# Testing of AI models for Air-Interface Applications

- Masters Thesis -

Project Report

Filip Ivanović

Aalborg University
Electronics and IT

**Electronics and IT**
Aalborg University
http://www.aau.dk

**AALBORG UNIVERSITY**
STUDENT REPORT

**Title:**
Testing of AI models for Air-Interface Applications

**Theme:**
Signal Processing

**Project Period:**
Spring Semester 2023

**Project Group:**
976

**Participant(s):**
Filip Ivanović

**Supervisor(s):**
Carles Navarro Manchón
Alan Anderson

**Copies:** 1

**Page Numbers:** 108

**Date of Completion:**
May 30, 2023

**Abstract:**

Artificial Intelligence (AI) is a rapidly developing field of technology being implemented in many contexts, including in air interface communications. Alongside the development of AI comes the development of systems designed to test AI, ensuring its performance and understanding its characteristics. This project, done in collaboration with Keysight Technologies and using their xpl[AI]ned framework, stands as a study into the usability and relevance of a few different testing methods on AI models designed for the air interface communication space. This is done through a process of researching various different aspects of AI and testing methods and applying them to two AI models that are chosen to be representative of potential real world implementations. The two main testing methods applied are Monte Carlo dropout and the Fast Gradient Sign Method (FGSM) for testing adversarial robustness. Monte Carlo dropout is found to be useful in quantifying the efficiency of the construction of the models, but due to the nature of the context its quantification of certainty is not found to be useful. FGSM is found to be extremely useful with its capability to show if a model is vulnerable to adversarial perturbations, as well as generating adversarial data that can be analysed to further determine model characteristics. A few other methods and testing paths were also looked into to gain further clarity. The overall result of this investigation is a resounding success for the xpl[AI]ned concept as well as a greater understanding of the types of methods relevant when testing air interface AI models.

# Contents

# Acronyms

$E_b/N_0$ Energy per Bit to Noise Power Spectral Density Ratio. iii, viii–xii, 23, 26–30, 33, 34, 36–40, 42–44, 47–52, 54, 57–60, 62, 64, 71

**AI** Artificial Intelligence. i, 2, 3, 11, 17, 18, 25, 42, 68–70, 72–75

**BCE** Binary Cross Entropy. 14, 21, 24

**BER** Bit Error Rate. viii–xiii, 11, 26–29, 34, 37–50, 53–55, 60–65, 68, 71, 83–87, 98, 99

**BLER** Block Error Rate. 26, 71

**BMD** Bit-Metric Decoding rate. 21

**CEM** Contrastive Explanation Method. iii, 35, 61, 62, 70

**CNN** Convolutional Neural Network. 5, 8, 16

**DL** Deep Learning. 4

**FGSM** Fast Gradient Sign Method. i, iii, x–xii, 7, 32–34, 46–52, 55–60, 69–71, 74, 104–108

**LDPC** Low Density Parity-Check Code. 21, 22, 26

**LLR** Log Likelihood Ratio. x, xi, 21, 22, 24, 26, 29, 34, 46, 49, 55

**LS** Least Squares. 19, 20

**MAE** Mean Absolute Error. 14

**ME** Mean Error. 14

**ML** Machine Learning. 3, 9, 71, 76

**MSE** Mean Squared Error. 14

**NN** Neural Network. 6, 11

**OFDM** Orthogonal Frequency-Division Multiplexing. 5, 20, 21, 51

**PN** Pertinent Negatives. 35

**PP** Pertinent Positives. 35

**QAM** Quadrature Amplitude Modulation. 25, 61

**QPSK** Quadrature Phase Shift Keying. 19, 25, 53

**ReLU** Rectified Linear Unit. 13, 24

**RL** Reinforcement Learning. xii, xiii, 24, 26, 64–67

**RMSE** Root Mean Squared Error. 14

**SGD** Stochastic Gradient Descent. 5, 15, 24

**SHAP** SHapley Additive exPlanations. iii, 34, 35, 61, 62, 70, 71, 75

**xpl[AI]ned** Keysight's new AI model testing and development software. iii, ix, xiv, 8–10, 17–19, 32, 33, 35–37, 46, 51, 61, 62, 68–70, 72, 74, 75

# List of Tables

# List of Figures

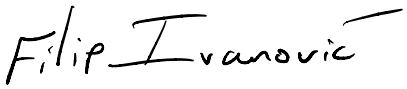
# Preface

The author would like to clarify that this project was done in collaboration with the xpl[AI]ned team at Keysight Technologies. The team brought expertise in the area of AI-related programming, while the author brought experience with the air interface context and developing/programming relevant testing solutions. In order to develop the testing methods described within and adapt them for use to air interface AI models, the xpl[AI]ned team would develop the basics of the technical backend, and then any necessary work needed to mate the model to the testing methods was done in collaboration between them and the author. All front end work involving building testing solutions, data analysis and drawing of relevant conclusions was then done soley by the author. The author would like to thank the xpl[AI]ned team, those being Lukas Klose, Carl Schiller and Till Hajek, for their tireless work and assistance. This project would not have been possible without their expertise and experience.

The author would like to thank Dr Carles Navarro Manchon and Dr Alan Anderson for their insightful supervision and excellent advice and guidance throughout the course of the project. Dr. Manchon's guidance insured that the project was always heading in the correct direction with the correct motivation, and Dr. Anderson's technical experience provided a wealth of knowledge to aid in all conclusions drawn within.

The author would like to thank Alice Hundt, for her endless patience, assistance and reassurance throughout the entirety of the project cycle.

Aalborg University, May 30, 2023



Filip Ivanovic
<fivano21@student.aau.dk>

# Chapter 1

# Introduction

## 1.1 Introduction

As we move into the third decade of the 21st century, it is becoming clear that the defining technology of this decade will most likely be the advancement of Machine Learning systems, colloquially called 'AI'. This technology is permeating almost all areas of engineering and development, and this includes cellular communication technologies. The generations of technology in this field are often defined by one disruptive development, and AI is poised to be that for 6G. 6G is being built with AI technologies in mind, in terms of both using AI to design communication systems and making sure that 6G is compatible with large distributed learning systems.[19]

The advancement of this type of technology naturally comes with issues and growing pains as existing infrastructure has to adapt. One of the most disruptive parts of AI is the inherent lack of transparency in its operation. This means that while these machine learned systems may work well for their intended purpose, the exact nature of their operation and the factors that contribute to different aspects of their performance remain a mystery to the layman and severely obfuscated at the very least to engineers and technicians working in the field. This means that improvements, testing and validation operations can be difficult to perform without doing arduous processes such as retraining and brute force data validation. This is especially disruptive to companies like Keysight Technologies, who specialise in providing testing equipment and solutions; if you can't put in input $x$, observe operation $+ y$ and see result $x + y$, how can you make a reliable testing solution?

With this motivation in mind, Keysight has begun to undertake multiple projects in order to ensure that they aren't left behind when 6G arrives and by the advancement of AI enabled technologies in general. One of these projects involves investigations into more advanced and granular methods of examining AI models - how they work, which parts of them contribute the most to their performance, and how they can be adversely affected by

different external conditions and internal failures. This project has Keysight as a part of the European Union (EU) funded CENTRIC project, in which they are the main partner in charge of research into testing methods [7].

Within Keysight, tools are being developed which can implement new methods for testing AI. These tools comprise the standardised templates for the development of methods for use in investigating the robustness, accuracy and internal structure of AI models. So far, these have been used in image recognition fields, but work is beginning on distributing them within the company so that various departments can begin experimenting with them. This includes the Aalborg office, which will be one of the locations participating in the 6G research.

It is this office, in collaboration with the Edinburgh office, that put forward the proposal for this project, which is as follows: **To conduct an investigation into the possibilities for new methods of testing AI models related to the air interface through the use of these newly developed tools.** With the proposal being open ended by design, the project will take on a explorative nature, focusing on research and drawing conclusions from gathered data rather than the production of a specific testing system/method.

This report follows the exploration of this proposal from the initial background research stages, the clarification of the goals and requirements of the proposal, the work done in researching and integrating a set of testing methods, discussing the performance of said methods and the implications of their results, as well as the conclusions drawn from the research and development process.

## 1.2 Background

### 1.2.1 AI models

The development and use of AI models is currently one of the fastest growing fields in many different engineering spaces. Spurred on by advances in both hardware and software, engineers are finding new ways to improve how AI models work and to make their integrations with other systems more efficient. It is a complex field with many different intersecting definitions, so it is worthwhile to go over the most common ones and define what areas of study and development they cover.

At the highest level, the subset of data science that deals with the development of systems that can simulate human intelligence processes is referred to as Artificial Intelligence (AI). While to the layman this brings forth images of malevolent machines with red eyes bent on destruction, in reality AI development deals less with sentient computers and more with having computers work in a way that can emulate how a human being thinks

to aid them in solving issues that would be difficult through traditional programming. Some of the most basic versions of AI come in the form of knowledge bases, large sets of data that can be searched by a computer through the use of keywords to provide necessary information, for example in a customer support setting[13].

Going one level further in complexity, we get to Machine Learning (ML), a subset of AI that deals with developing machines that can learn from their experience and improve over time without the need to have all possible interactions programmed directly. All ML models are AI, but the opposite is not necessarily true. ML has two main methods of operation relating to the method by which it learns, those being supervised and unsupervised learning, although semi-supervised learning is also possible. In supervised learning, training data that has been manually labeled and classified is provided. The model learns from this data to be able to make predictions on new datasets. Unsupervised learning feeds the model unlabeled, raw data. The model is not told what to do and must learn how to classify the data from scratch by itself. This is extremely useful in situations where data may have an unidentified hidden structure that cannot be labeled by hand but can be discerned by the model through the learning process.

ML models are most commonly designed to solve two different types of task: Regression and Classification. In regression, the model is used to determine the relationship between some independent variable/feature and another dependant variable/outcome, like estimating a person's weight based on their height and various other factors. In signal processing, regression problems are called estimation problems. Classification models on the other hand assign labels and categories to test data, such as determining the colours present in a video feed. These are equivalent to detection problems in signal processing. In all forms, machine learning is highly dependant on the representation of the data that is passed to it; it is expecting the data it receives to be of a certain format and it needs it to be of that format for it to provide a suitable output[13].

However, sometimes data can be incredibly complex and the best way to format said data into a representation that can be used efficiently by the model is a task in itself. This leads to the concept of representation learning, a subset of ML where the expected format of the data itself is part of the training process for the model. This allows for the processing of complex data and adapting to changing situations with minimal human intervention. However, when data sets reach certain levels of complexity, even these models can struggle to put together a good representation of the data. A common example of this tends to be things that we as humans 'perceive', such as the presence of human speech. In these situations, a representation learning model may be no faster than human engineers identifying/designing complex features for the model to use[13].

In these situations where we have complex representations, we use a type of ML model

called a Deep Learning (DL) model. These models employ deep neural networks, that is to say neural networks with many layers, which contain many neurons that allow complex representations to be broken down into sets of interconnected simpler ones. This process of building complex systems out of simple concepts is what makes DL so powerful. The base data is presented to the model at the visible layer. The visible layer contains variables that are observable to the outside, as opposed to the following hidden layers which extract abstract features from the provided data. The greater the number of hidden layers, the more capable the model is at finding increasingly abstract and useful patterns within the data[13].

### 1.2.2 Use of AI in the Air Interface

Research into the use of AI models in air interface applications has increased in frequency in the recent years due to the increased capability and availability for technology to support it. Developments are being made to begin integrating AI into the current generation (5G), enhancing specific areas such as Channel state information, Beam management and Positioning [25]. All of these show a lot of promise in terms of performance gains for 5G. On the other hand, 6G is being developed with the possibility of using AI to design and continually modify parts of the system itself, not just enhance and build on top of pre-existing designs [19]. Studies cover the performance possibilities for AI integration for almost all aspects of the air interface communication chain, with many offering solid evidence that the possibility exists to enhance every step within it.

A common area of interest and implementation for AI models currently is on the receiver side, all the way from smaller models that handle channel decoding to models which fully generate the receiver stack from reception to presenting the received message[17]. Due to the nature of the air interface communication and the effects of the channel, AI models here can be used to more efficiently optimise complex calculations and allow these systems to more seamlessly adapt to changing environments. Specifically, the complexity inherent in the process of channel decoding means that this is fertile ground for the introduction of a neural network for the purposes of decoding incoming messages transmitted over the channel [16]. With these implementations, the large amount of potential codewords poses a limitation due to the curse of dimensionality, which means that training the network on every possible codeword is not feasible. In some studies, a small structured training set is used by the system to develop a decoding algorithm to be able to infer the rest of the code words [16]. In this case, the learning method plays a big role in the accuracy of the final system due to the small size of the initial dataset. As well as this, if the initial code set is random and does not follow a set structure in the generation of its code words, the performance of the system is significantly decreased due to its inability to form a consistent algorithm. However, with structured code systems the performance

is promising to an extent that it acts as evidence that neural networks can be used for the production of a decoder that potentially rivals or exceeds the performance of currently implemented methods.

This idea of the enhancement of complex tasks with neural networks has also been implemented on a more granular level, improving individual components of the processing chain. When looking at Orthogonal Frequency-Division Multiplexing (OFDM), a common data transmission method, replacing the entirety of the process with a neural network equivalent has lead to issues with regards to the restriction of certain performance capabilities. However, individual steps within the method have been replaced with AI equivalents with promising results. An example can be seen in [15], which replaces both the channel estimation error second order statistics and demapping steps with a Convolutional Neural Network (CNN) and finds it improves performance across the board.

Studies also exist that expand the scope to include the creation of full end to end systems through the use of AI, incorporating both the transmission and reception of a signal. This would entail training a neural network to handle signal generation, transmission, channel estimation and reception. The obvious issue in this case is the fact that such a design would not be able to comply with the need to have a differentiable channel model. One way that this issue has been circumvented is systems in which the channel model is avoided in the learning process [4]. Instead, the receiver and transmitter are trained in an alternating pattern by using the true gradient of the loss function and an approximation of it respectively. In each cycle, the receiver is initially trained by the transmitter sending a batch of messages over the channel. The receiver compares the received messages to the library of test messages, generating a probability matrix for each received message. This is then optimised through the use of Stochastic Gradient Descent (SGD). The transmitter is then trained by having it once again send messages across the channel and having the receiver generate a probability matrix. The per message losses are then sent to the transmitter via a feedback link, where the loss gradient is then estimated through SGD. This system shows promise in terms of performance, but suffers in terms of the complexity of the training phase and the requirement of a feedback link.

A lot of these studies and implementations have been prototypes or simulations, and there has been a notable recent push towards actually implementing some of these proposed designs in more relevant hardware, to identify potential areas of issue and validate the proposed gains in performance. Moving to these implementations will have an effect on the performance due to hardware limitations, and especially if they are implemented with a fixed point instead of a floating point number system. While the floating point implementation allows for greater flexibility and accuracy, a fixed point implementation is more representative of real world consumer computing devices. This is due to the fact that it allows compute units to be simplified and hardware resource and energy usage to

be reduced. This in turn makes any implemented NN more efficient and a more realistic consideration. Despite these simplifications and reductions in complexity, studies have found that given enough bits in sent and received messages, there is no noticeable loss in performance [3].

### 1.2.3 AI Model Testing Methods

The field of how to test and verify AI model performance grows and develops alongside the field of AI model development. The unique qualities of this technology necessitated that the methods used to perform tests are often just as complicated. Testing functionality remains as important as ever, but in recent years there has been a push to expand testing to include understanding of a models functionality and verifying the validity of the underlying logic.

A basic 'smoke test' for an AI model, a test that ensures basic functionality and key features are implemented as expected, is the most basic and most common method of testing an AI model. These types of tests often involve passing data through the model and noting if the results are as expected. While important and useful especially during the early phases of a development process, smoke testing is a design tool more so than an testing one, and as such it gives very little insight into the actual performance of the model.

Cross validation methods are used to test how a model performs when exposed to unseen data. In the K fold cross validation method, the test data is split into k parts. Each iteration, one of the parts becomes the test set and the rest become the training set. By averaging the generated metrics, more reliable estimates of accuracy and prediction error can be found [12, 39]. This method can also be taken to its extreme with Leave-One-Out Cross Validation, where the k number of parts is equal to the total amount of data, meaning that each iteration uses every data point bar one to make a prediction about that said data point. This is computationally more expensive and sometimes useful, but has issues with correlation for its estimates and can lead to high variance in its averages [30]. These methods are among the most commonly implemented in terms of validating performance and functionality, but struggle when dealing with edge case scenarios and also do not do much to increase the transparency of a model.

The edge case issues present in cross validation methods can be negated by performing Coverage methods. These methods are used to explore as much of the input space as possible to identify which parts of said space are not being covered adequately. Many of these methods attempt to use the inherent structure of the model to make evaluations about its coverage. This includes examining the fraction of activated neurons when processing a

test set or looking at the layers themselves and the combinations of their top neurons [29, 24].

Tests are also performed on more 'soft' aspects of the model, such as its bias, fairness or its privacy. Bias and fairness are crucial for ensuring that AI models make just classifications and decisions, and that the model has not carried over skewed viewpoints from its training set or its creators. These tests can be difficult to standardise, but often involve multiple different sub-methods to analyse the source and cause of any identified areas of bias and unfairness [32]. The privacy of a model relates to how well it protects its data, either data that is subject to patent or personal data relating to users, if the model is operating within this area. Methods that examine privacy often attempt to attack and 'trick' a model into revealing information it is not meant to, or exposing some type of back door through which the internal details of said model can be accessed [28].

In recent years, there has been a push towards methods which can measure a models resistance to adversity, be it intentional or incidental. This has given rise to the term and type of model testing method known as Adversarial Robustness, which describes methods which test a models ability to withstand various different types of onslaught once they are deployed [8]. These attacks would force the model to provide incorrect results, negatively impacting any infrastructure relying on the model. The attacks come from two main vectors, those being poisoned data and weight perturbation. Poisoned data attacks aim to adjust the data fed to a model to force it to act in unexpected ways [8]. If they are subtle enough, these attacks can be difficult to notice while having strong knock-on effects. Weight perturbation on the other hand attacks the weight parameters which help to determine a models output. Through making small adjustments to these values, the models prediction accuracy can be skewed, once again in subtle enough ways that make detection difficult [8].

Testing methods that measure adversarial robustness often subject a model to either of these types of attack, measuring how easy the attack is to execute, how negative the effects are and how easy it would be to detect. One such method is Fast Gradient Sign Method (FGSM), which attacks data by adding small perturbations to it to force misclassifications [14]. These perturbations effectively maximise the cost function instead of minimising it, generating data that will make the model perform mathematically worse. This field is new, and new methods are being developed all the time to ensure models are safe from these types of attack.

The main tools in terms of improving the transparency of the operation of AI models are methods that focus on Explainability. The lack of transparency is one of the main hurdles in terms of the implementation of AI models in many different environments, as the lack of trust means that they cannot be relied upon in situations that require 100% assur-

ance for the operator. This problem is exacerbated as the complexity of models increases; structures like neural networks are extremely difficult to interpret and understand when looking from the top down.

Explainability methods come in two main forms. The first are methods that do not know or take account of specific details regarding the model. These are known as Model Agnostic. These perform entirely independently of the model itself, relying on making changes to inputs and observing model reactions. One framework that makes use of these methods is the Local Interpretable Model-Agnostic Explanations toolkit, commonly referred to as LIME. LIME perturbs the original input around its neighbourhood and observes the reactions of the model. These datapoints are then weighted based on proximity to the original, and a new model is created based upon them. This new model is then used as a resource to explain why the original made the choices that it did [31]. On the other hand, Model Specific methods take into account the model that is under test. These offer the most transparency, but the methods themselves can be dependant on the model having a specific structure, which means they are not universal. For example, some of these methods are based on deconvolution and traverse the path of the CNN in reverse and point out specific features of the data contribute the most to the final decision. While taking a long amount of time and computational resources, the level of understanding offered by these methods is second to none, allowing for full explainability and customisation [34].

### 1.2.4 xpl[AI]ned - Keysight's new AI testing toolkit

xpl[AI]ned is Keysight's new internal toolset that is being developed to ensure the company remains ahead of the game in terms of testing as the explosion of AI enabled devices and technologies continues. As a company whose core business is allowing engineers to understand the relationship between given inputs and outputs, the complexity of the I/O relation for AI models poses a serious risk. Without this understanding, AI cannot be deployed into safety critical environments, as the technology cannot be developed to a high degree of confidence by the manufacturer and cannot be trusted or understood by engineers. At its core, xpl[AI]ned aims to standardise the process for creating explainable AI by combining existing testing methods including interpretability, certainty estimation and adversarial robustness into a single framework. The key difference that makes xpl[AI]ned unique is its design goal in so far as it aims to be used in every phase of the life-cycle of the development of a machine learning model. From problem formulation, data collection, through training, validation and final deployment, in this way ensuring a deep and fundamental understanding of the functionality of the model. The various tests it implements can be combined into an overall score that can rate the safety of the deployment of the model and accurately report as to where necessary adjustments need to be made. This automation and ability to provide insight means that when fully developed and deployed,

xpl[AI]ned should be an invaluable tool for engineers in many different fields.

## 1.3 Clarified Goals and Requirements

The initial proposal set out by Keysight Technologies was purposefully vague, as there were a lot of possible directions this project could have been taken and a lot of unknowns at the time of the proposals writing. With the essential background research performed, the goals and requirements of the project could be elaborated on. The aim of the project is the twofold investigation of the possibilities of modern AI model testing techniques in the field of air interface communication models, as well as the potential that all-in-one frameworks such as xpl[AI]ned have in the design, testing and evaluation process for these types of models. With the limitation of time and manpower in mind, the project aims to investigate two models with two different testing methods, adding on more methods in the later phases of testing if time allows. These tests should produce data that can be used to inform about the structure, functionality and performance of the model, and allow for conclusions that directly relate to the model's real world use, i.e., in an air interface communication system. If we can take the results generated by these testing systems and create direct links to the real world application, we can prove that the methods in use are valid and have value in this context.

With this in mind, a structured set of tasks/goals for the project is as follows:

1. **Select two models that exemplify a common use case for ML models in the 5/6G air interface**
2. **Understand the performance of both models to be used as a baseline**
3. **Select relevant and available testing methods to subject the models to**
4. **Apply the methods to the selected models and document/interpret the generated results**
5. **Extract conclusions about the relevance of selected testing methods, how they inform future design decisions for testing air interface ML components and what the results say about intricacies of working in this space**

Even so, the project remains open ended, with the scope more relating to the availability of resources rather than the level of technical development. The chosen models under test should be kept simple, for the purposes of simplifying the adaptation of testing methods to them and for allowing them to be easily explainable, both internally and in the context of this paper. The models should not require intensive rewrites of their internal logic and structure in order to fit with chosen testing methods, as this would invalidate

the overarching goal of the project and the xpl[AI]ned development within Keysight.

Another factor that is important to note is that while the production of interesting data that can be used to draw conclusions is the preferred goal from any given testing method, a testing method being incompatible and not working as well with a model is also a valid and interesting result. The project aims to investigate the possibility for the use of these methods in this context, and if a certain method does not work well, the reasons as to why can be helpful for future choice of testing methods for these types of models.

The report is structured into chapters, each covering a key part in the project development cycle. Chapter 1 has covered the initial background research and motivation behind the project. Chapter 2 presents a more in depth technical analysis into the necessary technical aspects of the project, including a detailed look at Neural Networks, potential models and the performance of the selected models. Chapter 3 is a record of the testing performed and results generated from said tests. This includes the two main testing methods as well as further smaller tests that were conducted. Chapter 4 goes into the implications of both the results, discussing what they mean and what they can tell us about the models, methods and working within this context. Finally, Chapter 5 wraps up the report with a look at the most important conclusions that can be drawn from over the course of the entire project.

# Chapter 2

# Technical Analysis

This chapter covers the more in-depth research that was performed when the direction/-motivation of the project was clarified. As previous research had shown that most air interface models made use of them, Neural Networks are looked into in more detail to understand their functionality, structure, training and any intricacies that can be expected of them. A short discussion about the process of choosing models for the project is also included, as well as an in-depth analysis of each chosen model and a record of their performance in terms of Bit Error Rate and confidence to use as a baseline for future testing.

## 2.1 Neural Networks

Neural networks are at the forefront of AI development when it comes to complex tasks such as air interface communications. Almost all models that we identified as potential candidates for this investigation featured neural networks of some description, so it is worth going into them in more detail.

### 2.1.1 Overview

As mentioned previously, neural networks consist of layers of interconnected artificial neurons whose activation within the network are controlled by what are called activation functions. These neurons have a set of weights that they multiply all their inputs by and then add together with a bias value. This is then passed to the activation function which decides the final value of the neurons output. During training, these weights are adjusted through back-propagation to minimise the value of the chosen loss function. Every neural network has its constituent layers organised into three main categories. The input layer is the one visible to the user on the front end of the network and it represents the dimensions of the input vector. At the other end, the output represents the final result generated by

the network. In between these two are all of the layers which calculate the final output based on the input. These layers are not visible/accessible, and so are called hidden layers [21].

A mathematical generalisation of a simple, fully connected neural network can be seen in (2.1). $r$ is the number of layers, $W$ is the set of weight matrices of which there is one per layer, $b$ is the set of bias vectors and $g$ is the set of activation functions. It is worth noting that the number of neurons and the activation function may differ on a layer by layer basis [22]. Fig. 2.1 shows a graphical example of such a network. The nodes represent neurons, and the lines between neurons represent the corresponding weights. A red line indicates negative weight and a blue line indicates positive, with the opacity of each indicating the magnitude.

$$\hat{y} = a^{[r]} = g^{[r]}(W^{[r]}a^{[r-1]} + b^{[r]}) \tag{2.1}$$

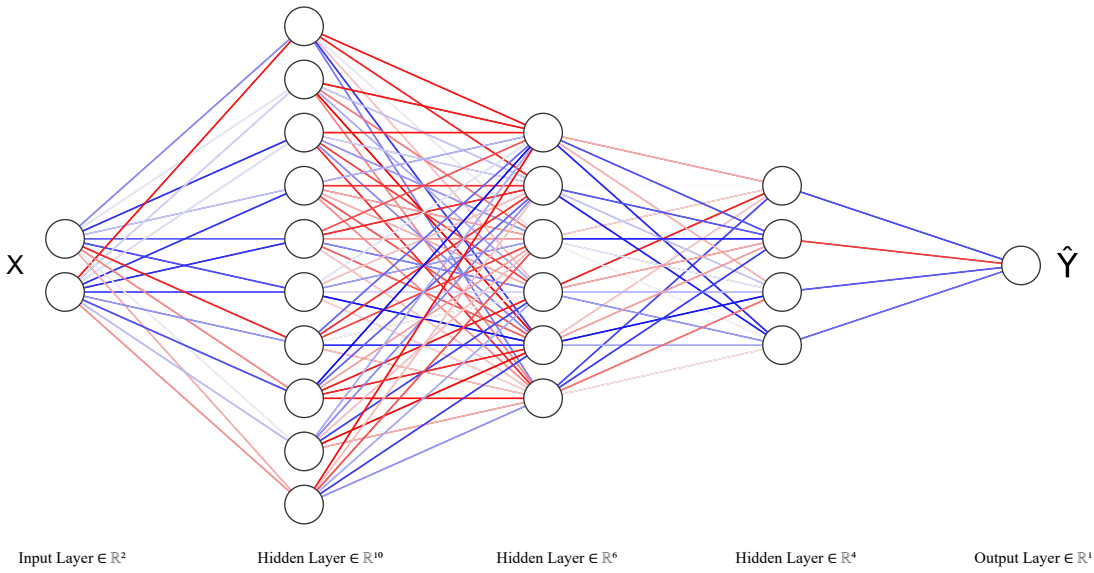



**Figure 2.1:** An example of a neural network portrayed graphically. The nodes represent neurons, with the lines between representing weights. The red lines represent negative weights and the blue lines represent positive weights. The opacity of each indicates the magnitude. This network has an input layer of size 2, an output layer of size 1, and 3 hidden layers

### 2.1.2 Weights

Every connection a neuron has to any neuron in the next layer has some "weight" associated with it, that is to say a coefficient by which its output is multiplied before being passed to the neuron in question. These weights can be considered to represent how much impact the value of the neuron has on all the neurons it is connected to. A high weight will mean the value of the neurons output is very important to the output of the connected neuron, a low weight means it will have very little effect. The output of any given neuron is found by taking the output of all of the neurons from a previous layer it is connected to, multiplying them by their respective weights, and adding on some bias value. The output is then fed through the neurons activation function to determine the final output [33].

### 2.1.3 Activation Functions

Activation functions work by normalising the output of each neuron and passing them on in the correct format. A linear activation function, shown in (2.2), means the neuron will output in full, multiplied by some coefficient with some added constant. A binary step function will go to one given a certain condition is fulfilled and be zero otherwise. A Rectified Linear Unit function, shown in (2.3) acts as a rectifier, only passing positive outputs [2]. A large variety of different kinds of these functions exist, but they can be broken down into three main categories. Ridge activation functions act on a linear combination of the input variables of the neuron [23]. Radial activation functions have an output that depends only on the distance between the input and some point, usually the euclidean distance [5]. Finally, folding activation functions are used in pooling layers, as they perform an aggregation over the inputs of the neuron.

$$\phi(x) = a + bx \tag{2.2}$$

$$\phi(x) = max(0, a + bx) \tag{2.3}$$

### 2.1.4 Loss Functions

The loss function is a key part of how neural networks function, and the choice of a loss function is highly dependant on the networks intended purpose. As we train the model, we feed data in and pass it through the model in a process known as forward propagation. Once an output is generated, it is compared against the desired output through a process known as back propagation, where the models parameters are adjusted so that the next instance of forward propagation produces an output that is more similar to the desired one. This comparison between the desired output and the generated output is the role of the loss function. The aim is to minimise the output of this function through every

instance of back propagation.

There are a few different types of loss function, but two specifically that are most commonly used for neural networks. Binary Cross Entropy (BCE)(also referred to as Log Loss), or just Cross Entropy when dealing with a multi-class problem, is the standard choice for neural networks whose purpose is classification [37]. Entropy is the measure of uncertainty associated with a given distribution, and cross entropy is the measure of uncertainty of a distribution based on another, different distribution. In a classification situation, a BCE function determines the uncertainty of the output based on its knowledge of the input. The more similar the output and the input become, the lower the entropy is. The final value or "loss" is an average of the individual losses from each datapoint in both distributions, and it is this that is optimised through the training process. The equation can be seen in (2.4). N is the number of datapoints in the test distribution, $y_n$ is the data label and $p(yn)$ is the probability given by the neural network for a given datapoint to be $y_n$

$$\text{Average BCE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n \bullet log(p(y_n)) + (1-y_n) \bullet log(1-p(y_n))] \tag{2.4}$$

For neural networks solving regression problems, where the problem is not one of assigning a category to data but instead generating discrete or continuous data that corresponds to some input, Mean Error (ME) loss functions are most commonly used for training [38]. These functions take the difference between individual results and their expected values, and then find the overall mean of this result. There are a few different versions of these functions. One of the most common, MSE has the advantage of ensuring outlier predictions are identified due to the weight that these types of predictions are assigned due to the squaring nature of the function. However, the amplification of outlier errors can make this function impractical. To solve this, Mean Absolute Error (MAE) functions take the absolute value of the error instead of squaring it, which is essentially the opposite in terms of advantages and disadvantages. These two functions can also be combined, to attempt to get the best out of both of them. Root Mean Squared Error (RMSE) can also be used, taking a square root of the MSE result to return the error in a more interpretable way and also reduce the effects of outliers on the error. The equations for these functions can be seen in 2.5, 2.6 and 2.7. N is the total number of datapoints, $y_i$ is the predicted value, and $\hat{y}_i$ is the actual value.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.5}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.6}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.7}$$

### 2.1.5 Training

Training an Neural Network is an iterative process that involves adjusting the previously mentioned weights and biases within each neuron based on the output of the loss function. If we take one data sample and forward propagate it through a network that is being trained, the input is fed through the layers, passing through neurons until the network generates as its output what it believes to be the correct response to the input. Assuming a supervised learning situation, we can take the our known expected result and compare it to the output of the network. The loss of the network is the average of the losses across all of the datapoints, determined by the loss function in use. The training process involves pushing this network loss to its theoretical minimum value. To do this, the loss of each individual data point needs to be considered. This loss value tells us proportionally how much the weighted sum of all the inputs for that particular output neuron needs to change in order for that output to be at the desired value. To change this value in the most efficient way, the neurons with the highest value from the previous layer for this particular output neuron should have their weights increased, and the ones with the lowest value should have their weights decreased. This in essence is incentivising the neurons that are contributing to the "correct" decision, while disincentivising the ones that are pushing towards an incorrect one. This is then done for every single neuron in every single layer, with the final result being a calculated average "best" change for each weight present in the network, also known as the gradient. This is then applied to the network, and another forward propagation can performed. This process is called Back-propagation and is the core of how neural networks "learn" [33]. Very often, to reduce computational complexity, the required weight changes are not calculated for each individual datapoint, but instead for small randomised batches of them, decreasing the accuracy of the gradient calculation but increasing the overall speed of finding it in a process known as Stochastic Gradient Descent (SGD) [33].

