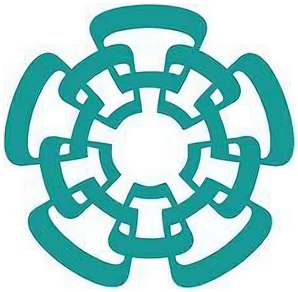

Centro de Investigación y de Estudios Avanzados del I.P.N

Unidad Guadalajara

Exploring Deep Learning Techniques for Equalizing Doubly Dispersive Channels

A thesis presented by:

Luis Emilio Tonix Gleason

to obtain the degree of:

Master in Science

in the subject of:

Electrical Engineering

Thesis Advisors:

Dr. Ramón Parra Michel

Dr. Fernando Peña Campos

Guadalajara, Jalisco April 15, 2023

Acknowledgment

Thank you for reading this; I appreciate it. I'm expecting that your work will benefit greatly from this material. I won't hesitate to say that you might have done your job more effectively than I did. Progress in science is a way to skepticism and having the best information from multiple sources to meet your individual standards.

Resumen

En la actualidad, la tecnología de redes de comunicación inalámbricas de alta movilidad está en constante evolución, y tiene aplicaciones en diversos ámbitos, tales como la seguridad vial, la conducción autónoma, el monitoreo remoto de vehículos, el vuelo de drones y los requisitos de 6G. Los transmisores RF, que se emplean en los sistemas de telecomunicaciones para transmitir información, codifican dicha información a través de la amplitud y la fase de una señal portadora, comúnmente conocida como datos IQ. Sin embargo, al viajar a través de un canal de alta movilidad, estos datos sufren distorsión, debido a la dispersión doble de dicho canal, lo que implica que las propiedades de la señal se ven afectadas por el desplazamiento, difusión en el tiempo y la frecuencia. Aunque existen algoritmos tradicionales que requieren ecualizadores iterativos complejos, que pueden ser lentos en términos de tiempo de ejecución pero precisos, también existen ecualizadores más sencillos, aunque menos precisos. El propósito de este trabajo es examinar diversas soluciones basadas en redes neuronales para la tarea de equalización en el procesamiento de señales. El enfoque principal es encontrar un equilibrio óptimo entre la complejidad temporal de la solución y su rendimiento en la tarea de equalización. Para lograr este objetivo, se seguirá una metodología que involucra la evaluación de una serie de redes neuronales organizadas desde las más sencillas hasta las más complejas. Cada red neuronal será evaluada en términos de su rendimiento y complejidad para compararlas adecuadamente y determinar los compromisos necesarios para su implementación.

Es posible afirmar que los algoritmos convencionales han demostrado ser eficaces en la tarea de equalización en esquemas de modulación como LTE y Wi-Fi. Sin embargo, los recientes avances en redes neuronales han simplificado problemas que antes eran intratables, como el descifrado de secuencias de ADN o la creación de imágenes a partir de texto. En resumen, abordan con éxito problemas no lineales. El objetivo de este trabajo es imitar técnicas de ecualización conocidas con naturaleza lineal, como los ecualizadores Zero Forcing, MSE, LMMSE y no lineales como OSIC y NearML. Durante las pruebas que se realizaron, se hizo un seguimiento de diversos tamaños de constelaciones y tasas de error de bits en una variedad de situaciones con interferencia y ruido. En los resultados de la investigación experimental, se demostró que la tasa de error de bits(BER) y la tasa de bloques (BLER) disminuyó y se acercó o incluso superó los modelos de oro, que son los ecualizadores mencionados anteriormente.

Los últimos enfoques en investigación literaria han demostrado diversas estrategias que se aproximan a lograr una buena ecualización con redes nueronales. Sin embargo, estos enfoques se basan en métodos tradicionales y no cubren todas las etapas de la ecualización, además de tener escenarios relativamente simples en comparación con el canal que se estudiará en este trabajo. Por lo tanto, se abordan incluso condiciones tales como canales de línea de vista o la falta de ella. Por último, pero no menos importante, la presentación de los métodos desarrollados en esta investigación busca establecer un punto de partida para futuras investigaciones en un campo de la comunicación prácticamente inexplorado por algunos equipos.

Abstract

Due to their primary uses in control, road safety, autonomous driving, remote monitoring of vehicles, drone flying, and 6G needs, the development of high-mobility wireless communication networks is cutting-edge technology. The RF broadcasts, which are used by telecommunications systems to transfer encoded information over the amplitude and phase of a carrier signal, are normally called IQ data. However, this data is distorted as it travels through a high mobility channel. High mobility channels are doubly dispersive, which means that the signal properties are mixed with some temporal frequency shifting and spreading. There are some traditional algorithms that may require complex iterative equalizers that can be slow in terms of execution time. However, there are also simpler equalizers that may not be as precise. The aim of this study is to investigate different neural network-based solutions for signal processing equalization. The primary objective is to achieve an optimal balance between the solution's temporal complexity and its performance in equalization. To attain this objective, we will follow a methodology that entails evaluating a range of neural networks arranged from simple to complex. Performance and complexity will be assessed for each neural network to facilitate a fair comparison and determine the required trade-offs for implementation.
Conventional algorithms have demonstrated their effectiveness in the equalization task for modulation schemes like LTE and Wi-Fi. However, recent advances in neural networks have simplified problems that were previously intractable, such as deciphering DNA sequences or creating images from text. In short, they successfully tackle non-linear issues. The objective of this work is to mimic well-known equalization techniques with linear properties such as Zero Forcing, MSE, and LMMSE, as well as non-linear techniques like OSIC and NearML equalizers. Various constellation sizes and bit error rates were monitored during the tests conducted in various scenarios with interference and noise. The experimental results indicated that the bit error rate (BER) and block error rate (BLER) decreased and approached, or even exceeded, the gold standard models of equalizers mentioned earlier.

Recent studies in literature have proposed multiple strategies to achieve effective equalization with neuronal networks. However, these methods rely on traditional approaches and fail to cover all aspects of equalization, and their scenarios are relatively simpler than the channel under consideration in this work. As a result, we investigate conditions such as line of sight channels or their absence. Finally, this study aims to provide a starting point for future investigations in a field of communication that remains unexplored by

some teams.

Contents

1 Introduction

The main goal of this chapter is to provide an overview of the technical aspects necessary to understand the problem statement of this study.

1.1 Document organization

The structure of this document is as follows:

- **Chapter 1 Theoretical Framework:** Fundamental ideas and concepts behind the current project. Outlines the core concepts necessary to comprehend this study.
- **Chapter 2 State of Art:** This section examines recent research on equalization using deep learning.
- **Chapter 3 Methodology:** This work presents a Python implementation of various models for training and testing different experiments, along with an overview of the software architecture that enables these experiments. The primary focus is on developing explainable deep learning models for Quadrature Amplitude Modulation (QAM) equalization using neural networks, including the design of the proposed models. Additionally, each network's features are described, such as preprocessing, number of parameters, hyperparameters, and references to the state-of-the-art that support certain development stages.
- **Chapter 4 Results and Analysis:** In this section, the conception and execution of the idea, the experiments conducted, and the analysis of the results are covered. A comparison of the studied models is provided in graphs and tables, with a focus on their bit error rate (BER), block error rate (BLER), floating-point operations per second (FLOPS), and complexity.
- **Chapter 5 Conclusion and Outlook:** A conclusion is drawn based on the results, contributions, and potential future directions of this study.

1.2 Background

It takes a lot of expertise in the field of communications to model different types of channels, account for various channel flaws, and create the best signaling and detecting systems that guarantee a reliable data flow. The design of transmit signals in communications enables simple analytical techniques for symbol detection for a range of channel and

system models, including multipath, doppler spread, and white Gaussian noise (AWGN) over constellation symbol detection. [2]

One of the guiding principles of communication system design is the division of signal processing into a series of distinct blocks, each of which performs a clearly defined and isolated functions such as source or channel coding, modulation, channel estimation, and equalization. [45].

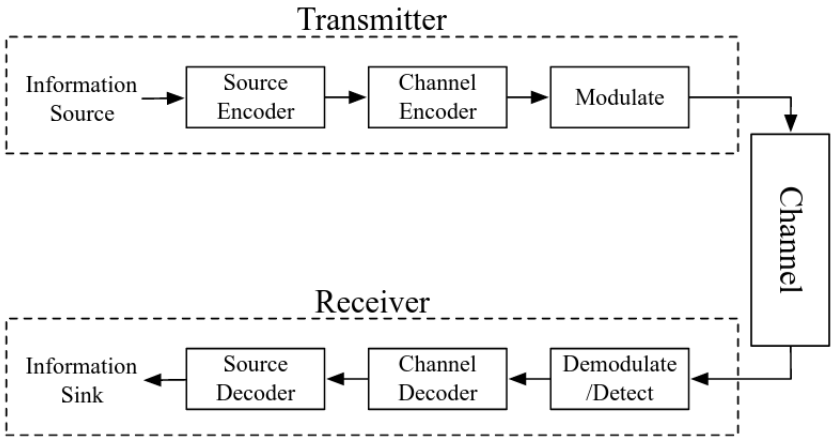

Figure 1: Typical communication system block diagram [14]

In this study, our main focus will be on the equalization stage. Our team has previously completed a project on channel estimation and equalization using classical QR methods, and we have also implemented an FPGA-based equalization stage for the 802.11p standard [27]. The 802.11p standard is commonly used for V2V (Vehicle-to-Vehicle) communication with the OFDM modulation scheme. [19]. Therefore, the data is processed with a size of 48 symbols, resulting in a 48x48 channel. As a result, this work does not prioritize the channel estimation component. Instead, the main objective is to cancel the effects of the received signal through the channel and noise, which is well-known due to the previous research.

By employing neural networks, we aim to generalize a pseudo-inverse non-linear system that cancels out channel and noise effects. However, it is important to note that the matrix inverse is not always well-defined, and its time complexity is $O(N^3)$. The pseudo-inverse is a well-known technique used to solve matrix inversion problems when the matrix is not invertible or singular[59]. Neural networks have been demonstrated to approximate any function [23], and current research has demonstrated an astounding aptitude for algorithmic learning [55]. Due to the challenge of defining real-world images or language with strict mathematical models, deep learning (DL) excels in fields like computer vision [25]

and natural language processing[64].

1.3 OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is a digital communication technique that divides the available bandwidth into a large number of narrowband subcarriers and transmits data by modulating the subcarriers with symbols. OFDM is widely used in wireless and wired communication systems, such as Wi-Fi, LTE, and DOCSIS, which are described by well-known standards, as referenced follows [40][1][11]

The basic blocks of an OFDM system are:

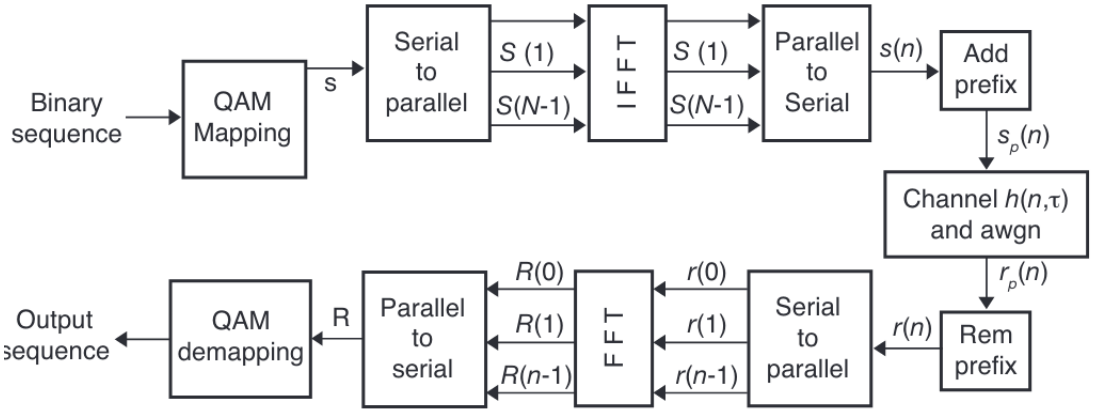

Figure 2: OFDM basic diagram [38]

- **Binary sequence**: This block generates the data to be transmitted. The data is usually a sequence of bits.
- **QAM mapping**: Bits are grouped into blocks called symbols usally named as alphabet $\mathbb{A}$. We have a set of bits $2^{\mathbb{A}}$ grouped inside the alphabet.
- **IFFT block**: This block applies an Inverse Fast Fourier Transform (IFFT) to the modulated symbols, dividing them into a set of subcarriers.
- **Cyclic prefix**: Helps OFDM symbols to reduce inter-symbol interference.

At regular intervals, the transmitter inserts recognizable symbols known as "pilots" into the transmitted signal. These pilots are selected to have a known value and to be separated from one another by a given number of symbols. This estimation involves analyzing the correlations between the received signal and the known pilot symbols, and then using this information to estimate the channel response. Using the estimated channel response, the receiver then performs channel equalization to remove the distortions caused by the communication channel from the received signal [30], and also there are more complex methods that can handle delay-time channel estimation in an efficient embbeded pilot [49].

1.3.1 Advantages and Disadvantages in OFDM

The advantages offered by OFDM systems in broadband systems are as follows [67]:

- **High spectral efficiency**: OFDM can transmit a large amount of data over a wide frequency band by dividing the available bandwidth into multiple narrowband subcarriers.
- **Robustness to channel impairments**: OFDM is less sensitive to frequency-selective fading and interference than other multiplexing techniques, making it well-suited for use in wireless communication systems.
- **Ease of implementation**: OFDM can be implemented using simple digital signal processing techniques, making it relatively easy to design and implement.

Some disadvantages of OFDM include:

- **High peak-to-average power ratio**: OFDM signals have a high peak-to-average power ratio, which can cause problems in power amplifier systems and limit the range of the transmitted signal.
- **Sensitivity to timing errors:** OFDM is sensitive to timing errors, which can cause inter-symbol interference and reduce the performance of the system.
- **Sensitivity to Doppler spread:** Doppler spread causes interference between the subcarriers.

1.4 QAM and IQ data

QAM (Quadrature Amplitude Modulation) is a type of digital modulation that encodes data onto a carrier signal by modulating the amplitude and phase of the signal. It is commonly used in digital communication systems to transmit digital data over analog channels. IQ data refers to the in-phase and quadrature components of a complex-valued signal. In digital communication systems, the IQ data is typically used to represent the amplitude and phase of the modulated carrier signal. It's ussualy represented with complex numbers in an alphabet $\mathbb{A} \in \mathbb{C}^n$.

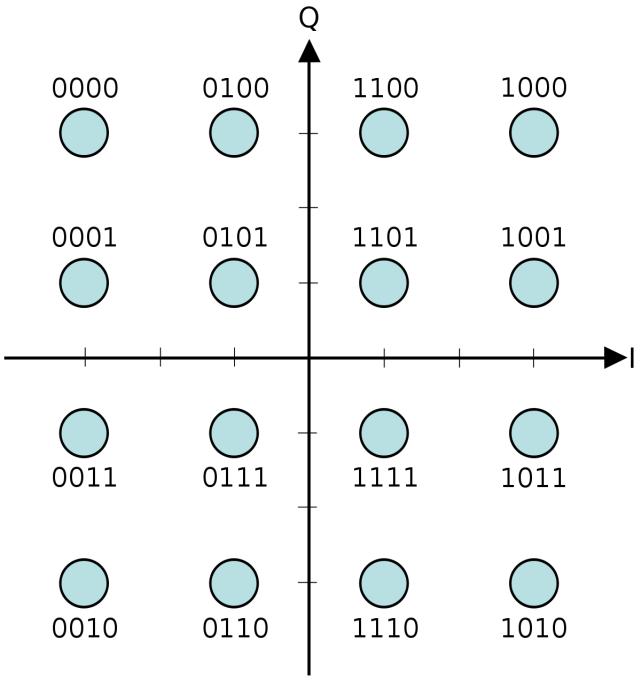

Figure 3: 16-QAM constellation [66]

In the field of communication systems, a QAM constellation refers to a graphical representation of the symbol points in an alphabet on the complex plane. Each point represents a specific combination of the in-phase and quadrature components of the modulated signal. The number of points in the constellation is determined by the number of possible combinations of in-phase and quadrature values, which is in turn determined by the number of bits per symbol used in the QAM modulation scheme. For instance, in a 16-QAM constellation, there are 16 symbol points arranged in a square grid, with each point corresponding to 4 bits of data. The distance between points in the constellation serves as an indicator of the signal-to-noise ratio required for reliable transmission of the data. It is common for QAM constellations to utilize gray code for assigning the symbol points in a manner that minimizes the error rate. However, this is not the only method that can be employed for this purpose

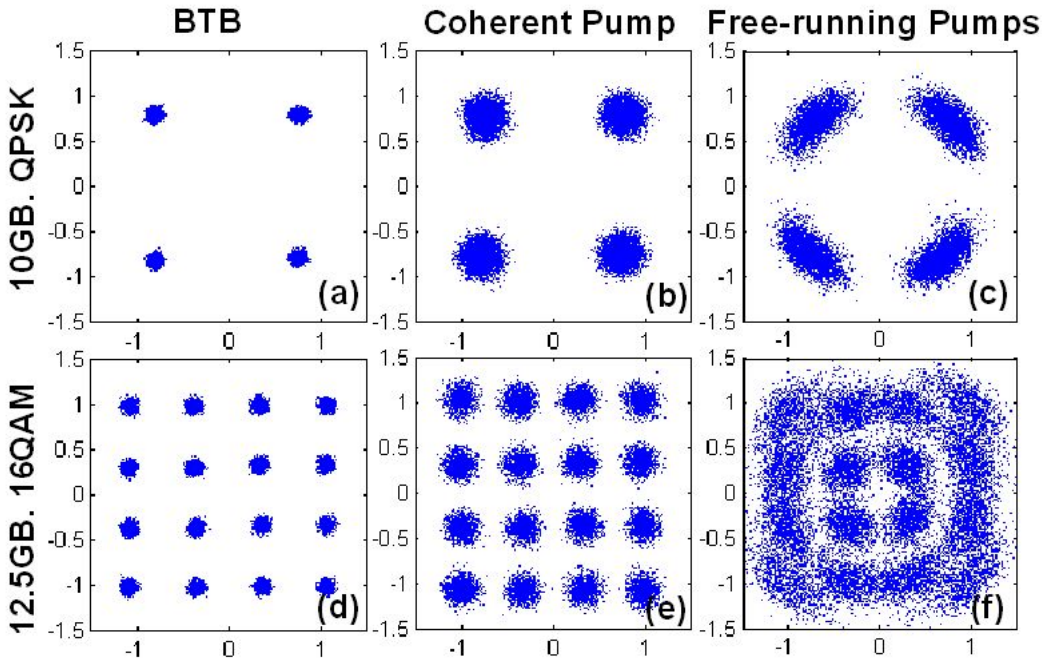

Figure 4: QPSK and 16 QAM with noise and phase displacement [33]

1.5 Channel

1.5.1 Dispersion and doubly dispersion

Time delay dispersion is a measure of the variation in the arrival times of a signal's components caused by transmission through a communication channel. It is a type of distortion that can occur in wireless communication systems and is due to the differences in the propagation paths taken by different parts of the transmitted signal.

Time delay dispersion can cause inter-symbol interference (ISI) in digital communication systems, which can lead to errors in the received signal. To mitigate the effects of time delay dispersion, techniques such as equalization, adaptive modulation, and error correction codes can be used.

A doubly dispersive channel is a type of communication channel that exhibits dispersion in two dimensions, such as time and frequency, or time and spatial dimensions. This means that the signals transmitted through the channel are spread out in both dimensions, which can cause additional distortion and complexity in the communication system. To overcome the challenges of a doubly dispersive channel, advanced techniques such as joint equalization and channel estimation can be employed. [48]

1.5.2 Multipath fading and doppler shift

A multipath fading channel is a type of wireless communication channel that experiences fading of the transmitted signal due to multiple paths of the signal between the transmitter and receiver. This can occur when the signal reflects off of obstacles such as buildings or terrain, or when it is refracted by the atmosphere. Multiple paths of the transmitted signal can cause constructive and destructive interference at the receiver, resulting in rapid fluctuations in the received signal strength. This can cause the signal to fade in and out, which can affect the quality and reliability of the communication. [5]. We begin with the ray-tracing technique and make advantage of the physical geometry of the propagation environment to create a deterministic model of the wireless channel. The delay of a signal refers to the time it takes for the signal to travel from the transmitter to the receiver, and the Doppler shift of a signal refers to the frequency shift of the signal due to the relative motion between the transmitter and receiver.

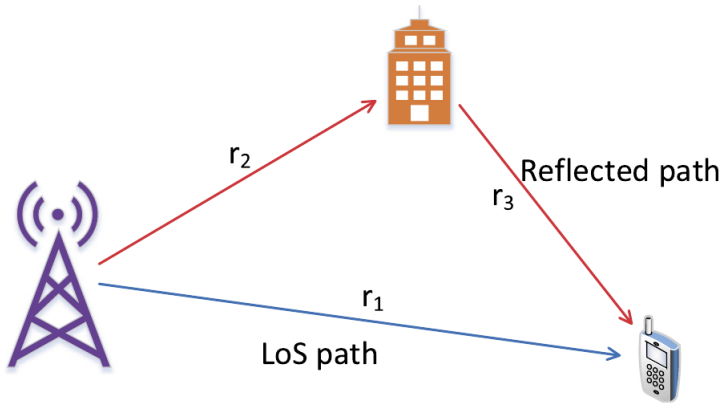

Figure 5: Delay multipath [22]

$$\mathbf{r(t) = g_1 s(t - \tau_1) + g_2 s(t - \tau_2)} \tag{1}$$

- $\mathbf{r(t)}$ recieved signal
- $\mathbf{g_n}$ baseband equivalent complex gain(attenuation)
- $\boldsymbol{\tau_1} = \frac{r_1}{c}$ where c is the speed of light
- $\boldsymbol{\tau_2} = \frac{(r_2 + r_3)}{c}$ delay in reflected path
- $\boldsymbol{\tau_2 - \tau_1}$ delay spread

In wireless communication, the Doppler shift can cause changes in the frequency of a signal as it travels from the transmitter to the receiver. This can happen when the transmitter or receiver (or both) are moving relative to each other, causing the frequency of the signal to shift. The Doppler shift can affect the performance of a wireless communication system

by causing changes in the signal-to-noise ratio and the signal-to-interference ratio, which can degrade the quality of the signal and make it more difficult to detect and decode.[35]

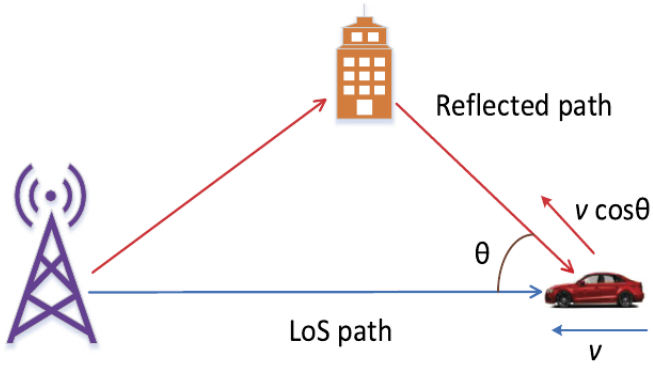

Figure 6: Doppler shifts due to the different angles of arrival [22]

$$\mathbf{r(t) = g_1 e^{j2\pi\nu_1(t-\tau_1)} s(t-\tau_1) + g_2 e^{j2\pi\nu_2(t-\tau_2)} s(t-\tau_2)} \tag{2}$$

- $\boldsymbol{\nu_1 = \frac{v}{c} f_c}$ LOS doppler shift
- $\boldsymbol{\nu_1 = \frac{v*cos(\theta)}{c} f_c}$ doppler shift in reflected path

We can generalize for time dependent function the gain as follows:

$$\mathbf{g(\tau_i, t) = g_i e^{j2\pi\nu_i(t-\tau_i)}} \tag{3}$$

Therefore impulse time-frequency response of channel at fixed time t, can be obtained by taking fourier transform along the delay dimension of $\boldsymbol{g(\tau, t)}$

$$\boldsymbol{H(f,t) = \int g(\tau,t) e^{-j2\pi f\tau} d\tau} \tag{4}$$

A time-frequency channel is a type of communication channel that is characterized by time-varying frequency-selective fading. This means that the channel experiences changes in its frequency response over time, resulting in variations in the amplitude and phase of the signals transmitted through it. Because a channel is assumed to have a slow time-varying function of t, we refer to this phenomenon as having a wide sense of being stationary. [48]

1.6 Equalization

A telecom channel equalizer is a device or algorithm used in telecommunications systems to compensate for distortion or other impairments in a communication channel. The equal-

izer uses signal processing techniques to estimate the characteristics of the channel, such as the impulse response or the frequency response, and then applies a correction to the transmitted signal to counteract the effects of the channel on the received signal. This can improve the performance of the communication system by reducing errors and increasing the data rate or signal-to-noise ratio. Bluetooth, WiFi, IOT, drones, V2V, wireless broadband, and satellite communications are just a few of the everyday applications they can be used for. [16]

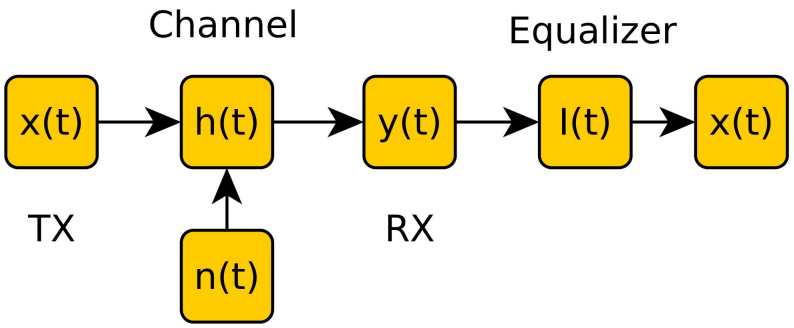

Figure 7: Basic Equalizer

Our goal is to develop an equalizer based on deep learning techniques and research how it performs in terms of timing and memory complexity. We take as a reference the classical methods that manage a good bit error rate, and we will take them as a golden model of accuracy.

1.7 Equalizers

1.7.1 Linear

- Zero Forcing Equalizer (ZFE) : Employed to mitigate intersymbol interference (ISI) that results from multiple signal paths in a communication channel. Despite its efficacy, the ZFE has some limitations, including susceptibility to noise and inability to handle certain types of channel distortion.
- Least Squares(LS): A method used in signal processing to estimate a signal from noisy data. In this method, the estimate is chosen to minimize the sum of the squared differences between the observed data and the predicted data.
- Linear Minimum Mean Square Error (LMMSE): Signal processing technique used to

estimate a signal from noisy data, given knowledge of the noise statistical properties.

- Complexity:

  - Zero Forcing Equalizer (ZFE): This method is the fastest and has a time complexity of $O(N)$, making it an efficient equalization method. It involves dividing the main diagonal of the channel matrix.

  - Least Squares (LS): This method has a higher time complexity of $O(N^3)$ as it includes matrix inversion.
  - Linear Minimum Mean Square Error (LMMSE): This method also has a time complexity of $O(N^3)$ as it includes matrix inversion. However, it is a more advanced signal processing technique that estimates a signal from noisy data, given knowledge of the signal's statistical properties.

1.7.2 Nonlinear

- Approach to symbol detection:

  - OSIC: This algorithm processes the received signal in a sequential manner, starting from the strongest to the weakest received symbol. It detects the strongest symbol first and then removes its interference from the received signal before detecting the next symbol. This process continues until all symbols are detected.
  - NearML: Is a tree search algorithm that explores all possible symbol combinations and computes the distances between the received signal and candidate symbols. It then selects the candidate with the minimum total distance as the detected transmitted symbol. The algorithm employs pruning techniques to reduce the search space while still maintaining good detection performance.

- Complexity:

  - OSIC: The complexity of the algorithm is relatively lower compared to NearML, as it processes the received signal in a sequential manner and does not require an exhaustive search of all possible symbol combinations. This makes OSIC faster and more suitable for real-time applications with lower computational resources.
  - NearML: The complexity of NearML is higher, as it involves an exhaustive search of all possible symbol combinations in the tree. However, the pruning techniques

employed in the algorithm help reduce the search space and improve computational efficiency.

1.8 Problem statement

The OFDM modulation scheme based on 802.11p standard for V2V communication is susceptible to channel impairments during data transmission, particularly multipath fading and Doppler shift. Additionally, the problem of ISI in the received symbols introduces distortion in the IQ data, where data is received and demodulated, leading to incorrect transmitted symbols. As discussed earlier, linear equalization methods rely on finding the pseudo-inverse of certain matrix factors, which can be computationally expensive $O(N^3)$ and numerically unstable. Matrix instability arises when the determinant is very small, which can result in large perturbations in the true solution and a complex circuit architecture for numerical calculations. On the other hand, non-linear methods can reduce BER further, but with higher complexity. Their implementations also require more time and computational resources due to their iterative nature.

1.9 Hypothesis

Deep learning techniques has the ability to learn and model complex non-linear relationships between the received signal and the transmitted signal. Therefore, it can be used as an alternative to traditional methods of channel equalization, including linear techniques such as LS and LMMSE, and non-linear techniques such as OSIC and NearML. This approach can help to overcome the non-linear distortions introduced by the channel, which is particularly useful given the multiple factors affecting the received symbols, including all channel effects such as ISI and noise. Furthermore, most deep learning methods can be evaluated in a single evaluation.

1.10 Objectives

1.10.1 General Objective

- Analyze and test neural network techniques for frequency channel equalization in OFDM systems.
- Breaking the equalization process into simpler parts, delegating complex tasks like matrix inversion to neural networks while handling lighter tasks in a preprocessing stage.
- Combine preprocessing methods (e.g., H^H, zero forcing) with non-linearity of neural networks

- Evaluate performance across various noise levels (5dB to 45dB)

1.10.2 Particular Objective

- Investigate deep learning techniques such as feedforward layers, complex numbers layers, convolutional layers, and transformers.
- Recover QAM and PSK received symbols distorted by LOS and NLOS channel conditions using feedforward neural network layers that correct both phase and magnitude.
- Employ non-linear dimensionality reduction using CNN (Convolutional Neural Networks) to apply zeroforing or some related method.
- Achieve BER and BLER comparable to "golden models," which include some linear techniques such as LS, LMMSE, Zero Forcing, and non-linear techniques like OSIC, NearML.
- Examine transformers and attention mechanisms to reduce ISI.
- Use the FLOPS metric and time complexity for benchmarking and analysis.
- Benchmark and tradeoff between suitable neural network architectures for different situations.

2 State of Art

2.1 Introduction

This section begins with an exploration of basic concepts and literature on estimators, followed by an explanation of how equalizers work within the OFDM scheme modulation, ranging from linear to non-linear equalizers. The study then delves into fundamental concepts of neural networks and convolutional neural networks, which serve as a solid introduction to understanding several key papers relevant to the project and part of the cutting-edge research.

2.2 Estimators

An estimator is a statistical function that maps an observation to an estimate of a parameter of the signal being observed, normally known as $\hat{\theta}$ .It is a mathematical function that takes the data, usually in the form of a sample, and produces an estimate of an unknown parameter of the underlying probability distribution. It's important to note that the quality of an estimator is usually measured by some metric such as Mean Squared Error (MSE), Mean Absolute Error (MAE) or Likelihood that indicates how well the estimator is able to estimate the true parameter [58].

In the context of estimation theory, there are two key concepts to understand: posterior estimators and biased/unbiased estimators.[29]

- **Posterior estimator**: Is a method of estimating a parameter by incorporating prior knowledge or belief about the parameter's distribution. This prior information is combined with the observed data to derive a posterior distribution for the parameter. The posterior distribution represents our updated belief about the parameter after considering the observed data. The estimator is then derived from this posterior distribution. The posterior estimator can be the mean, median, or mode of the posterior distribution, depending on the specific problem and desired properties of the estimator. Let's denote the parameter of interest as $\boldsymbol{\theta}$ and the observed data as X. Bayes' theorem can be written as

$$\boldsymbol{P(\theta|X)} = \frac{\boldsymbol{P(X|\theta)} * \boldsymbol{P(\theta)}}{\boldsymbol{P(x)}} \tag{5}$$

  - $\boldsymbol{P(\theta|X)}$ is the posterior distribution of the parameter $\boldsymbol{\theta}$ given the data $\boldsymbol{X}$.

  - $P(X|\theta)$ is the likelihood of observing the data X given the parameter θ.
  - $P(\theta)$ is the prior distribution of the parameter θ, representing our beliefs about θ before observing the data.
  - $P(X)$ is the marginal likelihood of the data, which can be thought of as a normalization constant.

- **Biased estimator:** Is an estimator whose expected value does not equal the true value of the parameter being estimated. In other words, the estimator consistently overestimates or underestimates the true parameter value. Bias can be due to the estimator's functional form, the presence of outliers in the data, or other factors that systematically affect the estimator.

$$Bias\left(\hat{\theta}\right) = E\left(\hat{\theta}\right) - \theta \tag{6}$$

- **Unbiased estimator:** On average, the estimator provides an accurate estimate of the parameter. However, it's important to note that an unbiased estimator can still have a large variance, which means that individual estimates can be far from the true parameter value.

$$E\left(\hat{\theta}\right) = \theta \tag{7}$$

2.2.1 MSE

Mean squared error (MSE) is a widely used performance metric in estimation theory to evaluate the accuracy of an estimator. It measures the average of the squared differences between the estimated values and the true parameter values. In other words, MSE quantifies the difference between the estimator and the true parameter value, considering both the bias and the variance of the estimator.

$$MSE(\hat{\theta}) = E[(\hat{\theta} - \theta)^2] \tag{8}$$

- $\boldsymbol{\theta}$ is the transmitted signal with the original values
- $\hat{\boldsymbol{\theta}}$ is the predicted values of the received signal.

However, reducing the MSE involves managing the trade-off between bias and variance. This trade-off can be more clearly seen when the MSE is decomposed into the sum of the squared bias and the variance, we can better visualize the idea rewritting the formula [29]:

$$\boldsymbol{MSE}(\hat{\boldsymbol{\theta}}) = \boldsymbol{E}\left\{\left[\left(\hat{\boldsymbol{\theta}} - \boldsymbol{E}\left(\hat{\boldsymbol{\theta}}\right)\right) + \left(\boldsymbol{E}\left(\hat{\boldsymbol{\theta}}\right) - \boldsymbol{\theta}\right)\right]^{\mathbf{2}}\right\} = \boldsymbol{var}(\hat{\boldsymbol{\theta}}) + \left(\boldsymbol{E}\left(\hat{\boldsymbol{\theta}}\right) - \boldsymbol{\theta}\right)^{\mathbf{2}} \quad (9)$$

$$\boldsymbol{MSE}(\hat{\boldsymbol{\theta}}) = \boldsymbol{var}(\hat{\boldsymbol{\theta}}) + \boldsymbol{Bias}^{\mathbf{2}}(\boldsymbol{\theta}) \quad (10)$$

The bias-variance trade-off refers to the challenge of finding an estimator that minimizes both bias and variance. In practice, this trade-off means that an estimator with low bias may have high variance, while an estimator with low variance may have high bias. The goal is to find the optimal balance between bias and variance, resulting in the lowest MSE and the best overall estimator performance.

2.2.2 Bias and variance in Deep Learning

In the context of deep learning models, including neural networks, bias refers to the error introduced by approximating a real-world problem with a simplified model, while variance represents the model's sensitivity to small fluctuations in the training data. A neural network with a high bias tends to produce simpler models that may not capture the true underlying relationship between inputs and outputs, resulting in underfitting. On the other hand, a network with high variance tends to overfit the training data, capturing noise and being sensitive to small changes in the input data.

In supervised learning with neural networks, the goal is to find the right balance between bias and variance to achieve good generalization performance on unseen data. This can be achieved by selecting the appropriate network architecture, using regularization techniques, and fine-tuning hyperparameters.

2.3 Equalization techniques

2.3.1 OFDM Time-Frequency Input-Output Relationship

The goal of this section is to provide a formal description of the key components used in the upcoming estimation sections, and provide and introduction about what estimation is looking for.

The transmitted and received frequency symbol blocks $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{C}^{\boldsymbol{N\times 1}}$ respectively, are obtained by computing the N-point discrete Fourier transform (DFT) of the time-domain blocks s and r $\boldsymbol{s}, \boldsymbol{r} \in \mathbb{R}^{\boldsymbol{N\times 1}}$. We denote the N-point DFT as $\boldsymbol{F_N}$ and is expressed as follows:

$$\boldsymbol{F}_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w_0 & w_0^2 & \dots & w_0^{N-1} \\ 1 & w_0^2 & w_0^4 & \dots & w_0^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_0^{M-1} & w_0^{2(M-1)} & \dots & w_0^{(N-1)^2} \end{bmatrix} \tag{11}$$

where $w_0 = e^{-\frac{2\pi i}{N}}$.

In the time domain, the received signal, $\boldsymbol{r(t)}$, is given by the convolution of the transmitted signal, $\boldsymbol{s(t)}$, and the channel impulse response, $\boldsymbol{h(t)}$, plus the noise, $\boldsymbol{n(t)}$:

$$\boldsymbol{r(t) = s(t) * h(t) + n(t)} \tag{12}$$

To analyze the system in the frequency domain, the DFT is applied to the received signal, transmitted signal, and noise:

$$\boldsymbol{y = F_N \times r}$$
$$\boldsymbol{x = F_N \times s}$$
$$\boldsymbol{n = F_N \times n(t)}$$

When transforming from the time domain to the frequency domain, the linear convolution operation in the time domain is converted into circular convolution. Thus, the relationship between the transmitted signal and the received signal in the frequency domain can be expressed as:

$$\boldsymbol{Y = H \odot X} \tag{13}$$

Each element of this relationship can be developed as follows:

$$\boldsymbol{y_c[n] = (x[n] \circledast h[n])_n = \sum_{m=0}^{N-1} x[m]h[(n-m) mod N]} \tag{14}$$

where H is the channel frequency response, which is the DFT of the channel impulse response, $h(t)$:

$$\boldsymbol{H} = \boldsymbol{F_N h(t)} \tag{15}$$

$$\boldsymbol{H} = \begin{bmatrix} h_{(0)(0)} & 0 & \cdots & h_{(0)(t-1)} \\ h_{(1)(0)} & h_{(1)(1)} & \cdots & h_{(1)(t-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & h_{(d-1)(0)} & \cdots & h_{(d-1)(t-1)} \end{bmatrix} \tag{16}$$

Here, d represents the number of delays (i.e., the length of the channel impulse response), and t represents the number of time samples in the OFDM symbol. Each element of the matrix is represented as $h_{(delay)(time)}$, where delay and time represent the indices of the corresponding delay and time samples, respectively.

To mitigate the effect of Inter-Symbol Interference (ISI) caused by circular convolution, a Cyclic Prefix (CP) is added to the transmitted signal before transmission. This helps maintain orthogonality between subcarriers and minimize ISI. After adding the CP and assuming that the CP length is greater than or equal to the channel impulse response length, the circular convolution becomes equivalent to linear convolution, and the relationship between the frequency-domain components becomes

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n} \tag{17}$$

where $\boldsymbol{y}$, $\boldsymbol{x}$, and $\boldsymbol{n}$ are vectors of length $\boldsymbol{N}$, and $\boldsymbol{H}$ is an $\boldsymbol{N} \times \boldsymbol{N}$ complex-valued matrix that represents the channel effects on the transmitted signal.

The objective of the receiver is to estimate the transmitted signal $\boldsymbol{x}$ from the received signal $\boldsymbol{y}$, which can be done by applying an inverse channel matrix to the received signal. This can be expressed as:

$$\hat{\boldsymbol{x}} = \boldsymbol{H}^{-1}\boldsymbol{y} \tag{18}$$

However, the inverse channel matrix may not always exist or may be difficult to compute, especially in the presence of noise and channel distortions. Therefore, various signal processing techniques such as equalization, filtering, and error correction codes are used to improve the accuracy and reliability of the transmitted signal estimation. These techniques involve manipulating the received signal Y to extract the transmitted signal x and minimize the effects of noise and channel distortions.

2.3.2 Zero forcing

The Zero Forcing Equalizer (ZFE) is a technique used in communication systems to reduce the impact of intersymbol interference (ISI), caused by the presence of multiple signal paths in a communication channel. Although the ZFE is effective, it has some limitations, such as being susceptible to noise and unable to handle certain types of channel distortions. Nevertheless, it can improve the performance of communication systems in specific scenarios. Given the section above in OFDM Time-Frequency Input-Output Relationship as the Input-Output relation. The main diagonal of the channel matrix H is taken and divided by recieved signal. This matrix is already in the frequency domain. To extract the main diagonal of the channel matrix the "diag(H)" operation is used.

$$\boldsymbol{H}_{\text{diag}} = \text{diag}(\boldsymbol{H}) = diag\left(\begin{bmatrix} h_{11} & 0 & \dots & 0 \\ 0 & h_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & h_{NN} \end{bmatrix}\right) = \begin{pmatrix} h_{11} \\ h_{22} \\ \vdots \\ \vdots \\ h_{NN} \end{pmatrix} \tag{19}$$

Ideally, if the off-diagonal elements of the channel matrix H are relatively small, the channel matrix can be approximated by its main diagonal. Then, after diagonal division the estimated transmitted symbols can be expressed as $\hat{\boldsymbol{\theta}} \simeq \boldsymbol{x} + \frac{\boldsymbol{n}}{\boldsymbol{diag(H)}}$, where n is the noise vector and diag(H) is the main diagonal of the channel matrix H. In the equations bellow the estimator is shown.

$$\hat{\boldsymbol{\theta}} = \boldsymbol{y} \circ \div \boldsymbol{diag}(\boldsymbol{H}) = (\boldsymbol{Hx} + \boldsymbol{w}) \circ \div \boldsymbol{diag}(\boldsymbol{H}) \tag{20}$$

Where $\circ\div$ represents the elementwise division.
Based on the paper[26], we will use zero forcing as a preprocessing stage for some of our experiments.

2.3.3 LS

Least squares (LS) equalization is a linear equalization method that aims to minimize the mean squared error (MSE) between the estimated and the transmitted symbols. The goal of the least squares equalization is to find an estimate of the transmitted signal, $\hat{\boldsymbol{\theta}}$, that minimizes the mean squared error (MSE) between the received signal and the predicted received signal. To achieve this, we define the error vector $\boldsymbol{e}$ as the difference between the received signal and the predicted received signal based on the estimated transmitted

signal $e = y - H\theta$,where $\theta = x$. The objective is to minimize the mean squared error (MSE) [29]:

$$\mathrm{J}(\theta) = E\left(\|e\|_2^2\right) = E\left(\|y - H\theta\|_2^2\right) = (y - H\theta)^T (y - H\theta) \tag{21}$$

where $\|\cdot\|_2$ indicates the 2-norm (Euclidean norm).

With the solution in the Moore-Penrose pseudoinverse H^+.The Moore-Penrose pseudoinverse is often used to solve linear least squares problems, which involve finding the values of variables that minimize the sum of the squares of the residuals (the differences between the observed values and the values predicted by the model). It can also be used to compute a "best fit" solution for systems of linear equations that do not have a unique solution. [59]

$$H^+ = (H^H H)^{-1} H^H \tag{22}$$

- H Channel matrix
- H^+Moore-Penrose pseudoinverse
- H^H Hermiatian transpose matrix. Complex square matrix that is equal to its own conjugate transpose

Finally the estimator is:

$$\hat{\theta} = (H^H H)^{-1} H^H \theta \tag{23}$$

Its resistance to noise makes it appealing in a variety of communication systems, however because the noise component is ignored, it cannot operate satisfactorily with low SNR (signal to noise ratio). The unbiasedness of the equalizer depends on the specific equalization problem and the assumptions made about the channel and noise characteristics, and the variance of the least squares equalization method quantifies the uncertainty in the estimated transmitted signal due to the presence of noise in the received signal. A higher variance implies a greater degree of uncertainty in the equalized signal.

2.3.4 MMSE

MMSE equalization is a linear equalization technique that aims to minimize the mean squared error between the transmitted signal and its estimate. The MMSE equalizer balances the trade-off between bias and variance, unlike the zero-forcing equalizer, which

focuses solely on eliminating intersymbol interference (ISI) without considering the noise amplification. MMSE equalization can be thought of as a compromise between the desire to eliminate ISI (which could introduce bias) and the need to limit noise amplification (which affects variance). The objective of MMSE equalization can be expressed as:

$$\boldsymbol{MMSE = min(E[(\hat{\theta} - \theta)^2])} \tag{24}$$

- $\boldsymbol{\theta}$ is the transmitted signal.
- $\hat{\boldsymbol{\theta}}$ is the estimated signal.

