and realistic multi-carrier environments. The research investigates and proposes improvements and modifications to the Transformer architecture and to the input features used to train the network, and assesses its capabilities across diverse signal conditions. The proposed model is compared against other neural network models, specifically a Long Short-Term Memory Neural Network (LSTMNN), a Bidirectional LSTM Neural Network (BiLSTMNN) and a Bidirectional Gated Recurrent Unit Neural Network (BiGRUNN), known for their effectiveness in DPD modeling. Additionally, the model has been evaluated alongside Ericsson's proprietary DPD models. Evaluative metrics include normalized mean square error (NMSE) and adjacent channel leakage ratio (ACLR). The Transformer-based model shows superior performance when compared to the other neural networks, and also comparable performance to Ericsson's DPD. However, the complexity of the model remains too high for practical FPGA implementation. This aspect is identified as a significant area for future research and optimization, although it falls outside the scope of this thesis.

Acknowledgement

I would like to express my sincere gratitude to all of my colleagues at Ericsson for their support in the process of completing this thesis. A special thanks to my supervisor, Sergio Gutiérrez, for his constant interest and invaluable insights which guided me during this work. I am also grateful to Mohammed Almoneer, whose assistance was essential at many stages of the project. I extend my thanks also to my Line Manager, Milan Stamenkovic, not only for his professional guidance but also for his personal support during my stay at Ericsson. Last but not least, I want to thank my family and friends, whose support made this achievement possible.

Contents

Chapter 1

Introduction

As telecommunication networks evolve, transitioning from 5G toward 6G, a critical challenge that must be addressed is ensuring that power amplifiers (PAs) meet the increasingly stringent standards set by the 3rd Generation Partnership Project (3GPP). PAs are devices responsible to amplify radio frequency signals to the necessary power levels for long-distance transmission. However, the nonlinear behavior of PAs introduces several complications, as it can lead to distortions that degrade signal quality and result in spectral regrowth, causing interference with adjacent channels.

To address this issue, digital pre-distortion (DPD) techniques have emerged as a solution for linearizing the PA's response. DPD works by applying a pre-distortion function to the input signal, effectively compensating for the nonlinear distortions. The accuracy and effectiveness of DPD rely on the quality of the model used to inverse the PA's behavior, which has traditionally been achieved through polynomial models, such as memory polynomials and generalized memory polynomials. In recent years, there has been a growing interest in leveraging machine learning, particularly neural networks (NNs), for DPD, due to their ability to model complex nonlinear functions. Examples of recurrent neural networks (RNNs) explored in literature include a Long Short-Term Memory Neural Network (LSTMNN) [2], a Bidirectional LSTM Neural Network (BiLSTMNN) [3], and a Bidirectional Gated Recurrent Unit Neural Network (BiGRUNN) [4]. These RNNs have demonstrated strong performance when compared to traditional DPD models. However, their sequential data processing represents a bottleneck in computational efficiency. Because of the inherent limitation of this type of networks, the exploration of Transformer-based models for DPD presents significant potential, as their self-attention mechanism allows them to employ parallel computation, accelerating both training and inference times.

This thesis builds upon another master thesis, presented in [1], where an Augmented Real-Valued Time Delay Transformer Neural Network (ARVTDTNN) was proposed for DPD modeling. The ARVTDTNN showed promising results in tests using a single carrier signal with a bandwidth of 100 MHz, outperforming other state-of-the-art models. However, the effectiveness of Transformer-based models in more complex scenarios is still unexplored.

The primary objective of this thesis is therefore to extend the investigation of Transformer-based architectures for DPD, specifically focusing on multi-carrier signals with narrow bandwidths. The research focuses on developing and evaluating a Transformer-based NN model, investigating improvements to the network and to the features used. The model is evaluated on various multi-carrier signals scenarios and compared with other established NN architectures, respectively a LSTMNN, a BiLSTMNN, and a BiGRUNN. Additionally, the model is evaluated against Ericsson's proprietary DPD model, though detailed comparative data are not disclosed due to

confidentiality agreements.

The thesis is structured as follows: Chapter 2 provides a basic understanding of PAs and DPD, with Section 2.1 giving an overview of the function of PAs, Section 2.2 describing the nonlinear characteristics of PAs and the consequent challenges, and Section 2.3 introducing the concept behind DPD. Chapter 3 reviews state-of-the-art methods for modeling PAs and DPDs. It begins with classical approaches in Section 3.1, such as Volterra series and memory polynomials. Section 3.2 then discusses recent NN approaches that can be found in literature. Chapter 4 presents the original Transformer model, detailing its application in natural language processing tasks. Chapter 5 describes the experimental setup. Section 5.1 defines the iterative learning control framework used to obtain the DPD parameters. Section 5.2 presents the proposed Transformer-based NN model for DPD used in this thesis. Section 5.3 details the performance metrics used to evaluate the compared models. Chapter 6 presents the experimental results obtained from applying the Transformer-based model and the other RNN-based models. Chapter 7 concludes the thesis with a discussion of the findings and potential future research directions.

Chapter 2

Power Amplifiers and Digital Pre-Distortion

2.1 An Overview of Power Amplifiers

A radio frequency (RF) power amplifier (PA) is an electronic device positioned at the final stage of a radio transmitter, with the purpose of taking RF signals which are initially low in power, and boost them to higher power levels, more suitable for transmission over long distances. These signals are received from earlier stage components of the transmitter chain, such as mixers and modulators, and carry with them information that is encoded in their amplitude, frequency, or phase. A well-designed PA therefore ensures that the signal has enough strength to propagate wirelessly through various media, including air and space, without losing the integrity of this information.

In order for a PA to function correctly, it needs to meet several requirements, the most important of which are its efficiency and linearity [5].

The efficiency of a PA, denoted by η, is expressed as a percentage, and defined as follows:

$$\eta = \left(\frac{P_{\text{out}}}{P_{\text{in}}}\right) \times 100\% \tag{2.1}$$

Here, P_{out} is the power delivered to the antenna, and P_{in} is the total power consumed by the amplifier, usually supplied as direct current power. Since the PA is the most energy-consuming component in the entire transmitter chain [5] [6], achieving high efficiency is of crucial importance, as it means that less input power is required to reach the desired output power, minimizing the total energy consumption. An higher efficiency also means a reduction of the overall heat generation and power losses of the amplifier, thereby reducing the cooling requirements of the system and extending the lifespan of the device.

Linearity, on the other hand, ensures that the amplification does not distort the original signal. A linear amplifier accurately reproduces the input signal at the output, maintaining all the original information and staying within the designated bandwidth. In contrast, a nonlinear amplifier distorts the input signal, possibly leading to a loss of information and causing the output signal spectrum to expand, which may result in interference with other transmissions.

2.2 A Focus on the PA Nonlinear Region

By observing the input/output characteristic of a PA, which is illustrated in Figure 2.1, it is easy to understand how achieving a high efficiency and also maintaining a certain linearity is clearly a trade-off.

As the input power provided to the PA gradually increases, its operation shifts from a linear region, within which a direct proportionality between the input and output signal amplitudes is maintained, to a nonlinear one, where any additional input power results in progressively smaller increases in output power, leading to a plateau effect. The threshold separating these regions of operation is commonly known as compression point.

Operating the PA beyond the compression point, as previously mentioned, results in increased signal distortions, which lead to spectral regrowth and intermodulation effects, degradating the signal quality and causing interference with adjacent frequency channels. Despite these issues, there evidently still is an increase in efficiency by operating in this region, making it the most exploited one in real world implementations.

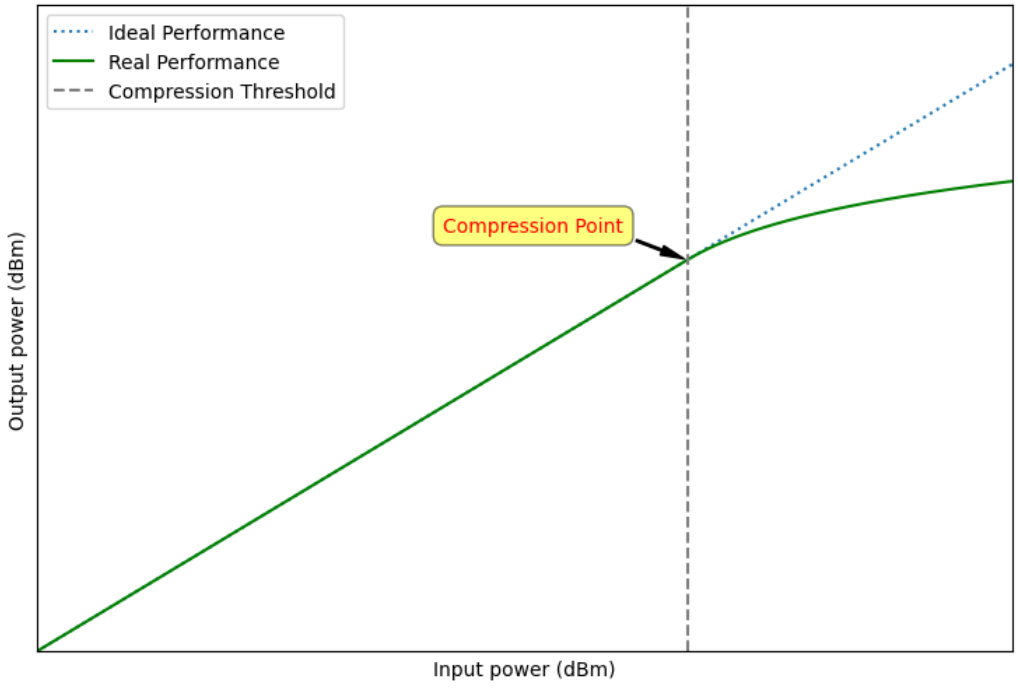

Figure 2.1: *Output versus Input Power Curve for a PA, with Visualization of the Compression Point.*

These last considerations highlight one of the most challenging aspects to consider when implementing a PA: find a way to manage and counteract the nonlinearities that arise past the compression point, while still benefiting from the higher energy efficiency offered in this region. Various solutions to this problem have been explored over the years, with some of the most effective ones being detailed in the following section.

2.3 About Digital Pre-Distortion

Among the various techniques that have been developed to reduce the nonlinear effects in PAs, methods such as feedback, feedforward, and pre-distortion have been widely used [7].

The feedback approach involves taking a portion of the output signal and feeding it back into the input. By comparing the feedback signal with the original one, any discrepancies (which represent distortion or errors) can be identified and corrected. Although this method can effectively improve the performance of a PA, it must be carefully managed, since too much feedback can lead to instability and oscillations, which might degrade the system's performance instead of improving it. On the other hand, feedforward techniques involve extracting the distortion components from the amplifier's output and creating an opposing signal to cancel out the distortion. This method is known to provide high linearity, but can be complex and costly to implement due to the additional circuitry required.

In contrast, digital pre-distortion (DPD) has emerged as a more efficient and technologically feasible solution, addressing this challenge by preemptively correcting the distortions before amplification [8].

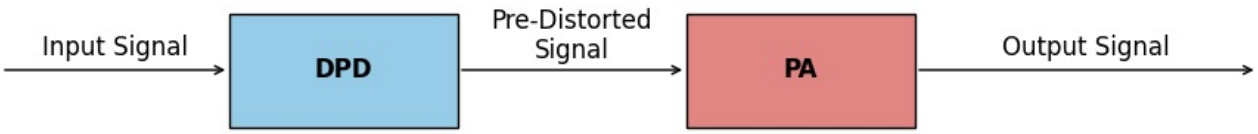

Figure 2.2: *Schematic representation of DPD in an RF PA chain.*

DPD works by modeling the nonlinear behavior of the PA and applying an inverse distortion to the input signal, thus ensuring that the final amplified output is linear.

Let's denote the nonlinear characteristic function of the PA as $f_{PA}(x)$, where x is the input signal to the amplifier. To counteract this nonlinearity, DPD introduces a pre-distortion function $f_{DPD}(x)$, which is designed to be the inverse of the amplifier's characteristic.

The relationship can be described by the following equations:

$$f_{DPD}(x) = f_{PA}^{-1}(x) \tag{2.2}$$

$$f_{DPD}(f_{PA}(x)) \approx G \cdot x \tag{2.3}$$

In Equation 2.3, G represents the gain of the PA, indicating that the output of the pre-distorted signal, when processed through the PA, results in a signal that is linearly amplified.

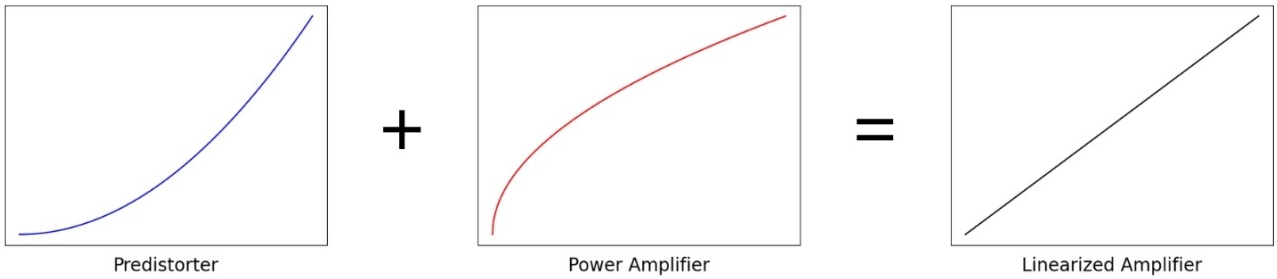

Figure 2.3: *Fundamental Concept Behind DPD to Achieve Desired PA Characteristics.*

As illustrated in Figure 2.2, the schematic block diagram outlines the process flow. The 'Input Signal' first passes through the DPD module, where it is pre-distorted to counteract the anticipated nonlinear effects of the PA. The resulting 'Pre-Distorted Signal' then enters the PA, where it is amplified. Figure 2.3 shows the concept employed to achieve the linearized characteristic of the system.

Chapter 3

An Overview of State-of-the-Art Power Amplifier and Digital Pre-Distorter Modeling

In this chapter, an overview of the current modeling techniques used to characterize both power amplifiers (PAs) and digital pre-distorters (DPDs) is provided.

It is worth noting that when creating a DPD model to correct the distortions of a specific PA, having an accurate model of the amplifier is extremely valuable, as it allows the DPD testing to be conducted in simulated environments, without the need of running the physical PA.

The following described models are categorized as either PA behavioural models or DPD models, but each one of them can be used for both purposes, reinforcing the idea that the DPD function should act as the inverse of the PA, which means both model types share the same physical principles.

3.1 Classical Approaches to PA and DPD Modeling

This first section looks at the traditional methods used in the field. It's important to understand how these models are built and the variables they use to capture PA nonlinearities, as they lay the basis for studying modern techniques based on neural networks (NNs).

3.1.1 Volterra Series

PA nonlinearities are mostly the result of memory effects in the system caused by physical properties like thermal behavior, charge storage, and inherent capacitances in the circuitry. As a consequence, the PA's output not only depends on the current input, but also significantly on previous inputs. The Volterra series is a foundational model used to describe the relationship between input and output of the PA with these memory effects.