### 2.1.6 Hidden Layers

The layers within the hidden section can take multiple different forms. These forms are usually defined by the way that the layers are interconnected. A fully connected layer for example is one in which every neuron within the layer is connected and influences every neuron within the next layer [11]. These layers are used sparingly and for specific purposes, as they are very sensitive to growth in the size of their inputs, which leads to a exponential growth in how computationally expensive they are. They are commonly used for classification. Convolutional layers, as the name suggests, perform a convolution on

their input and pass the result on to the next layer [11]. This result is an abstraction of the input, sometimes referred to as a feature map, which allows the complex and large input to be represented by a smaller set of abstract features that can be used by the network for classification. A de-convolutional layer naturally does the opposite, taking feature maps and returning them to their original formats [11]. Pooling layers take the outputs of large clusters of neurons and reduce them into the output of a single neuron that is passed onto the next layer [1]. This is done to reduce the dimensions of data and therefore the computational complexity of the network. Layers can be provided with a form of "memory" through the use of Recurrent layers, which as an input take the output of the previous layer but also the previous iteration of their own output [11]. This iterative nature allows for better performance in situations where input data is of a sequential/deterministic nature.

### 2.1.7 Types of Neural Network

All of these factors can be put together in a myriad of different ways to generate different types of neural networks [21]. A few of the more complex and relevant examples include a Multi-layer Perceptron, which features only fully connected layers and bi-directional propagation. While good for deep learning due to the high level of interconnectedness between the neurons, these networks struggle to be built and maintained due to their inherent complexity. A Convolutional Neural Network attempts to solve this issue by introducing convolutional layers, which means fewer parameters are required but the network can function slower depending on the number of hidden layers. Radial Basis Function neural networks make use of radial basis activation functions in their layers to perform classifications based on euclidean distance. The most relevant of these types of network for our purposes is the Convolutional Neural Network, as its balance of lower computational complexity and high performance ceiling means it is very commonly implemented in the air interface communication space.

### 2.1.8 Intricacies of Neural Network training

Training a given network to perform well on training data is the process of optimisation, but it truly becomes machine learning when a network can be shown to work well on previously unseen data. The ability to work well on unseen inputs is called generalisation, and the error when the network is operating in this context is called the generalisation error or the test error. While training works on making the training error as small as possible, the minimisation of the test error is the real goal of network design and training. If we assume i.i.d samples for both training and test data, the training error and the test error should be equivalent, so the sign of a well trained network is one where the training error is very close to the test error [13]. Two main training errors arise when consider-

ing this performance metric, those being underfitting and overfitting. Underfitting occurs when the training error cannot be optimised to a sufficiently low value, leading to an under-performing network that does not 'understand' the underlying patterns in the data. Overfitting occurs when the training error is small, but the gap between it and the test error is excessively large. This occurs when the network 'memorises' aspects and patterns that occur within the training set that do not represent the overall trend of the data and therefore lead to it making mistakes when exposed to unseen data. These two problems are often a trade-off: we want our network to be able to generalise and understand the underlying trends in data while also ensuring it can extrapolate those trends to data it has never seen before [13].

A models 'capacity' is its ability to fit a wide variety of functions and is the characteristic that is altered to control a models likelihood to over or under fit. There are many ways to change the capacity of a model, including changing the size of the input layer, changing the type of model, and changing the types of function that the model can represent [13]. When looking at the functions specifically, a simpler one will be more likely to generalise, but the function must be complex enough in order to accurately represent the trends in the data. Modifying a models capacity is a balancing act: leave it too small and the model is incapable of solving complex tasks, while a capacity that is too large for the task at hand can lead to overfitting. The optimal capacity is one where the model can generalise well, while also minimising the gap between the training and test errors [13].

## 2.2 Models

### 2.2.1 Acquisition

The first step of the investigation on the technical side was to look into what models were available and which models would be suitable for the purposes and scope of the project. The requirements of the project, with it being an exploration of testing possibilities rather than a direct measure of performance of specific models, meant that the scope was quite broad. The age, source and relative performance of the models in comparison to others was less important than their purpose being common and representative of the field as well as being of a common format that would be easy to adapt to the xpl[AI]ned software. As the search progressed, two main potential sources were identified: models sourced from research papers covering development of AI in the air interface space, and models sourced from Nvidia's Sionna research software.

### Models from Research Papers

A lot of research papers were studied during the early phases of the investigation to develop an understanding of the current state with regards to AI in the air interface and potential future developments. A lot of the models within the papers were deemed unsuitable, as they represented a very niche or complicated process that would pose unnecessary difficulty in terms of implementation and testing. The xpl[AI]ned toolkit was untested in this field, so models with less specialised purposes were preferable. After looking at all of literature used for the project, two candidates stood out, both related to channel decoding. The first was a model developed in a paper titled "On Deep Learning Based Channel Decoding"[16]. This model uses a deep neural network to decode structured codes, for the purpose of proving that neural networks can be used to decode algorithms, rather than just as classifiers. The restrained scope and simple structure lends itself well to our application. The second was a more modern update to the first, titled "Graph Neural Networks for Channel Decoding"[6]. The aim of this paper was the same as the first, but with a more advanced structure and performance potential. With this in mind, it was also earmarked as a potential option.

### Models related to Sionna

Sionna, an open source library made by Nvidia for the purposes of 6G research, contains within it and supports many different models relating to the air interface [18]. These are either open source examples of possible Sionna implementations or models that are used within Sionna as a key part of its functionality. Sionna was suggested as a possible resource for this project by Keysight themselves, who are working alongside the developers Nvidia for the previously mentioned CENTRIC project. Of the variety of models present within the library, two were identified as being fit for our purposes, both of which coming from tutorials on the Sionna documentation. The first model comes from a tutorial in the Sionna documentation titled "Neural Receiver for OFDM SIMO Systems" [27]. This tutorial detailed the creation of a model that could be used to simulate a OFDM receiver, and was set aside as a possible option due to the fact that the model simulated a complex system in a seemingly simple way. Another tutorial, "End to end Learning with Autoencoders", describes a model which consists of a full communication system, from transmission to reception [26]. This model represents a much simpler system than the previous one, and is much lighter and easier to run. As well as this, some aspects of the models structure seemed relevant and interesting. Both of these models also share the advantage of very well documented code and design documents, owing to them both being the subject of tutorials.

### Final Model Choice

The final choice of models to be analysed and tested came down to a few factors. The most important of these were how easily the models could be made to work with the xpl[AI]ned framework, how straightforward the purpose of the model was and how understandable the inner functionality of the model was for the author but also for prospective readers. With all of these in mind, a final decision was made to use the two models related to Sionna. The readability and explainability of these models was far above any other prospective ones, and the ample documentation, age, and consistent formatting of these two models meaning that the initial infrastructure and setup work that needed to be done by both the author and the developers working on xpl[AI]ned was minimised. This last advantage is magnified by choosing both of these models, as their similar structures should reduce the amount of backend work required. The structure and functionality of both chosen models, as well as the steps taken in order to prepare them for testing is described in Sections 2.2.2 and 2.2.3.

## 2.2.2 Model 1: Neural Receiver for OFDM SIMO Systems

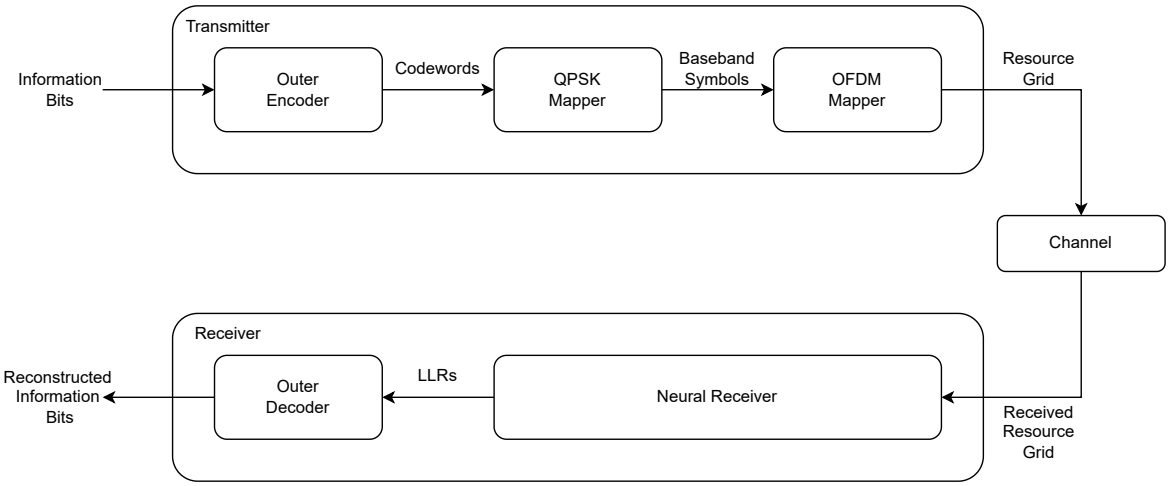



**Figure 2.2:** Block diagram showing top level design of the Neural Receiver model. The model also features the possibility to run two different non AI baseline implementations, these have not been represented here[27].

The first model is an implementation of a trained receiver for OFDM detection [36]. This is a complex operation, involving knowledge of the channel, through estimation or full knowledge of the channel response, some form of equalisation, and demapping based on the type of modulation used(QPSK in this case). The common way that this is done, as shown in the Sionna tutorial for this model, is through the use of Least Squares baseline estimation alongside an LMMSE equaliser. A diagram of this method can be seen in Fig. 2.3.

The OFDM resource grid is received and fed into a LMMSE equaliser and an LS channel estimator. This estimator uses the transmitted pilot data to estimate the channel characteristics using nearest neighbour interpolation [27]. With this data, the LMMSE equaliser can then extract the symbols from the grid and feed them to a demapper and decoder in order to reconstruct the initial information bits.

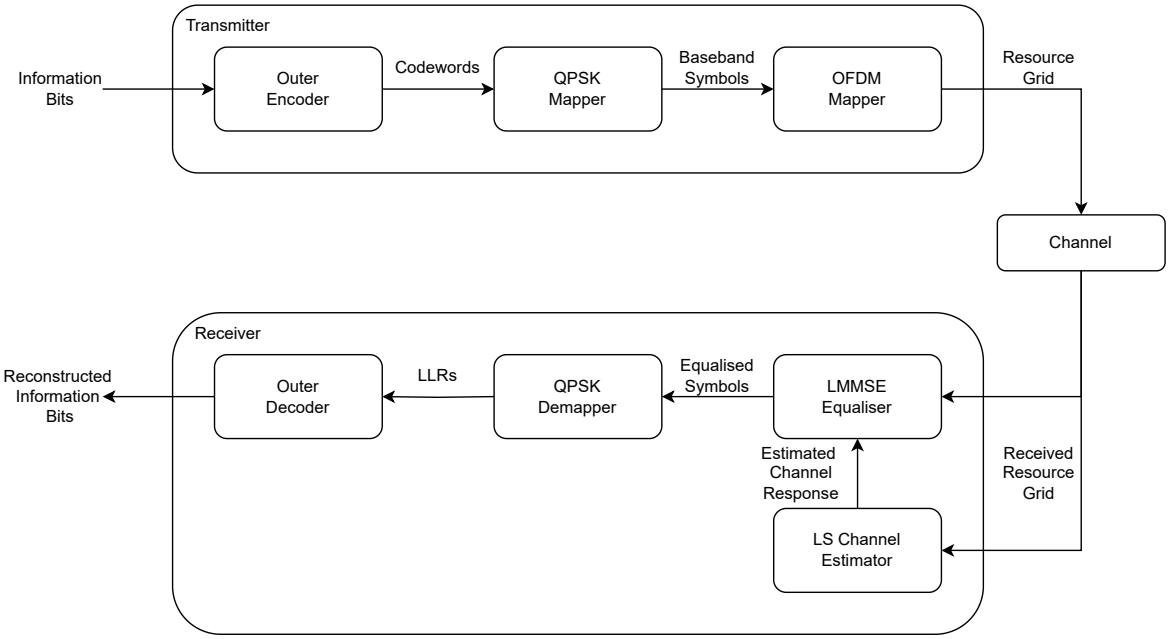



**Figure 2.3:** Block diagram showing an example implementation of OFDM detection without the use of a neural network

The model replaces channel estimation, equalisation and demapping with a neural network. The Sionna example/documentation also includes surrounding infrastructure allowing the model to be trained and tested, such as a channel simulator, en/decoders and mappers. A top level diagram can be seen in Fig. 2.2. The receiver itself is made up of Keras convolutional layers. These layers are put together into residual blocks, with residual connections implemented to avoid the gradient vanishing problem. Diagrams describing the structure of the Residual blocks and the Neural receiver can be seen in Fig. 2.4 and Fig. 2.5.

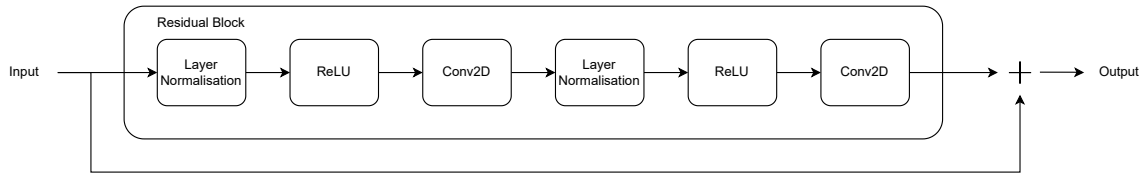



**Figure 2.4:** An example of one of the residual blocks that make up the Neural Receiver.

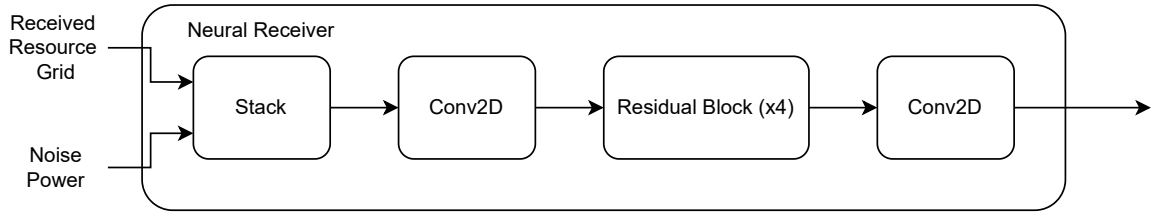



**Figure 2.5:** A detailed look at the internal structure of the Neural receiver in the Neural Receiver model.

In terms of the data structure of the model as it relates to I/O and training, while the input to the whole infrastructure is the initial information bits, the input to the model itself is the FFTs of the received resource grid and the output is a set of Log Likelihood Ratio (LLR)s with the magnitude of negative results showing the models confidence in a given bit being zero, and the magnitude of positive results showing the confidence in it being a one. During training, a modified Binary Cross Entropy (BCE) loss function is used to train the model. The Bit-Metric Decoding rate (BMD)is calculated by computing the average entropy between the encoded information bits and the output LLRs and subtracting them from 1. The inverse of this value is the value being optimised during training. This loss function can be seen in equation 2.8. R is the Bit-metric decoding rate [35], S is the batch size, N is the number of subcarriers, M is the number of OFDM symbols, K is the number of bits per symbol, $B_{s,n,m,k}$ is the kth coded bit transmitted on resource element $(n, m)$ for the sth batch example, $LLR_{s,n,m,k}$ is the LLR computed by the neural receiver corresponding to the kth coded bit transmitted on the resource element $(n, m)$ for the sth batch example and BCE is binary cross entropy in log base 2. During operation, the LLRs the model generates are converted back to information bits through the use of a decoder.

$$R = 1 - \frac{1}{SNMK} \sum_{s=0}^{S-1} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} BCE(B_{s,n,m,k}, LLR_{s,n,m,k}) \tag{2.8}$$

Observing the structure of the model shown in Fig. 2.2 we can see that the output of the code provided by Sionna is not actually the output of the model; instead being the output of a Low Density Parity-Check Code (LDPC) decoder which turns the LLRs produced by the model into the reconstructed information bits as mentioned previously. This poses an issue with regards to the presentation of our results. A lot of the methods we would like

to implement involve procedures that will induce errors in the model. The LDPC decoder corrects errors, and while given enough errors the degradation in performance would still be evident, this error correction would make it difficult to make conclusions about the performance of the model itself, as opposed to the performance of the entire receiver structure.

To solve this, modifications had to be made to the structure of the model, or more accurately the infrastructure surrounding it. The first step was to identify where the output of the model itself was, and to "disconnect" it from the decoder. The model output as mentioned was a set of LLRs, 2784 to be exact, corresponding to the 1392 symbols with 2 bits each that comprised each message that the model processed. As mentioned, a positive LLR value indicated a predicition of a bit being one, and vice versa. This meant that a basic interpreter could be added to the output of the model to convert these LLRs to the bits that the model predicted, and then the output of this interpreter could be set as the output of the overall model infrastructure itself.

In order to compute the initial bit error rate and run future tests, the input and the output of the model needed to be of the same format, and the input had to be a known set. As mentioned, the output of the model now matched the format of the codewords generated by the encoder from the information bits that the binary source within the model generated randomly. This part of the model was then also modified to no longer use the random binary source but to take an array as an input, an array that would be a batch of known, pre-generated codewords, 2784 bits in length. With both of these modifications, we now had full control of both the input and output of the model, as well as assurance that any mistakes made by the model would be accurately detected. A block diagram of the modified model infrastructure can be seen in Fig. 2.6.

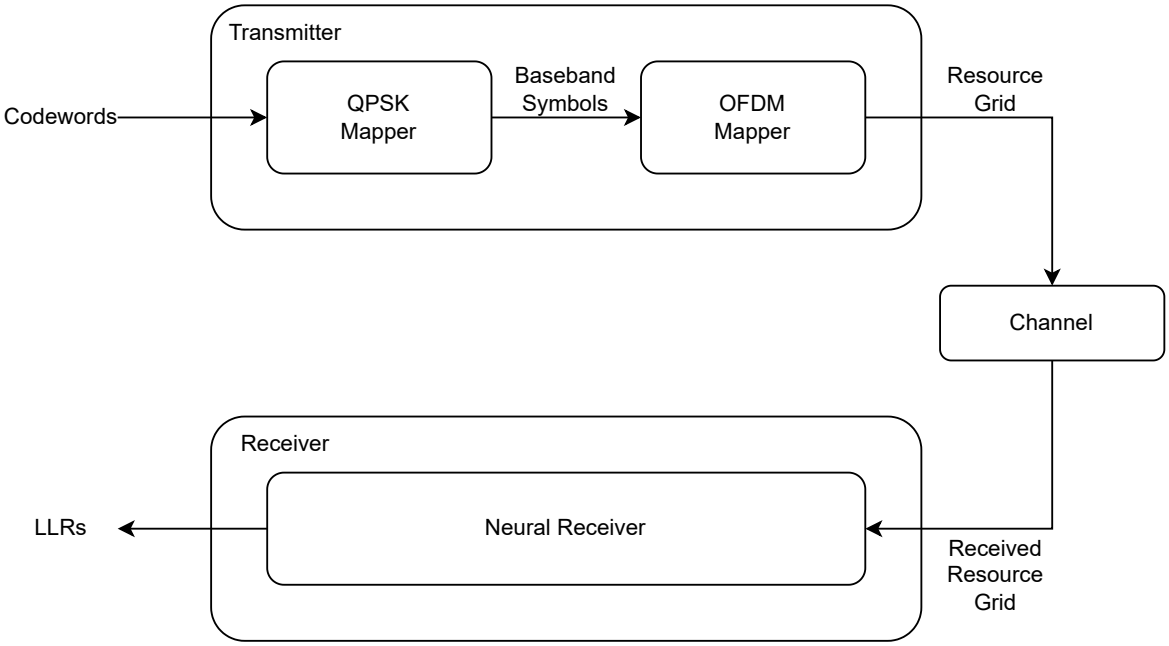



**Figure 2.6:** A block diagram showing the top level of the Neural receiver once necessary modifications had been made to allow for accurate testing and control of input and output.

### Training

This model was trained using the base parameters specified in the Sionna tutorial, with 30,000 iterations on a batch size of 128 codewords with an $E_b/N_0$ range of -5 dB to 10 dB.

## 2.2.3 Model 2: End to End Learning with Autoencoders

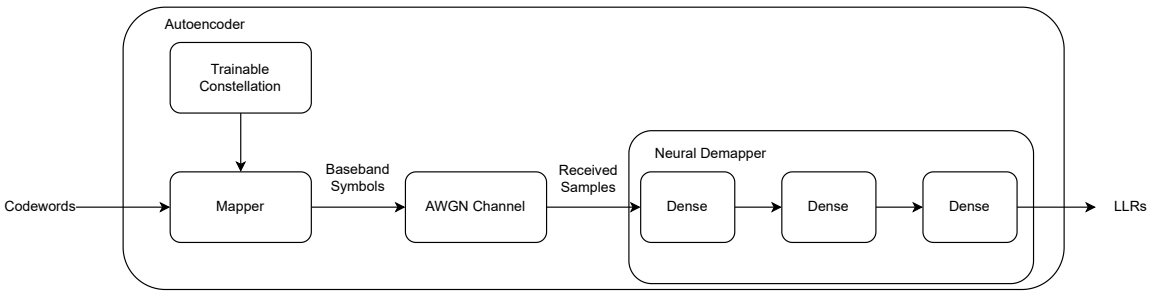



**Figure 2.7:** A block diagram showing the top level functionality of the End to End Autoencoder model

The second model that would be put under test instead modeled an entire end to end autoencoder, where a full communication system from transmission to reception is trained.

A typical baseline for this type of system can be made using a fixed QAM constellation/mapper, AWGN channel emulator and AWGN demapper. The model itself consists of a trained constellation instead of a fixed one, and a neural network taking the place of the demapper [26]. This demapper features three layers, with the first two comprising 128 neurons and using the Rectified Linear Unit activation method. The final one features as many neurons as the number of bits per communication symbol, in this case 6, and has a linear activation function. A block diagram of the models structure can be seen in Fig. 2.7

This model had two different potential training methods. It could be trained through conventional means, through the use of SGD and back-propagation of the gradients through the channel. This inherently assumes the presence of a differentiable channel model and knowledge about the channel characteristics. The other option does not make this assumption, and instead uses Reinforcement Learning (RL) by alternating between training the receiver and then training the transmitter based on the results from the receiver training. This training method is based on a study that is mentioned in Section 1.2.2, and was the implementation that ended up being chosen for testing. Diagrams showing these two training modes can be seen in Fig. 2.8.

With regards to the data structure of the model, the transmitter takes the base information bits as an input and provides mapped baseband symbols as an output, while the neural demapper takes samples received from the channel and noise as input and outputs a set of LLRs. During training, the demapper is trained using a basic BCE loss function and SGD, with the loss for the demapper computed as the average BCE value. This BCE value is fed back to the transmitter through the reinforcement link. Backpropagation is not possible here due to the assumption of no known channel model, so the gradient has to be calculated differently. During transmitter training a known perturbation is applied to the transmitter output. By combining the gradient of this perturbed output with the BCE loss computation from the receiver, an estimation of the transmitter weight gradients can be obtained by approximating the loss function. These gradients are optimised over time to train the transmitter. This entire process is performed iteratively, improving each section based on the improvements to the other section from the previous iteration.

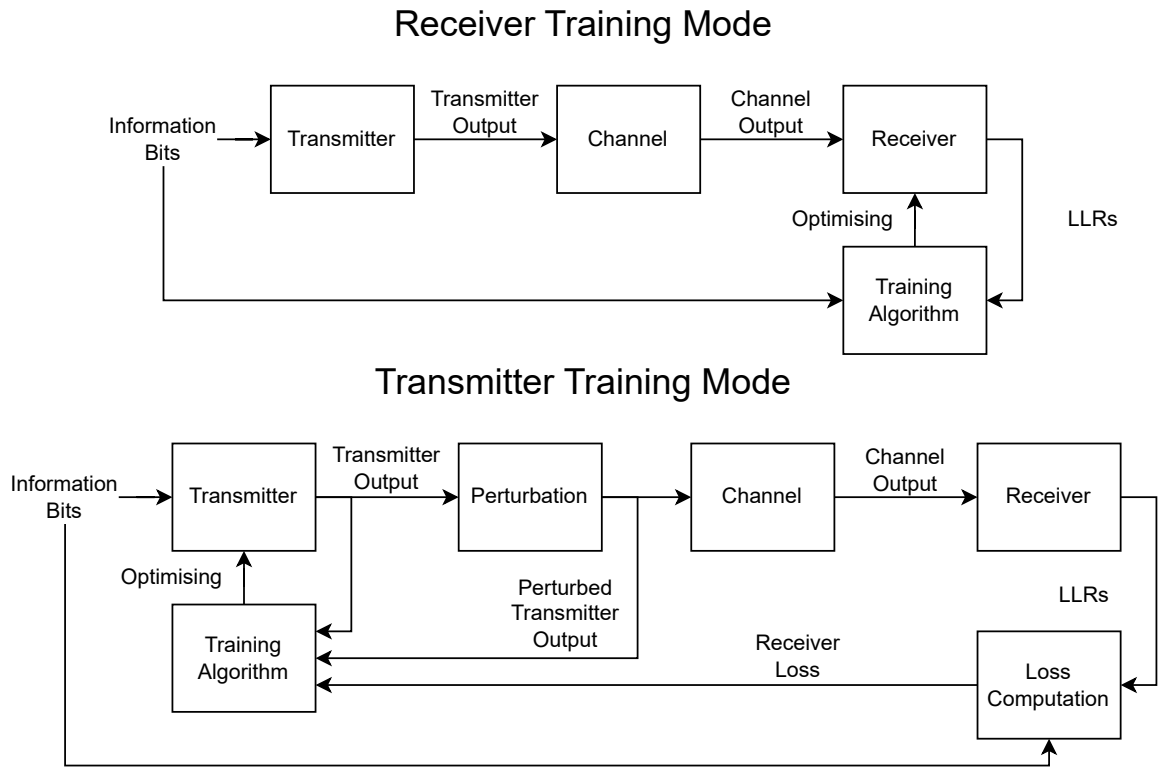



**Figure 2.8:** Block diagrams showing the iterative training setups for the End to End model in its Reinforcement Learning Training mode. These diagrams are taken directly from the literature the training method is based on [4]

This model required very similar changes to the Neural Receiver model, with the models standard random input generation being circumvented to allow for a consistent, known input to be used. Once again, the output was changed to LLRs originating directly from the models output instead of the bitstream from the built in decoder. This model used very different parameters to the Neural Receiver model, with a 64-QAM modulation implementation instead of QPSK and a smaller codeword length of 1500. The possibility was initially considered that these could be changed to allow for greater parity to the Neural Receiver model for the purposes of comparing results. However, considering how different the two models are in terms of complexity and their implementation, matching certain parameters would not necessarily bring these models into any sort of parity. Furthermore, for the purposes of this investigation these models are acting as tools with which to examine the usefulness of certain AI testing methods, so being able to compare their performance like for like is not a requirement.

**Training**

For the purposes of the investigation, the RL training method presented the more interesting and relevant implementation, as it more accurately represented what was more likely to be the real world situation. It was then trained with a total of 10,000 iterations on a batch size of 128 codewords with an $E_b/N_0$ range of 5 dB to 8 dB.

## 2.3 Determining Model Performance

With both models now prepared for further testing, the first step was to establish a performance baseline for both, by investigating their respective bit error rates. The Sionna tutorials where both of these models featured had documentation about the block error rates, but considering the modifications made to both models and the fact that quite a few of the potential testing methods required more granularity, calculating the bit error rate was necessary. To do this, the first step was to generate a data-set that would be used across both models. This data-set of codewords would be the input, and once the models processed the data-set and returned their interpreted LLRs, a basic bitwise XOR operation could be performed to find any bits that differed between the two sets. The sum of this value divided by the total number of bits in the data set would then give the bit error rate. This operation could be performed across multiple different Energy per Bit to Noise Power Spectral Density Ratio ($E_b/N_0$) steps to produce a graph showing the overall performance of each model.

The results for both models can be seen in Fig. 2.9 and Fig. 2.10, each showing an average Bit Error Rate (BER) curve with runs of a batch size of 10,000. Both models reach their minimum BER between 10 and 20 dB(13 for End to End, 18,5 for Neural Receiver). This is higher than the zero Block Error Rate (BLER) points in the tutorials, but this is due to the Bit Error Rate being more granular than BLER, and the removal of the LDPC decoder at the output which would perform a certain amount of error correction. Both models also show some unexpected behaviour. Looking at Fig. 2.10, The Neural Receiver model experiences lot of volatility in its BER at the high $E_b/N_0$ values, with the overall trend being a decrease but not as smoothly as it is for the End to End model. Both models also show some unexpected behaviour as the $E_b/N_0$ is increased further. We would expect that as the $E_b/N_0$ rises, the quality of transmission increases, and the BER should remain low/zero. However, if we look at Fig. 2.11, the opposite is observed. This plot shows the results for the End to End model, but this behaviour is consistent for both models.

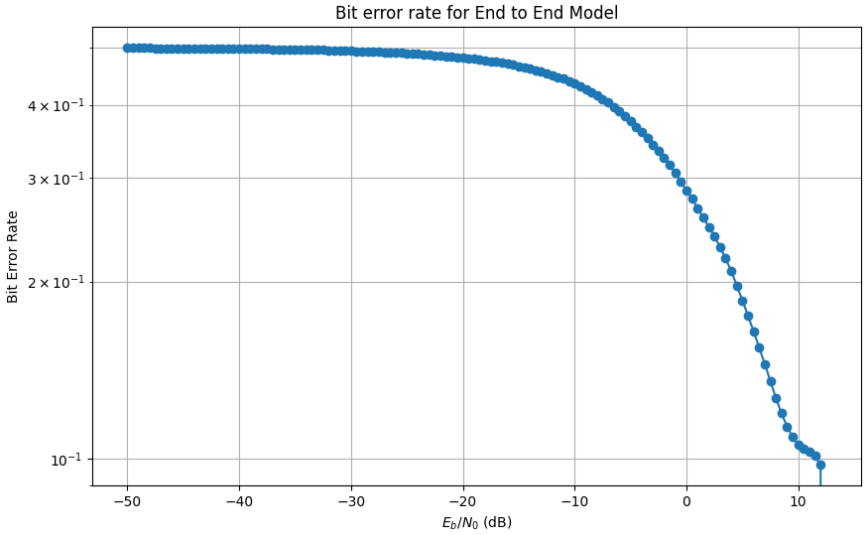



**Figure 2.9:** Bit error rate curve for the End to End model, with a minimum BER value of 0.098 reached at 13dB

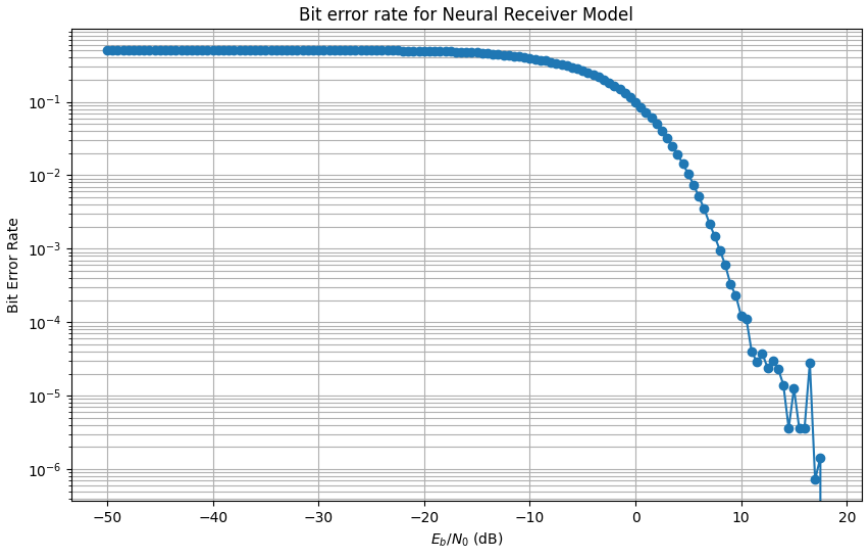



**Figure 2.10:** Bit error rate curve for the Neural Receiver model, showing minimum BER at 18,5dB but also showing very inconsistent results at the higher $E_b/N_0$ levels.

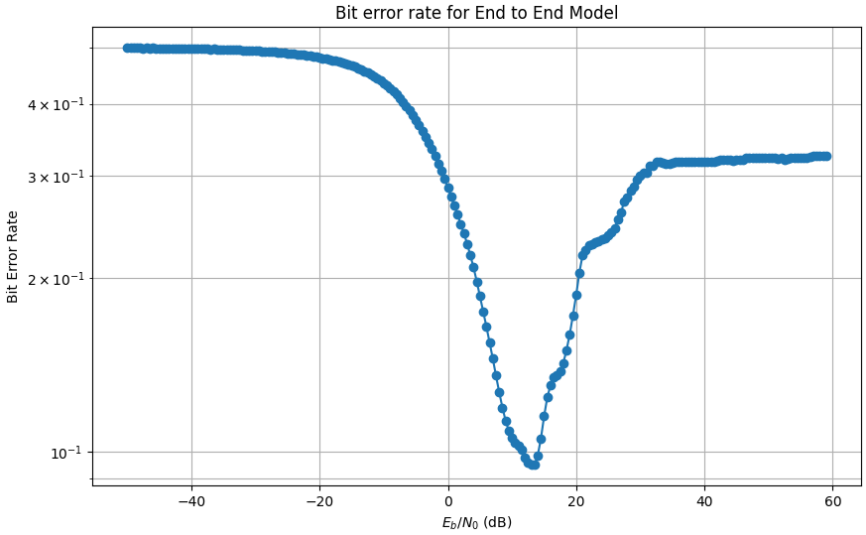



**Figure 2.11:** Extended bit error rate curve for the End to End model, showing the unexpected performance at higher $E_b/N_0$ levels.