Mathematically, the MMSE estimator is defined as $\hat{\boldsymbol{\theta}}_{\boldsymbol{MMSE}} = \boldsymbol{E}[\boldsymbol{\theta}|\boldsymbol{y}]$

Here, $\hat{\boldsymbol{\theta}}_{\boldsymbol{MMSE}}$ is the estimate of the transmitted signal $\boldsymbol{\theta}$ given the received signal$\boldsymbol{y}$. The MMSE estimator computes the conditional expectation of $\boldsymbol{\theta}$ given the observed signal $\boldsymbol{y}$. This means that the MMSE estimator takes into account both the prior information about $\boldsymbol{\theta}$ and the likelihood of observing $\boldsymbol{y}$ given $\boldsymbol{\theta}$.

2.3.5 LMMSE

The Linear Minimum Mean Squared Error (LMMSE) estimator is a linear version of the MMSE estimator. While the MMSE estimator can be nonlinear, the LMMSE estimator restricts itself to linear functions of the observed data. The goal of the LMMSE estimator is to find a linear estimate of the transmitted signal that minimizes the mean squared error between the transmitted signal and its linear estimate, given the received signal. Compared to the Least Squares (LS) algorithm, this can perform better when there is low signal-to-noise ratio (SNR). To put it another way, the LMMSE equalization is a method that can be applied to restore precision to a signal that has been distorted by noise, especially when the noise level is high. When there is a low signal-to-noise ratio (SNR) and a significant quantity of noise in the signal, it performs exceptionally well. Mathematically, the LMMSE estimator is defined as:

$$\hat{\boldsymbol{\theta}}_{\boldsymbol{LMMSE}} = \boldsymbol{E}[\boldsymbol{\theta}] + \boldsymbol{C}_{\boldsymbol{\theta y}}\boldsymbol{C}_{\boldsymbol{yy}}^{\boldsymbol{-1}}(\boldsymbol{y} - \boldsymbol{E}[\boldsymbol{y}]) \tag{25}$$

- $\boldsymbol{E}[\boldsymbol{\theta}]$ is the prior expectation of the transmitted signal.
- $\boldsymbol{C}_{\boldsymbol{\theta y}}$ is the cross-covariance matrix between the transmitted signal and the received signal.

- C_{yy} is the covariance matrix of the received signal.
- $E[y]$ is the prior expectation of the received signal.

The LMMSE estimator seeks to find a linear estimate of θ given the received signal y:

$$\hat{\theta}_{LMMSE} = Wy \tag{26}$$

where W is the LMMSE equalizer matrix.

Using the LMMSE estimator definition $\hat{\theta}_{LMMSE}$, we can derive the LMMSE equalizer matrix W:

$$W = C_{\theta y} C_{yy}^{-1} \tag{27}$$

We know that the covariance matrix of y, C_{yy}, is given by:

$$C_{yy} = HC_{\theta\theta}H^H + \sigma^2 I \tag{28}$$

where$C_{\theta\theta}$ is the covariance matrix of θ, and H^H denotes the conjugate transpose (Hermitian) of the matrix H.

Now, we need to find the cross-covariance matrix between θ and y

$$C_{\theta y} = E[(\theta - E[\theta])(y - E[\theta])^H] = E[\theta y^H] = C_{\theta\theta}H^H \tag{29}$$

Assuming that the transmitted symbols are uncorrelated with equal power, we have

$$C_{\theta\theta} = E[\theta\theta^H] = \rho I \tag{30}$$

whereρ is the average symbol power, and for simplicity we keep at unitary power. Substituting back the elements from Eq. 37, we obtain the final estimation equation.

$$W = H(HH^H + \sigma^2 I)^{-1} \tag{31}$$

$$\hat{\theta}_{LMMSE} = (H^H H + \sigma^2 I)^{-1} H^H \theta \tag{32}$$

This equation consider AWGN (Additive White Gaussian Noise) with variance σ^2. It is called "white" because it has a flat power spectral density, meaning that it has equal power at all frequencies. It is called "additive" because it can be added to a signal without changing its distribution.

2.4 Non-linear equalizers

NearML (Near Maximum Likelihood) and OSIC (Ordered Successive Interference Cancellation) are two non-linear signal equalization techniques used in digital communication systems to mitigate the challenges posed by signal distortions. These methods provide better symbol detection accuracy and overall system performance, making them invaluable tools in modern communication systems design. However, they compromise time complexity due to the iterative nature of their applications. In our research team, these methods have been previously applied in MATLAB for benchmarking purposes in the work titled “Evaluation of OFDM Systems With Virtual Carriers Over V2V Channels” [12]. We needed to perform a coding language translation to Python in order to make a fairer metric comparison under the same conditions as all the other Equalization methods and Nueronal Networks. Additionally, there was a well-planned architecture to integrate these methods in an almost transparent manner. Additional information about the software architecture can be found in the methodology section.

2.4.1 OSIC

Ordered Successive Interference Cancellation (OSIC) detection method. This method is used for detecting transmitted symbols from a received signal in digital communication systems. The function accepts four inputs: the received signal vector, the channel matrix, the constellation points, and the indices of the detected symbols. The function starts by initializing several variables needed for the algorithm. It then goes through each column of the channel matrix in reverse order. For each column, it estimates the corresponding transmitted symbol by dividing the received signal by the channel matrix value. Then, it calculates the squared distance between the estimated symbol and each constellation point.

The algorithm selects the constellation point with the smallest squared distance as the detected symbol and updates the received signal vector accordingly. This process helps to cancel the interference caused by previously detected symbols and improves the overall detection performance. Finally, the detected symbols are rearranged according to the given indices and stored in an output tensor. The function returns this tensor containing the estimated transmitted symbols.

- $\mathbf{y}$: The received signal, which is a complex-valued column vector of length $\boldsymbol{N}$.
- $\mathbf{H}$: The channel matrix of size $\boldsymbol{N} \times \boldsymbol{N}$, which represents the channel coefficients
- $\boldsymbol{A}$: The set of constellation points, which represents the possible symbol values in the modulation scheme used for the transmitted signal.

- $\mathbf{r}$: A temporary variable used for storing the residual signal during the algorithm. It is initialized as the received signal $\mathbf{y}$ and updated in each iteration of the loop.
- $\boldsymbol{k}$: The loop index representing the current column of the channel matrix being considered.
- $\boldsymbol{a_{est}}$: The estimated transmitted symbol value for the k-th column of the channel matrix, computed as the ratio between the k-th element of the residual signal $\mathbf{r}$ and the k-th diagonal element of the channel matrix $\mathbf{H}$.
- dist: A vector of squared distances between the estimated transmitted symbol value a_{est} and the constellation points in $\mathbb{A}$.
- $\boldsymbol{\hat{\theta}_{osic}}$: The estimated transmitted symbol vector of length N. It is initialized as an empty vector and updated with the minimum-distance constellation point in each iteration of the loop.

Algorithm 1 OSIC Detection Algorithm

procedure OSIC_DET($\mathbf{y}$, $\mathbf{H}$, $\mathbb{A}$)
  $\mathbf{r} \leftarrow \mathbf{y}$
  for $k \leftarrow N-1, N-2, \ldots, 0$ **do**
    $a_{est} \leftarrow \frac{\mathbf{r}[k]}{\mathbf{H}[k,k]}$
    dist $\leftarrow |a_{est} - \mathbb{A}|^2$
    $\hat{\theta}_{osic}[k] \leftarrow \arg\min_{a \in \mathbb{A}}$ dist
    $\mathbf{r} \leftarrow \mathbf{r} - \hat{\theta}_{osic}[k] \cdot \mathbf{H}[:, k]$
  end for
  return $\hat{\theta}_{osic}$
end procedure

The outer loop iterates through the columns of the channel matrix, which has a size of $\boldsymbol{N}$. Therefore, the outer loop has a complexity of $\boldsymbol{O(N)}$. Inside the outer loop, there's a calculation of squared distances between the estimated symbol and each constellation point. The constellation points have a size of $|\mathbb{A}|$. This operation has a complexity of $\boldsymbol{O}\left(|\mathbb{A}|\right)$.

Since the squared distance calculation is inside the outer loop, the overall complexity of the OSIC_Det algorithm :

$$\boldsymbol{O(N * |\mathbb{A}|)} \tag{33}$$

2.4.2 NearML

The Near-ML detection algorithm functions as a tree search with nodes representing constellation points and depth equal to the number of transmitters. It computes distances between received signals and candidate symbols, selecting M best candidates at each level for pruning. By backtracking and updating accumulated distances, it maintains the

M best candidates and performs additional pruning when necessary. The algorithm ultimately selects the candidate with the minimum total distance, resulting in a near-optimal solution with reduced search space and good detection performance.

For the Near Maximum Likelihood (Near-ML) detection algorithm, the goal is to estimate the transmitted symbols, denoted by $\hat{\boldsymbol{\theta}}_{\boldsymbol{NML}}$, given the received signal $\boldsymbol{Y}$, channel matrix $\boldsymbol{H}$, and the constellation points $\mathbb{A}$.

- $\boldsymbol{M}$: The number of best candidates to be stored at each level of the tree search.
- $\boldsymbol{QRM}$: $|\mathbb{A}|$.
- $\boldsymbol{a_{est}}$: The temporary estimation of the transmitted symbol at a specific level, calculated as the received signal divided by the corresponding channel coefficient.
- $\boldsymbol{d}$: The distance between the temporary estimation $\boldsymbol{a_{est}}$ and the constellation points.
- $\boldsymbol{d_{min}}$: The minimum total distance found so far, used for pruning.
- $\boldsymbol{d_{total}}$: The total distance of each candidate, used to find the best candidates.
- $\boldsymbol{d_{minf}}$: The minimum total distance found at the end of the tree search, used to determine the best candidate.
- $\boldsymbol{s_{est3}}$: The reordered detected symbols according to the original index.
- $\boldsymbol{index[k]}$: The original index of the detected symbols.

Algorithm 2 Near Maximum Likelihood Detection

procedure NEARML($\boldsymbol{y_p}$, $\boldsymbol{H}$, $\mathbb{A}$, M)
  Initialize tree search, set nt as the number of columns of $\boldsymbol{H}$, and set $QRM = |\mathbb{A}|$
  Initialize variables, tensors, and arrays
  Compute the distances at level nt: $a_{est} = \frac{y_p[nt-1]}{H[nt-1,nt-1]}$, $d = |a_{est} - \mathbb{A}|^2$
  Sort the distances and initialize the parent nodes and parent received signals
  for n in range(QRM) **do**
    for k in range($nt - 1$, 0, -1) **do**
      Compute the distances at level k: $a_{est} = \frac{y_p[k-1]}{H[k-1,k-1]}$, $d = |a_{est} - \mathbb{A}|^2$
      Sort the distances and update the best candidates
      Perform pruning: check if $d_{min} > d_{total}$
      if pruning condition met **then**
        Skip the remaining branches and move to the next subtree
      end if
    end for
    Store the M best candidates
    Update the minimum total distance: $d_{min} = \min(d_{total})$
    if a new tree is opened **then**
      Check the pruning condition: $d_{min} > d_p$
      Reset the skip flag if needed
    end if
  end for
  Determine the vector with the minimum distance: $d_{minf} = \min(d_{total})$
  Reorder the detected symbols according to the original index: $\hat{\theta}_{\text{NearML}}[index[k]] = s_{est3}[k]$
end procedure

Based on the provided NearML code, the time complexity can be analyzed by considering the nested loops and the operations inside them. The outer loop iterates $\boldsymbol{QRM}$ times, where $\boldsymbol{QRM}$ is the number of constellation symbols (denoted as $|\mathbb{A}|$). Inside this loop, there is another loop that iterates nt times, where $\boldsymbol{nt}$ represents the number of columns in the channel matrix $\boldsymbol{H}$ which is $\boldsymbol{N}$. Within the inner loop, there are operations that take $\boldsymbol{M}$ and $\boldsymbol{QRM}$ time.

Therefore, the overall time complexity of the NearML algorithm can be approximated as :

$$\boldsymbol{O}\left(|\mathbb{A}| \times \boldsymbol{N} \times (\boldsymbol{M} + |\mathbb{A}|)\right) \simeq \boldsymbol{O}\left(|\mathbb{A}|^{\boldsymbol{2}} \times \boldsymbol{N} \times \boldsymbol{M}\right) \tag{34}$$

2.5 Signal quality metrics

2.5.1 SNR

SNR stands for Signal-to-Noise Ratio which contrasts the strength of the signal with the strength of the noise. The most common way to measure it is in decibels (dB). In general, higher numbers indicate a better specification because there is a greater ratio of useful information (the signal) to unwanted data (the noise). Given this source [65], it can be explained how the noise is calculated for noise simulation with the requirement of a certain

SNR in dB.

1. Firstly, a vector "s" that represents the signal transmitted over a communication channel is considered. The objective is to derive a vector "r" that represents the signal received at the receiver after passing through the Additive White Gaussian Noise (AWGN), this noise type is described in [42]. The level of noise introduced by the AWGN channel is determined by the provided Signal-to-Noise Ratio (SNR), and it is denoted as "γ".
2. The length of the vector "s" is denoted by N. The signal power of the vector "s" can be expressed as follows:

$$P_{signal} = \frac{1}{N} \sum_{i=0}^{N-1} |s_i|^2 \tag{35}$$

   (a) If we have a complex IQ value $s_i = a_i + jb_i$, the modulus or magnitude of s_i can be represented by $|s_i| = \sqrt{a_i^2 + b_i^2}$, where a and b are real numbers.

3. To compute the power spectral density of the noise vector "n", we use the following equation:

$$N_o = \frac{P_{signal}}{\gamma} = P_{signal} \times 10^{-\frac{SNR_{dB}}{10}} \tag{36}$$

   (a) Where $\gamma = 10^{\frac{SNR_{dB}}{10}}$ is the SNR in dB to linear scale

4. Assuming a complex IQ plane for all digital modulations, the noise variance (noise power) required for generating Gaussian random noise can be expressed as follows:

$$\sigma^2 = \frac{N_0}{2} \tag{37}$$

5. Then, the Gaussian random noise vector "n" of length N is generated, with its samples drawn from a Gaussian distribution having a mean of zero and a standard deviation as calculated using the last equation.

$$\boldsymbol{f(x)} = \begin{cases} \boldsymbol{\sigma} \times \boldsymbol{\mathcal{N}_N(0,1)}, & \text{if } \boldsymbol{s} \text{ is real} \\ \boldsymbol{\sigma} \times \left[\boldsymbol{\mathcal{N}_N(0,1)} + \boldsymbol{j} * \boldsymbol{\mathcal{N}_N(0,1)}\right], & \text{if } \boldsymbol{s} \text{ is complex} \end{cases} \tag{38}$$

(a) Here, $\boldsymbol{\mathcal{N}_N(0,1)}$ represents the white Gaussian noise vector with a mean of 0 and a standard deviation of 1

6. Lastly, the generated noise vector is added to the original signal $\boldsymbol{s}$.

$$\boldsymbol{r} = \boldsymbol{s} + \boldsymbol{n} \tag{39}$$

2.5.2 BER

In digital communication systems, the transmission of data is not always error-free. Bit errors may occur due to various factors such as noise, interference, distortion, and fading. Bit Error Rate (BER) is a common metric used to quantify the quality of a digital communication system by measuring the probability of bit errors. BER is defined as the ratio of the number of erroneous bits to the total number of transmitted bits. It is usually expressed in terms of power, as the probability of error is dependent on the signal-to-noise ratio (SNR) and the channel conditions. The BER can be mathematically represented as follows:

$$\boldsymbol{BER} = \frac{\boldsymbol{N_{err}}}{\boldsymbol{N_{tot}}} \tag{40}$$

where N_{err} is the number of erroneous bits and N_{tot} is the total number of transmitted bits.

The bit error rate (BER) can be defined in terms of the probability of error (POE). The POE can be calculated using three other variables: the error function (erf), the energy per bit (Eb), and the noise power spectral density (No).

- The error function,$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$, is used to calculate the POE, and its value depends on the modulation method used in the communication system.
- The energy per bit, Eb, is a measure of the signal strength and can be calculated by dividing the carrier power by the bit rate.
- The noise power spectral density, No, is a measure of the noise present in the communication channel and is typically expressed in units of power per Hz.

By using these three variables, the POE can be calculated, and thus the BER can be expressed in terms of POE. This provides a measure of the communication system's performance that takes into account both the strength of the signal and the noise present in the channel.

$$\mathbf{POE} = \frac{\mathbf{1}}{\mathbf{2}}(\mathbf{1} - \mathbf{erf})\sqrt{\frac{\mathbf{E_b}}{\mathbf{N_o}}} \tag{41}$$

The BER is a critical parameter for evaluating the performance of a digital communication system. A high BER indicates poor system performance, which can lead to loss of data, degraded audio or video quality, or even complete system failure. Therefore, the BER is used as a benchmark to ensure that the digital communication system meets the required performance specifications.

2.5.3 BLER

Block Error Rate (BLER) is a measure of the reliability of a communication system or a data transmission system by blocks. BLER is defined as the ratio of the number of blocks of data received with errors to the total number of blocks of data transmitted. It is commonly used in wireless communication systems, where the transmission of data can be affected by various factors such as fading, interference, and noise. In such systems, BLER is often used to evaluate the performance of the system and to determine the optimal operating parameters such as power, coding, and modulation schemes.

$$\boldsymbol{BLER} = \frac{\boldsymbol{BlockErrCount}}{\boldsymbol{TotalBlocks}} \tag{42}$$

To check for errors in a block of data, a cyclic redundancy check (CRC) is often used [43]. Given the estimated values and the ground truth, instead of comparing bit by bit, a set of symbols is taken, and a CRC is calculated for a given chunk of the data. If the CRC of the equalized chunk is equal to the real value, there are no errors. However, if the CRC differs, the chunk is marked as incorrect, and it is added to the BLER count.

2.6 Neuronal networks

Neural networks are a type of artificial intelligence system designed to mimic the functioning of the human brain. They consist of interconnected nodes, or "neurons," which are capable of processing information and making decisions based on that information. These networks are usually organized into layers, with each layer containing a different number of neurons. The input layer receives input from the external environment, while the output layer produces the final result or decision based on that input. The layers in between the input and output layers are called hidden layers, and they perform various intermediate

Algorithm 3 Block Error Rate Calculation with CRC

Require:
Input data is 48 blocks of 48 symbols
$txbits$: array of transmitted blocks
$rxbits$: array of received blocks
crc_func: CRC function for generating CRC codes
Ensure:
$BLER$: Block Error Rate of the communication system

$total\_blocks \leftarrow 48$
$bad\_blocks \leftarrow 0$
for $n \leftarrow 0$ to 47 **do**
    $tx\_crc \leftarrow crc\_func(txbits[n].tobytes())$
    $rx\_crc \leftarrow crc\_func(rxbits[n].tobytes())$
    if $tx\_crc \neq rx\_crc$ **then**
        $bad\_blocks \leftarrow bad\_blocks + 1$
    end if
end for
$BLER \leftarrow bad\_blocks/total\_blocks$
return $BLER$

calculations and processing tasks. Neural networks are trained using large amounts of data, allowing them to learn and make predictions or decisions based on that data.

2.6.1 Linear Layer

A linear layer in a neural network is a type of layer that applies a linear transformation to the input data. This transformation can be represented by a **matrix** of weights, denoted as $\boldsymbol{W_n}$,and a biases **vector**, denoted as $\boldsymbol{b_n}$, which are learned during the training process. The subscript $\boldsymbol{n}$ refers to the nth layer. The output of a linear layer is calculated by performing a matrix and vector product between the input data $\boldsymbol{X_{n-1}}$ and the weights $\boldsymbol{W_n}$ as well as adding the biases.

$$\boldsymbol{X_n} = \boldsymbol{W_n X_{n-1}} + \boldsymbol{b_n} \tag{43}$$

- $\boldsymbol{W_n}$weight matrix at layer n.
- $\boldsymbol{b_n}$ bias at layer n
- $\boldsymbol{X_{n-1}}$Input or last layer data
- $\boldsymbol{X_n}$ Output data

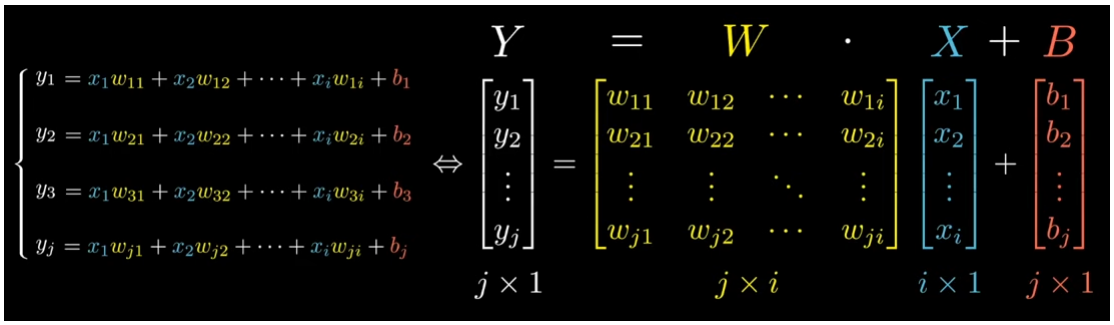

Figure 8: Output as Y and input as X. Vector matrix representation of system. Desing done with manim

2.6.2 Backpropagation

The goal of the layer is to optimize the weights parameters so that they fit a target referred to as the ground truth. The error, denoted by E, is calculated as the difference between the predicted output ($\hat{\boldsymbol{X}}$) and the actual target output ($\boldsymbol{X_n}$), where n is the index of the sample in the dataset. To achieve this, we will utilize the backpropagation algorithm, which is a common method in the field of artificial neural networks for training the network by adjusting the weights between neurons in the network.

$$\boldsymbol{Err} = \hat{\boldsymbol{X}} - \boldsymbol{X_n} \tag{44}$$

This method analyzes the error rate in relation to the weights and inputs. As we adjust our trainable parameters, {$\boldsymbol{W_n}$, $\boldsymbol{b_n}$}, the error will change accordingly. The goal is to minimize the error, or to find a point where the error gradient is zero.[17]

$$\boldsymbol{\nabla W} = \frac{\boldsymbol{\partial E}}{\boldsymbol{X}_{n+1}} \times \boldsymbol{X}_{n-1}^{\boldsymbol{T}} \tag{45}$$

$$\boldsymbol{\nabla X}_{n-1} = \boldsymbol{W}_{\boldsymbol{n}}^{\boldsymbol{T}} \times \frac{\boldsymbol{\partial E}}{\boldsymbol{X}_{n+1}} \tag{46}$$

- $\times$Matrix multiplication
- $\boldsymbol{\nabla W}$Gradient of weights.The gradient is a multi-variable generalization of the derivative.

2.6.3 Learning rate

When training a neural network, the weights $\boldsymbol{W_n}$ and bias terms $\boldsymbol{b_n}$ are updated iteratively using an optimization algorithm such as gradient descent. The learning rate γ is a hyper-

parameter that plays a crucial role in this process by determining the size of the update step applied to the weights and biases at each iteration.

The learning rate γ is a scalar value that is multiplied by the gradient of the loss function with respect to the weights of the network. This gradient provides information about the direction and magnitude of the weight update required to minimize the loss function. A smaller learning rate leads to smaller updates and slower convergence, while a larger learning rate results in larger updates and faster convergence. [51]

$$\boldsymbol{W_n} = \boldsymbol{W_n} - \gamma \boldsymbol{\nabla W} \tag{47}$$

$$\boldsymbol{b_n} = \boldsymbol{b_n} - \gamma \frac{\boldsymbol{\partial E}}{\boldsymbol{X_n}} \tag{48}$$

If the learning rate is set too low, the optimization process may become stuck in a local minimum or local maximum. A local minimum is a point in the optimization landscape where the cost function has a lower value than the surrounding points, but is not the global minimum. A local maximum is a point where the cost function has a higher value than the surrounding points, but is not the global maximum. This can lead to suboptimal performance or even failure of the optimization process. On the other hand, if the learning rate is set too high, the optimization process may oscillate or diverge, also leading to suboptimal performance. It is important to choose an appropriate learning rate for the optimization process in order to avoid these problems.

2.6.4 Activation functions

Activation functions are used in neural networks to introduce non-linearity into the network. This is important because many real-world problems are non-linear in nature and a neural network with only linear functions would not be able to model such problems accurately. Activation functions allow the network to learn more complex patterns in the data and improve the accuracy of the network. They also help to prevent the network from becoming stuck in a local minimum or plateau during training.

However, the traditional activation functions pose some challenges when it comes to embedded devices. They often involve computationally expensive operations such as exponentiation, division, and floating-point arithmetic. These operations can significantly increase the processing time and power consumption, making them less attractive for resource-constrained embedded systems. To address these concerns, researchers have proposed hardened versions of activation functions. Hardened activation functions are designed to be computationally less demanding while maintaining similar performance

to their traditional counterparts. Examples of hardened activation functions include the binary step function, RELU, Hardtahn, Hardsigmoid.

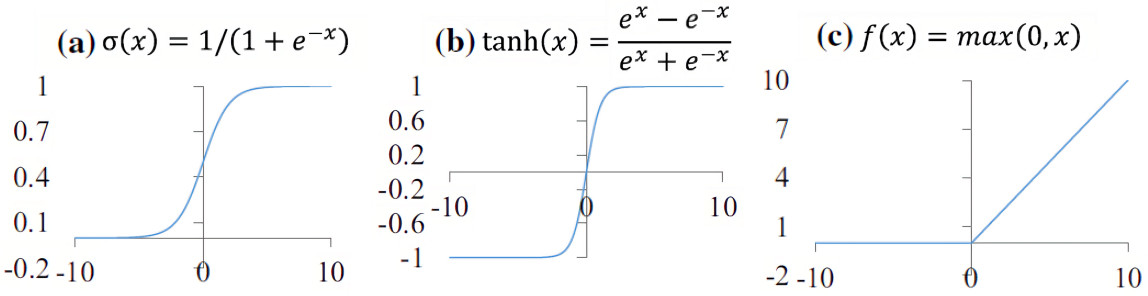

Figure 9: Activation function. (a) Sigmoid, (b) tanh, (c) ReLU. [63]

The most widely used activation functions in neural networks include the sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) functions. The sigmoid function transforms any input value into a range from 0 to 1, while the tanh function adjusts input values to fall within the -1 to 1 range. The ReLU function operates linearly, setting all negative input values to 0 and retaining all positive input values in their initial form. [54]

As we have seen before, there exist "hardened" variants of activation functions, such as hardtanh and hardsigmoid, that offer more efficient computational evaluations while maintaining comparable outcomes to their traditional counterparts. These hardened activation functions enable faster neural network computations while still delivering similar results, with the Hardtanh function being the most used for this research that can be described as follows:

$$\boldsymbol{hardtanh}(\boldsymbol{x}) = \begin{cases} -\mathbf{1}, & \text{if } \boldsymbol{x} < -\mathbf{1} \\ \boldsymbol{x}, & \text{if } -\mathbf{1} \leq \boldsymbol{x} \leq \mathbf{1} \\ \mathbf{1}, & \text{if } \boldsymbol{x} > \mathbf{1} \end{cases} \tag{49}$$

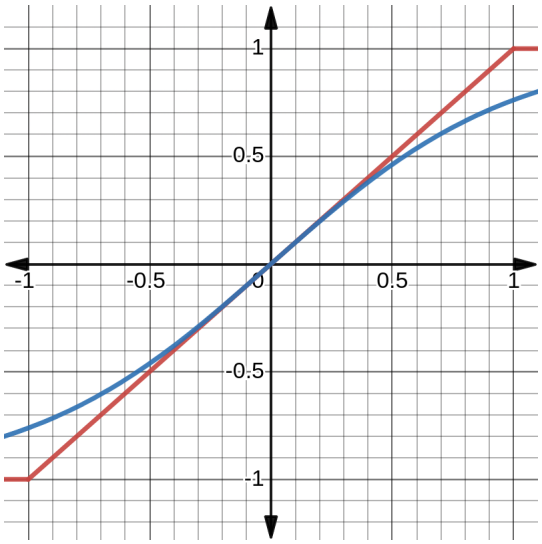

Figure 10: Hardtanh(red) and Tanh(blue)

2.7 Loss functions

A loss function is often defined as a scalar function of the model's parameters, the input data, and the true output. It quantifies how well the model is able to fit the data, and it's commonly used to evaluate the performance of different models and to select the best one. In terms of estimators, a loss function can be seen as a measure of how well the estimator is able to estimate the true parameter. The goal of training a machine learning model is to find the sub-optimal set of parameters that minimize the loss function. There are different types of loss functions, each one is suitable for different types of problems. There are various types of loss functions, each tailored to specific types of problems. For example, in a regression problem, the mean squared error (MSE), as mentioned in section (2.2.1) , is a commonly used loss function. Conversely, in classification problems, the cross-entropy loss is often employed. Due to its significance, we will delve deeper into the cross-entropy loss in the following section, discussing how it operates within the context of deep learning.

2.7.1 Cross Entropy Loss

Previously, we have discussed the Mean Squared Error (MSE) in the context of estimators. However, there is another cost function, known as the cross-entropy loss, which can be utilized for evaluating models in the context of classification problems. Prior to elaborating on the cross-entropy loss, it is imperative to also discuss the softmax function.

The softmax function is a mathematical technique which transforms a vector of real numbers into a probability distribution over the classes. The output of the softmax function

is a vector of values between 0 and 1 that sum up to 1, which can be interpreted as probabilities. The softmax function is employed in this context as it provides a means to convert the output, or scores, of the model into a probability distribution that represents the uncertainty of the model's predictions. Additionally, the softmax function ensures that the probability of each class falls within the range of 0 and 1 and that the sum of all class probabilities is 1, which is a necessary requirement for a probability distribution.

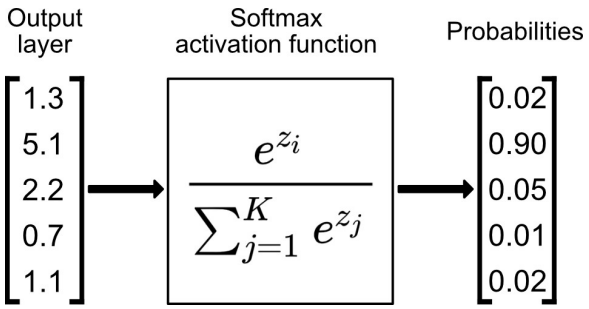

Figure 11: Softmax used to map real numbers to probabilty distribution. [47]

After the softmax output, it is measured by the cross-entropy loss function, which checks the dissimilarity between the predicted and true probability distributions. One of the many advantages of this function is that it is also easy to compute and differentiate, making it a suitable loss function for gradient-based optimization algorithms. The goal of the training process is to minimize the Kullback-Leibler divergence.

$$\boldsymbol{D_{KL}(P||Q) = \sum P(i) log \frac{P(i)}{Q(i)}} \tag{50}$$

In this equation, $\boldsymbol{D_{KL}(P \mid\mid Q)}$ represents the KL divergence between two probability distributions $\boldsymbol{P}$ and $\boldsymbol{Q}$, where $\boldsymbol{P}$ is the true distribution and $\boldsymbol{Q}$ is the approximating distribution. The double vertical bars (||) in the notation denote "divergence between" the two distributions. The KL divergence is widely used in information theory and machine learning to evaluate the dissimilarity between two distributions. It is important to note that the Kullback-Leibler (KL) divergence is not symmetric. In other words, the KL divergence between distributions P and Q, denoted as $\boldsymbol{D_{KL}(P \mid\mid Q)}$, is not equal to the KL divergence between $\boldsymbol{Q}$ and $\boldsymbol{P}$, denoted as $\boldsymbol{D_{KL}(Q \mid\mid P)}$. The asymmetry of the KL divergence implies that it should not be regarded as a true distance metric in the strict mathematical sense. Instead, it serves as a measure of dissimilarity between two probability distributions. This characteristic is also reflected in its alternative name, relative entropy.

The KL divergence plays a crucial role in determining which class is more probable. The predicted class distribution, denoted by $P(y \mid x_i; \theta)$, is influenced by the parameters θ, while the true class distribution is represented by $P*(y \mid x_i)$. Both of these distributions

are taken into account when assessing the dissimilarity between them.

$$\boldsymbol{D_{KL}(P^*(y|x_i)||P(y|x_i;\theta)} = \sum \boldsymbol{P^*(y|x_i)log\frac{P^*(y|x_i)}{P(y|x_i;\theta)}} \tag{51}$$

Rewriting the logarithm as two individual sections, we can observe that the left part of the equation below does not depend on the θ parameter.

$$\sum \underbrace{\boldsymbol{P^*(y|x_i)log(P^*(y|x_i))}}_{\text{Independent of } \boldsymbol{\theta}} \boldsymbol{-P^*(y|x_i)P(y|x_i;\theta)} \tag{52}$$

And then we aim to make both distributions as similar as possible using the best estimator.

$$\underset{\boldsymbol{\theta}}{\text{argmin}}\boldsymbol{D_{KL}(P^*||P)} \equiv \underset{\boldsymbol{\theta}}{\text{argmin}} - \sum_{\boldsymbol{y}} \boldsymbol{P^*(y|x_i)P(y|x_i;\theta)} \tag{53}$$

2.8 Convolutional Neuronal Networks

During the research, we will need to work with channel matrices as inputs for neural networks. As these are matrices rather than vectors, we will discuss convolutional neural networks (CNNs), which are a type of deep learning neural network primarily used for image recognition and dimensional reduction. This type of network excel at feature extraction, which is a critical aspect of their success in various applications, especially in image recognition and computer vision tasks. Feature extraction is the process of identifying and extracting relevant patterns or features from raw data, helping the model to discern and recognize important characteristics within the input. The network is composed of multiple layers, including **convolutional layers**, **activation layers**, **pooling layers** and **linear layers**.

The convolutional layers apply a set of filters to the input data, where each filter is a small matrix of weights. This ones are used to identify features in the data such as **edges**, **textures**, and **shapes**, specifically, we will be utilizing these layers to extract the relationship of intercarrier symbol interference (ISI) and to perform dimensionality reduction. [31][17]

- The **activation layers** or activation functions introduce non-linearity to the network,

allowing it to learn complex representations of the input data.

- The **pooling layers** reduce the spatial dimensions of the data, which helps to reduce overfitting and computational cost.

  - Max pooling is used to pick the feature with the highest activation in a small region of the input feature map
  - Average pooling is used to reduce the spatial size of the input data by taking the average of the values of a small region of the input feature map.

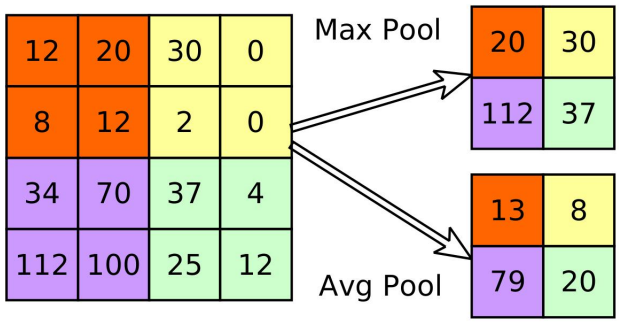

Figure 12: Average and max pooling

- **Linear layers** classify the features extracted by the convolutional layers into the desired output.

It should be noted that what is commonly referred to as 'convolution' in the context of convolutional neural networks (CNNs) is actually a cross-correlation operation, denoted with symbol $\star$ and convolution usually is used $*$. The term 'convolution' is used only for convention purposes. The basic concept behind cross-correlation is to take a small matrix, referred to as a kernel or filter, and slide it over the input data (such as an image or audio signal). At each position, the kernel is multiplied element-wise with the underlying data, and the results are summed to produce a single output value, referred to as a feature map.

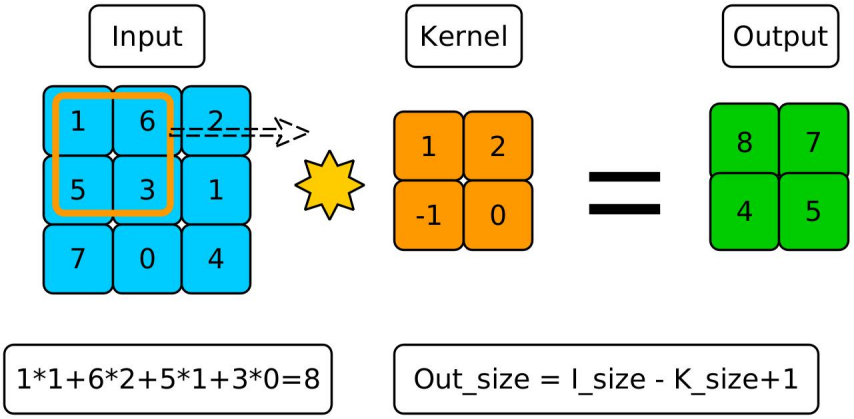

Figure 13: Basic cross-correlation operation

The output of the convolution operation is a feature map, where each element in the feature map is computed as follows:

$$\boldsymbol{Y}_{ij} = \sum \boldsymbol{K}_{ab} \circ \boldsymbol{I}_{i+a,j+b} \tag{54}$$

Where kernel(a,b) is the value of the filter at position (a,b), input(i+a,j+b) is the value of the input at position (i+a,j+b) and output(i,j) is the output value at position (i,j)

2.8.1 Channels

A channel refers to a specific feature or dimension of the input data. For example, in the case of image data, a channel can represent a color channel such as red, green, or blue. These channels are used to extract different features of the input image, and they are processed separately by the CNN. In our case study, we can use channels as a division between the real and imaginary parts, or for feature extraction of intercarrier symbol interference (ISI). We can have N channels as input and M channels as output, depending on how many features we want to deal with. In the image below, we show a case of 3 channel input and two channel output, also with a bias term.

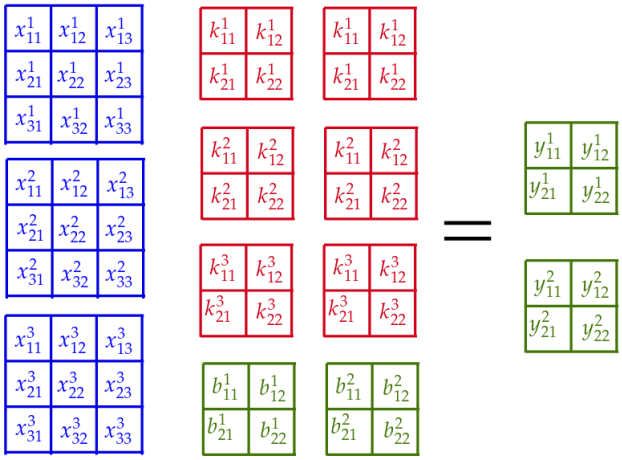

Figure 14: Convolutional Neural Network with 3-channel Input and 2-channel Output, including bias term

2.8.2 A Generalized View of Linear Layers through Convolutional Neural Networks

Let's take a more detailed look at the math, given the following terms.

- j input channels with$\boldsymbol{X}_j$ matrices
- d ouput channels $\boldsymbol{Y}_d$matrices and this ones an output size of $\boldsymbol{X}_j - \boldsymbol{K}_{ij}$
- $\boldsymbol{K}_{ij}$ kernels,where i maps to $\boldsymbol{Y}_d$ and j to $\boldsymbol{X}_j$
- $\bigstar$ cross-correlation

$$\boldsymbol{Y}_d = \boldsymbol{B}_d + \sum_{j=1}^{n} \boldsymbol{X}_j \bigstar \boldsymbol{K}_{ij} \tag{55}$$

We can think of the matrices as individual blocks and visualize them in the image below.

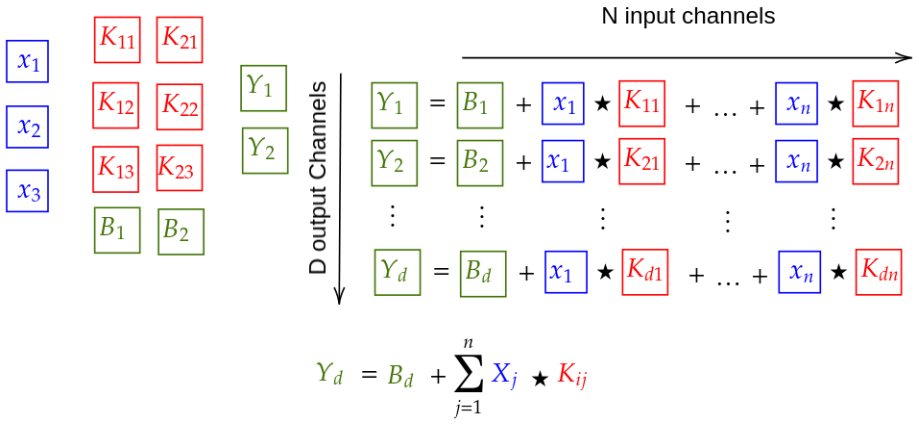

Figure 15: Y_d output given kernels

Finally, with the use of abstraction, we can further simplify the problem by representing it as a generalized version of tensors. The internal tensor is given by the sum of the cross-correlations between X channels and kernels. In a more general perspective, this can be viewed as the inner product of two tensors.

$$\boldsymbol{Y} = \boldsymbol{B} + \langle \boldsymbol{K}, \boldsymbol{X} \rangle \tag{56}$$

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_d \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_d \end{bmatrix} + \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ K_{d1} & K_{d2} & \cdots & K_{dn} \end{bmatrix} \dot{\times}\bigstar \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

$$Y = B + \langle K, X \rangle$$

Figure 16: Higher dimensionality abstract version

It's worth noting that when we consider a kernel of 1 dimension and an input of 1 channel, we can see that a dense layer is just a specific case of a 2D convolutional layer (Conv2D). Just as equation (43).

2.9 The Roadmap to 6G – AI Empowered Wireless Networks [32]

The integration of AI technologies in communication networks promises a more efficient and reliable future for 6G and beyond. The "intelligent PHY layer" paradigm, with its ability to self-learn and self-optimize, ensures that the system remains efficient and reliable despite various hardware and channel effects. Sometimes, the hardware referred to includes low-cost devices that may not be well fine-tuned but are commonly used. This AI-driven approach aims to adapt and optimize communication systems even when operating with less-than-ideal hardware components. This model leverages AI technologies to enhance communication efficiency and performance, and can autonomously learn and enhance performance through the integration of cutting-edge sensing and data-gathering tools.

Hardware heterogeneity necessitates system redesign for different hardware settings, which can be overcome using transfer learning. This approach adjusts the neural network weights to work with custom hardware architecture, regardless of the training on floating point or fixed point backpropagation. Therefore, network architecture is more crucial than numeric resolution in contrast to traditional methods. In the image bellow, a roadmap is presented that depicts the expected evolution of deep learning in the coming years.

In the same way the non-linear equalizers performr better than the neural network from low noise scnearios greater than 30db SNR values, this happens because the model trucate to error in the trainning stage, and to achive lower BER other preprocesing stages may be need. Therefore, to balance the trade-off between signal recognition over noise and overfitting, sacrificing performance beyond 30 dB should be one option if there are not other preprocesing stages different from the hermitian matrix.

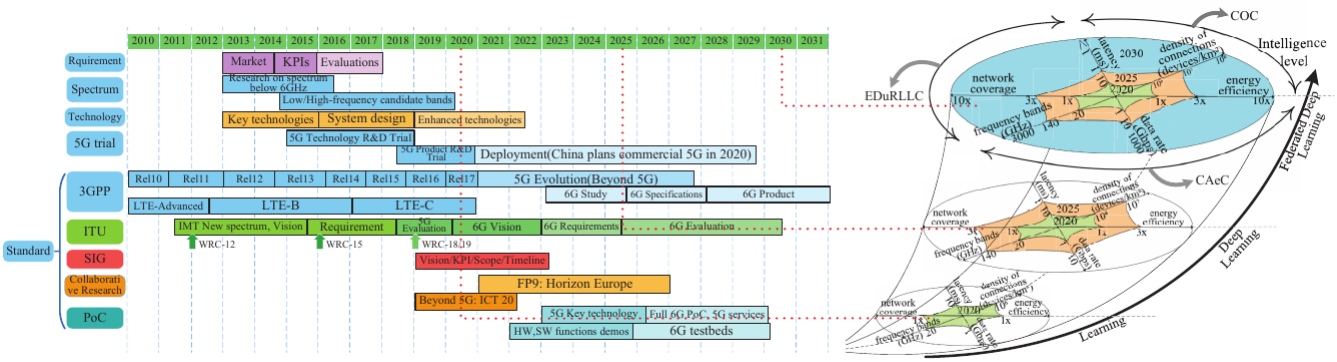

Figure 17: Roadmap showing the evolution of deep learning models in telecom and justifying our research [32]

It should be noted that much of the referenced work focuses on virtual physical resources and layers ranging from the MAC layer to higher ones. Despite this, it is essential to progress and supplement the ideas in the physical layer to implement an equalization stage effectively. By doing so, a more comprehensive and cohesive approach to communication system optimization can be achieved.