The discrete Volterra series represents the output signal $y(n)$ as a sum of multi-dimensional convolutions of the input signal $x(n)$ with a series of kernels h_p, each corresponding to different orders of nonlinearity and spans of memory. The mathematical expression is given by:

$$y(n) = \sum_{p=1}^{P} \sum_{i_1=0}^{M} \sum_{i_p=0}^{M} h_p(i_1, \ldots, i_p) \prod_{j=1}^{p} x(n - i_j), \tag{3.1}$$

where $h_p(i_1, \ldots, i_p)$ models the interaction between p past values of the input signal $x(n)$ at different lags $i_1, \ldots, i_p$, effectively capturing the impact of the PA's memory.

Despite its robustness, using the Volterra series in real implementations results challenging due to its computational demands. As the series order and memory depth increase, the number of terms and the complexity of calculations rise, requiring practical compromises like truncating the series or simplifying higher-order terms.

3.1.2 Memory Polynomials

Memory polynomials are a more streamlined alternative to the Volterra series, designed to capture the key characteristics of memory effects with less computational effort [9]. They simplify the Volterra series by condensing its complex structure into a polynomial form, focusing on the important memory and nonlinearity effects with fewer parameters.

Memory polynomials are expressed as:

$$y(t) = \sum_{p=1}^{P} \sum_{m=0}^{M} \beta_{p,m} x(t-m)|x(t-m)|^{p-1}, \tag{3.2}$$

where $x(t)$ is the input signal at time t, $y(t)$ is the output signal, $\beta_{p,m}$ are the polynomial coefficients, P dictates the order of the nonlinearity, and M defines the memory depth.

3.1.3 Generalized Memory Polynomials

Generalized memory polynomials (GMP) build on memory polynomials by including cross-terms between different delayed samples of the input signal, improving the ability to capture some complex dynamics that simple memory polynomials might miss, particularly in high-order nonlinear systems [10].

The mathematical formulation of GMP is given by:

$$\begin{aligned} y(t) = &\sum_{p=1}^{P} \sum_{m=0}^{M} \beta_{p,m} x(t-m)|x(t-m)|^{p-1} \\ &+ \sum_{p=1}^{P} \sum_{m=0}^{M} \sum_{g=1}^{G} \gamma_{p,m,g} x(t-m)|x(t-m-g)|^{p-1} \\ &+ \sum_{p=1}^{P} \sum_{m=0}^{M} \sum_{g=1}^{G} \delta_{p,m,g} x(t-m)|x(t-m+g)|^{p-1} \end{aligned} \tag{3.3}$$

where $y(t)$ is the system output at time t, $x(t-m)$ is the input signal delayed by m time steps. Coefficients $\beta_{p,m}$, $\gamma_{p,m,g}$, and $\delta_{p,m,g}$ modulate the impact of input delays and nonlinearities. The indices p, m, and g respectively represent nonlinearity order, primary, and secondary memory depths, influencing how past and shifted inputs affect the output. Variables P, M, and G set the bounds for these sums.

Despite the increase in computational complexity compared to simple memory polynomials, they are still more manageable than the full Volterra series, making them suitable for practical applications.

3.2 Emerging Neural Network Approaches in PA and DPD Modeling

In recent years, some modern approaches using NNs for PA and DPD modeling have been explored in literature.

The reason why NNs are believed to be particularly well-suited for this task is due to their ability to approximate complex functions. According to the Universal Approximation Theorem [11], a feedforward NN with a single hidden layer can approximate any continuous function to a desired degree of accuracy, given a sufficient number of neurons in the hidden layer. This makes them a strong alternative to the traditionally complex Volterra-based models, which often still involve high computational demands, especially in more advanced transceiver architectures like massive multiple-input multiple-output (MIMO) systems [6].

This section starts by describing a simple shallow NN approach to give a general understanding of the commonly used input features. Then, it covers several recurrent neural networks (RNNs) implementations, which are better suited for the task due to their ability to handle data sequences, a significant characteristic when dealing with real-time radio frequency (RF) signal samples. Finally, a new Transformer-based NN approach is described, aiming to improve RNN performance through parallel computation.

3.2.1 Augmented Real Value Time-Dependent Neural Network

Among shallow NN proposed models, this paper [12] introduces the Augmented Real Value Time-Dependent Neural Network (ARVTDNN). This work builds on previous research in NN-based PA and DPD modeling, combining features from two different types of architectures, respectively Cartesian and Polar ones. By doing so, an enhanced input feature set to feed the NN with is created, leveraging the strengths of both architectures in order to improve the network's performance.

Cartesian architectures, as the name suggests, use Cartesian components, which are the in-phase (I) and quadrature (Q) signals representing the real and imaginary parts of a complex RF signal. On the other hand, Polar architectures use Polar components, which are the signal's amplitude and phase, with the aim of simplifying the processing of these complex signals.

The ARVTDNN model includes the I and Q components, as well as the amplitude of the input signal and its nonlinear versions, integrating the benefits of both approaches. Moreover, the ARVTDNN model addresses memory effects by including past and current samples of the input signal and their envelope terms.

The input signal vector for the ARVTDNN is represented as:

$$x(n) = \begin{bmatrix} I_{\text{in}}(n), & I_{\text{in}}(n-1), & \dots, & I_{\text{in}}(n-M), \\ Q_{\text{in}}(n), & Q_{\text{in}}(n-1), & \dots, & Q_{\text{in}}(n-M), \\ |X_{\text{in}}(n)|, & |X_{\text{in}}(n-1)|, & \dots, & |X_{\text{in}}(n-M)|, \\ |X_{\text{in}}(n)|^2, & |X_{\text{in}}(n-1)|^2, & \dots, & |X_{\text{in}}(n-M)|^2, \\ |X_{\text{in}}(n)|^3, & |X_{\text{in}}(n-1)|^3, & \dots, & |X_{\text{in}}(n-M)|^3 \end{bmatrix} \tag{3.4}$$

Here, $I_{\text{in}}(n)$ and $Q_{\text{in}}(n)$ denote the in-phase and quadrature components of the current samples, while $I_{\text{in}}(n-k)$ and $Q_{\text{in}}(n-k)$ represent the in-phase and quadrature components of past samples for $k = 1, 2, \ldots, M$. $|X_{\text{in}}(n)|$ is the amplitude of the current sample, and $|X_{\text{in}}(n-k)|$ is the amplitude of the past samples. Three orders of nonlinearity are included for the amplitude components, while the memory depth M is optimized based on the system's memory effects.

The model's output represent the I and Q components of the predicted signal. The structure of the network is illustrated in Figure 3.1.

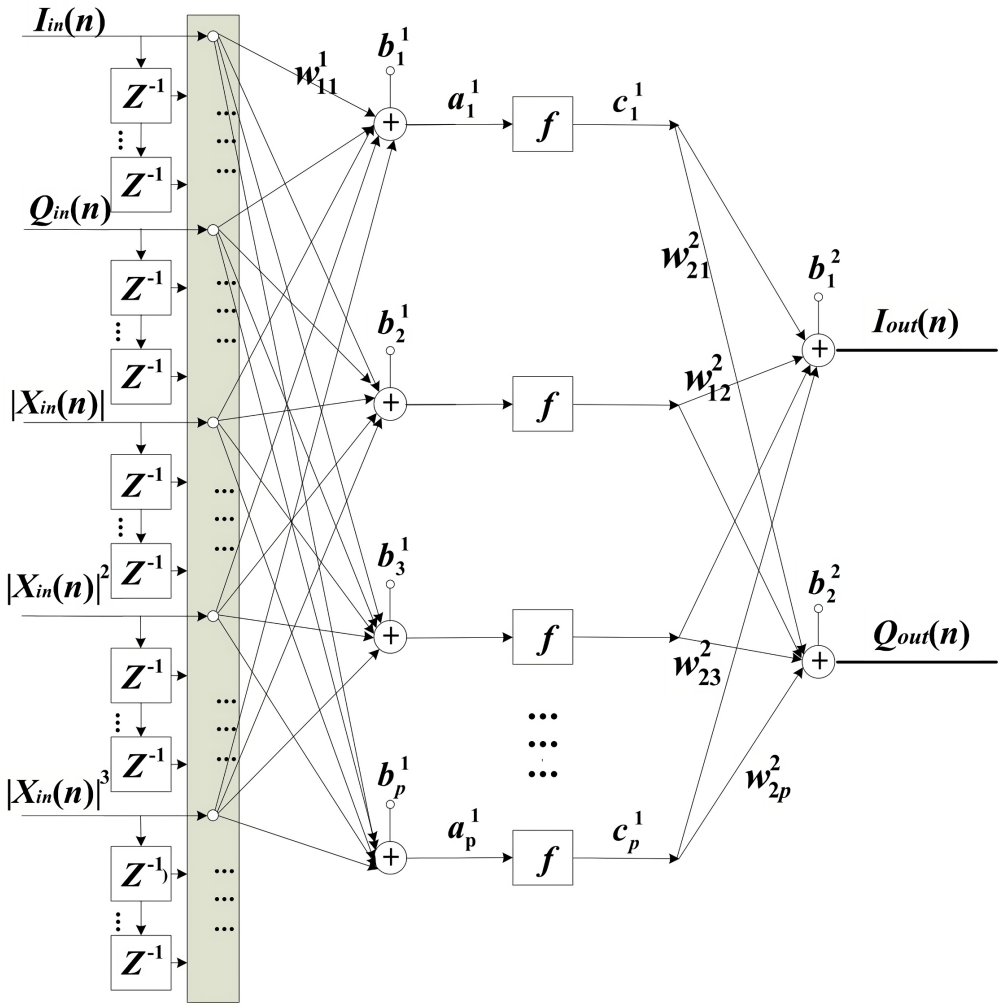

Figure 3.1: *ARVTDNN structure, from [12].*

3.2.2 Long Short-Term Memory-Deep Neural Networks based Predistortion Linearizer

The ARVTDNN model has shown its effectiveness in mapping the inverse function of a PA. However, it struggled with generalization, especially under varying system conditions such as IQ imbalance and DC offset [2]. This study [2] introduces a Long Short-Term Memory-Deep Neural Networks based Predistortion (LSTM DNN based PD) Linearizer, designed to address PA linearization and also able to improve the generalization capabilities compared to the previously proposed network.

LSTMs are a type of RNN particularly suited for handling time series data due to their ability to remember and use past information. LSTM units, shown in Figure 3.2, consist of cells and three types of gates: input, output, and forget gates. These gates work like neurons in a feedforward NN, calculating activations based on the weighted sums of their inputs, while peephole connections within the unit allow the gates to use the memory cell's state from the previous time step, helping the model to remember long-term dependencies. Thanks to these characteristics, this type of NN represents an ideal fit for the PA modeling task.

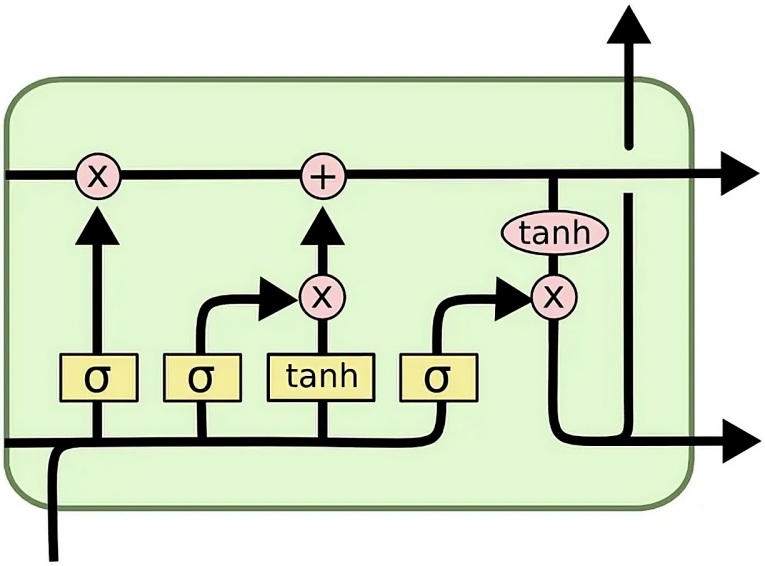

Figure 3.2: *The image illustrates the structure of a LSTM unit, showing the input gate, forget gate, and output gate, all represented by the sigmoid function σ. The cell state, depicted by the full horizontal line, acts as the memory, carrying information across time steps. The tanh function is used to update the cell state and regulate the output. Multiplication and addition operations combine the gates' outputs to update the cell and hidden states. The arrows indicate the flow of information through the unit. From [13].*

The proposed LSTM DNN PD Linearizer, whose architecture is depicted in 3.3, uses an LSTM layer to process the signal sequence, which consists of current and past samples of the I and Q components of the baseband signal. The outputs from the LSTM layer are then fed into a fully connected (FC) layer with rectified linear unit (ReLU) activation. Finally, the output layer uses a linear activation function to produce the desired I and Q outputs.

The network input vector $X_{\text{in}}(n)$ is detailed as follows:

$$X_{\text{in}}(n) = \Big[I_{\text{in}}(n), I_{\text{in}}(n-1), \ldots, I_{\text{in}}(n-m), \\ Q_{\text{in}}(n), Q_{\text{in}}(n-1), \ldots, Q_{\text{in}}(n-m)\Big], \tag{3.5}$$

where m is the memory depth.

As previously mentioned, the suggested PD Linearizer successfully captures and corrects the memory effects of the PA, showing better overall performance than shallow NNs, also demonstrating superior generalization capabilities.

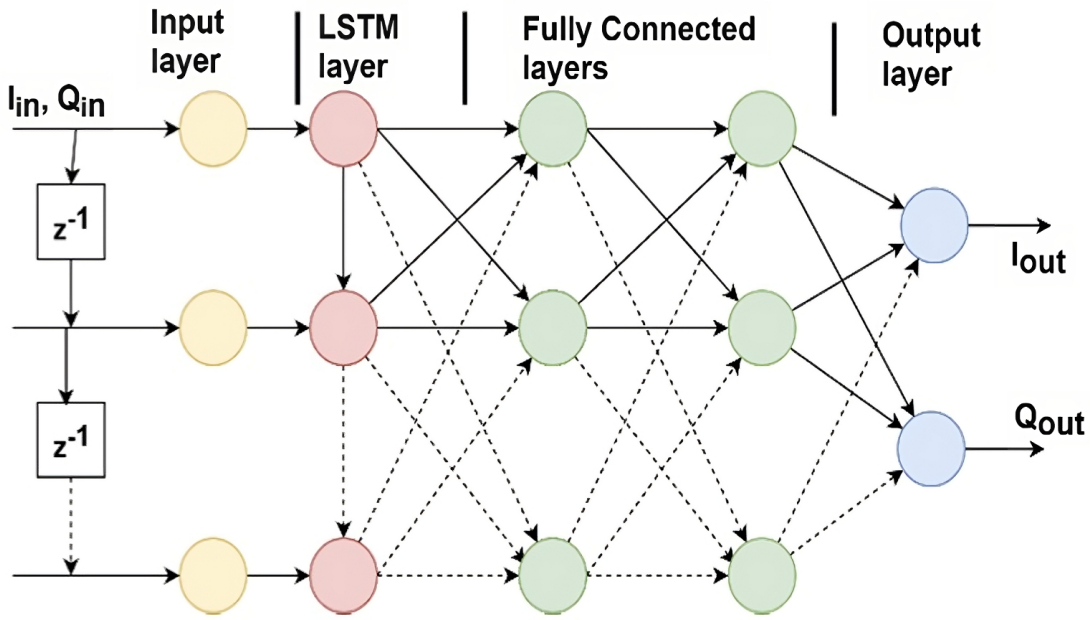

Figure 3.3: *LSTM-DNN PD Linearizer structure, from [2].*

3.2.3 Bidirectional Long Short-Term Memory Neural Network

In [3], a Bidirectional Long Short-Term Memory (BiLSTM) Neural Network behavioral modeling architecture is introduced.