In an attempt to investigate the inconsistency at higher $E_b/N_0$ levels for the Neural Receiver, it was retrained, this time with the maximum possible $E_b/N_0$ in the training range increased to 20 dB from 10 dB, as it was noticed that the inconsistency in its results appeared to start at 10 dB. The result of this retraining can be seen in Fig. 2.12.

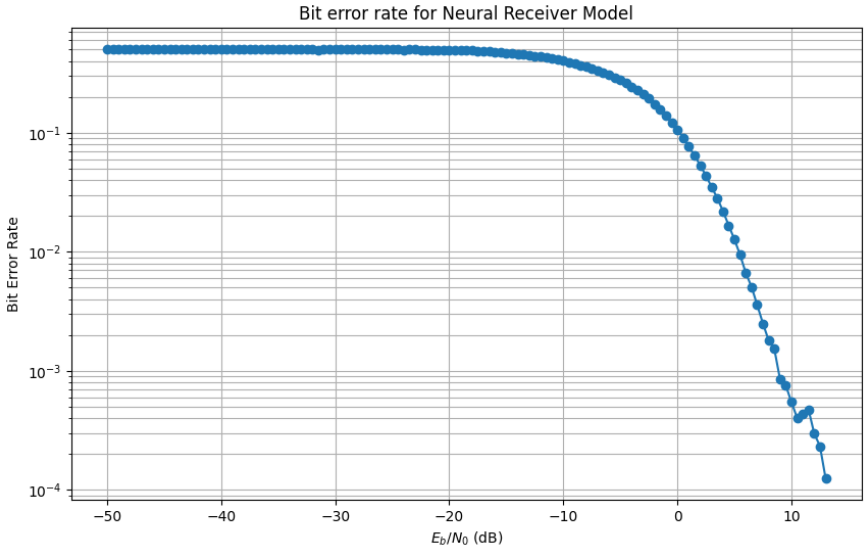



**Figure 2.12:** Bit error rate curve for the Neural Receiver model after retraining, with minimum BER value reached at 13dB and showing reduced inconsistency prior to reaching the minimum value

This retraining seemed to have two effects. Firstly, the performance of the model ap-

pears to have improved, with a minimum BER ($1.3 * 10^{-4}$) reached at 13 dB. As well as this, the inconsistency in the results prior to reaching the minimum appears reduced significantly. However, once past the minimum BER mark, the inconsistency returns, and the BER continues on an upward trend. For the sake of better consistency, this retrained version will be used for future testing.

The issue of undesirable performance at higher $E_b/N_0$ values is interesting, but does not pose any issues in terms of disrupting tests that we wish to run. It could potentially be investigated in the future to identify the source and attempt to alleviate it.

A more in depth understanding of both models performance can be gained by going one step deeper into the model and collecting data about the model's confidence in its result. This value is represented by the Log Likelihood Ratio (LLR)s in the case of both of our models, a measure of the relative likelyhood of the decisions the model can make. For example an LLR of -15 shows a higher likelyhood of a bit being zero than an LLR of -5, while positive LLR values represent the likelyhood of a one. For both models, the maximum magnitude for these LLRs in practice was around 20, although in theory their range is infinite. Understanding these LLR values would help to put the BER results into context, and also act as an important baseline for any future testing methods which would affect the confidence of the models' decisions.

These results can be seen in Figs. 2.13 and 2.14 where the confidence of each model at a given $E_b/N_0$ level is represented as an average of the absolute value of the LLR output at said level, henceforth called the Decision Confidence. It is defined in (2.9), with $N$ in this case is the number of codewords.

$$\text{Decision Confidence} = \frac{\sum_{i=1}^{N} abs(\text{LLR Output})}{N} \tag{2.9}$$

The results are shown from -20 dB, right when the performance for both models ceases to be random, to 20dB. The same pattern is evident in both graphs, but to different extents. In both models, the confidence at -20 starts off relatively high. In this region, the noise level is so high that it overpowers the signal, and the model attempts to classify the noise itself, causing the high confidence but essentially random model output. As the models' performance with respect to the BER begins to improve, the confidence decreases from this artificially inflated value as the model is able to identify and classify the signal, but poorly. The confidence continues to increase as the minimum BER point is reached, and in Neural Receivers case remains consistent, while in End to Ends case continues increasing. This is despite the fact that for both models the BER performance begins to become inconsistent and/or degrade past the minimum BER point. For the End to End model, a possible explanation for this is epistemic uncertainty, due to the fact that the model was not trained on these higher $E_b/N_0$ values. However, with the retraining performed previously the Neural

Receiver model has been trained to up to 20 dB, so the high confidence in this area does not have an immediate explanation.

All of the effects described above are much more noticeable and of a higher magnitude in the End to End model (Fig. 2.14). The most likely reason for this is due to the difference in training ranges of the 2 models, with End to End having a range of 3dB compared to Neural Receivers 25dB. This very small range means that any effects caused by the model dealing with data outside of its range will be exacerbated when comparing both models across the same range of $E_b/N_0$ values.

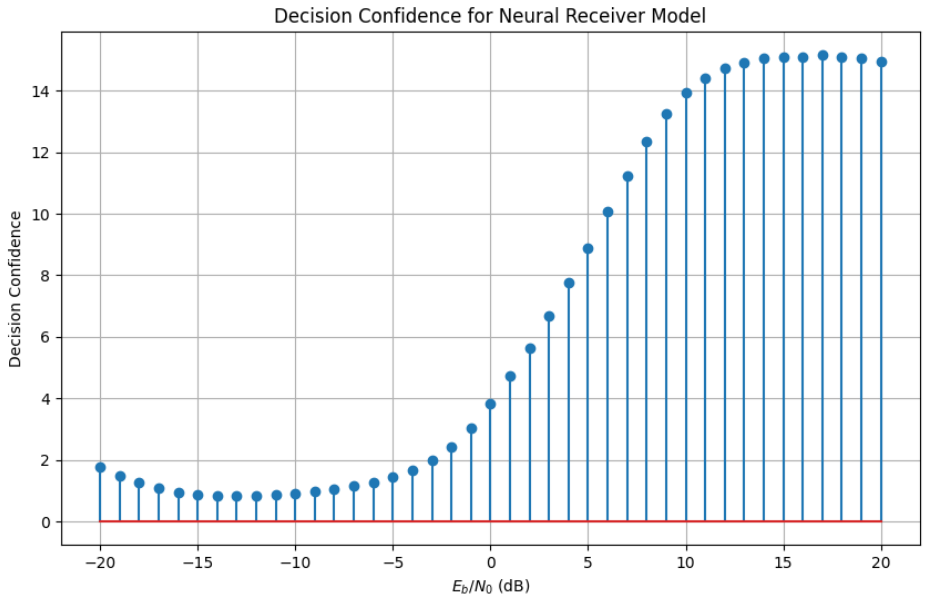



**Figure 2.13:** Decision confidence for the Neural Receiver model. Confidence starts off higher in the region where the models output is essentially random, lowers and then gradually increases in the region where the models performance begins improving but noise is still high, and then rises rapidly as the model approaches its optimum operating $E_b/N_0$ value. It continues increasing past this value, even though the model performance begins to degrade.

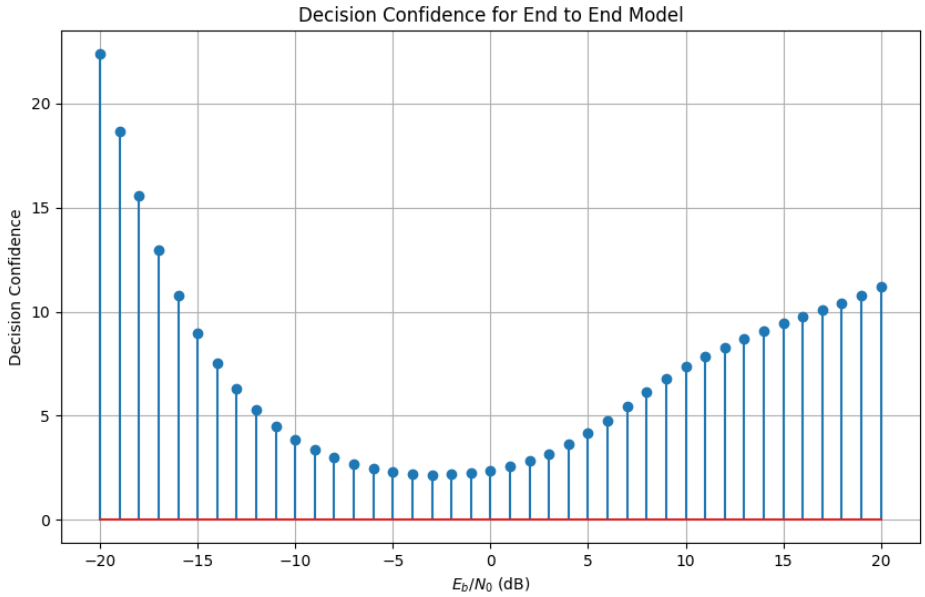



**Figure 2.14:** Decision confidence for the End to End model. Confidence starts off higher in the region where the models output is essentially random, lowers and then gradually increases in the region where the models performance begins improving but noise is still high, and then rises rapidly as the model approaches its optimum operating $E_b/N_0$ value. It continues increasing past this value, even though the model performance begins to degrade.

# Chapter 3

# Testing

This chapter covers the implementation of various testing methods and the results generated from said methods. Two main methods are outlined and implemented, those being Monte Carlo Dropout and Fast Gradient Sign Method (FGSM). Documentation about the implementation of a couple of other methods is also present, as well as the results of some independent testing motivated by the results produced by the main testing methods.

## 3.1 Testing Methods

With the models ready and their functionality and performance understood, the next step was to determine which testing methods they would be subjected to. A large limiting factor here was the methods that were available/possible to implement currently within the xpl[AI]ned framework. We knew that we wanted to run tests in different areas, such as robustness, certainty or explainability, in order to understand how they could be used in the context of an air interface model. After meeting with the xpl[AI]ned team and discussing the various possibilities that already existed or could be developed, a few main testing methods were decided upon.

#### Monte Carlo

The first method was Monte Carlo Dropout, a method which helps to determine certainty as well as improve the efficiency of a model. Dropout in a neural network context refers to neurons that can be switched off during the iterative training process. These are randomly switched off at some pre-determined rate, anywhere from 0 to 100% and are used as a technique to prevent overfitting, as they allow for information to spread across the network more evenly and make the model better at generalisation. As well as this, as different neurons are dropped during each training iteration, the final model can be considered an

average of many slightly different architectures, resulting in a better overall performance. The Monte Carlo dropout method takes these dropout neurons and activates the dropout during the testing phase, instead of the training phase. This has multiple different consequences and potential areas for analysis. It means that, if the method is active, each instance of the model being run is actually a run made by a slightly different architecture. These different runs therefore have the potential of delivering different results. By looking at the errors made during the testing of each of these different architectures, we can analyse and see how resistant the model is to neuron failure; how much performance is lost if 10% of the neurons malfunction, or 50%. As well as this, it can give us information about how certain a model is about its result. If at small dropout rates, the error in the models predictions does not change by a relevant amount, we can conclude that the model has a high degree of certainty. This can also give some insight about the model's efficiency, as if no drop in performance is observed after a high degree of dropout, the model might be too large, and could stand to lose neurons without expecting a drop in performance. Conversely, if the model experiences significant drops in performance with very low dropout rates, it could indicate that the model is as efficient as possible, and that every single constituent neuron is necessary to maintain its performance.

In our context, Monte Carlo dropout would be used to examine the effect on the bit error rate that losing certain proportions of neurons during operation would be. This would be done across a range of different $E_b/N_0$ values, to determine how much of the information can be lost while still maintaining acceptable performance. Due to the nature of the information being sent and the lack of redundancy, the effects of this are likely to be extreme, especially at the input and output layers. The hidden layers may have some sort of robustness however, the exact nature of which will be determined. The conclusions about the model's efficiency will be arguably more useful, as hardware in this context is often limited and unnecessarily heavy models would be a big drain on potential resources.

#### FGSM

The second method that showed promise was Fast Gradient Sign Method (FGSM). This method had been implemented by the xpl[AI]ned team for use in automotive AI implementations, but we were interested in the possibilities for its use within the air interface. FGSM is an adversarial type method, which attempts to generate adversarial inputs that will force a neural network to mis-classify or make wrong decisions. These inputs are generated by applying small but intense and 'worst case' perturbations to the dataset. For FGSM specifically, the necessary perturbations are found by working backwards through the model and finding the signed gradients that minimise the loss function. By taking these signed gradients, multiplying them by a small value, and adding them to the original input, FGSM generates the adversarial inputs[14]. The equation for FGSM can be seen in (3.1). X is the input, Epsilon is the small value that is added to the input,$\nabla$ is the gradi-

ent of the loss function with respect to X, L is the loss function and Y is the data label for X.

$$X_{Adversarial} = X + \varepsilon \cdot sign(\nabla_x L(X, Y)) \tag{3.1}$$

This method of essentially applying 'gradient ascent' means that these inputs force the model into mistakes for which it could still report a high level of confidence, if its adversarial robustness is low. The level to which these adversarial inputs reduce the model's confidence in its output can be measured and analysed to determine how resistant the model is to these types of attacks and therefore how adversarially robust it is. As well as this, the inputs themselves can be useful, as collecting a lot of them and then identifying patterns within them can lead to identifying key areas in which the model is vulnerable or insufficiently prepared to deal with.

In the context of this project, similarly to Monte Carlo, analysing how the BER was affected across different $E_b/N_0$ levels would be useful to determine the magnitude of the impact of adversarial perturbation through the form of some sort of interference. However, as the purpose of FGSM is to force the model into making 'confident' mistakes, it would be useful to go one step deeper, and examine the LLRs at the output of the model to determine how much the adversarial inputs affected the model's decision confidence. We expect to see large drops in confidence for a robust model, and no changes for one that has a weak adversarial robustness. As well as this, the adversarial inputs themselves could be analysed in terms of amplitude or phase, identifying patterns in them which could then be linked to real world situations or causes. With this, it could be possible to build a picture about the expected performance of the model depending on the outside factors of its operating environment. However, the inherent complexity of the models and the context of the data means that identifying these patterns and extracting useful information from them could be a significant undertaking.

#### SHAP

The possibility also existed to implement SHapley Additive exPlanations (SHAP) testing as a stretch goal given enough time in the project development cycle. SHAP is an explainability method, that aims to increase the transparency and interpretability of a model. It is based on the Shapley value, which is a concept developed from game theory that is used to quantify the average of the marginal contributions to a task from all actors across all permutations. When applied to the field of machine learning, the task is the minimisation of the loss function, and the actors are the features that contribute to the output of the model. In this way, an understanding can be gained about the models input, and what features of said input are the most responsible for correct or incorrect predictions by the model. As a method, SHAP involves the calculation and/or estimation of these values by

various means. These values then provide the user with global interpretability, showing the overall contribution of each feature on a correct prediction, and local interpretability, where each individual instance has its own SHAP values to help with transparency.

The specific version of SHAP considered for implementation was called the Kernel Explainer [10]. A subclass within the overall SHAP library, the Kernel explainer works by putting together a weighted linear regression from the base data labels, the output of the model and the function that generates the output. The SHAP values are then calculated based on the method for calculating the original Shapley values and the coefficients of the linear regression [20]. In our case, the Kernel explainer would ideally allow us to pinpoint the key features that lead to predictions of a zero or a one in the input of our models.

#### CEM

Another explainability method posed as a stretch goal was Contrastive Explanation Method (CEM). Similar to SHAP in its goals but producing simpler results, CEM provides seeks to provide contrastive explanations. A common analogy used to describe these comes from medicine, where to prove someone has the flu a doctor must show they have a temperature, a cough, but also that they do not have chills, confirming they have a flu and not pneumonia [9]. Two main types of information are provided by applying CEM. Pertinent Positives (PP) finds the features within data that contribute the most to a correct prediction, ergo, those having the highest feature weight. On the other hand, Pertinent Negatives (PN) find the features that have the largest negative output on the prediction, and ideally should be absent from an ideal prediction instance. While this is less overall information than methods like SHAP, it is provided in a more understandable way that then allows for further investigation.

Due to the binary nature of the classification in both models, the identification of PNs could potentially be very useful in terms understanding the key features that influence the choice of the model. If these features can be identified and isolated, they can be used to identify situations in which the model makes errors due to the presence of these PNs, and also to improve the functionality of the model in the future.

## 3.2 xpl[AI]ned and the Testing Environment

As mentioned previously, a key part of this project is the use of Keysight's new xpl[AI]ned testing framework. Structurally, it consists of three main sections. The first of these is the data-loader, which is loaded with the required input data for the given model/test, in the correct format and specifying the amount of said data to be used for the test. The

parameters for the model under test are loaded into the model-loader, along with the location of the saved weights for the model and the type of model being loaded. Finally, the model-loader and data-loader are passed to the method call, alongside any variables required for the models call function and any other data required by the method. Each of these three components inherit their basic characteristics from templates developed by the xpl[AI]ned team. A diagram of the basic structure of xpl[AI]ned can be seen in Fig. 3.1

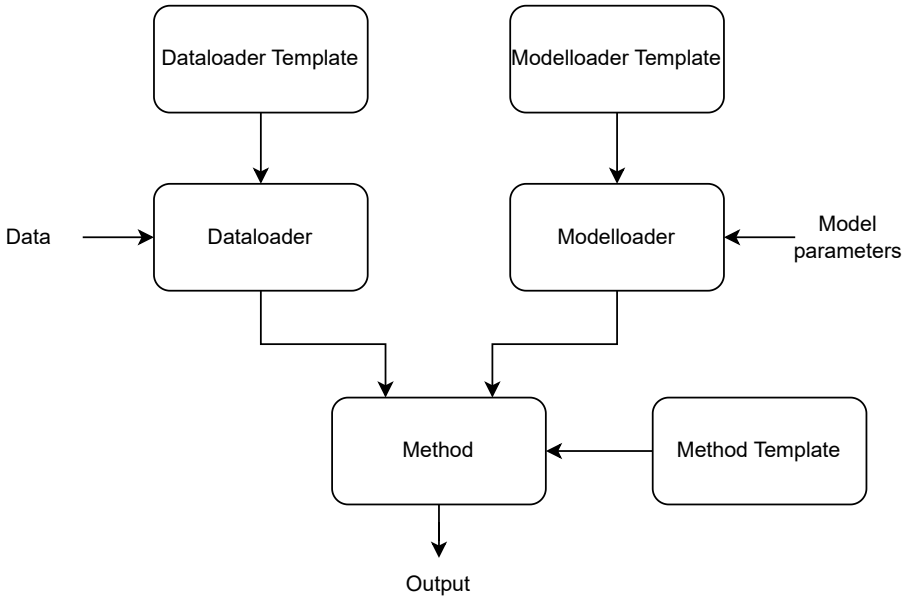



**Figure 3.1:** A block diagram showing the structure and functionality of the xpl[AI]ned testing framework. The data for a given test and the model parameters are loaded into their respective loaders, which inherit their baseline characteristics from the basic templates within xpl[AI]ned. These loaders are then passed to the method, which itself takes its baseline characteristics from a method template. The output of this method is the results for the given test

The main work with regards to xpl[AI]ned required during the testing phase was mating the chosen models and their context specific requirements to the framework. This included changes in the input format for the models/methods, modifying internal code relating to the calculation of certain metrics and changing the type of output that the methods generated. The specific changes are described in Sections 3.3 and 3.4 where relevant. When these modifications were made, the testing infrastructure for any given method or test was built around the structure of xpl[AI]ned, with loops used to sweep variables through ranges of values. These variables included ones unique to each testing method, but also the $E_b/N_0$ as the xpl[AI]ned methods were set up to do tests for one $E_b/N_0$ level at a time.

Early on in the project's development cycle, it was decided that the bulk of the development of the testing infrastructure should be done through the medium of Jupyter Note-

books. The reason for this was twofold. Firstly, it was recommended by the xpl[AI]ned team, as in their experience this was the most efficient and flexible way to work within their framework. As well as this, the powerful computer provided by Keysight for large scale testing was most easily accessible through a Jupyter server. This meant that if Jupyter Notebooks were used, the testing infrastructure could be developed within xpl[AI]ned and then simply uploaded to the testing machine with minimal changes required. Each model had its own notebook for each testing method. The notebook contained the necessary code to build and load the model, as well as the required code to interface the model with xpl[AI]ned and run a variety of different tests. These notebooks shared similar structures due to the consistency in the design of xpl[AI]ned, with each testing method instantiated in essentially the same way, limiting the amount of unique infrastructure needed for each model-method combination.

## 3.3 Monte Carlo Dropout

### 3.3.1 Methodology and Implementation

The underlying theory behind this method is described in Section 3.1. The work needed to implement Monte Carlo dropout with xpl[AI]ned was fairly straightforward. The modifications made to both models described in Section 2.2 meant that the formats for both the input and output were already correct. To add the capability to modify dropout efficiently, a new parameter was added to the model call function, passing a number between 0 and 1 which corresponded to the desired dropout rate. The dropout layers themselves were manually added/changed within each model's structure for each test, using the dropout parameter as their input. The method call itself would provide as an output the mean BER for the baseline runs, the mean BER for the model runs done with dropout introduced, and the difference between the two values

The Monte Carlo test for both models would be looking at two main factors: The model's response to dropout being introduced, and how that response changed depending on where the dropout was introduced. Each model had multiple layers between which dropout could occur. For each location, the dropout rate would be swept from 0% to 80% in increments of 10%, and at each of these intervals the bit error rate would be calculated for a range of $E_b/N_0$ values. This range would go from -20 dB, as neither model experienced a relevant change in bit error rate performance with a smaller $E_b/N_0$ value, to 20 dB, as both models reached their minimum bit error rate with a smaller $E_b/N_0$ value. The sweep would feature 10,000 codewords at each $E_b/N_0$ level, with 10 baseline (no dropout) and 10 dropout runs done for each sample. An average value would be calculated for each of these, and the final output would be the difference between these two averages, or the average impact on the BER due to induced dropout. This dataset would give de-

tailed information as to how the loss of a differing number of neurons affected the model performance with reference to its expected performance at a given $E_b/N_0$ level. When introduced at input or output layers, we would expect severe degradation as key information with no form of redundancy is lost. The expected result for hidden layers is unknown and depends on the structure of both models and how rightly sized they are in terms of amount of neurons.

### 3.3.2 Model 1: Neural Receiver

The core of the model consists of 4 residual blocks, and within each of these blocks are 2 convolutional layers. There were also 2 additional convolutional layers acting as input and output layers. This mean dropout could be introduced at 10 distinct locations. This model proved to be extremely heavy even for the powerful computer that we were using to run the tests, and the number of samples had to be decreased to 3000 to maintain consistency and avoid out-of-memory errors occurring. This was still a sizeable amount of data especially considering multiple dropout tests would be performed on each input, so the risk of loss of statistical significance was minimal.

The results for the input and output convolutional layers show a high susceptibility to dropout as seen in Fig 3.2 and 3.3. This makes sense, as denying the model access to initial input information and cutting out information generated by the hidden layers right before the output will have an obvious impact on the BER. In both cases, the model is essentially rendered useless at the higher dropout rates, reaching BERs of 0.5.

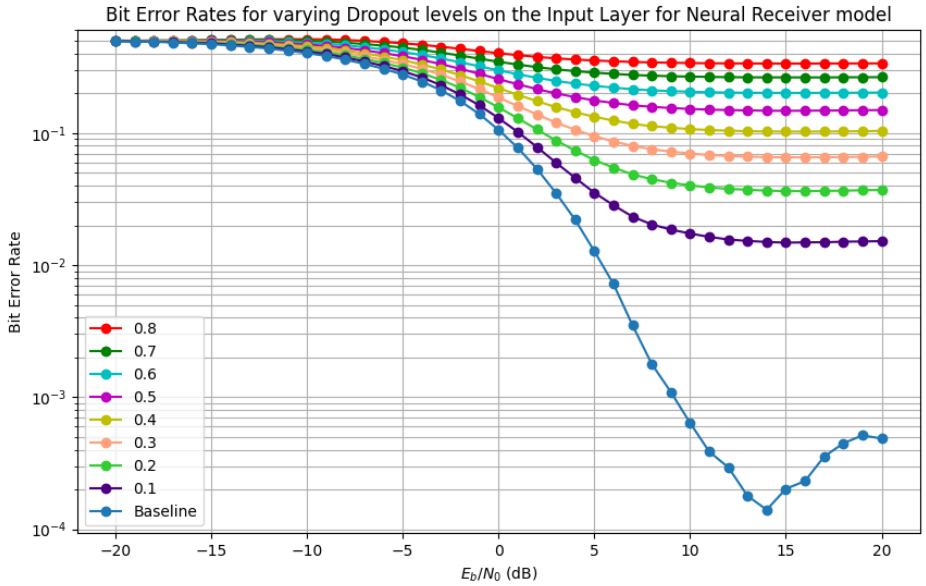



**Figure 3.2:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its input layer. Dropout here appears to have a significant negative effect on the performance, with the higher rates essentially rendering the model unusable at all $E_b/N_0$ levels.

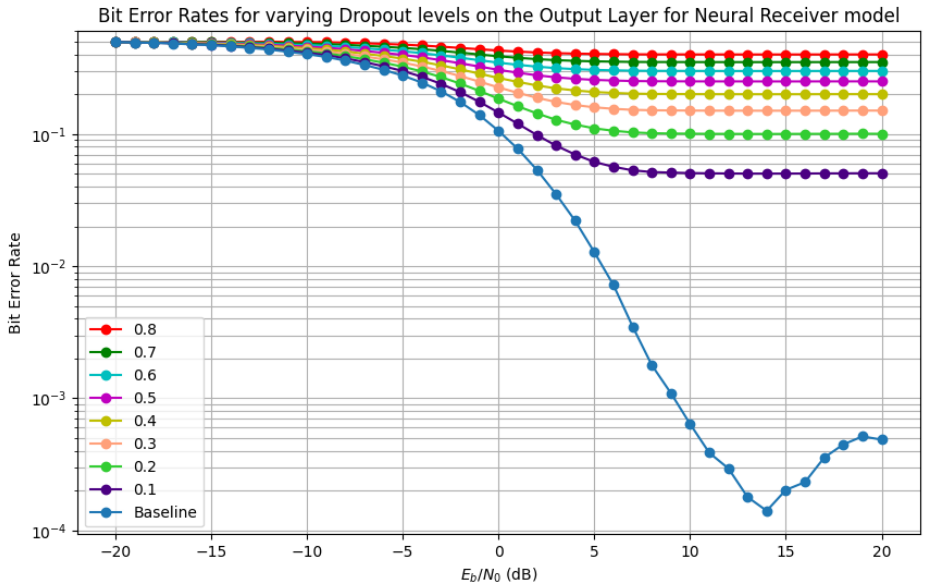



**Figure 3.3:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its output layer. Dropout here appears to have a significant negative effect on the performance, with the higher rates essentially rendering the model unusable at all $E_b/N_0$ levels.

The overall results for all the hidden layers in the residual blocks follow a distinct pat-

tern: these layers are relatively resistant to dropout, but the negative impact increases the further away from the input layer the dropout is applied. This is expected, as the further down the chain the dropout is applied the fewer following layers there are to correct for any mistakes that occur. Consistently across all hidden layers, anywhere up to 50% dropout has an effect on the BER of less than 0.01. The absolute maximum impact in the hidden layers occurs in the eighth, where at 80% dropout a maximum increase of around 0.06 is observed between 10 and 15dB. The graphs for the first, fifth and eighth layer are shown in Figs. 3.4, 3.5 and 3.6. Fig. 3.7 shows the BER at all $E_b/N_0$ levels for every hidden layer at a dropout rate of 0.5, showing how the effect of dropout increases as it is applied deeper into the model. The full set of graphs and data are shown in Appendix A.2

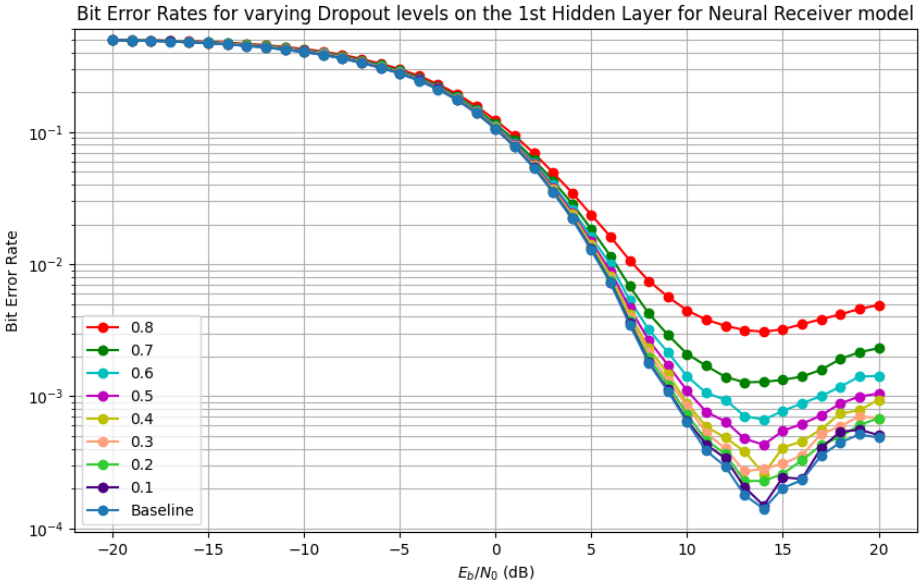



**Figure 3.4:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its first hidden layer. Being the first of 8 other hidden layers, dropout has the least effect here, with 80% dropout having a maximum effect on the BER of around 0.003

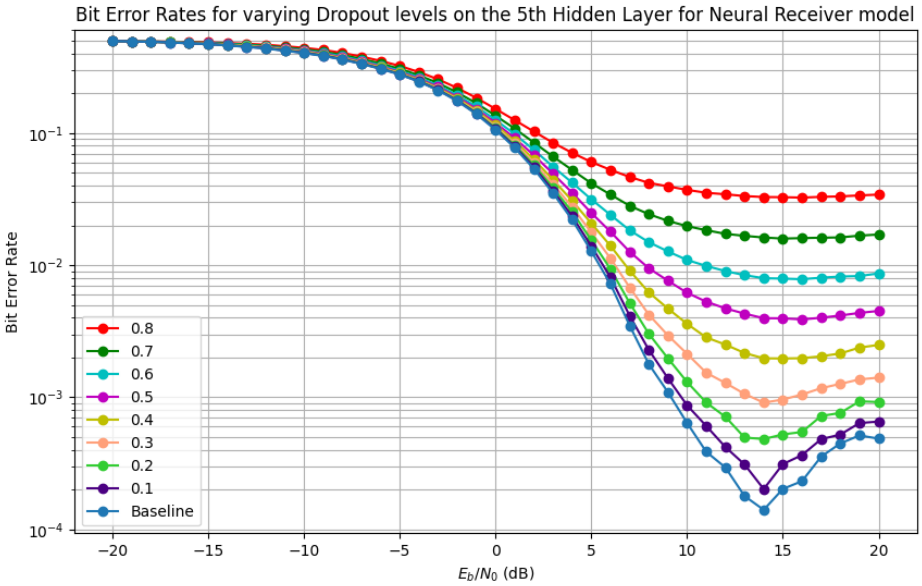



**Figure 3.5:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its fifth hidden layer. Being around halfway through the model structure, the impact of dropout here is more significant than in the first layer, with a maximum impact of 0.03 at 80% dropout.

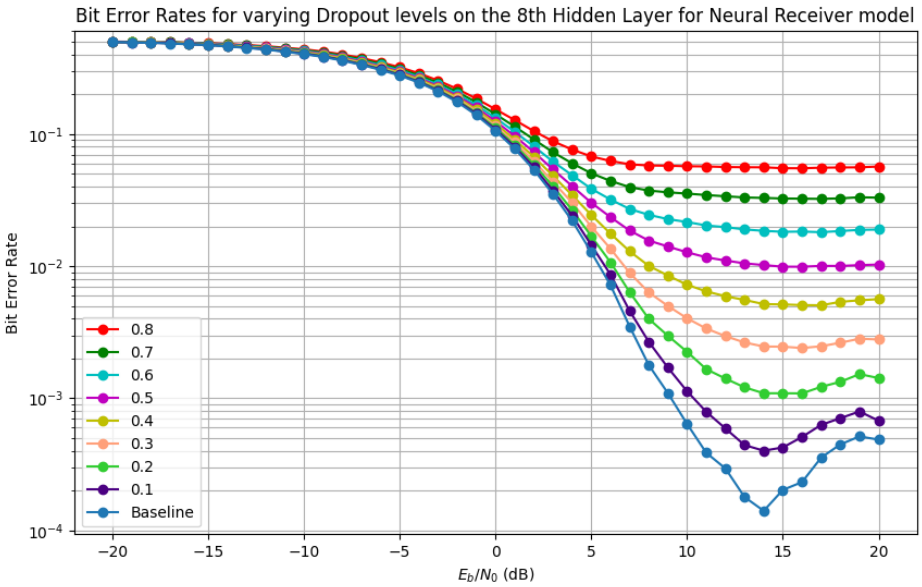



**Figure 3.6:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to the last hidden layer. This layer is the most affected out of all of the hidden layers, with a maximum impact of around 0.06 at 80% dropout.