2.10 A Novel OFDM Equalizer for Large Doppler Shift Channel through Deep Learning [26]

This documentation describes the proposed neural architecture called Cascaded Net (CN) for equalization in OFDM systems. The use of a zero-forcing preprocessor aims to prevent the network from getting stuck in a saddle point or local minimum point. The CN neural architecture is designed to address the equalization problem in OFDM systems with Rayleigh fading and large Doppler shifts. By cascading a deep trainable network behind a zero-forcing preprocessor, the CN architecture achieves superior performance in comparison to traditional equalization methods.

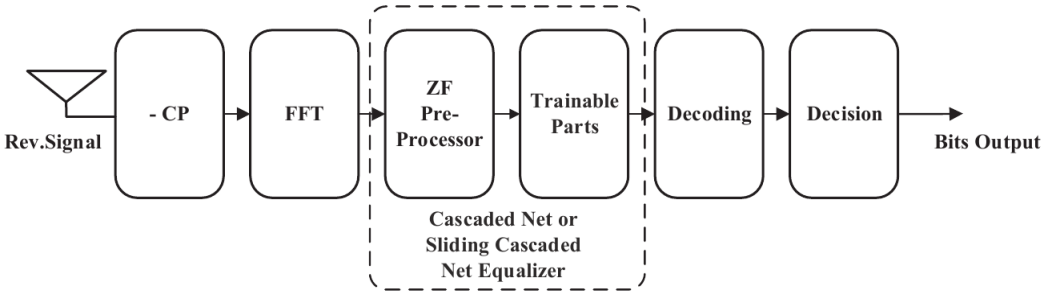

Figure 18: Deep learning equalizer with prepocesing stage. [26]

As they are performing frequency domain equalization, their base equation is the same as ours. $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{W}$

The Cascade net is defined as follows.

$$\boldsymbol{X_0} = (\boldsymbol{H^H}\hat{\boldsymbol{H}})^{-1}\boldsymbol{H^H}\boldsymbol{Y} \tag{57}$$

$$\boldsymbol{z_i} = \boldsymbol{w_i}\begin{bmatrix} \boldsymbol{H^H}\boldsymbol{Y} \\ \hat{\boldsymbol{X}}_{\boldsymbol{i}} \\ \boldsymbol{H^H}\boldsymbol{H}\hat{\boldsymbol{x}}_{\boldsymbol{i}} \end{bmatrix} + \boldsymbol{b_i} \tag{58}$$

$$\hat{\boldsymbol{X}_{\boldsymbol{i+1}}} = \boldsymbol{\phi}(\boldsymbol{z_i}) \tag{59}$$

Where $\boldsymbol{w}$ and $\boldsymbol{b}$ representes weights and bias as trainnable parameters.

$$\boldsymbol{\phi} = \boldsymbol{tanh}\left(\frac{\boldsymbol{x}}{|\boldsymbol{t_i}|}\right) \tag{60}$$

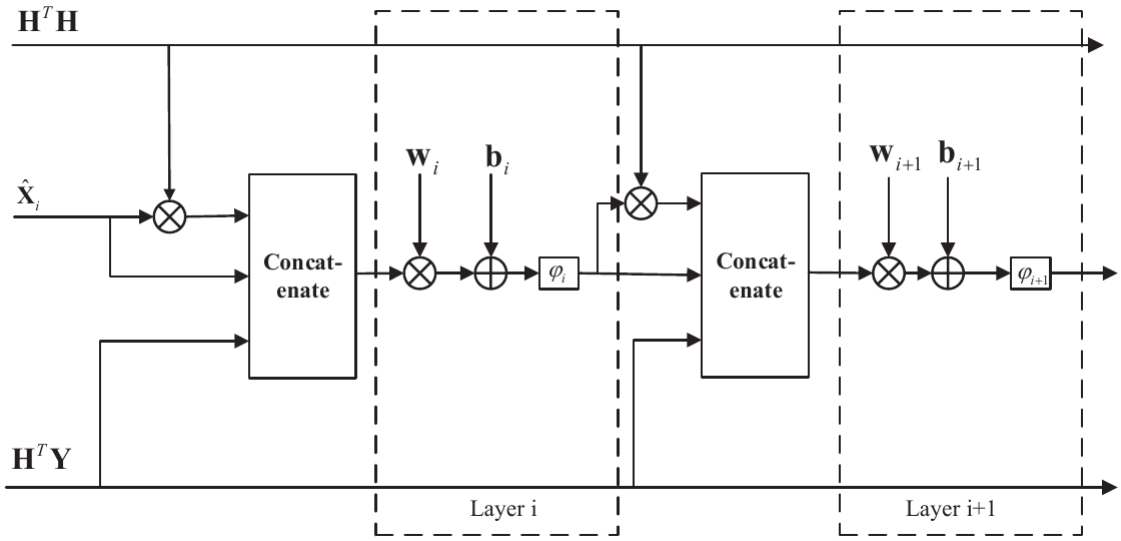

Figure 19: Cascade Net [26]

Their loss function or estimator is based on the Euclidean distance, with a logarithmic regularization of outliers and the objective of minimizing the distance between points. They accumulate the total estimation for each layer i and finally sum it up.

$$\mathbf{loss}(\mathbf{X}_\theta^{\mathbf{CN}}(\mathbf{H},\mathbf{Y})) = \sum_{\mathbf{i=1}}^{\mathbf{L}} \mathbf{log(i)}||\mathbf{X} - \hat{\mathbf{X}}||^{\mathbf{2}} \tag{61}$$

To deal with complex matrix number they make a reformulation in the matrix as follows.

$$\begin{bmatrix} \Re\{\mathbf{Y}\} \\ \Im\{\mathbf{Y}\} \end{bmatrix} = \begin{bmatrix} \Re\{\mathbf{H}\} & -\Im\{\mathbf{H}\} \\ \Im\{\mathbf{H}\} & \Re\{\mathbf{H}\} \end{bmatrix} \begin{bmatrix} \Re\{\mathbf{X}\} \\ \Im\{\mathbf{X}\} \end{bmatrix}$$

Figure 20: Matrix reformulation [26]

2.10.1 BER perfomance

The image below shows a benchmark of different equalizers, including ZF (Zero Forcing), PIC (Parallel Interference Cancellation), DET (Deep MIMO Detection Network)[52]], and CN (Cascaded Net). The modulation they used in the experiment below is QPSK. and they assume that CSI (Channel State Information) is perfect. The receiver is trained off-line for the single-user system with fixed Doppler shift. If Subcarrier number N is 32, f_N equals to 0.16.

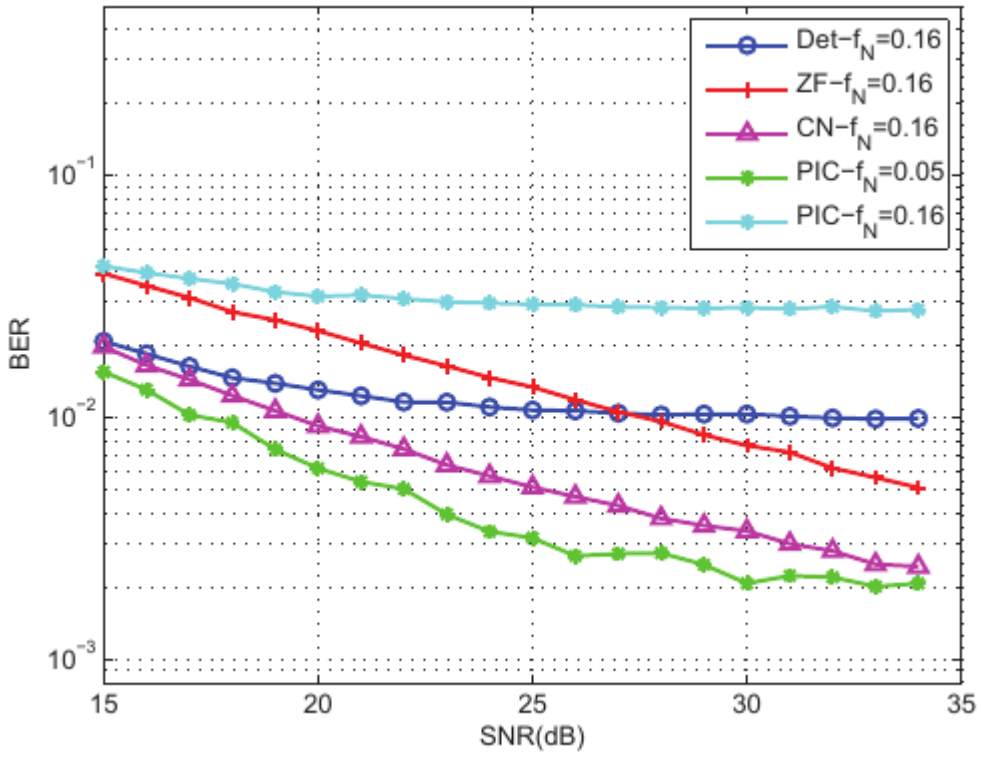

Figure 21: QPSK equalization with different strategies in the paper [26]

Note, that in practice, perfect CSI is often difficult to achieve due to the dynamic nature of the wireless channel, which can change rapidly due to factors such as fading, multipath propagation, and interference.

1. Classical PIC methods perform well in low subcarrier frequency scenarios (f_N less than 0.1). However, their performance degrades significantly with an increase in subcarrier frequency due to significant error propagation, as stated in the first section. Subcarrier frequency refers to the frequency at which the data is transmitted in a multi-carrier communication system, such as (OFDM). The data is divided into several subcarriers that are transmitted simultaneously, each using a different frequency.
2. Deep MIMO detection (Det) faces a flat error curve in high SNR scenarios.

   (a) Cascaded Net (CN) consistently performs well compared to Zero Forcing (ZF) and (Det), which is in line with the argument made in the third section.

2.11 An Introduction to Deep Learning for the Physical Layer [39]

This paper provides a detailed overview of neural networks and their application to channel equations, with formal mathematical description included. The paper describes the formal structure of feed-forward neural networks, as well as some applications of convolutional neural networks, that was described in the first section. It also introduces autoencoders for end-to-end communication systems and proposes the idea that an autoencoder can be used to characterize a complete channel. What's remarkable about this approach is that it can be extended to channel models and loss functions for which the optimal solutions are unknown, making it highly versatile.

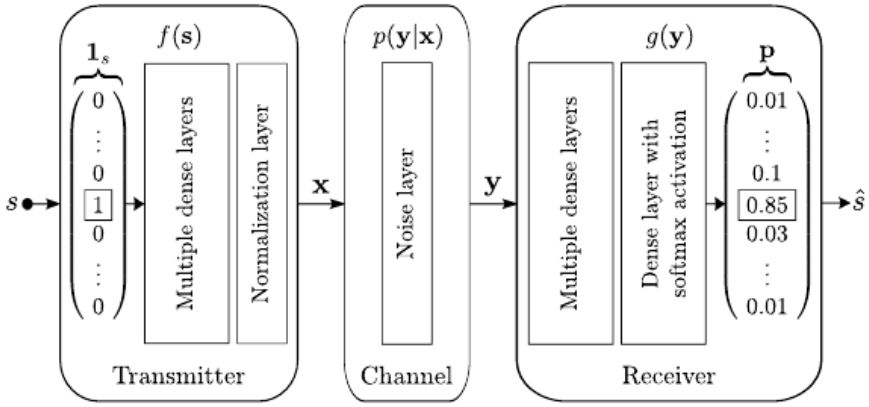

Figure 22: Autoencoder as Channel [39]

In addition, the authors use adversarial networks to manage multiple transmitter-receiver pairs with competing capacity. Although the Multiple-Input Multiple-Output (MIMO) scenario is not currently implemented, it could be a promising area for future work. Finally, the authors also discuss modulation classification, which involves using Convolutional Neural Networks (CNNs) to automatically detect the modulation scheme used in a communication process.

2.11.1 Autoencoder [6]

An autoencoder is a type of artificial neural network used primarily for unsupervised learning tasks, such as dimensionality reduction, feature extraction, and data compression. It is designed to learn efficient representations of input data by encoding and decoding the data through a neural network architecture. The primary goal of an autoencoder is to reconstruct the input data with the highest possible accuracy while learning a compact and meaningful representation of the data.

The architecture of an autoencoder can be likened to a communication system, consisting of two main components: the encoder, which acts as the transmitter, and the decoder,

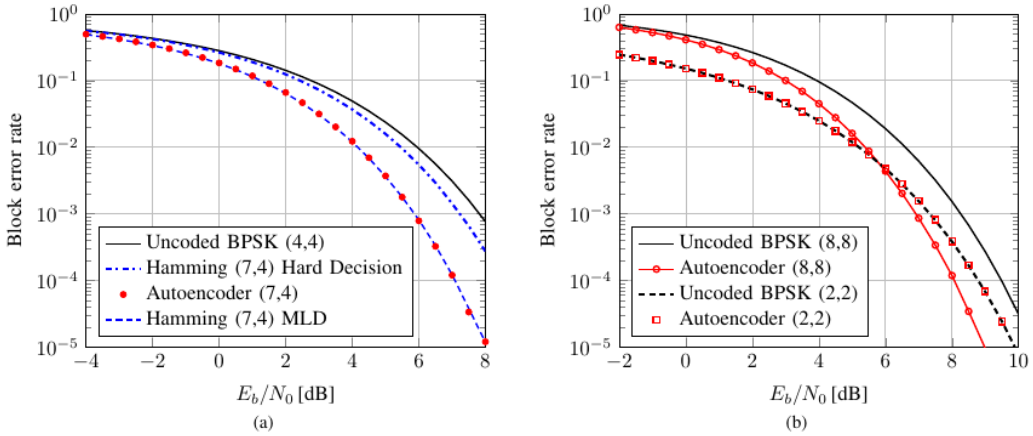

Figure 23: BLER for the AE in differen (n,k) baseline communication.[39]

which serves as the receiver. The encoder's responsibility is to compress the input data into a lower-dimensional representation, similar to the role of a transmitter in a communication system. This compressed representation, often referred to as the latent space or bottleneck, is then sent through the channel.

On the other hand, the decoder, which is analogous to the receiver in a communication system, takes the lower-dimensional representation from the channel and attempts to reconstruct the original input data. In essence, the autoencoder architecture mirrors the basic structure of a communication system, with the encoder and decoder functioning as transmitter and receiver, respectively.

The goal of the transmitter is to send one of M possible messages,$s \in M = \{1, 2, ..., M\}$, to the receiver using n discrete uses of the communication channel. To achieve this, the transmitter applies a transformation, $\boldsymbol{f} : \boldsymbol{M} \rightarrow \boldsymbol{Rn}$, to the message s, generating a transmitted signal $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{s}) \in \mathbb{R}^n$. Typically, the transmitter hardware imposes constraints on x, such as an energy constraint ($||\boldsymbol{x}||_2^2 \leq \boldsymbol{n}$), an amplitude constraint ($|\boldsymbol{xi}| \leq 1 \; \forall \boldsymbol{i}$), or an average power constraint ($\boldsymbol{E}|\boldsymbol{xi}| \leq 1 \forall \boldsymbol{i}$).

The communication rate of this system is $\boldsymbol{R} = \frac{\boldsymbol{k}}{\boldsymbol{n}}$ [bit/channel use], where $\boldsymbol{k} = \boldsymbol{log_2(M)}$.In the notation (n,k), the system is able to send one of $\boldsymbol{M} = \boldsymbol{2^k}$ messages (i.e., k bits) through n channel uses. The communication channel is characterized by the conditional probability density function $\boldsymbol{p(y|x)}$, where $\boldsymbol{y} \in \boldsymbol{Rn}$ represents the received signal. After receiving $\boldsymbol{y}$, the receiver applies a transformation $\boldsymbol{g} : \mathbb{R}^n \rightarrow \boldsymbol{M}$ to generate the estimate $\hat{\boldsymbol{s}}$ of the transmitted message $\boldsymbol{s}$. As finall loss function it is used cross entropy loss, which is mention in chapter one.

2.11.2 Analysis

The paper examined the possibility of using an autoencoder to learn channel represen-

tation by incorporating all available information in the system. However, it did not explore the use of more complex constellations, such as 16-QAM, and the frame size used in this research was smaller than intended. Despite these limitations, the concepts related to coding and decoding data in different dimensional spaces offer valuable insights for the project. Thus, this paper is useful for our research, particularly for developing encoding-based solutions.

2.12 A Survey of Complex-Valued Neural Networks [7]

It is widely recognized that IQ data is represented by complex numbers, which can pose challenges when working with them. This paper offers a detailed description of a new loss function and the necessary adjustments required to perform backpropagation in complex networks. Additionally, the paper demonstrates how to preprocess data, activation functions, and cost functions, which can be valuable for conducting experiments with complex data or complex neural networks.

2.12.1 Normalize unitary circle

If we look at data in the complex plane we should first at deal with normalization, and normalization is given by diviing the data by his absolute max, which lead all poins inside a unitary circle.

$$f(z) = \frac{z}{max(|z|)} \tag{62}$$

The image below demonstrates how a complex network can handle polar segmentation, and how weights and biases can be used to define a vector in the complex plane

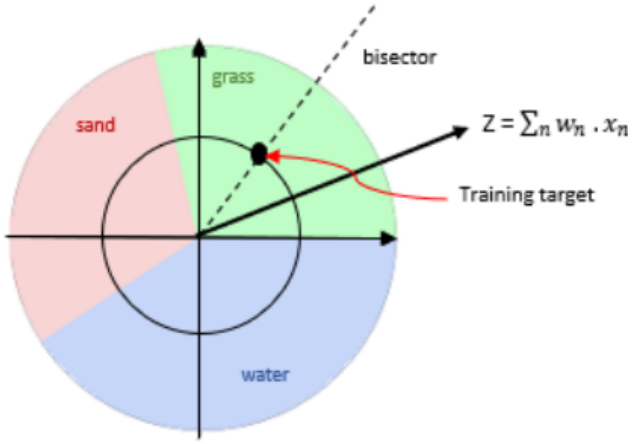

Figure 24: A Geometric Interpretation of Segmentation and Function Evaluation in the Complex Plane. [7]

2.12.2 Activation functions

The hyperbolic tangent is an example of a fully complex activation function and has been utilized in [34]. It is apparent that singularities in the output may arise due to values on the imaginary axis. To prevent an explosion of values, it is necessary to appropriately scale the inputs, which mention in the normalization in the section above. According to some researchers, imposing the strict constraint of requiring the activation function to be holomorphic may not be necessary. A holomorphic function is a complex-valued function of a complex variable that is complex differentiable at every point within its domain. In other words, a function $f(z)$ is holomorphic at a point z if the limit of the difference quotient of $f(z)$ as z approaches that point exists and is independent of the path of approach. Therefore, for a basic implementation, we should split the evaluation into real and imaginary parts as separate sections to activate the values. It's important to note that the real and imaginary parts should not be mixed.

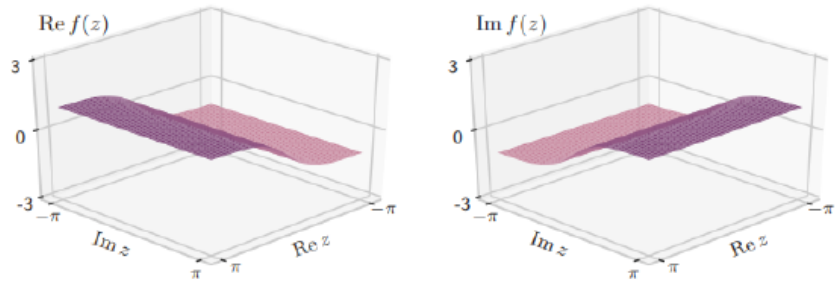

Figure 25: Complex function separate for each real and imaginary part. [8]

2.12.3 Loss functions

One approach to dealing with the error is to take the complex magnitude of the differences between the estimator and the ground truth. Given $\boldsymbol{d} \in \mathbb{C}^N$(ground truth) and $\boldsymbol{o} \in \mathbb{C}^N$(estimated output), the error $\boldsymbol{e} = \boldsymbol{d} - \boldsymbol{o}$ we can calculate the complex mean square loss in a non-negative scalar.

$$\boldsymbol{L(e)} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} |\boldsymbol{e_k}|^{\boldsymbol{2}} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} \boldsymbol{e_k}\overline{\boldsymbol{e_k}} \tag{63}$$

If we take a polar approach where $d = re^{i\phi}$ and $o = \hat{r}e^{i\hat{\phi}}$, we can convert the error into a log function to bring points closer and cancel the exponentials. Rewritten in this way, the equation becomes:

$$(\boldsymbol{e_{log}}) := \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} \boldsymbol{e_k}\overline{\boldsymbol{e_k}} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} (\boldsymbol{log(o_k) - log(d_k)}) * \overline{(\boldsymbol{(log(o_k) - log(d_k))})} \tag{64}$$

$$\boldsymbol{log}\left(\tfrac{\boldsymbol{o_k}}{\boldsymbol{d_k}}\right) * \overline{\boldsymbol{log}\left(\tfrac{\boldsymbol{o_k}}{\boldsymbol{d_k}}\right)} = \boldsymbol{log}\left(\tfrac{\hat{\boldsymbol{r}}\boldsymbol{e}^{i\hat{\boldsymbol{\phi}}}}{\boldsymbol{re}^{i\boldsymbol{\phi}}}\right) * \boldsymbol{log}\left(\tfrac{\hat{\boldsymbol{r}}\boldsymbol{e}^{-i\hat{\boldsymbol{\phi}}}}{\boldsymbol{re}^{-i\boldsymbol{\phi}}}\right) = \boldsymbol{log}\left(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}\boldsymbol{e}^{i(\hat{\boldsymbol{\phi}}-\boldsymbol{\phi})}\right) * \boldsymbol{log}\left(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}\boldsymbol{e}^{-i(\hat{\boldsymbol{\phi}}+\boldsymbol{\phi})}\right)$$

$$\left[\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) + \boldsymbol{log}\left(\boldsymbol{e}^{i(\hat{\boldsymbol{\phi}}-\boldsymbol{\phi})}\right)\right] * \left[\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) + \boldsymbol{log}\left(\boldsymbol{e}^{-i(\hat{\boldsymbol{\phi}}-\boldsymbol{\phi})}\right)\right] =$$

$$\left[\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) + \boldsymbol{i}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\right] * \left[\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) - \boldsymbol{i}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\right] =$$

Multiply out the terms

$$\boldsymbol{log^2}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) + \boldsymbol{i}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) - \boldsymbol{i}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})\boldsymbol{log}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) - \boldsymbol{i^2}(\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})^2$$

Simplify

$$\boldsymbol{log^2}(\tfrac{\hat{\boldsymbol{r}}}{\boldsymbol{r}}) + (\hat{\boldsymbol{\phi}} - \boldsymbol{\phi})^2$$

We multiply the angle and radius by 0.5 in the loss function to give them equal importance since both are equally significant for the loss.

$$\boldsymbol{Loss}(e_{log}) = \frac{1}{2}\left(\boldsymbol{log}\left[\frac{\hat{r}}{r}\right]^2 + \left[\hat{\phi} - \phi\right]^2\right) \tag{65}$$

We now have an explicit representation of magnitude and phase in the loss function, which could be helpful in the polar equalization approach.

2.13 MobileNet [25]

MobileNet achieves its high efficiency by using depthwise separable convolutions, which are a combination of a depthwise convolution and a pointwise convolution, while still maintaining high accuracy. In terms of time complexity, MobileNet has a lower multiplications compared to regular Convolutional Neuronal Networks. This results in faster inference times and lower memory requirements,as its name says usefull for mobile applications, also include embbeded system applications for edge computing. It is important to mention that MobileNet's efficiency comes at the cost of slightly lower accuracy compared to conventional CNNs.

2.13.1 Depthwise Separable Convolution

Convolution is a mathematical concept that measures the overlap between two functions as one of them slides over the other. Mathematically, it can be expressed as a sum of products. However, standard convolution operations can be slow to perform due to the number of multiplications required. An alternative method called depth-wise separable convolution can be used to speed up the process. This method breaks down the convolution process into two parts: depthwise convolution and pointwise convolution.

Let us briefly review the fundamental concepts of convolution on an input volume. Let's take an input volume F with dimensions DF x DF x M, where DF represents the width and height of the input volume and M is the number of input channels. In the case of a color image, M would be equal to 3 for the R, G, and B channels. We perform convolution on a kernel K, which has dimensions DK x DK x M. The output will be in the shape of DG x DG x 1. When we apply N kernels to the input, we obtain an output volume G with dimensions DG x DG x N.

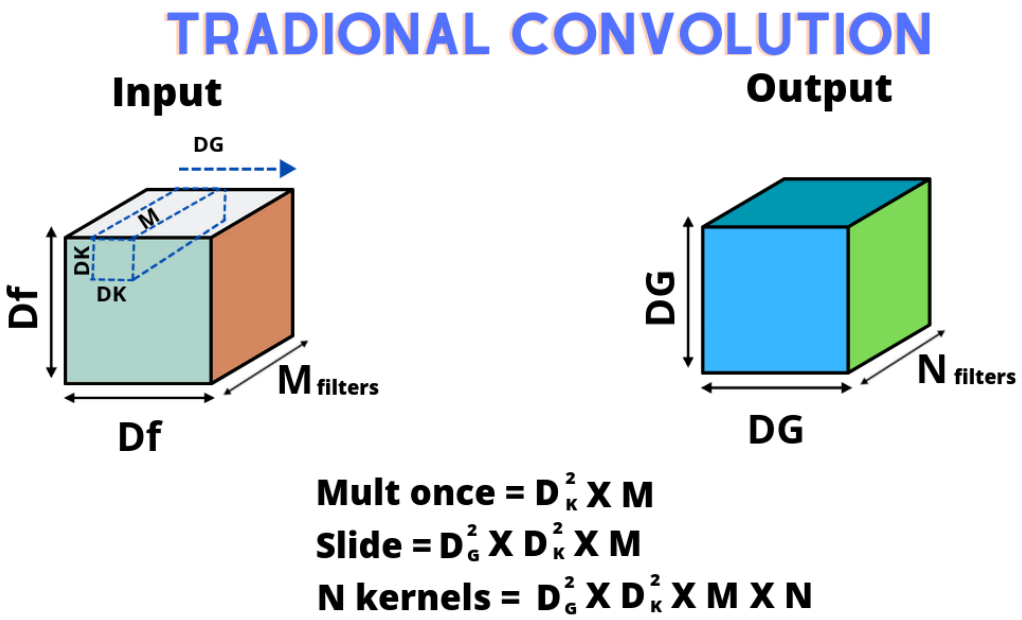

Figure 26: Traditional convolution

In a traditional convolution, filters are applied across all input channels and their values are combined in a single step. However, in depthwise separable convolution, this process is split into two stages. The first stage is depthwise convolution, which applies convolution to a single input channel at a time. To perform depthwise convolution, we use filters or kernels K, which are of shape DK x DK x 1. Here, DK is the width and height of the square kernel, and it has a depth of 1 because this convolution is only applied to a single channel. Therefore, we require M such DK x DK x 1 kernels over the entire input volume F, where F has a shape DF x DF x M. For each of these M convolutions, we get an output of DG x DG x 1 in shape. By stacking these outputs together, we obtain an output volume G of shape DG x DG x M, which marks the end of the first phase, that is, the end of depthwise convolution. The number of multiplications in the depthwise convolution phase is obtained by applying these multiplications to all M input channels separately, with each channel having its own kernel. Therefore, the total number of multiplications in this phase is :

$$DW = M \times {D_G}^2 \times {D_k}^2 \tag{66}$$

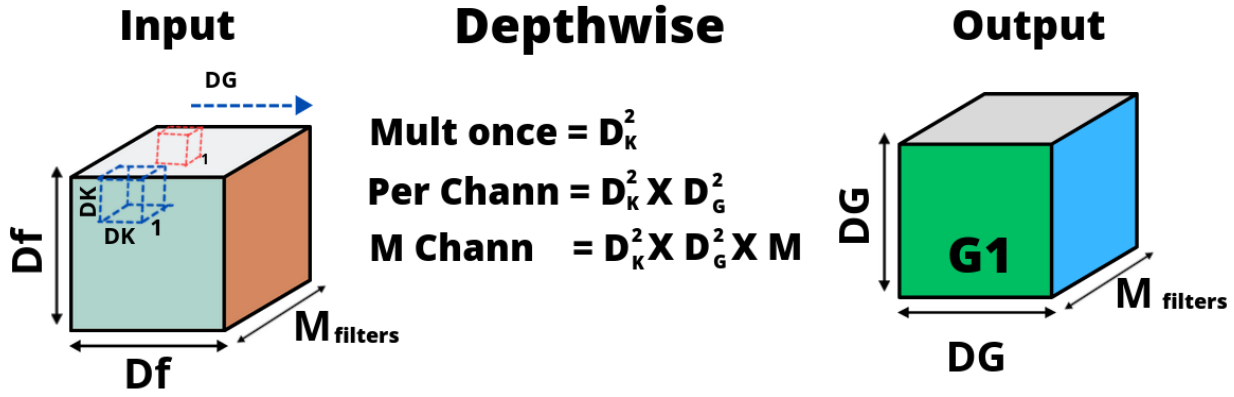

Figure 27: Depthwise convolution

Pointwise convolution refers to a 1x1 convolution operation applied to each of the output channels generated by the depthwise convolution. In this step, the input is a volume of shape DG x DG x M, where M is the number of output channels generated by the depthwise convolution. The filter used for this operation, denoted as KPC, has a shape of 1 x 1 x M, which means that it is applied across all M output channels. The resulting output has the same width and height as the input DG x DG, and the number of output channels can be controlled by using N filters. Therefore, the final output volume of the depthwise separable convolution has a shape of DG x DG x N. And hence, the number of multiplications for one instance of convolution is M. This is applied to the entire output of the first phase, which has a width and height of DG. So the total number of multiplications for this kernel is DG x DG x M. So for some N kernels, we'll have this multiplications :

$$PW = N \times DG \times DG \times M \tag{67}$$

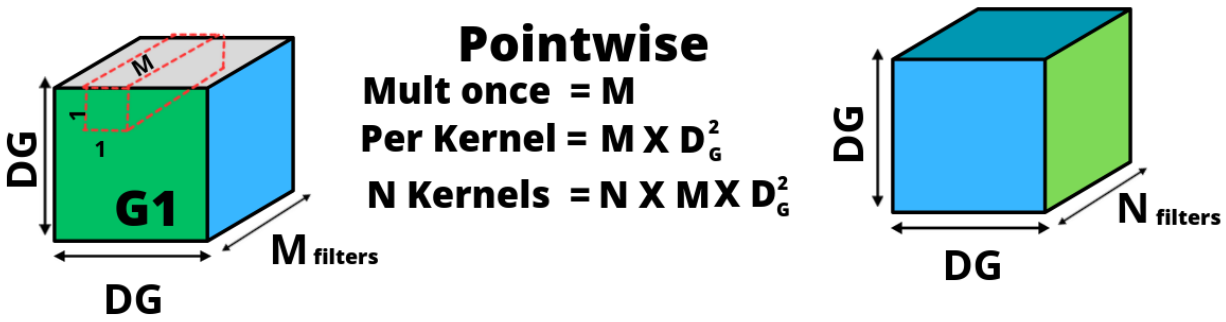

Figure 28: Pointwise convolution

Hence, the total multiplication count is the sum of multiplication counts in the depthwise and pointwise convolution stages.

$$\mathbf{Total} = \mathbf{DW} + \mathbf{PW} = \mathbf{M} \times \mathbf{D_G}^2 \times \mathbf{D_k}^2 + \mathbf{N} \times \mathbf{DG} \times \mathbf{DG} \times \mathbf{M} \tag{68}$$

$$\boldsymbol{Total} = \boldsymbol{M} \times \boldsymbol{D_G^2}(\boldsymbol{D_K^2} + \boldsymbol{N}) \tag{69}$$

We can compare the computational efficiency of standard convolution with depthwise convolution by computing their ratio. This ratio is obtained by summing the reciprocal of the depth of the output volume, denoted as N, and the reciprocal of the squared dimensions of the kernel, denoted as DK.

$$\frac{\boldsymbol{DepthWise}}{\boldsymbol{Standard}} = \frac{\boldsymbol{M} \times \boldsymbol{D_G^2}(\boldsymbol{D_K^2} + \boldsymbol{N})}{\boldsymbol{N} \times \boldsymbol{D_G^2} \times \boldsymbol{D_k^2} \times \boldsymbol{M}} = \frac{\mathbf{1}}{\boldsymbol{N}} + \frac{\mathbf{1}}{\boldsymbol{D_k^2}} \tag{70}$$

To better understand this, let's take an example. Suppose the output feature volume N is 1024, and the kernel size is 3, which means DK is 3. Plugging these values into the equation, we obtain a ratio of 0.112. This indicates that standard convolution requires 9 times more multiplications than depthwise separable convolution. This significant difference in computational power can have a considerable impact on the performance and efficiency of convolutional neural networks.

Table 1: Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Mult-Adds (Million)	Parameters (Million)
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

2.13.2 MobileNetV3 [24]

The MobileNetV3 architecture uses a combination of depthwise separable convolutions, linear bottlenecks, and other optimizations to reduce the number of parameters and computational complexity while maintaining high accuracy. It also includes new design elements, such as dynamic activation functions and network architecture search techniques, to improve performance on a variety of tasks.

Table 2: MobileNet architectures

Model	**Input Resolution**	**FLOPS**	**Parameters**
MobileNet v1	224x224	569 M	4.2 million
MobileNet v2	224x224	300 M	3.4 million
MobileNet v3	224x224	219 M	5.4 million

FLOPS stands for "Floating Point Operations Per Second". It is a measure of a computer's performance based on the number of floating point operations (such as additions,

subtractions, multiplications, and divisions) it can perform in a second. It is commonly used to measure the performance of computer processors or the computational requirements of machine learning models.

2.13.3 Reduced complexity for activation functions

One of the key components of his low computational cost is the implementation of hard-sigmoid and hardswish function, which is quite similar that we done with hardtanh but with other activation function type.

$$hswish[x] = x\frac{RELU6(x+3)}{6} \tag{71}$$

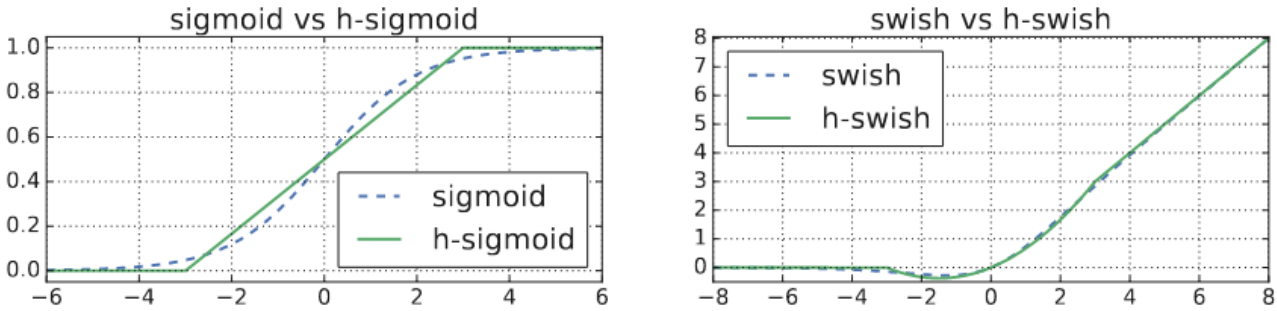

Figure 29: Hard functions [24]

These functions are integrated into the final stage of MobileNet to serve as more efficient functions.

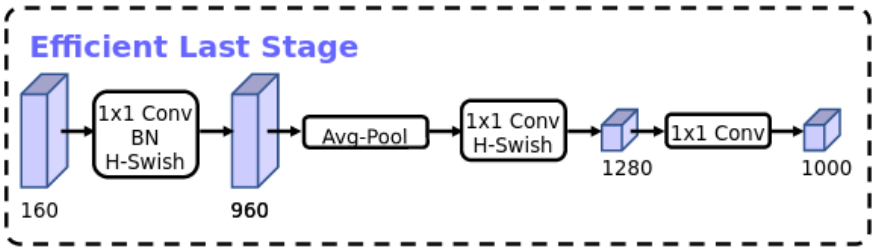

Figure 30: Last section optimized [24]

2.14 Recent Advances in Neural Network Techniques For Channel Equalization:A Comprehensive Survey [68]

This paper provides an overview of various channel equalization methods, including the Multilayer Perceptron (MLP) equalizer, Functional Link Artificial Neural Network (FLANN) equalizer, Chebyshev Neural Network (NN) equalizer, and Radial Basis Function NN

(RBFNN) equalizer. Additionally, it presents a literature review and application of Recursive Neural Network (RNN) and Fuzzy Neural Network equalizers.

2.14.1 FLANN

The primary difference between FLANN hardware and MLP configuration is that the non-linear mapping replaces only the input, output, and hidden layers. This mapping uses a non-linear function to transform the input vector, mapping it to a higher-dimensional space. The expansion function, called the functional link, is typically a polynomial function of the input variables. In our final experiments, we explored the concept of searching for equalization in a higher-dimensional space, but did not utilize the polynomial function approach.

2.14.2 RBF

Radial basis functions (RBFs) are a type of basis function used in function approximation and machine learning algorithms. RBFs are a class of functions that depend only on the distance from the center of the function, and their output decreases as the distance from the center increases. Gaussian RBF: This function takes the form of $e^{(-r^2/2)}$, where r is the Euclidean distance from the center of the function. This RBF is widely used in machine learning and function approximation algorithms due to its smoothness and symmetry.

2.14.3 RNN

The article concludes by stating that Recurrent Neural Networks (RNNs) generally outperform feed-forward neural networks (FNNs) and other methods. RNNs approximate a finite impulse response (IIR) filter, whereas other methods approximate a finite impulse response (FIR) filter. It is worth noting that IIR filters are known to be unstable, but recent advances in neural networks, such as LSTM, GRU, and Transformers, have been developed to overcome this limitation

2.14.4 Overview and Insights

Based on research, Recurrent Neural Networks (RNNs) [20] are the most effective solutions for BER performance. RNNs have made significant advancements over the years, leading to the emergence of powerful tools like Sequence to Sequence [61], which can handle longer sequences with the Attention mechanism. This development allows the model to weigh different parts of the input sequence based on their relevance to the current decoding step, improving interpretability of the model's predictions. Next this models

has an evolution on transformers [64] models, which had emerged as an alternative, relying solely on self-attention to process input sequences, leading to greater parallelization during training and reduced computational complexity compared to RNN-based models. However, it's important to note that the best results found in this research are based on an outdated evolution of sequential models. The latest research on transformers and attention mechanisms will be presented to explain these advancements and show readers how to merge the latest advances in signal equalization and deep learning to create new state-of-the-art models. The figure below shows an evolution of sequential networks to illustrate how outdated RNNs are.

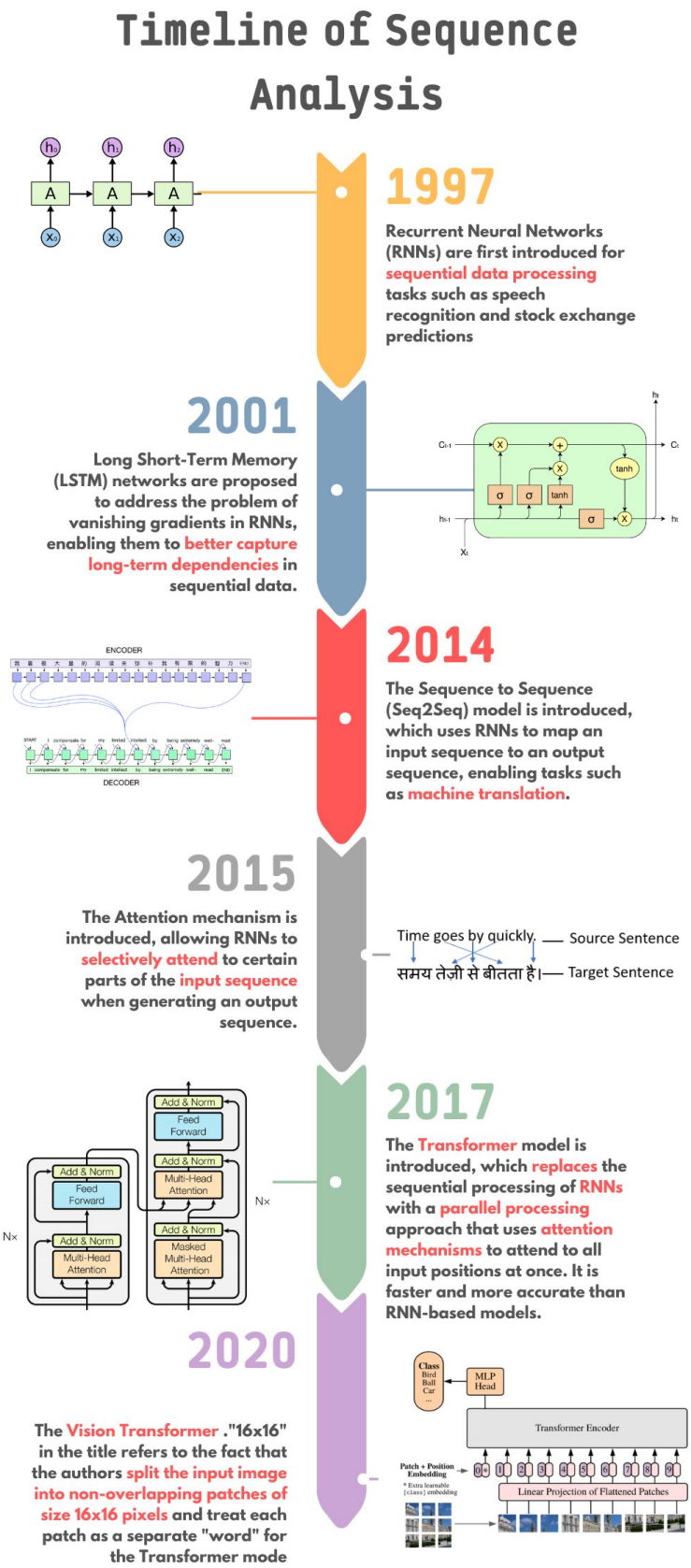

Figure 31: From RNN to Image Transformers

2.15 Attention Is All You Need [64]

"Attention is All You Need" is a research paper published by Google in 2017 that introduced the Transformer model, a neural network architecture for sequence-to-sequence modeling. The Transformer model is designed to handle variable-length sequences of input data and generate variable-length output sequences, making it particularly useful for tasks such as machine translation and natural language processing.

The Transformer model differs from previous neural network architectures by relying solely on attention mechanisms for input and output processing, eliminating the need for recurrent[20] and convolutional layers[31]. The attention mechanism[4] allows the model to focus on different parts of the input sequence when generating the output sequence, making it more accurate and efficient than previous models.

2.15.1 Embedding

The embedding refers to the process of representing **discrete input** tokens as **continuous vectors** that can be processed by the model. This is achieved using an embedding layer, which maps each input token to a high-dimensional $\mathbb{R}^N$ vector in a learned embedding space. Words cannot be represented as numbers directly. Normally, numerical data is transformed into a discrete representation, but in order to process it through a transformer network, these discrete representations need to be mapped to a continuous vector space using an embedding layer.

The input size of an embedding layer is typically the size of the vocabulary $|A|$, i.e., the number of unique tokens that the model can expect to encounter in the input. For example, if the vocabulary size is 10,000, then the input size of the embedding layer would be 10,000. The output size of the embedding layer is determined by the desired dimensionality of the embedding space $\mathbb{R}^M$, which is a hyperparameter that can be tuned by the model developer.

Normally, each word is assigned a unique index, and this index corresponds to a row in the embedding matrix. To obtain the embedding vector for a word, we retrieve the corresponding row from the matrix and multiply it by each element in row to get the new vector V.