BiLSTM networks, whose concept is depicted in 3.4, unlike standard LSTM networks that process data in a single direction, process data in both forward and backward directions. This approach allows the model to capture dependencies and patterns that may be missed when looking at the sequence in only one way. The rest of the BiLSTM unit is structured similarly to a normal LSTM one, also consisting of cells and three types of gates: input, output, and forget gates.

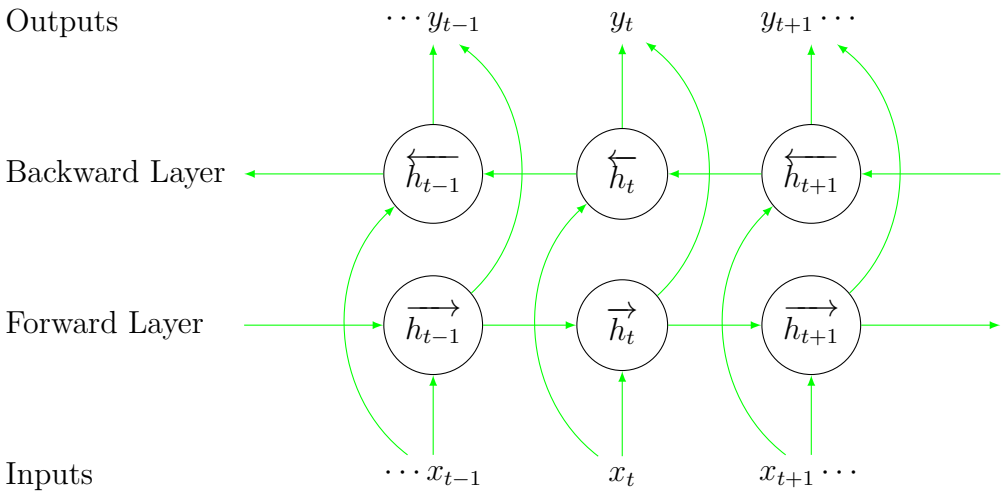

Figure 3.4: *This diagram illustrates the architecture of a BiLSTM network. The BiLSTM consists of two LSTM layers: a forward layer that processes the input sequence x_t from $t = 1$ to $t = T$, and a backward layer that processes the sequence from $t = T$ to $t = 1$. The forward hidden states $\overrightarrow{h_t}$ and backward hidden states $\overleftarrow{h_t}$ are concatenated at each time step to form the context vector for that step. The output y_t at each time step is generated by combining information from both directions.*

The proposed architecture comprises five layers: an input layer, a BiLSTM layer, and three FC layers. Empirical results show how the proposed BiLSTM-based behavioral model outperforms existing deep learning approaches, especially when dealing with PAs that exhibit complex memory effects.

3.2.4 Bidirectional Gated Recurrent Deep Neural Network

A Bidirectional Gated Recurrent Deep Neural Network (BGDNN) is proposed in [4].

Gated Recurrent Units (GRUs), illustrated in 3.5, address some limitations found in traditional RNNs and LSTM networks, such as the vanishing gradient problem, also reducing the total number of parameters used. Unlike LSTMs, which use the input, output, and forget gates, GRUs simplify the structure by utilizing just two of them: the reset gate and the update gate. These gates help manage the information flow within the network, determining what to retain from prior states and how new inputs should affect the current state. This streamlined gating mechanism allows GRUs to capture temporal dependencies more efficiently and often with fewer parameters, leading to quicker training times and reduced computational complexity.

Similarly to the difference between a BiLSTM unit and a LSTM one, the BiGRU improves the GRU data processing by examining sequences in both forward and backward directions. Each BiGRU therefore consists of two GRU cells: one processes the sequence forward from start to finish, while the other processes it backward from end to start.

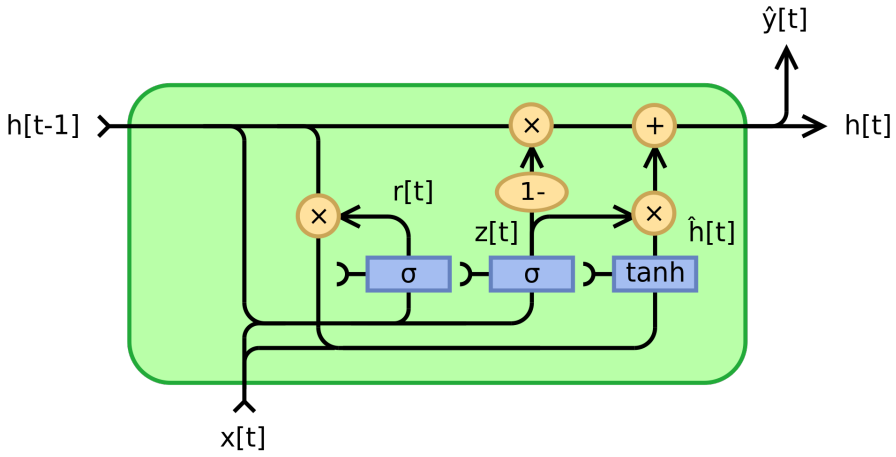

Figure 3.5: *This diagram illustrates the structure of a GRU, including the update gate $z[t]$ and reset gate $r[t]$, both of which are activated by a sigmoid function σ. The reset gate determines how much of the previous state $h[t-1]$ to forget, while the update gate controls how much of the new input $x[t]$ and the previous state to retain. The candidate activation $\tilde{h}[t]$ is calculated using the* tanh *function. The final hidden state $h[t]$ is a linear interpolation between the previous hidden state and the candidate activation, weighted by the update gate. From Jeblad [14].*

In the proposed model, following the BiGRU layer, a single FC layer is added. Several simulations demonstrate that the BGDNN outperforms other recent RNN-based PA behavioral modeling methods.

3.2.5 Augmented Real-Valued Time Delay Transformer Neural Network

In [1], an Augmented Real-Valued Time Delay Transformer Neural Network (ARVTDTNN) is introduced to address the limitations of RNN-based models.

This proposed model is based on the Transformer architecture [15], which utilizes a self-attention mechanism to process entire data sequences simultaneously rather than one step at a time, like traditional RNNs do. This parallelism significantly improves computational efficiency, reduces training times, and enhances the scalability of the model.

The architecture of the ARVTDTNN is depicted in Figure 3.6. It consists of a Transformer encoder, which is preceded by layers for input embedding and positional encoding, and followed by an FC layer and an output layer. The input features include past and present I and Q components of the signal, and various nonlinear orders of the amplitude. Empirical tests show that this model outperforms several state-of-the-art NNs in various performance metrics when evaluated using a single carrier signal with a bandwidth of 100 MHz. This research lays the foundation for this thesis, which aims to investigate and test a Transformer-based architecture in multi-carrier scenarios, which are known to be more complex and difficult to model.

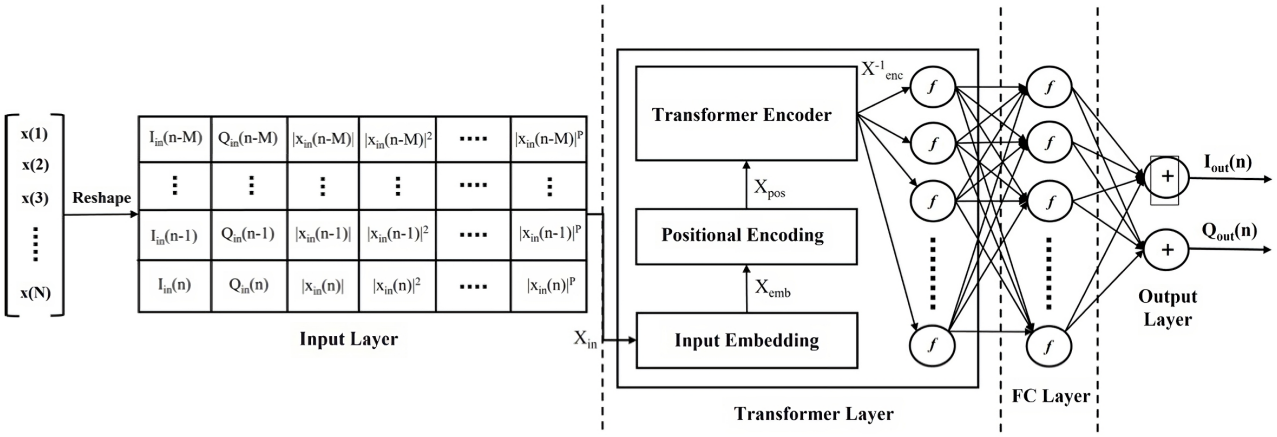

Figure 3.6: *This diagram illustrates the structure of the ARVTDTNN. The model begins with an input layer where the raw input sequence x is reshaped into a matrix including in-phase I_{in}, quadrature Q_{in}, and various nonlinear terms of the signal x. This matrix X_{in} is then processed through the Transformer layer, comprising input embedding, positional encoding, and a single Transformer encoder layer. The output of the encoder is fed into a FC layer, which further processes the data before passing it to the output layer. The output layer generates the final I_{out} and Q_{out} components. From [1].*

Chapter 4

The Transformer Architecture

This chapter provides a detailed description of the original Transformer architecture, proposed for the first time in [15]. By understanding how this model improves on earlier recurrent neural networks (RNNs), it is easier to comprehend its potential as an effective solution for power amplifier (PA) linearization.

In the previous chapter several types of RNNs were detailed, highlighting how their sequential approach allows them to maintain a form of memory over the inputs that they have previously processed, making them particularly suitable for tasks that involve managing sequences, such as natural language processing (NLP), which is their most common application, but also for the problem of digital pre-distortion (DPD). However, it was also mentioned that these networks' way of processing sequences one step at a time poses limitations in terms of their computation efficiency, leading to extensive training sessions and long inference times.

The Transformer architecture finally overcomes this limitation by using its self-attention mechanism to process all elements of a sequence simultaneously. This mechanism allows each position in the sequence to dynamically pay attention to every other position, helping the model to understand the context and assign appropriate weights to each element based on its relevance, achieving a full understanding of the whole sequence all at once. Consequently, this design allows for parallel processing of data, as the computation of the attention scores and subsequent outputs does not depend on any sequential step-by-step processing, allowing the Transformer to make full use of devices such as graphical processing units (GPUs) and tensor processing units (TPUs), significantly reducing both training and inference times.

The following sections take a closer look at the layers of the Transformer architecture, shown in Figure 4.1. Each part is explained in detail, particularly in the context of NLP tasks, to facilitate a better understanding.

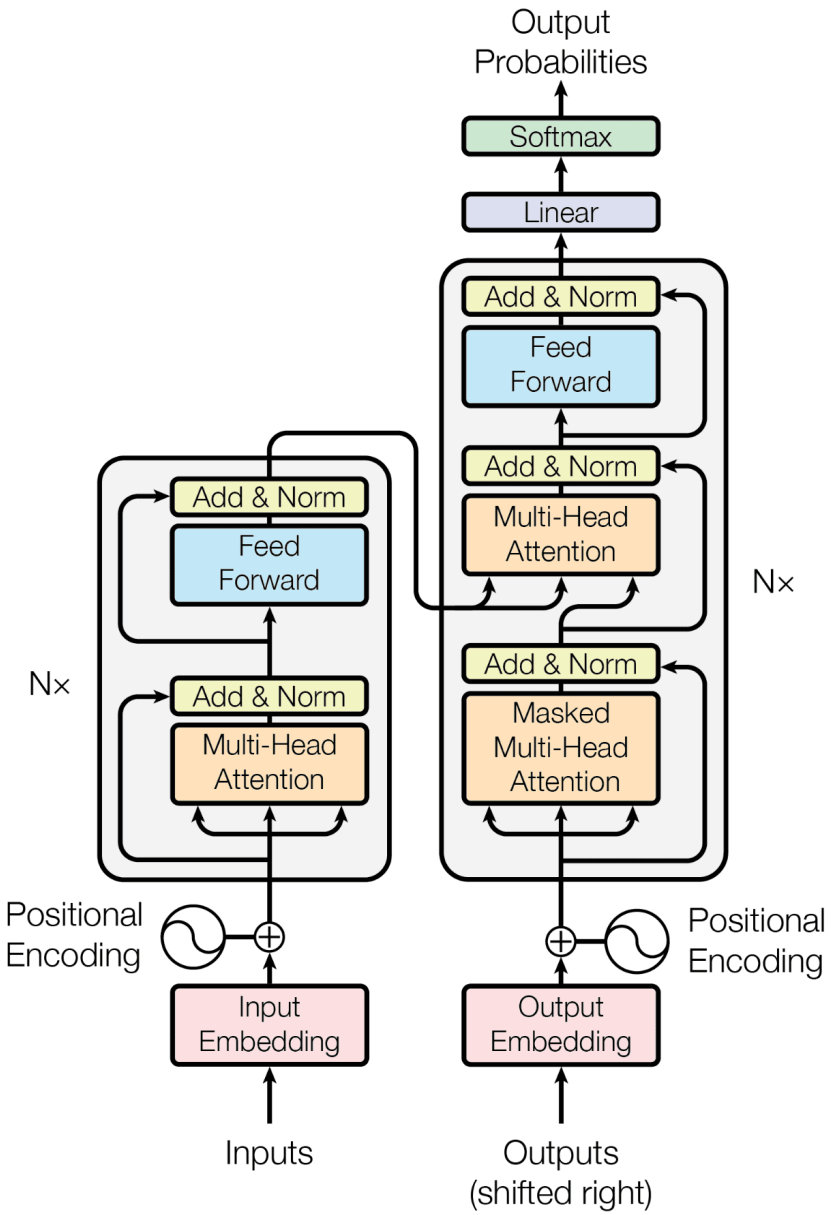

Figure 4.1: *The Transformer-model architecture, from the paper "Attention is all you need" [15].*

4.1 Input Embeddings

The model begins with an input embedding block, which transforms discrete elements like words or tokens used in NLP into fixed-size, continuous, high-dimensional vectors. This transformation allows the network to handle inputs in a numerical format, by creating a multi-dimensional space where each word's position reflects its meaning, and the distance and direction between these vectors show how similar or related the words are to each other, capturing their semantic relationships.