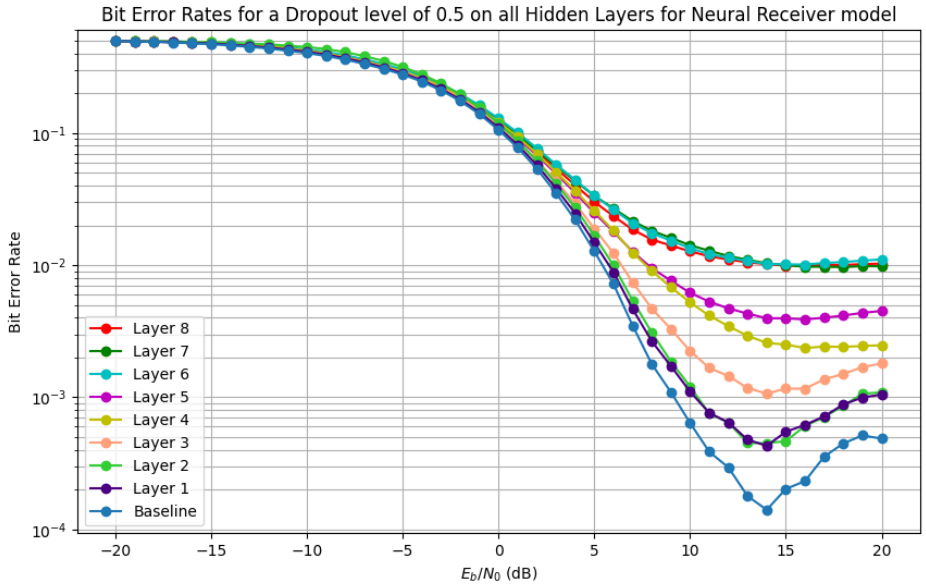



**Figure 3.7:** Bit Error Rate curves for the Neural Receiver model at a dropout rate of 0.5 applied to each hidden layer. The increasing degradation of performance as dropout is applied closer to the output is clear from this graph, although at this level the BER at layers 6-8 appears to plateau at a similar value.

### 3.3.3 Model 2: End to End Autoencoder

While this model technically consists of 2 independently trained components, only one of these actually ends up being a neural network, that being the Receiver. On the transmitter side, the constellation and the bit labelling are optimised during the training process, but once complete both of these are set in stone, and do not act as AI models. This means that dropout can only be performed on the neural demapper side. Following the methodology for Monte Carlo dropout, the combined $E_b/N_0$/Dropout sweep was performed for each of the three layers that comprise the neural demapper.

The results for the input layer can be seen in Fig. 3.8. As expected and similarly to the Neural Receiver model, dropout applied to the input layer has a significant impact on performance.

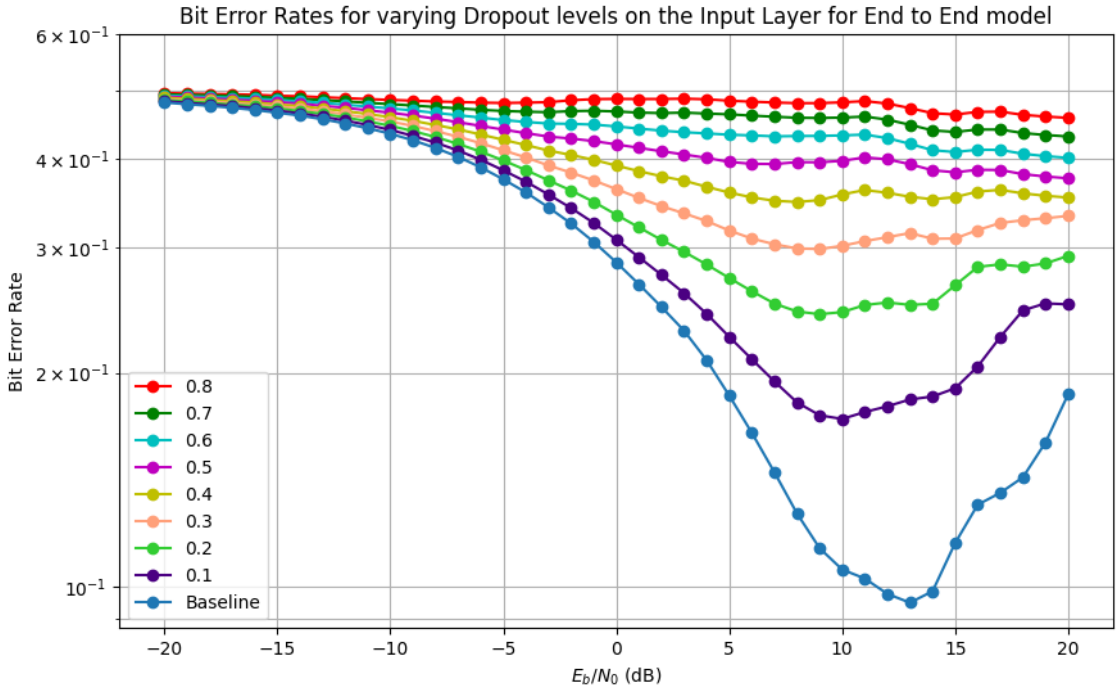



**Figure 3.8:** Bit Error Rate curves for the End to End model with different dropout rates applied to its input layer. Dropout here appears to have a significant negative effect on the performance, with the higher rates essentially rendering the model unusable at all $E_b/N_0$ levels.

The lower $E_b/N_0$ values, where the bit error rate is high to begin with show very little change due to dropout as expected. Once the $E_b/N_0$ reaches around 0dB, the effects of dropout start becoming significant, exceeding an increase of 0.1 in the BER once 40% of the neurons are missing. This trend continues, and at 5dB and above the effects are increasingly drastic, especially considering the low base BER at these $E_b/N_0$ levels. At the higher performance levels from 10dB onward, the BER change is almost 0.5 for the higher dropout rates, rendering the model essentially useless.

The effects on the performance due to dropout when applied to the hidden layer are noticeably reduced in comparison, but appear to be more significant than the results observed for Neural Receiver. This is most likely due to the network being shallower, that is to say having fewer layers and therefore less protection against dropout. The results can be seen in Fig. 3.9.

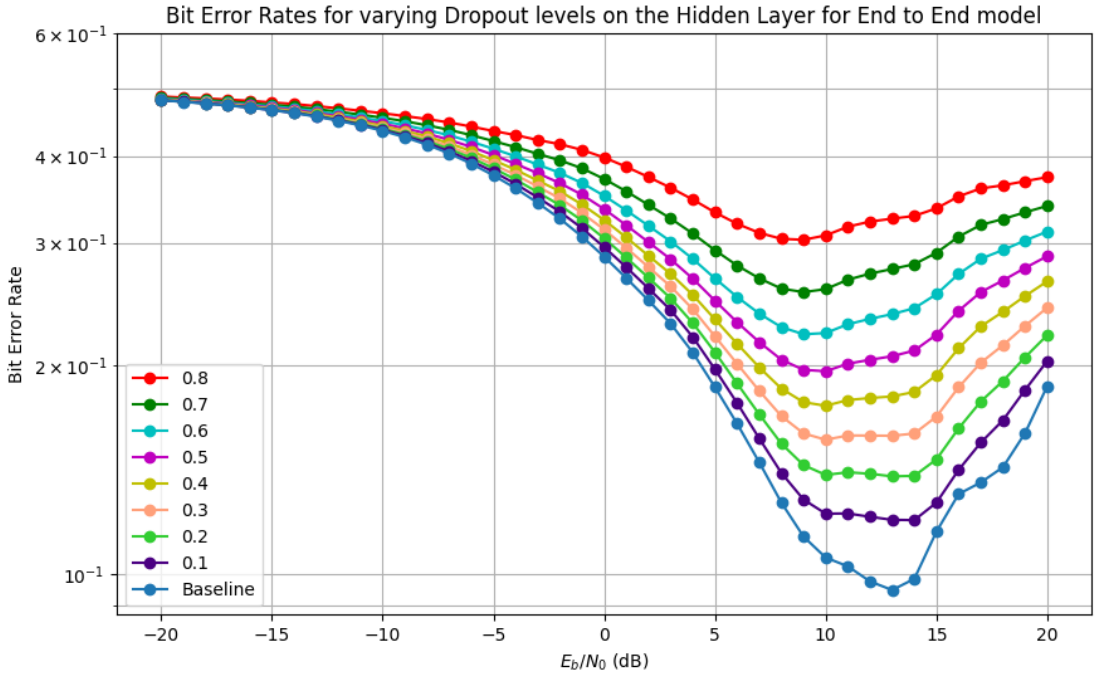



**Figure 3.9:** Bit Error Rate curves for the End to End model with different dropout rates applied to its first hidden layer. The effect of dropout here is reduced compared to the input layer, although the impact around the models optimum range of between 10 and 15dB is still significant, especially when compared to Neural Receiver.

Once again, the lower $E_b/N_0$ levels are barely affected by the introduction of dropout. The effects between -5dB to 10dB have been decreased compared to the input layer, with changes of less than 0.1 for dropout rates of up to 50%. However, in the optimal region of between 10 and 15dB, 40% dropout and above have an increase exceeding 0.1, with a maximum of around 0.3 for 80%, which still shows an untenable BER of 0.4 and above consistently.

The effects of dropout on the output layer are less than the ones seen for the first layer, but more significant than the effects on the hidden layer. Up to 20% dropout remains below a 0.2 BER in the region between 10 and 15dB, but performance at higher dropout rates is significantly degraded, with 70 and 80% dropout making the model unusable due to BERs of 0.4 and above. These results can be seen in Figure 3.10, and full tables of data for all three dropout tests for the End to End model can be seen in Appendix A.3

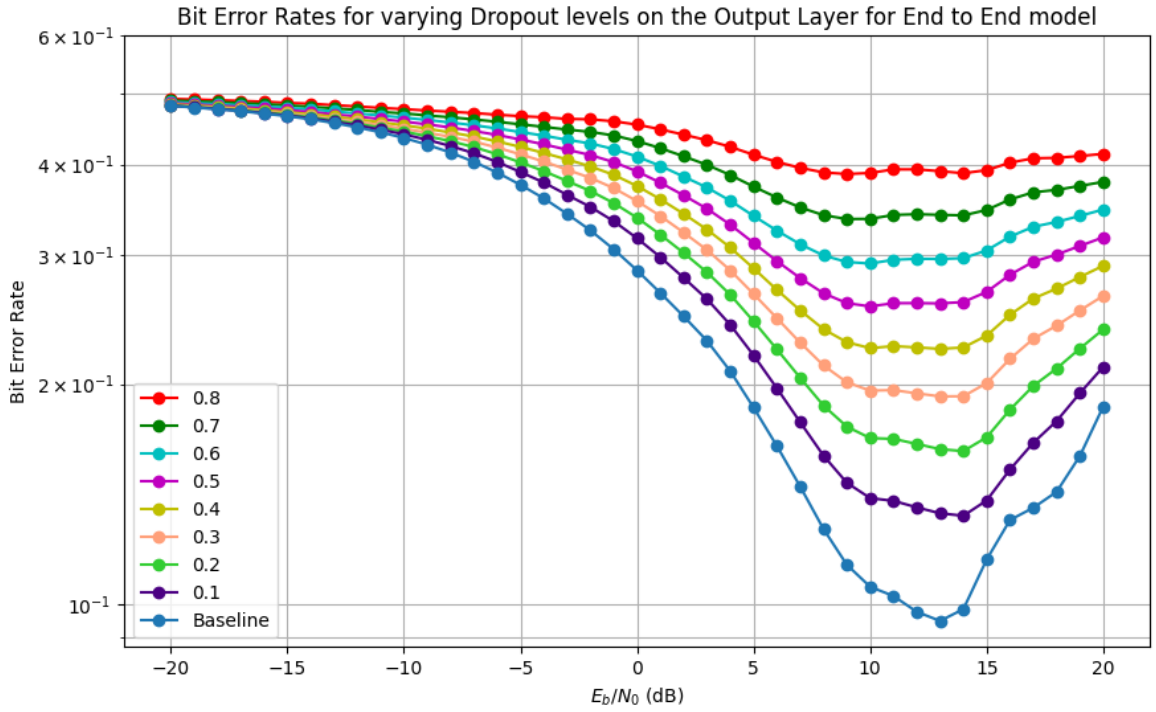



**Figure 3.10:** Bit Error Rate curves for the End to End model with different dropout rates applied to its output layer. Much like its input layer, the impact here is significant, albeit still lesser overall

### 3.3.4 Initial Conclusions

The generated results for Monte Carlo Dropout fit with what we expected to see. In both models, we see the input and output layers being disproportionately affected. This makes sense, as denying the model input data or withholding its outputs will obviously have an adverse affect on its performance. When looking at the hidden layers, the End to End model saw worse performance in this area than the Neural Receiver, owing to the fact that it only has 1 small hidden layer as opposed to Neural Receiver's 8. While the quantification may be of some use, the knowledge that the loss of any amount of input leads to degraded performance is not groundbreaking for this context. On the other hand, the response to dropout in the hidden layers tells us that the End to End model is probably as small as it could feasibly be, while the Neural Receiver could have some design space for optimisation considering how relatively unaffected its performance was by dropout in its hidden layers. This is compounded when looking at the difference in the minimum BER that both models reach, with Neural Receiver having more room for error.

## 3.4 FGSM

### 3.4.1 Methodology and Implementation

The underlying theory behind this method is described in Section 3.1. The methodology and process required to get FGSM working for both of these models was a lot more complicated than Monte Carlo dropout. Whereas Monte Carlo dropout was able to fit directly over the top of both models after a minimal amount of work on the backend of the xpl[AI]ned framework, FGSM required a full rework of the method and both loaders, as well as a significant deconstruction of the models themselves.

The key issue was the interaction between the structure of the models, specifically relating to the surrounding infrastructure that makes the models work, and the nature of the types of operations that FGSM needs to perform. As FGSM steps backwards through the model to calculate the gradients and eventually apply the perturbations, it needs an understanding of the format of the input and output in order to be able to generate a valid adversarial input. Initial efforts to apply FGSM on the full structure of both models returned many issues, including the fact that the method was apparently unable to determine the necessary gradients correctly. This issue plagued the project for a while until an analysis of xpl[AI]neds implementation of the method and the structure of both models revealed the cause of the issues.

In this project, we have referred to both of the structures developed by Sionna as models when referring to them for the sake of brevity, but it is worth remembering that in both cases the model constitutes a small part of said structure, surrounded by necessary signal processing infrastructure and other data processing methods that are necessary for the model to do its job correctly. It is this structure that caused the most issues, as attempting to apply FGSM to generate adversarial inputs in the form of the initial information bits or codewords failed as neither of these were the actual input to the model. FGSM had to be applied to just the neural network portion of the model itself, which in both cases meant the actual inputs were received samples/resource grid after the effects of the channel and the outputs were the generated LLRs. With this in mind, the FGSM implementation would act on only the neural networks themselves, but the surrounding signal/data processing steps would need to be included in the methods in order to allow for them to be used on the front end with input/output data that made sense. This significantly increased the complexity of the method, requiring multiple signal processing steps to be performed each time it was called. A number of variables that needed to be passed upon initiation for both the model and the method, including the epsilon value and the loss object. With all these changes implemented, the output of the method call would be the mean BER as a result of both the original and adversarial inputs, the average change in decision confidence due to the adversarial inputs, as well as both sets of model inputs.

A few different types of test would be run through the means of FGSM. Firstly, similarly to Monte Carlo, the Bit Error Rate response to the adversarial inputs would be investigated. This would be done by sweeping through a range of perturbation/epsilon values(0.1 to 0.8) for a range of $E_b/N_0$ values, once again from -20 to 20 dB (Refer to equation (3.1) as a reference). As well as this, separate sweeps would be performed to investigate the changes in decision confidence due to the adversarial inputs. FGSM aims to generate inputs that will have the model making mistakes but retaining a high level of confidence, so these results can help to show how resistant the models are to these types of attack, or how easy they would be to detect. Due to the differences in complexity of their design, we expected to see Neural Receiver perform better than End to End in this regard.

The adversarial inputs themselves are also of interest, as if they can be extracted and compared to the baseline inputs for the model, it is possible to deduce patterns in the data that lead the models to make mistakes. These inputs would be in the form of frequency domain samples in both cases, and as such the magnitudes of both sets of inputs could be plotted in a spectrum and the differences between them at different epsilon levels can be analysed. Any trend found here could then be investigated further to isolate potential real world causal effects behind the degraded performance.

### 3.4.2 Model 1: Neural Receiver

As mentioned in the methodology section, implementing FGSM for the Neural Receiver model required a lot of work, including recreating the pre-model signal processing steps within the testing method itself. An unexpected downside of this was that the model, a significantly heavy one already that used a lot of memory during operation, became even heavier due to the increased amount of temporary/intermediate files that it was storing. This meant that the maximum possible batch size was significantly reduced down to 500, potentially reducing the the statistical significance and reliability of the results.

The first test investigated the impact of the adversarial inputs on the BER, with the results shown in Fig. 3.11. The impact of the adversarial inputs increases as the $E_b/N_0$ increases, becoming noticeable at around 0dB. The higher epsilon values(0.5-0.8) cause the BER to plateau in the range of 0.1 to 0.2, while the lower ones have a lower but still noticeable effect that increases as the $E_b/N_0$ increases.

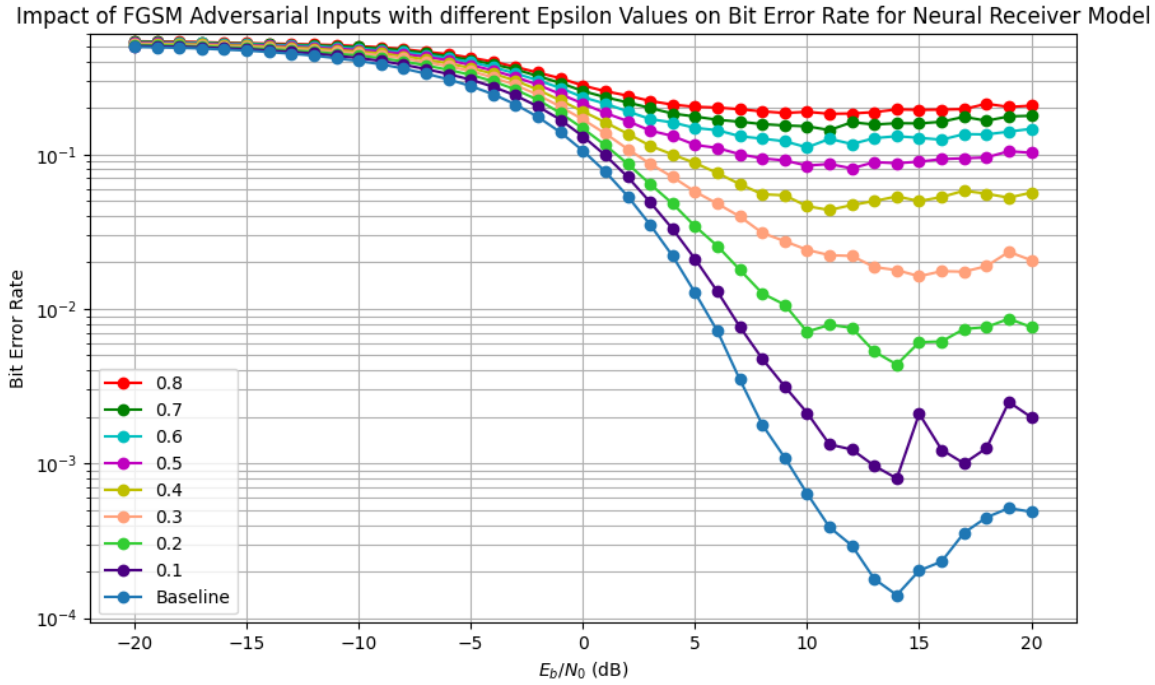



**Figure 3.11:** Bit Error Rate curves for the Neural Receiver model due to adversarial inputs of different epsilon values. The results are as expected, with the higher epsilon values causing greater impact, especially in the region of 5 to 15dB where the model can be said to be operating optimally

With the stated goal of FGSM being to force the model to make errors with high confidence, the more useful measure is the change of the model's decision confidence due to FGSM which can be seen in Fig. 3.12. From this figure, we can see that as the $E_b/N_0$ increases, the drops in confidence at each epsilon level also increase, with the relationship at each $E_b/N_0$ level being approximately linear. The most important piece of information to glean is that the drops in confidence are noticeable even for the lower epsilon values from 0dB onward, which coincides with where the significant degradation of performance due to the adversarial inputs begins(Fig. 3.11).

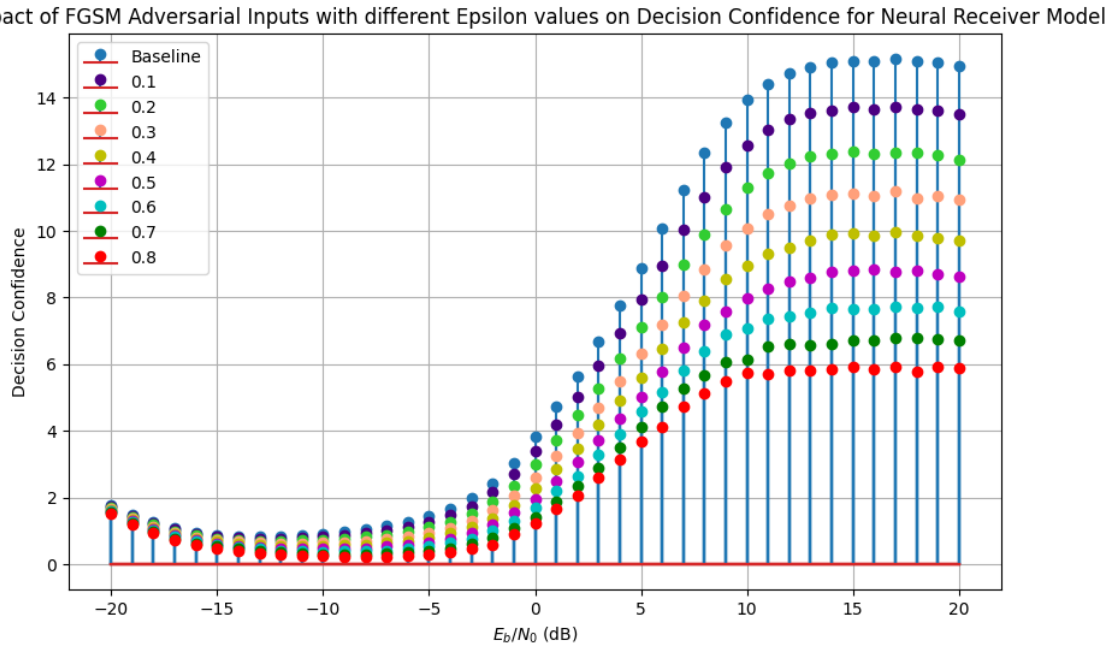



**Figure 3.12:** New Decision Confidence values for the Neural Receiver model due to adversarial inputs with different epsilon values. This is represented by subtracting the average change in magnitude for the LLRs from the baseline Decision Confidence. The impacts of the adversarial inputs are noticeable even for the low epsilon values, meaning that even if the models input data was perturbed it would be detectable, implying the model is relatively robust.

To better quantify the impact to both the BER and the confidence, we can look at how both look in terms of percentages. Fig. 3.13 shows the BER increase per epsilon value at each $E_b/N_0$ level as a percentage of the original BER value, with Fig 3.14 showing the same but for the decreases in confidence. In Fig. 3.13 it is evident how significant the degradation of performance is for the model, with even FGSM inputs with epsilon values of 0.1 reaching up to a 1000% reduction in performance at 15dB. At the same $E_b/N_0$ level, epsilon values of 0.6, 0.7 and 0.8 show degradations of 100,000%. This is to be expected due to the exceptionally low base BER values exhibited by this model. This drop in performance is severe, but when combined with the results in Fig. 3.14 we can see that this model actually has a good amount of adversarial robustness. The drops in confidence are consistent and noticeable, with even epsilon values of 0.1 conferring a 10% drop. Perhaps the only downside seems to be that the drops in confidence do not increase with the $E_b/N_0$ as the performance degrades, but the consistency of the reduction means that this model should still be very resistant to adversarial perturbations.

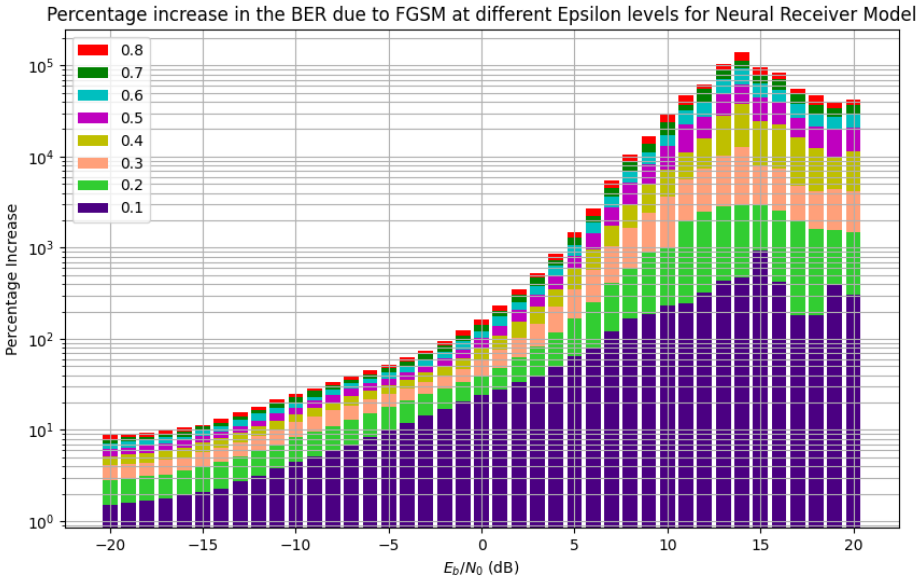



**Figure 3.13:** The increase in BER as a percentage of the baseline value due to FGSM at different epsilon values for the Neural Receiver model. This graph quantifies the severity of the loss of performance for this model, with even epsilon values of 0.1 reaching an increase of 1000%

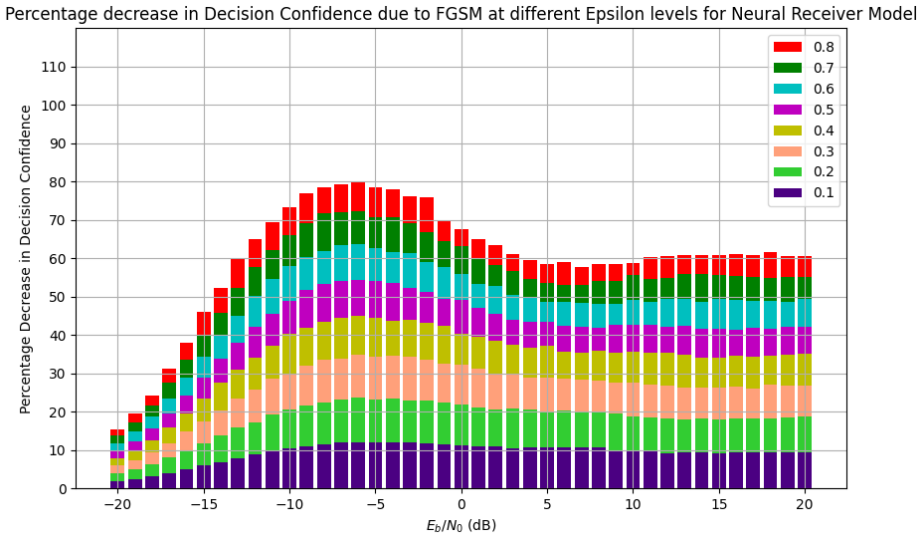



**Figure 3.14:** The decrease in Decision Confidence as a percentage of the baseline value due to FGSM at different epsilon values for the Neural Receiver model. The models adversarial robustness is clearest here, as there are consistent and marked decreases in confidence at all epsilon values and $E_b/N_0$ levels.

### Adversarial Input Analysis

By analysing the characteristics of the baseline inputs to the model and the modified adversarial versions and the differences between them, it was theoretically possible to look for patterns to try and identify what conditions caused the model to have decreased performance. The xpl[AI]ned FGSM implementation could output both the adversarial and unaltered inputs for any given run of the method. Individual frequencies of an OFDM symbol could be separated, the magnitudes of which calculated and these plotted into a spectrum across all of the subcarriers. The difference between the amplitudes of both the original and adversarial inputs could then be found and also plotted, with this entire process done for multiple epsilon values. Examples can be seen in Figs. 3.15, 3.16 and 3.17. A consistent pattern emerged across these plots, that being that a lot of the adversarial inputs seemed to on average have a consistently larger amplitude than the original ones. Table 3.1 shows the average difference between the two sets over 1000 codewords. This pattern has implications about aspects of the models structure, and the steps taken to prove those implications are described in Section 3.5.

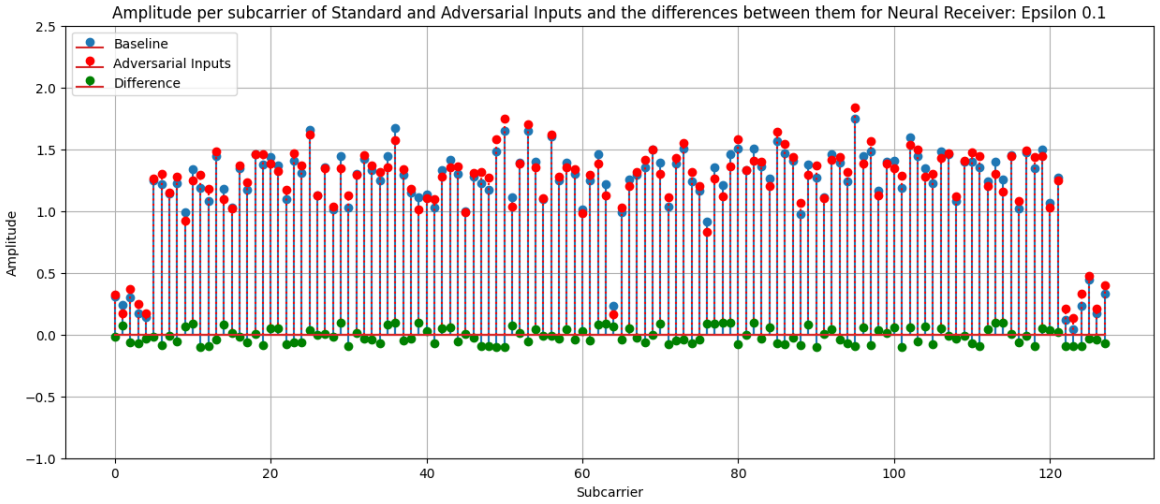



**Figure 3.15:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.1 and $E_b/N_0$ of 13dB for the Neural Receiver model. Even at this low level, the tendency for the adversarial inputs to be of a greater amplitude than the original ones can be observed.

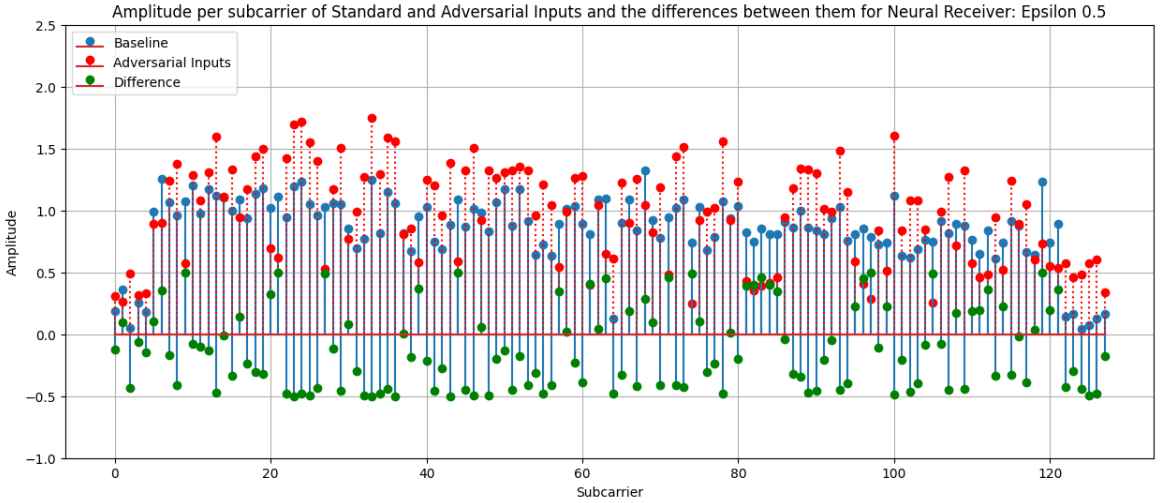



**Figure 3.16:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.5 and $E_b/N_0$ of 13dB for the Neural Receiver model. The tendency for adversarial inputs to be of a greater amplitude than the original ones is clearer from this graph.