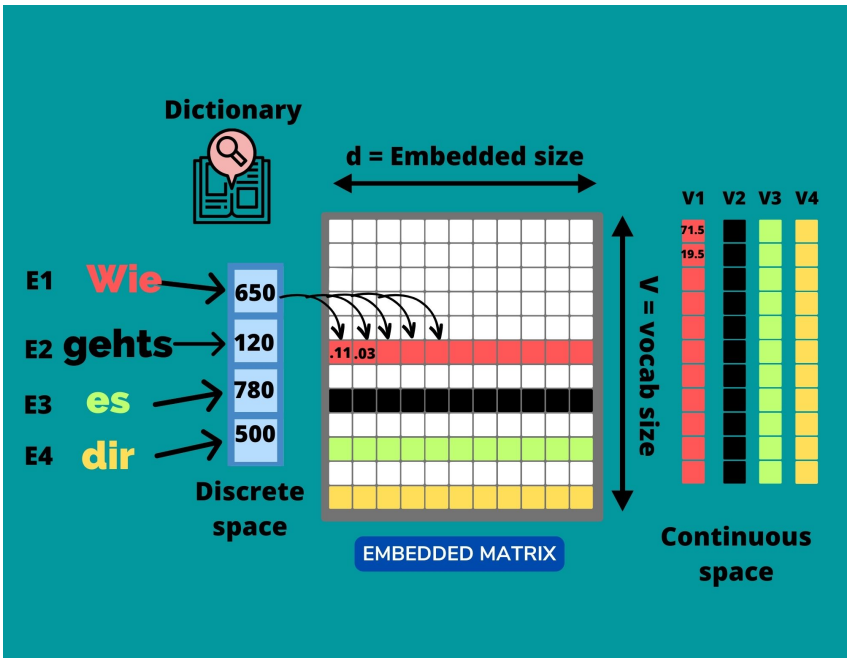

Figure 32: Embbeded Matrix

The embedding layer allows the model to capture the semantic relationships between different input tokens, and enables it to perform tasks such as language modeling and machine translation. Is typically followed by positional encoding, which adds information about the position of each input token in the sequence. Together, the embedding and positional encoding components provide a way for the model to process variable-length input sequences of discrete tokens in an efficient and effective manner.

Let X be an input sequence of length T, where each element x_i is an integer representing the i-th token in the vocabulary. Let E be a learned embedding matrix of size$V \times d$, where V is the size of the vocabulary and d is the dimension of the embedding space.

The embedding layer can be defined as follows:

$$\mathrm{E}(X) = [\mathrm{e}_1, \mathrm{e}_2, \ldots, \mathrm{e}_T] \tag{72}$$

Positional encoding can be added to the embeddings as follows:

$$\mathrm{E'}(X) = [\mathrm{e'}_1, \mathrm{e'}_2, \ldots, \mathrm{e'}_T] \tag{73}$$

where $\mathrm{e'}_i = \mathrm{e}_i + \mathrm{PE}_i$, and PE_i is the i-th row of a learned positional encoding matrix of size $T \times d$.

The resulting embeddings $\mathrm{E'}(X)$ are continuous vectors in a high-dimensional embedding space that can be processed by the transformer model.

2.15.2 Multihead attention

One of the key components of the Transformer model is multi-head attention. Multi-head attention allows the model to attend to different parts of the input sequence simultaneously, by splitting the input data into multiple representations and computing attention on each of them. This allows the model to capture complex relationships between different parts of the input sequence, and enables it to handle long sequences more efficiently. In multi-head attention mechanism of a transformer model, the input sequence is split into multiple vectors (heads), and each of these vectors is processed independently. The attention mechanism then operates on these vectors to compute weighted combinations that represent different aspects of the input sequence.The multi-head attention mechanism consists of three linear transformations: Query, Key, and Value. These transformations are learned parameters that are used to compute the attention scores and weights for each head.

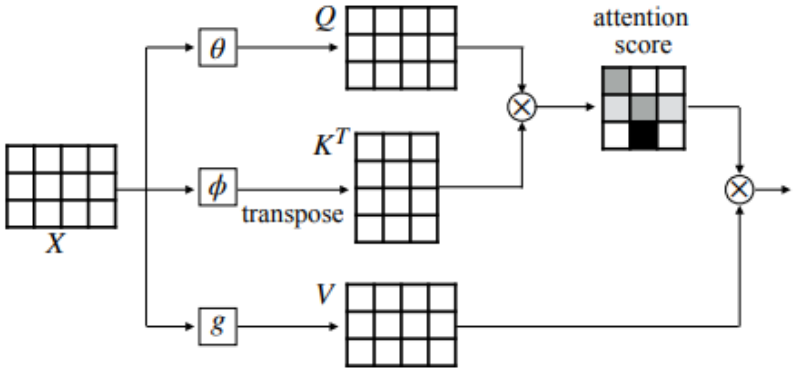

Figure 33: Attention respect to input vectors [57]

- Query: This transformation takes the current decoder state as input and maps it to a query vector. The query vector is used to compute the similarity between the decoder state and each of the key vectors.
- Key: This transformation takes the encoder output as input and maps it to a key vector. The key vector is used to compute the similarity between the decoder state and each of the query vectors.
- Value: This transformation takes the encoder output as input and maps it to a value vector. The value vector is used to compute the weighted sum of the encoder output, based on the attention weights calculated from the query and key vectors.

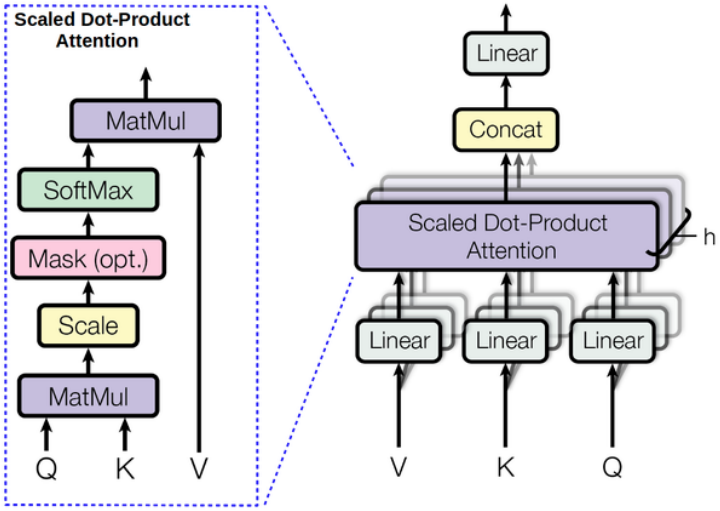

Figure 34: Multihead attention [64]

Let Q, K, and V be the query, key, and value matrices, respectively, for one head of the multi-head attention mechanism. Let d_k be the dimension of the key vectors, and let d_v be the dimension of the value vectors. Let h be the number of heads. Then, the multi-head attention mechanism can be defined as follows:

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, headh_h)W^O \tag{74}$$

where

$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and

$W_i^Q \in \mathbb{R}^{dmodel \times d_k}$,$W_i^K \in \mathbb{R}^{d_{model} \times d_k}$,$W_i^V \in \mathbb{R}^{d_{model} \times d_v}$and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are learned weight matrices.

The Attention function can be defined as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}} + Mask)V \tag{75}$$

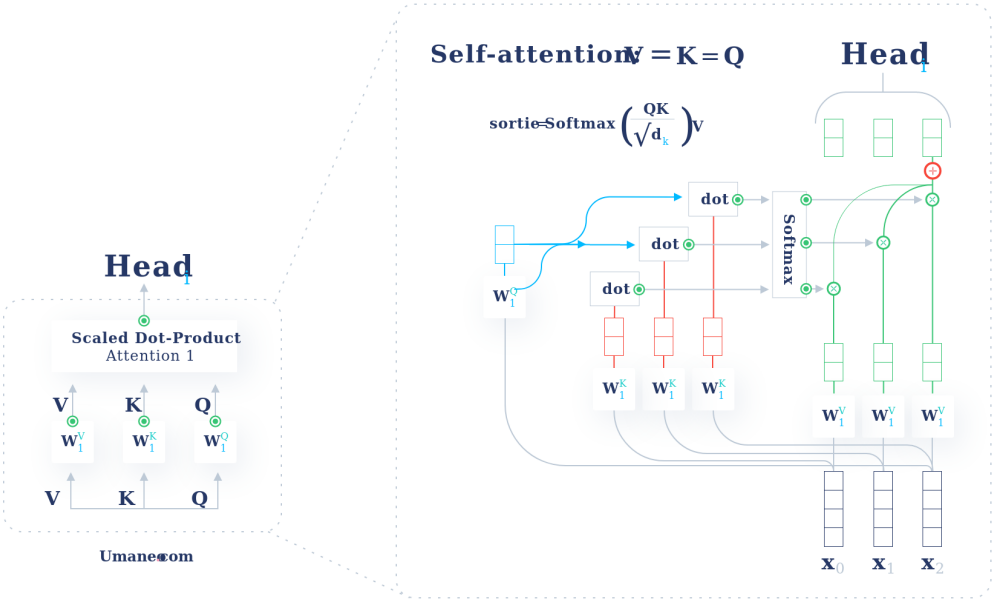

Figure 35: Self attention dot product [57]

2.15.3 Mask

In multi-head attention mechanism of a transformer model, a mask is a binary matrix that is used to selectively prevent certain elements of the input sequence from being attended to by the model.

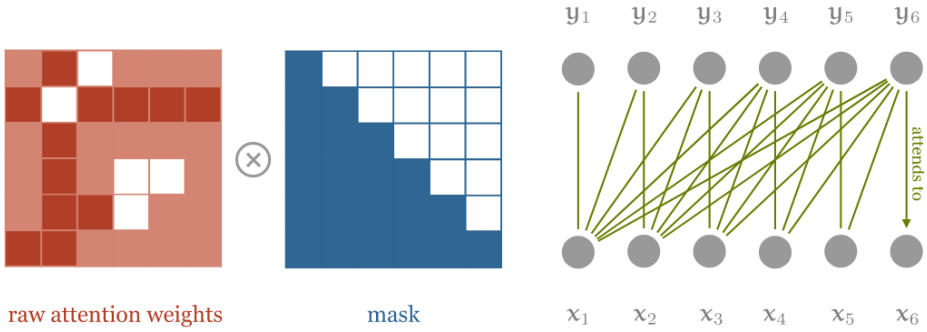

Figure 36: Mask example

There are two types of masks commonly used in the transformer model: padding masks and sequence masks.

- Padding masks: These masks are used to ignore padding tokens in the input sequence, which are added to ensure that all input sequences are of the same length. Padding tokens have no semantic meaning and should not be attended to by the

model. A padding mask is a binary matrix that has a value of 0 for padding tokens and 1 for all other tokens.

- Sequence masks: These masks are used to ensure that each position in the output sequence can only attend to positions that have already been processed. This is important for tasks such as language modeling, where the model is trained to predict the next word in a sequence based on the previous words. A sequence mask is a binary matrix that has a value of 0 for all future positions in the sequence and 1 for all other positions.

Masks are applied to the attention mechanism by adding them to the attention weights before computing the weighted sum of the input sequence. The mask ensures that certain elements of the input sequence are not attended to by the model, which can improve the accuracy and efficiency of the model.

2.15.4 Encoder Decoder

Once we have all building blocks we can say that transformer model consists of an encoder and a decoder. The encoder processes the input sequence, using multi-head attention and position encoding to create a fixed-length representation of the input. The decoder then generates the output sequence, using multi-head attention and position encoding to attend to the input representation and generate each output token. The model is trained using maximum likelihood estimation, where the goal is to minimize the cross-entropy loss between the predicted output sequence and the ground truth.

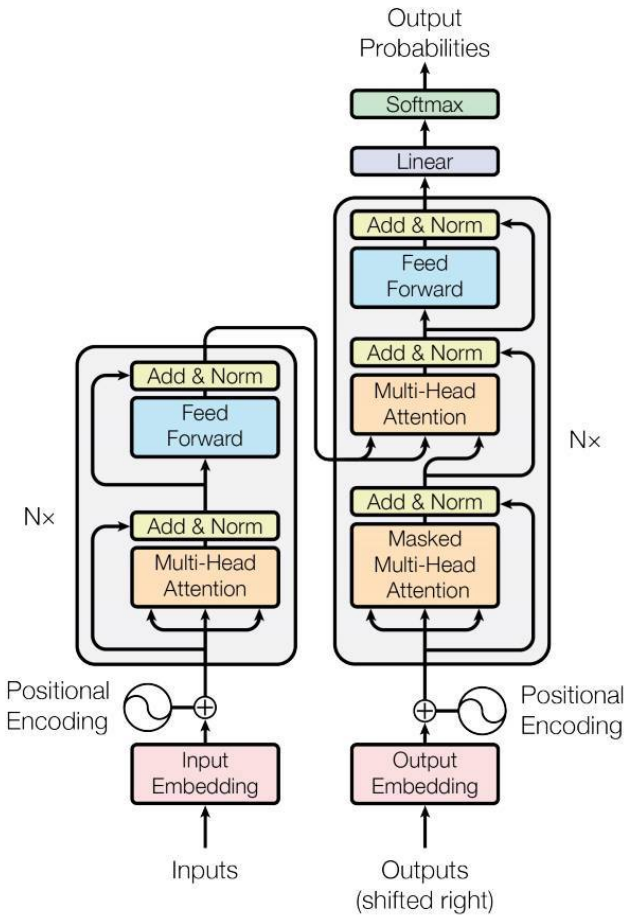

Figure 37: Transformer with encoder and decoder sections

In the transformer diagram, "Nx" is typically used to represent the number of encoder-decoder layers in the model. For example, a transformer model with 6 encoder layers and 6 decoder layers might be represented as "Nx=6" in the diagram.

The output of the Transformer model is a sequence of tokens, which can be interpreted as a sequence of words, or other discrete units depending on the task at hand. These tokens can be further processed by other components of a larger system, such as a language model or a downstream task-specific model.

2.15.5 Autoregresion

In an autoregressive mode in the context of an autoencoder, the decoder network is designed to generate the output sequence one element at a time, based on the previously generated elements. In other words, the decoder network takes the previous elements of the output sequence as input and generates the next element of the sequence. This is in

contrast to a non-autoregressive mode, where the decoder network generates the entire output sequence at once, without considering the previously generated elements.

2.15.6 Overview and Insights

In the equalization stage, the Transformer architecture can be applied to cancel Inter-Symbol Interference (ISI) using attention mechanisms, as the data is sequential and can be interpreted symbol by symbol. However, positional encoding may be lost when treating continuous data, which poses a challenge. Furthermore, the Transformer architecture outputs symbols instead of continuous values, necessitating a mechanism to discretize the received data Y and apply a token for each position, treating equalization as a machine translation.

2.16 An image is worth 16x16 words [13]

The Vision Transformer is a deep learning architecture that is specifically designed for image classification tasks. Unlike traditional Convolutional Neural Networks (CNNs) that use convolutional layers to extract features from images, the ViT uses self-attention mechanisms to capture relationships between different parts of the image. Its architecture takes as input a sequence of patches, where each patch represents a small square region of the image.

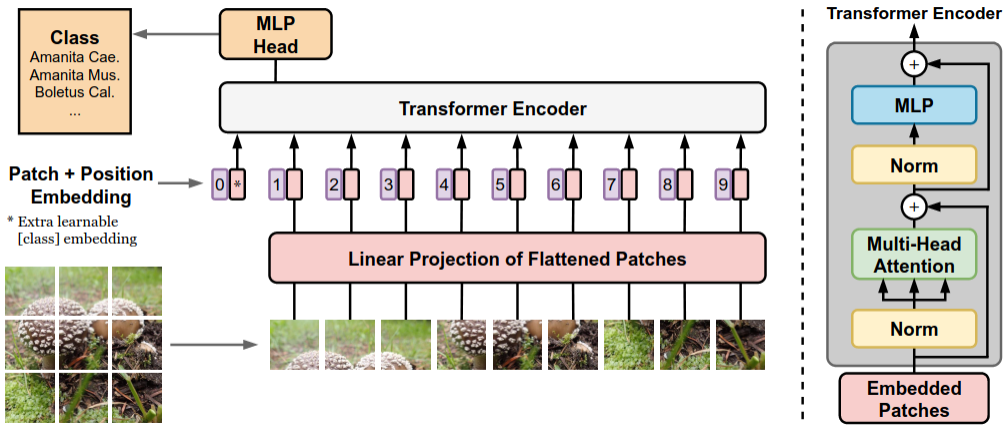

Figure 38: Basic example of image splitting in a grid for automatic fungi recognition. [44]

These patches are then flattened and passed through a linear projection layer to obtain a sequence of embeddings, which are then fed into a series of transformer encoder layers. The transformer encoder layers use self-attention mechanisms to model the relationships

between the different patches in the input sequence. This allows the ViT to capture long-range dependencies between different parts of the image, which is particularly useful for image classification tasks where the spatial relationship between different parts of the image is important. It takes as input an image and outputs a class label or a set of class probabilities, based on the features extracted from the image.

2.16.1 The Grid

The "16x16 words" phrase refers to the fact that the input image is divided into a grid of 16x16 patches, each of which is treated as a "word" in the input sequence. This means that the ViT processes the image as a sequence of 256 patches, each represented by an embedding, instead of as a single 2D array of pixel values. However, there is a trade-off between grid size and computational complexity, as larger grids require more memory and processing power to train.

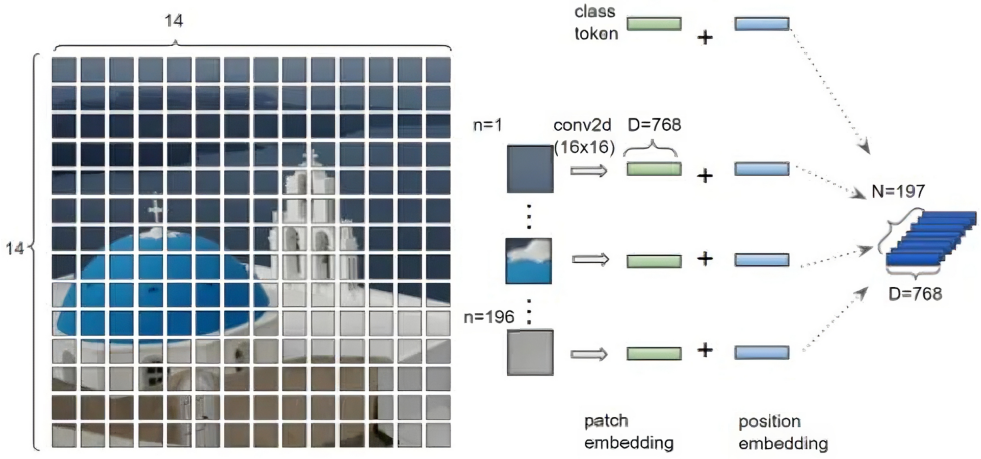

Figure 39: Vision transforrmer example of 14X14 [21]

3 Methodology

3.1 Effective Techniques for Improving Model Generalization

In the field of machine learning, it is important to ensure that the models we build are able to accurately and effectively make predictions on new data. However, it is common for models to suffer from issues such as overfitting or poor generalization to new data. In this section, we will explore three techniques that can be used to improve the performance of machine learning models: **regularization**, **normalization**, and **standardization**. By properly applying these techniques, we can mitigate the risks of overfitting and improve the ability of our models to generalize to new data. [10, 3, 53]

3.1.1 Regularization

Reegularization works by adding a penalty term to the objective function that the model is trying to minimize. This penalty term discourages the model from learning relationships that are too complex, and encourages it to learn simpler relationships that generalize better. There are several methods for regularization, including [17]:

1. L1 Lasso regularization: This method adds a penalty term to the cost function that is proportional to the absolute value of the weights.
2. L2 Ridge regularization: This method adds a penalty term to the cost function that is proportional to the square of the weights.
3. Dropout regularization: This method randomly sets a fraction of the weights in the model to zero during training, which helps to prevent overfitting by reducing the number of parameters in the model. Dropout is only applied during the training process, and all neurons are available during evaluation.

In two-dimensional space, the L1 regularization term takes the shape of a diamond centered at the origin because it represents the sum of absolute values of the weights.The L2 regularization term takes the shape of a circle centered at the origin because it represents the sum of the squared values of the weights.

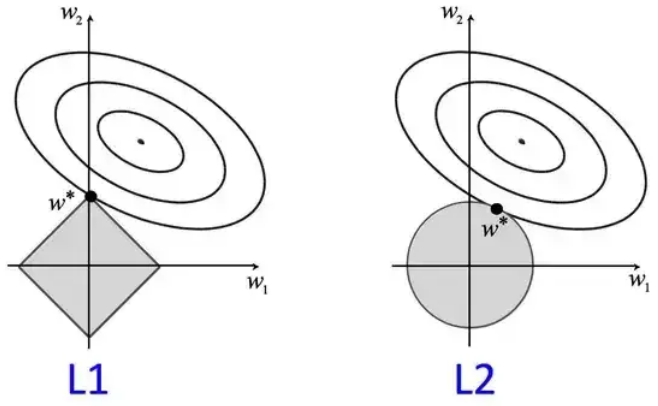

Figure 40: Level sets of the loss function and L1,L2 regularization [37]

To implement regularization, you can modify the cost function of the model to include the regularization term. For example, in L2 regularization, the cost function would be modified to include the sum of the squares of the weights, as shown in the following equation:

$$\boldsymbol{J}(\boldsymbol{W}) = \frac{\lambda}{2}||\boldsymbol{w}||^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2 \tag{76}$$

- $\boldsymbol{\lambda}$ regularization parameter
- $\boldsymbol{J}(\boldsymbol{W})$ Weight regularization

3.1.2 Normalization

Data is scaled using a process called normalization to give it a unit norm (or length). This is frequently done to improve the data's suitability for particular machine learning algorithms, such as those that use gradient descent or have a set range for acceptable input data. When comparing various features, normalization can also be used to scale down the data to a common scale. Data normalization methods include min-max normalization, mean normalization, and z-score normalization.

Normalization of complex numbers involves dividing a complex number by its magnitude (or absolute value) to obtain a complex number with a magnitude of 1. This is typically done to simplify calculations and make it easier to compare complex numbers. The normalized form of a complex number is often written as $\hat{z} = \frac{z}{|v|}$, where z is the original complex number and $\hat{z}$ is the normalized form and $|v|$ is the max magnitud of the entire vector of the complex numbers. Normalization of complex numbers is useful in many applications, including signal processing and control systems, where it is often necessary to compare complex numbers on an equal footing. [9]

$$\hat{z} = \frac{z}{|v|} \tag{77}$$

- $|v|$ Maximum magnitud of the vector
- z Input value
- $\hat{z}$ Normalized value

3.1.3 Standarization

Standardization is a method used in machine learning to transform the values of a feature or set of features to a standard scale. The standard scale is typically defined as having a mean of 0 and a standard deviation of 1. Standardization is often used as a preprocessing step before training a model, as it can help to improve the performance and convergence of the model. Standardization can be useful when the features in the dataset have different scales or units, as it can help to bring them onto a common scale and make it easier for the model to learn from the data. Standardization can be applied to both real and complex-valued data.

$$\hat{z} = \frac{z - \mu}{\sigma} \tag{78}$$

- μ Mean of the data
- σ Standard deviation of the data
- z Input value
- $\hat{z}$ Standarized value

3.2 Z-score

The Z-score, also known as standard score, is a statistical measure that indicates how many standard deviations an observation or data point is from the mean of a data set. It is used to standardize data and compare individual observations to a population or sample mean. The magnitude of the z-score represents how far away the data point is from the mean in terms of standard deviations. It helps us to remove outliers and make our trainning stage more stable. Outliers are data points that are significantly different from the other data points in the data set and can skew statistical analyses or machine learning models.

To identify outliers using z-score, we first calculate the z-score for each data point in the data set. We can then set a threshold z-score value, usually between 2 and 3, beyond which any data point is considered an outlier. Data points that have a z-score above the

threshold are identified as outliers and can be removed from the data set. This can help to improve the accuracy and reliability of our results or predictions.

$$z = \frac{x_i - \mu}{\sigma} \tag{79}$$

We use the z-score to filter outliers with a desired level of confidence, which is typically set at 95%. This corresponds to a z-score of 1.96, which means that approximately 95% of the data points would fall within the confidence interval. The 95% confidence level is commonly used in statistical analyses as a standard level of confidence.

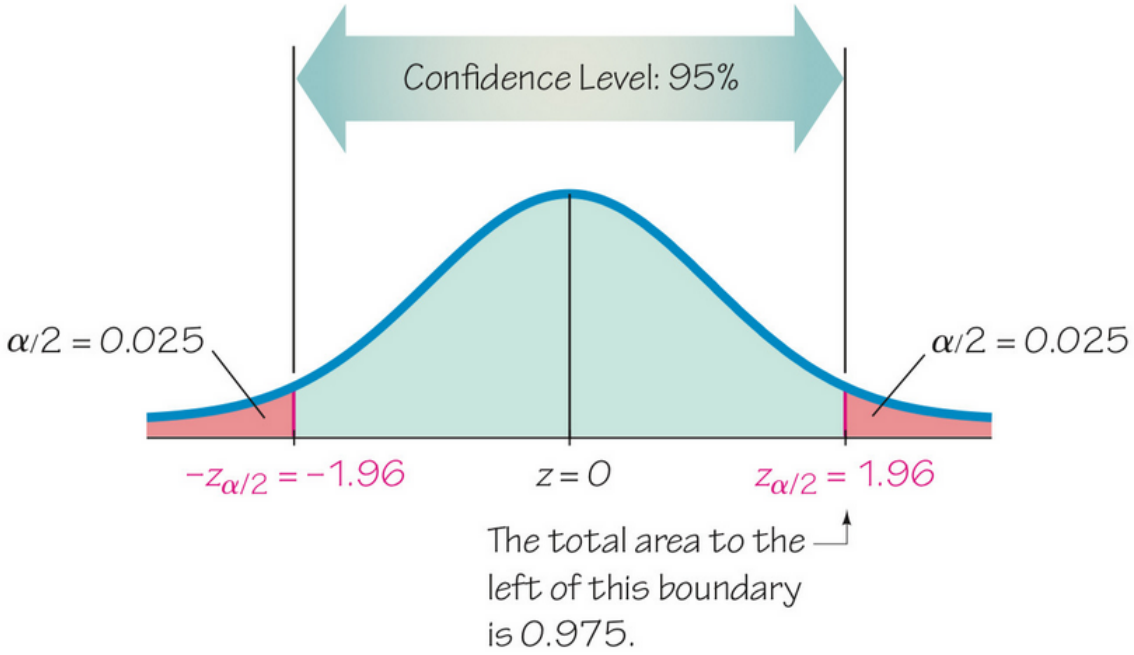

Figure 41: 95% Confidence interval [28]

In this project, we used the z-score to assess the dispersion of each block of 48 symbols. The aim was to ensure that the symbols in each block were within the required confidence interval and that the relations between the symbols were not too dispersed. By using the z-score, we were able to analyze the standard deviation of the symbols in each block and identify any blocks that fell outside the desired range. It is worth noting that the received symbols in this project were subject to distortions and interference resulting from the communication channel and environmental noise. This made it imperative to carefully assess the dispersion of each block of symbols in order to identify and correct any errors in the transmission.

3.3 FLOPS

FLOPS stands for "floating point operations per second." It is a measure of the computational performance of a computer or processor, and it indicates how many floating point arithmetic operations can be performed in one second. Floating point operations include addition, subtraction, multiplication, and division, as well as more complex operations like trigonometric functions, logarithms, and exponentials.

FLOPS is commonly used as a benchmark for evaluating the performance of CPUs, GPUs, and other computing devices. It is often used in the context of high-performance computing, scientific simulations, and machine learning applications, where large amounts of data must be processed quickly.

One approach to measuring FLOPS that we used in this work is to calculate the execution time and multiply it by the number of operations in each equalizer. This allows us to visualize the time complexity and efficiency of the equalizers.

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{1}}{\boldsymbol{ExecTime}} * \boldsymbol{Operations} \tag{80}$$

Performance variations in parallel algorithms stem from differences in design, implementation, and parallelism efficiency. Factors like memory access patterns, communication overhead, and load balancing also influence performance on multi-core systems. As a result, floating-point operation rates can change depending on algorithm effectiveness and work distribution among cores.

3.4 Dataset

This project has a dataset of 20,000 channel realizations, each with a size of 48x48. The dataset is divided into two groups of 10,000: the first group consists of Line-of-Sight (LOS) channel realizations, and the second group consists of Non-Line-of-Sight (NLOS) channel realizations. Each value in the matrix is a complex number with a format of complex128, float64 for the real and imaginary parts. The data is stored in the .mat format, which is commonly used for storing variables on disk from Matlab code. However, the dataset is used in Python using the Scipy library. To ensure robust predictions, the LOS and NLOS channels have been shuffled, with one group following the other. As neural networks typically do not work well with complex numbers, the real and imaginary parts of the channels have been separated into two channels in an image.

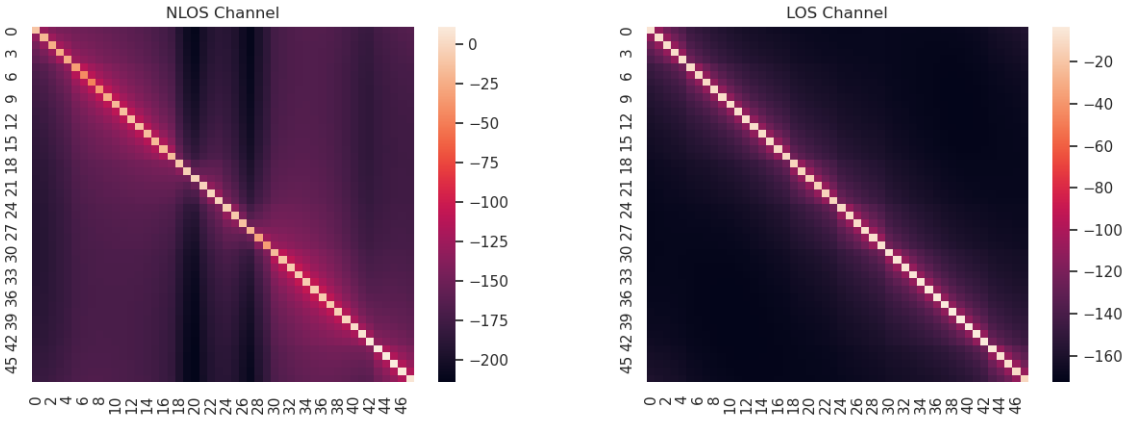

Figure 42: A comparison of LOS and NLOS channel magnitud in a log scale representation.

3.5 Software Architecture

3.5.1 Data set

In the field of machine learning, data is typically split into three sets: training, validation, and testing.

1. **Training set**: The training set is a subset of the data used to train the model. The model uses the training data to learn the relationships between the input features and the target output. The model parameters are updated during the training process to minimize the prediction error.

2. **Validation set**: The validation set is a subset of the data used to evaluate the model during training. The purpose of the validation set is to ensure that the model is not overfitting to the training data. Overfitting occurs when the model is too complex and learns the noise in the training data instead of the underlying relationships. The model is evaluated on the validation set after each training epoch, and its performance is used to determine when to stop training or to adjust the model's hyperparameters.

3. **Testing set**: The testing set is a subset of the data used to evaluate the model's performance after training. The model is never trained on the test set and its performance on the test set provides an estimate of its generalization performance to new, unseen data. The test set is used to determine the final accuracy of the model and its ability to make predictions on new data.

3.5.2 Class design and documentation

To achieve code reusability, it is important to write modular and maintainable code that incorporates class inheritance and shared attributes. In order to further enhance the object-oriented programming (OOP) structure of our code, we implement the factory pattern. The factory pattern allows us to create multiple experiments with little changes but have the same base build and experiment structure, increasing code efficiency and reusability [56].

This, combined with the PyTorch Lightning framework, has made the development process much easier and faster. PyTorch Lightning also offers tools for distributed training that can be used to scale the training of big models across several GPUs[46], TPUs [62], or machines. This tooling also facilitates the concept of batches, which involves having multiple realizations of the dataset in the format [BATCH, seq_len] or [BATCH, channel, height, width]. By using batches, the training time is reduced and it becomes more manageable to handle large amounts of data.

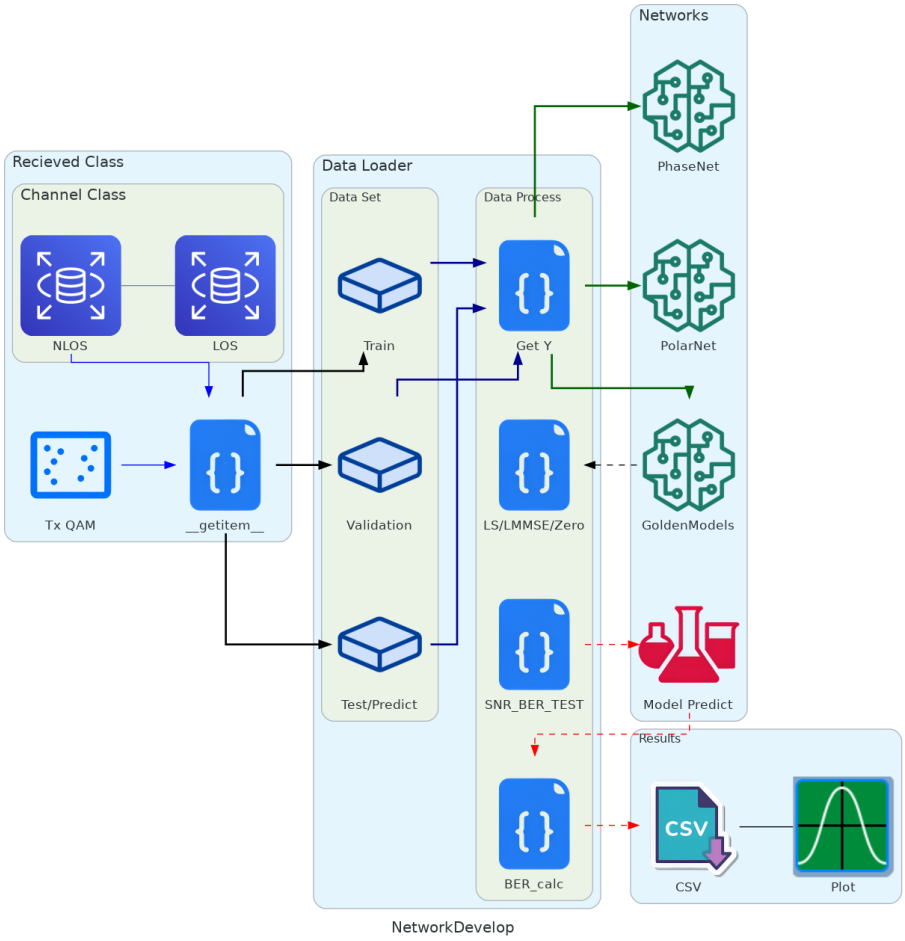

Figure 43: Software Architecture Diagram Planning

These are the main classes used in the software development:

- **Channel**
  - Loads a .mat file containing complex channel coefficient data and converts it into a numpy array. The class has a constructor __init__ that takes a Boolean parameter LOS (default value True) indicating whether to load the line-of-sight (LOS) or non-line-of-sight (NLOS) channel data.

- The Channel class has a single attribute con_list that is the numpy array containing the channel data. The class also defines a __getitem__ method that returns the channel data for a specific index in the array.
- The code also imports the scipy.io and numpy libraries and sets up the path to the directory containing the .mat files.
- Imports the Download_Mat_files function from the DownloadFiles module, and adds the path to the conf and tools directories to the Python path.

- **QAM**

  - This class QAM is used to generate QAM modulation schemes and perform demodulation on a complex symbol vector.
  - The constructor initializes the QAM modulation scheme. It takes in the following parameters:

    * num_symbols: The number of QAM symbols to generate.
    * constelation: The size of the QAM constellation. Valid values are 4, 16, 32, and 64.
    * cont_type: The type of constellation used. Valid values are "Data", "Unit_Pow", and "Norm".

      · **Data**: constelation as it is.
      · **Unit_Pow**: normalized power constelation
      · **Norm**: Normalized constelation to max magnitud to 1 in the complex plane.

    * noise: Deprecated
    * noise_power: Deprecated
    * load_type: A flag indicating the type of loading. Valid values are "Complete" and "Alphabet".

      · **Complete**: This is used to get the complex plane data points
      · **Alphabet**: Get the token values, which could be used to retrieve the symbols sent without having to put them in the complex domain.

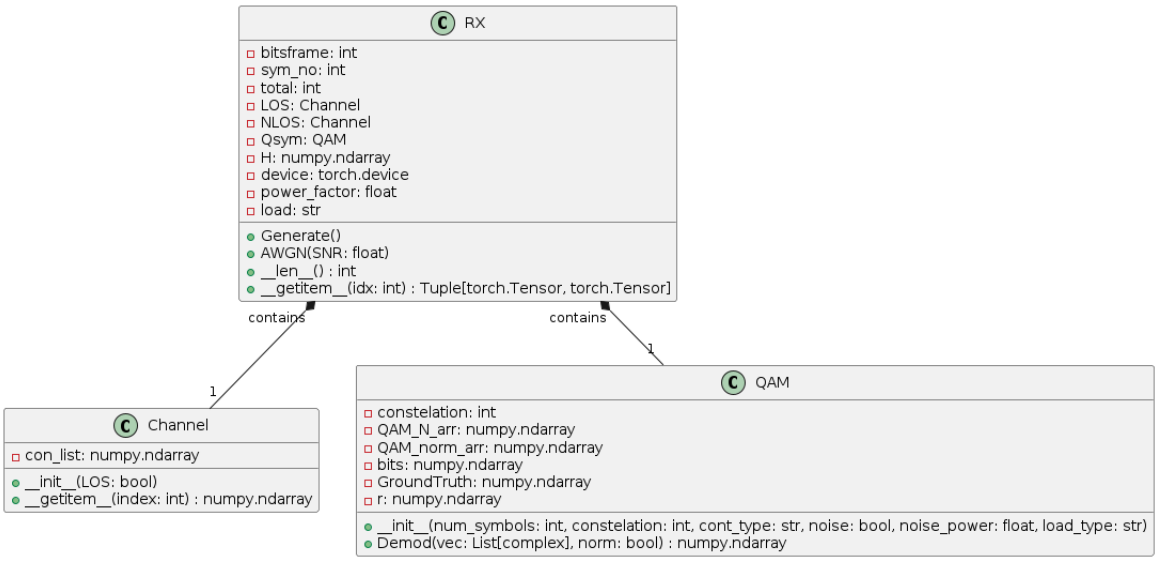

Figure 44: Rx and their aggregated classes

- **Rx:** This class is a PyTorch Dataset that generates a dataset for a communication channel. The channel is defined by a complex matrix H, and QAM symbols are generated with specific parameters such as constellation size, constellation type, and load type. The dataset consists of a total of 20000 realizations, with each realization containing 48 QAM symbols. The class generates QAM symbols, generates a complex matrix H, applies the channel to the symbols, and adds additive white Gaussian noise to the signal. The class also defines the AWGN method to add noise with a given SNR. The dataset can be loaded as batches using PyTorch's DataLoader. The class overloads the __getitem__ method to return the channel tensor and the corresponding transmitted tensor. The channel tensor is a 48x48x2 tensor of the real and imaginary parts of the complex matrix H, and the transmitted tensor is either the QAM symbol bits or the QAM symbol complex values depending on the load type.

- **RX_loader**: This class also splits the dataset into three parts for training, validation, and testing. The SNR_BER_TEST method of the class is used to calculate the bit error rate (BER) for different signal-to-noise ratio (SNR) values. This method uses the predict method of the trainer object to predict the values of the output of the model and then calculate the BER. Class also defines methods to calculate the output of the system given the input, including Get_Y, MSE_X, LMSE_X, and ZERO_X. These methods calculate the channel output, estimate the transmitted symbols, and perform equalization to reduce the effect of the channel on the received symbols. The class also defines a method to filter the data based on z-score.

- **init**: Initializes the Rx_loader object with the specified batch size, QAM and loading type. It also loads the RX dataset and splits it into training, validation and testing sets.
- **SNR_calc**: Calculates the signal-to-noise ratio (SNR) and bit error rate (BER) given the predicted output and the actual output. This is used for internal evaluation in the predict section of the Lightning API.
- **SNR_BER_TEST:** Performs an SNR-BER test by iterating through a range of SNR values and printing the BER for each value. It also saves the BER values to a CSV file.
- **Get_Y:** Takes in three arguments: H (channel tensor), x (transmitted symbol tensor), conj (boolean flag to indicate whether to take the complex conjugate of h), and noise_activ (boolean flag to indicate whether to add noise to the received signal). It returns the received signal tensor Y of shape (batch_size, 48).

  * **If noise_activ is True**, it adds complex Gaussian noise to the received signal Y[i]. The noise power is computed as the ratio of the signal power to the signal-to-noise ratio (SNR) in decibels. The noise is generated using the PyTorch function torch.randn to create random Gaussian noise with zero mean and standard deviation of sqrt(Pn/2), where Pn is the noise power. More detail in 2.5.1
  * **If conj is True**, it takes the complex conjugate of h using the method conj().resolve_conj(), and performs matrix multiplication with Y[i]. This helps in data preprocesing in section 2.8.2 more detail in Cascade Net [26] image.

- **MSE_X**: Computes the minimum mean square error (MMSE) estimate of the transmitted signal x given the channel matrix H and received signal Y. It returns a tensor of shape (batch_size, 48) of complex values.
- **LMSE_X**: Computes the linear minimum mean square error (LMMSE) estimate of the transmitted signal x given the channel matrix H, received signal Y and SNR. It returns a tensor of shape (batch_size, 48) of complex values.
- **Chann_diag**: Extracts the diagonal of each channel matrix in the tensor and returns them as a tensor of shape (batch_size, 48) of complex values.
- **ZERO_X**: The method starts by computing the diagonal elements of the channel tensor using the **Chann_diag** method. This helper method extracts the diagonal of each matrix in the tensor and combines them into a new tensor. Next, the method applies Zero Forcing (ZF) equalization by dividing the received signal

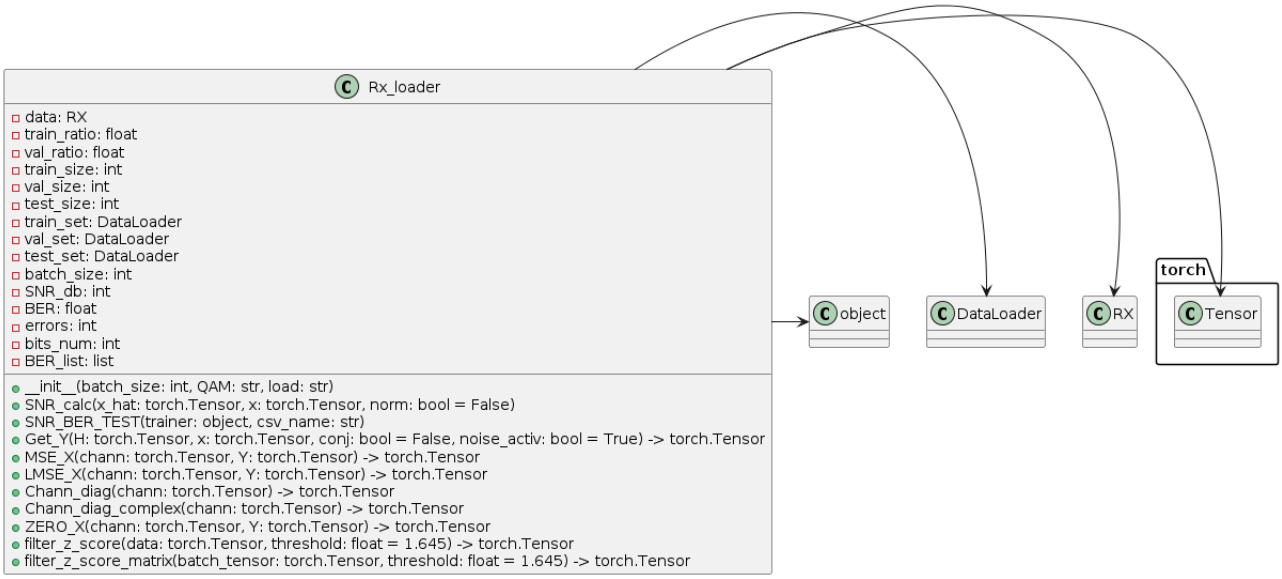

Figure 45: Rx Data Loader and usefull classes

tensor Y by the diagonal tensor of the channel tensor chann. The output is an estimated signal tensor x_hat, which is a tensor of complex values with a shape of (batch_size, 48). The method returns this tensor as its output.

- **filter_z_score**: Filters out outlier data points from the tensor based on the z-score. It returns a filtered tensor and the indices of the valid data points.
- **filter_z_score_matrix**: Filters out outlier data points from the diagonal tensor of a batch of channel matrices based on the z-score. It returns a filtered tensor and the indices of the valid data points.