Figure 4.2 illustrates the relationships between different word pairs using a simple 3D visualization of their embeddings. Note that in real applications, the dimension of the embedding space is usually much higher. In the first plot, "king" is close to "queen" in the same way "man" is close to "woman," indicating a similar gender relationship. In the second plot, "walking" is near

"walked" and "swimming" is near "swam," reflecting the change in verb tense. In the third plot, "Canada" is close to "Ottawa" and "Italy" is close to "Rome," showing the relationship between countries and their capitals. These distances illustrate how word embeddings capture the semantic relationships between words, with similar concepts being closer together in the embedding space.

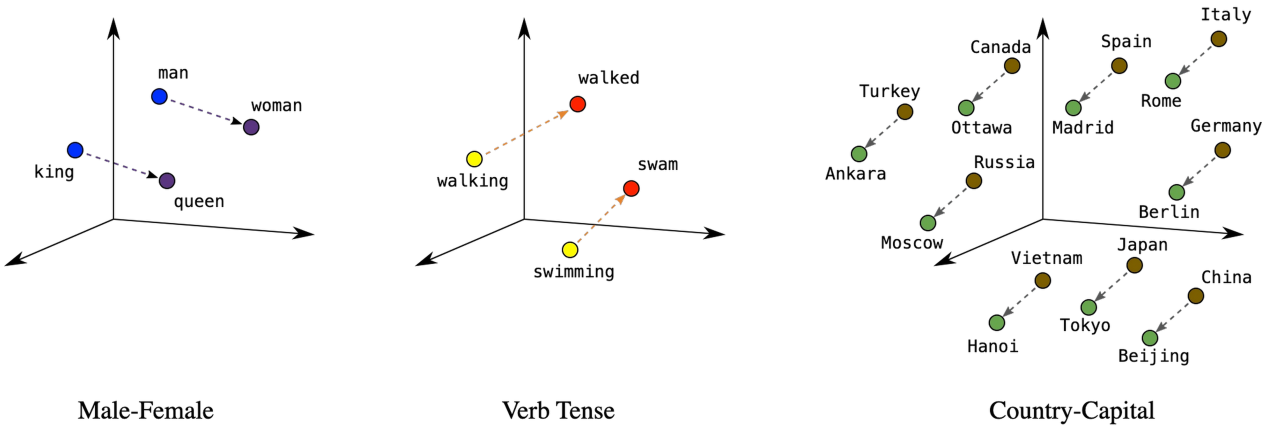

Figure 4.2: *Visual representation of 3D word embeddings, showing relationships between male-female pairs, verb tenses, and country-capital pairs. Distances and directions between points indicate semantic similarities and relationships. From [16].*

4.2 Positional Encoding

The next layer is about positional encodings, which is the way the model use to include the sequence order information into the embeddings. As stated above, Transformers process input tokens in parallel, unlike RNNs that account for sequence position through their sequential processing. To address this, these positional encodings are added directly to the input embeddings, providing necessary context about order within the sequence.

These encodings use a sinusoidal function based on the token's position in the sequence, generating a unique positional signal for each token. Specifically, for each position *pos* and dimension i within the embedding, the encoding uses sine and cosine functions oscillating at different frequencies:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.1}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4.2}$$

where d_{model} represents the dimensions of the embedding.

4.3 Encoder

The encoder's role is to transform the input sequence into a rich, continuous representation that captures both the contextual meaning and positional relationships in the data. This

transformation prepares the input for further processing by the decoder.

This block is composed of a series of identical layers stacked sequentially. Each layer in this stack features two main components: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network (FFN). Both of them are also followed by an Add & Norm block. The input to the encoder consists of the encoded embeddings previously described.

4.3.1 Self-Attention

The first component is the multi-head attention block, but in order to explain it effectively, it is crucial first to understand how self-attention works, also referred to as scaled dot-product attention.

Self-attention is a mechanism that allows the model to assess how different parts of the input relate to each token, helping it understand the context and the interdependencies among the tokens.

In mathematical terms, it can be described by the following function:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.3}$$

Here, Q, K, and V represent matrices for queries, keys, and values, respectively, which are terms borrowed from information retrieval and database indexing. By imagining that each piece of input data starts a quest to find the most relevant information it needs, each query acts like a question, looking around for answers. The keys are like clues that help match these questions to the right spots where answers are hidden. When a query finds a good match with a key, it points directly to a value, which is the information or answer the input was originally looking for. This whole process helps the model make sense of and remember where everything important is located in the sequence, making it easier to use this information in its tasks.

These matrices are derived from the same input embeddings but transformed by different weight matrices W^Q, W^K, and W^V, as follows:

$$Q = XW^Q \tag{4.4}$$

$$K = XW^K \tag{4.5}$$

$$V = XW^V \tag{4.6}$$

Where X denotes the matrix of input embeddings and W^Q, W^K, W^V are the parameter matrices that are learned during training.

The self-attention mechanism therefore starts by computing the attention scores, taking the dot product of the query with all keys and scaling the result by the square root of the dimensionality

of the keys, d_k:

$$\text{Score} = \frac{QK^T}{\sqrt{d_k}} \tag{4.7}$$

These scores are then normalized using the softmax function, converting them into a probability distribution that sums to one across all tokens:

$$\text{Attention Scores} = \text{softmax}\left(\text{Score}\right) \tag{4.8}$$

The output of the self-attention layer is computed as a weighted sum of the values, where the weights are given by the attention scores:

$$\text{Output} = \text{Attention Scores} \cdot V \tag{4.9}$$

This output is finally an enhanced representation of the input sequence, where each token is enriched with context from the entire sequence.

4.3.2 Multi-Head Attention

The concept of multi-head attention, illustrated in Figure 4.3, can be introduced as an extension of self-attention.

While self-attention calculates a single set of queries, keys, and values, multi-head attention performs this process multiple times using different learned linear transformations.

The operation is described by the following function:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O \tag{4.10}$$

where each head_i is a separate self-attention operation with its own set of weight matrices:

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \tag{4.11}$$

In this configuration, X is the embedded input sequence, W_i^Q, W_i^K, and W_i^V are the parameter matrices specific to each head, and W^O is an additional weight matrix used to linearly transform the concatenated output of all heads.

The idea behind multi-head attention is to allow the model to capture different types of relationships, such as syntactic and semantic connections, at various positions in a sentence more effectively than a single head could. By using different sets of learned weights to project the queries, keys, and values, each head can focus on different aspects of the context within the input

sequence. For instance, one head might work on understanding grammar, helping the model recognize rules like subject-verb matching, while another head might specialize on understanding the meanings of words and how they relate to each other, which is useful for understanding synonyms or how words change meaning depending on their context. A third head could instead pay attention to how different sections of text are connected, in order to maintain the context even in long pieces of text. By having these different focuses, the model can get a more complete understanding of the text, leading to better performance overall.

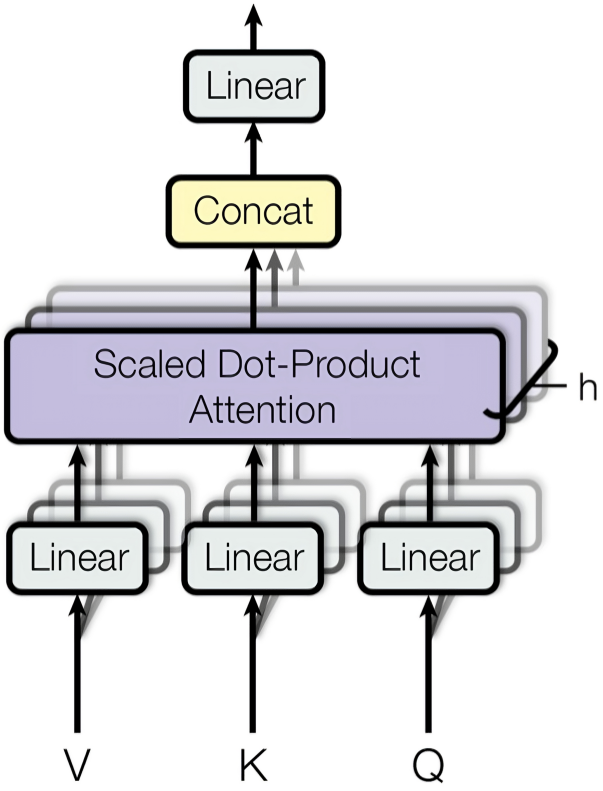

Figure 4.3: *Diagram illustrating the multi-head attention mechanism in a Transformer model. It shows how queries (Q), keys (K), and values (V) are processed through individual linear transformations, combined in parallel heads, and then concatenated and linearly transformed to produce the final output. This structure allows the model to handle information from different perspectives at the same time. From [15].*

4.3.3 Feed-Forward Network

In each layer of the encoder, after the multi-head attention mechanism, there is a position-wise fully connected FFN, which transforms the attention-modulated representations into higher-level features, further improving the model's ability to process information.

The FFN applies the same two linear transformations with a ReLU activation in between to each position separately and identically. This means that the same operation is performed on all positions independently, preserving the positional information.

The mathematical formulation of the FFN is given by:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4.12}$$

Here, x represents the input to the FFN, which is the output from the preceding multi-head attention layer for each position. W_1 and W_2 are the weight matrices for the two linear transformations, and b_1 and b_2 are their corresponding bias vectors. The ReLU function, represented by $\max(0, z)$, introduces nonlinearity into the network.

4.3.4 Add & Normalization Blocks

Each sub-layer within the encoder, specifically the multi-head attention and the FFN, is followed by an additional structure known as the Add & Norm block, which helps in maintaining the stability of the network and improves the training efficiency.

The Add & Norm block consists of two main components: a residual connection and layer normalization. The structure can be described mathematically as follows:

$$\text{Layer Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \tag{4.13}$$

Here, x represents the input to the sub-layer, and $\text{Sublayer}(x)$ is the output from either the multi-head attention mechanism or the FFN.

The residual connection, represented by $x + \text{Sublayer}(x)$, helps to solve the vanishing gradient problem that occurs during training. By adding the input x to the sub-layer's output, gradients can flow directly through the layers, preventing them from becoming too small and stopping the network from learning.

After the residual connection, layer normalization (LayerNorm) is applied. This process adjusts and scales the input data within each layer, standardizing the means and variances. Layer normalization computes the mean and variance from all the inputs to the neurons in a layer for a single training case, unlike batch normalization, which works across the batch dimension. The calculations are:

$$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i \tag{4.14}$$

$$\sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2 \tag{4.15}$$

$$\text{LayerNorm}(x_i) = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta \tag{4.16}$$

where H is the number of hidden units, x_i is the input to a unit, γ and β are learnable parameters, and ϵ is a small constant added for numerical stability.

4.4 Decoder

The decoder is designed to transform the encoded information into the final output sequence that the model generates. It has a structure similar to the encoder, but includes additional features specifically for generating sequences.

It consists of several identical layers stacked on top of each other, where each layer has three components: masked self-attention, encoder-decoder attention, and a position-wise fully connected FFN.

The first component is the masked attention mechanism. This feature is a variation of the attention mechanism used in the encoder, but with an important modification: it uses a mask to prevent the decoder from accessing future positions in the output sequence. This ensures that during the prediction phase, the decoder is 'blind' to future tokens, so that in the process of generating a sentence, it can only 'see' and consider the words it has already produced. This setup mirrors NLP tasks, such as language translation or text generation, where each word choice depends on the preceding context and not on future words not yet generated. The formula for masked attention is:

$$\text{Masked Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V \tag{4.17}$$

Here, M represents a mask matrix that sets future tokens to a value that approaches negative infinity before the softmax operation, ensuring that they do not contribute to the output.

Following the masked attention, the next component is the encoder-decoder attention mechanism, which enables the decoder to use the output from the encoder. Here, the output from the masked self-attention is used as the queries (Q), while both the keys (K) and values (V) are taken from the encoder. Essentially, it enables the decoder to 'look back' at the entire input sequence as it generates each token in the output sequence, ensuring that the generated content is contextually relevant and coherent with the input. The formula is the following:

$$\text{Encoder-Decoder Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.18}$$

Each layer then contains a position-wise fully connected FFN, which applies the same linear transformations to each position independently.

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{4.19}$$

Similarly to the encoder, each of these components is followed by an Add & Norm block, which again consists of a residual connection and layer normalization, as previously described.

$$\text{Layer Output} = \text{LayerNorm}(x + \text{Sublayer}(x)) \tag{4.20}$$

Through the combination of these modules, the decoder generates an output sequence step-by-step, which uses the processed information to determine the most appropriate next elements in the sequence, continuing until it produces an end-of-sequence token, indicating that the task is complete.

Chapter 5

Experimental Setup and Proposed Transformer-based Neural Network Model for Digital Pre-Distortion

This chapter presents the methodological framework used to develop and evaluate the proposed Transformer-based neural network (NN) model for digital pre-distortion (DPD). The discussion is structured into three sections for better clarity: the first section describes the method used to determine the model parameters, the second one focuses on the details of the model structure, while the last section provides an overview of the performance metrics used to assess the performance of the compared NN architectures.

5.1 Iterative Learning Control

In this section, the method used to identify the Digital Pre-Distortion (DPD) model parameters is detailed.

Traditionally, two primary learning architectures are used for this task: the indirect learning architecture (ILA) or the direct learning architecture (DLA).

ILA operates on a two-step process. In the first step, the system identifies the PA's behavior by capturing the input and output signals of the PA and developing an approximate model of the PA's nonlinear function. This model aims to replicate the actual behavior of the PA as closely as possible. Once the PA model is established, the second step involves deriving the predistorter parameters by computing the inverse of this modeled transfer function. This inverse model is then used as the DPD.

The primary advantage of ILA is that it allows the modeling and updating of the DPD based on the measured performance of the PA. However, this approach is susceptible to two main issues. First, the presence of noise in the PA's output signal can lead to biased parameter estimates, making the predistorter less effective at compensating for the PA's nonlinearities [17]. Second, ILA struggles to perform effectively when dealing with PAs that exhibit severe nonlinear characteristics, as the inverse modeling approach may not accurately capture the complex dynamics of the amplifier [17].