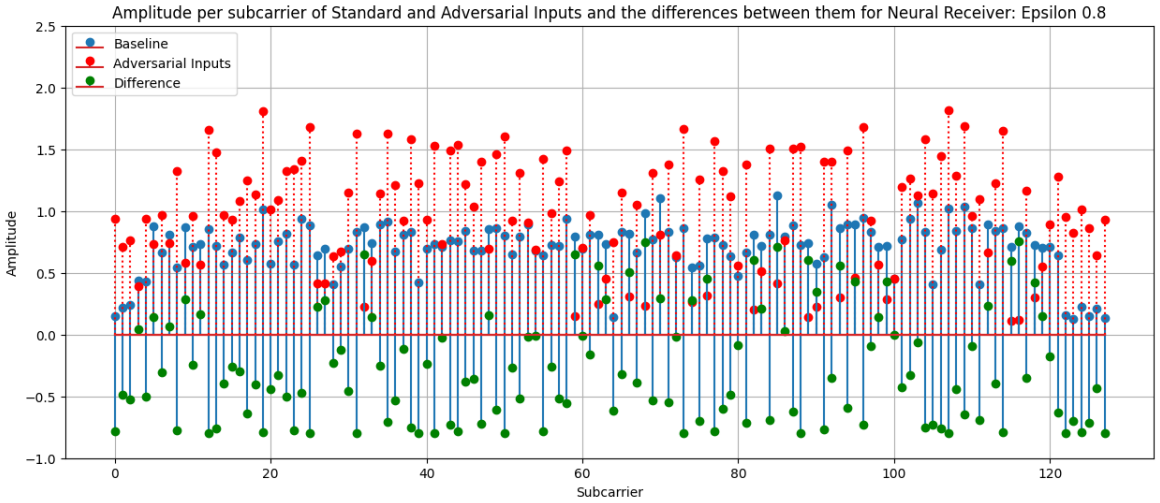



**Figure 3.17:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.8 and $E_b/N_0$ of 13dB for the Neural Receiver model. The tendency for adversarial inputs to be of a greater amplitude than the original ones is very clear from this graph.

**Table 3.1:** Average difference in amplitude between original and adversarial inputs for Neural Receiver model. On average, the adversarial inputs have a greater amplitude, and this difference increases with the epsilon value

| Epsilon Value | Average Difference |
|---|---|
| 0.1 | -0.00195 |
| 0.2 | -0.01536 |
| 0.3 | -0.04729 |
| 0.4 | -0.07889 |
| 0.5 | -0.09274 |
| 0.6 | -0.13509 |
| 0.7 | -0.20815 |
| 0.8 | -0.23711 |

The modulation type used by the Neural Receiver is Quadrature Phase Shift Keying (QPSK). In principle, this form of modulation should be robust to changes in amplitude (despite the evidence seen thus far) while being sensitive to changes in phase. With this in mind, it was deemed worth investigating the difference in phase between the original and adversarial inputs. However, across multiple attempts and large scale data gathering efforts, no consistent pattern was able to be elucidated in the phase angle results. The phase angle between the two sets of data would differ but no consistent trend for this difference could be found except for a slight tendency to increase with the epsilon value, and even this was not observed every time. This is the opposite of what should be expected for QPSK, and could be earmarked for future investigation.

### 3.4.3 Model 2: End to End Autoencoder

The End to End model, being a much smaller and less complex implementation than the Neural Receiver, did not suffer the same issues with the increase in memory requirement despite also requiring a significant amount of work to implement. As such all of its tests were run with the full batch size of 10.000 codewords.

The first test was the basic bit error sweep, with the results seen in Fig. 3.18. The impacts of the adversarial inputs are lesser relatively speaking compared to the ones seen for the Neural Receiver model, but the effect on performance is still significant with BER changes of around 0.03 in the optimum operating range between 5 and 15dB. The real impact however can be seen in Fig. 3.19, where it is evident that these performance degradations do not come with consistent drops in confidence, especially in the models optimum operating range.

Once again to better quantify these effects, Fig 3.20 and 3.21 show the changes to the

BER and confidence as a percentage of their baseline value. Looking at Fig 3.20, the relative degradation of performance for the End to End model is significantly reduced compared to Neural Receiver, with epsilon values of 0.1 never causing an impact greater than 20%, and an overall maximum impact of around 45% from an epsilon value of 0.8 at 0dB. Once again, this is to be expected due to the significantly poorer baseline performance of this model compared to the Neural Receiver model. This relative advantage however is deceptive, as Fig. 3.21 shows just how much this model lacks adversarial robustness. In between -10 and 0dB, a noticeable decrease to the confidence can be seen, but as the $E_b/N_0$ increases into the optimum operating (and far more relevant) range for the model, the reduction in confidence becomes significantly smaller. At 10dB and above, even epsilon values of 0.8 cause a change of 10% or less , with the smaller epsilon values cause negligible changes. This model is therefore extremely susceptible to adversarial perturbations, as while the relative effects to its performance are smaller, the lack of significant change to its confidence means that these performance degradations would not be easily detected.

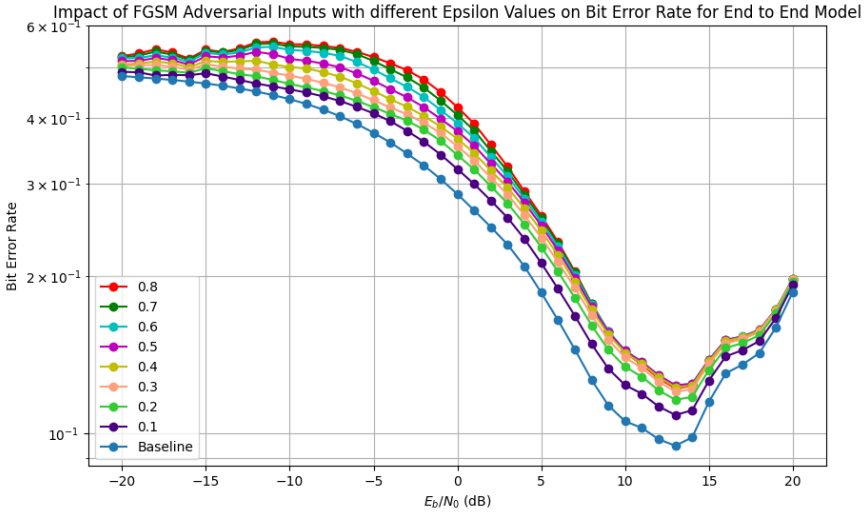



**Figure 3.18:** Bit Error Rate curves for the End to End model showing the effect of adversarial inputs with different epsilon values. The results show a very consistent increase for each epsilon value, with some convergence at the higher values starting around 5dB.

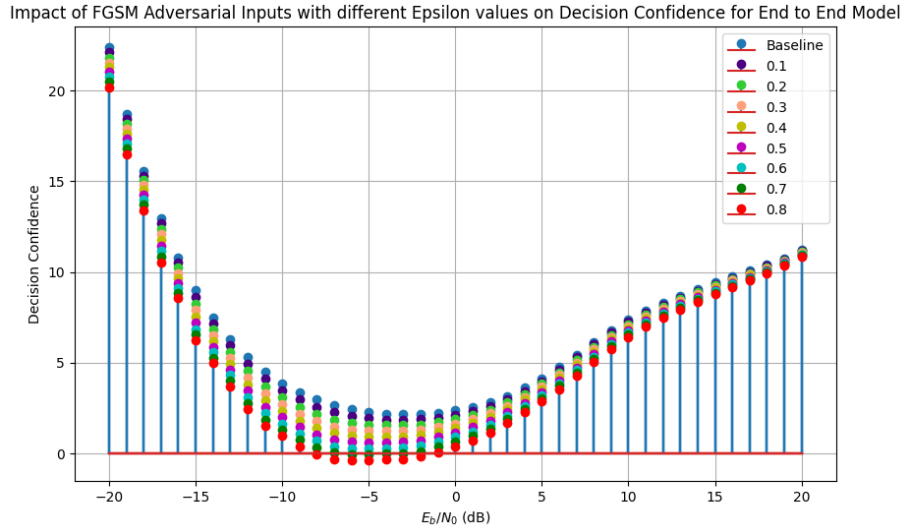



**Figure 3.19:** Effect on Decision Confidence for the End to End model due to adversarial inputs with different epsilon values. This is represented by subtracting the average change in magnitude for the LLRs due to FGSM from the baseline Decision Confidence. The impacts of the adversarial inputs are small for all but the largest epsilon values, implying that this model is not robust.

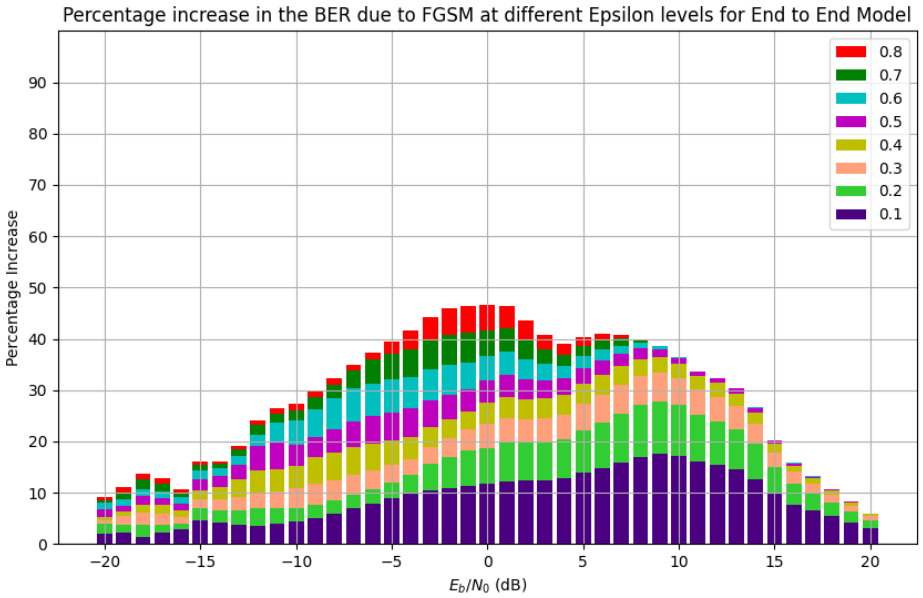



**Figure 3.20:** The increase in BER as a percentage of the baseline value due to FGSM at different epsilon values for the End to End model. This graph shows how the performance degradation for the End model is relatively lower than for the Neural Receiver, never going above 50%. However, this is due in large part due to the worse baseline performance of this model in general.

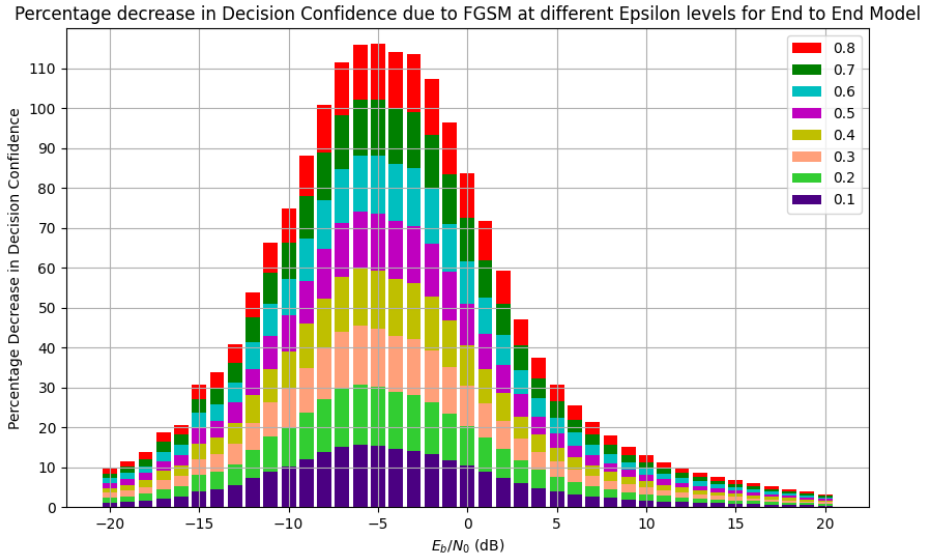



**Figure 3.21:** The decrease in Decision Confidence as a percentage of the baseline value due to FGSM at different epsilon values for the End to End model. The lack of adversarial robustness is clear here, as in the optimal operating region the decrease in confidence barely exceeds 10% for the highest epsilon values, with the lower values showing negligible differences.

### Adversarial Input Analysis

Once again, the adversarial inputs generated by FGSM and their original counterparts were extracted in order to compare and analyse. Comparing the amplitudes of the original inputs and adversarial ones again, we see a similar pattern emerge. As the epsilon values increase, on average the adversarial values increase in amplitude, as seen from Table 3.2. However, looking at Figs. 3.22, 3.23 and 3.24, we can also see that a significant amount of these inputs have identical or near identical amplitudes, potentially implying that there may be another factor at play compared to the Neural Receiver model. The implications of the patterns in amplitude are investigated further in Section 3.5

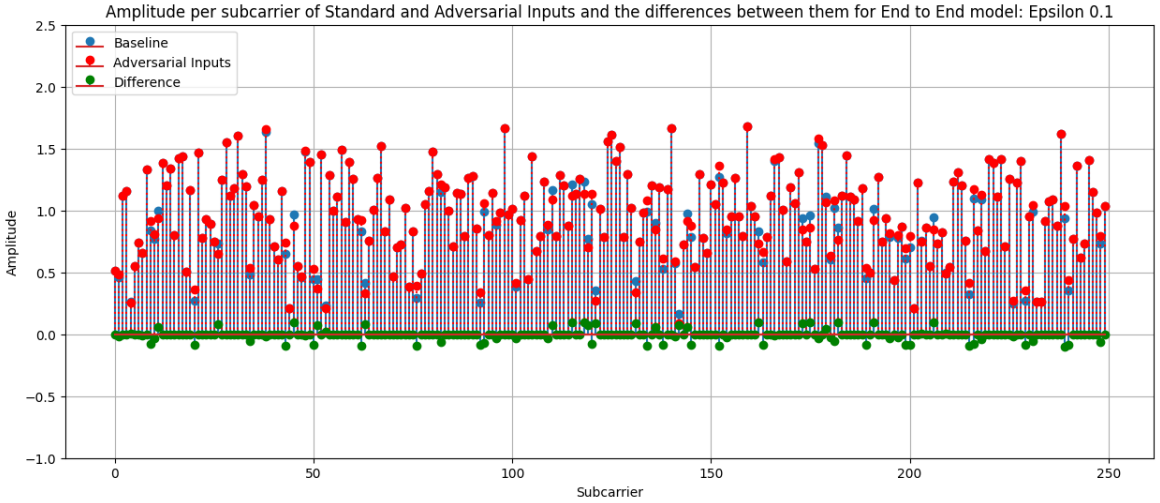



**Figure 3.22:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.1 and $E_b/N_0$ of 13dB for the End to End model. No pattern is present immediately except for the large amount of input pairs that have identical amplitudes.

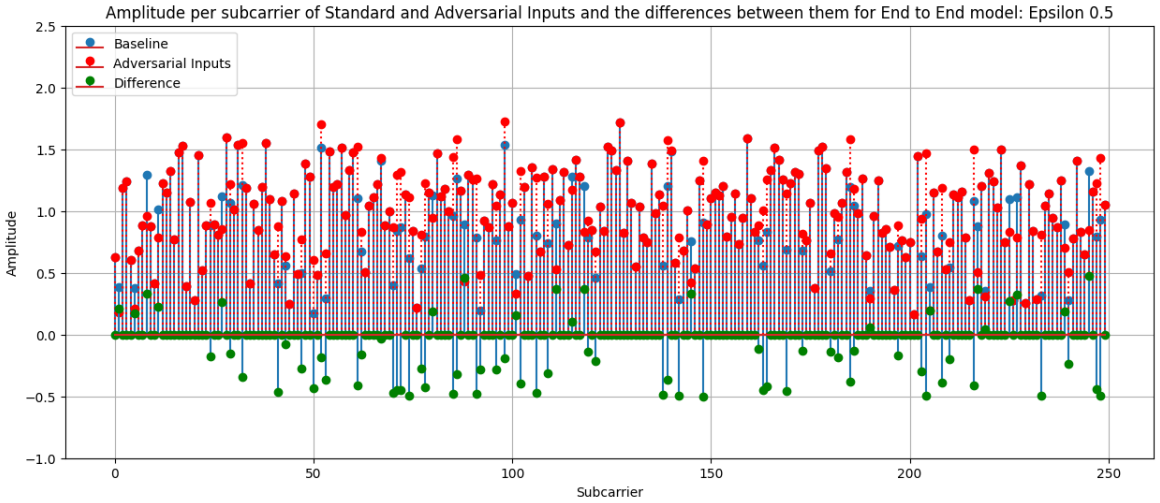



**Figure 3.23:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.5 and $E_b/N_0$ of 13dB for the End to End model. Overall, there appear to be a significant number of adversarial inputs of a greater amplitude, but a larger amount of ones with identical amplitudes.

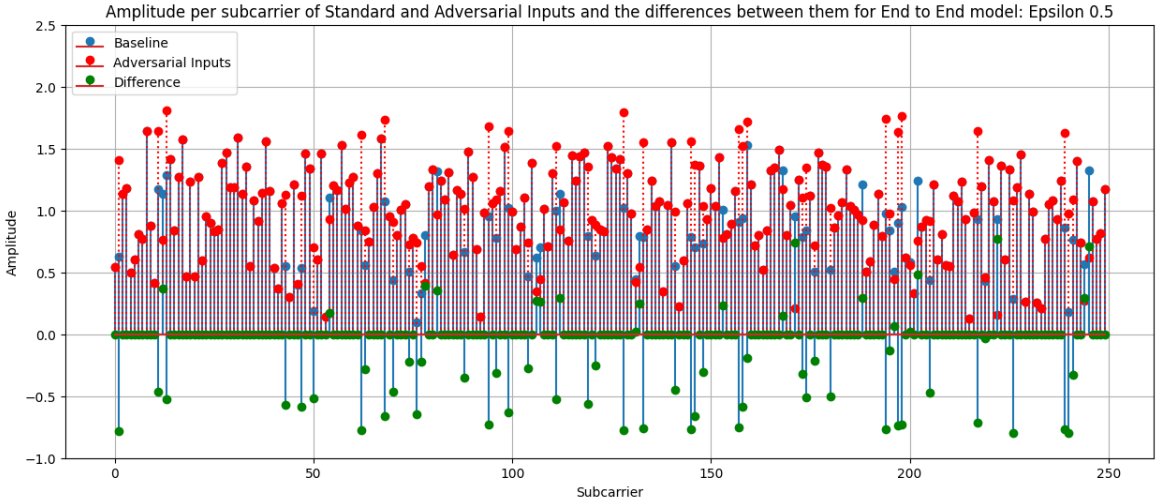



**Figure 3.24:** Amplitude of the original inputs, adversarial inputs and the difference between said amplitudes for an FGSM test run at an Epsilon value of 0.8 and $E_b/N_0$ of 13dB for the End to End model. Overall, there appear to be a significant number of adversarial inputs of a greater amplitude, but a larger amount of ones with identical amplitudes.

**Table 3.2:** Average difference in amplitude between original and adversarial inputs for End to End model. On average, the adversarial inputs have a greater amplitude, and this difference increases with the epsilon value

| Epsilon Value | Average Difference |
|---|---|
| 0.1 | -0.00295 |
| 0.2 | -0.00936 |
| 0.3 | -0.01785 |
| 0.4 | -0.02936 |
| 0.5 | -0.04338 |
| 0.6 | -0.02932 |
| 0.7 | -0.07774 |
| 0.8 | -0.09599 |

The presence of a large amount of identical amplitudes could imply that significant differences between the two sets of inputs could be elsewhere, such as there being significant differences in phase between the two inputs. However, further testing showed that while there did appear to be differences in phase between the two sets, there was still a significant amount of inputs with identical phase. An example can be seen in Fig. 3.25. This together with the amplitude results implies that a lot of the inputs between the two sets are identical. This in turn implies that potentially very few inputs actually need to be changed in order to induce significant performance degradation for the End to End model.

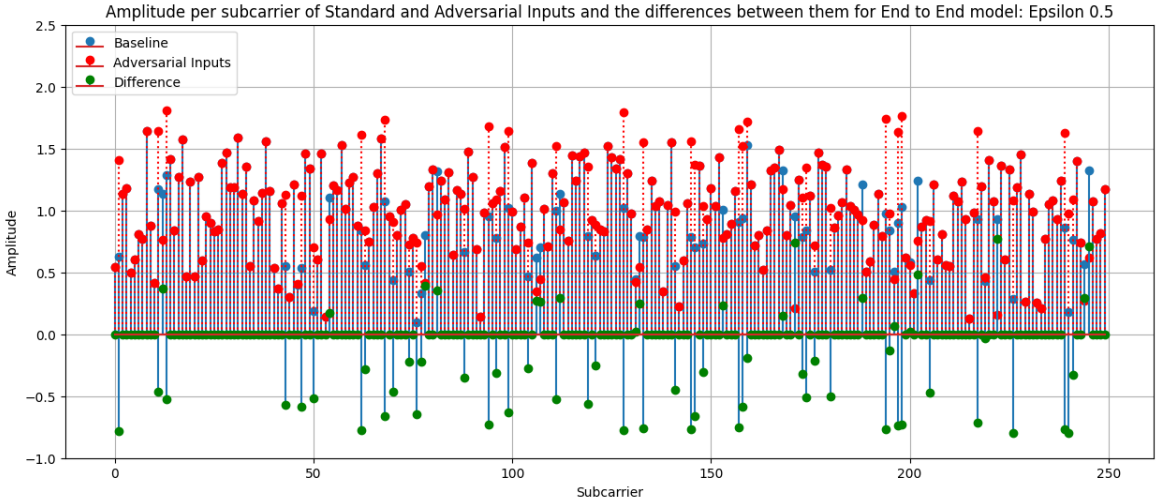



**Figure 3.25:** Phase angle of the original inputs, adversarial inputs and the difference between said phase angles for an FGSM test run at an Epsilon value of 0.8 and $E_b/N_0$ of 13dB for the End to End model. Comparing this data for different epsilon values shows a pattern of the average difference increasing, but the majority of inputs appear to have remained the same, in both amplitude and phase.

### 3.4.4 Initial Conclusions

Despite the difficulties encountered in its implementation due to the large amount of back-end work required, FGSM ended up being a very successful method. It was able to fulfil its stated purpose of measuring the robustness of both models when exposed to adversarial attacks, with Neural Receiver showing itself to be very robust, experiencing large drops in its confidence when attacked. On the other hand, the End to End model is the opposite, as its confidence was essentially unaffected in its optimum operation region. The perturbation of input data is a distinct possibility in the context of air interface communication, be it through interference or malicious actions such as jamming, so knowing the adversarial robustness of a model would be paramount. Furthermore, extracting the adversarial inputs for analysis allowed for the discovery of certain patterns in what types of inputs cause issues for both models. There is the possibility for more to be done in this area specifically, given more complex analysis methods. Overall, this method stands out as very useful for this context in the design, validation and implementation stage of a prospective models life cycle.

## 3.5 Further Testing

### 3.5.1 Investigating the effect of increased input amplitude

During FGSM testing, a pattern was observed when comparing the original inputs to the adversarial ones for both models, where they showed a distinct trend of the adversarial inputs having consistently larger amplitudes. While not true of every single input point, this pattern is consistent enough that it warranted a question: is the performance of both models susceptible to changes in amplitude? To investigate this, the structure of both models was modified to multiply the output of the channel by a set amount. This multiplication would be performed on both the signal and the noise to maintain a constant $E_b/N_0$. The results for this can be seen in Figs. 3.26 and 3.27. Interestingly, scaling factors of 2 and 4 appear to have very little effect and even improve the performance of Neural Receiver to an inconsistent degree, however in all other situations for both models the performance is significantly degraded.

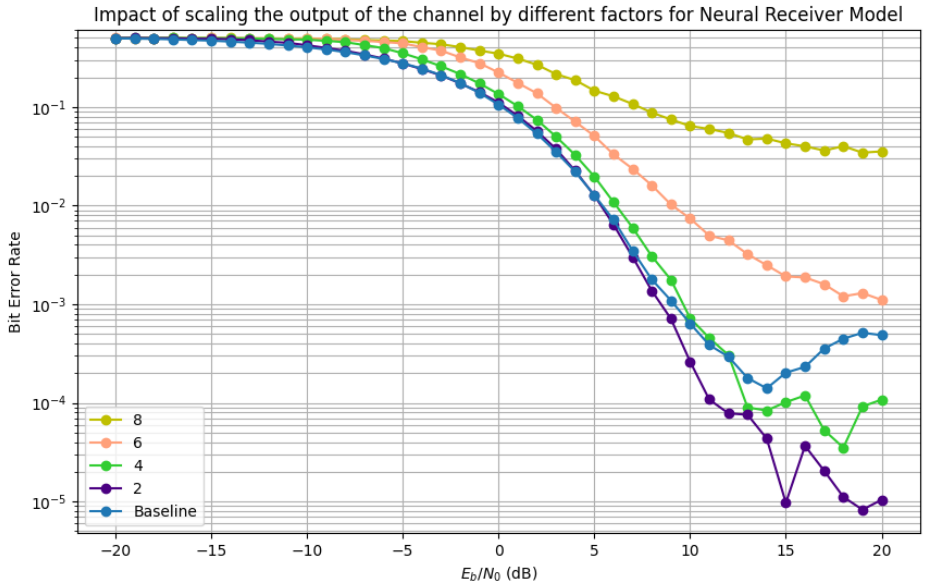



**Figure 3.26:** Effect of scaling the output of the channel(input of the model) on the BER for the Neural Receiver model. Scaling by factors of 2 and 4 appear to improve performance(albeit inconsistently), while scaling by 6 and 8 degrade it significantly.

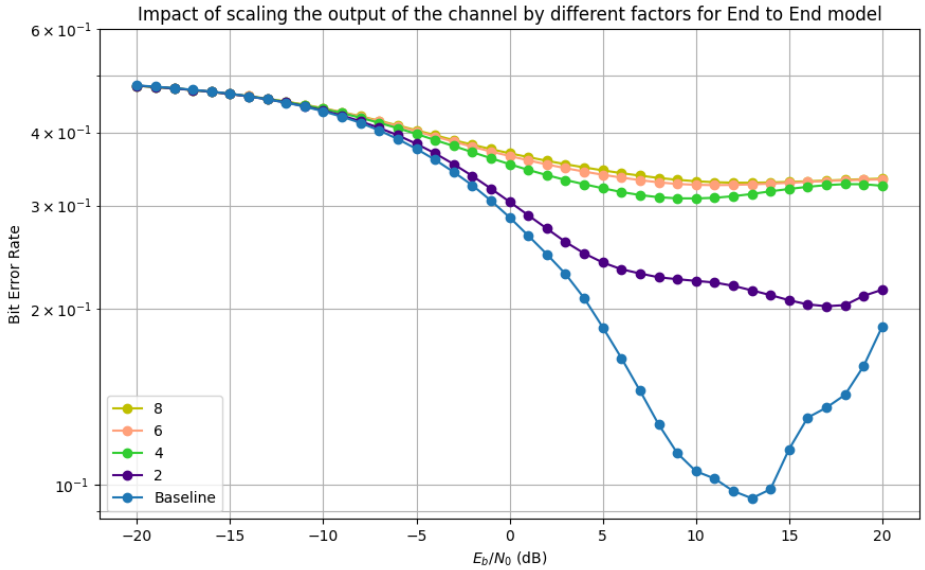



**Figure 3.27:** Effect of scaling the output of the channel(input of the model) on the BER for the End to End model. Any scaling appears to degrade performance significantly.

Looking at these graphs, it is evident that the Neural Receiver performs some amount of equalisation/channel estimation as it appears to have some sort of robustness against amplitude scaling to some extent. When the input is scaled by larger factors, the model fails as the inputs are too different to the training inputs. This is probably exacerbated by the fact that the training dataset involves some degree of normalisation, so the model is not used to dealing with inputs that are so large. On the other hand, the End to End model is operating with an AWGN channel which does not attenuate/amplify the transmitted symbol and only adds noise, which means the model does not have to perform any kind of channel equalisation. This is compounded by the models choice of modulation, that being 64 state Quadrature Amplitude Modulation (QAM). This explains the results seen in the graph, where it appears to be extremely vulnerable to input scaling. While such oversights are understandable due to the tutorial nature of both models, it does mean that to some extent both models have a very basic attack vector that can be used to degrade their performance, representing a significant weakness in their design.

### 3.5.2 SHAP and CEM

Due in part to the efficiency of working with the xpl[AI]ned system, there was time during the project period to look into both SHAP and CEM. In order to make the most of the time that was available, the choice was made to focus on the End to End model, as its relative simplicity compared to Neural Receiver increased the likelihood of success. Starting with SHAP, while a lot of time was spent trying to implement it and generate values, the im-

plementation was not possible in the end. After some investigation, this appears to be due to issues of dimensionality; the SHAP kernel explainer expects a maximum 2 dimensional array for both the input and the output, with a 1 dimensional array being preferable. The I/O to both chosen models had a larger dimensionality than this, meaning this implementation could not work directly. The possibility existed to change parts of the internal structure of the model in order to make the dimensionality fit with the requirements, but due to time issues and the fact that this would constitute a significant change and therefore essentially invalidate any generated results, this was not done.

Unfortunately, CEM was also not implemented in time, although this time the issue was one of incompatibility and lack of time. The xpl[AI]ned implementation for CEM used certain tools that were present in tensorflow v1.x but not in tensorflow v2.x. Both of the models were designed around tensorflow 2.x, which meant that the xpl[AI]ned CEM implementation required a rewrite from the ground up. Due to the fact that CEM was being attempted so late in the project cycle, there was not enough time to test this potential new implementation.

### 3.5.3 Investigating performance at high $E_b/N_0$ levels

As mentioned throughout this report and evident in many of the figures, both models experience a marked degradation in performance at the higher $E_b/N_0$ levels. The focus of this investigation is not the performance of these models in particular, but investigating this phenomena is still worthwhile as it presents an interesting intersection between issues with model design and issues relating to the air interface context. The main theory for this quirk in performance was thus: the models were trained on $E_b/N_0$ levels with significant amounts of noise, so as they tried to process data at $E_b/N_0$ rates with very low noise levels, they were mis-characterising parts of the signal as noise, leading to the errors. To test this hypothesis, both models could be retrained at higher $E_b/N_0$ ranges. Ideally, a similar pattern should emerge, with the models performing well within their range but having the performance drop significantly soon after.

Both models were retrained in a range from 10 to 25dB, and the results can be seen in Figs. 3.28 and 3.29. The results for the Neural Receiver model are as expected. When trained with the lower levels of noise present, the performance of the model significantly increases in the 15 to 25dB range, even reaching 0 in a few instances. As predicted, once the noise level starts decreasing(in this case around 30dB) the performance begins to deteriorate, becoming inconsistent as the BER begins to increase. However, the results for the End to End model do not replicate this, as it instead appears that this retraining has resulted in the model breaking entirely, with the BER never going below 0.3.