    * The default value of the threshold parameter in the filter_z_score() method of the Rx_loader class is 1.96, which corresponds to a 95% confidence level assuming a normal distribution. This means that data points whose absolute z-score is greater than 1.96 will be considered as outliers and filtered out. However, this value can be adjusted by the user based on their specific requirements.

The Rx_loader class is implemented in all classes throughout the project for multiple experiments. The golden models also use this class because all final application classes inherit from this class and use the PyTorch Lightning framework to create powerful combinations between custom datasets and helper functions required for preprocessing.

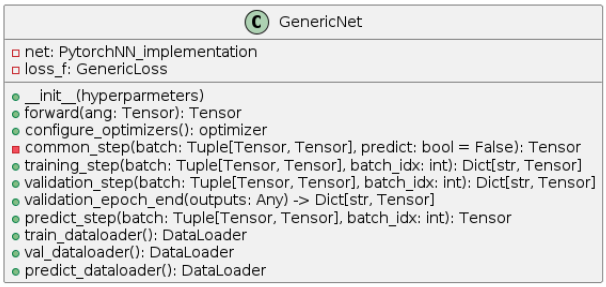

Figure 46: Generic Net

3.5.3 Generic Network Implementation

The training_step and validation_step methods both call the common_step method, which contains the shared implementation between the two. This approach reduces code duplication and helps keep the implementation DRY (Don't Repeat Yourself)[36].

The common_step method takes a batch of data and an optional flag to indicate whether the batch is being used for prediction. It then performs the **pre-processing** of the data, which includes different strategies such as Zero forcing, obtaining Y with its conjugate (19), normalizing values (2.12.1), and filtering outliers by z-score (3.2).

After the pre-processing is done, the method evaluates the model by computing the predicted output and comparing it with the target output. The **evaluation** process involves passing the pre-processed data through the neural network model, which generates the predicted output. The predicted output is then compared with the target output using a loss function, which calculates the difference between them. The method returns the loss value, which is used for updating the model's parameters during training.

By separating the common implementation from the specific training and validation logic, we can easily reuse the same code for different stages of the training process, and focus on implementing the logic that is specific to each stage. This results in cleaner and more maintainable code, as well as faster development times.

3.6 Golden Model

In this project, the golden model serves as a benchmark for evaluating the performance of new models. The golden model is based on three equalizers: Least squares (LS), Linear Minimum Mean Squared Error (LMMSE), and zero forcing, which are discussed in more detail in sections 2.3.3, 2.3.5 , 2.3.2 of this document,respectively, and for the non-linear models there was implemented OSIC and NearML.

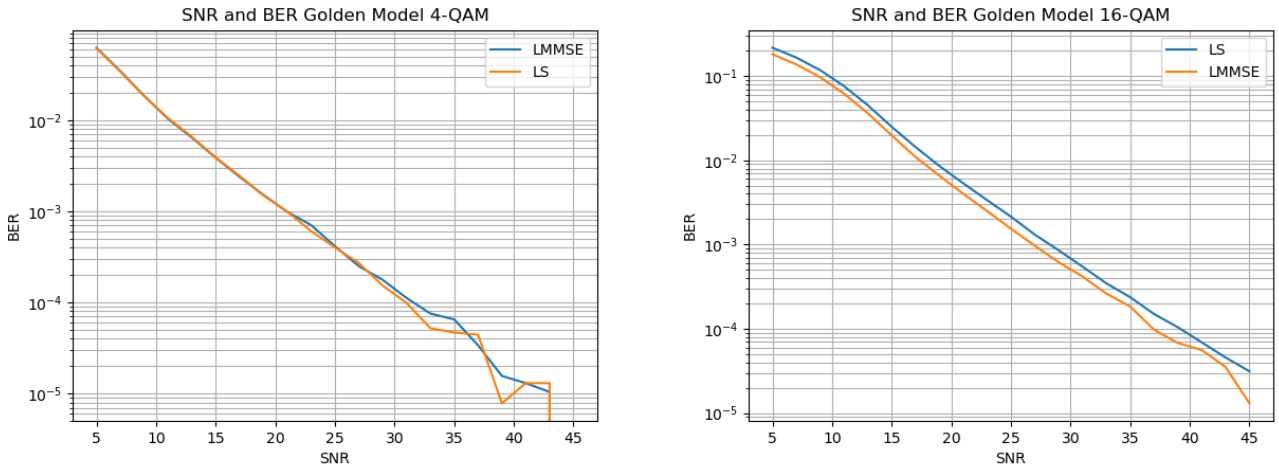

Figure 47: Golden linear models plot QPSK and 16QAM

When evaluating the performance of a communication system, it is standard practice to plot both the Bit Error Rate (BER) and Signal-to-Noise Ratio (SNR) on the same graph, with SNR on the x-axis and BER on the y-axis. BER measures the number of errors that occur in a transmitted data stream, relative to the total number of bits transmitted, while SNR measures the strength of the signal relative to the background noise. Comparing the BER and SNR values allows us to evaluate the system's performance under different levels of noise.To create a more representative view, it was decided to average the results of 5 tests and plot the average. Additionally, for each test, the SNR was incremented by a step of 2. Therefore, an interpolation was performed to smooth the curves.

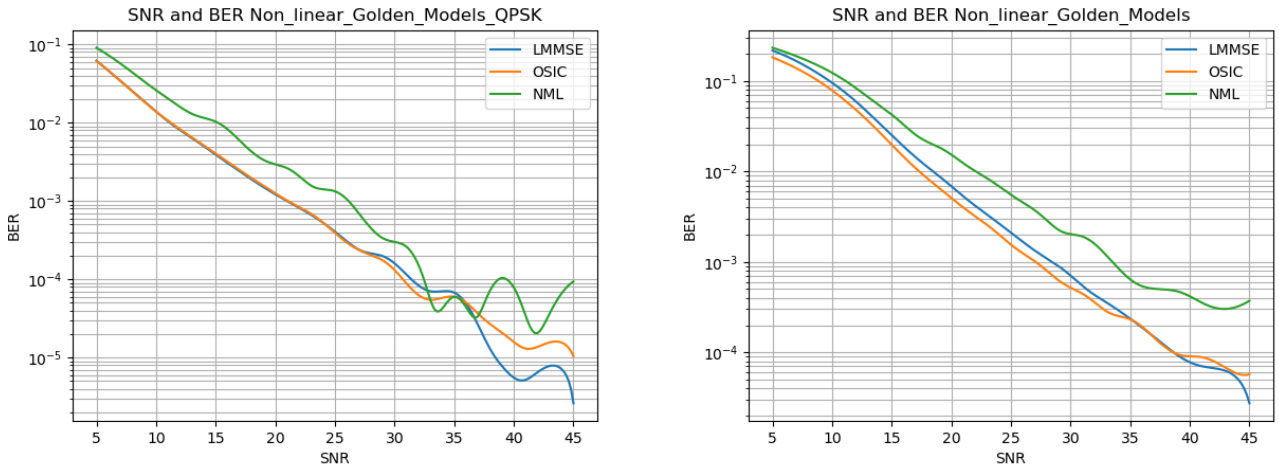

Figure 48: Golden LMMSE and non-linear models plot QPSK and 16QAM

Another metric we are using is the block error rate (BLER) which is a common metric

used to measure the performance of digital communication systems, and it is an important factor in determining the quality of the transmission. A lower block error rate indicates that fewer errors are occurring during transmission and the communication system is performing better. A "block" is a fixed-length sequence of bits, and in a communication system, data is transmitted in these fixed-length blocks, for this case we built the blocks of 48 which is the same size as frame.

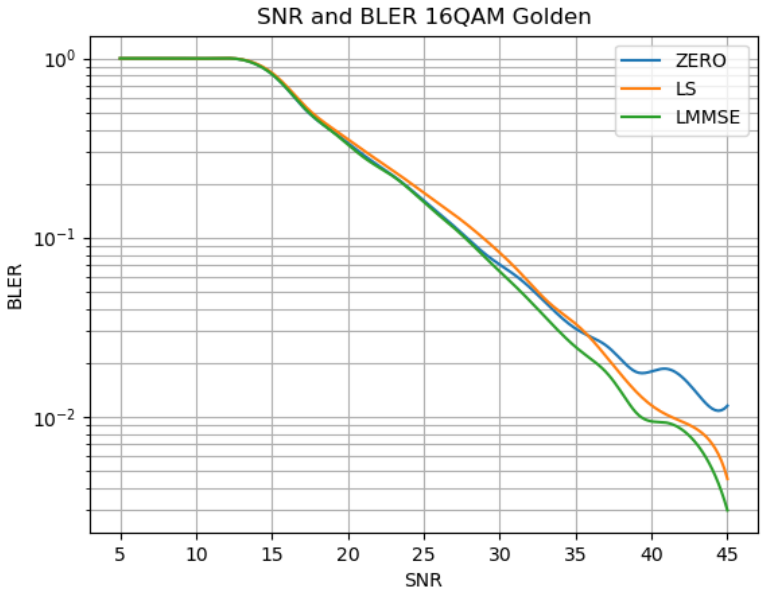

Figure 49: 16-QAM linear models BLER: LS, LMMSE, ZeroForcing

Defining the Golden class, which takes the mode ("MSE"(LS), "LMSE", or "ZERO") as input and initializes the Rx_loader superclass with the given BATCHSIZE, QAM, and "Complete" load. Depending on the mode, the class sets the self.estimator attribute to the MSE_X, LMSE_X, or ZERO_X method. The class also defines the forward method, the configure_optimizers method, and the predict_step method, which calculates the bit error rate (BER) for a given batch of data. Finally, the class defines the predict_dataloader method, which returns the test_loader. In the main section of the script, a PyTorch Lightning Trainer object is instantiated. Finally, the SNR_BER_TEST method of the Golden object is called with the Trainer object and a file name for the output log.

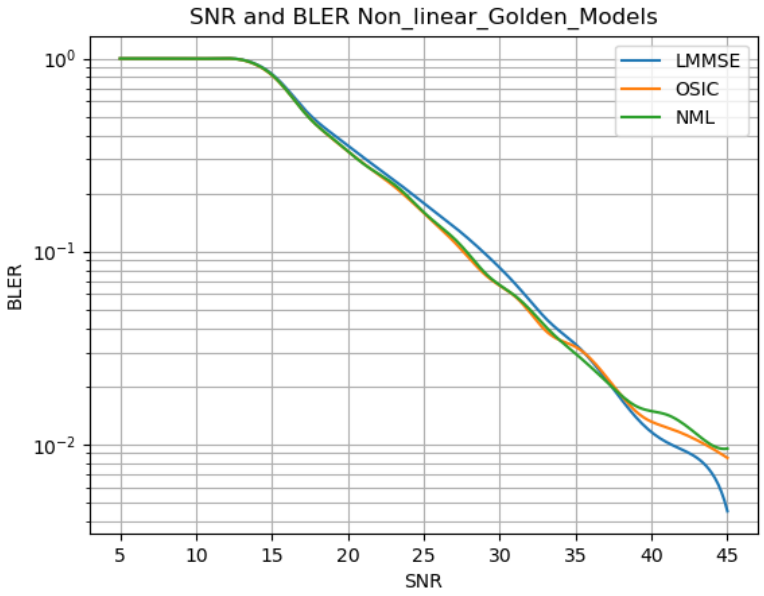

Figure 50: Golden LMMSE and non-linear models BLER 16QAM

3.7 PhaseNet

This neural network corrects the distortion and restores the original phase given channel and noise. It can be used in applications such as phase modulation or PSK. However, the network encounters a problem when evaluating the error between the estimated and real values. When the estimated value is in the second quadrant of the complex plane and the real value is in the third quadrant, the error becomes quite large. This is because the error is measured as the difference in angles, and traversing the second, fourth, and third quadrants causes the error to increase significantly. It is important to note that angle values are usually represented within the range of $-\pi$ to π in pytorch and numpy libraries.

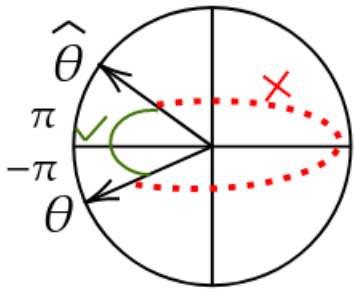

Figure 51: Angle error - Red represents high error and green represents smaller error.

To address the issue of computing the error and achieving precise adjustment of the

estimation, we adopted a new technique. This technique involves generating a complex number with a fixed radius of 1 and a variable angle. Since neural networks only output angles, we truncated the ground truth complex values to a radius of one. We used Euler's formula to construct this complex number in rectangular form, based only on the angle. Specifically, we used the output of the neural network, denoted as $\boldsymbol{O(\hat{\theta})}$, with a radius of one to build the complex number.

$$\boldsymbol{O(\hat{\theta}) = e^{i\hat{\theta}} = cos(\hat{\theta}) + i * sin(\hat{\theta})} \tag{81}$$

By using this approach, we can concentrate solely on the phasors (angles) and keep a constant radius. We can objectively measure the difference between the complex value of the estimated and real values using the method of least squares. This method allows us to calculate the difference between two quadrants more accurately, resulting in better estimation correction. However, to avoid getting a complex error and obtain only a numerical error value, we calculate the mean squared error (MSE) separately for the real and imaginary parts.Furthermore, by limiting the radius to 1, the loss is bounded, preventing it from exploding and introducing a large error. This can help with the convergence of the neural network. The equation below shows the error measurement that was developed.

$$\boldsymbol{MSE\left(\hat{\theta}\right) = \frac{1}{2}\left(E\left[(cos(\hat{\theta}) - x_{real})^2\right] + E\left[(sin(\hat{\theta}) - x_{imag})^2\right]\right)} \tag{82}$$

However this expresion can be rewriten and it can be expand each term as follows:

$$\boldsymbol{E\left[\cos^2(\hat{\theta})\right] - 2x_{\text{real}}E\left[\cos(\hat{\theta})\right] + x_{\text{real}}^2 + E\left[\sin^2(\hat{\theta})\right] - 2x_{\text{imag}}E\left[\sin(\hat{\theta})\right] + x_{\text{imag}}^2}$$

We can simplify this by noting that:

$$\boldsymbol{E\left[\cos^2(\hat{\theta})\right] + E\left[\sin^2(\hat{\theta})\right] = E\left[\cos^2(\hat{\theta}) + \sin^2(\hat{\theta})\right] = E\left[1\right] = 1}$$

and

$$\boldsymbol{-2x_{\text{real}}E\left[\cos(\hat{\theta})\right] - 2x_{\text{imag}}E\left[\sin(\hat{\theta})\right] = -2(x_{\text{real}}\cos(\hat{\theta}) + x_{\text{imag}}\sin(\hat{\theta}))}$$

Substituting these simplifications back into the previous equation gives:

$$\boldsymbol{\frac{1}{2}\left(1 - 2(x_{\text{real}}\cos(\hat{\theta}) + x_{\text{imag}}\sin(\hat{\theta})) + x_{\text{real}}^2 + x_{\text{imag}}^2\right)}$$

which can be further simplified to:

$$\boldsymbol{\frac{1}{2}\left((\cos(\hat{\theta}) - x_{\text{real}})^2 + (\sin(\hat{\theta}) - x_{\text{imag}})^2\right)} \tag{83}$$

The final result is the same as the squared Euclidean distance, multiplied by a factor of 1/2. In complex form, it is represented by the squared magnitude of the difference between the complex numbers, which is equivalent to equation (63) with a fixed radius of 1

$$\boldsymbol{MSE}\left(\hat{\boldsymbol{\theta}}\right) = |\boldsymbol{O}(\hat{\boldsymbol{\theta}}) - \boldsymbol{x}|^{\mathbf{2}} \tag{84}$$

Where $|.|$ is the magnitud of the complex number.

3.7.1 Preprocesing

Table 3: Preprocessing flags PhaseNet

Preprocessing	Enable
Conjugate	True
Z-score	True
Zero-Forcing	False

Based on the experimentation, it can be concluded that preprocessing plays a crucial role in enabling the neural network to converge successfully. As demonstrated in the paper *A Novel OFDM Equalizer for Large Doppler Shift Channel through Deep Learning* on page 48, preprocessing helped to achieve a successful outcome. In this network, our first preprocessing stage is to multiply Y by the Hermitian transpose of the channel, $H^H Y$. This results in $H^H Y$ being the input to the network. Next, we apply a z-score filter on page 76 with a 95% confidence interval to eliminate outliers. Finally, we normalize angles from -pi to pi to -1 to 1.

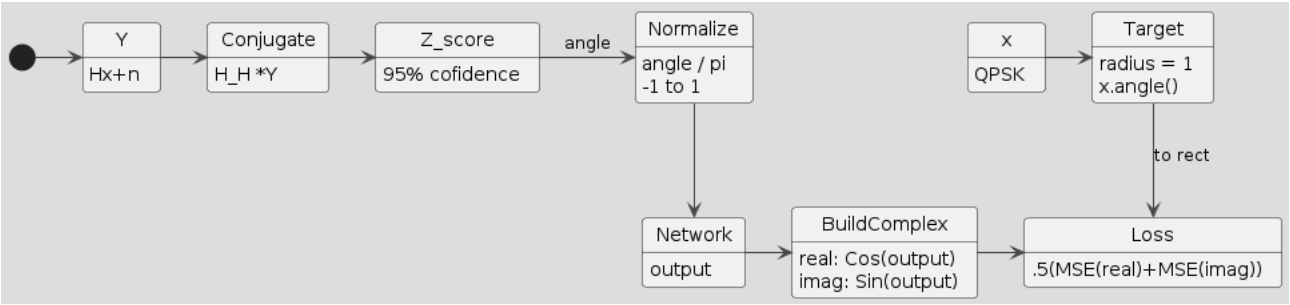

Figure 52: Preprocesing Stages

3.7.2 Parameters

Table 4: Number of Parameters in the PyTorch nn.Sequential block

Layer	Output Shape	Number of Parameters
`nn.Linear(48, 240, bias=True)`	`(batch_size, 240)`	`(48 + 1) * 240 = 11,760`
`nn.Hardtanh()`	`(batch_size, 240)`	`0`
`nn.Linear(240, 720, bias=True)`	`(batch_size, 720)`	`2 * 240^2 + 2 * 240 = 346,320`
`nn.Linear(720, 240, bias=True)`	`(batch_size, 240)`	`4 * 240^2 + 2 * 240 = 1,388,160`
`nn.Hardtanh()`	`(batch_size, 240)`	`0`
`nn.Linear(240, 48, bias=True)`	`(batch_size, 48)`	`(240 + 1) * 48 = 11,568`
	Total	**1,757,808 = 13.39 MB**

3.7.3 Hyperparameters

SNR stands for Signal-to-Noise Ratio during training. This value is applicable to all networks. During testing, the SNR is subjected to varying values.

Table 5: Hyperparameters

Hyperparameter	Value
BATCHSIZE	10
QAM	16
NUM_EPOCHS	2
INPUT_SIZE	48
HIDDEN_SIZE	240
LEARNING_RATE	8e-5
CONJ	True
SNRdB	35

3.8 PolarNet

The Polar_Net consists of two networks: PhaseNet, which we've seen before, and a new network called Mag_net. The Polar_Net is an integration of these two as pretrained networks. Since we're already familiar with PhaseNet, let's now focus on describing MagNet.

3.8.1 MagNet Preprocesing and loss function

The architecture of MagNet is quite similar to that of Polar_Net, but with fewer parameters in the hidden layer. As a result, we won't go into as much detail about the layer architecture. However, we'll still pay close attention to the preprocessing stage. Unlike PhaseNet, MagNet doesn't use the Hermitian transpose of the channel. Instead, it uses the diagonal of the channel to perform zero forcing equalization as a preprocessing step (2.3.2). Next, a z-score filter is applied, followed by complex normalization by absolute.

The loss function for MagNet remains quite simple, let $O(\hat{\theta})$ be the output of the network, which approximates the magnitude of the ground truth, and the magnitude of a complex number is denoted by $|.|$, the folowing error is given:

$$MSE(\hat{\theta}) = E\left[\left(O(\hat{\theta}) - |\theta|\right)^2\right] \tag{85}$$

To put it another way, the network produces its output in terms of the magnitude of the complex numbers. Then, for the loss function, the absolute value of the complex value of the ground truth is taken.

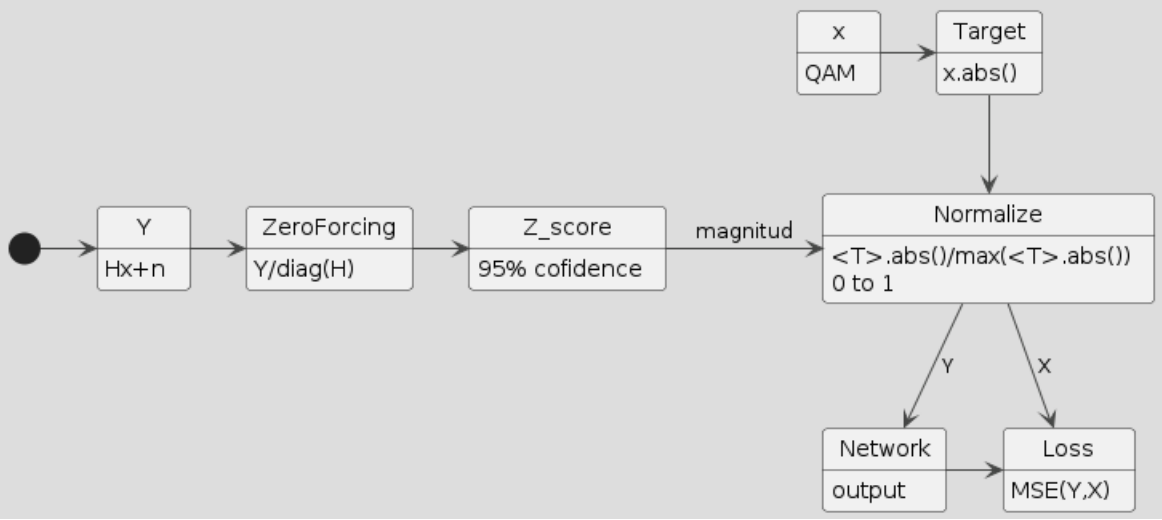

Figure 53: MagNet Preprocesing

3.8.2 Magnet Parameters

Table 6: Number of Parameters in the PyTorch nn.Sequential block

Layer	Output Shape	Number of Parameters
`nn.Linear(48, 120, bias=True)`	`(batch_size, 120)`	`(48 + 1) * 120 = 5,880`
`nn.Hardtanh()`	`(batch_size, 120)`	`0`
`nn.Linear(120, 240, bias=True)`	`(batch_size, 240)`	`2 * 120^2 + 2 * 120 = 29,040`
`nn.Linear(240, 120, bias=True)`	`(batch_size, 120)`	`4 * 120^2 + 2 * 120 = 57,840`
`nn.Hardtanh()`	`(batch_size, 120)`	`0`
`nn.Linear(120, 48, bias=True)`	`(batch_size, 48)`	`(120 + 1) * 48 = 5,856`
	Total	**98,716 = 789.7 KB**

3.8.3 MagNet Hyperparameters

Table 7: Hyperparameters

Hyperparameter	Value
BATCHSIZE	10
QAM	16
NUM_EPOCHS	10
INPUT_SIZE	48
HIDDEN_SIZE	120
LEARNING_RATE	5e-5
SNRdB	25

3.8.4 All Together

The final size of the network is the sum of the Magnet parameters and the Phase parameters, which is approximately **0.753 MB + 13.39 MB = 14.143 MB**. The reason why Magnet has fewer parameters is that more parameters in the magnitude can lead to overfitting and the network may not generalize well with new data.

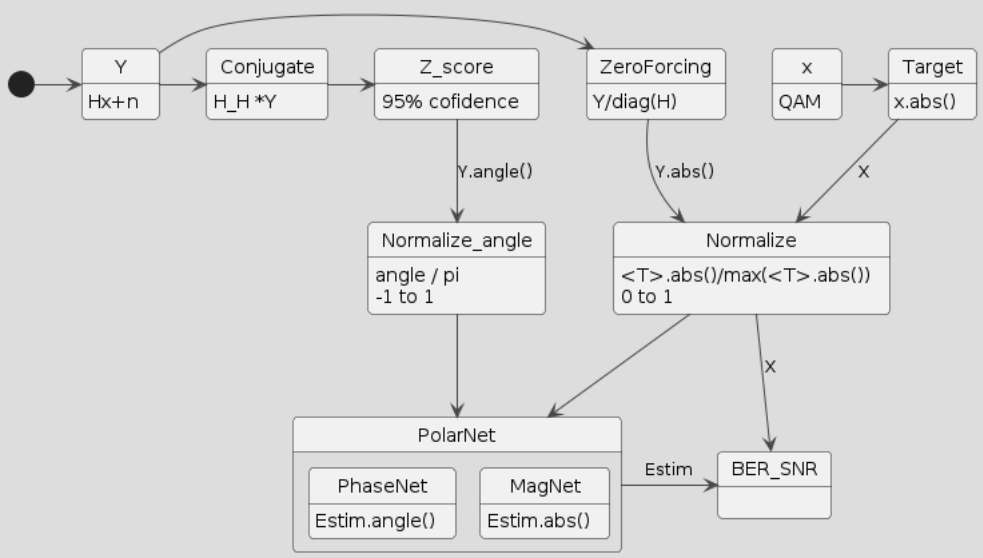

Figure 54: Overview of PolarNet

3.9 Complex Net

This network was motivated in the paper A Survey of Complex-Valued Neural Networks on page 54. The main idea is to investigate the use of complex-valued neural networks and attempt to reduce the number of network parameters. However, this approach may increase the time complexity due to the need for complex number operations between each layer. Additionally, we aim to utilize state-of-the-art loss functions proposed in mathematical formulas equation (63) and equation (65).Finally, for preprocessing, the hermitian transpose and normalization steps are applied.

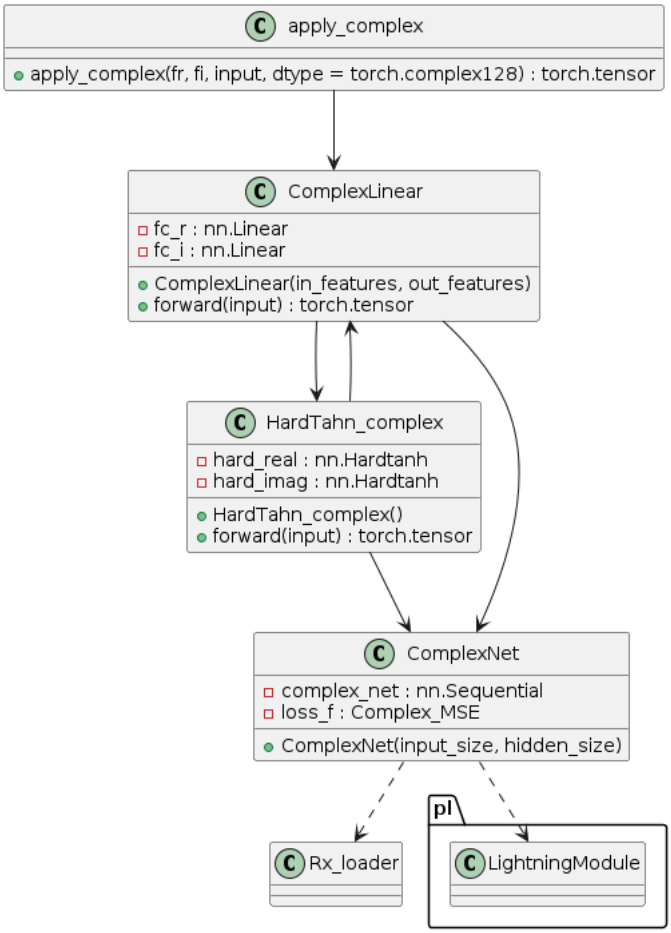

Figure 55: Complex Network Implementation

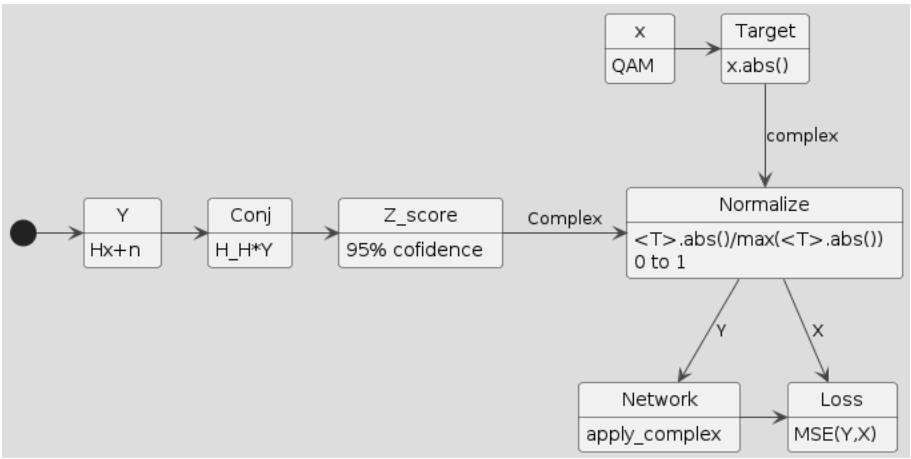

Figure 56: Complex Preprocesing

In the upcoming subsections, we will explain the components required to construct a complex network.

3.9.1 Apply Complex

One of the key components of layers in this stage is the apply_complex functions which is part of the complexLinear layers. The apply_complex(fr, fi, input, dtype = torch.complex128) function takes as input the real and imaginary parts of the weights fr and fi, the input tensor input, and a data type dtype for the output tensor. It applies a complex-valued linear transformation to the input tensor, given by the expression:

$$\boldsymbol{Re}(\boldsymbol{z}) = \boldsymbol{Re}(\boldsymbol{fr}) * \boldsymbol{Re}(\boldsymbol{x}) - \boldsymbol{Im}(\boldsymbol{fi}) * \boldsymbol{Im}(\boldsymbol{x}) \tag{86}$$

$$\boldsymbol{Im}(\boldsymbol{z}) = \boldsymbol{Re}(\boldsymbol{fi}) * \boldsymbol{Re}(\boldsymbol{x}) + \boldsymbol{Im}(\boldsymbol{fr}) * \boldsymbol{Im}(\boldsymbol{x}) \tag{87}$$

3.9.2 Complex Linear

The ComplexLinear(in_features, out_features) class defines a custom PyTorch module for a complex-valued linear layer. It has two attributes: fc_r is a standard PyTorch Linear module that applies a linear transformation to the real part of the input, and fc_i is another Linear module that applies a linear transformation to the imaginary part of the input. The forward(input) method of this module applies the complex-valued linear transformation separately to the real and imaginary parts of the input tensor using the apply_complex function.

3.9.3 Hardtanh Complex

The HardTahn_complex class defines a custom PyTorch module for a complex-valued hard tanh activation function. It has two attributes: hard_real is a standard PyTorch Hardtanh module that applies the hard tanh function to the real part of the input tensor, and hard_imag is another Hardtanh module that applies the hard tanh function to the imaginary part of the input tensor. The forward(input) method of this module applies the hard tanh function separately to the real and imaginary parts of the input tensor, and then combines them into a complex-valued tensor using torch.float64 and the +1j* syntax to create a complex number.

3.9.4 Parameters

Layer	Input Size	Output Size	Number of Parameters
Linear 1	48	120	11,520
HardTanh 1	120	120	0
Linear 2	120	240	138,240
Linear 3	240	240	115,440
Linear 4	240	120	57,720
HardTanh 2	120	120	0
Linear 5	120	240	138,240
Linear 6	240	48	23,088
Total	48	48	484,248 = 3.87 MB

Table 8: Number of parameters in the complex network.

3.9.5 HyperParameters

Parameter	Value
Batch Size	50
QAM	16
Number of Epochs	100
Learning Rate	0.00001
SNRdB	40

Table 9: Values for network hyperparameters.

3.9.6 Loss functions

In the experiments, we evaluate the performance of the complex network using both custom loss functions equation (63) and equation (65). The results of these experiments are presented in the next section.

```
def Complex_MSE(self,output,target):
    return torch.mean(torch.abs((target-output)))

def Polar_MSE(self,output,target):
    return .5*torch.mean((torch.square(torch.log(torch.abs(target)/torch.abs(output)))\
        +torch.square(torch.angle(target)-torch.angle(output))))
```

Figure 57: Loss functions code

3.10 MobileNet zeroForcing

We conducted experiments with this network to study how well Convolutional Neuronal Networks can be developed in signal equalization in the state of art Autoencoder [6]. However, we opted to employ the efficient architecture of MobileNet as our network's goal is to improve the effectiveness of the zero-forcing method for communication channels. Given

the relative simplicity of the zero-forcing algorithm, we determined that the MobileNetV3 [24] architecture was well-suited for our purposes. Its time efficiency and competence relative to zero-forcing made it a suitable choice. However, in the presence of noise in the channel data, the conventional zero-forcing approach may not be effective as it does not consider noise in its nature.

The channel is initially a 1x48x48 matrix, but it must be transformed into a single vector of length 1x48 for the zero-forcing approach to be applicable. Thus, this implementation employs two modified MobileNets to process the channel data. One network is responsible for extracting the absolute value of the channel matrix, while the other network extracts the angle matrix of the channel. Notably, the first and last layers were modified to fit the specific problem. In the image below, they are marked in red.

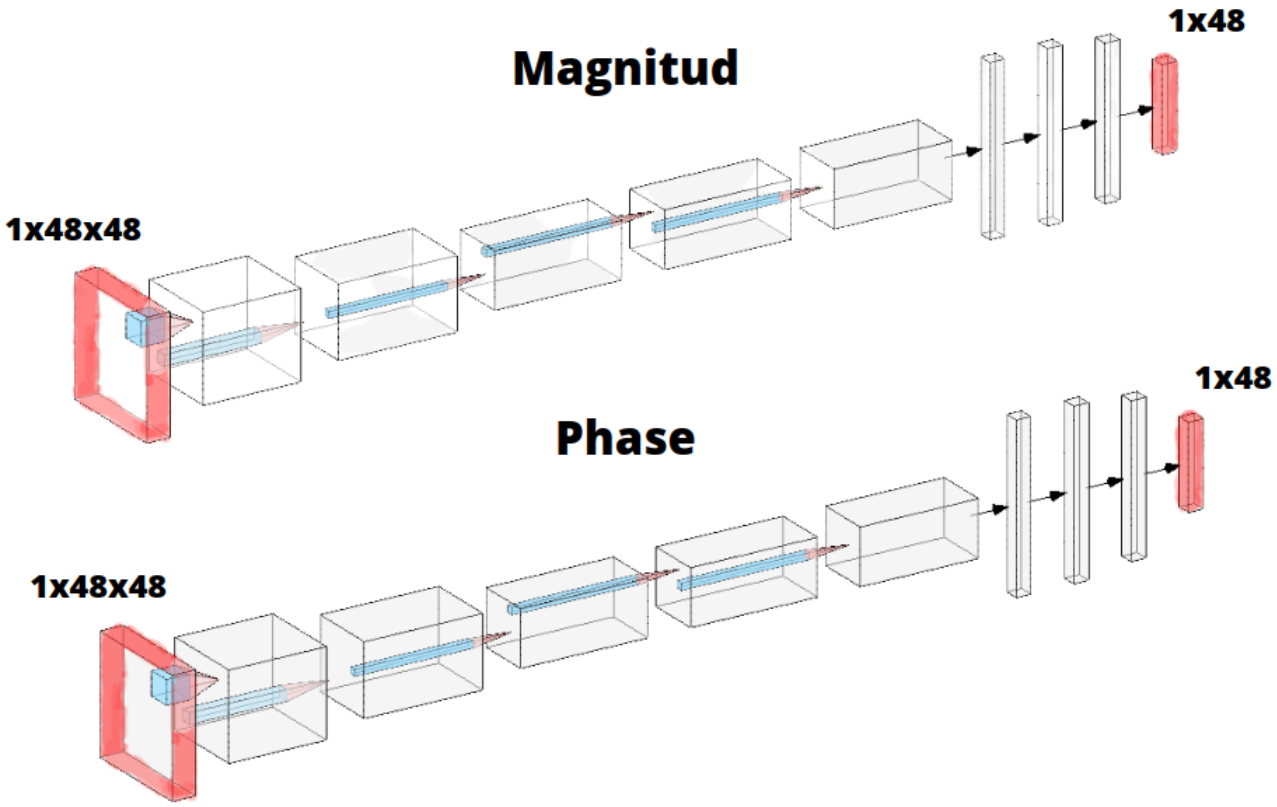

Figure 58: Ilustartion of mobileNet customized for our target problem

The models used in this project were taken from the PyTorch framework, so there was no need to test their functionality. We did not use pre-trained weights and instead trained the models from scratch. Thanks to the GPU, we were able to train the models faster.

The idea behind this is to replicate the process of traditional zero-forcing $\frac{\boldsymbol{Y}}{\boldsymbol{H}}$, where the channel matrix is represented by $\boldsymbol{H}$ and the received signal is represented by $\boldsymbol{Y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{n}$. In this process, we estimate the channel using the equation:

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{Y}}{\hat{\boldsymbol{\theta}}} \tag{88}$$

where $\hat{\boldsymbol{\theta}}$ represents the output of the model. However, as we want the estimator values to be in the range of $0 < \hat{\boldsymbol{\theta}} < 1$, we need to rearrange the estimation equation as follows:

$$\hat{\boldsymbol{x}} = \boldsymbol{Y} * \hat{\boldsymbol{\theta}} \tag{89}$$

As we are working with polar coordinates, we should note that when multiplying two phasors, their amplitudes are multiplied, and their angles are added together. So basically net compensate the distorted phasor Y and estimation equations has this form:

$$\boldsymbol{O}(\hat{\boldsymbol{\theta}}) = \boldsymbol{Y}_{\boldsymbol{mag}} * \hat{\boldsymbol{\theta}_{mag}} \angle \hat{\boldsymbol{\theta}}_{\boldsymbol{angle}} + \boldsymbol{Y}_{\boldsymbol{angle}} \tag{90}$$

Finally the MSE loss between this estimated $O(\hat{\theta})$ and the true value is given this form.

$$\boldsymbol{MSE}(\hat{\boldsymbol{\theta}}) = \boldsymbol{E}\left[\left(\boldsymbol{O}(\hat{\boldsymbol{\theta}}) - \boldsymbol{\theta}\right)^{\mathbf{2}}\right] \tag{91}$$

3.10.1 Software Implementation

In the common_step method, the input data is processed by the PredesignedModel network, which is a pre-trained MobileNetv3 network that is modified to fit the problem. The output of the PredesignedModel network is passed through two separate layers: one layer for the magnitude and one layer for the angle of the signal. The angle layer is used to estimate the phase shift of the received signal.
It's worth noting that the channel is treated as an image of two channels, one for the real and the other for the imaginary part. In the software implementation, a complex128 channel of 1 channel is built, and then the absolute and angle of this 1 channel matrix is taken to feed both mobile nets.
In this code implementation, the channel magnitude is normalized to the maximum magnitude of 1. This normalization is done after building the Y, to avoid altering the results of calculating the input signal. Then the z-score is computed, and outliers are filtered with a confidence interval of 99%. Finally, for the angle error calculation, the same ideas as in PolarNet are followed.

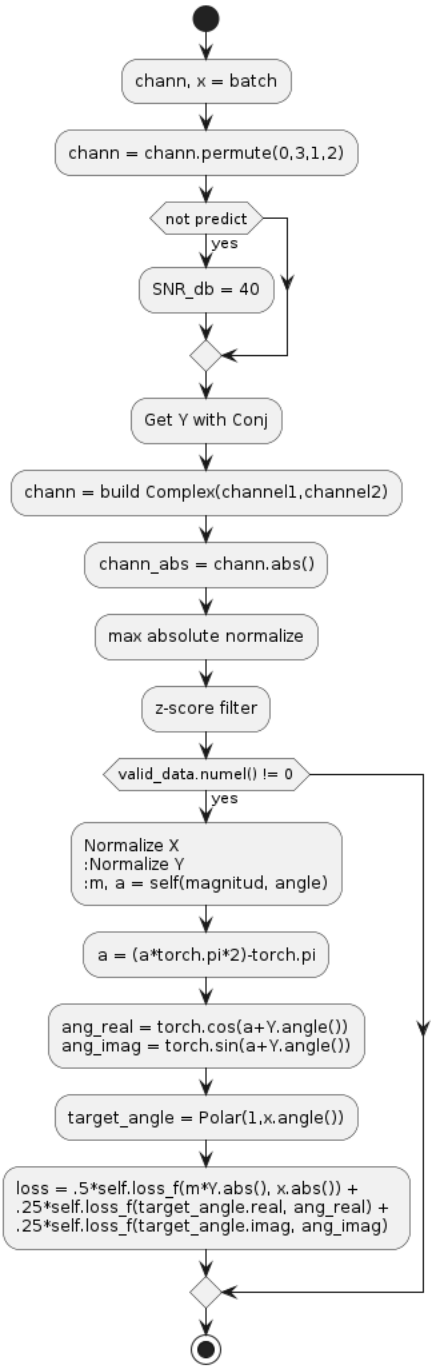

Figure 59: MobileNet Forward process

3.10.2 Loss Function

For the loss function we face the same issue as PhaseNet regarding the calculation of angle errors. To avoid this problem, we adopt a similar solution, which involves fixing the radius to 1 and comparing the angles (83). So we end up with 3 evalautons for a single loss function. However, in this loss, we chose to emphasize the Phasor concept to make

it more intuitive.

$$\boldsymbol{Loss} = \begin{array}{c} \frac{1}{2}\boldsymbol{MSE}(\boldsymbol{Y}_{mag} * \hat{\boldsymbol{\theta}_{mag}}, \boldsymbol{x}_{mag}) + \\ \frac{1}{4}\boldsymbol{MSE}\left(\left[\boldsymbol{1}\angle\hat{\boldsymbol{\theta}}_{angle} + \boldsymbol{Y}_{angle}\right]_{real}, [\boldsymbol{1}\angle\boldsymbol{x}_{angle}]_{real}\right) + \\ \frac{1}{4}\boldsymbol{MSE}\left(\left[\boldsymbol{1}\angle\hat{\boldsymbol{\theta}}_{angle} + \boldsymbol{Y}_{angle}\right]_{imag}, [\boldsymbol{1}\angle\boldsymbol{x}_{angle}]_{imag}\right) \end{array} \tag{92}$$

1. The first term computes the MSE between the magnitudes of the estimated signal $\boldsymbol{Y}_{mag} * \hat{\boldsymbol{\theta}_{mag}}$ and the true signal magnitud $\boldsymbol{x}_{mag}$. Here, $\boldsymbol{Y}_{mag}$ is the magnitude of the received signal $\boldsymbol{Y}$, and $\hat{\boldsymbol{\theta}_{mag}}$ is the estimated magnitude given by network.
2. The second term computes the mean squared error (MSE) between the real components of the estimated phasor, with fixed radius plus phase correction, and the real value of the angle of x
3. The same as second term in loss, but with imaginary part.

The function also includes weighting factors for each term. The first term is weighted by 1/2, while the second and third terms are each weighted by 1/4.

3.10.3 Parameters

MobileNetV3 model has about 2.24 million parameters in an efficient way.

Name	Type	Params
0	loss_f	MSELoss _ 0
1	abs_net	PredesignedModel = 1.6 M
2	angle_net	PredesignedModel = 1.6 M
3	final_merge_abs	Sequential = 18.6 K
4	final_merge_ang	Sequential = 18.6 K
Total size		**3.2 MB + 37.2 KB**

Table 10: Model architecture

3.10.4 HyperParameter

Parameter	Value
Batch Size	50
Number of Epochs	50
Learning Rate	0.0001
SNRdB	30

Table 11: Neural network hyperparameters

3.11 GridNet

At the moment we were conducting this research, nothing had been done before with transformers and signal equalization. We are aware that transformers are a heavier network, but recent advances in Edge Computing TPU processors [18] are making them a more feasible solution at a relatively low cost. While there is a paper called "Radio Transformer"[41] in the state of the art, it is used for modulation recognition, not for signal equalization. Additionally, historically, this paper does not align with "Attention is All You Need" [64] paper since it was written in 2016, whereas "Attention is All You Need" was written in 2017.

While we were initially inspired by the concept of an image being worth 16x16 words [13] as described in the state of the art section 2.16, we took this idea further by considering the encoder and decoder for the original transformer design figure 37 . Additionally, we challenged the assumption that the grid must be based on Euclidean geometry and instead rethought the grid in terms of a polar geometry. This paradigm shift was justified by the fact that our best results were obtained using polar representation. Furthermore, polar representation results in a non-uniformly distributed grid, where values closer to the center are penalized more heavily. This is useful because, in the presence of additive white noise, values are not expected to be centered, but rather more commonly found towards the edges of the polar representation.