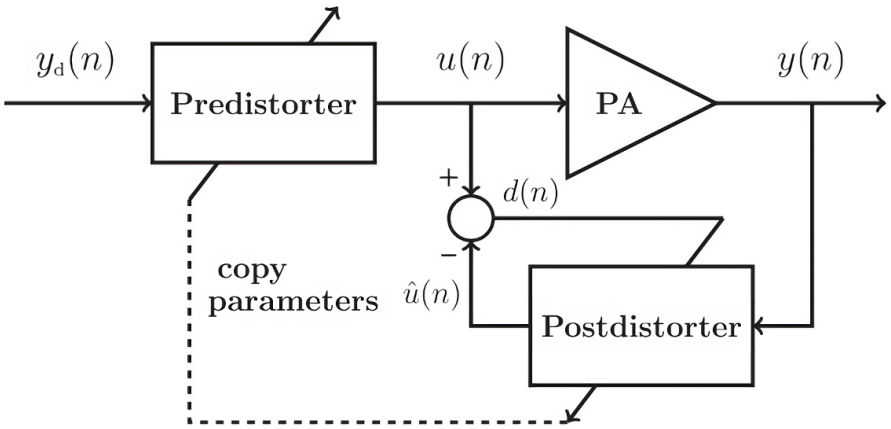

Figure 5.1: *Block diagram of the ILA, from [17]. The input signal $y_d(n)$ undergoes processing by the predistorter to produce $u(n)$, which is then amplified by the PA to generate the output $y(n)$. This output is subsequently processed by the postdistorter to generate $\hat{u}(n)$. The distortion error, represented by the difference $d(n)$ between $\hat{u}(n)$ and $u(n)$, is calculated and used to adjust the postdistorter's parameters. These optimized parameters are then transferred back to the predistorter, continuously improving its performance.*

On the other hand, DLA overcomes some of the problems associated with ILA by directly adjusting the predistorter parameters based on the comparison between the desired input and the error observed at the PA's output. This method avoids the need for an intermediate model of the PA's behavior, potentially offering more accurate and unbiased parameter estimates. However, DLA also comes with its own challenges, primarily related to computational demand and complexity. Additionally, these algorithms often exhibit slow convergence rates, making them less ideal for scenarios where rapid deployment or real-time adjustments are necessary [17].

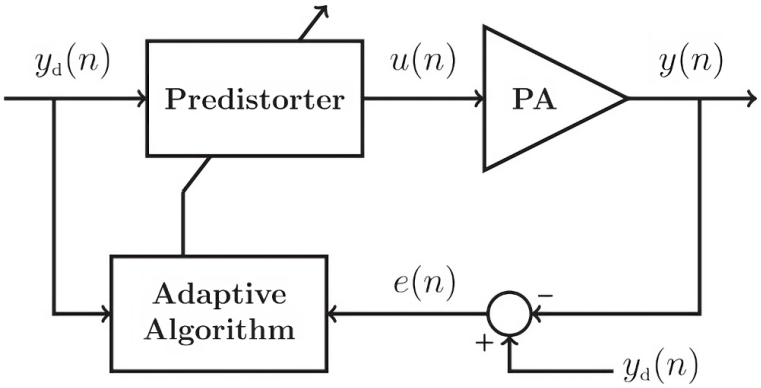

Figure 5.2: *Block diagram of the DLA, from [17]. The input signal $y_d(n)$ enters a Predistorter, where it undergoes modification to preemptively compensate for expected distortions. The modified signal, denoted as $u(n)$, is then amplified by a PA, resulting in the output signal $y(n)$. To align the output closer to the desired output, the signal $y(n)$ and the original input $y_d(n)$ are compared. The difference, or error signal $e(n)$, is fed back into the Adaptive Algorithm. This algorithm adjusts the Predistorter's parameters to minimize the error in subsequent signals.*

The iterative learning control (ILC) method, which is the one used in this thesis, is a technique derived from control theory, proposed for the first time in [17] as a solution for the DPD parameters estimation. The key idea behind this method is its iterative process, which is used

to accurately invert the behavior of a nonlinear system.

Unlike conventional techniques that aim to identify the predistorter's parameters directly, ILC prioritizes finding the optimal input signal for the PA (i.e. the ideal pre-distorted signal) to achieve the desired linear output. This is done through an iterative learning algorithm that adjusts the PA input based on the deviation from the desired output. With each iteration, the input signal is refined according to the error observed in the previous output, progressively approaching an input signal that minimizes this error.

Once the optimal PA input signal is determined, the predistorter's parameters are then estimated using standard modeling techniques. This approach represents a major change from traditional DPD methods, since ILC allows for the optimization of the input signal before even considering the predistorter parameters.

Figure 5.3 shows the process to find the ideal pre-distorted signal. The goal is to make the PA output $y(n)$ match a desired response $y_d(n)$. This involves finding an optimal input $u^*(n)$ such that the output closely approximates $y_d(n)$. This optimal input is the signal after it has passed through the DPD process. The learning controller iteratively calculates the error $e_k(n) = y_d(n) - y_k(n)$ for the $k-th$ iteration and uses it to update the input signal to $u_{k+1}(n)$ for the next iteration. This process continues until the output closely matches the desired response, and the optimal input $u^*(n)$ is identified.

In the next step, illustrated in Figure 5.4, $y_d(n)$ and $u^*(n)$ are used to model the predistorter. Here, $y_d(n)$ acts as the input to the model, and $u^*(n)$ is treated as the output. This approach allows for the precise design of the predistorter parameters, since the target output $u^*(n)$ is already known.

An additional advantage of the ILC-DPD scheme is that it allows testing different predistorter model structures and settings without using a model of the PA. Directly comparing the predicted pre-distorted signal with the ideal one already provides a reliable performance metric, reflecting the ability of the model to counteract the PA's distortions.

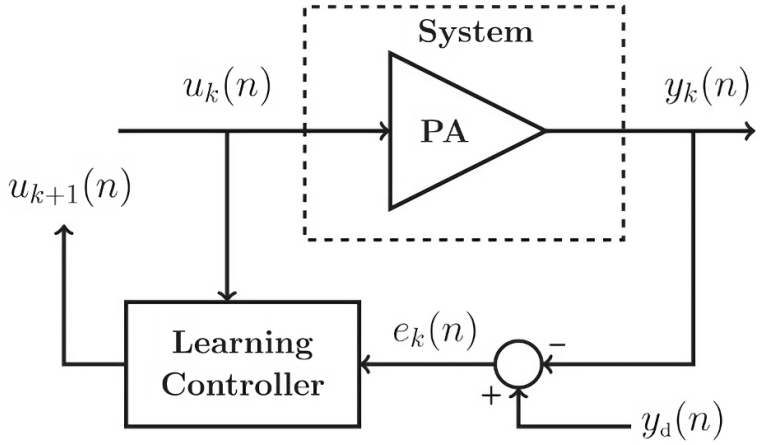

Figure 5.3: *ILC Scheme showing how the ideal pre-distorted signal is obtained, from [17]. The objective is to adjust the PA input $u_k(n)$ so that the PA output $y_k(n)$ matches the desired response $y_d(n)$. The learning controller iteratively computes the error $e_k(n) = y_d(n) - y_k(n)$ at each iteration k and updates the input signal to $u_{k+1}(n)$ for the next iteration. This process continues until the PA output closely approximates the desired response, resulting in the identification of the optimal input signal $u^*(n)$.*

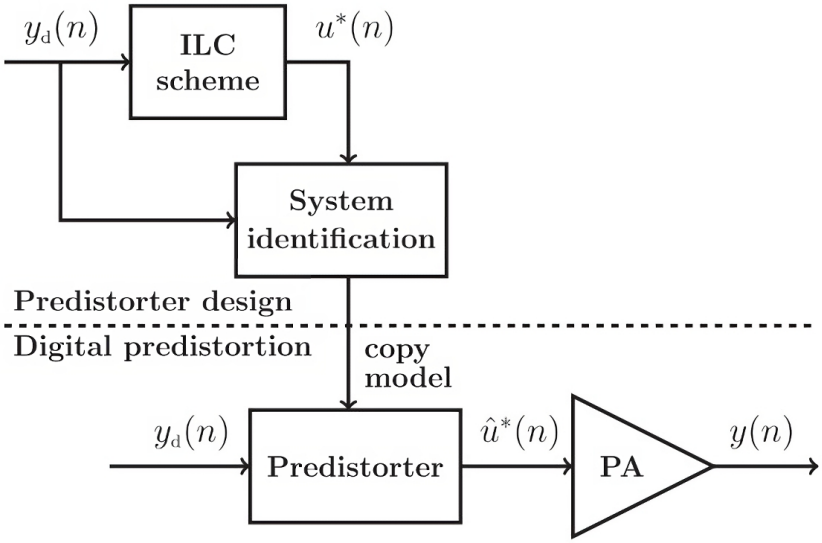

Figure 5.4: *ILC Scheme showing how the DPD model parameters are obtained, from [17]. In this approach, $y_d(n)$ serves as the input to the system identification model, while $u^*(n)$ is the target output. This method ensures accurate determination of the predistorter parameters, since the optimal output $u^*(n)$ is pre-defined.*

5.2 Proposed Transformer-based Neural Network Model

In this section, the Transformer-based NN model for DPD proposed in this thesis is detailed.

While Transformers have traditionally been used in natural language processing (NLP) due to their architecture being optimized for such tasks, recent literature has explored their potential in time series forecasting and regression. This is mainly due to their ability to process sequential data and their parallel computational capabilities, facilitated by the attention mechanism.

As previously described in Chapter 3, in [1] an Augmented Real-Valued Time Delay Transformer Neural Network (ARVTDTNN) for DPD modeling has been proposed. The ARVTDTNN showed promising results in tests using a single carrier signal with a bandwidth of 100 MHz, outperforming other state-of-the-art models. The scope of this thesis is therefore to continue the investigation on the Transformer architecture for DPD modeling, evaluating alternatives and proposing modifications to the model architecture, and also introducing a new set of input features to improve its performance in more complex scenarios, such as multi-carrier signals with narrow bandwidths.

Unlike the conventional Transformer, which comprises both encoder and decoder components, the proposed model focuses exclusively on the encoder. This choice is made because DPD needs strong feature extraction and regression modeling, not sequence generation, which is what the decoder usually handles.

The subsequent sections will provide a comprehensive description of the input features used and of the model's architecture, which is illustrated in 5.5, detailing the specific adaptations made and the various approaches tried.

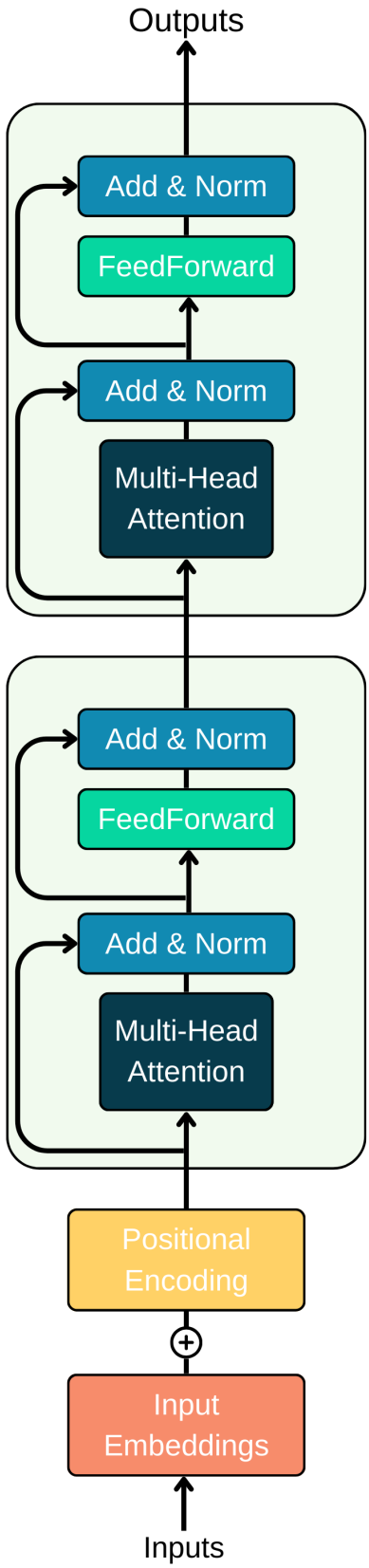

Figure 5.5: *Proposed Transformer-based NN Model architecture.*

5.2.1 Input Preprocessing

Before being fed into the NN, the input I and Q values are preprocessed and reshaped into a 2D matrix known as X. This matrix not only includes the standard I and Q components and envelope-dependent features from current and past signals, but incorporates future signal samples as well. Each column in the matrix corresponds to a specific feature, while each row represents different time steps within the sequence.

The preprocessed input is arranged as follows:

$$X = \begin{bmatrix} I_{\text{in}}(n-M) & Q_{\text{in}}(n-M) & |x(n-M)| & \cdots & |x(n-M)|^K & \frac{n-M}{N} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ I_{\text{in}}(n) & Q_{\text{in}}(n) & |x(n)| & \cdots & |x(n)|^K & \frac{n}{N} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ I_{\text{in}}(n+F) & Q_{\text{in}}(n+F) & |x(n+F)| & \cdots & |x(n+F)|^K & \frac{n+F}{N} \end{bmatrix} \tag{5.1}$$

In this matrix, $I_{\text{in}}(n)$ and $Q_{\text{in}}(n)$ represent the in-phase and quadrature components of the complex envelope $x(n)$ of the current input signal. The magnitude of the signal is denoted by $|x(n)|$. The terms $I_{\text{in}}(n-m)$, $Q_{\text{in}}(n-m)$, and $|x(n-m)|$ for m ranging from 1 to M represent historical samples, while $I_{\text{in}}(n+f)$, $Q_{\text{in}}(n+f)$, and $|x(n+f)|$ for f ranging from 1 to F denote future samples. M indicates the memory depth, or the number of past time steps considered, and F is the number of future time steps included. The nonlinearity order is denoted by K.

Including future samples in the matrix X along with past and current data has significantly improved the model's predictive accuracy. Any concerns about delays affecting real-time system performance are mitigated by the fact that these delays are quite minimal and remain within acceptable limits for real-time operations. Additionally, tests have shown that adding future samples enhances prediction more effectively than simply increasing the number of past samples.

Furthermore, the relative timestamp $\frac{n-i}{N}$ is included to provide temporal context, as the positional encoding discussed later on is found to be insufficient on its own, possibly due to the shorter sequence lengths and smaller d_{model} dimension used here compared to typical NLP tasks. Adding this timestamp feature has proven to boost the model's performance.

5.2.2 Input Embedding Layer

The next layer in the model is the input embedding layer, which employs a simple feed-forward network (FFN) to transform the input features, preparing the data for the subsequent layers of the model.

Initially, the preprocessed input sequence X consists of $K+3$ features, where K is the nonlinearity order, and the additional three features represent the I component, Q component, and the relative timestamp. The number of input features is denoted as $d_{\text{input}} = K+3$. The input embedding layer maps these d_{input} features to a higher-dimensional space required for the Transformer's multi-head attention mechanism. Specifically, it increases the number of features from d_{input} to d_{model}.

This transformation can be mathematically described as follows: the input embedding layer takes the input sequence X of size (T, d_{input}), where $T = M + F + 1$ is the number of time steps, and d_{input} is the number of features. The FFN within the input embedding layer processes each input feature vector to increase its dimensionality from d_{input} to d_{model}. Thus, the embedded input sequence E_{in} of size (T, d_{model}) is produced, where $E_{\text{in}} = \text{FFN}(X)$.

5.2.3 Positional Encoding

After the input embedding layer, the model applies positional encoding, which is the same as the one proposed in the original Transformer paper [15], to the embedded input sequence.

Mathematically, the positional encoding for a given position *pos* and dimension i is defined as follows:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{5.2}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{5.3}$$

where *pos* is the position in the sequence, i is the dimension index, and d_{model} is the dimensionality of the input embeddings. This results in a matrix that is the same size as the embedded input sequence E_{in}.