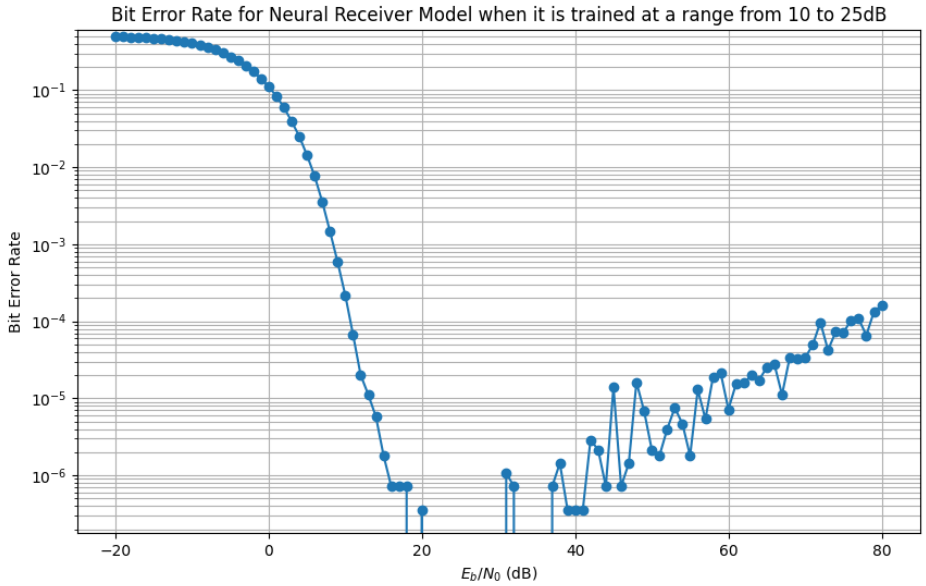



**Figure 3.28:** BER curve for the Neural Receiver model after having been retrained with data in a range from 10 to 25dB. This graph shows the expected result, with the performance improving in the area around 15-20dB where it was poor before, and then beginning to degrade around 30dB

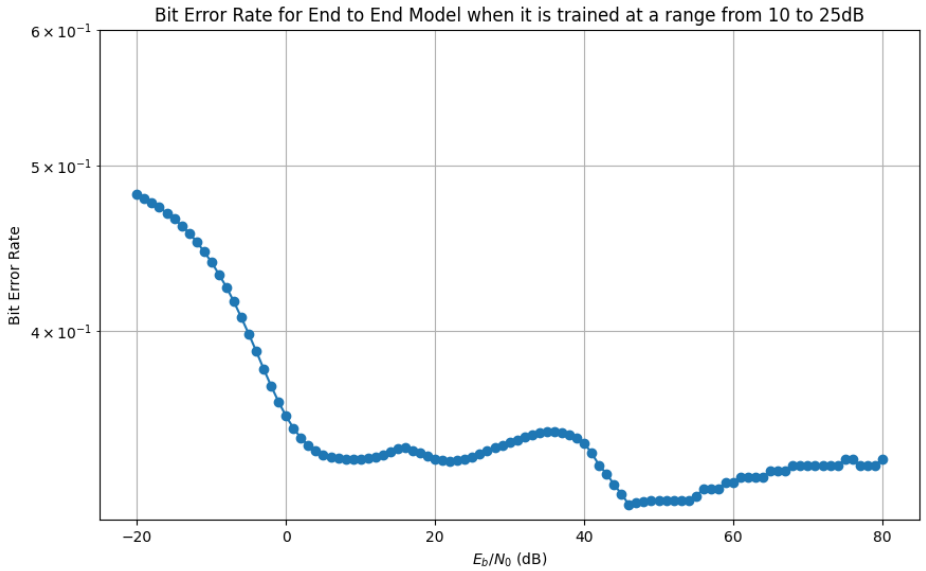



**Figure 3.29:** BER curve for the End to End model after having been retrained with data in a range from 10 to 25dB. This result does not match expectations, and the model appears to be entirely useless.

The models share a lot of aspects of their basic structure, with the main difference being the training structure, with the End to End model having its transmitter and receiver

trained separately via reinforcement link as opposed to the standard back-propagation method seen in Neural Receiver. In order to test what effect this training structure had on the high $E_b/N_0$ performance, the End to End model was retrained in its 'conventional' mode, both at the baseline range of 5 to 8dB and at the high end range of 10 to 25dB. At the baseline range, the performance of both implementations is very similar, as seen in Fig. 3.30. However, when retrained at a high range(Fig. 3.31), the conventionally trained model shows the expected performance that was also observed with Neural Receiver. This implies that the stated hypothesis might be correct with models trained using traditional back propagation, as it appears they are not able to discern between signal and noise at higher ranges. On the other hand, it seems that the RL implementation for the end to end model struggles at high $E_b/N_0$ values in general, possibly due to the way in which it uses approximation during training.

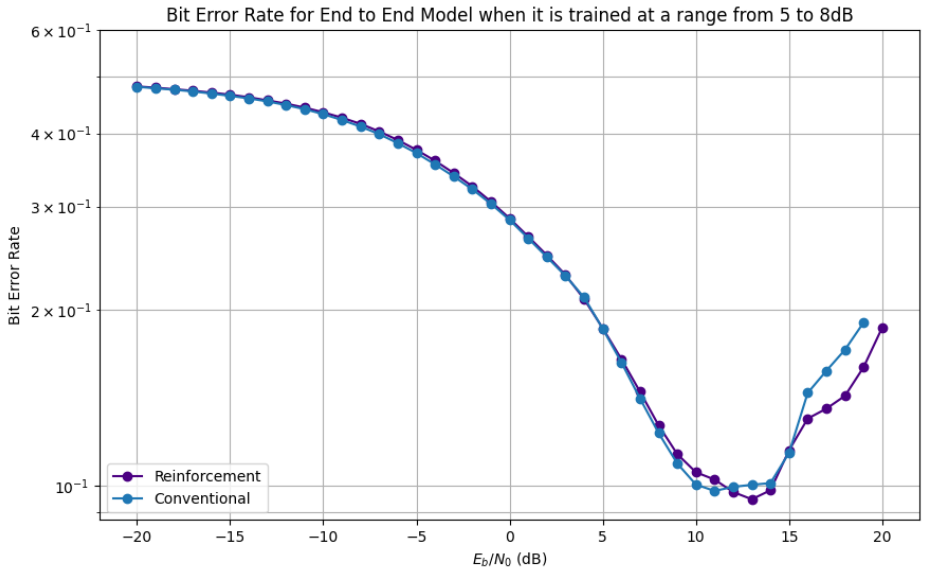



**Figure 3.30:** BER curves for the End to End model for both its conventional and reinforcement training implementations when trained at a range of 5-8dB. There appears to be minimal difference in performance between the two implementations

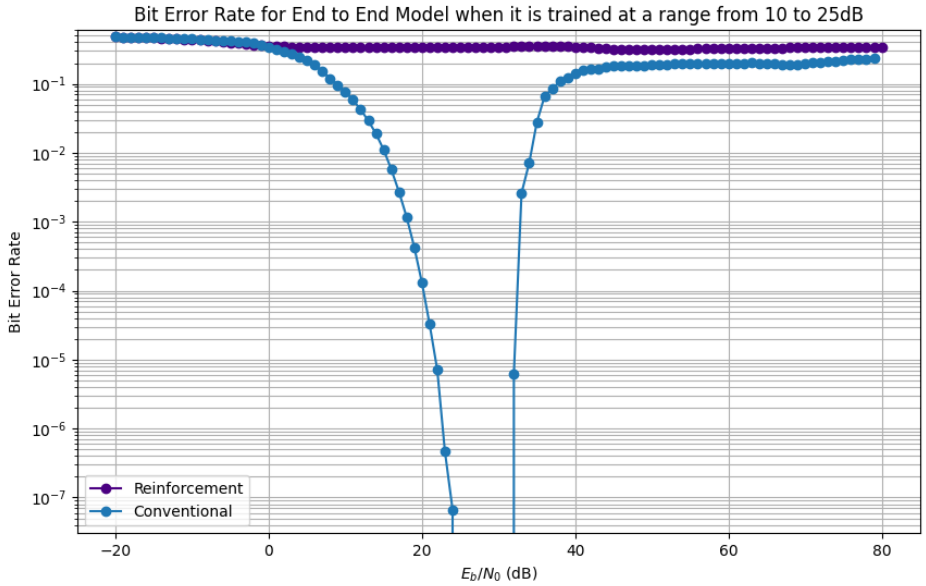



**Figure 3.31:** BER curves for the End to End model for both its conventional and reinforcement training implementations when trained at a range of 10-25dB. As shown previously, the RL implementation appears to break entirely, while the conventional implementation shows similar performance to the Neural Receiver model, with its 'optimum range' moved up and a severe degradation present after said range.

In an attempt to further understand the intricacies of the performance of the End to End model, the mapping constellation was investigated for both training implementations. The constellation in both instances was also trained during the training process, so could provide some insight into the performance and the difference between the two implementations. When looking at the constellations for training in the base range of 5-8dB, as shown in Figs. 3.32 and 3.33, there appears to be a consistent pattern. In both constellations, a lot of the mappings seem to be grouped into pairs or sets of 3/4 that differ by 1-2 bits, with this being more common with the lower magnitude mappings. This observation provides an explanation for the relatively mediocre performance of both implementations in terms of BER when trained at this range, as these groups imply that the model knows with a high degree of certainty that a given datapoint is one of the values in the group, but is not able to reliably determine which one of said values it is.

When retrained at 10-25dB, both constellations lose this pattern and take on a more expected grid-like pattern. This is most likely due to the lower overall noise presence, and also explains the significantly improved BER performance in the optimal region for the conventional implementation. With this in mind however, the poor performance of the RL implementation becomes an even bigger mystery. These results can be seen in Figs. 3.34 and 3.35.

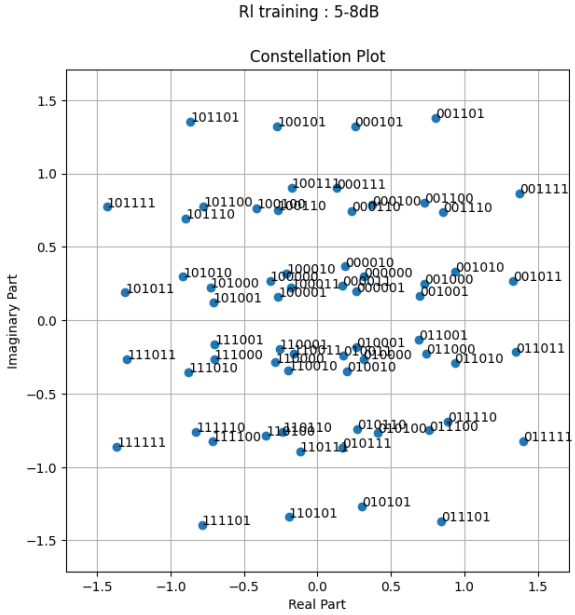



**Figure 3.32:** Trained constellation for RL implementation of the End to End model trained at 5-8dB. A clear pattern of points existing in groups of 2-4 that differ by 1-2 bits is visible, implying that the model is able to easily determine that a given point is within a group, but struggles to determine which point within said group.

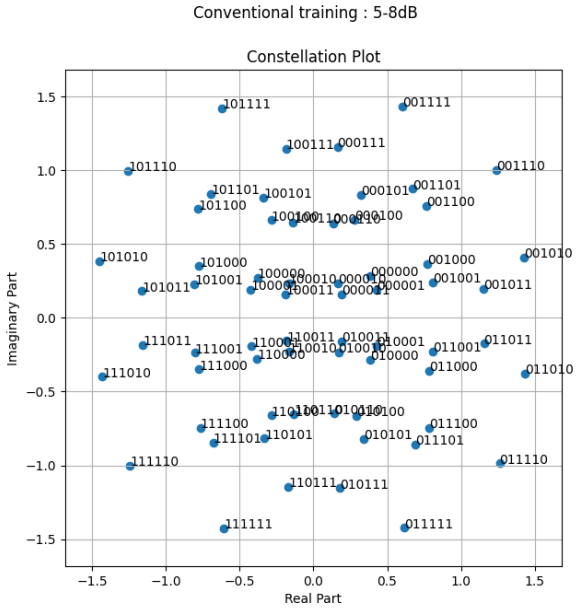



**Figure 3.33:** Trained constellation for conventional implementation of the End to End model trained at 5-8dB. A clear pattern of points existing in pairs that differ buy 1 bit is visible, implying that the model is able to easily determine that a given point is one of the two, but struggles to determine which one.

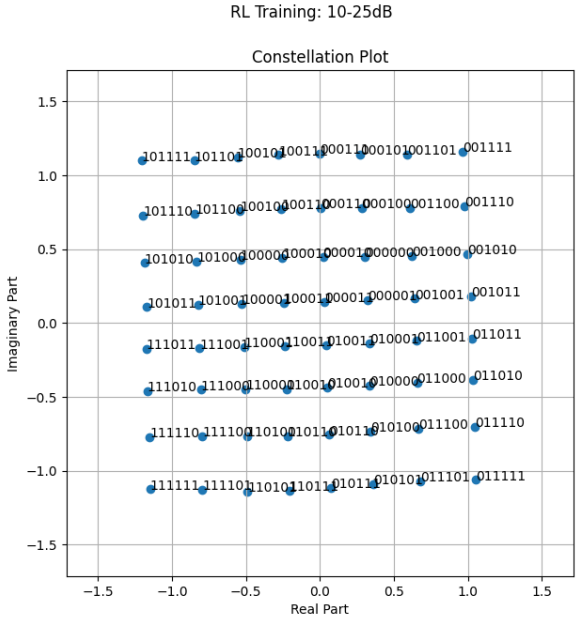



**Figure 3.34:** Trained constellation for RL implementation of the End to End model trained at 10-25dB. The previous pattern is gone and the constellation has taken a grid-like appearance, implying that demapping should be more consistent.

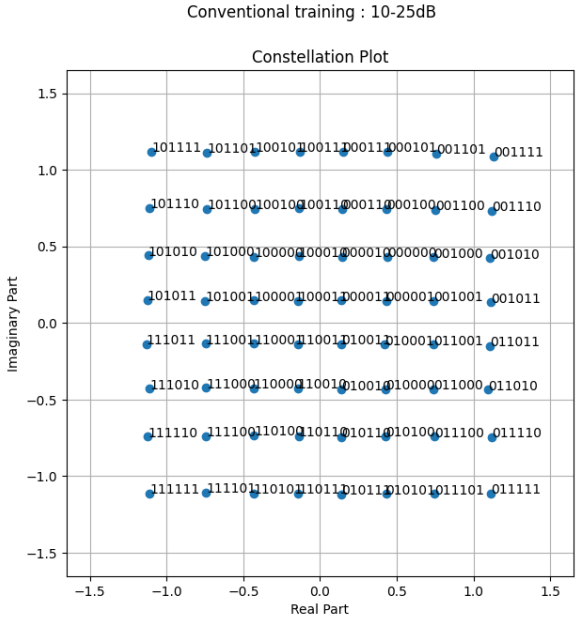



**Figure 3.35:** Trained constellation for conventional implementation of the End to End model trained at 10-25dB.The previous pattern is gone and the constellation has taken a grid-like appearance, implying that demapping should be more consistent.

# Chapter 4

# Discussion

## 4.1 Monte Carlo Dropout

The data generated from all of the testing methods applied to both models is quite useful. While the data reveals information about the specific performance of both models, this is of less importance to this project than what the data can tell us about the viability of the testing methods for the air interface context. Looking at Monte Carlo dropout first, this was one of the first methods implemented by the xpl[AI]ned team for the purpose of automotive image processing AI testing. In this context, dropout is a useful certainty test as image classification tends to feature a lot of redundancy; you can lose 50% of a picture of a car and still visually identify that it is a car. Therefore, dropout can help to identify just how much of an image a model needs in order to make a confident prediction. Moving this method over to the air interface communication space, this use case does not carry over to a large extent. By design, air interface communication works with the exact necessary amount of redundancy for the particular operating conditions. This means there is not a lot of flexibility, so the introduction of data loss through dropout would lead to the model having an inability to fully reconstruct the original message. This is shown by both models showing a significant negative impact when dropout is introduced in their input and output layers. While knowing just how much loss of input data degrades performance is useful, it also isn't anything new; losing part of the original information will obviously make reconstructing said original information more difficult.

Of course, dropout can also be used as a design and validation tool, ensuring that a model isn't unnecessarily large or determining the level of structural redundancy. This is done by performing dropout on hidden layers, and when done here both models showed an increase in BER as a result of dropout. The Neural Receiver model was more resistant in these tests than the End to End model, which suggests it is better optimised, while the End to End model could use some expansion as its BER suffered greatly. Considering how limited and efficient air interface communication hardware needs to be, the use of dropout

during development of these models makes sense. An unnecessarily large model requires more hardware, be it memory or processing power, making for less efficient communication systems.

Overall, while fairly easy to implement, the usefulness of Monte Carlo dropout as a testing method for air interface AI models is mixed. As a tool for certainty it is almost useless in this context, as we very rarely have redundant data that can be lost without affecting the communication performance in a meaningful way. This means that when used in this way, the method does not tell us anything we do not already know. On the other hand, implementing the method to investigate model size and efficiency remains useful as well as very easy. The differences in design between the two models were made clear using dropout, with the End to End model being as optimised as possible in terms of size while the Neural Receiver model showing a reduced response to dropout in comparison, especially at the lower rates of dropout. Based on this result, future work could be done to attempt to identify optimisation possibilities for Neural Receiver in more detail, in order to investigate how much lower the hardware requirement could be made while maintaining an acceptable level of performance.

## 4.2 FGSM

Another method ported over from xpl[AI]neds automotive AI ventures, FGSM ended up being a far more successful implementation than Monte Carlo dropout. This is perhaps not a surprise, as the field of air interface communications is known to be susceptible to data being compromised, so measuring a models resistance to it should give relevant information. A lot of work had to be done on the back end to make FGSM work, and while that is to be expected considering the amount of post and pre-processing done in this field, it was exacerbated by the choice of model. The Sionna models, being designed for tutorials, are put together in such a way where they are very open with many "disconnected" parts. Both the author and the xpl[AI]ned team believe that implementation for models intended for actual deployment, while still requiring some effort, would be a lot less time consuming than the experience with these models. While the final conclusions showing Neural Receiver as having a high adversarial robustness and End to End as having a low one are perhaps unsurprising due to their design philosophies, the ability to quantify and visualise both of these conclusions was extremely valuable, leading to further testing that could be expanded in the future.

This method was also taken further with the investigations into analysing the adversarial features, leading to the discovery of both models struggles with regards to equalisation. There is far more that could be done with this thread of investigation, and the potential exists for the use of FGSM as a tool to discover what types of interference a given air

interface model is weak to. This could be interference from other signals present in the operating environment, or malicious interference from jamming and other such means. The possibilities of this area of analysis have not been fully explored, and are perhaps worth an individual investigation themselves.

Overall, FGSM shows a lot of promise as a testing method for models in the air interface context. The results we have been able to collate show that it is useful in every stage of the development life cycle of an air interface AI model, with unique advantages when used in this context. It does require a considerable amount of development work, especially if the model needs to be surrounded by signal processing steps in order to function, but the potential of the results are such that when the work is done, FGSM produces results which greatly benefit the development of the model and the operators knowledge of its capabilities and limitations.

## 4.3 General Discussion

**Explainability Methods**

Explainability methods are perhaps some of the most interesting with regards to their potential for improving our understanding of model performance. Two such methods were set aside as stretch goals for the project, those being SHAP and CEM. They were relegated to stretch goals due to their inherent complexity and the fact that the xpl[AI]ned team did not have experience with implementing them yet. Due to a variety of factors, there was enough time to attempt some basic testing with both methods. SHAP was not implemented successfully as there were issues with regards to the dimensionality of the models that required significant changes to their structure in order to fix. This could potentially be indicative of a systemic issue, as very often when implementing signal processing methods we rely on having large multi dimensional matrices to store our results. It remains to be seen if this issue persists with more complex models in the future. It is possible CEM could have produced some interesting results as the xpl[AI]ned team believed it would not have had the same issues as SHAP, but there was not enough time to develop the implementation.

Overall, the inability to implement either chosen explainability method is perhaps the most unfortunate result of the project, as these methods are on the cutting edge and there is not a lot of documentation of them being used in this context. They remain as the first choice for avenues in which the work in this project could be taken further.

### Notes on performance degradation

The consistent issues with high $E_b/N_0$ performance, while having very little impact on the development during the project, are an interesting abnormality. As documented in Section 3.5.3, the hypothesis for the cause behind the inconsistencies was that the models had a poor grasp of the differing noise levels, and when the levels began to be too low compared to the training dataset, they would begin to mis-characterise parts of the signal itself as noise. The retraining of both models at higher $E_b/N_0$ ranges gave mixed results, although when the End to End model was implemented in its conventional form, the results fit more closely to what was expected. These results imply a potential underlying flaw in design that could perhaps be linked to Sionna with further testing.

It is worth noting that in both cases, the $E_b/N_0$ ranges the models were tested at in this report far exceed the ranges used in the documentation for both. The main reasoning behind this is the projects use of Bit Error Rate as a metric instead of Block Error Rate, meaning that a larger $E_b/N_0$ range was required to find the optimum value. With this in mind, it is possible that these architectures, especially the End to End model, were not designed to be operated at these levels and are simply unsuitable. Whatever the case may be, the anomalous performance of both models stands as a testament to the fact that training a model on a dataset does not guarantee that it will 'understand' the underlying pattern that the dataset is based on, which can lead to undesired performance at unknown extremes. This observation shows one of the biggest limitations of ML based models in this space. Receivers based on classical stochastic models of the system are able to operate on wide $E_b/N_0$ ranges, even when such ranges exceed the ones they were designed for. This is due to the fact that the structure they define does not change as the noise power changes, while ML based methods do not necessarily have this advantage.

### Other potential testing methods and future work

This project managed to exceed its desired scope in many ways, but there is still a lot than can be explored in the field. Many methods that potentially show a lot of promise could be implemented. Adversarial robustness was analysed through the use of FGSM, but this is one of the simplest methods for doing so and perhaps more insight can be gained with more complex implementations, especially for the issue of weight perturbation which was not covered. The analysis of the adversarial inputs generated by FGSM represents perhaps the most potential for future expansions of this project. The conclusions we were able to draw were fairly basic, and future work could begin to really dig into understanding the patterns the method generated and how they relate to real world interference. As well as this, explainability remains a field of testing with a lot of potential to be explored, especially considering the lack of success with implementing SHAP. It would be prudent to implement certain model aware methods as well, given the openness of the Sionna

structure. These explainability methods are critical for this context due to the complexity of the operations being performed, as understanding the model behaviour is necessary in order to have a concrete idea of what any communication system will do once deployed.

Perhaps most obvious development would be graduating to a more complex and representative set of models to put under test. The Sionna models were chosen for their ease of use and understanding, and now that a better understanding of development and good types of methods within the context has been gained it is important to investigate how the conclusions drawn from this report are supported/changed by running tests on different types of models. This comes with potential advantages and disadvantages. The Sionna models were put together in a way such as to be as open and easy to understand as possible, but aspects of that openness lead them to be difficult to develop for, especially with methods that required gradient analysis. Potentially, more 'professionally' minded models would not have this issue. However, this advantage goes hand in hand with the fact that such models would not be as explainable and malleable as the Sionna models were, especially if we do not have access to their internals. This uncertainty is exactly the reason why testing with more 'realistic' models is paramount to future development in the field.

**Usefulness of xpl[AI]ned**

The feasibility of this project proposal was predicated on the use of Keysight's xpl[AI]ned software. Ideally, this software would allow for the streamlining of the process of implementing new testing methods, as well as providing a universal framework that would allow for the development of front end testing code to be more efficient. The big unknown in this situation was the context, as xpl[AI]ned had thus far been developed for automotive AI purposes and not for use with air interface communication models. By the authors estimation, the use of xpl[AI]ned reduced the time taken for completion for the project by 1-2 months. Its use of templates for its methods, model and data loaders completely circumvented the complex work that would have been required to code everything required for each testing method manually. As well as this, it also simplified work on the front end, as test result generation code could be written with the knowledge of xpl[AI]ned's structure in mind and so could be reused between multiple models and methods.

That is not to say that there wasn't work to be done, as the template based structure also meant that the software expected to be presented with models and data in specific ways, so work had to be done to incorporate the various signal processing steps that surrounded our models in order for the test steps to work. However, the author and the xpl[AI]ned team believe that this downside would be lessened through the use of a model built for deployment with a more conventional structure. When comparing this work to any alternative development paths that could have been taken, it is clear that without xpl[AI]ned the project in the form that it is presented in this report would not exist. The

software's advantages have allowed the scope of this project to be expanded and the author believes that the concept has serious potential to improve the AI development and validation pipeline in the future.

# Chapter 5

# Conclusions

The stated goal at the start of this project was an exploration of the application of certain testing methods on air interface communication AI models through the means of Keysight Technologies' new xpl[AI]ned system. Given Keysight's area of expertise, they were interested in how this system could adapt to the context, and the author was interested in gaining an understanding of what air interface AI looked like in terms of structure and performance. As the development of the project progressed, these goals were refined, and at the conclusion of the project it stands as documentation of the implementation, relevance and results of two main testing methods and other smaller explorations on two air interface related neural network models. The author believes that the project has achieved its stated goal and can act as a record of an exploration of the space, but not without some caveats.

The two main testing methods that were implemented were Monte Carlo Dropout and the Fast Gradient Sign Method for adversarial robustness. These methods represent two current industry standard methods for AI model testing, and together cover a models certainty, structural efficiency and response to adversarial perturbation. Attempts were made to implement two explainability methods as well, but these were met with technical issues. The results generated from these models, despite being unexpected occasionally due to certain aspects of the model performance, come together to begin to form a picture about working within this space and how model complexity affects some of their more hidden characteristics. While some of the results showed expected and known trends, the meta-results of this project with regards to difficulty of implementation and interpretation are perhaps more interesting and useful as a whole. The air interface communication space is a complex context to develop models within, and understanding these complexities is key to enabling better development in the future.

A big reason behind why data extracted from these methods was useful was due to the smart decisions made with regards to model choice. By keeping the models simple and

broad in terms of their representation but also making them opposite sides of the spectrum in terms of complexity, this report shows the wide range of results that can be expected when implementing the chosen testing methods. On the other hand, the model choice also hampered certain aspects of result generation. The structure of both models and the 'openness' with which they were put together due to being essentially tutorials lead to some issues with implementing certain testing methods. As well as this, the inconsistent performance of the models in certain aspects limits the usefulness of the generated data, and ideally these testing methods could now be taken and implemented on models that are more representative of industry standards.

The choice to directly tie this project to Keysight's xpl[AI]ned infrastructure meant that the amount of work that was achieved within the project period was significantly more than would otherwise have been possible. While the choice of two models with similar structures also helped in this regard, the fact that testing infrastructure could essentially be reused due to the consistency of the way methods were implemented in xpl[AI]ned was a huge factor. The project managed to achieve all of its stretch goals despite any work that had to be done in order to adapt xpl[AI]ned to this context. However, it is possible that the requirements of xpl[AI]ned in terms of format limited certain tests, such as SHAP, but without it it is questionable if reaching these more advanced testing methods would have even been possible.

Overall, the author believes that this project has been a significant success. From a vague and open ended proposal, a report has been put together that documents not only new methods of standardising the application of testing methods for AI models, but also how certain methods interact with air interface communication models. The experiences of implementation and the generated results come together to paint a picture of a situation which can have various unique requirements when compared to many other contexts, and as such special considerations must be applied. This is especially true when working with models that simulate small parts of complex systems as described in this report, as surrounding processing steps often have a large impact on the format of the I/O, requiring careful engineering in order to implement certain testing methods. At the end of the day it is the authors belief that this project can stand as an introductory piece of work that can point the way in terms of development in the field of air interface communication AI model testing.

In terms of how the work undertaken by this project can be expanded upon further, there are multiple different paths that can be taken. As previously mentioned, this project lacks documentation of the results of explainability methods, of which there appears to be very little documentation of in general in this space. A significant part of the project is focused on a study of robustness, but this could be taken even further. The effects of data poisoning on the training data could be studied to determine the robustness of the

models in a training context. Comparing this robustness against the robustness to deviations in test data documented within this report could be a useful tool to inform future model design. A key aspect of the air interface context not represented by this report is the fact that multiple different ML models are often implemented in concurrence when used in this space. These models perform complex operations and pass data amongst themselves as they execute various tasks. Combining models in this way leads to new testing opportunities that could be explored, such as developing methods to test the aggregate effect of all of these models put together and what the vulnerabilities are of such systems as a whole, rather than of the individual components within as this project has investigated. This field is a rapidly expanding one, and almost every aspect covered by this report could be investigated further to make more granular and complete conclusions.

# Bibliography


[1] *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*. URL: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

[2] *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. URL: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[3] Faycal Aoudia and Jakob Hoydis. "Towards Hardware Implementation of Neural Network-based Communication Algorithms". In: July 2019, pp. 1–5. DOI: 10.1109/SPAWC.2019.8815398.

[4] Fayçal Ait Aoudia and Jakob Hoydis. "Model-free training of end-to-end communication systems". In: *IEEE Journal on Selected Areas in Communications* 37.11 (2019), pp. 2503–2516.

[5] Martin D Buhmann. "Radial basis functions". In: *Acta numerica* 9 (2000), pp. 1–38.

[6] Sebastian Cammerer et al. "Graph neural networks for channel decoding". In: *2022 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2022, pp. 486–491.

[7] *CENTRIC*. URL: https://centric-sns.eu/.

[8] Pin-Yu Chen. *Securing AI systems with adversarial robustness*. 2021. URL: https://research.ibm.com/blog/securing-ai-workflows-with-adversarial-robustness.

[9] Amit Dhruandhar. *Contrastive Explanations Help AI Explain Itself by Identifying What is Missing*. 2018. URL: https://www.ibm.com/blogs/research/2018/05/contrastive-explanations/.

[10] SHAP Documentation. *shap.KernelExplainer*. URL: https://shap-lrjball.readthedocs.io/en/latest/generated/shap.KernelExplainer.html.

[11] *Four Common Types of Neural Network Layers*. URL: https://towardsdatascience.com/four-common-types-of-neural-network-layers-c0d3bb2a966c#:~:text=The\%20four\%20most\%20common\%20types,how\%20they\%20can\%20be\%20used..

[12] Tadayoshi Fushiki. "Estimation of prediction error by using K-fold cross-validation". In: *Statistics and Computing* 21 (2011), pp. 137–146.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016. Chap. 1.

[14] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[15] Mathieu Goutay et al. "Machine Learning-enhanced Receive Processing for MU-MIMO OFDM Systems". In: *2021 IEEE 22nd International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE. 2021, pp. 246–250.

[16] Tobias Gruber et al. "On Deep Learning-based Channel Decoding". In: *2017 51st annual conference on information sciences and systems (CISS)*. IEEE. 2017, pp. 1–6.

[17] Mikko Honkala, Dani Korpi, and Janne MJ Huttunen. "DeepRx: Fully convolutional deep learning receiver". In: *IEEE Transactions on Wireless Communications* 20.6 (2021), pp. 3925–3940.

[18] Jakob Hoydis et al. "Sionna: An open-source library for next-generation physical layer research". In: *arXiv preprint arXiv:2203.11854* (2022).

[19] Jakob Hoydis et al. "Towards a 6G AI-Native Air Interface". In: *IEEE Communications Magazine* May 2021 (2021), pp. 76–81.

[20] Chris Kuo. *Explain Any Models with the SHAP Values — Use the KernelExplainer*. URL: https://towardsdatascience.com/explain-any-models-with-the-shap-values-use-the-kernelexplainer-79de9464897a.

[21] Great Learning. *Types of Neural Networks and Definition of Neural Network*. URL: https://www.mygreatlearning.com/blog/types-of-neural-networks/.

[22] Benoit Liquet, Sarat Moka, and Yoni Nazarathy. *General Fully Connected Neural Networks*. URL: https://deeplearningmath.org/general-fully-connected-neural-networks.html#n-layers-neural-network.

[23] Benjamin F Logan and Larry A Shepp. "Optimal reconstruction of a function from its projections". In: (1975).

[24] Lei Ma et al. "Deepgauge: Multi-granularity testing criteria for deep learning systems". In: *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*. 2018, pp. 120–131.

[25] Juan Montojo. *AI/ML for NR Air Interface*. URL: https://www.3gpp.org/technologies/ai-ml-nr.

[26] NVlabs. *End-to-end Learning with Autoencoders*. URL: https://nvlabs.github.io/sionna/examples/Autoencoder.html.

[27] NVlabs. *Neural Receiver for OFDM SIMO Systems*. URL: https://nvlabs.github.io/sionna/examples/Neural_Receiver.html.

[28] Ayodeji Oseni et al. "Security and privacy for artificial intelligence: Opportunities and challenges". In: *arXiv preprint arXiv:2102.04661* (2021).

[29] Kexin Pei et al. “Deepxplore: Automated whitebox testing of deep learning systems”. In: *proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 1–18.

[30] Francesc Pozo Montero. “Validation Procedures. Leave-one-out cross-validation (LOOCV) and k-fold cross-validation”. In: (2019).

[31] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “" Why should i trust you?" Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.

[32] Drew Roselli, Jeanna Matthews, and Nisha Talagala. “Managing Bias in AI”. In: *Companion Proceedings of The 2019 World Wide Web Conference*. WWW ’19. San Francisco, USA: Association for Computing Machinery, 2019, 539–544. ISBN: 9781450366755. DOI: 10.1145/3308560.3317590. URL: https://doi.org/10.1145/3308560.3317590.

[33] Grant Sanderson. *Neural networks*. 2023. URL: https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.

[34] Ramprasaath R Selvaraju et al. “Grad-CAM: Why did you say that?” In: *arXiv preprint arXiv:1611.07450* (2016).

[35] K. Pavan Srinath and Jakob Hoydis. *Bit-Metric Decoding Rate in Multi-User MIMO Systems: Theory*. 2022. arXiv: 2203.06271 [cs.IT].

[36] Keysight Technologies. *Concepts of Orthogonal Frequency Division Multiplexing (OFDM) and 802.11 WLAN*. URL: https://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/wlan-ofdm/content/ofdm_basicprinciplesoverview.htm.

[37] *Understanding binary cross-entropy / log loss: a visual explanation*. URL: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a.

[38] *Understanding the 3 most common loss functions for Machine Learning Regression*. URL: https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression.

[39] Tzu-Tsung Wong and Po-Yang Yeh. “Reliable accuracy estimates from k-fold cross validation”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.8 (2019), pp. 1586–1594.