The main idea is to divide the complex plane into grids, with each grid slot considered as a bucket. When a received value $\boldsymbol{Y_i}$ arrives, it is placed in an incorrect bucket $\boldsymbol{B_i}$,which is assigned a number $\boldsymbol{A_i}$. To train the network, we assign the ground truth value with its correct bucket number $\boldsymbol{B_j}$and calculate the maximum likelihood cost function between the estimation and ground truth , similar to correcting mistranslated text in a language task. Multihead attention improves Bucket calculation, because it takes care of the values passed before. In the context of signal processing, this attention mechanism helps to handle the ISI (Inter-Symbol Interference), which is a major challenge in dealing with a doubly dispersive Channel

.

Finally, after predicting the bucket $\boldsymbol{B_j}$,the value passes through the degrid process, which involves placing the value in the middle of the patch coordinates. This process helps to further equalize the value as it is forced to be placed in only this position or any other patch.

3.11.1 Square Grid

The encoding process involves binning the real and imaginary parts of the data into a set of discrete bins and mapping them onto a 2D grid. The decoding process reverses this process by mapping the values back to their original locations in the complex plane. The smallest number used for the grid is 4 because number 2 is reserved for the start of sentence token (<sos>) and 3 is reserved for the end of sentence token (<eos>), making it compatible with language models. The largest number corresponds to the number of bins used in the encoding process. In other words number of bins is the size of our alphabet.

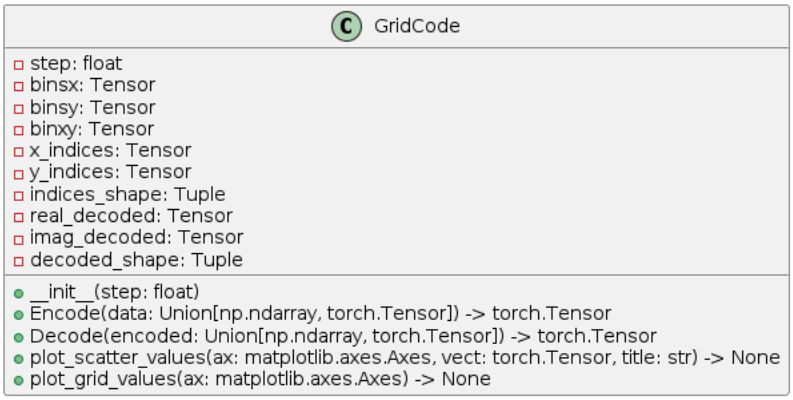

Figure 60: Class attibutes to do the grid with bins

- **step**: A float value that represents the step size for binning, from -.85 to .85. Note that step is for quadrant and not all plane.
- **binsx**: A tensor that contains the bins for the real part of the data.
- **binsy**: A tensor that contains the bins for the imaginary part of the data.
- **binxy**: A tensor that contains a 2D bin index matrix for encoding.
- **x_indices**: A tensor that stores the indices of the bins for the real part of the data.
- **y_indices**: A tensor that stores the indices of the bins for the imaginary part of the data.
- **indices_shape**: A tuple that stores the shape of the input data for encoding.
- **real_decoded**: A tensor that stores the decoded values for the real part of the data.
- **imag_decoded**: A tensor that stores the decoded values for the imaginary part of the data.

- **decoded_shape**: A tuple that stores the shape of the input data for decoding.

Encoding Recipe:

1. Loop over the binsx for the real and binsy for the imaginary parts, and check if the data value falls in the range of bin i and bin i+1.
2. Save all the values that fall in the bin as x_indices and y_indices. These indices behave more like coordinates for the matrix of binsxy.
3. Encoding is done by putting the indices in the binxy vector. The binxy vector returns a numerical token that represents the position of the complex plane.

Here's an elegant code implementation

```
        # Loop over the bins for the real and imaginary parts
        for i in range(len(self.binsx) - 1):
            # Find the indices of the data that lie within the current bin for the real part
            self.x_indices[(self.binsx[i] <= data.real) & (data.real < self.binsx[i + 1])] = i
            # Same for imaginary part
            self.y_indices[(self.binsy[i] <= data.imag) & (data.imag < self.binsy[i + 1])] = i

        # Encode the data by selecting the corresponding bin indices from the bin index matrix
        encoded = self.binxy[self.y_indices, self.x_indices]
        return encoded
```

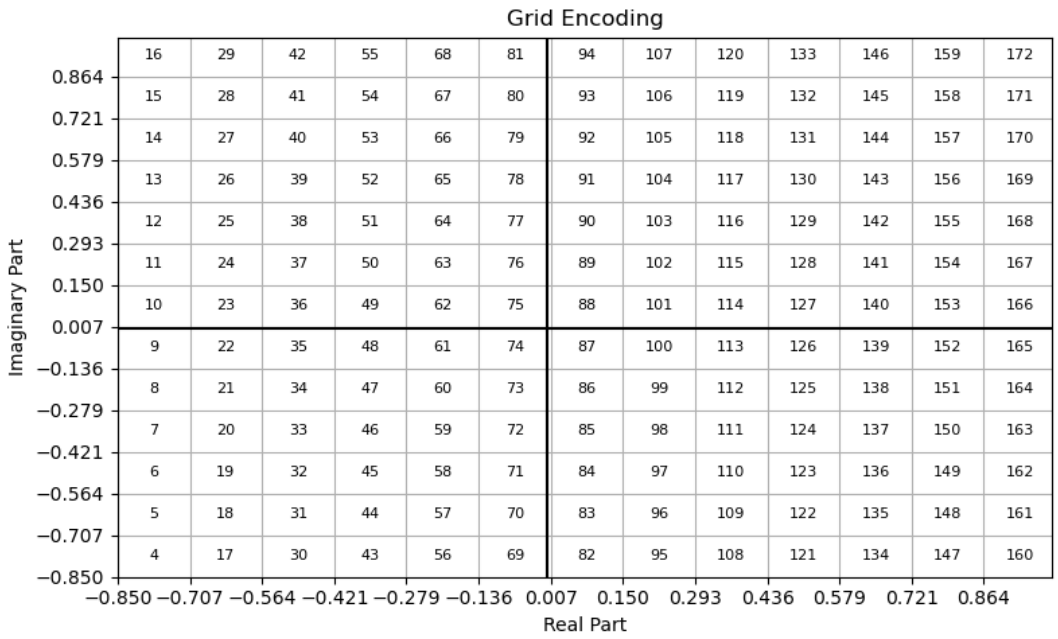

Figure 61: Square grid enconding values with a step of 1/7 per cuadrant

Decoding Recipe:

1. Loop over the binsxy matrix and create a mask of indices that correspond to the encoded values
2. Use the mask of indices to assign values from the binsx and binsy vectors to the corresponding real and imaginary decoded vectors. Add half a step to each value to place it in the center of the bin.
3. Build a complex number vector using the real and imaginary decoded vectors.

Here's an elegant code implementation

```
        for i in range(len(self.binsx) - 1):
            for j in range(len(self.binsy) - 1):
                # Find the indices of the encoded values that corresponsed
                # the current bin
                indices = (encoded == self.binxy[j, i])
                # Fill the decoded values for the real part
                self.real_decoded[indices] = self.binsx[i] + self.step/2
                # Fill the decoded values for the imaginary part
                self.imag_decoded[indices] = self.binsy[j] + self.step/2

        # Create a tensor for the decoded data
        data_decoded = torch.complex(self.real_decoded, self.imag_decoded)
        return data_decoded
```

The gridding step ensures that all possible values in the complex plane are now bounded by the alphabet size |A|, which can be beneficial for neural networks even if it doesn't compromise the space between QAM points spacing. In the picture bellow we show some points that was enconded and decoded, and we appreciate how they are centered after decoding.

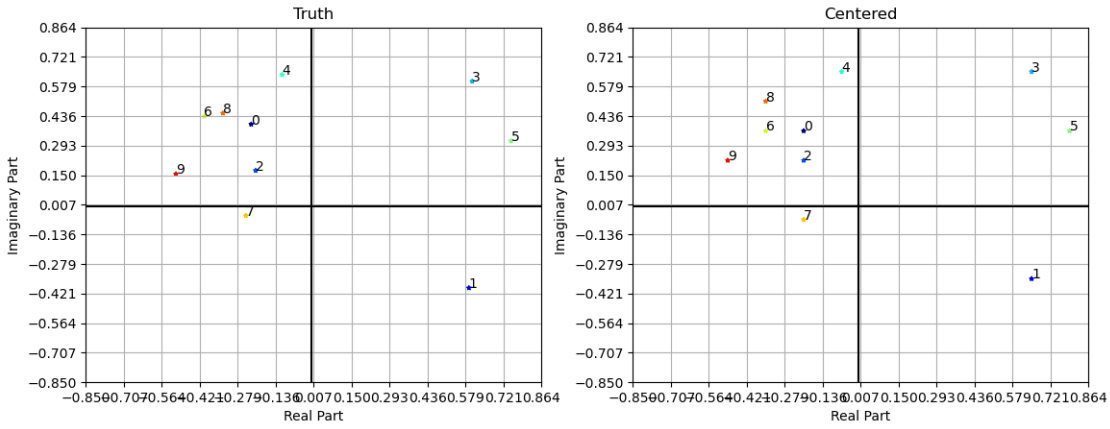

Figure 62: Points after encoding and decoding, shown in a centered position of the binxy

3.11.2 Polar Grid

The Polar Grid is quite similar to the number ordering of the Square Grid, as it starts with 4 as its initial number. However, in this code implementation, instead of having x and y bins, we will have radius and angle bins. One thing to note is that now we have two steps that can be totally independent: one step for the radius and the other for the angle. In the Square Grid, the step was the same for both x and y coordinates. One main advantage of the polar grid is that we don't need to take care of the angle measurement error, as the encoding will handle it within the transformer. However, the polar grid uses more space with fewer patches compared to the square grid.

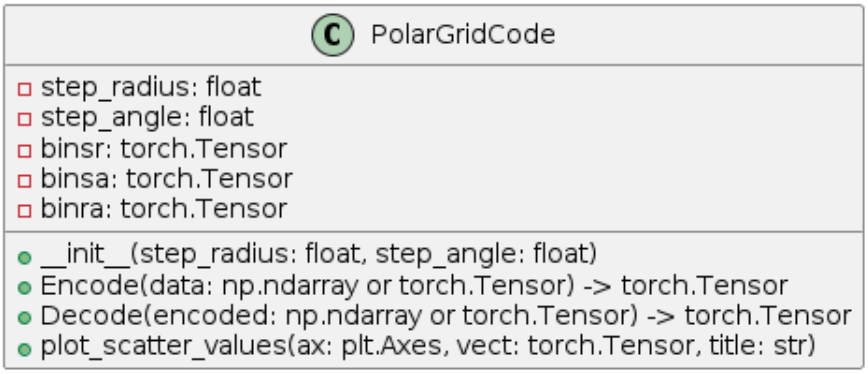

Figure 63: Class attributes for the Radial Grid

- **step_radius**: Steps between 0 and 1
- **step_angle**: Steps between 0 and π
- **binsr**: Number of bins for radius
- **binsa**: Numbers of bins for angles
- **binra**: Combined Grid between radius and angle.

Encoding Recipe:

1. Loop over the binsr for the radius and binsa for the angle, and check if the data value falls in the range of bin i and bin i+1.
2. Save all the values that fall in the bin as r_indices and a_indices. These indices behave more like coordinates for the matrix of binsra.

3. Encoding is done by putting the indices in the binra vector. The binra vector returns a numerical token that represents the position of the complex plane.

Here's an elegant code implementation

```
        # Loop over the bins for the radius and angle coordinates
        for i in range(len(self.binsr) - 1):
            for j in range(len(self.binsa) - 1):
                # Find the indices of the data that lie within
                # the current bin for the radius and angle coordinates
                r_indices[(self.binsr[i] <= r) & (r < self.binsr[i + 1])] = i
                a_indices[(self.binsa[j] <= a) & (a < self.binsa[j + 1])] = j

        # Encode the data by selecting the corresponding
        # bin indices from the bin index matrix
        encoded = self.binra[r_indices, a_indices]
        return encoded
```

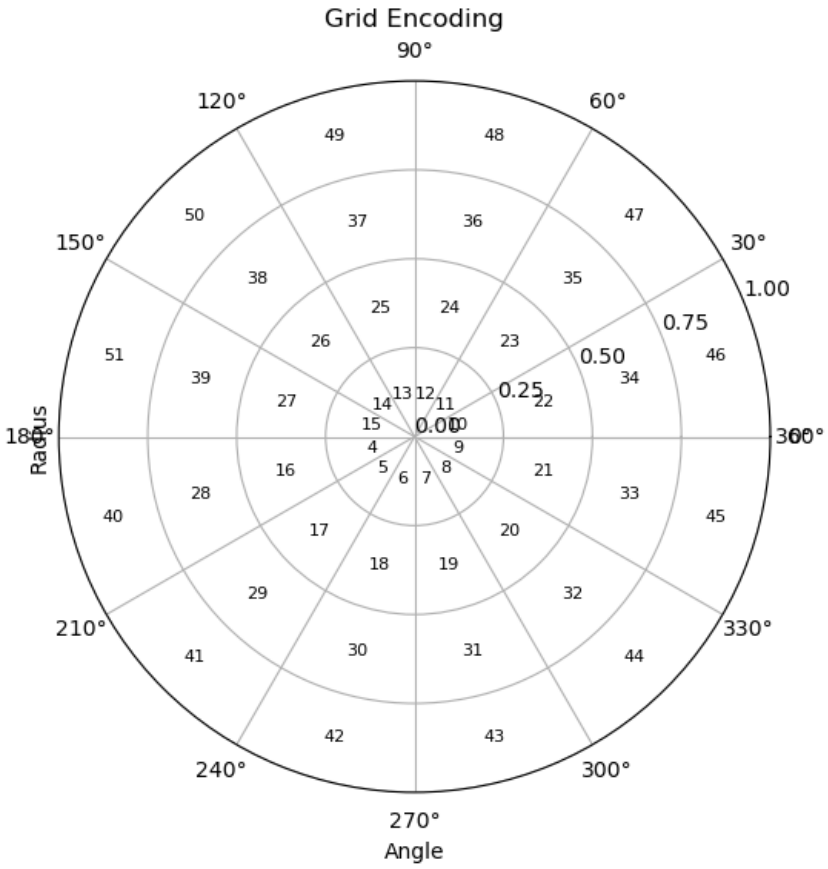

Figure 64: Radial Grid encoding values. Radius step = 0.25 and angle step = pi/6

Decoding Recipe:

1. Loop over the binsra matrix and create a mask of indices that correspond to the encoded values

2. Use the mask of indices to assign values from the binsr and binsa vectors to the corresponding radius and angle decoded vectors. Add half a step to each value to place it in the center of the bin.

3. Build a complex number vector using the magnitud and phase decoded vectors.

Here's an elegant code implementation

```
        # Loop over the bins for the radius and angle coordinates
        for i in range(len(self.binsr) - 1):
            for j in range(len(self.binsa)-1):
                # Find the indices of the encoded values that correspond
                # to the current bin for the radius coordinate
                indices = (encoded == self.binra[i, j])

                # Fill the decoded values for the radius coordinate
                r_decoded[indices] = self.binsr[i] + self.step_radius / 2

                # Fill the decoded values for the angle coordinate
                a_decoded[indices] = self.binsa[j] + self.step_angle /2

        # Convert back to rectangular coordinates
        data_decoded = torch.polar(r_decoded, a_decoded)
        return data_decoded
```

Similarly to the square grid, the values in the polar grid are also centered in the grid during decoding, thus closing again the possible states of the points in the complex plane.

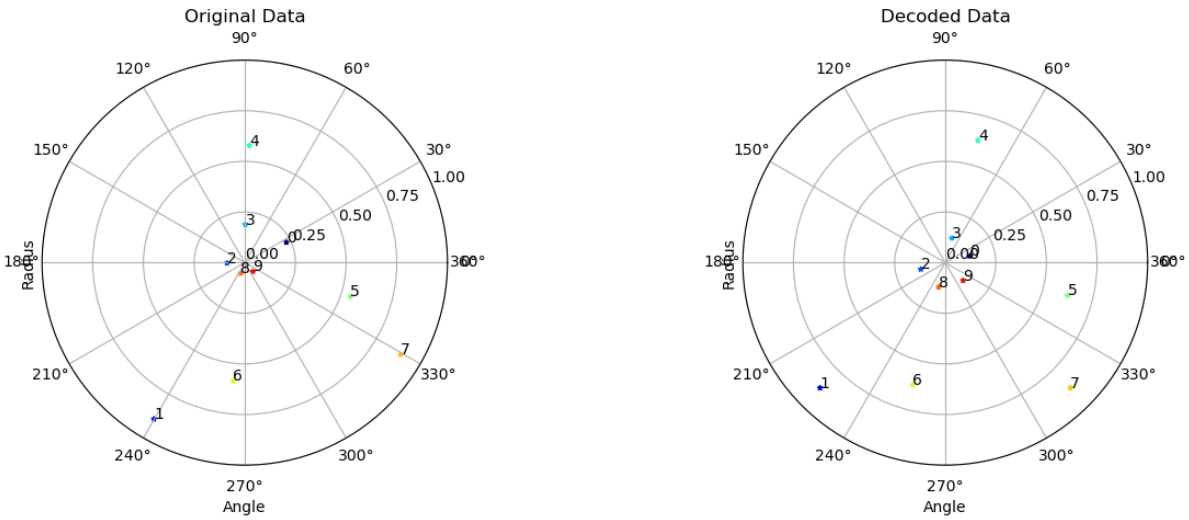

Figure 65: Points after encoding and decoding, shown in a centered position of the binra

3.11.3 Data Augmentation for Improving Error Performance in PolarGrid

Due to the superior ICI cancellation performance of Polar Net, some of the high SNR, had BER errors were reduced to zero. However, this does not indicate a perfect system; rather, it suggests that the error rate did not fail often enough across all of the samples. Therefore, we combined the validation set and training set to artificially augment the data

and allow the system to fail more often. It is important to note that this step of merging the validation and testing set was carried out only after confirming that the network was functioning well with the established parameters. This decision does not affect the training stage in any way.

3.11.4 Transformer implementation

Our transformer implementation can handle both grids by simply changing a hyperparameter, which provides a generic interface for experimentation. For positional embedding, we use a logical sequence of numbers ranging from 0 to 48 positions. Although there were some experiments with the sin embedding, it did not work well for this task. We apply a padding mask to the source and pass it to the transformer. The transformer is not a custom implementation but rather a predefined block already available in PyTorch that contains an encoder and decoder. Note that we implemented a transformer from scratch during the study of this network, but we decided to use PyTorch's standard APIs instead.

The vocabulary size is determined based on the number of patches we have in the grid strategy. This means that the number of possible values that a patch can take in the grid is equal to the number of patches in the grid. For example, if we have a grid with 25 patches, then the vocabulary size is 25.

The loss function used in the training process is given by the cross-entropy section 2.7.1. This is a commonly used loss function for multi-class classification tasks, such as image classification. Mathematically, the cross-entropy loss measures the distance between the predicted probability distribution and the true probability distribution. In the context of the grid encoding task, the predicted probability distribution is the distribution over the patches in the grid, and the true probability distribution is a one-hot vector representing the true patch in the grid. The figure figure 37 in the state of the art can be used to visualize the transformer architecture , at the right is placed the transformer decoder which its has output the output probabilities for symbol A_i using the softmax function figure 11.

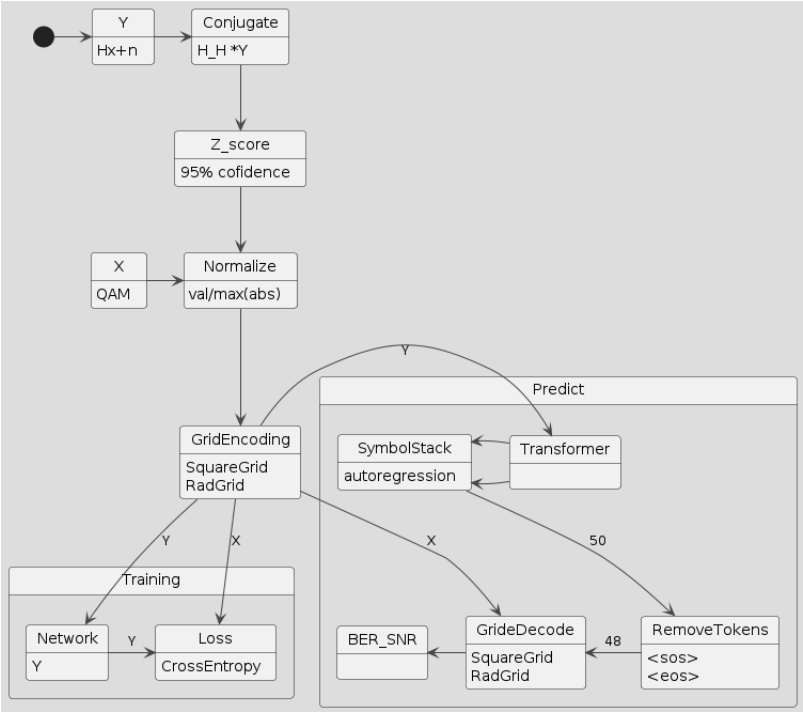

Figure 66: GridNet Workflow

3.11.5 Autoregression in predicting mode

Autoregression in a Transformer refers to the technique of feeding previously generated output tokens as input to the model in order to generate subsequent tokens. In other words, the model generates one token at a time, and each token is conditioned on the previously generated tokens.

This autoregression technique is achieved in a Transformer through a process called self-attention. Self-attention allows the model to capture dependencies between different positions in the input sequence by weighing each input token's relevance to the current token being generated. During training, the model learns to attend to the most relevant tokens from the previous generated output, which in turn influences the distribution of probabilities over the vocabulary for the next token.

In practice, autoregression in a Transformer involves a process called decoding, where the model generates tokens one at a time, starting with a special start-of-sequence <sos> token and continuing until a special end-of-sequence <eos> token is generated or a maximum sequence length is reached. The output tokens generated by the model at each decoding step are fed back as input to the model for the next step, creating a feedback loop that allows the model to incorporate information from previously generated tokens to generate subsequent ones.

3.11.6 Noise in Trainning

This network is the only one that uses variable noise ratios during training to help the model generalize better. The noise varies each epoch and is determined by the following equation:

$$\boldsymbol{SNR_{dB}} = 45 - 5 \times (\boldsymbol{mod(Epoch}, 4)) \tag{93}$$

3.11.7 Transformer hyperparameters

In this section, we first list the hyperparameters that directly affect the final parameter size of the model. Before we show the table, let's review the meaning of the parameters to better understand the hyperparameters table.

- **Embedding_size**: the embedding size refers to the size of the vector that is used to represent each input token in the network. Each token in the input sequence is mapped to a high-dimensional vector of fixed size, which is called the embedding. The embedding captures the semantic meaning of the token in a continuous vector space, which can be learned by the network during training. In practice, the embedding size is often set to a value between 100 and 1000, depending on the size of the input vocabulary and the complexity of the task
- **Num_heads**: Is a hyperparameter that controls the number of parallel self-attention mechanisms in the network. Self-attention is a mechanism that allows the model to weigh the importance of different positions in the input sequence when making a prediction. Self-attention is applied to the input sequence multiple times, with different linear projections of the input sequence used as inputs to each attention mechanism
- **Encoder and Decoder Layers**: The output of the encoder layer is a sequence of hidden representations that captures the relevant information from the input sequence.Both the encoder and decoder layers can be stacked multiple times to improve the quality of the feature representation and the final output sequence.
- **Forward Expansion**: After performing multi-head attention on the input, the result is passed through a feedforward neural network (FFN) layer, which typically consists of two fully connected layers with a ReLU activation function in between, in our case was ever setup to GELU activation function.
- **Dropout**: The dropout rate to apply during training to prevent overfitting. This value typically ranges between 0.1 and 0.3.

Other parameters, such as CONJ_ACTIVE, refer to the same preprocessing step of taking the conjugate of the channel $H^H Y$.

Hyperparameters	Value
BATCHSIZE	100
QAM	16
NUM_EPOCHS	24
SNR	45-25
CONJ_ACTIVE	True
NOISE	True
LEARNING_RATE	.0001
GRID	Square
STEP	1/7

Table 12: Square GridNet hyperparameters

Hyperparameters	Value
BATCHSIZE	100
QAM	16
NUM_EPOCHS	13
SNR	45-20
CONJ_ACTIVE	True
NOISE	True
LEARNING_RATE	.0001
GRID	Polar
STEP_RADIUS	0.25
STEP_ANGLE	$\pi/6$

Table 13: Polar GridNet hyperparameters

3.11.8 Parameters

Name	Value
loss_f	CrossEntropyLoss
src_word_embedding	Embedding (101 K)
src_position_embedding	Embedding (25.6 K)
trg_word_embedding	Embedding (101 K)
trg_position_embedding	Embedding (25.6 K)
transformer	Transformer (44.1 M)
fc_out	Linear (102 K)
dropout	Dropout (0)
Total params: 44.5 M Total estimated model params size (MB): 177.990	

Table 14: Square Grid Transformer Parameters

Name	Value
loss_f	CrossEntropyLoss
src_word_embedding	Embedding (26.1 K)
src_position_embedding	Embedding (25.6 K)
trg_word_embedding	Embedding (26.1 K)
trg_position_embedding	Embedding (25.6 K)
transformer	Transformer (69.3 M)
fc_out	Linear (26.2 K)
dropout	Dropout (0)
Total params: 69.5 M Total estimated model params size (MB): 277.842	

Table 15: Polar Grid Transformer Parameters

To summarize, the Radial Grid prioritizes investing less in the vocabulary size and compensating for it by adjusting other hyperparameters such as multi-head attention and increasing the number of feedforward parameters in order to achieve better results.

4 Results

In order to assess the effectiveness of various equalization methods, we performed a series of five experiments for each method in a consistent manner, and subsequently calculated the average of their results. This approach is based on the principles of Monte Carlo tests[50], aiming to obtain representative statistics of the ensemble, rather than only smoothing out the curves. By conducting multiple experiments and averaging the outcomes, we can ensure a more reliable representation of each method's performance. Although the channel dataset remains constant, the transmitted information is described by a normal distribution with random data for each iteration. This also promotes the use of Monte Carlo techniques.

In our research, we evaluated the performance of our method in both noisy and non-noisy scenarios across a wide range of signal-to-noise ratios (SNRs). The SNR range we considered was from 45 to 5, with a step of 2 between each SNR. By including SNRs below the typical operating range, we could assess the system's robustness to low signal levels. Although we did not implement a MIMO system in this study, high SNR values are also relevant for MIMO systems, and this evaluation could provide insight into their performance under different conditions. The same about high SNR values apply to the LOS channels.

This comprehensive evaluation helped us to identify the strengths and weaknesses of each method and determine which methods are most effective in different scenarios.

Our performance metrics are the BER (Bit Error Rate) and BLER (Block Error Rate) , which have been explained in the theoretical framework. BER measures the ratio of incorrectly received bits to the total number of transmitted bits, while BLER measures the ratio of incorrectly received blocks to the total number of transmitted blocks. A lower value for both metrics indicates better performance of the system.

4.1 Multiplications and Big O analysis

Using execution time and source code analysis, we determined the FLOPS for each equalizer in this section and evaluated the model's complexity. We narrowed our focus to the multiplication operations exclusively in order to make our study more portable to other CPU architectures. The number of parameters was estimated through the source code analysis for the Big O notation analysis. Big O notation is a mathematical notation that expresses the upper bound of the growth rate of an algorithm in terms of its input size. It provides a way to express the worst-case time complexity of an algorithm in terms of a simple function of the input size, ignoring constant factors and lower-order terms. The efficiency and scalability of the algorithm can be analyzed through this analysis. A de-

tailed description of the system specifications used for the time measurement experiment is provided in the table below.

Table 16: Processor specifications

Property	Value
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
Address sizes	43 bits physical, 48 bits virtual
Byte Order	Little Endian
CPU(s)	16
Model name	AMD Ryzen 7 3700X 8-Core Processor
CPU family	23
Model	113
Thread(s) per core	2
Core(s) per socket	8
Frequency boost	enabled
CPU max MHz	4426.1709
CPU min MHz	2200.0000
BogoMIPS	7186.16

4.1.1 LMMSE

Using equation (32) to invert the channel H requires the naive matrix inversion operation, which has a computational complexity of $\boldsymbol{O(N^3)}$ operations, where N is the matrix size [60].

Similarly, the total number of internal complex multiplications involved in matrix multiplication is N^3. In the following table we will describe the total complex multiplications required.

Table 17: Number of Operations by Type

Operation	No. Complex Mults	Big O
H^H	0	$O(N^2)$
$H^H H$	48^3	$O(N^3)$
σI	0	$O(N)$
$(H^H H + \sigma I)^{-1}$	48^3	$O(N^3)$
$(H^H H + \sigma I)^{-1} H^H$	48^3	$O(N^3)$
$(H^H H + \sigma I)^{-1} H^H H$	48^3	$O(N^3)$
$(H^H H + \sigma I)^{-1} H^H H Y$	48^2	$O(N^2)$
Total	**552,960**	$O(4N^3 + 3N^2 + N)$

The time measurements in the code give an average time of 5.96e-02, which is equal to t = 59.6 ms. Therefore, the estimated FLOPS considering only the complex multiplications is calculated as follows:

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{552,960}}{\mathbf{59.6}\boldsymbol{ms}} = \mathbf{9.27 \times 10^6} \tag{94}$$

4.1.2 Zero Forcing

In the equation (20) we can describe this equalizer. This involves 48 extractions of diagonal elements and then an elementwise divison by each element of vector Y. This involves 48 multiplications of complex numbers. The time measurements in the code give an average time of 2.62e-03, which is equal to t = 2.62 ms. Therefore, the estimated FLOPS is calculated as follows:

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{48}}{\mathbf{2.62}\boldsymbol{ms}} = \mathbf{18.320} \times \mathbf{10^3} \tag{95}$$

For the time complexity of this algorithm is conisdered a $\boldsymbol{O(N)}$ complexity, because diagonal is size of N.

4.1.3 PhaseNet

The preprocesing stage for Phase Net shown in Figure 52 uses the Hermitian transpose multiplication with the received signal $\boldsymbol{Y}$ with an $\boldsymbol{O(N^2)}$ complexity. Next, we need to extract the phase of the rectangular form of the complex numbers, which involves an arctan function that is considered almost linear. For the network stage, based on Table 4 in 3.7.2, we can visualize all the number of parameters implied in the networks, and each linear layer basically performs a matrix multiplication of weights times the last input. In the case of PhaseNet, there are 4 linear layers, which is equivalent to $\boldsymbol{O(4N^2)}$ plus the preprocesing is $\boldsymbol{O(5N^2)}$. Note that only normal products are involved here, not complex ones, which leads to a lower constant.

Table 18: Complexity of PhaseNet

Preprocessing Stage	Complexity
$H^H Y$	$O(N^2 + N)$
$arctan$	$O(N)$
Network Stage	
4 linear layers	$O(4N^2)$
Total complexity including preprocessing	$O(5N^2 + 2N)$

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{1,761,808}}{\mathbf{.119}} = \mathbf{14.8} \times \mathbf{10^6} \tag{96}$$

4.1.4 PolarNet

This network has the same preprocessing as PhaseNet, plus additional zero forcing, with their corresponding Big O growth rates PhaseNet, Zero Forcing, respectively. Additionally the network has MagnitudNet has the same preprocessing stage as PhaseNet, which

results in a final growth rate of $O(10N^2 + 2N)$. Taken from the 3.8.2 we finally calculate the FLOPS.

$$\boldsymbol{FLOPS \simeq \frac{1,761,808 + 5,856}{.392} = 4.5 \times 10^6} \tag{97}$$

4.1.5 ComplexNet

This network deals with complex operations, and to make it portable with the gradients and machine learning environment, the "apply complex" function was needed, which can be seen in section 3.9.1 and results in a complexity of $O(4N^2)$ per each linear complex layer. Based on Table 8 in 3.9.4 the network has 6 linear layers, plus the preprocessing (which involves complex conjugation) and leads to a final complexity of $O(13N^2)$. Again, in Table 8, the number of parameters used in the FLOPS calculation can be found.

$$\boldsymbol{FLOPS \simeq \frac{484,248}{.173} = 2.7 \times 10^6} \tag{98}$$

4.1.6 MobileNet

The original MobileNet model has a time complexity of approximately $O(n^{2.3})$, MobileNetV2 has a time complexity of approximately $O(N^2)$ and MobileNetV3 has a time complexity of approximately $O(N \times log(N))$ [25, 24]. It's important to note that the actual runtime performance of MobileNet can also be affected by factors such as the hardware platform, software optimization, and batch size.

For the final growth rate of our MobileNetV3 implementation, we modified the final layer with a linear layer, as shown in the Figure 58, and applied a pre-processing step using a Hermitian matrix, and the pointwise division to emulate zero-forcing (20), the network has a final time complexity of $O(N^2 + 2N + Nlog(N))$. To access all the number of parameters, we used a software API that iterated over all parameters and added up their total size to calculate it.

Algorithm 4 Count Parameters

```
function COUNT_PARAMETERS(model)
    total_parameters ← 0
    for p in model.parameters do
        if p.requires_grad then
            total_parameters ← total_parameters + numel(p)
        end if
    end for
    return total_parameters
end function
```

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{3,170,688}}{\mathbf{.834}} = \mathbf{3.801 \times 10^6} \tag{99}$$

4.1.7 GridNet

Based on Algorithm 4 we can estimate the same for GrideNet versions. For the GridNet Square we have this FLOPS:

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{67,773,587}}{\mathbf{1.93}} = \mathbf{35.11 \times 10^6} \tag{100}$$

and for GridNet Polar is

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{69,460,531}}{\mathbf{2.5}} = \mathbf{27.8 \times 10^6} \tag{101}$$

The Transformer model, has a time complexity of $\boldsymbol{O(N^2)}$, where $\boldsymbol{N}$ is the length of the input sequence. This is due to the self-attention mechanism, which computes a similarity score between each pair of tokens in the input sequence, resulting in an $\boldsymbol{N \times N}$ matrix. The computation of this matrix requires $\boldsymbol{O(N^2)}$ time.

In addition to the self-attention mechanism, the Transformer also includes feedforward layers and normalization layers, but their time complexity is typically much lower than that of the self-attention mechanism. Therefore, the overall time complexity of the Transformer can be approximated as $\boldsymbol{O(N^2)}$ [64].

However, there are optimizations that can be applied to reduce the time complexity of the Transformer, such as restricting the maximum length of the input sequence or using approximate attention mechanisms. These optimizations can reduce the time complexity to$\boldsymbol{O(N * log(N))}$ or even $\boldsymbol{O(N)}$, but at the cost of reduced accuracy or increased memory usage. Adding the preprocing stage of grids which can be viewed on Figure (61) and Figure (64) with N cells denoted as $|\mathbb{C}|$ and the Hermitian Conjugate applied in other networks, the final complexity can be described as $\boldsymbol{O(N^2 + N * log(N) + |\mathbb{C}|)}$.

4.1.8 OSIC

Based on the analysis in subsection 2.4.1 we end up with the big O notation found in (33). Based on the algorithm in the same section 2.4.1 we can make a table due to the short of algorithm looks like.

Operation	Count (multiplications/additions)	Iterations	Total Operations
Element-wise subtraction (a_est - conste)	16	48	768
Element-wise squaring	16	48	768
Sorting (torch.argsort(sest2))	$16 * \log_2(16)$	48	~768
Scalar multiplication (sest[ind] * mag2)	48	48	2304
Scalar addition (rm - ...)	48	48	2304

Table 19: Approximate number of products in the OSIC_Det function

Total number of products (multiplications and additions) ≈ 48 * (16 + 16 + (16 * log2(16)) + 48 + 48) = 9216

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{9216}}{\mathbf{640 \times 10^{-3}}} = \mathbf{14.4 \times 10^{3}} \tag{102}$$

4.1.9 NearML

Based on the analysis in subsection 2.4.2 we end up with the big O equation found in (34). Due to algorithm is more complex than OSIC we can aproach the numer of multiplications as $(\mathbb{A} * N + N)L$, with $\mathbb{A} = 16$ and $N = 48$ and $L = 4$ branching levels $(\mathbb{A} * N + N)L = 3264$. And finally with time $t = 640ms$ let's calculate the total floating-point operations per seconds (FLOPS):

$$\boldsymbol{FLOPS} \simeq \frac{\mathbf{3264}}{\mathbf{7.35}} \simeq \mathbf{444} \tag{103}$$

4.1.10 Performance Analysis: FLOPs and Time Complexity Benchmark

This FLOPS value works like an estimate and should not be taken as definitive. One of the main advantages of deep learning techniques is that they are easy to parallelize, and FLOPS can be significantly reduceded. Big O notation is a mathematical notation used to describe the upper bound of the growth rate of a function or algorithm in terms of its input size.

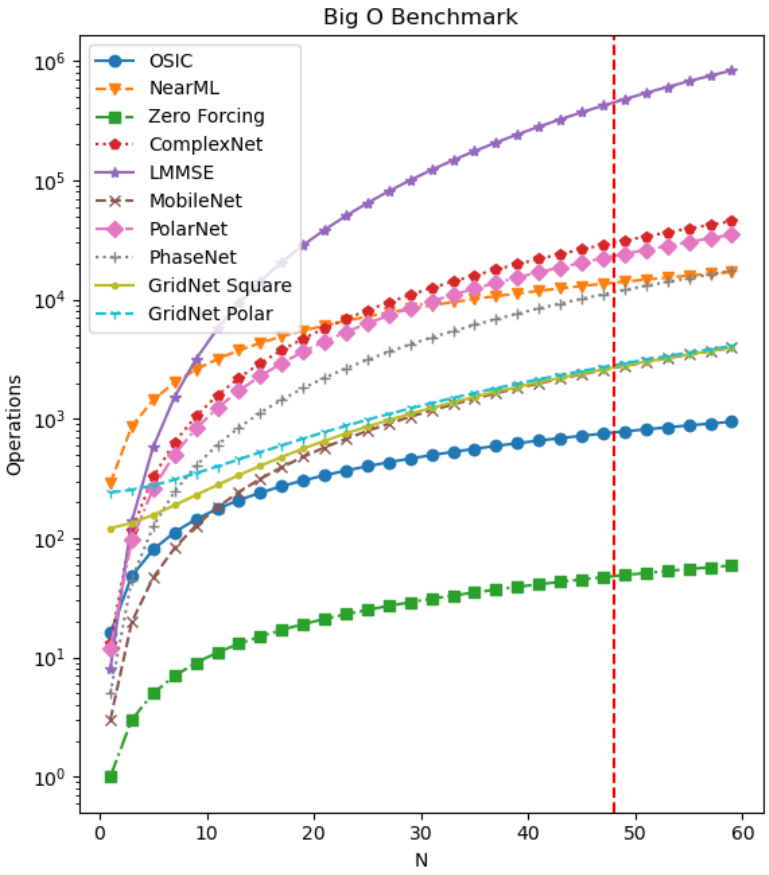

Figure 67: Big O Benchmark for 16QAM. Dotted red line is the case for N=48

Table 20: FLOPS and Time Complexity

Name	FLOPS	Big O Complexity
OSIC	14.4×10^3	$O(N \times \lvert\mathbb{A}\rvert)$
NearML	444	$O(\lvert\mathbb{A}\rvert \times N \times (M + \lvert\mathbb{A}\rvert))$
Zero Forcing	18.320×10^3	$O(N)$
ComplexNet	2.7×10^6	$O(13N^2)$
LMMSE	9.27×10^6	$O(4N^3 + 3N^2 + N)$
MobileNet	3.801×10^6	$O(N^2 + 2N + Nlog(N))$
PolarNet	2.7×10^6	$O(10N^2 + 2N)$
PhaseNet	14.8×10^6	$O(5N^2)$
GridNet Square	35.11×10^6	$O(N^2 + N * log(N) + \lvert\mathbb{C}\rvert)$
GrideNet Polar	27.8×10^6	$O(N^2 + N * log(N) + \lvert\mathbb{C}\rvert)$

4.2 BER and BLER

To provide a detailed comparison of the BER and BLER values across different SNR values, we will analyze the results obtained from various methods. Firstly, we will compare the linear methods, followed by a comparison of the non-linear methods. This approach will provide a comprehensive view of both plots.

Neural networks are typically run in custom-built Python frameworks, as shown in Figure 43. This approach enables a consistent environment for each evaluation execution, where all BER and BLER values are obtained from a processed CSV file and then plotted using the Matplotlib Python library. The resulting visualizations provide an effective means of analyzing and interpreting the performance of these neural networks.

In order to facilitate faster testing evaluations, the SNR values in the x-axis are incremented by a step of 2, ranging from 5 to 45 SNR. To generate smoother curves, a cubic interpolator is applied to the plotted data, which may introduce artifacts and hills in the curves. It is important to note that these hills and artifacts should not be interpreted as final results. To achieve completely smooth curves without valleys or hills, additional realizations may be required to have more realizations falling in this values.

4.2.1 PhaseNet

In Figure 68 the PhaseNet model shows an upgrade of up to one order of magnitude. The model plateaus at 10^{-5} BER from 30db to higher values. This can be achieved due to the preprocessing stage, which sets all phase points to the closest points but with incorrect phase. This enables the networks to classify well between only four possible angle positions in the QPSK scneario. This turn allows the network to handle noise scenarios well and correct the ISI by comparing all values at the same time due to the parallel input nature with the feedforward linear layer described in (43).

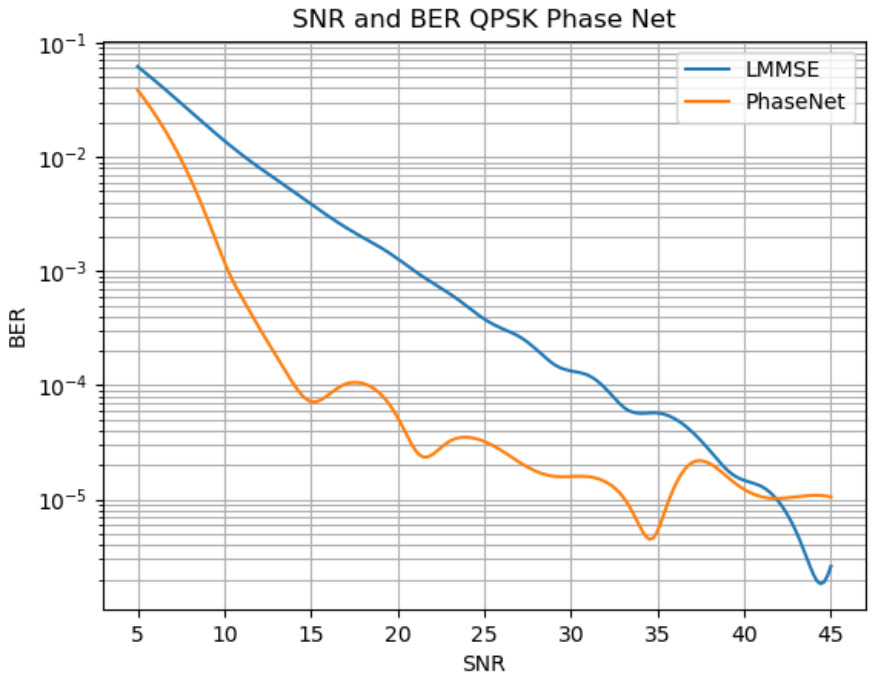

Figure 68: QPSK BER : PhaseNet vs. LMMSE

Regarding the block errors, the same gain of one order of magnitude is observed in the noisiest scenarios, up to 25 dB. However, beyond this point, the model cannot infer any further relation between the points, and as a consequence, the BLER flattens out at $BLER = 9 \times 10^{-3}$.