The positional encoding vectors are then added to the embedded input sequence to produce the final input to the model. This process can be described as:

$$E_{\text{in, pos}} = E_{\text{in}} + PE \tag{5.4}$$

where $E_{\text{in, pos}}$ is the positionally encoded input, E_{in} is the embedded input sequence, and PE is the positional encoding matrix.

Other methods for positional encoding have also been explored in this work. In particular, an absolute position encoding method tailored for time series data, known as time Absolute Position Encoding (tAPE), which was introduced in this paper [18], and also a concatenated positional encoding approach. However, neither method demonstrated a significant improvement in the model's performance.

It is important to note that relative positional encoding has not yet been explored in this context. Investigating this approach could be a promising direction for future work, potentially offering enhancements to the model.

5.2.4 Encoder

Following the positional encoding, the model employs two stacked Transformer encoder layers, identical to those described in the original Transformer architecture [15]. Each encoder layer consists of two main sub-layers: a multi-head self-attention mechanism and a position-wise FFN.

The encoder layer processes the positionally encoded input $E_{\text{in, pos}}$ by first applying the multi-head self-attention mechanism. This sub-layer enables the model to focus on different parts of the input sequence simultaneously by splitting the input into multiple "heads." Each head performs scaled dot-product attention independently, and the results from all heads are concatenated and linearly transformed. The output of the multi-head self-attention mechanism is then passed through a position-wise FFN, which consists of two linear transformations with a ReLU activation in between. Each sub-layer within the encoder is followed by a residual connection and layer normalization to stabilize training and improve gradient flow. Mathematically, the output of the encoder layer can be described as follows:

First, the multi-head self-attention mechanism is applied:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{5.5}$$

where each head is calculated as:

$$\text{head}_i = \text{Attention}(E_{\text{in, pos}}W_i^Q, E_{\text{in, pos}}W_i^K, E_{\text{in, pos}}W_i^V) \tag{5.6}$$

and W_i^Q, W_i^K, W_i^V, W^O are learned projection matrices.

Next, a residual connection and layer normalization are applied:

$$E_{\text{attn}} = \text{LayerNorm}(E_{\text{in, pos}} + \text{MultiHead}(Q, K, V)) \tag{5.7}$$

The position-wise FFN is then applied to the output of the self-attention sub-layer:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{5.8}$$

Another residual connection and layer normalization follow:

$$E_{\text{enc}} = \text{LayerNorm}(E_{\text{attn}} + \text{FFN}(E_{\text{attn}})) \tag{5.9}$$

The second encoder layer processes the output of the first encoder layer in the same manner. The output from the second encoder layer serves as the final encoded representation of the input sequence, ready for the output layer.

5.2.5 Output Layer

The final stage of the model is the output layer, which transforms the encoded representation of the input sequence into the desired output values. Specifically, this layer applies a linear transformation that reduces the input dimensionality to match the one of the output space,

which is 2, representing a single pair of I and Q values.

Mathematically, if E_{enc} is the output from the last encoder layer, the output layer applies the following transformation to produce the final output Y_{out}:

$$Y_{\text{out}} = E_{\text{enc}} W_{\text{out}} + b_{\text{out}} \tag{5.10}$$

Here, W_{out} is a learned weight matrix, and b_{out} is a bias term. The resulting Y_{out} consists of the predicted I and Q values for the entire input sequence.

5.3 Digital Pre-Distorter Performance Metrics

In the evaluation of DPD models, it is important to employ robust criteria to verify their accuracy and effectiveness. This section introduces the performance metrics used in this thesis to assess the efficacy of the compared models: normalized mean squared error (NMSE), and adjacent channel leakage ratio (ACLR). Additionally, the metrics used for evaluating the models' complexity are detailed.

5.3.1 Normalized Mean Squared Error

NMSE is used in this research to evaluate the accuracy of a model in predicting the ideal pre-distorted signal, following the iterative learning control (ILC) method.

The NMSE is calculated with the following formula:

$$\text{NMSE} = \frac{\sum_{i=1}^{N} (s_i - \hat{s}_i)^2}{\sum_{i=1}^{N} s_i^2} \tag{5.11}$$

where s_i are the measured I and Q samples of the ideal pre-distorted signal, and $\hat{s}_i$ are the I and Q samples predicted by the model, with N representing the total number of measurements within the specified bandwidth of interest.

When assessing the NMSE of a signal compared to the ideal target signal, this metric is presented in decibels (dB), using a logarithmic scale to express the power ratio. A lower NMSE value in dB indicates that the model more accurately predicts the ideal pre-distorted signal, showing minimal deviation from these target values.

5.3.2 Adjacent Channel Leakage Ratio

ACLR is an important metric for assessing how well PA linearization techniques like DPD are performing. ACLR calculates how much power from the PA leaks into nearby frequency channels relative to the power it sends out in its designated channel. This measure helps to ensure that the PA does not interfere with adjacent channels.

The ACLR is calculated using the formula:

$$\text{ACLR} = 10 \cdot \log_{10}\left(\frac{P_{adj}}{P_{main}}\right), \tag{5.12}$$

where P_{adj} represents the power in the adjacent channel and P_{main} is the power in the main channel.

By reducing ACLR, DPD models prove their ability to linearize the PA, making sure the output signal complies with stringent spectral regulations while maintaining high efficiency and performance. A visualization of ACLR on a signal's power spectral density (PSD) can be seen in Figure 5.6.

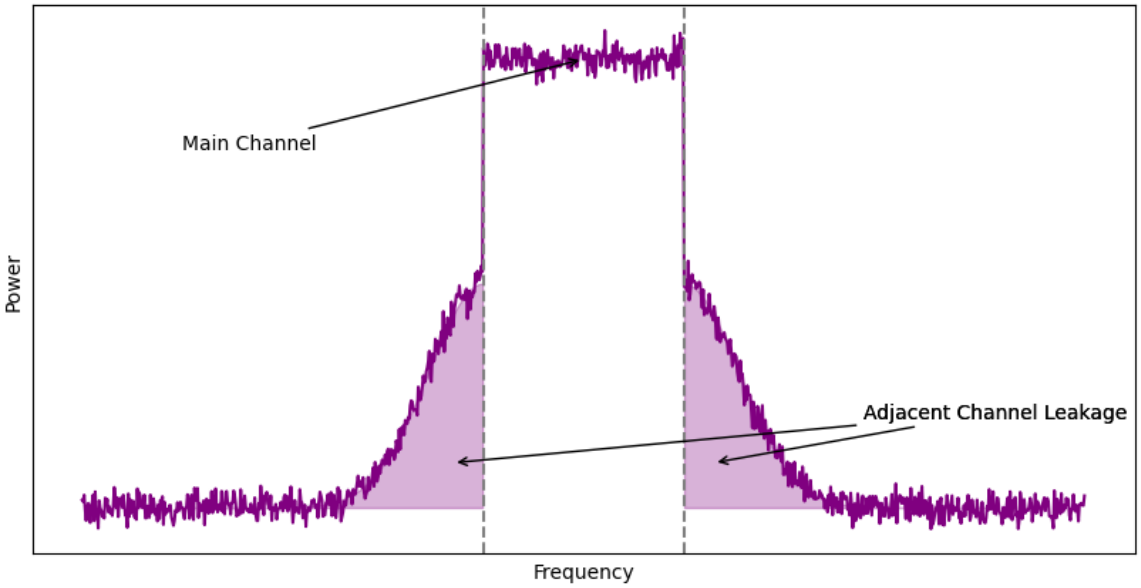

Figure 5.6: *This graph shows the power spectral density of a signal, displaying how power is distributed across different frequencies. The tallest peak represents the main channel, where most of the signal's power is concentrated. The shaded area to the sides of this peak highlights the leakage of power into frequencies next to the main channel.*

5.3.3 Complexity Metrics

The complexity of the models is analyzed using three main metrics: the number of parameters, the floating point operations per second (FLOPS), and multiply-accumulate operations (MACs).

The number of parameters serves as an indicator of the model's size, reflecting the total count of trainable elements within its architecture. Typically, a model with more parameters is considered more complex, potentially improving its accuracy in making predictions. However, this increased complexity also demands more computational power and memory, which can be problematic, especially when implementing these models in hardware such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs).

FLOPS measures the number of floating-point operations a model performs in one second, which helps to understand how efficiently a model can operate under various hardware conditions.

High FLOPS might indicate a model's higher performance capacity, but it also suggests greater power consumption and processing requirements, which are critical factors to deal with when it comes to finally deploy the models.

MACs count the number of multiply-and-accumulate operations performed by the model. This metric is particularly important for understanding the computational efficiency in neural network layers that involve dense matrix multiplications, providing a direct measure of the most frequent and computationally intensive operations in many deep learning architectures.

Chapter 6

Experimental Results

In this final chapter, the results of the thesis are presented, giving a detailed overview of the experimental outcomes. The core objective of the project was to benchmark the proposed Transformer-based neural network (NN) model against the existing digital pre-distorter (DPD) models currently in use at Ericsson. The results demonstrate that the Transformer model achieves results that are comparable, if not superior, in terms of normalized mean squared error (NMSE) and comparable in terms of adjacent channel leakage ratio (ACLR), depending on the scenarios outlined below. While the Transformer model shows promising performance, it does present a higher complexity compared to current Ericsson's models. In the next chapter, some ways to reduce the model's complexity are proposed, although these suggestions are meant for future research and go beyond the scope of this thesis.

Due to confidentiality agreements, specific data and comparisons with Ericsson's proprietary DPD models are not disclosed in this thesis. Instead, the focus here is on a comparison of the Transformer model against other recognized recurrent neural network (RNN) architectures.

6.1 Experimental Setup

The following experiments are designed to compare the proposed Transformer-based NN model with several other NNs, specifically a Long Short-Term Memory Neural Network (LSTMNN), a Bidirectional LSTM Neural Network (BiLSTMNN), and a Bidirectional Gated Recurrent Unit Neural Network (BiGRUNN). Each model is built, trained, and tested using the PyTorch framework, with performance metrics evaluated through MATLAB functions accessed via the MATLAB Python API. The method used to identify the DPD models parameters is the iterative learning control (ILC) method. Performance evaluation is conducted using NMSE and ACLR metrics. A NN-developed power amplifier (PA) model serves as a standard to evaluate the linearization capabilities of each DPD model under examination.

All the signal data used in this thesis follows the Wideband Code Division Multiple Access (WCDMA) modulation scheme and are obtained from Ericsson's radio laboratory, where a PA was tested under different conditions. This study considers three distinct test scenarios, each involving different configurations of carrier count and bandwidth:

- In the first scenario, data were captured from a system configured with four carriers, each having a bandwidth of 20 MHz.
- The second scenario involved four carriers, each with a bandwidth of 10 MHz.
- The third scenario used four carriers with bandwidths of 5 MHz each.

These configurations are chosen to evaluate the performance and adaptability of the models under varied and incrementally challenging conditions. Each dataset used is composed of 524288 I/Q samples, which are split into training and testing sets and normalized to ensure consistency in input data scale and to help in the convergence of the NNs during training.

In the following section, the specific settings and adjustments made to optimize the performance of the proposed model and the other NNs are detailed.

6.2 Models Configuration and Hyperparameters tuning

For the Transformer-based NN model developed in this thesis, each hyperparameter was manually tuned to optimize the model's performance. The memory depth M is set at 19, therefore including 19 past time steps along with the current one, providing a wide historical context. The inclusion of future samples is limited to 2 (denoted as F), allowing the model to have insights about the upcoming signal states while ensuring there's no significant delay that could affect real-time operations. The nonlinearity order K is set at 2, as higher values were tested but did not improve the model's efficiency. These input features' settings are found to offer the best performance for this specific model.

The dimension of the model's embeddings, d_{model}, is set to 16. This small size helps keep the model efficient while still allowing it to properly understand the necessary data relationships. The attention mechanism within the transformer utilizes 8 heads (n_{heads}). The feedforward network within each encoder layer has a dimension of 16. To prevent overfitting, a dropout rate of 0.1 is used in the encoder layers of the transformer.

The model undergoes extensive training over 500 epochs, using a DataLoader with a batch size of 128, configured to shuffle the training dataset, to increase model robustness and prevent the learning process from being biased by the order of data. An initial learning rate of 8×10^{-4} is applied, with a learning rate scheduler reducing this rate by 0.95 every 5 epochs to refine the model's weight adjustments progressively. The Adam optimizer is chosen, and mean squared error loss (MSE) is utilized as the performance criterion during the training process.

Concerning the models constructed for comparison, the BiLSTMNN and the BiGRUNN are designed to handle the same input data as the Transformer-based NN, consisting of I/Q samples with a memory depth of $M = 19$, a future depth of $F = 2$, and a nonlinear degree of $K = 2$. The LSTMNN has been deprived of the future samples in the input matrix, since its architecture is not suited for handling future data. Each of these models incorporates an RNN layer, followed by two fully connected (FC) layers. To clearly demonstrate how different RNN architectures affect model performance and complexity, all models are configured with the same inner and FC layer sizes, which are tuned to balance complexity and performance. Specifically, the inner size for each model is set at 30, with the FC layers sized 20 and 10, respectively.

This standardized approach not only allows for a direct comparison of the different RNN layers—whether standard LSTM, bidirectional LSTM, or bidirectional GRU—but also ensures that differences in performance are directly attributable to the RNN architecture itself, not variations in network size.

Each of these models is also trained over 500 epochs using a DataLoader configured with a batch size of 1024. The Adam optimizer is used across all models and the initial learning rate

is set at 4×10^{-4}, with a learning rate scheduler that reduces this rate by a factor of 0.95 every 5 epochs to fine-tune the model parameters gradually. The MSE loss is chosen as the criterion.

Table 6.1 better shows the comparison of hyperparameters across all the different models.

Table 6.1: Comparison of Hyperparameters Across Different Models

Model	**M**	**K**	**F**	**d_model**	**hidden size**	**Heads/Layers**	**FC Dim.**
Transformer NN	19	2	2	16	-	8 heads	16
LSTMNN	19	2	-	-	30	LSTM + 2 FC	20, 10
BiLSTMNN	19	2	2	-	30	BiLSTM + 2 FC	20, 10
BiGRUNN	19	2	2	-	30	BiGRU + 2 FC	20, 10

6.3 Complexity Comparison

The complexity of the models is analyzed using three main metrics: the number of parameters, the floating point operations per second (FLOPS), and multiply-accumulate operations (MACs).

As shown in Figure 6.1, the BiLSTMNN model is the one with the highest number of parameters, about 10.09 thousand, due to its complex bidirectional network structure. The BiGRUNN follows, also having bidirectional layers, but utilizing fewer parameters—about 7.93 thousand—thanks to the simplified gating mechanisms of the GRU unit compared to the LSTM one. The LSTMNN model has a smaller configuration with approximately 5.17 thousand parameters, while the Transformer-based NN has the least, with just 3.5 thousand parameters.