# Appendix A

# Raw Data

## A.1 Model Performance

**Table A.1:** Bit Error Rate for Neural Receiver Model(NR) and End to End Model(E2E) from -20 to 20dB

| $E_b/N_0$ | **Average BER NR** | **Average BER E2E** |
|---|---|---|
| **-20** | 4.96E-01 | 4.81E-01 |
| **-19** | 4.93E-01 | 4.79E-01 |
| **-18** | 4.90E-01 | 4.76E-01 |
| **-17** | 4.85E-01 | 4.73E-01 |
| **-16** | 4.77E-01 | 4.70E-01 |
| **-15** | 4.71E-01 | 4.66E-01 |
| **-14** | 4.60E-01 | 4.61E-01 |
| **-13** | 4.49E-01 | 4.55E-01 |
| **-12** | 4.36E-01 | 4.50E-01 |
| **-11** | 4.19E-01 | 4.43E-01 |
| **-10** | 4.03E-01 | 4.35E-01 |
| **-9** | 3.83E-01 | 4.25E-01 |
| **-8** | 3.60E-01 | 4.15E-01 |
| **-7** | 3.33E-01 | 4.03E-01 |
| **-6** | 3.06E-01 | 3.90E-01 |
| **-5** | 2.77E-01 | 3.75E-01 |
| **-4** | 2.45E-01 | 3.59E-01 |
| **-3** | 2.10E-01 | 3.42E-01 |
| **-2** | 1.75E-01 | 3.25E-01 |
| **-1** | 1.39E-01 | 3.06E-01 |
| **0** | 1.06E-01 | 2.86E-01 |
| **1** | 7.74E-02 | 2.67E-01 |
| **2** | 5.36E-02 | 2.48E-01 |
| **3** | 3.51E-02 | 2.29E-01 |
| **4** | 2.20E-02 | 2.08E-01 |
| **5** | 1.27E-02 | 1.86E-01 |
| **6** | 7.19E-03 | 1.65E-01 |
| **7** | 3.49E-03 | 1.45E-01 |
| **8** | 1.78E-03 | 1.27E-01 |
| **9** | 1.09E-03 | 1.13E-01 |
| **10** | 6.39E-04 | 1.05E-01 |
| **11** | 3.87E-04 | 1.02E-01 |
| **12** | 2.94E-04 | 9.76E-02 |
| **13** | 1.79E-04 | 9.48E-02 |
| **14** | 1.41E-04 | 9.83E-02 |
| **15** | 2.02E-04 | 1.15E-01 |
| **16** | 2.32E-04 | 1.30E-01 |
| **17** | 3.56E-04 | 1.35E-01 |
| **18** | 4.47E-04 | 1.42E-01 |
| **19** | 5.14E-04 | 1.59E-01 |
| **20** | 4.86E-04 | 1.86E-01 |

**Table A.2:** Average Confidence for Neural Receiver Model(NR) and End to End Model(E2E) from -20 to 20dB. Average confidence is the average absolute value for the LLRs generated by the models

| $E_b/N_0$ | **Average Confidence NR** | **Average Confidence E2E** |
|---|---|---|
| **-20** | 1.79E+00 | 2.24E+01 |
| **-19** | 1.50E+00 | 1.87E+01 |
| **-18** | 1.27E+00 | 1.56E+01 |
| **-17** | 1.08E+00 | 1.30E+01 |
| **-16** | 9.65E-01 | 1.08E+01 |
| **-15** | 8.91E-01 | 8.98E+00 |
| **-14** | 8.53E-01 | 7.51E+00 |
| **-13** | 8.43E-01 | 6.29E+00 |
| **-12** | 8.57E-01 | 5.31E+00 |
| **-11** | 8.86E-01 | 4.52E+00 |
| **-10** | 9.28E-01 | 3.88E+00 |
| **-9** | 9.95E-01 | 3.37E+00 |
| **-8** | 1.07E+00 | 2.98E+00 |
| **-7** | 1.16E+00 | 2.67E+00 |
| **-6** | 1.29E+00 | 2.45E+00 |
| **-5** | 1.46E+00 | 2.29E+00 |
| **-4** | 1.69E+00 | 2.20E+00 |
| **-3** | 2.00E+00 | 2.16E+00 |
| **-2** | 2.44E+00 | 2.18E+00 |
| **-1** | 3.05E+00 | 2.25E+00 |
| **0** | 3.84E+00 | 2.38E+00 |
| **1** | 4.73E+00 | 2.56E+00 |
| **2** | 5.65E+00 | 2.83E+00 |
| **3** | 6.68E+00 | 3.18E+00 |
| **4** | 7.76E+00 | 3.62E+00 |
| **5** | 8.90E+00 | 4.15E+00 |
| **6** | 1.01E+01 | 4.77E+00 |
| **7** | 1.12E+01 | 5.45E+00 |
| **8** | 1.23E+01 | 6.14E+00 |
| **9** | 1.32E+01 | 6.80E+00 |
| **10** | 1.39E+01 | 7.38E+00 |
| **11** | 1.44E+01 | 7.87E+00 |
| **12** | 1.47E+01 | 8.30E+00 |
| **13** | 1.49E+01 | 8.69E+00 |
| **14** | 1.50E+01 | 9.05E+00 |
| **15** | 1.51E+01 | 9.42E+00 |
| **16** | 1.51E+01 | 9.75E+00 |
| **17** | 1.51E+01 | 1.01E+01 |
| **18** | 1.51E+01 | 1.04E+01 |
| **19** | 1.50E+01 | 1.08E+01 |
| **20** | 1.49E+01 | 1.12E+01 |

## A.2 Monte Carlo Dropout: Neural Receiver

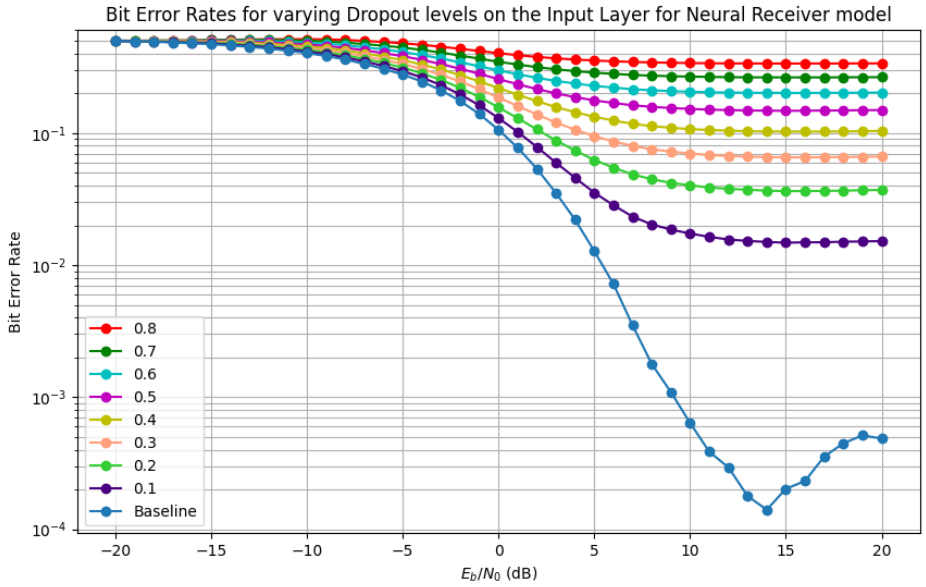



**Figure A.1:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its input layer.

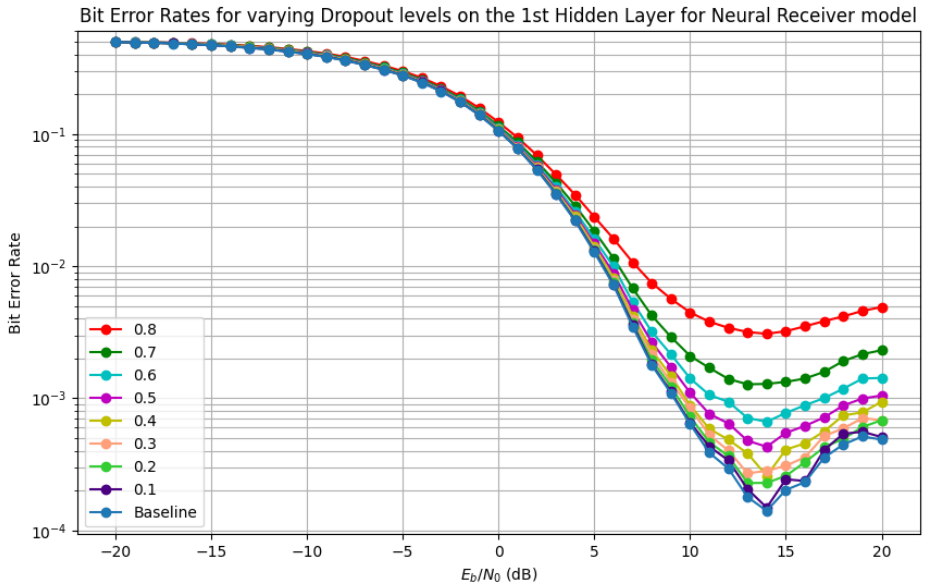



**Figure A.2:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its first hidden layer.

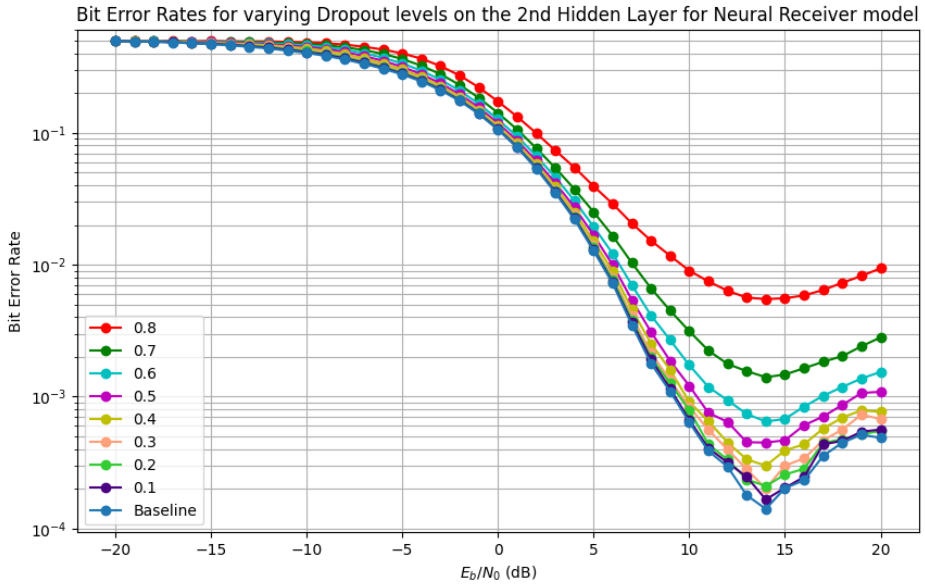



**Figure A.3:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its second hidden layer.

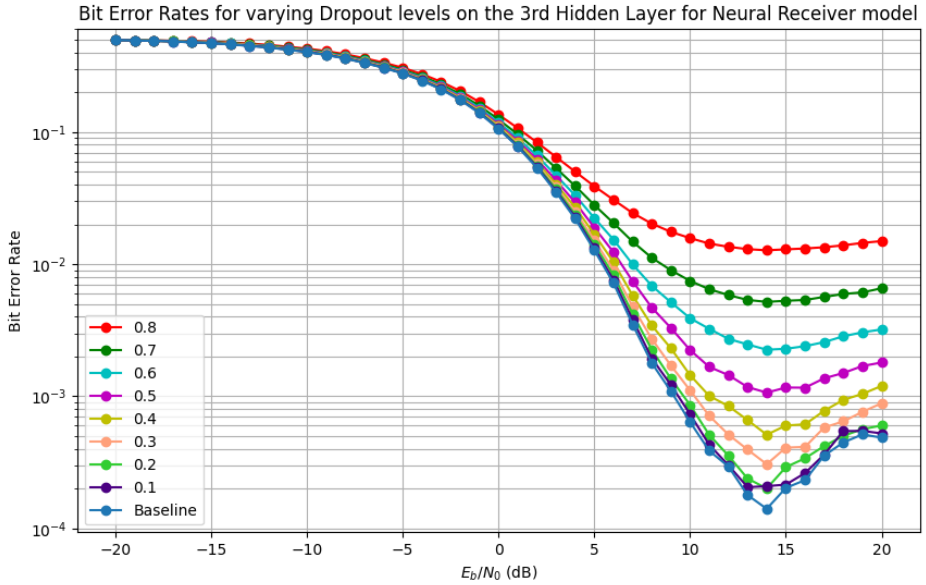



**Figure A.4:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its third hidden layer.

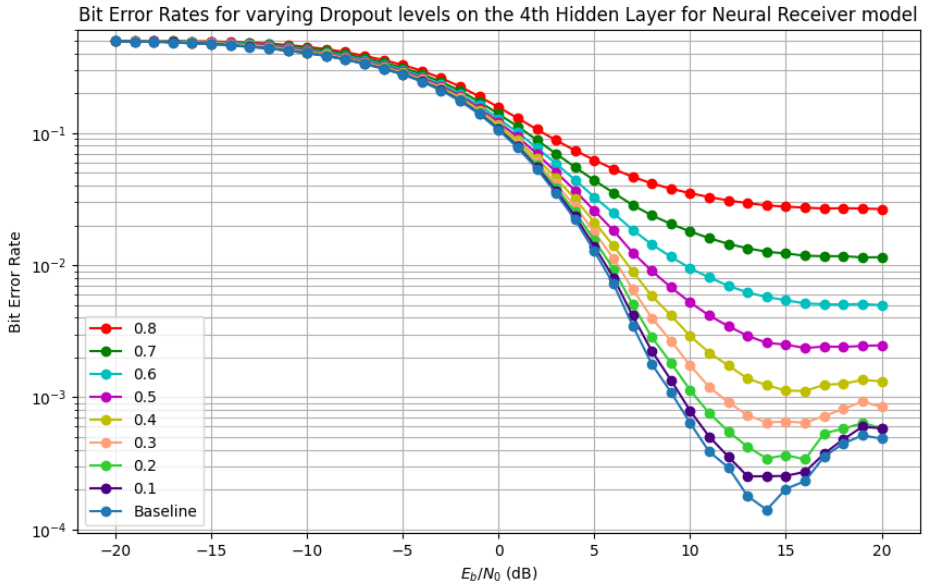



**Figure A.5:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its fourth hidden layer.

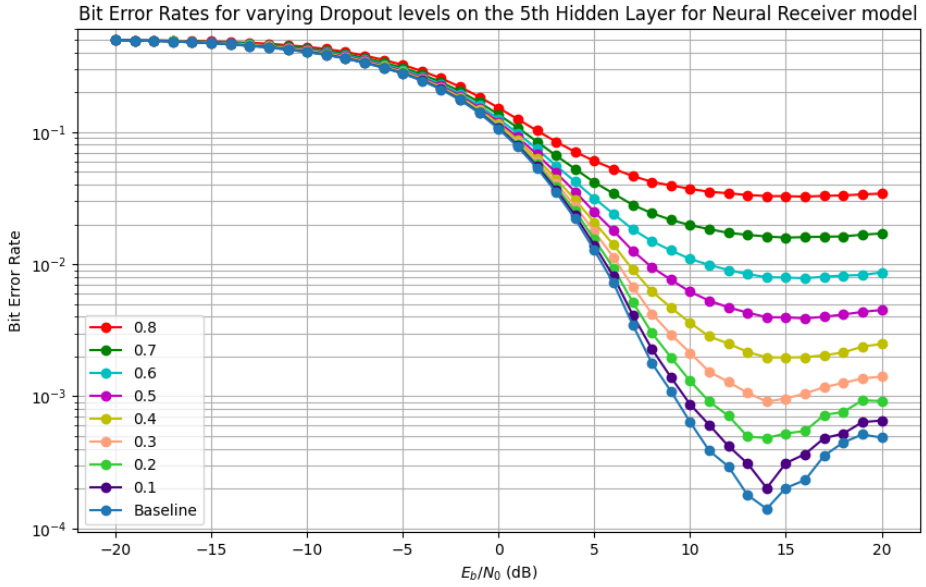



**Figure A.6:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its fifth hidden layer.

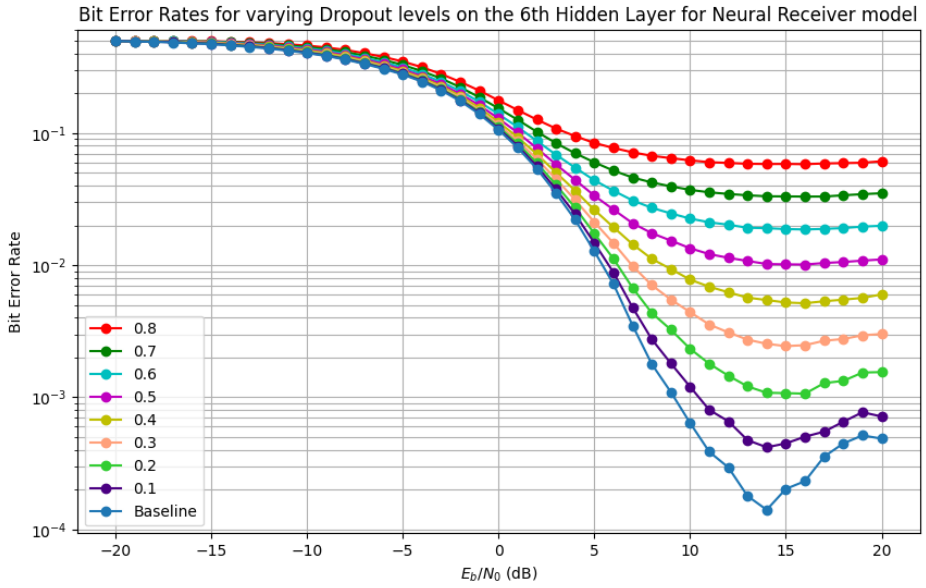



**Figure A.7:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its sixth hidden layer.

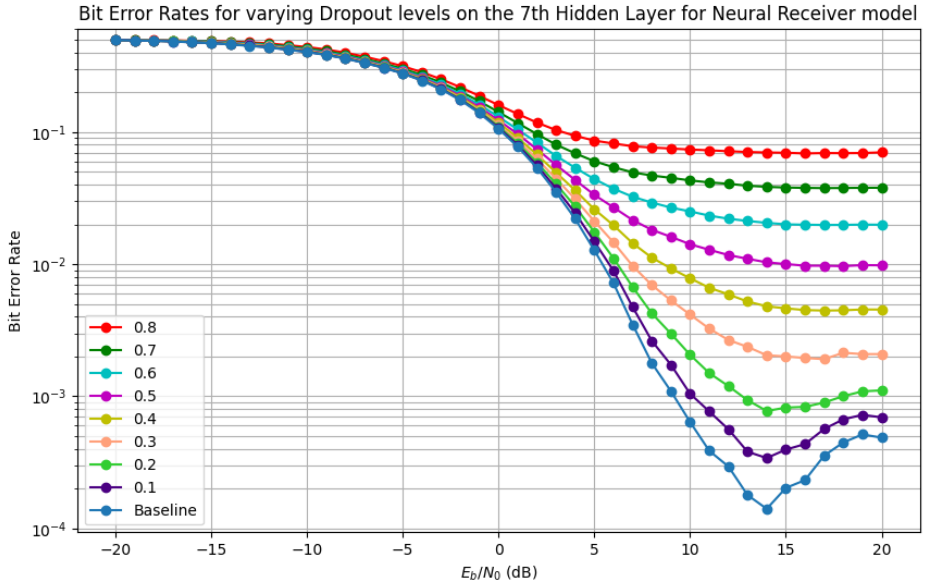



**Figure A.8:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its seventh hidden layer.

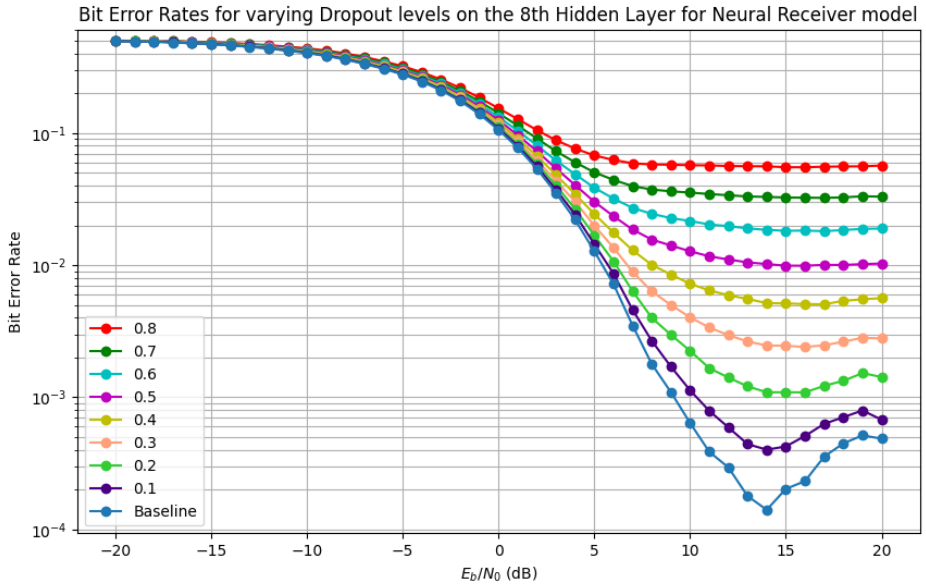



**Figure A.9:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its eighth hidden layer.

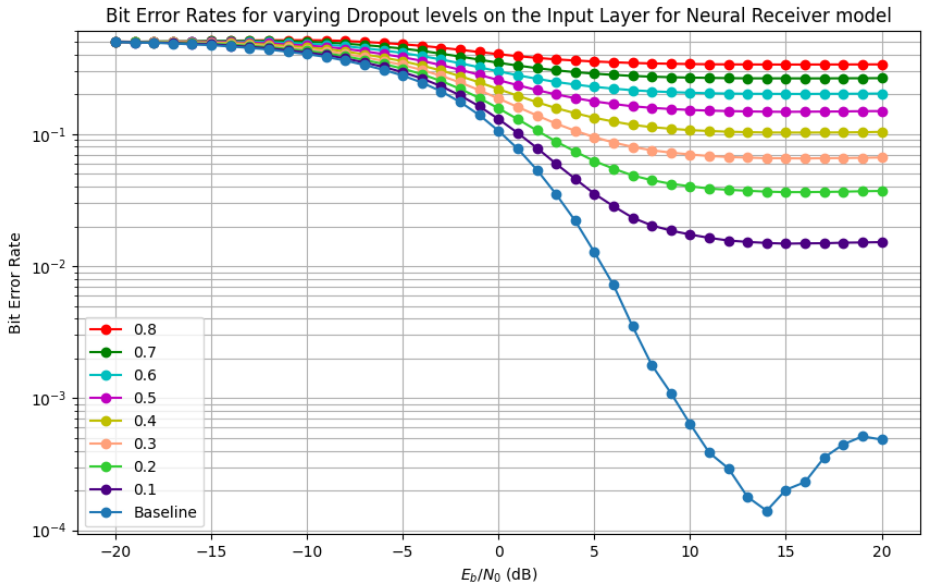



**Figure A.10:** Bit Error Rate curves for the Neural Receiver model with different dropout rates applied to its output layer.

**Table A.3:** Average Changes to BER for Neural Receiver Due to Dropout applied to Input Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000794 | 0.001413 | 0.002222 | 0.002411 | 0.002734 | 0.003605 | 0.003794 | 0.004047 |
| **-19** | 0.001225 | 0.002808 | 0.003964 | 0.005046 | 0.00578 | 0.005846 | 0.005447 | 0.00634 |
| **-18** | 0.002341 | 0.004331 | 0.006536 | 0.0076 | 0.008196 | 0.00997 | 0.01031 | 0.010271 |
| **-17** | 0.003334 | 0.006887 | 0.009842 | 0.011694 | 0.014059 | 0.015505 | 0.016464 | 0.017078 |
| **-16** | 0.004778 | 0.009505 | 0.014283 | 0.017469 | 0.020892 | 0.022991 | 0.024807 | 0.025704 |
| **-15** | 0.006419 | 0.012099 | 0.017975 | 0.023406 | 0.028333 | 0.031827 | 0.035061 | 0.036277 |
| **-14** | 0.006746 | 0.014642 | 0.021456 | 0.029008 | 0.035883 | 0.041656 | 0.046244 | 0.048248 |
| **-13** | 0.007814 | 0.016104 | 0.024713 | 0.033597 | 0.042772 | 0.051274 | 0.05762 | 0.061638 |
| **-12** | 0.009285 | 0.018657 | 0.028293 | 0.038445 | 0.0497 | 0.060792 | 0.070155 | 0.076431 |
| **-11** | 0.010257 | 0.020503 | 0.032161 | 0.0446 | 0.056816 | 0.069732 | 0.083482 | 0.093431 |
| **-10** | 0.011207 | 0.023635 | 0.036838 | 0.049959 | 0.064777 | 0.080164 | 0.096399 | 0.110392 |
| **-9** | 0.01251 | 0.025325 | 0.040908 | 0.056659 | 0.072993 | 0.091273 | 0.109506 | 0.128508 |
| **-8** | 0.013277 | 0.02798 | 0.044317 | 0.061693 | 0.081137 | 0.102273 | 0.123199 | 0.146605 |
| **-7** | 0.01451 | 0.030036 | 0.048053 | 0.067414 | 0.089325 | 0.112352 | 0.137905 | 0.164596 |
| **-6** | 0.015635 | 0.032741 | 0.051218 | 0.072993 | 0.096269 | 0.123004 | 0.151971 | 0.18246 |
| **-5** | 0.016629 | 0.035487 | 0.055996 | 0.078574 | 0.104271 | 0.133372 | 0.166118 | 0.200766 |
| **-4** | 0.018365 | 0.038249 | 0.060378 | 0.084741 | 0.112613 | 0.144262 | 0.180553 | 0.220005 |
| **-3** | 0.019602 | 0.041492 | 0.065494 | 0.091996 | 0.121939 | 0.156095 | 0.195262 | 0.239448 |
| **-2** | 0.021151 | 0.044982 | 0.070795 | 0.099445 | 0.131501 | 0.168441 | 0.210942 | 0.2592 |
| **-1** | 0.022844 | 0.048017 | 0.07563 | 0.106733 | 0.141396 | 0.180864 | 0.226197 | 0.27824 |
| **0** | 0.023936 | 0.050533 | 0.079798 | 0.11294 | 0.149889 | 0.192047 | 0.240226 | 0.295969 |
| **1** | 0.024469 | 0.052107 | 0.082732 | 0.117459 | 0.156635 | 0.20115 | 0.252318 | 0.311237 |
| **2** | 0.024808 | 0.052831 | 0.084599 | 0.120614 | 0.161521 | 0.208157 | 0.261663 | 0.323251 |
| **3** | 0.024494 | 0.052609 | 0.08477 | 0.121944 | 0.164179 | 0.212551 | 0.268248 | 0.33206 |
| **4** | 0.023718 | 0.051694 | 0.083769 | 0.121635 | 0.164835 | 0.214644 | 0.271898 | 0.337904 |
| **5** | 0.022548 | 0.049604 | 0.081842 | 0.119844 | 0.163715 | 0.214661 | 0.27344 | 0.341081 |
| **6** | 0.021216 | 0.047505 | 0.079043 | 0.117022 | 0.161513 | 0.213343 | 0.273075 | 0.341925 |
| **7** | 0.019743 | 0.045006 | 0.076363 | 0.113762 | 0.158598 | 0.210883 | 0.271593 | 0.341388 |
| **8** | 0.018536 | 0.042925 | 0.073508 | 0.11086 | 0.155935 | 0.208634 | 0.269765 | 0.340304 |
| **9** | 0.017463 | 0.040845 | 0.071147 | 0.108506 | 0.153391 | 0.206354 | 0.267936 | 0.339098 |
| **10** | 0.0167 | 0.039561 | 0.069331 | 0.106326 | 0.151462 | 0.204661 | 0.266542 | 0.337835 |
| **11** | 0.015963 | 0.03823 | 0.067747 | 0.104952 | 0.149861 | 0.203004 | 0.265464 | 0.336801 |
| **12** | 0.015329 | 0.037512 | 0.06666 | 0.103735 | 0.148831 | 0.202133 | 0.264421 | 0.336206 |
| **13** | 0.01506 | 0.036989 | 0.06605 | 0.103107 | 0.148031 | 0.201434 | 0.263716 | 0.335942 |
| **14** | 0.014828 | 0.03647 | 0.065495 | 0.10248 | 0.147364 | 0.20125 | 0.263554 | 0.335582 |
| **15** | 0.014579 | 0.036096 | 0.065303 | 0.102357 | 0.146934 | 0.200941 | 0.26354 | 0.33529 |
| **16** | 0.014632 | 0.036082 | 0.065075 | 0.102256 | 0.147267 | 0.200677 | 0.263322 | 0.335401 |
| **17** | 0.01454 | 0.036071 | 0.065244 | 0.102265 | 0.147462 | 0.201098 | 0.263243 | 0.335515 |
| **18** | 0.01459 | 0.036154 | 0.065289 | 0.102333 | 0.147723 | 0.201105 | 0.263588 | 0.335555 |
| **19** | 0.014583 | 0.03635 | 0.065544 | 0.102644 | 0.14833 | 0.201403 | 0.263786 | 0.335816 |
| **20** | 0.0147 | 0.036565 | 0.065911 | 0.103326 | 0.148559 | 0.202044 | 0.26413 | 0.335997 |

**Table A.4:** Average Changes to BER for Neural Receiver Due to Dropout applied to 1st Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000195 | 0.000633 | 0.000146 | 0.000497 | 0.00061 | 0.000581 | 0.000523 | 0.001388 |
| **-19** | 0.000165 | 0.000564 | 0.000718 | 0.000828 | 0.001188 | 0.001907 | 0.001664 | 0.002424 |
| **-18** | 0.000195 | 0.000707 | 0.001181 | 0.001748 | 0.001774 | 0.002371 | 0.003034 | 0.004509 |
| **-17** | 0.000864 | 0.000758 | 0.001487 | 0.002267 | 0.002848 | 0.004 | 0.005063 | 0.006273 |
| **-16** | 0.000582 | 0.000794 | 0.002221 | 0.003354 | 0.00406 | 0.005648 | 0.007089 | 0.008737 |
| **-15** | 0.000297 | 0.001439 | 0.00313 | 0.004326 | 0.005498 | 0.006965 | 0.008534 | 0.01206 |
| **-14** | 0.000722 | 0.002061 | 0.003091 | 0.004644 | 0.006091 | 0.007819 | 0.01054 | 0.014546 |
| **-13** | 0.000674 | 0.00282 | 0.0034 | 0.005267 | 0.007324 | 0.009477 | 0.012305 | 0.016617 |
| **-12** | 0.001439 | 0.002377 | 0.004389 | 0.005611 | 0.007936 | 0.010385 | 0.014287 | 0.018331 |
| **-11** | 0.001316 | 0.003356 | 0.004799 | 0.006461 | 0.009 | 0.011483 | 0.014815 | 0.020871 |
| **-10** | 0.001105 | 0.002599 | 0.004749 | 0.006549 | 0.009115 | 0.012126 | 0.015971 | 0.02214 |
| **-9** | 0.001388 | 0.002754 | 0.00435 | 0.006747 | 0.009405 | 0.012503 | 0.0169 | 0.023709 |
| **-8** | 0.001539 | 0.002577 | 0.004184 | 0.006323 | 0.008588 | 0.012362 | 0.01698 | 0.024431 |
| **-7** | 0.001082 | 0.002578 | 0.004175 | 0.005459 | 0.008489 | 0.011452 | 0.016338 | 0.024088 |
| **-6** | 0.000959 | 0.002171 | 0.003406 | 0.005341 | 0.007482 | 0.01051 | 0.014983 | 0.023228 |
| **-5** | 0.000862 | 0.00206 | 0.002853 | 0.004694 | 0.006697 | 0.009525 | 0.013822 | 0.022123 |
| **-4** | 0.000979 | 0.001547 | 0.002984 | 0.004079 | 0.005985 | 0.008534 | 0.012714 | 0.021145 |
| **-3** | 0.000685 | 0.001359 | 0.002429 | 0.003595 | 0.005382 | 0.007596 | 0.011597 | 0.01956 |
| **-2** | 0.000325 | 0.001351 | 0.002099 | 0.003209 | 0.004788 | 0.007278 | 0.011127 | 0.018913 |
| **-1** | 0.000481 | 0.001184 | 0.001929 | 0.002998 | 0.004199 | 0.006487 | 0.010257 | 0.018046 |
| **0** | 0.000453 | 0.001216 | 0.001655 | 0.002802 | 0.003966 | 0.006143 | 0.009677 | 0.017193 |
| **1** | 0.000421 | 0.000812 | 0.001725 | 0.002535 | 0.003845 | 0.005581 | 0.008987 | 0.016432 |
| **2** | 0.000412 | 0.000784 | 0.001439 | 0.00222 | 0.003489 | 0.005316 | 0.008481 | 0.015484 |
| **3** | 0.000274 | 0.000811 | 0.001301 | 0.001789 | 0.003091 | 0.004605 | 0.007687 | 0.014247 |
| **4** | 0.000166 | 0.000588 | 0.001056 | 0.001765 | 0.002592 | 0.00408 | 0.00664 | 0.012608 |
| **5** | 0.000253 | 0.000446 | 0.000831 | 0.00135 | 0.002032 | 0.003298 | 0.005645 | 0.010838 |
| **6** | 0.000114 | 0.000412 | 0.000621 | 0.000988 | 0.00166 | 0.002748 | 0.004318 | 0.00894 |
| **7** | 0.000129 | 0.000223 | 0.000456 | 0.000716 | 0.001162 | 0.001857 | 0.003308 | 0.007207 |
| **8** | 2.91E-05 | 0.000171 | 0.000357 | 0.000525 | 0.000873 | 0.001405 | 0.002443 | 0.005678 |
| **9** | 2.96E-05 | 0.000137 | 0.000232 | 0.000385 | 0.000627 | 0.001056 | 0.001821 | 0.004566 |
| **10** | 1.58E-05 | 8.65E-05 | 0.000204 | 0.000247 | 0.000465 | 0.000775 | 0.001451 | 0.003825 |
| **11** | 4.56E-05 | 7.91E-05 | 0.000141 | 0.000201 | 0.000375 | 0.000673 | 0.001312 | 0.00341 |
| **12** | 4.47E-05 | 7.12E-05 | 0.000107 | 0.000191 | 0.000351 | 0.000646 | 0.001105 | 0.003117 |
| **13** | 2.61E-05 | 4.94E-05 | 9.16E-05 | 0.000204 | 0.000298 | 0.00053 | 0.001097 | 0.002994 |
| **14** | 8.26E-06 | 8.83E-05 | 0.000142 | 0.000118 | 0.000289 | 0.000523 | 0.001144 | 0.002935 |
| **15** | 4.1E-05 | 5.63E-05 | 0.000106 | 0.000207 | 0.000345 | 0.000572 | 0.001129 | 0.003015 |
| **16** | 4.64E-06 | 9.64E-05 | 0.000126 | 0.000224 | 0.000386 | 0.000654 | 0.001178 | 0.003273 |
| **17** | 5.13E-05 | 7.1E-05 | 0.000163 | 0.000206 | 0.000361 | 0.000648 | 0.00123 | 0.00348 |
| **18** | 9.09E-05 | 4.55E-05 | 0.000145 | 0.000294 | 0.000435 | 0.000734 | 0.001467 | 0.003718 |
| **19** | 4.4E-05 | 9.23E-05 | 0.000193 | 0.000268 | 0.000482 | 0.000895 | 0.001644 | 0.004071 |
| **20** | 2.2E-05 | 0.000196 | 0.000195 | 0.000456 | 0.000562 | 0.000941 | 0.001823 | 0.004418 |