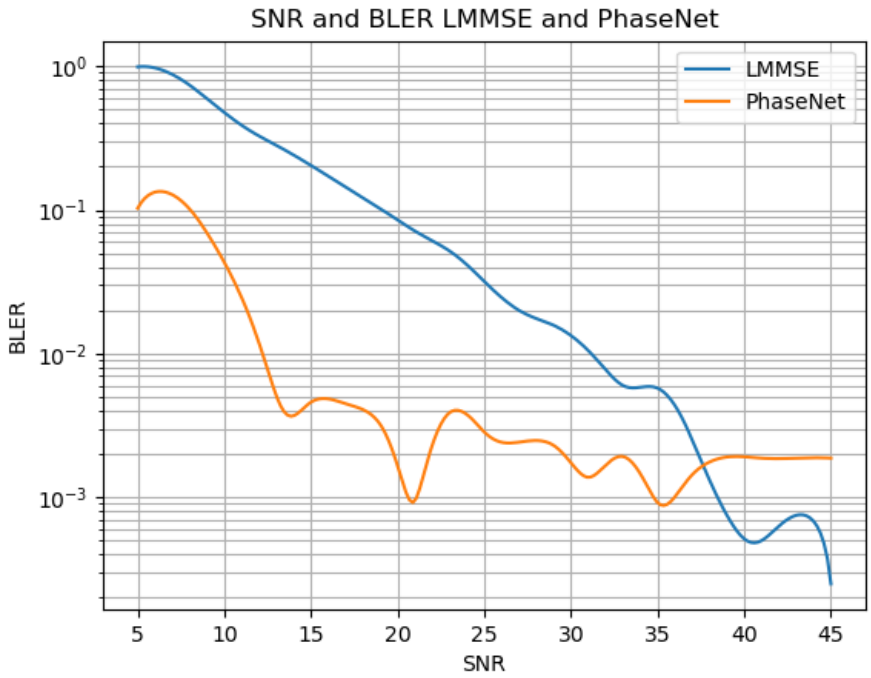

Figure 69: QPSK BLER : PhaseNet vs. LMMSE

As seen in Figure 70, the non-linear model exhibits similar linear growth in relation to lower SNR values, while PhaseNet follows a positive concave shape with an improvement of one order of magnitude.

One positive aspect of the PhaseNet model is that the number of symbols for bigger PSK does not affect the time complexity of the networks. Only retraining is necessary, as only the number of symbols in the frame matters for the networks and the relative positions are learned. In fact, more constellation points lead to a more complex and less accurate precision.

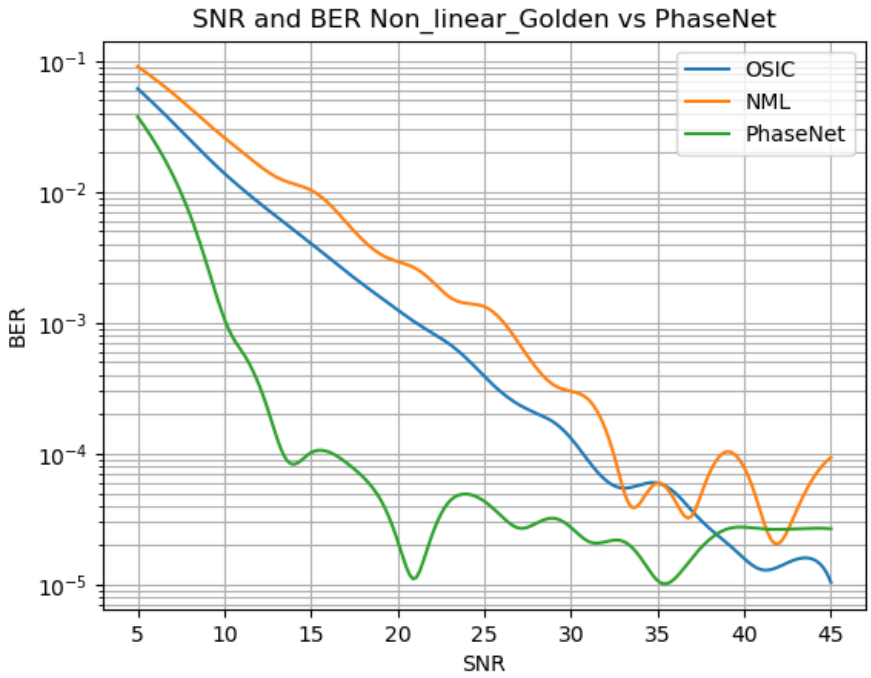

Figure 70: QPSK BER: PhaseNet vs. OSIC and NML

In the case of BLER, the non-linear equalizer exhibits similar performance, while PhaseNet performs one order of magnitude better on average.

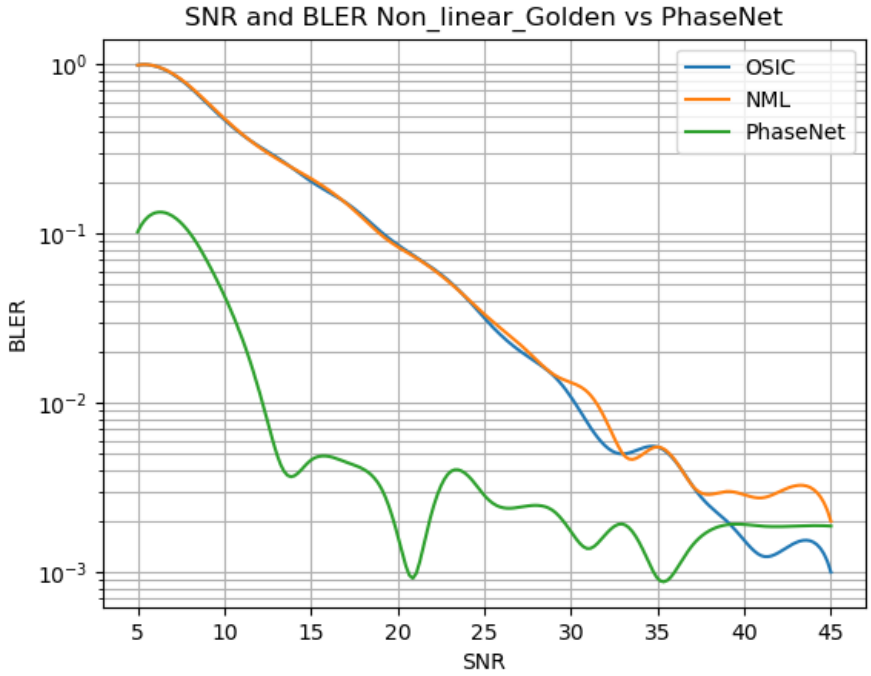

Figure 71: QPSK BLER: PhaseNet vs. OSIC and NML

4.2.2 PolarNet

This network combines a 16-PSK PhaseNet equalization with an additional MagNet equalization. In this model, we assume that phase and magnitude are completely separate from each other, see in subsection 3.8 for more detail. One of the main reasons is to maintain a more modular architecture, which means that in a future implementation, it can be equalized a 16-PSK or 16-QAM with almost the same processing stages. Another key feature is that each network is specialized, reducing the error to only its specific task. Therefore, in theory, this will make the error not influence each other.

Observing the graph below, it can be seen that the model flattens at a BER of 10^{-3} at 30dB. The main reason for this result is that, since we used the Mean Squared Error (MSE) to correct the weights, smaller error values than 10^{-3} were not sufficient to modify the internal network weights. This leads to the model plateauing, where it reaches a point where it cannot improve any further. Also we keep in mind that this models has A^2 elements in comparison with QPSK, so curves will look closer to the golden models.

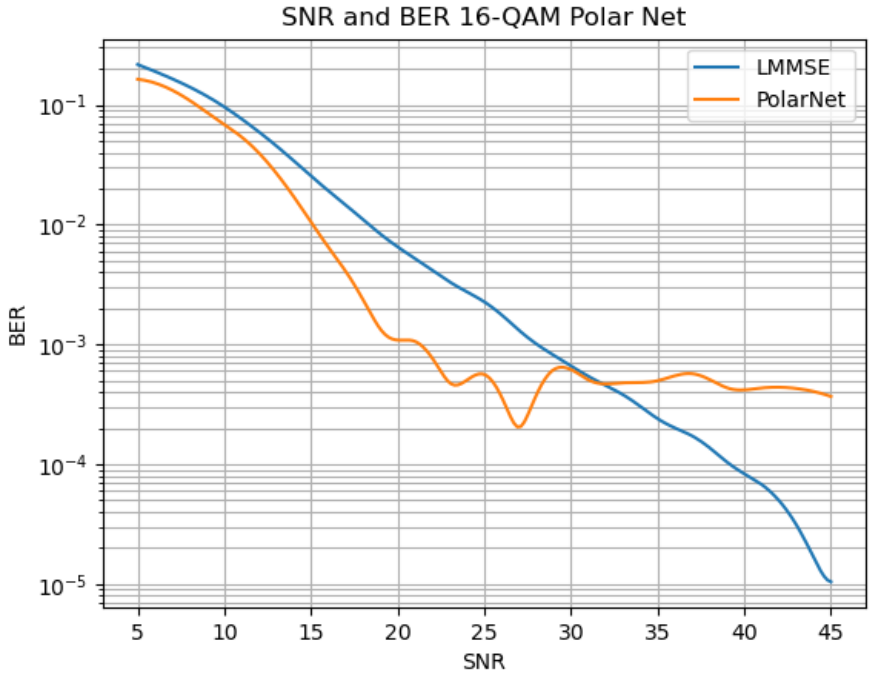

Figure 72: 16-QAM BER: PolarNet vs. LMMSE

Regarding the BLER in Figure 73 , it is shown that the system can consistently equalize a certain quantity of blocks and prove that most of the errors are given from far away points in the constellation. However, in general, the network is capable of clustering the data points around their corresponding constellation points.

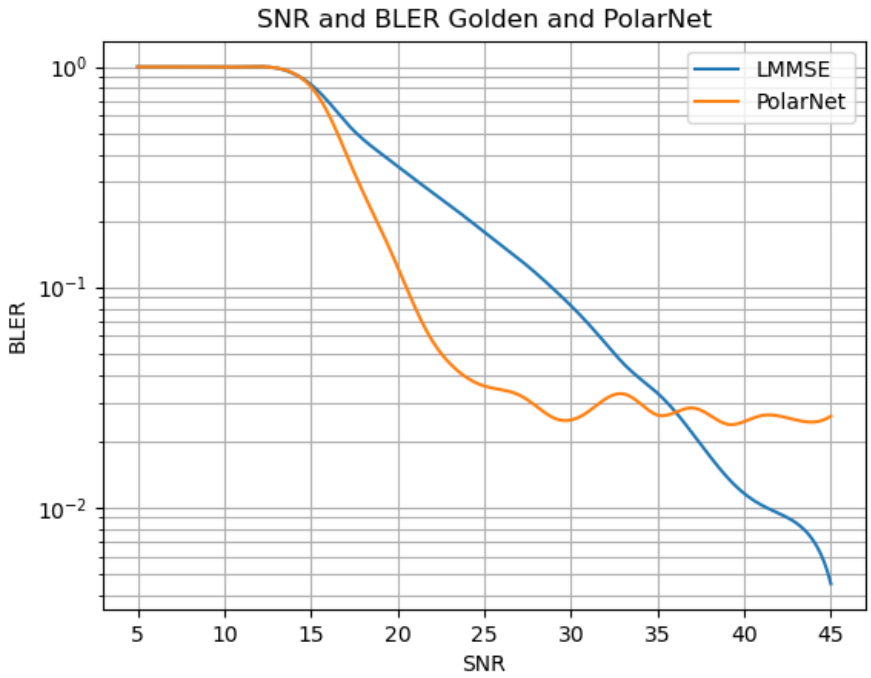

Figure 73: 16-QAM BLER: PolarNet vs. LMMSE

Similarly, the non-linear equalizers outperform the neural network in low noise scenarios with SNR values greater than 30 dB. This is because the neural network truncates error during the training stage and additional preprocessing stages may be necessary to achieve lower BER. To balance the trade-off between signal recognition over noise and overfitting, sacrificing performance beyond 30 dB may be necessary, unless alternative preprocessing stages, different from the Hermitian matrix, can be implemented.

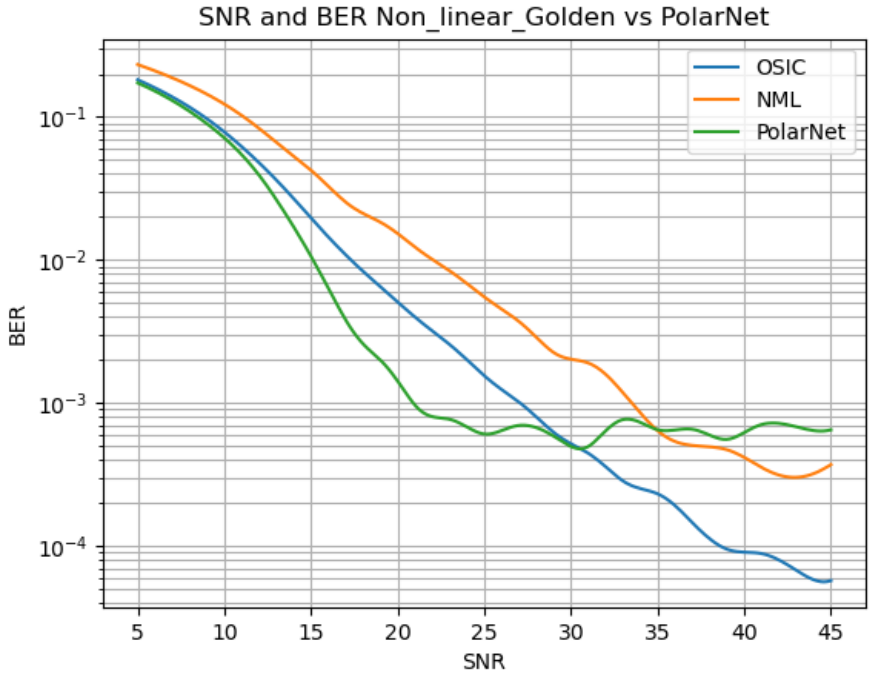

Figure 74: 16-QAM BER: PolarNet vs. OSIC and NML

Despite the model being truncated for a BLER of 35dB, the compromise is reasonable since the model outperforms the non-linear equalizers within a range of 15dB to 35dB. This suggests that most of the errors reported in the BER section are outliers. By centering the input values around the mean, neural networks can learn patterns in the data more effectively without being overly influenced by extreme values. Therefore, values closer to the mean can provide a balanced representation of the data and be well-suited for neural networks, in this case it can be seen in BLER plot.

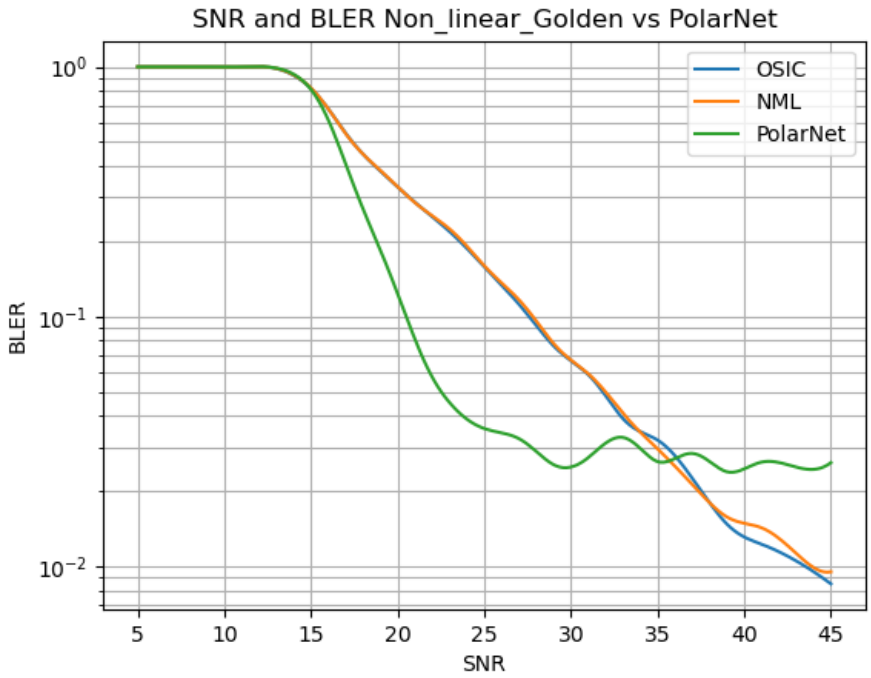

Figure 75: 16-QAM BLER: PolarNet vs. OSIC and NML

4.2.3 ComplexNet

In complex net we try two different loss functions to traing the neuronal network. However, logarithmic polar loss in equation (65) was not as effective because it produced high loss numbers that were larger than 1, which slow network error convergence and make high jumps in the gradient. As a result, the network had difficulties updating its weights and optimizing its performance using this loss function. On the other hand, the complex MSE loss (63) was much more effective because its output values were lower than those of the initial loss functions. This made it easier for the networks to determine which parameters were good or not. By using an effective loss function, the network can optimize its performance and achieve higher accuracy in its predictions. This is particularly important in communication systems where accuracy and reliability are critical for successful operation.

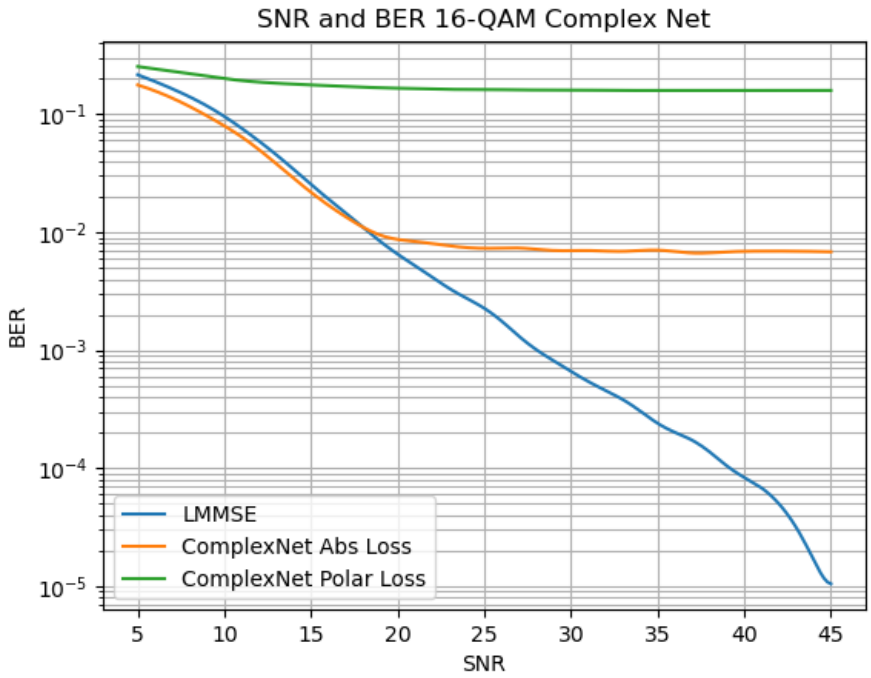

Figure 76: 16-QAM BER: ComplexNet with Two Losses vs. LMMSE

In the BLER results, we avoided plotting the ComplexNet Polar logarithmic loss because it performed poorly with a BLER of 1 across all blocks. On the other hand, Abs loss networks attempted to approximate the golden standard, but were not quite good enough. One of the main reasons for this is that the proposed methods in the state-of-the-art literature and the ones implemented in our research are not mature enough to help with the equalization task. As seen in [7], there are some assumptions about the activation functions in the complex plane. Additionally, we used automatic differentiation provided by the pytorch framework, which may have struggled with complex conjugation. Implementing a solution from complex differentiation can be challenging and may require a separate research project for someone with strong mathematical skills.

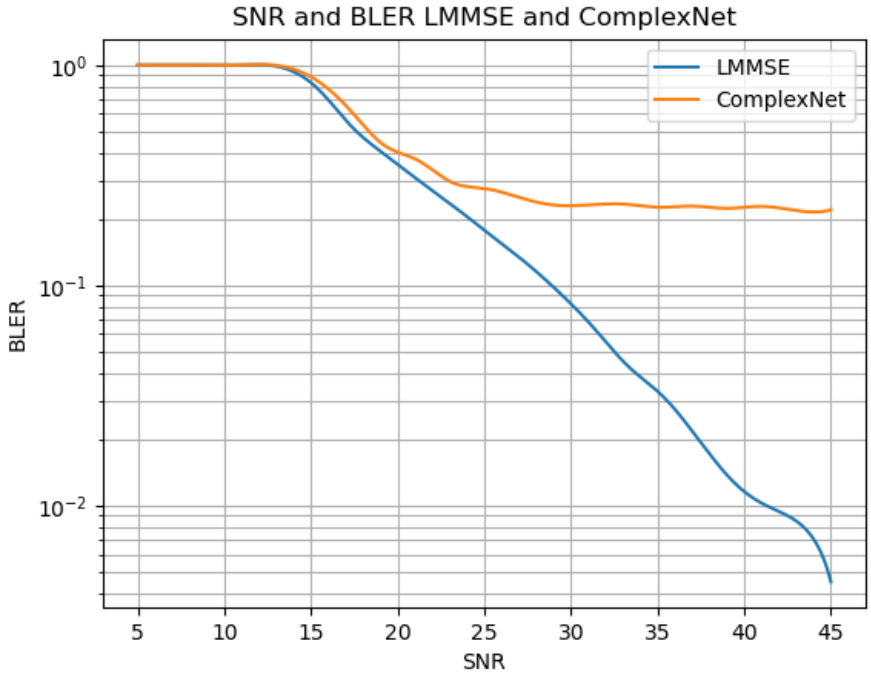

Figure 77: 16-QAM BLER: ComplexNet with Two Losses vs. LMMSE

ComplexNet performs marginally better than NML in situations with higher noise levels and is nearly identical to OSIC for values below 15dB. Intriguingly, the model's loss closely mirrors that of OSIC, indicating that the neural networks are attempting to generalize a solution similar to OSIC. The primary advantage of these neural networks is their greater parallelizability. Concerning time complexity, the network's growth is only dependent on the N scale and not the alphabet of symbols. Therefore, for larger QAM systems, this network may be a more suitable solution. It is recommended to conduct further testing with higher QAM constellations. However, higher QAM constelations is outside this research scope.

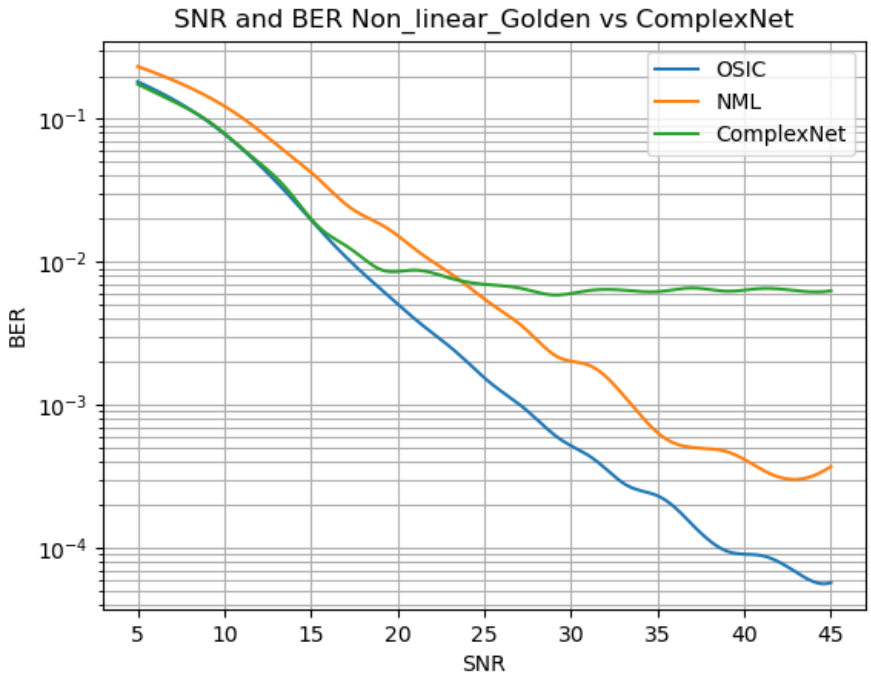

Figure 78: 16-QAM BER: ComplexNet vs. OSIC and NML

For the BLER graph the model does not take into account whether the data is close to the mean or an outlier, as it faces difficulties in generalizing effectively. Consequently, the presence or absence of outliers in the data has little impact on the model's inherent error. This could be attributed to the error function, and as previously mentioned, the research on complex values for neural networks is still in its early stages. Therefore, it is reasonable to expect that the current state-of-the-art implementation may not be well-suited for this specific problem.

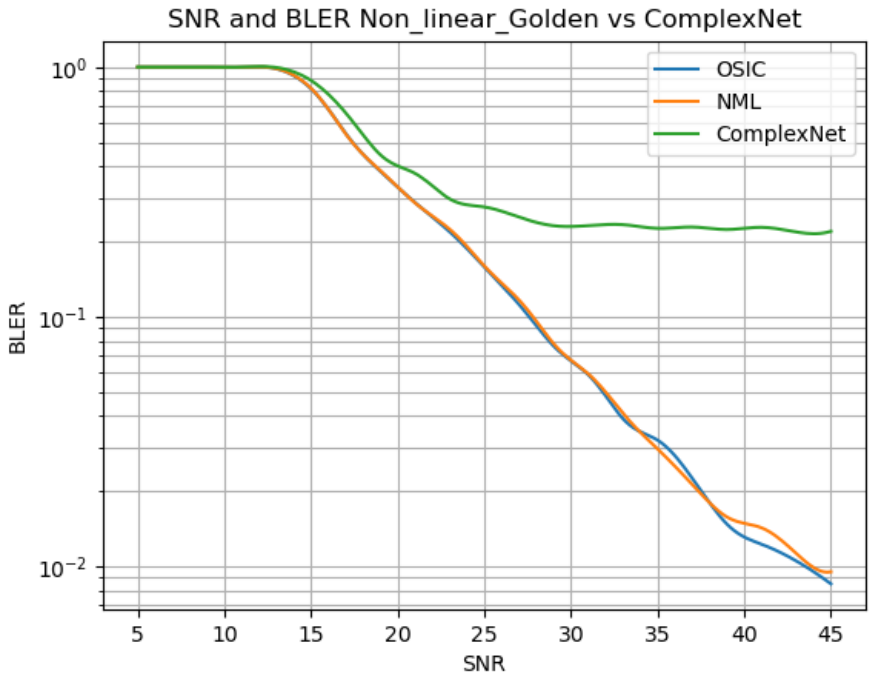

Figure 79: 16-QAM BLER: ComplexNet vs. OSIC and NML

4.2.4 MobileNet

Although MobileNet is not the most accurate convolutional neural network, it is still considered one of the fastest models in terms of processing speed and FLOPS among state-of-the-art methods. The main idea behind MobileNet is to provide channel compression into a vector seen in Figure 58 and then perform zero forcing with the received signal, which offers a better solution than single zero forcing in subsection 2.3.2 , this because netowrk take in accounts noise and ISI. For SNR values of 23 dB and below, it could be a viable option instead of zero forcing. In fact, it performs better than zero forcing in the range of 15 dB to 25 dB, which is encouraging because it supports the hypothesis that it is essentially a zero-forcing method with noise consideration.

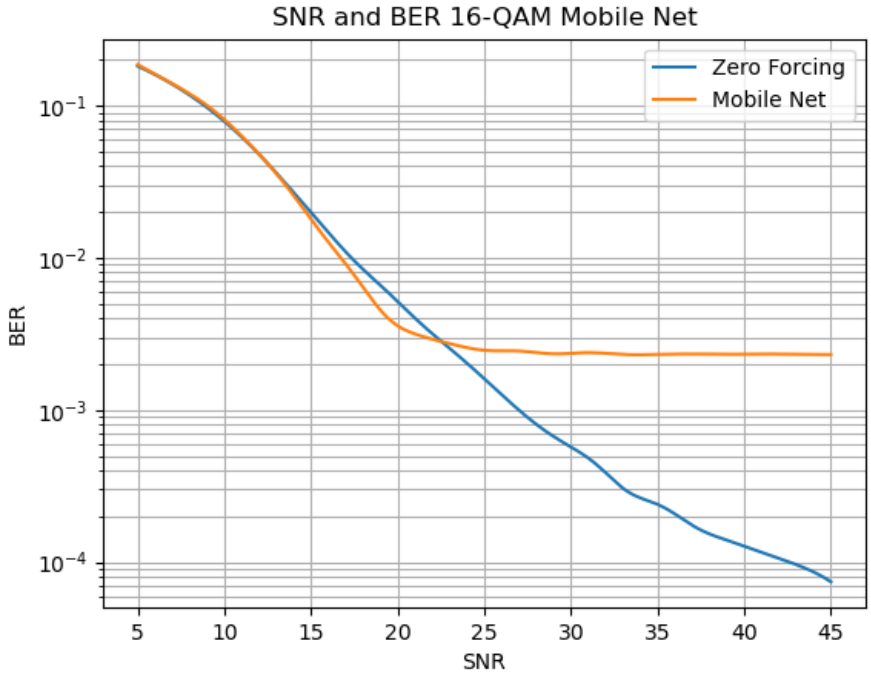

Figure 80: 16-QAM BER: MobileNet vs. ZeroForcing

Due to its ability to account for noise, MobileNet can identify signals more accurately than single zero-forcing, resulting in a lower block error rate than the golden model until the model flattens at 30 dB. It is important to note that MobileNet has a higher computational complexity than single zero-forcing. However, the lower block error rate achieved by MobileNet justifies the additional resources required for this equalization strategy.

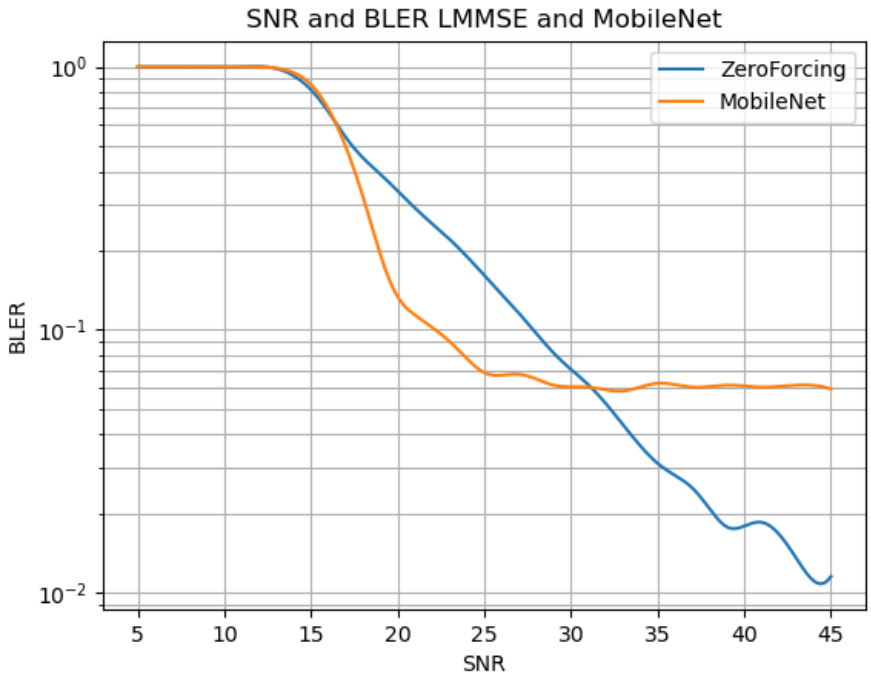

Figure 81: 16-QAM BLER: MobileNet vs. ZeroForcing

For SNR values below 30 dB, MobileNet can be a viable choice due to its lower computational complexity compared to non-linear models, while also achieving a better BER and exhibiting behavior similar to that of OSIC.

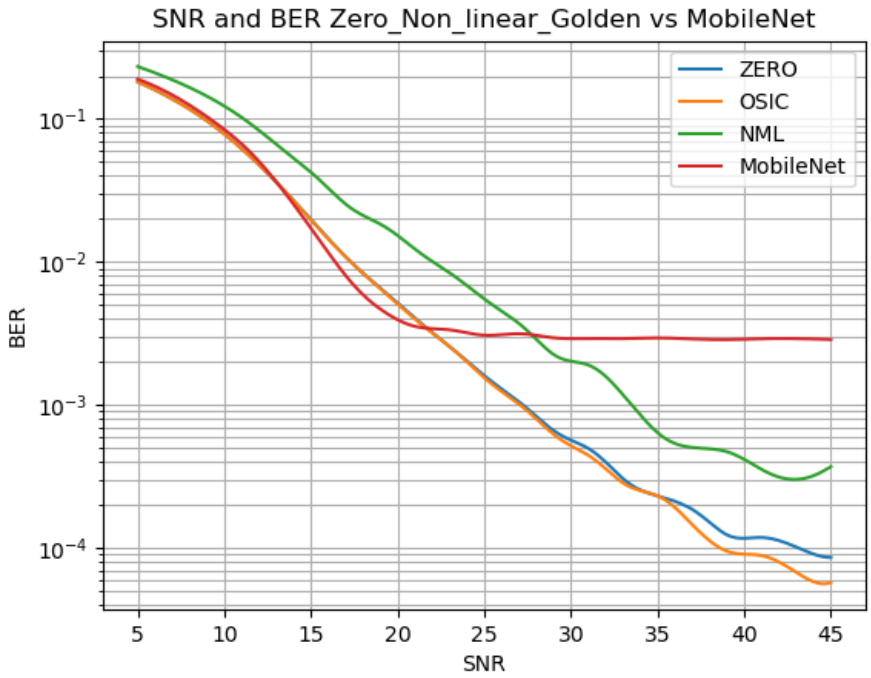

Figure 82: 16-QAM BER: MobileNet vs. Zero forcing, OSIC and NML

The block error rate is quite good for values lower than 30dB, which can be attributed to the intelligent zero-forcing approach that effectively cancels the Intersymbol Interference (ISI), resulting in an improved BLER. This helps to ensure data integrity is maintained as much as possible. One potential future strategy could involve using this network as a preprocessing step to feed data into other advanced networks, further enhancing their performance.

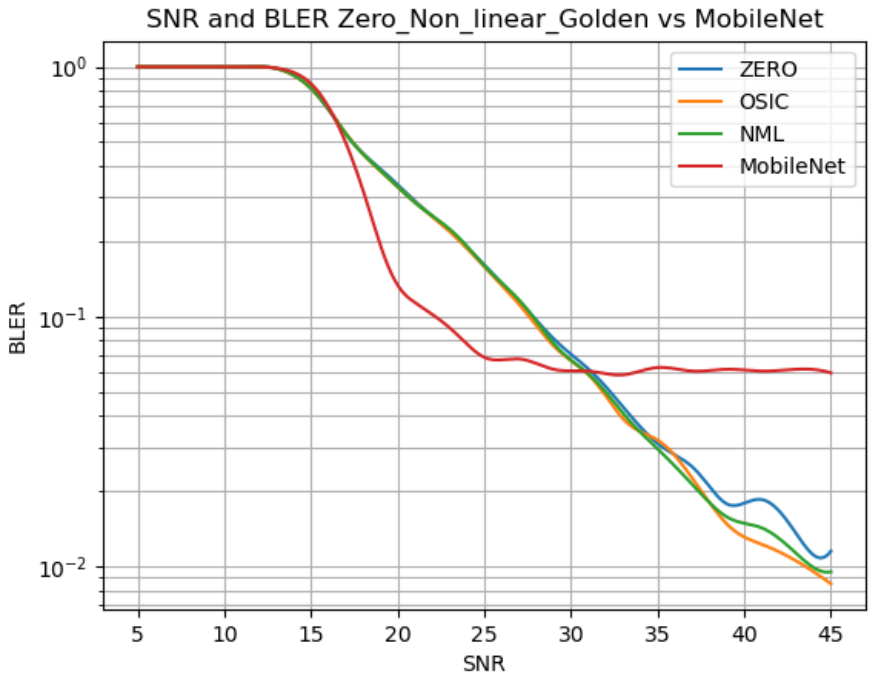

Figure 83: 16-QAM BLER: MobileNet vs. Zero forcing, OSIC and NML

4.2.5 GridNet Square

We experimented with different grid steps for GridNet Square and found 1/7 to be the best, shown in figure 61. More cells provide higher resolution but lower noise tolerance, leading to overfitting. Larger steps group noisy points in the same or nearby cells, beneficial in noisy scenarios. Limited cells require small sizes to avoid overlap in QAM constelation. In general, the BER results for the tested method were significantly worse than the golden model LMMSE, showing a difference of one order of magnitude.

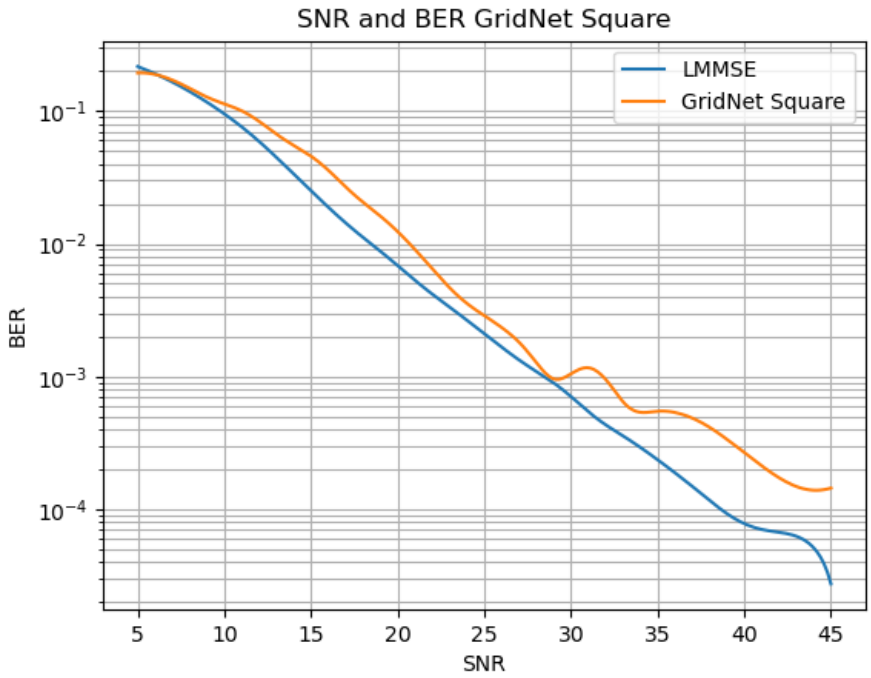

Figure 84: 16-QAM BER: GridNet Square Grid vs LMMSE

The Blocks error metrics are not performing as expected at high SNR levels, as the model truncates at 35dB, and cannot utilize additional information provided by the grid. One of the main reasons for the poor performance is that all points in the grid have equal weighting and relevance, leading to position uncertainty. To address this issue, the polarGrid was developed, where points near the center are closer together and weighted differently from border points. This design reduces uncertainty and provides a more accurate position perspective, as seen in Figure 64.

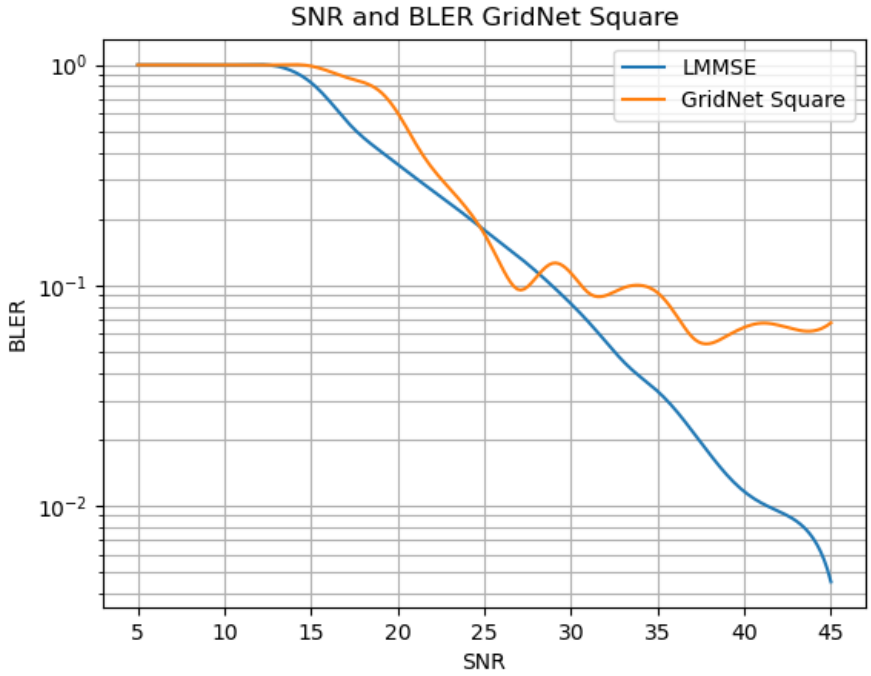

Figure 85: 16-QAM BLER: GridNet Square Grid vs. LMMSE

Regarding the non-linear models, all the results are parallel to each other. The performance of GridNet Square Grid can be considered an intermediate trade-off between OSIC and NML in terms of BER/SNR performance. Compared to NML, GridNet Square offers the main advantage of faster performance and the possibility of parallel processing in some sections. These features make it a good candidate for practical applications.

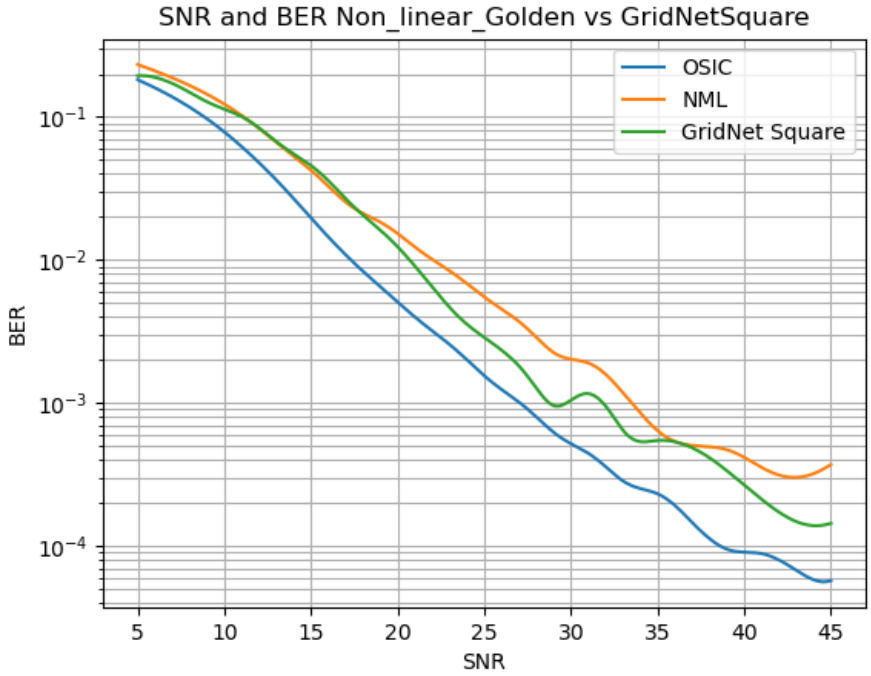

Figure 86: 16-QAM BER: GridNet Square Grid MobileNet vs.OSIC and NML

Overall, the non-linear methods outperform GridNet Square by one order of magnitude in terms of block error rate (BLER). However, in terms of bit error rate (BER), the degridding process used in GridNet Square narrows the noise by converting the grid patch to a complex point in the constellation and placing the calculated value in a fixed position based on the grid. This results in a slightly lower BER due to the reduced and discrete possible values. However, it is important to note that this process does not guarantee the integrity of the blocks and reduce block presicion.