Figure 6.2 provides more detail on the computational demands of these models through FLOPS, which measure the number of executable operations. The BiLSTMNN model, with its higher parameter count, also leads in this metric with 341.27 thousand FLOPS, showing the direct correlation between the number of parameters and FLOPS. Following this pattern, the BiGRUNN results in 256.07 thousand FLOPS, while the LSTMNN has 170.87 thousand FLOPS. Despite remaining the lighter model, the Transformer-based NN still requires 145.79 thousand FLOPS, highlighting the complexity of its self-attention mechanisms that, although parameter-efficient, are computationally dense.

In Figure 6.3, the MACs of each model are shown. The LSTMNN model registers the lowest MACs at 0.82 thousand, while the BiLSTMNN and BiGRUNN models show an identical MAC count of 1.42 thousand. Most notable is the Transformer model, which, despite having the fewest parameters, demonstrates a substantially higher MAC count at 69.37 thousand. This emphasizes the computational density of the Transformer's self-attention mechanisms. While these mechanisms afford it significant gains in modeling capabilities, especially in tasks involving complex sequence dependencies, they also result in a much higher requirement for computational operations.

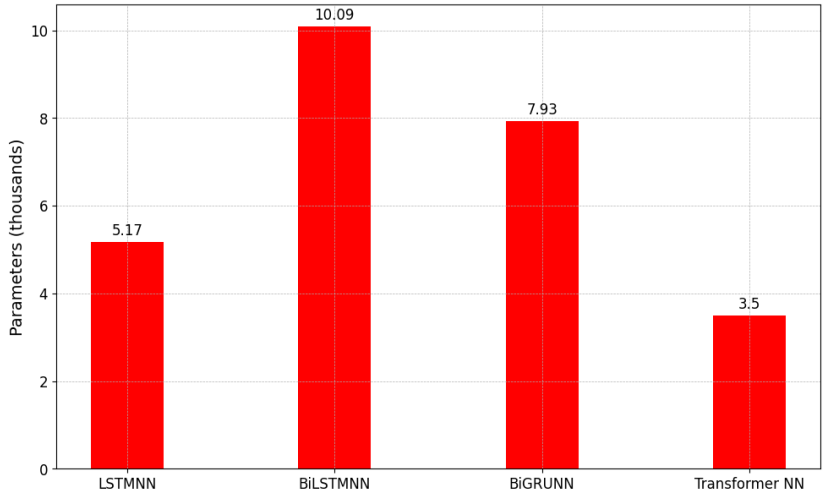

Figure 6.1: *Parameters of different NN models.*

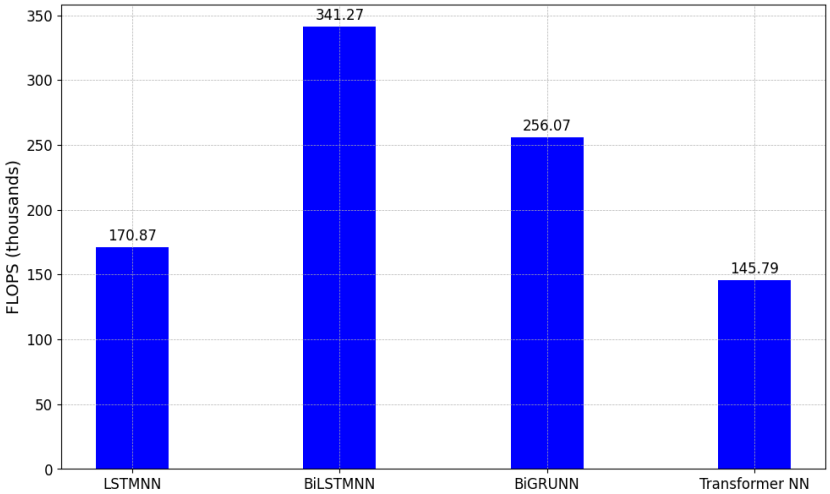

Figure 6.2: *FLOPS of different NN models.*

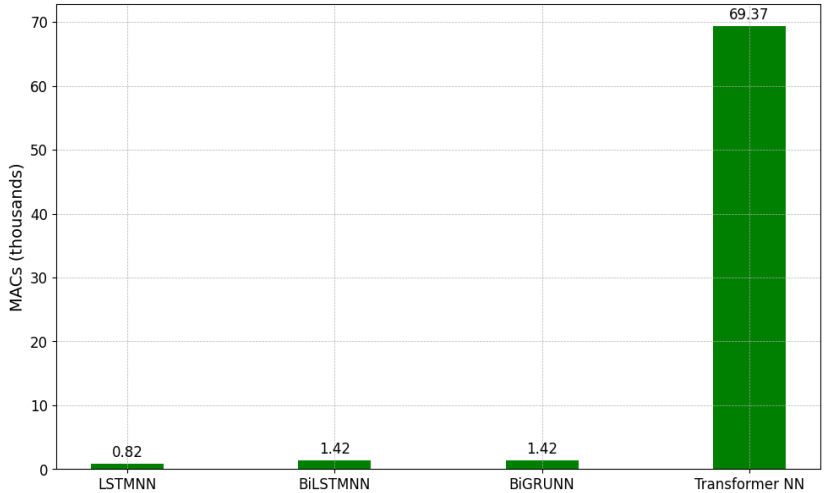

Figure 6.3: *MACs of different NN models.*

6.4 Performance Evaluation

In this section, the results from the modelling of NN DPDs using multi-carrier signals are discussed. As previously stated in Section 5.1, the ILC framework allows assessing the models' performances just by comparing the predicted signals against the ideal pre-distorted signal. Therefore, in the following sections, the models are compared using the NMSE metric. A lower NMSE directly corresponds to a lower ACLR, and vice versa. Moreover, constructing a PA model that truly reflects the behaviour of the real component is not an easy task; therefore, the PA model used in this thesis serves the only purpose of giving a qualitative overview of each model's ability to prevent the output signal from leaking into adjacent channels. However, an ACLR value for the Transformer model is included for completeness, although it may be somewhat biased by the PA model.

6.4.1 Four Carriers with 20 MHz Bandwidth

In this first scenario, the signal configuration involves four carriers, each with a bandwidth of 20 MHz. Here, the Transformer-based NN model is the one which shows the best results in terms of NMSE, achieving -36.9313 dB as recorded in Table 6.2. The BiLSTMNN model also performed quite well, while both the LSTMNN and the BiGRUNN resulted in very low NMSE values.

Table 6.2: NMSE of Different Models for the Four Carriers, 20 MHz scenario.

Models	**NMSE (dB)**
LSTMNN	-32.4769
BiLSTMNN	-36.1734
BiGRUNN	-33.7502
Transformer NN	**-36.9313**

The first plot, shown in Figure 6.4, presents the Power Spectral Density (PSD) of the predicted pre-distorted signals by each model, highlighting how each model's output compares to the ideal one. The second plot, Figure 6.5, illustrates the output of the PA when fed with these pre-distorted signals, showing the effectiveness of each model in reducing spectral emissions and controlling signal leakage.

The ACLR measured for the Transformer NN in this scenario is -47.46 dB.

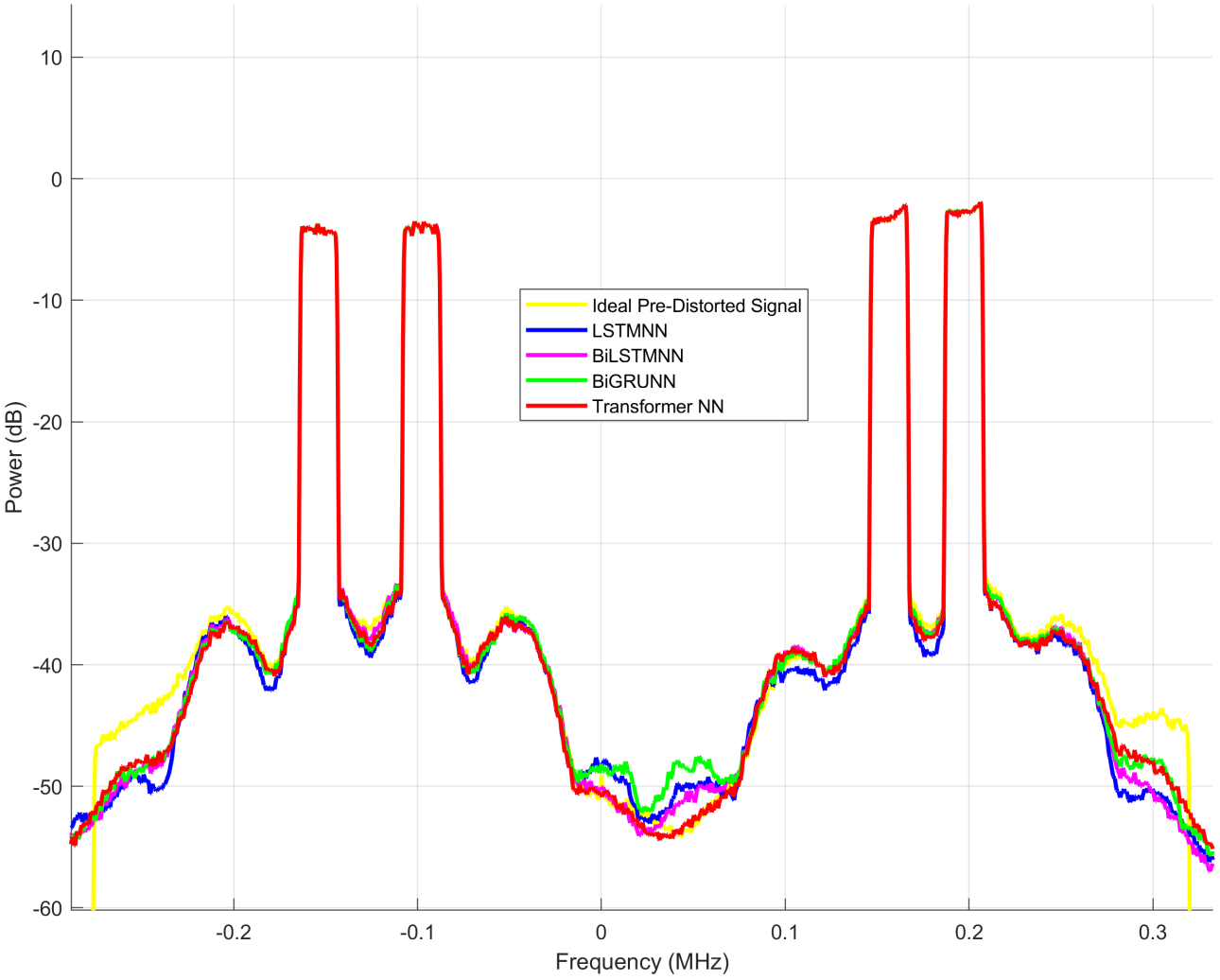

Figure 6.4: *PSD comparison of the predicted pre-distorted signals by LSTMNN, BiLSTMNN, BiGRUNN, and Transformer NN models against the ideal signal obtained via ILC for the Four Carriers, 20 MHz scenario.*

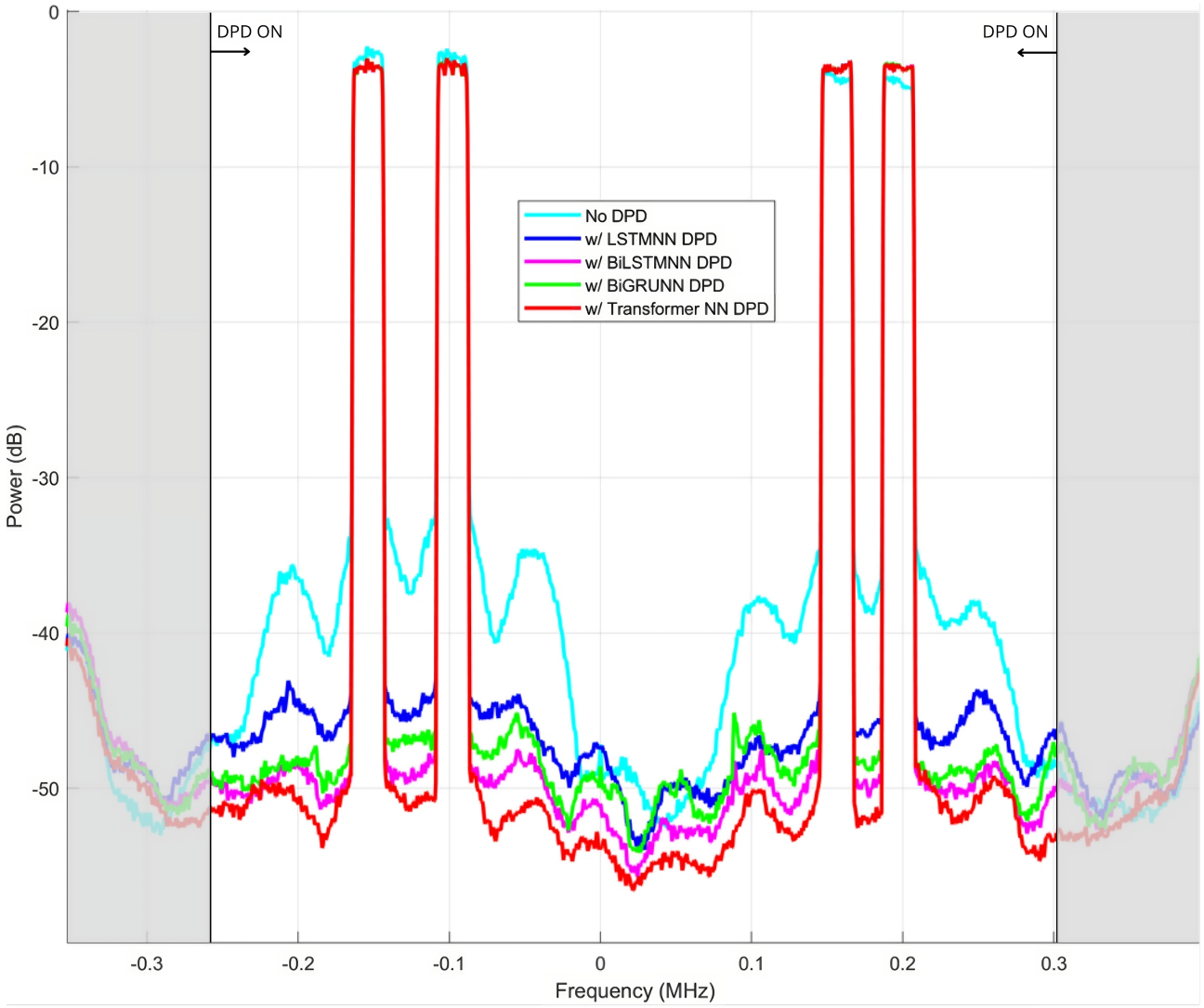

Figure 6.5: *PSD of PA outputs processed by DPD models compared to the untreated signal (No DPD) for the Four Carriers, 20 MHz scenario. The shadowed areas indicate bandwidth regions not addressed by DPD models.*

6.4.2 Four Carriers with 10 MHz Bandwidth

In the scenario involving four carriers each with a 10 MHz bandwidth, the Transformer-based NN continues to demonstrate superior performance, with a NMSE of -36.8773 dB, as detailed in Table 6.3. The rest of the models give similar results to the previous case.

Here, only the final PA output signals are shown, reducing the redundancy of showing the predicted pre-distorted signals. The ACLR achieved for this case by the Transformer NN is -48.47 dB.

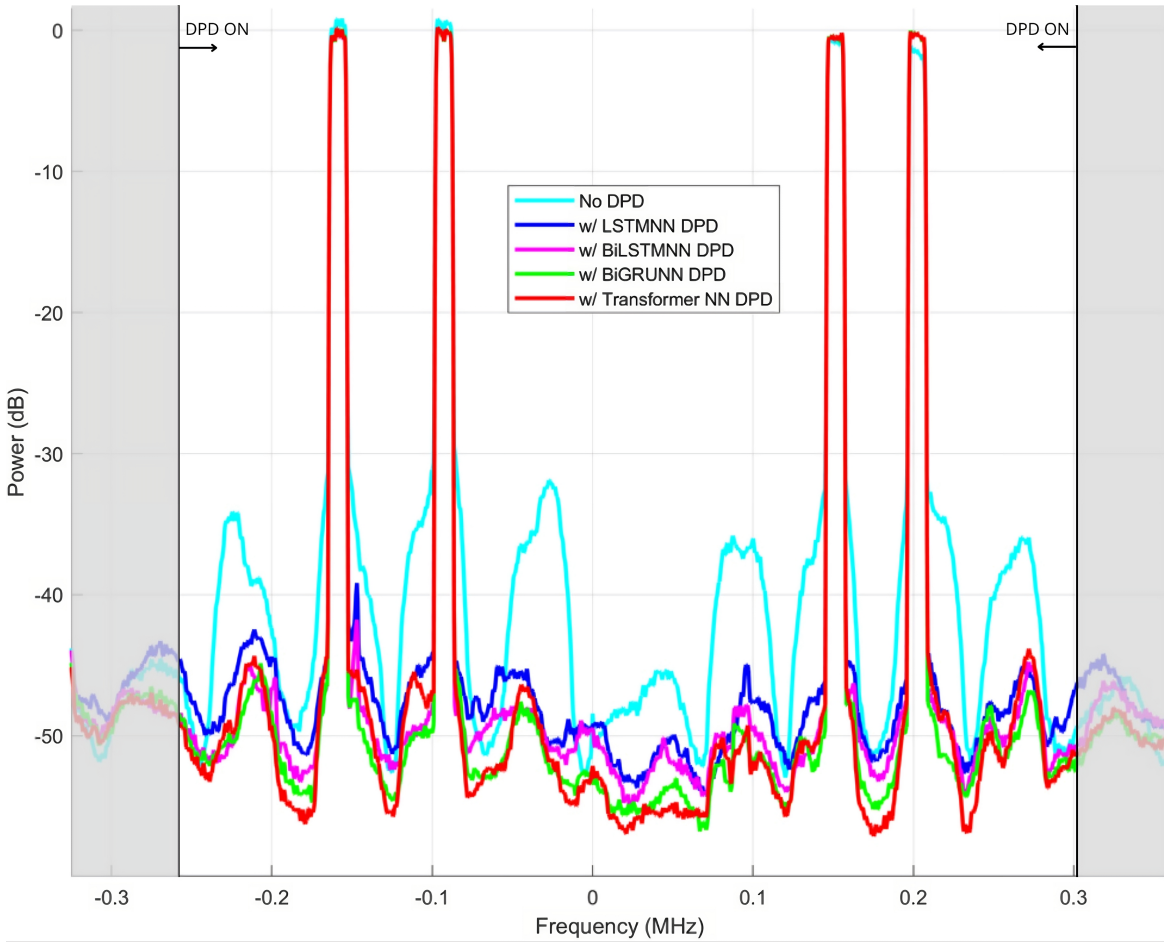

Figure 6.6: *PSD of PA outputs processed by DPD models compared to the untreated signal (No DPD) for the Four Carriers, 10 MHz scenario. The shadowed areas indicate bandwidth regions not addressed by DPD models.*

Table 6.3: NMSE of Different Models for the Four Carriers, 10 MHz scenario.

Models	**NMSE (dB)**
LSTMNN	-33.9786
BiLSTMNN	-35.9613
BiGRUNN	-33.7909
Transformer NN	**-36.8773**

6.4.3 Four Carriers with 5 MHz Bandwidth

In the last testing scenario, which involves four carriers each with a 5 MHz bandwidth, the Transformer-based NN is once again the top performing in terms of NMSE with -37.5514 dB, as outlined in Table 6.4. Here also the BiGRUNN and the BILSTMNN show good results, with a NMSE of -37.0551 and -36.8183, respectively. The measured ACLR value for the Transformer NN is -49.87 dB.

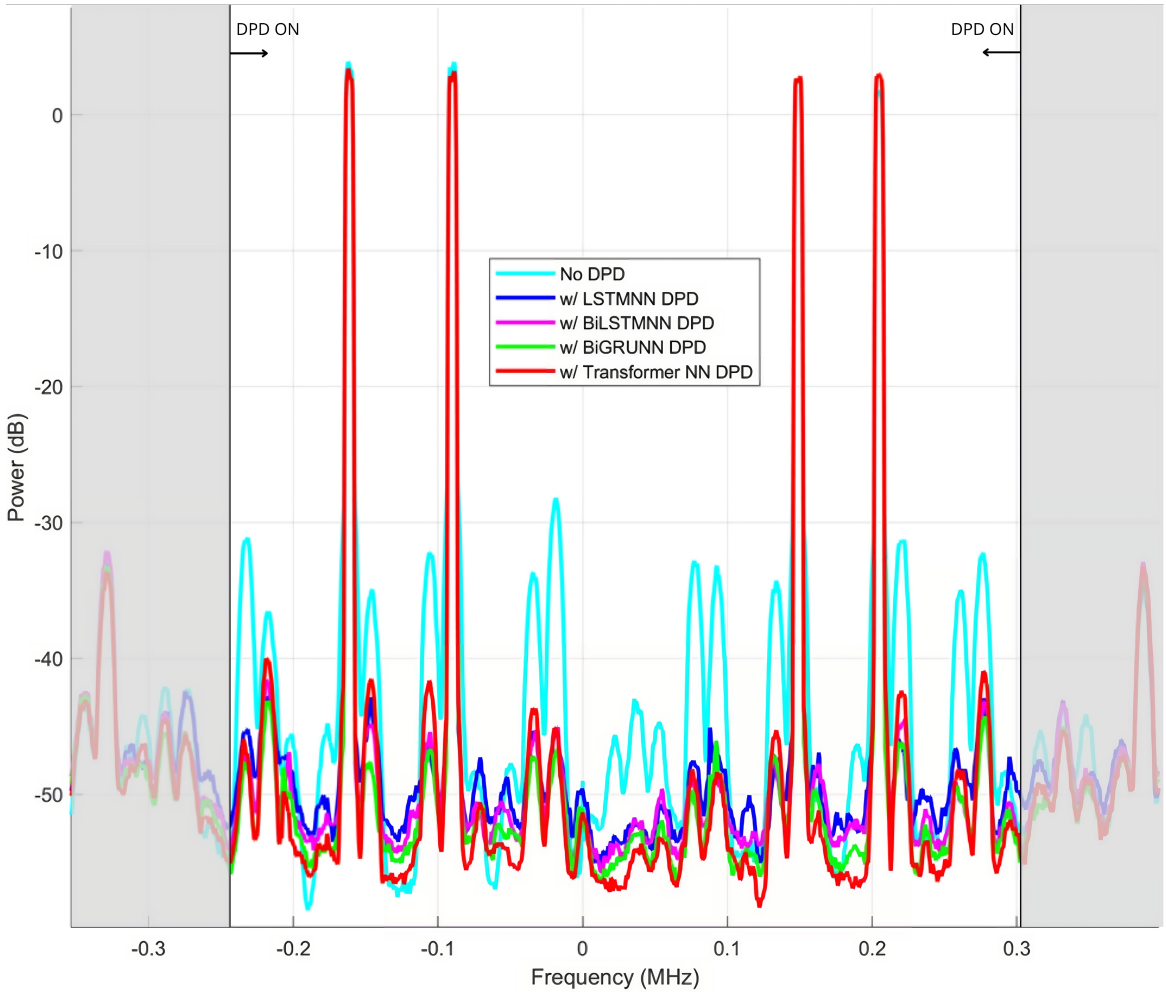

Figure 6.7: *PSD of PA outputs processed by DPD models compared to the untreated signal (No DPD) for the Four Carriers, 5 MHz scenario. The shadowed areas indicate bandwidth regions not addressed by DPD models.*

Table 6.4: NMSE of Different Models for the Four Carriers, 5 MHz scenario.

Models	**NMSE (dB)**
LSTMNN	-35.8598
BiLSTMNN	-36.8183
BiGRUNN	-37.0551
Transformer NN	**-37.5514**

Chapter 7

Conclusion and Future Works

This thesis has expanded the investigation of Transformer-based architectures for digital predistortion (DPD) in complex multi-carrier signal environments. The experimental results revealed that the proposed Transformer-based model generally shows superior performance to traditional neural network (NN) architectures such as Long Short-Term Memory Neural Network (LSTMNN), Bidirectional LSTM Neural Network (BiLSTMNN), and Bidirectional Gated Recurrent Unit Neural Network (BiGRUNN) in terms of normalized mean square error (NMSE) and consequently of adjacent channel leakage ratio (ACLR). The Transformer-based model also showed comparable performance to Ericsson's proprietary DPD models, confirming its effectiveness and demonstrating that it could be relevant for industrial use.

This thesis also evaluated different ways to improve the encoding of the sequence position into the input embeddings. In particular, it assessed an absolute position encoding technique known as time Absolute Position Encoding (tAPE), introduced in [18] where it showed to be effective for time series data with low d_{model} values, and a concatenated positional encoding approach. However, neither method demonstrated a significant improvement in the model's performance. The addition of the relative timestamp to the input matrix has instead led to learning improvements, providing additional temporal context to the network. Future studies exploring relative positional encodings could potentially further boost the model's effectiveness.

Regarding complexity, the Transformer-based model exhibits both a lower number of parameters and fewer floating point operations per second (FLOPS) compared to other NNs evaluated. However, it demonstrates a significantly higher number of multiply-accumulate operations (MACs), highlighting the intensive computational demands of the Transformer's self-attention mechanisms, which, despite its parameter efficiency, require extensive computational resources. As a result, it currently still exceeds the complexity suitable for practical FPGA and ASIC implementation, mainly due to the high energy consumption and processing power required by this large number of MACs. Future studies should focus on simplifying the Transformer model without sacrificing performance, exploring techniques such as pruning, quantization, and knowledge distillation to reduce the model's computational load. Additionally, it should be noted that the high number of memory taps used as inputs to the evaluated NNs in this thesis is particularly high, due to the low sampling rate of the signals used for training. Reducing these high values is therefore a simple first step to reduce the MACs required by the model.

Bibliography

[1] Dimuthu Lesthuruge. "Transformer NN-based behavioral modeling and predistortion for wideband pas". MA thesis. D. Lesthuruge, 2023.

[2] Deepmala Phartiyal and Meenakshi Rawat. "LSTM-Deep Neural Networks based Predistortion Linearizer for High Power Amplifiers". In: *2019 National Conference on Communications (NCC)*. 2019, pp. 1–5. DOI: 10.1109/NCC.2019.8732178.

[3] Jinlong Sun et al. "Behavioral Modeling and Linearization of Wideband RF Power Amplifiers Using BiLSTM Networks for 5G Wireless Systems". In: *IEEE Transactions on Vehicular Technology* 68.11 (2019), pp. 10348–10356. DOI: 10.1109/TVT.2019.2925562.

[4] Huilan Wu et al. "A Lightweight Deep Neural Network for Wideband Power Amplifier Behavioral Modeling". In: *2021 7th International Conference on Computer and Communications (ICCC)*. 2021, pp. 1294–1298. DOI: 10.1109/ICCC54389.2021.9674651.

[5] Joel Vuolevi and Timo Rahkonen. *Distortion in RF power amplifiers*. Artech house, 2003.

[6] Pere L Gilabert et al. "Machine learning for digital front-end: a comprehensive overview". In: *Machine Learning for Future Wireless Communications* (2020), pp. 327–381.

[7] Allen Katz, John Wood, and Daniel Chokola. "The Evolution of PA Linearization: From Classic Feedforward and Feedback Through Analog and Digital Predistortion". In: *IEEE Microwave Magazine* 17.2 (2016), pp. 32–40. DOI: 10.1109/MMM.2015.2498079.

[8] Kelly Mekechuk et al. "Linearizing power amplifiers using digital predistortion, eda tools and test hardware". In: *High Frequency Electronics* 3.4 (2004), pp. 18–25.

[9] Jungsang Kim and Konstantinos Konstantinou. "Digital predistortion of wideband signals based on power amplifier model with memory". In: *Electronics Letters* 37 (2001), pp. 1417–1418. URL: https://api.semanticscholar.org/CorpusID:111026212.

[10] Dennis R Morgan et al. "A generalized memory polynomial model for digital predistortion of RF power amplifiers". In: *IEEE Transactions on signal processing* 54.10 (2006), pp. 3852–3860.

[11] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural Networks* 4 (1991), pp. 251–257. URL: https://api.semanticscholar.org/CorpusID:7343126.

[12] Dongming Wang et al. "Augmented Real-Valued Time-Delay Neural Network for Compensation of Distortions and Impairments in Wireless Transmitters". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.1 (2019), pp. 242–254. DOI: 10.1109/TNNLS.2018.2838039.

[13] S KUMAR, Vergin M, and L. Anbarasi. "Predictive Analytics of COVID-19 Pandemic: Statistical Modelling Perspective". In: *Walailak Journal of Science and Technology (WJST)* 18 (Aug. 2021). DOI: 10.48048/wjst.2021.15583.

[14] Jeblad. https://commons.wikimedia.org/w/index.php?curid=66225938. [Image]. Accessed on [access date]. Available under a Creative Commons Attribution-ShareAlike License (CC BY-SA 4.0).

[15] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[16] Google Developers. *Image from Google Developers Documentation*. https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space. Licensed under the Creative Commons Attribution 4.0 License. 2023.

[17] Jessica Chani-Cahuana et al. "Iterative Learning Control for RF Power Amplifier Linearization". In: *IEEE Transactions on Microwave Theory and Techniques* 64.9 (2016), pp. 2778–2789. DOI: 10.1109/TMTT.2016.2588483.

[18] Navid Mohammadi Foumani et al. "Improving position encoding of transformers for multivariate time series classification". In: *Data Mining and Knowledge Discovery* 38.1 (Sept. 2023), pp. 22–48. ISSN: 1573-756X. DOI: 10.1007/s10618-023-00948-2. URL: http://dx.doi.org/10.1007/s10618-023-00948-2.