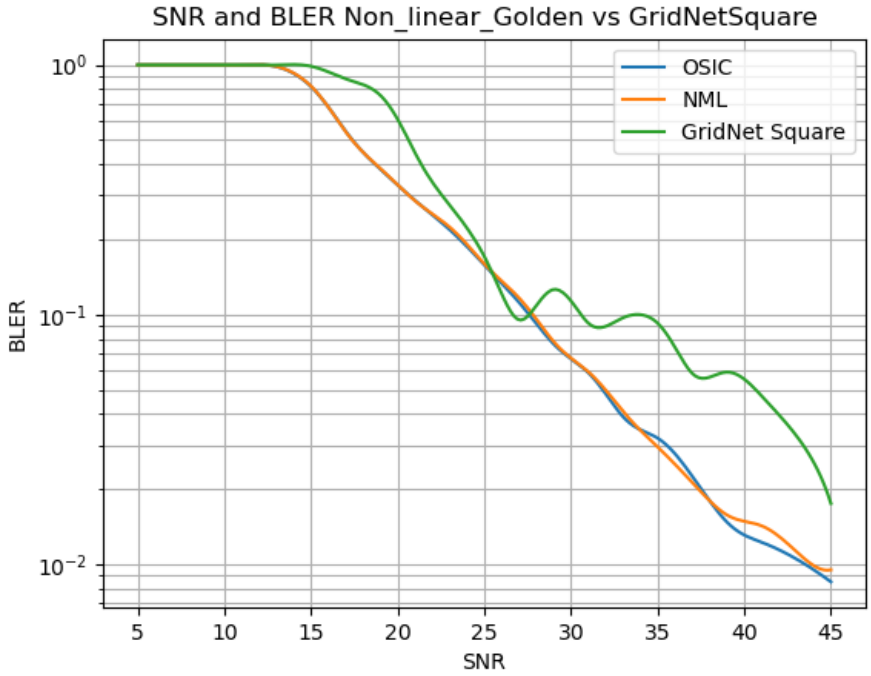

Figure 87: 16-QAM BLER: GridNet Square Grid MobileNet vs.OSIC and NML

4.2.6 GridNet Polar

GridNet with a polar grid achieved the best performance between 25 dB and 45 dB, demonstrating that the attention mechanisms with varying cell sizes weighted more towards the center effectively reduced inter-symbol interference (ISI). However, below 25 dB, the model struggled to handle the noise and performed similarly to LMMSE. We trained the model with noise levels ranging from 25 dB to 45 dB, and the image below supports our hypothesis that the optimal equalization occurs within this range. Although providing mores noise ranges in the trainning may appear to improve performance, it would cause the network to learn more about noise than relevant data, resulting in worse performance. The network's best BER was observed in the scenario with the highest noise level of 5 dB. This result is particularly noteworthy since there were enough BER to support a statistically robust analysis, providing confidence that the result was not an artifact.

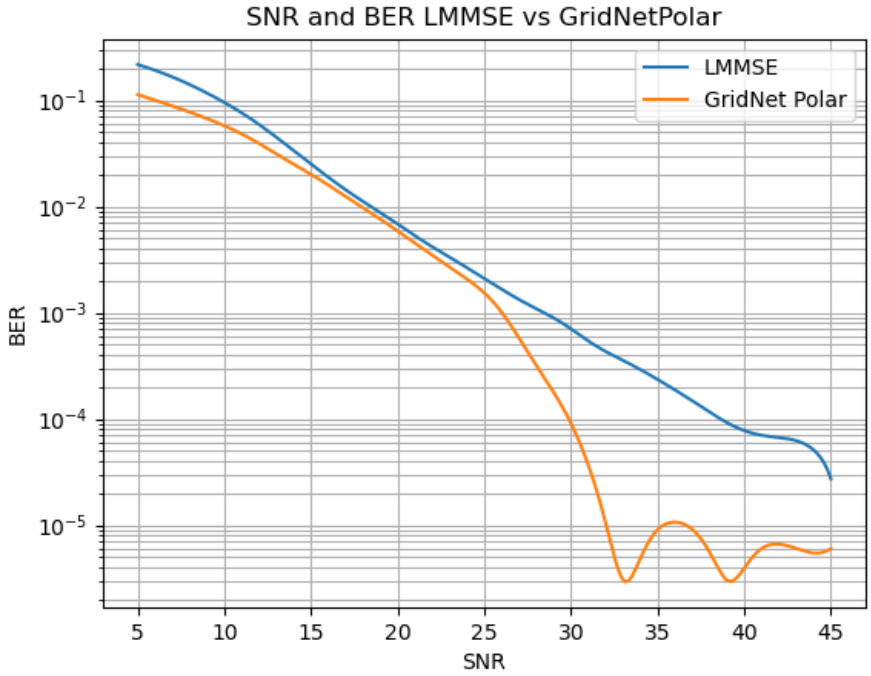

Figure 88: 16-QAM BER: GridNet Polar Grid vs LMMSE

Polar Grid exhibits one of the lowest block error rates (BLER) for high signal-to-noise ratio (SNR) values between 33 dB and 45 dB. The autoregression attention mechanisms employed by the transformer model are effective in capturing the relationship between the first and last values, which reduces inter-symbol interference (ISI) and leads to better performance. However, at lower SNR values, the noise starts to impact the network's attention mechanisms, causing the BLER to increase exponentially over testing data outside of the noise levels used in training.

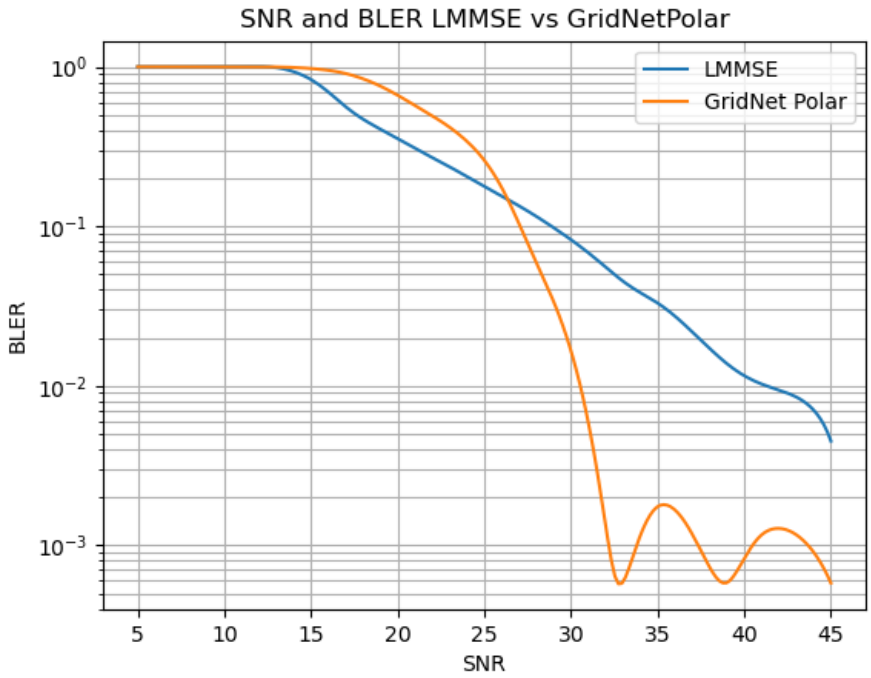

Figure 89: 16-QAM BLER: GridNet Polar Grid vs LMMSE

OSIC and PolarGrid operate similarly, utilizing attention mechanisms and grids to achieve superior performance at high signal-to-noise ratio (SNR) values. Comparing the non-linear models, it can be seen that OSIC yields the same bit error rate (BER) in certain low SNR ranges. However, for the lowest SNR, GridNet outperforms OSIC.

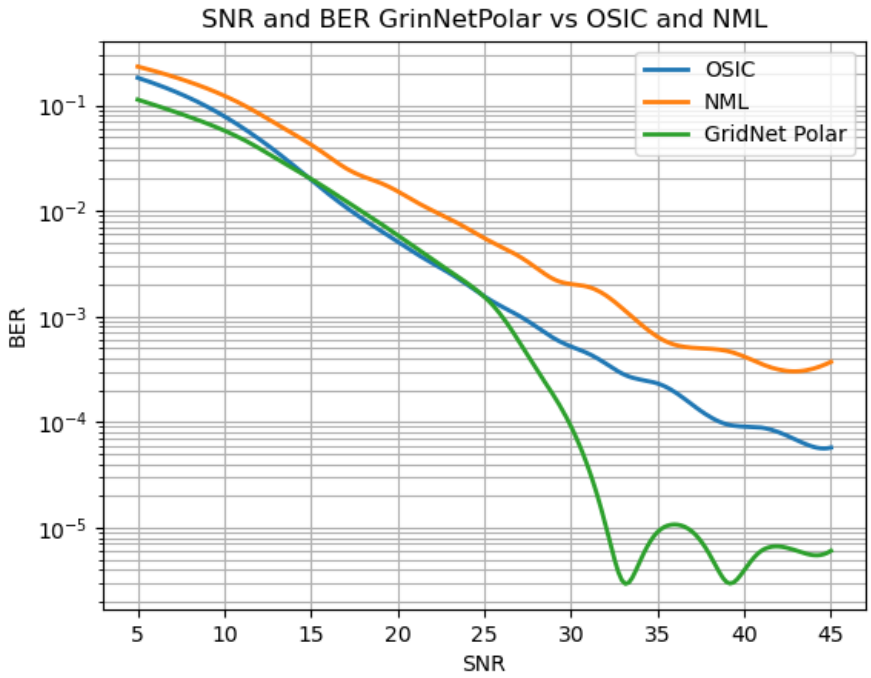

Figure 90: 16-QAM BER: GridNet Polar Grid MobileNet vs.OSIC and NML

The block error rate (BLER) graph shows that the models perform better at higher signal-to-noise ratio (SNR) values above 25 dB. However, for lower SNR values, both non-linear methods perform better. Nonetheless, GridNet is faster than NML and represents a promising implementation option with a small performance sacrifice for low SNR scenarios.

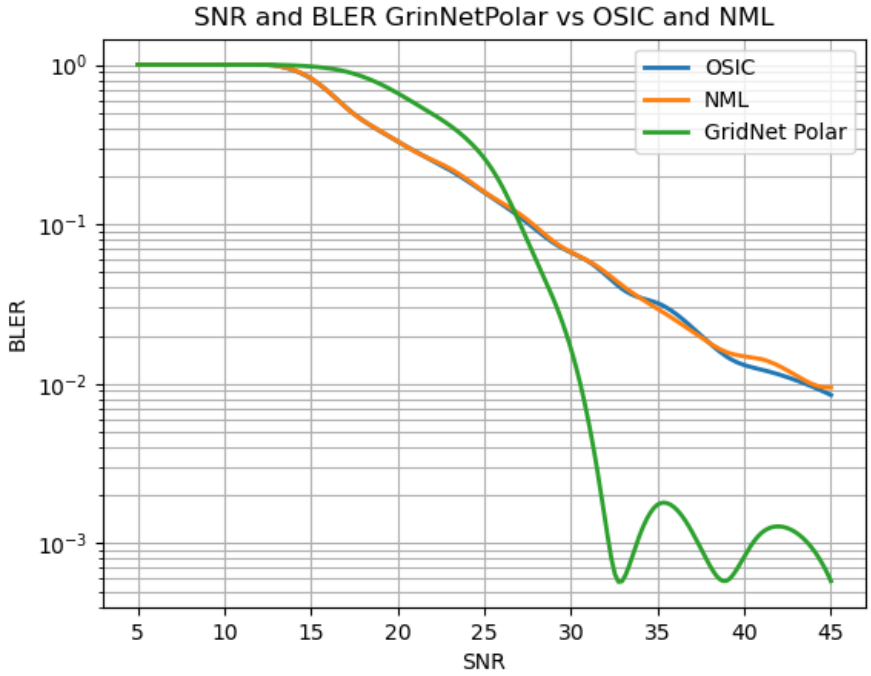

Figure 91: 16-QAM BLER: GridNet Polar Grid MobileNet vs.OSIC and NML

4.2.7 GridNet Both

PolarGrid has the potential to use fewer parameters because the number of patches reduces the alphabet size employed by the transformer, resulting in a smaller data model. While there is at least one order of magnitude difference between PolarGrid and SquareGrid in terms of bit error rate (BER), SquareGrid exhibits more consistent and predictable BER/SNR slope, whereas PolarNet shows rapid BER growth over a short range of SNR values, which can sometimes be undesirable, particularly if the communication system operates within this range. On a positive note, neither of the grids performs worse than NML, which has the highest BER.

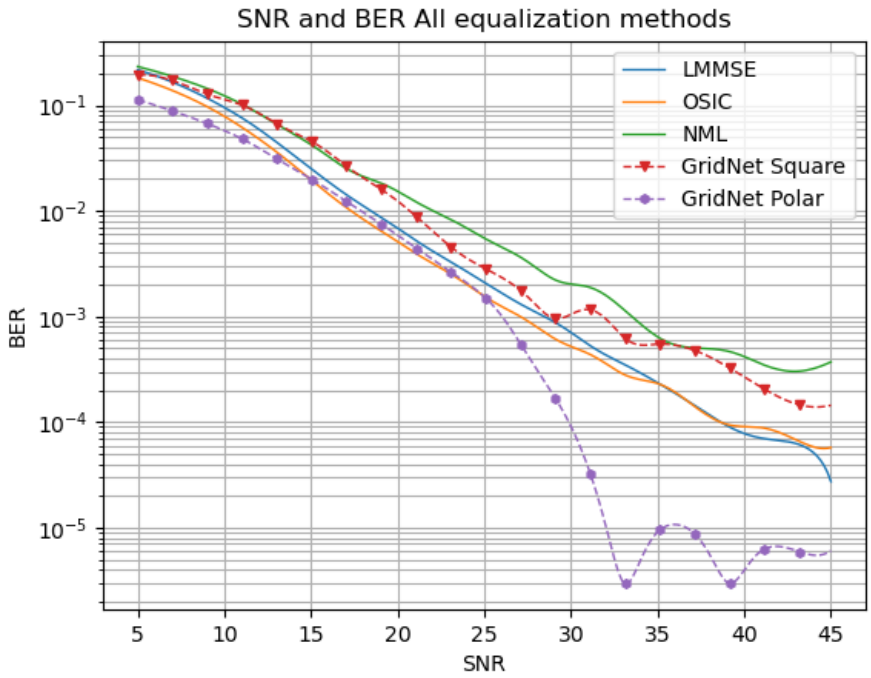

Figure 92: 16-QAM BER: GridNet, Polar and Square Grid vs LMMSE

In the Figure 93 , it can be appreciated that SquareGrid behaves more similarly to NML for lower SNR values, while GridNetPolar works similarly to OSIC, with an improvement for SNR values lower than 15 dB.

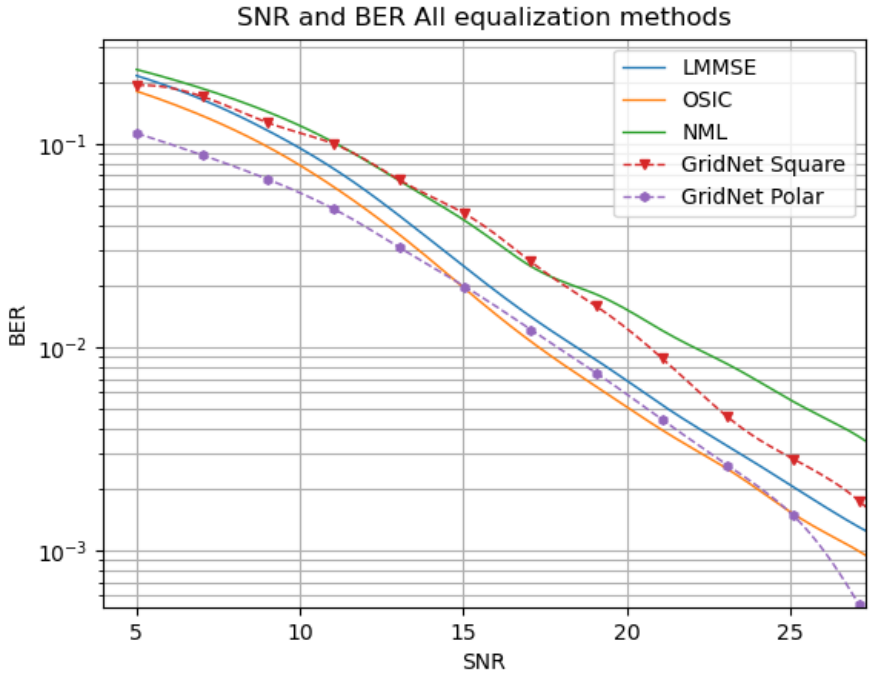

Figure 93: 16-QAM BER: GridNet, Polar and Square Grid vs LMMSE from 5dB to 27dB

Regarding the block error rate (BLER), both methods perform well for signal-to-noise ratio (SNR) values higher than 25 dB. However, SquareGrid exhibits a slower growth rate than PolarNet between 27 dB and 20 dB, and for lower SNR values, other non-linear methods achieve lower BLER values. One of the main drawbacks of these iterative methods is that they are less easily parallelizable compared to transformers, making the use of transformers a viable option despite the trade-off in BLER performance.

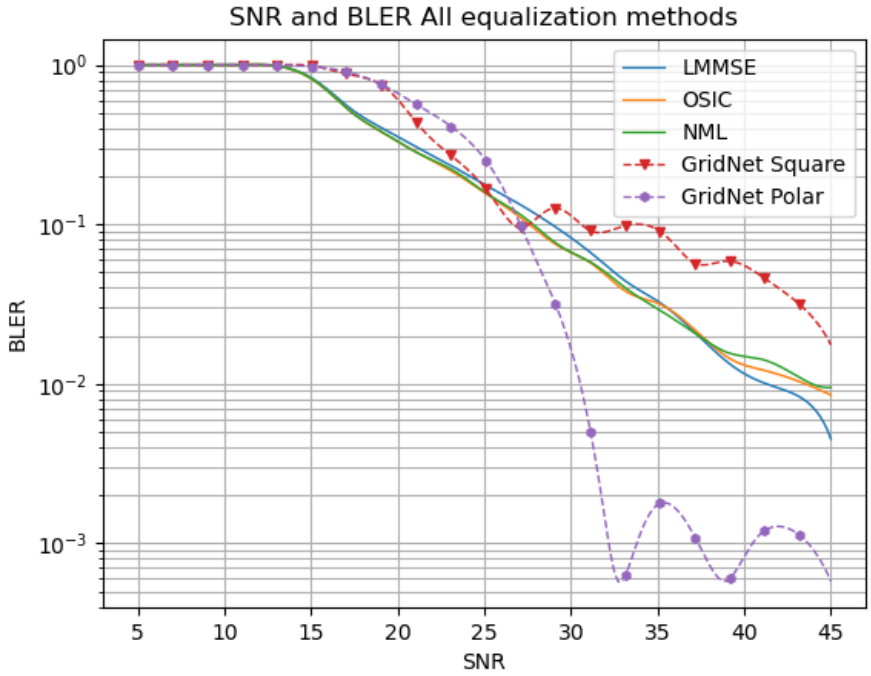

Figure 94: 16-QAM BLER: GridNet, Polar and Square Grid vs LMMSE

Square grid works better with lower SNR than the Polar grid. Basically both grids are a finnaly trade off between consisten noise mitigation or ISI cancelation.

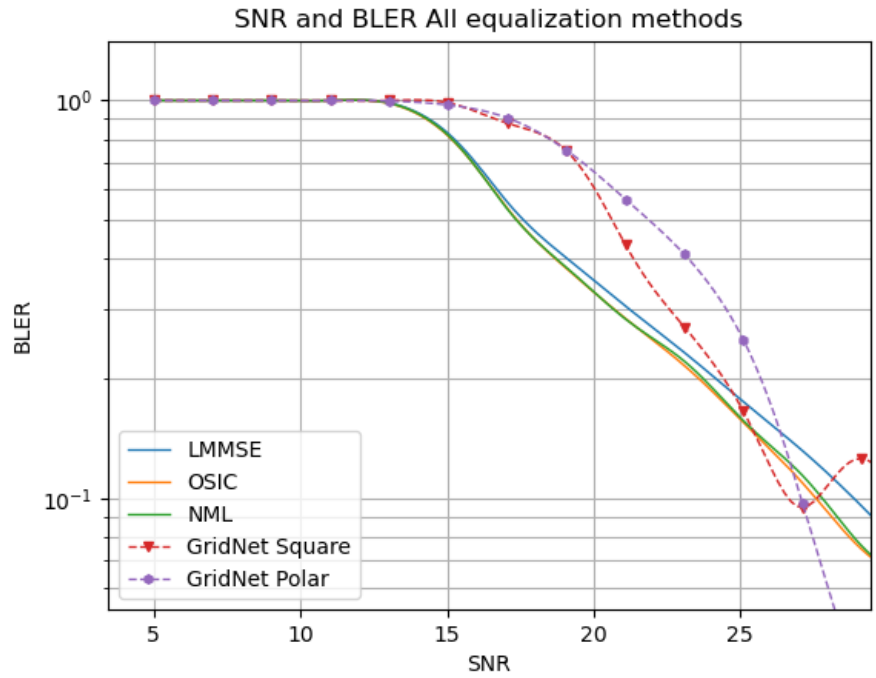

Figure 95: 16-QAM BLER: GridNet, Polar and Square Grid vs LMMSE from 5dB to 30dB

4.3 Outlook of results

The results presented here are based on the visual and experimental representation of the Block Error Rate (BER) and Bit Error Rate (BLER) with different Signal-to-Noise Ratio (SNR) scenarios. The results provide valuable insights into the strengths and weaknesses of each equalization method and can inform the development of more effective equalization techniques for equalization task.

- PolarNet appears to perform well overall SNR values, with one of the best performances in the range between 15 dB to 25 dB for reducing bit errors. Additionally, it exhibits a smooth increase in BER, making its behavior well predictable, also has is an intermidiate term of complexity.
- MobileNet performs well for SNR values lower than 25 dB, with lower BER than all golden models and some networks such as GridNet Square. Additionally, it strikes a good balance between the number of parameters and time complexity, making it a good candidate for embedded applications.
- Complex is slightly better than NML and GridNetSquare for SNR values lower than 18 dB.
- GridNet Polar has the lower BER values from 27dB to 45dB

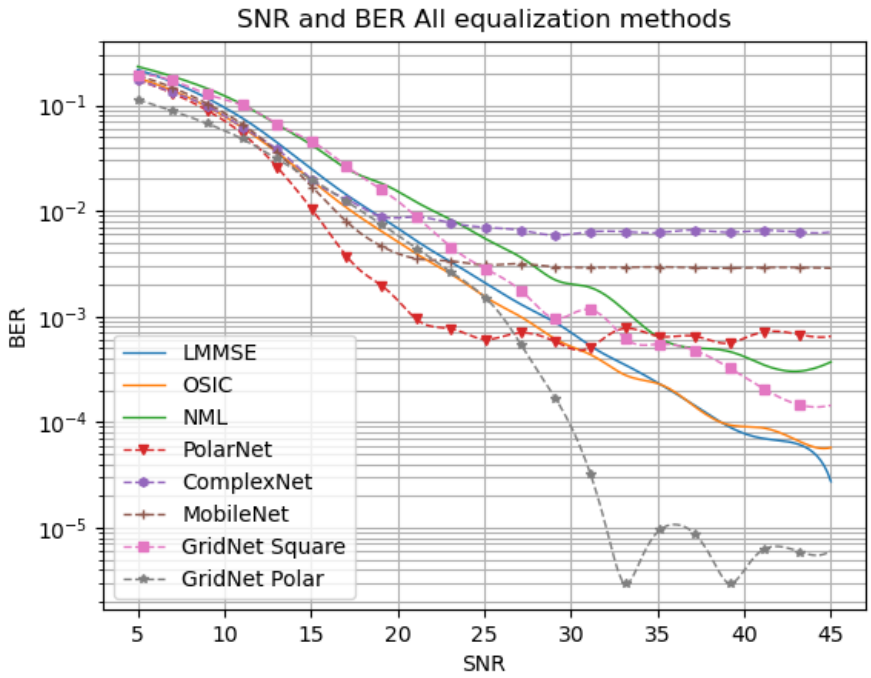

Figure 96: 16-QAM BER: All methods

- GridNet Square behaves similar to NML
- MobileNet, PolarNet and ComplexNet converge to the NML model for the lowest SNR values, from 5dB to 15dB

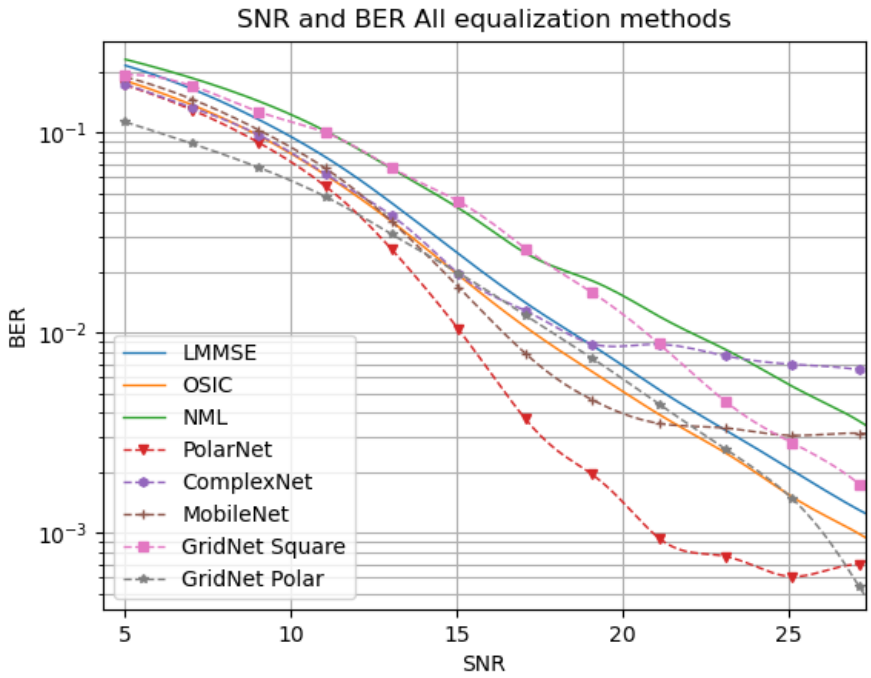

Figure 97: 16-QAM BER: All methods from 5dB to 27dB

- GridNetPolar performs well at high SNR, it degrades faster from 35dB to lower values, resulting in the worst BLER below 25dB. This means that while PolarGrid reduces bit errors, it consistently fails over the blocks.
- PolarNet outperforms all methods for SNR values lower than 30 dB, while also exhibiting the same growth relation with its BER metric. This mean a good outlier and noise handling for this Network.
- MobileNet shows promising results as an upgrade to the standard zero-forcing equalizer, handling with noise and ISI. This can be observed as it displays the second-best BLER plot among all the models.
- MobileNet exhibits a very close BER performance to PolarNet.

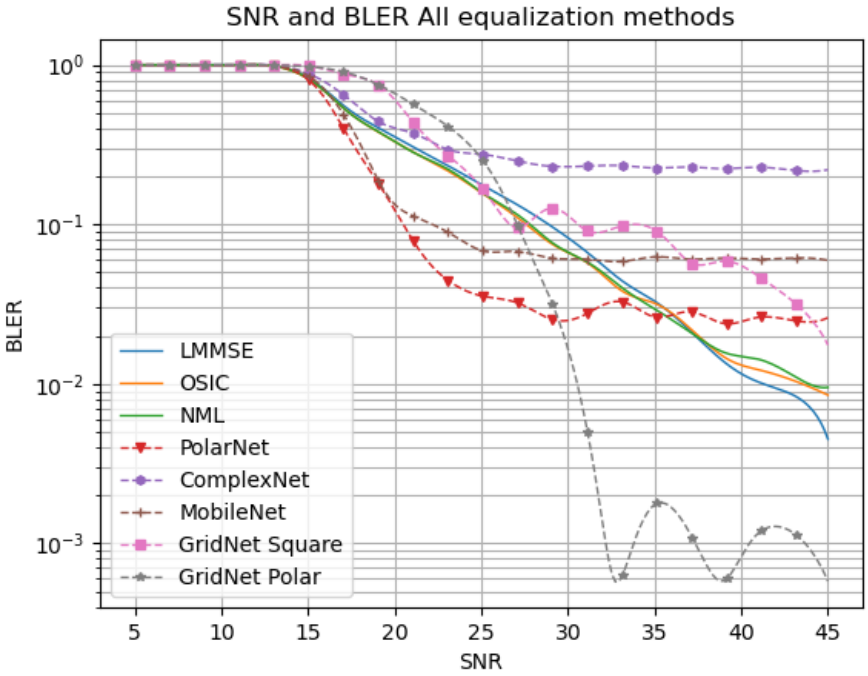

Figure 98: 16-QAM BLER: All methods from 5dB to 45dB

- Both GridNet are the first networks to reach the block error rate of 1.
- ComplexNet has better BLER than GridNet from 22dB to lower.

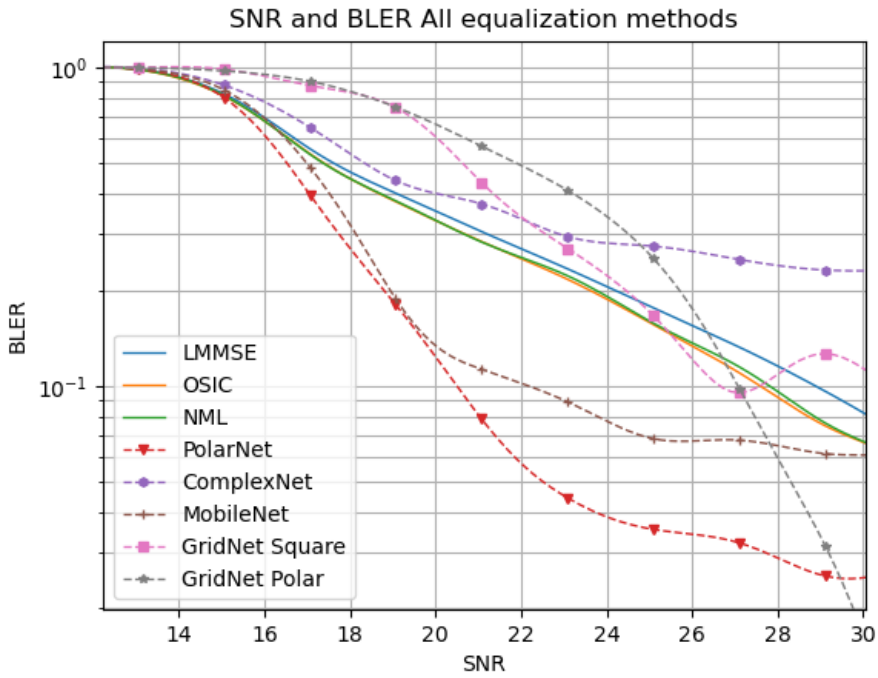

Figure 99: 16-QAM BLER: All methods from 14dB to 30dB

5 Conclusion

5.1 Contributions

- Compared to the state of art that use neural networks for signal equalization, we employed larger frame sizes and higher constellation points, such as 16QAM. Typically, smaller frame sizes and lower constellation points, such as QPSK, are used:
  - Larger frame sizes of 48
  - A channel of 48x48
  - Support up to 16QAM.
- In order to help the neural network generalize better during the training stage, the z-score was utilized to reduce outliers. This approach was particularly necessary for the equalization task, as it prevented the neural network from having to adapt to all possible solutions and improved its ability to generalize effectively.
- **PhaseNet**:
  - Designed for PSK systems equalization.
  - Generated the idea to use a unitary circle to exploit the concept of using distance as a metric of the phase, setting a constant radius of one but measuring the error distance based on how well the angle is estimated by the network. This avoids wrong angle measurements, which would result in an incorrect error being passed to the network.
  - Network learns faster and generalizes better for phase prediction with the distance error metric.
- **PolarNet**:
  - A combination of PhaseNet and MagNet, which is an equalizer that separately deals with phase and magnitude error reduction and estimates them as separate values in a phasor manner. The idea behind this network is to make it more modular, as a system can work with either PSK equalization or QAM, depending on the requirement of the system.

  - Combination of different preprocesing stages on for magnitud and another for phase.

- **ComplexNet**:

  - First Implementation of complex of a feedforward neural network for OFDM equalization.
  - Although it did not produce the best results for equalization, it opens new avenues for possible implementation and mathematical study to upgrade gradients and network evaluation, to make a better estimate of the ground truth.

- **MobileNet**:

  - A network typically used for classification modified for our research, it was utilized for a regression task to simulate zero-forcing with known noise.
  - Modification of the first and final layer to ensure the network fits with the data dimensionality.
  - Using one of the most efficient convolutional neural networks for an equalization task, striking a balance between accuracy and time complexity
  - The network significantly reduced the BER and BLER with just one single vector of size 48 performing zero-forcing.

- **Identified a gap** in the existing literature regarding the use of transformers in the signal processing scenario.

  - Taking inspiration from image-to-image translation models, we implemented a square grid that divides the IQ plane into buckets. These buckets are a quantization of all infinite possible values in the plane into a reduced alphabet.
  - Translated the uncorrected image given by the data, which had channel deformation and noise, and recovered the image with the desired buckets based on the ground truth.

- **PolarGrid** preprocessing stage in GridNet:

  - Although existing square grids are commonly used for image processing in image transformers, the use of PolarGrid was unexplored before this research. Which is quite useful in this case, as our image represents the complex plane that can be analyzed in polar geometry.

- **GridNet**:

  - The first transformer applied to signal equalization in the modulation scheme for OFDM.

5.2 Outlook

In this work, new networks for equalizing variant channels were presented. The proposed networks met the objectives by presenting the following advantages:

- MobileNet and PolarGrid enable adequate mitigation of impairments caused by ISI and doppler effects.
- PhaseNet, PolarNet, and ComplexNet allow for adequate noise reduction.
- The networks time complexity can be parallelized without implementing complex architectures.
- Except for ComplexNet, the networks outperform the non-linear classical methods OSCI and NML for lower SNR values.
- PolarGridNet is proof that transformers can be used in the telecom industry with adequate preprocessing.

The following observations were made regarding the work presented in this study:

- Neural networks work better with closely spaced data points, hence a pre-processing stage plays a fundamental role.
- Attention mechanisms assign weights to input data, enabling the model to focus on relevant information and reduce inter-symbol interference (ISI). However, in the presence of noise, this can cause attention mechanisms to fail in assigning correct weights, leading to a focus on irrelevant features and increased block error rate (BLER).

- Variable SNR values work better for GridNet training, while constant SNR works better for the other models.
- Training with low SNR values can result in the network learning noise patterns instead of accurately representing the signal.

  - Our experiments showed that a minimum SNR value of 20dB is required for training.

- All networks should be implemented in Python due to the GPU and TPU APIs support, which accelerates the training process.
- The use of a larger dataset is recommended to increase the robustness of the network testing.

In conclusion, the field of neural networks has seen tremendous advancements in recent years, with the development of more complex architectures, such as deep neural networks, convolutional neural networks, and transformers. It is essential to strike a balance between accuracy and memory footprint for signal equalization to avoid affecting data throughput. These advancements have led to significant improvements in the ability of neural networks to process complex data, including doubly dispersive channels. These developments have resulted in numerous practical applications, ranging from self-driving cars and drone flying to the development of new 6G technologies. However, despite the remarkable progress made in the field, there are still many challenges that need to be addressed, such as reducing noise overfitting, managing outliers, MIMO signal cleaning, and the need for more formal mathematical descriptions of some network architectures. Further research is necessary to continue to advance the field and unlock the full potential of neural networks in the telecom area, which is an area that is just starting to show its potential.

5.3 Future Work

1. Model quantization is a technique for reducing the memory footprint and computational cost of deep neural networks by converting the weights and activations of the network to a lower numerical precision format. While it can significantly reduce the memory and power requirements of neural networks, it can also lead to a loss of accuracy. Adjustments to the network architecture and training process are needed to ensure that the quantized network still achieves good performance. Google's Coral platform is an example of a system that uses model quantization for efficient inference,

including a set of tools for quantizing and compressing neural network models to run on the Coral Edge TPU. Coral's quantization tools include both post-training and quantization-aware training methods, and are compatible with popular deep learning frameworks such as TensorFlow and PyTorch. [18]

2. One potential future work is to use the latest reinforcement learning algorithms to discover a matrix inversion algorithm. AlphaTensor is a well-known network that discovers many provably correct matrix multiplication algorithms that improve over existing algorithms in terms of the number of scalar multiplications, and is adaptable to different use-cases, including discovering algorithms for structured matrix multiplication and optimizing for actual runtime. The results demonstrate that the space of matrix multiplication algorithms is richer than previously thought, and AlphaTensor can efficiently search this space to discover novel algorithms that outperform human-designed ones on the same hardware. Therefore, a possible future direction could be to create an algorithm that outperforms existing ones and achieves a time complexity of almost $O(N^2)$ for matrix inversion.[15]

3. Equalization in MIMO scenarios could be a potential candidate, as it involves classification tasks that are typically more effective with neural networks.[52]

6 References

References

[1] 3rd Generation Partnership Project (3GPP). 3gpp ts 36.201 version 12.2.0. https://www.etsi.org/deliver/etsi_ts/136200_136299/136201/12.02.00_60/ts_136201v120200p.pdf, 2010.

[2] Hamid Aghvami and Pheng-Ann Heng. *Channel Modeling for Wireless Communication Systems*. John Wiley Sons, 2005.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473, 2014.

[5] Constantine A. Balanis. *Advanced Engineering Electromagnetics*. John Wiley amp Sons, 2012.

[6] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27, pages 37–49. PMLR, 2012.

[7] Joshua Bassey, Lijun Qian, and Xianfang Li. A survey of complex-valued neural networks. https://arxiv.org/abs/2101.12249, 2021.

[8] Joshua Bassey, Lijun Qian, and Xianfang Li. A survey of complex-valued neural networks. 01 2021.

[9] Joshua Bassey, Lijun Qian, and Xianfang Li. A survey of complex-valued neural networks. https://arxiv.org/abs/2101.12249, 2021.

[10] Ekaba Bisong. *Regularization for Deep Learning*, pages 415–421. Apress, Berkeley, CA, 2019.

[11] CableLabs. DOCSIS 4.0 specification. https://www.cablelabs.com/technologies/docsis-4-0-technology, 2020. Accessed: March 14, 2023.

[12] J.A. Del Puerto-Flores, R. Parra-Michel, F. Pena-Campos, Joaquin Cortez, and E. Romero-Aguirre. Evaluation of ofdm systems with virtual carriers over v2v channels. In *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 882–886, 2018.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[14] Nariman Farsad, Weisi Guo, and Andrew Eckford. Tabletop molecular communication: Text messages through chemical signals. *PloS one*, 8:e82935, 12 2013.

[15] Huang Fawzi. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022.

[16] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[18] Google. Accelerator module. `https://coral.ai/products/accelerator-module/#documentation`, 2020. Accessed: March 6, 2023.

[19] Gonzalo Gutierrez Ramos. Analisis de punto fijo y arquitectura digital de modulo de descomposicion qr para receptores de comunicacion v2v. Master's thesis, Centro de Investigacion y de Estudios Avanzados Unidad Guadalajara, 2019.

[20] Sepp Hochreiter and JÃŒrgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[21] Hiroto Honda. An unofficial colab walkthrough of vision transformer. `https://medium.com/@hirotoschwert/an-unofficial-colab-walkthrough-of-vision-transformer-8bcd592ba26a`, December 2020.

[22] Yi Hong, Tharaj Thaj, and Emanuele Viterbo. *Delay-Doppler Communications: Principles and applications*. Academic Press, an imprint of Elsevier, 2022.

[23] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. `https://arxiv.org/abs/1905.02244`, 2019.

[25] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. `https://arxiv.org/abs/1704.04861`, 2017.

[26] Qisheng Huang, Chunming Zhao, Ming Jiang, Xiaomin Li, and Jing Liang. A novel ofdm equalizer for large doppler shift channel through deep learning. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pages 1–5, 2019.

[27] IEEE. Ieee standard for information technology telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 5: Enhancements for next generation v2x. *IEEE Std 802.11bd-2022 (Amendment to IEEE Std 802.11-2020 as amended by IEEE Std 802.11ax-2021, IEEE Std 802.11ay-2021, IEEE Std 802.11ba-2021, IEEE Std 802.11-2020/Cor 1?2022, and IEEE Std 802.11az-2022)*, pages 1–144, 2023.

[28] Alan Imgur, https://i.stack.imgur.com/U2mxR.jpg.

[29] Steven M. Kay. *Fundamentals of Statistical Signal Processing: Estimation Theory*. Prentice Hall, 1993.

[30] C. Komninakis and R.D. Wesel. Pilot-aided joint data and channel estimation in flat correlated fading. In *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99. (Cat. No.99CH37042)*, volume 5, pages 2534–2539 vol.5, 1999.

[31] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[32] Khaled B. Letaief, Wei Chen, Yuanming Shi, Jun Zhang, and Ying-Jun Angela Zhang. The roadmap to 6g: Ai empowered wireless networks. *IEEE Communications Magazine*, 57(8):84–90, 2019.

[33] Guo-Wei Lu. Optical signal processing for high-order quadrature- amplitude modulation formats. In Sudhakar Radhakrishnan, editor, *Applications of Digital Signal Processing through Practical Approach*, chapter 1. IntechOpen, Rijeka, 2015.

[34] Danilo P Mandic. Complex valued recurrent neural networks for non-circular complex signals. In *2009 International Joint Conference on Neural Networks*, pages 1987–1992. IEEE, June 2009.

[35] Ian Marsland, Calvin Plett, and John Rogers. *Radio Frequency System Architecture and Design*. 2013.

[36] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[37] Arun Mohan. An overview on regularization, Jan 2019.

[38] Nor Noordin, Borhanuddin Ali, S.s Jamuar, Tharek Abd Rahman, and Mahamod Ismail. Adaptive spatial mode of space?time and spacefrequency ofdm system over fading channels. *Jurnal Teknologi*, 45:59–76, 12 2006.

[39] Timothy O Shea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):563–575, 2017.

[40] Institute of Electrical and Electronics Engineers (IEEE). Ieee std 802.11-2016 (revision of ieee std 802.11-2012). `https://ieeexplore.ieee.org/document/7786995`, 2016.

[41] Timothy J. OShea, Latha Pemula, Dhruv Batra, and T. Charles Clancy. Radio transformer networks: Attention models for learning to synchronize in wireless systems. In *2016 50th Asilomar Conference on Signals, Systems and Computers*, pages 662–666, 2016.

[42] Athanasios Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill Education, 2002.

[43] W. W. Peterson and D. T. Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, 1961.

[44] Lukas Picek, Milan Sulc, Jiri Matas, Jacob Heilmann-Clausen, Thomas Jeppesen, and Emil Lind. Automatic fungi recognition: Deep learning meets mycology. *Sensors*, 22:633, 01 2022.

[45] John G. Proakis. *Communication Systems Engineering*. Prentice Hall, 2002.

[46] PyTorch Lightning. GPU Training (Basic) - PyTorch Lightning. `https://pytorch-lightning.readthedocs.io/en/stable/accelerators/gpu_basic.html`, 2023. Accessed: March 4, 2023.

[47] Dario Radeci. Softmax activation function explained. `https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60`, Mar 2022.

[48] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 2002.

[49] P. Raviteja, Khoa T. Phan, and Yi Hong. Embedded pilotaided channel estimation for otfs in delaydoppler channels. *IEEE Transactions on Vehicular Technology*, 68(5):4906–4917, 2019.

[50] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer New York, 2004.

[51] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. *Parallel Distributed Processing, Vol. 1*, pages 318–362, 1985.

[52] Nithin Samuel, Tzvi Diskin, and Ami Wiesel. Deep mimo detection. *arXiv preprint arXiv:1706.0115*, 2018.

[53] M. Shanker, M.Y. Hu, and M.S. Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996.

[54] VK Sharma, MK Singh, and RK Jain. Activation functions in artificial neural networks: a review. *Neural Networks*, 61:87–117, 2014.

[55] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, 2018.

[56] Alexander Shvets. *Dive into Design Patterns*. 2021.

[57] Praphul Singh. Multi-head self-attention in nlp. Blog post, May 2020.

[58] M. D. Srinath, P. K. Rajasekaran, and R. Viswanathan. *Introduction to Statistical Signal Processing with Applications*. Prentice-Hall, Inc., USA, 1995.

[59] Gilbert Strang. *Linear Algebra and Its Applications*. Wellesley-Cambridge Press, 2019.

[60] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[61] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[62] PL team. PyTorch Lightning MNIST TPU Training Notebook. `https://pytorch-lightning.readthedocs.io/en/stable/notebooks/lightning_examples/mnist-tpu-training.html`, 2023. Accessed: March 4, 2023.

[63] Zhiqiang Teng, Shuai Teng, Jiqiao Zhang, Gongfa Chen, and Fangsen Cui. Structural damage detection based on real-time vibration signal and convolutional neural network. *Applied Sciences*, 10:4720, 07 2020.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, and Kaiser. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[65] Mathuranathan Viswanathan. *Digital Modulations using Python*. Self-published, 2019. PDF.

[66] Wikipedia. 16qam. `https://uk.wikipedia.org/wiki/16QAM`, 2023. Accessed: March 15, 2023.

[67] K. K. Wong and R. S. Cheng. *Orthogonal Frequency Division Multiplexing for Wireless Communications: Principles and Practice*. CRC Press, 2017.

[68] Muhammad Zahid and Zhang Meng. Recent advances in neural network techniques for channel equalization: A comprehensive survey. In *2018 International Conference on Computing, Electronics and Communications Engineering (iCCECE)*, pages 178–182, 2018.