**Table A.5:** Average Changes to BER for Neural Receiver Due to Dropout applied to 2nd Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000649 | 0.000377 | 0.000957 | 0.001712 | 0.001521 | 0.001854 | 0.002072 | 0.002524 |
| **-19** | 0.000424 | 0.001362 | 0.001394 | 0.002467 | 0.002381 | 0.00321 | 0.004342 | 0.004324 |
| **-18** | 0.000828 | 0.001555 | 0.002179 | 0.003086 | 0.004494 | 0.004929 | 0.00598 | 0.006863 |
| **-17** | 0.001186 | 0.00274 | 0.003559 | 0.005498 | 0.006883 | 0.008549 | 0.009567 | 0.011568 |
| **-16** | 0.001925 | 0.003801 | 0.005777 | 0.007447 | 0.01027 | 0.012216 | 0.014822 | 0.0176 |
| **-15** | 0.002494 | 0.005355 | 0.008065 | 0.010815 | 0.014806 | 0.018481 | 0.021367 | 0.025738 |
| **-14** | 0.003495 | 0.00689 | 0.011345 | 0.015878 | 0.020852 | 0.025067 | 0.029925 | 0.034688 |
| **-13** | 0.004645 | 0.009149 | 0.015317 | 0.020328 | 0.026488 | 0.03312 | 0.0386 | 0.044772 |
| **-12** | 0.005308 | 0.012133 | 0.018271 | 0.025167 | 0.033013 | 0.040729 | 0.048381 | 0.05691 |
| **-11** | 0.006245 | 0.013398 | 0.02152 | 0.030254 | 0.038995 | 0.049258 | 0.059121 | 0.068825 |
| **-10** | 0.006525 | 0.01479 | 0.022942 | 0.033775 | 0.045013 | 0.057148 | 0.069543 | 0.082276 |
| **-9** | 0.006175 | 0.014418 | 0.023818 | 0.035361 | 0.048745 | 0.06314 | 0.078227 | 0.095256 |
| **-8** | 0.006282 | 0.013608 | 0.022707 | 0.035071 | 0.049754 | 0.067009 | 0.085451 | 0.10634 |
| **-7** | 0.005374 | 0.012157 | 0.021302 | 0.032968 | 0.048282 | 0.067288 | 0.09022 | 0.115666 |
| **-6** | 0.004396 | 0.010553 | 0.018108 | 0.029666 | 0.044999 | 0.064336 | 0.090313 | 0.121188 |
| **-5** | 0.003883 | 0.008623 | 0.01565 | 0.025155 | 0.03896 | 0.058513 | 0.085647 | 0.122764 |
| **-4** | 0.002907 | 0.00724 | 0.012721 | 0.021054 | 0.032773 | 0.05045 | 0.077518 | 0.119671 |
| **-3** | 0.00228 | 0.005656 | 0.010346 | 0.01681 | 0.026698 | 0.041822 | 0.067475 | 0.109954 |
| **-2** | 0.001816 | 0.004264 | 0.008138 | 0.013369 | 0.02107 | 0.033799 | 0.056022 | 0.09729 |
| **-1** | 0.001222 | 0.003455 | 0.006354 | 0.010052 | 0.016632 | 0.026677 | 0.044942 | 0.081991 |
| **0** | 0.00097 | 0.002357 | 0.004672 | 0.007711 | 0.012701 | 0.020704 | 0.035691 | 0.067259 |
| **1** | 0.000673 | 0.001808 | 0.003681 | 0.00598 | 0.00992 | 0.016594 | 0.02853 | 0.055189 |
| **2** | 0.000612 | 0.001429 | 0.002852 | 0.004811 | 0.007847 | 0.01318 | 0.022909 | 0.045822 |
| **3** | 0.000581 | 0.001158 | 0.002281 | 0.003929 | 0.006493 | 0.010792 | 0.018962 | 0.038272 |
| **4** | 0.000413 | 0.000989 | 0.001742 | 0.003094 | 0.005205 | 0.008524 | 0.015387 | 0.032502 |
| **5** | 0.000286 | 0.000723 | 0.001261 | 0.002327 | 0.003961 | 0.006654 | 0.012317 | 0.026636 |
| **6** | 0.000137 | 0.000489 | 0.000855 | 0.001738 | 0.002763 | 0.004931 | 0.00934 | 0.02169 |
| **7** | 0.000155 | 0.000335 | 0.000598 | 0.001089 | 0.001906 | 0.003517 | 0.006811 | 0.01707 |
| **8** | 0.000128 | 0.000198 | 0.000359 | 0.000736 | 0.001307 | 0.002337 | 0.004839 | 0.013439 |
| **9** | 5.48E-05 | 0.00017 | 0.000238 | 0.000498 | 0.000773 | 0.001593 | 0.003397 | 0.010591 |
| **10** | 2.68E-05 | 0.00015 | 0.000197 | 0.000274 | 0.000567 | 0.001092 | 0.002493 | 0.008366 |
| **11** | 1.97E-05 | 4.77E-05 | 0.000173 | 0.000263 | 0.000366 | 0.000795 | 0.001846 | 0.007082 |
| **12** | 2.24E-05 | 3.91E-05 | 0.000103 | 0.000155 | 0.000349 | 0.000643 | 0.001474 | 0.00605 |
| **13** | 6.75E-05 | 5.68E-05 | 0.000102 | 0.000155 | 0.000272 | 0.000559 | 0.00138 | 0.005451 |
| **14** | 2.54E-05 | 6.92E-05 | 6.01E-05 | 0.000159 | 0.000308 | 0.000506 | 0.001262 | 0.005345 |
| **15** | 9.02E-07 | 5.4E-05 | 9.74E-05 | 0.000188 | 0.000264 | 0.000475 | 0.001267 | 0.005362 |
| **16** | 1.37E-05 | 5.17E-05 | 0.000108 | 0.000203 | 0.000375 | 0.000608 | 0.001409 | 0.005619 |
| **17** | 7.86E-05 | 8.9E-05 | 0.000111 | 0.000217 | 0.000351 | 0.000658 | 0.001481 | 0.00607 |
| **18** | 1.45E-05 | 2.77E-05 | 0.000115 | 0.000245 | 0.000417 | 0.000736 | 0.00158 | 0.006849 |
| **19** | 2.74E-05 | 3.54E-06 | 0.000215 | 0.000273 | 0.00055 | 0.000858 | 0.001891 | 0.007702 |
| **20** | 7.67E-05 | 6.29E-05 | 0.00019 | 0.00028 | 0.000602 | 0.001052 | 0.002325 | 0.008904 |

**Table A.6:** Average Changes to BER for Neural Receiver Due to Dropout applied to 3rd Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 9.92243E-05 | 0.000113 | 0.000409 | 0.000173 | 0.000439 | 0.000704 | 0.000684 | 0.00142 |
| **-19** | 0.000124777 | 0.00056 | 0.000865 | 0.000519 | 0.000909 | 0.00072 | 0.001273 | 0.001634 |
| **-18** | 8.26805E-05 | 0.000737 | 0.000543 | 0.000733 | 0.000821 | 0.001298 | 0.002526 | 0.003755 |
| **-17** | 0.000288928 | 0.001014 | 0.000826 | 0.001696 | 0.002413 | 0.002778 | 0.004246 | 0.005571 |
| **-16** | 0.000498205 | 0.001097 | 0.002338 | 0.0024 | 0.004076 | 0.005051 | 0.006594 | 0.008976 |
| **-15** | 0.000379825 | 0.001879 | 0.002772 | 0.003254 | 0.00565 | 0.00672 | 0.009173 | 0.012577 |
| **-14** | 0.001158435 | 0.001438 | 0.003177 | 0.004756 | 0.006422 | 0.008045 | 0.011244 | 0.015729 |
| **-13** | 0.000870043 | 0.001795 | 0.003237 | 0.005111 | 0.006631 | 0.009806 | 0.01264 | 0.018872 |
| **-12** | 0.00098526 | 0.002295 | 0.004006 | 0.005682 | 0.007694 | 0.010115 | 0.014224 | 0.020194 |
| **-11** | 0.00138208 | 0.002773 | 0.003997 | 0.005622 | 0.008234 | 0.011512 | 0.01585 | 0.022775 |
| **-10** | 0.001136574 | 0.002449 | 0.004852 | 0.006151 | 0.008653 | 0.012569 | 0.017601 | 0.025294 |
| **-9** | 0.00139969 | 0.003158 | 0.004264 | 0.006739 | 0.009393 | 0.012954 | 0.018148 | 0.026844 |
| **-8** | 0.001138451 | 0.003025 | 0.004693 | 0.006533 | 0.009965 | 0.013639 | 0.018683 | 0.027855 |
| **-7** | 0.001110114 | 0.00263 | 0.004866 | 0.007058 | 0.009866 | 0.013497 | 0.018815 | 0.028793 |
| **-6** | 0.001484868 | 0.00277 | 0.004707 | 0.006825 | 0.009652 | 0.013457 | 0.019344 | 0.029021 |
| **-5** | 0.001414946 | 0.002372 | 0.004372 | 0.006622 | 0.009448 | 0.013525 | 0.01917 | 0.029444 |
| **-4** | 0.001127067 | 0.00249 | 0.00429 | 0.00649 | 0.009315 | 0.013373 | 0.019071 | 0.029433 |
| **-3** | 0.001305944 | 0.002569 | 0.004055 | 0.006574 | 0.009265 | 0.013138 | 0.019267 | 0.029708 |
| **-2** | 0.001193457 | 0.002473 | 0.004122 | 0.006261 | 0.009064 | 0.012986 | 0.019109 | 0.029684 |
| **-1** | 0.001100499 | 0.002399 | 0.004004 | 0.006177 | 0.009041 | 0.012923 | 0.018983 | 0.029704 |
| **0** | 0.000922705 | 0.002429 | 0.003711 | 0.00604 | 0.008783 | 0.012659 | 0.0188 | 0.029687 |
| **1** | 0.000960095 | 0.002149 | 0.003699 | 0.005776 | 0.008639 | 0.012625 | 0.018737 | 0.029677 |
| **2** | 0.000869839 | 0.002142 | 0.003404 | 0.005784 | 0.008678 | 0.012628 | 0.018778 | 0.029923 |
| **3** | 0.000761126 | 0.00201 | 0.003324 | 0.005278 | 0.008024 | 0.011898 | 0.018163 | 0.029684 |
| **4** | 0.000706937 | 0.001699 | 0.002996 | 0.004867 | 0.007264 | 0.0111 | 0.017001 | 0.028049 |
| **5** | 0.000529016 | 0.001363 | 0.00245 | 0.004077 | 0.00614 | 0.009575 | 0.015262 | 0.026114 |
| **6** | 0.000408859 | 0.001066 | 0.001825 | 0.003173 | 0.005125 | 0.008147 | 0.013274 | 0.0235 |
| **7** | 0.000310047 | 0.00071 | 0.001282 | 0.002298 | 0.00393 | 0.006461 | 0.011299 | 0.020866 |
| **8** | 0.000155871 | 0.000472 | 0.000935 | 0.001648 | 0.002907 | 0.005074 | 0.009387 | 0.018464 |
| **9** | 0.000124563 | 0.000263 | 0.000614 | 0.001212 | 0.002193 | 0.004049 | 0.007896 | 0.016446 |
| **10** | 9.23147E-05 | 0.000219 | 0.00047 | 0.000806 | 0.001599 | 0.003257 | 0.006822 | 0.015098 |
| **11** | 4.27726E-05 | 0.000123 | 0.000323 | 0.00062 | 0.001281 | 0.00285 | 0.006075 | 0.01392 |
| **12** | 6.3729E-06 | 6.1E-05 | 0.000219 | 0.000548 | 0.001152 | 0.002424 | 0.005538 | 0.013233 |
| **13** | 2.64316E-05 | 5.98E-05 | 0.000216 | 0.000481 | 0.000995 | 0.002289 | 0.005179 | 0.012807 |
| **14** | 6.8027E-05 | 5.98E-05 | 0.000165 | 0.000371 | 0.000925 | 0.002109 | 0.005039 | 0.012586 |
| **15** | 1.21747E-05 | 8.84E-05 | 0.000206 | 0.000398 | 0.000961 | 0.002086 | 0.005078 | 0.012736 |
| **16** | 3.00663E-05 | 0.000107 | 0.000182 | 0.000381 | 0.00093 | 0.002162 | 0.005122 | 0.012899 |
| **17** | 6.56901E-06 | 6.43E-05 | 0.000225 | 0.000421 | 0.001011 | 0.002214 | 0.005295 | 0.01307 |
| **18** | 9.74114E-05 | 5.45E-05 | 0.000198 | 0.000491 | 0.001052 | 0.002396 | 0.005496 | 0.01343 |
| **19** | 3.26853E-05 | 4.99E-05 | 0.000248 | 0.000529 | 0.00118 | 0.002534 | 0.0056 | 0.013967 |
| **20** | 3.72414E-05 | 0.000114 | 0.000399 | 0.000712 | 0.001314 | 0.002717 | 0.00608 | 0.014453 |

**Table A.7:** Average Changes to BER for Neural Receiver Due to Dropout applied to 4th Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000218 | 0.001283 | 0.000985 | 0.001408 | 0.001134 | 0.002132 | 0.0023 | 0.002682 |
| **-19** | 0.000773 | 0.001085 | 0.002353 | 0.002665 | 0.003128 | 0.003214 | 0.003876 | 0.004272 |
| **-18** | 0.000944 | 0.002249 | 0.003423 | 0.004399 | 0.004611 | 0.005466 | 0.005913 | 0.006595 |
| **-17** | 0.001488 | 0.003877 | 0.004803 | 0.0064 | 0.007549 | 0.00893 | 0.010646 | 0.011118 |
| **-16** | 0.002322 | 0.004353 | 0.007099 | 0.009214 | 0.011694 | 0.013463 | 0.015255 | 0.016808 |
| **-15** | 0.002358 | 0.00526 | 0.008661 | 0.011676 | 0.015498 | 0.01782 | 0.020435 | 0.023336 |
| **-14** | 0.001912 | 0.00546 | 0.008837 | 0.013931 | 0.017494 | 0.021392 | 0.025533 | 0.029276 |
| **-13** | 0.001407 | 0.004667 | 0.008425 | 0.013949 | 0.018624 | 0.024584 | 0.029384 | 0.034663 |
| **-12** | 0.000799 | 0.003874 | 0.007979 | 0.013327 | 0.019223 | 0.025532 | 0.032363 | 0.039777 |
| **-11** | 0.0005 | 0.002798 | 0.006486 | 0.011776 | 0.018508 | 0.026123 | 0.033818 | 0.04367 |
| **-10** | 0.000246 | 0.002583 | 0.005618 | 0.011541 | 0.017676 | 0.025507 | 0.035639 | 0.046189 |
| **-9** | 0.000655 | 0.002303 | 0.005711 | 0.010981 | 0.01717 | 0.025367 | 0.0361 | 0.049119 |
| **-8** | 0.000717 | 0.002625 | 0.006015 | 0.01049 | 0.0163 | 0.025059 | 0.035934 | 0.050025 |
| **-7** | 0.001026 | 0.002833 | 0.00602 | 0.010509 | 0.016247 | 0.024386 | 0.035418 | 0.050141 |
| **-6** | 0.001122 | 0.003518 | 0.006199 | 0.010372 | 0.015744 | 0.024482 | 0.034961 | 0.050609 |
| **-5** | 0.001243 | 0.003392 | 0.006372 | 0.010382 | 0.016175 | 0.024044 | 0.034713 | 0.050631 |
| **-4** | 0.001606 | 0.003696 | 0.006891 | 0.010693 | 0.016253 | 0.023631 | 0.033951 | 0.050383 |
| **-3** | 0.001299 | 0.003762 | 0.006518 | 0.010522 | 0.015976 | 0.023033 | 0.033622 | 0.050208 |
| **-2** | 0.001629 | 0.003708 | 0.006537 | 0.010433 | 0.015692 | 0.022805 | 0.033343 | 0.049762 |
| **-1** | 0.001322 | 0.003421 | 0.006528 | 0.010251 | 0.015504 | 0.022817 | 0.033251 | 0.049704 |
| **0** | 0.001409 | 0.003526 | 0.006515 | 0.010249 | 0.015402 | 0.022793 | 0.03349 | 0.05107 |
| **1** | 0.001498 | 0.003822 | 0.006747 | 0.010648 | 0.016145 | 0.023496 | 0.0347 | 0.052074 |
| **2** | 0.001318 | 0.003717 | 0.006631 | 0.010776 | 0.016132 | 0.023804 | 0.034947 | 0.053121 |
| **3** | 0.001521 | 0.003734 | 0.006263 | 0.010417 | 0.01562 | 0.023531 | 0.034581 | 0.053026 |
| **4** | 0.001334 | 0.003122 | 0.005835 | 0.00964 | 0.014545 | 0.022076 | 0.033063 | 0.051633 |
| **5** | 0.001148 | 0.002722 | 0.005029 | 0.008379 | 0.013045 | 0.019981 | 0.031023 | 0.049705 |
| **6** | 0.000856 | 0.002175 | 0.004059 | 0.00682 | 0.011072 | 0.017557 | 0.028092 | 0.046225 |
| **7** | 0.000651 | 0.001542 | 0.003084 | 0.005491 | 0.008955 | 0.015 | 0.025029 | 0.043134 |
| **8** | 0.000454 | 0.001069 | 0.002204 | 0.004053 | 0.007262 | 0.012548 | 0.022015 | 0.039746 |
| **9** | 0.000259 | 0.000728 | 0.001581 | 0.003073 | 0.0057 | 0.010482 | 0.019406 | 0.036797 |
| **10** | 0.000151 | 0.000494 | 0.001101 | 0.002274 | 0.004582 | 0.008824 | 0.017342 | 0.034364 |
| **11** | 0.000116 | 0.00037 | 0.000808 | 0.001782 | 0.003764 | 0.007672 | 0.015615 | 0.032373 |
| **12** | 6.06E-05 | 0.000257 | 0.000622 | 0.001443 | 0.003155 | 0.00667 | 0.014173 | 0.030531 |
| **13** | 7.32E-05 | 0.000243 | 0.00055 | 0.001205 | 0.002739 | 0.006033 | 0.013231 | 0.029292 |
| **14** | 0.000112 | 0.000203 | 0.000503 | 0.001094 | 0.002439 | 0.005591 | 0.012476 | 0.02814 |
| **15** | 5.16E-05 | 0.000162 | 0.000452 | 0.000922 | 0.002301 | 0.005223 | 0.012051 | 0.027535 |
| **16** | 4.06E-05 | 0.000107 | 0.00041 | 0.000883 | 0.002126 | 0.004892 | 0.01155 | 0.026989 |
| **17** | 1.84E-05 | 0.000178 | 0.000365 | 0.000886 | 0.002065 | 0.004708 | 0.011328 | 0.026492 |
| **18** | 3.31E-05 | 0.000131 | 0.000368 | 0.000819 | 0.001958 | 0.004588 | 0.011245 | 0.026516 |
| **19** | 8.42E-05 | 0.000124 | 0.000413 | 0.00084 | 0.001932 | 0.004575 | 0.010926 | 0.026317 |
| **20** | 9.56E-05 | 9.92E-05 | 0.000359 | 0.000833 | 0.001987 | 0.004491 | 0.011001 | 0.026083 |

**Table A.8:** Average Changes to BER for Neural Receiver Due to Dropout applied to 5th Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000372 | 0.000211 | 0.000738 | 0.000308 | 0.000114 | 0.000195 | 0.000756 | 0.000387 |
| **-19** | 0.000419 | 0.000134 | 0.00068 | 0.000343 | 0.000673 | 0.00069 | 0.001237 | 0.001234 |
| **-18** | 0.000269 | 0.000513 | 0.0008 | 0.000428 | 0.001258 | 0.00182 | 0.002742 | 0.003097 |
| **-17** | 0.000253 | 0.000961 | 0.001156 | 0.001354 | 0.00244 | 0.003058 | 0.004628 | 0.006472 |
| **-16** | 0.000618 | 0.001298 | 0.00156 | 0.002664 | 0.00451 | 0.005625 | 0.007482 | 0.010537 |
| **-15** | 0.001011 | 0.002269 | 0.003138 | 0.004856 | 0.006748 | 0.008359 | 0.011047 | 0.014993 |
| **-14** | 0.001258 | 0.002925 | 0.004235 | 0.005752 | 0.008034 | 0.011281 | 0.014888 | 0.019653 |
| **-13** | 0.001164 | 0.003749 | 0.005116 | 0.00725 | 0.009824 | 0.013383 | 0.018507 | 0.02481 |
| **-12** | 0.001883 | 0.004082 | 0.005522 | 0.008477 | 0.012098 | 0.015853 | 0.021909 | 0.029753 |
| **-11** | 0.001893 | 0.004471 | 0.00715 | 0.010204 | 0.014054 | 0.018384 | 0.024914 | 0.034523 |
| **-10** | 0.001979 | 0.004077 | 0.007261 | 0.010163 | 0.014579 | 0.020158 | 0.027504 | 0.038637 |
| **-9** | 0.001948 | 0.004463 | 0.007349 | 0.010182 | 0.014635 | 0.021199 | 0.029225 | 0.042152 |
| **-8** | 0.001843 | 0.003688 | 0.006431 | 0.009881 | 0.01452 | 0.021475 | 0.0304 | 0.044391 |
| **-7** | 0.001413 | 0.003346 | 0.006144 | 0.009627 | 0.014086 | 0.020172 | 0.030226 | 0.045518 |
| **-6** | 0.001368 | 0.003108 | 0.005533 | 0.008683 | 0.013256 | 0.019745 | 0.029613 | 0.045454 |
| **-5** | 0.001065 | 0.003011 | 0.005385 | 0.008457 | 0.012675 | 0.01909 | 0.028931 | 0.045014 |
| **-4** | 0.001159 | 0.003134 | 0.005083 | 0.008196 | 0.012548 | 0.018771 | 0.028174 | 0.044289 |
| **-3** | 0.001203 | 0.003115 | 0.005292 | 0.008288 | 0.012583 | 0.018455 | 0.028271 | 0.044543 |
| **-2** | 0.001334 | 0.003063 | 0.005416 | 0.008662 | 0.012944 | 0.019002 | 0.028254 | 0.044637 |
| **-1** | 0.001484 | 0.003352 | 0.005827 | 0.008785 | 0.01323 | 0.019415 | 0.028935 | 0.045293 |
| **0** | 0.001548 | 0.003534 | 0.006084 | 0.009327 | 0.013513 | 0.020211 | 0.0296 | 0.046581 |
| **1** | 0.00169 | 0.003668 | 0.006228 | 0.009549 | 0.014138 | 0.020617 | 0.030631 | 0.047728 |
| **2** | 0.001582 | 0.003627 | 0.006281 | 0.009536 | 0.014259 | 0.02088 | 0.031059 | 0.049029 |
| **3** | 0.001601 | 0.003404 | 0.005947 | 0.00922 | 0.014168 | 0.020575 | 0.0313 | 0.049339 |
| **4** | 0.00134 | 0.003247 | 0.005499 | 0.00863 | 0.013293 | 0.020022 | 0.030423 | 0.048748 |
| **5** | 0.001181 | 0.002716 | 0.004888 | 0.007942 | 0.012089 | 0.018717 | 0.028889 | 0.047552 |
| **6** | 0.000889 | 0.002147 | 0.004077 | 0.006817 | 0.010708 | 0.016822 | 0.02701 | 0.04541 |
| **7** | 0.00065 | 0.001684 | 0.003293 | 0.005606 | 0.009142 | 0.014871 | 0.024636 | 0.043016 |
| **8** | 0.000502 | 0.001242 | 0.002441 | 0.004439 | 0.007695 | 0.013137 | 0.022515 | 0.040157 |
| **9** | 0.000299 | 0.000878 | 0.001846 | 0.003598 | 0.006529 | 0.011604 | 0.020533 | 0.03837 |
| **10** | 0.00023 | 0.000666 | 0.00149 | 0.002954 | 0.005536 | 0.010299 | 0.019084 | 0.036476 |
| **11** | 0.00022 | 0.000525 | 0.001145 | 0.00245 | 0.004903 | 0.009476 | 0.017991 | 0.034913 |
| **12** | 0.000126 | 0.000417 | 0.000989 | 0.002209 | 0.004412 | 0.00872 | 0.016958 | 0.03408 |
| **13** | 0.000131 | 0.000317 | 0.000882 | 0.001984 | 0.004114 | 0.008222 | 0.01648 | 0.033156 |
| **14** | 6.12E-05 | 0.000342 | 0.00077 | 0.00183 | 0.003825 | 0.007823 | 0.016042 | 0.032625 |
| **15** | 0.000109 | 0.000321 | 0.000759 | 0.001763 | 0.003754 | 0.007706 | 0.015684 | 0.032481 |
| **16** | 0.000133 | 0.000314 | 0.000818 | 0.001743 | 0.003651 | 0.007593 | 0.015784 | 0.032217 |
| **17** | 0.000126 | 0.000366 | 0.000816 | 0.001688 | 0.003643 | 0.007682 | 0.015768 | 0.032515 |
| **18** | 7.46E-05 | 0.000314 | 0.000816 | 0.001697 | 0.00371 | 0.007749 | 0.015765 | 0.032687 |
| **19** | 0.000124 | 0.000418 | 0.00085 | 0.001863 | 0.003837 | 0.007758 | 0.016208 | 0.033192 |
| **20** | 0.000168 | 0.000434 | 0.000922 | 0.002003 | 0.004015 | 0.00817 | 0.016573 | 0.033714 |

**Table A.9:** Average Changes to BER for Neural Receiver Due to Dropout applied to 6th Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.00048 | 0.000773 | 0.000647 | 0.001702 | 0.002128 | 0.002911 | 0.002606 | 0.00261 |
| **-19** | 0.001171 | 0.001469 | 0.00207 | 0.003228 | 0.003725 | 0.003741 | 0.004666 | 0.005387 |
| **-18** | 0.00153 | 0.002976 | 0.003874 | 0.004568 | 0.00597 | 0.006677 | 0.007823 | 0.008537 |
| **-17** | 0.002043 | 0.004122 | 0.006647 | 0.008013 | 0.009549 | 0.010127 | 0.0118 | 0.012493 |
| **-16** | 0.002556 | 0.005363 | 0.008257 | 0.0103 | 0.01282 | 0.0152 | 0.016387 | 0.018274 |
| **-15** | 0.003375 | 0.006582 | 0.009475 | 0.012936 | 0.016304 | 0.019356 | 0.021888 | 0.024704 |
| **-14** | 0.003126 | 0.007052 | 0.010917 | 0.014757 | 0.018908 | 0.023239 | 0.027778 | 0.031124 |
| **-13** | 0.00413 | 0.006661 | 0.011681 | 0.016224 | 0.021398 | 0.026187 | 0.032159 | 0.036898 |
| **-12** | 0.00378 | 0.006679 | 0.011822 | 0.017685 | 0.023314 | 0.029637 | 0.036404 | 0.043691 |
| **-11** | 0.004131 | 0.00727 | 0.012529 | 0.018355 | 0.024968 | 0.032343 | 0.040568 | 0.050455 |
| **-10** | 0.003379 | 0.007267 | 0.012868 | 0.019107 | 0.026206 | 0.035033 | 0.044726 | 0.05627 |
| **-9** | 0.003092 | 0.007719 | 0.012817 | 0.019855 | 0.027702 | 0.037012 | 0.04792 | 0.061408 |
| **-8** | 0.003297 | 0.008019 | 0.013088 | 0.020036 | 0.028073 | 0.037764 | 0.050425 | 0.065962 |
| **-7** | 0.00347 | 0.00715 | 0.01285 | 0.0195 | 0.027528 | 0.037764 | 0.051748 | 0.068503 |
| **-6** | 0.003292 | 0.006979 | 0.01202 | 0.018266 | 0.026676 | 0.037314 | 0.05162 | 0.070231 |
| **-5** | 0.003018 | 0.006621 | 0.01142 | 0.017664 | 0.025296 | 0.036048 | 0.0503 | 0.070702 |
| **-4** | 0.002733 | 0.006496 | 0.010661 | 0.016547 | 0.024187 | 0.03477 | 0.04911 | 0.069929 |
| **-3** | 0.002739 | 0.005867 | 0.010655 | 0.016205 | 0.023831 | 0.033445 | 0.048369 | 0.069251 |
| **-2** | 0.002408 | 0.00609 | 0.010391 | 0.016084 | 0.023003 | 0.033122 | 0.04749 | 0.069119 |
| **-1** | 0.0028 | 0.005746 | 0.010003 | 0.015673 | 0.023078 | 0.033138 | 0.047153 | 0.070267 |
| **0** | 0.002503 | 0.005624 | 0.009987 | 0.015352 | 0.023046 | 0.033176 | 0.048008 | 0.070954 |
| **1** | 0.002532 | 0.005826 | 0.009986 | 0.0156 | 0.02323 | 0.033534 | 0.048429 | 0.0718 |
| **2** | 0.002495 | 0.005764 | 0.009961 | 0.015523 | 0.02309 | 0.03356 | 0.048597 | 0.072657 |
| **3** | 0.002458 | 0.005488 | 0.009772 | 0.015136 | 0.02271 | 0.033408 | 0.048684 | 0.073014 |
| **4** | 0.002276 | 0.00515 | 0.009138 | 0.014593 | 0.022109 | 0.032581 | 0.04822 | 0.072655 |
| **5** | 0.001962 | 0.004595 | 0.008392 | 0.013436 | 0.020806 | 0.031144 | 0.047246 | 0.07133 |
| **6** | 0.001614 | 0.003979 | 0.007481 | 0.012168 | 0.019143 | 0.029375 | 0.044711 | 0.069852 |
| **7** | 0.001283 | 0.003231 | 0.006348 | 0.01089 | 0.017253 | 0.027382 | 0.042534 | 0.067885 |
| **8** | 0.000965 | 0.002565 | 0.00527 | 0.009369 | 0.015669 | 0.025358 | 0.040472 | 0.065429 |
| **9** | 0.000709 | 0.002148 | 0.004392 | 0.008239 | 0.014234 | 0.023526 | 0.038348 | 0.063403 |
| **10** | 0.000563 | 0.001689 | 0.003771 | 0.007126 | 0.012783 | 0.021862 | 0.036497 | 0.06147 |
| **11** | 0.000416 | 0.001407 | 0.003164 | 0.006479 | 0.011724 | 0.020706 | 0.035179 | 0.059749 |
| **12** | 0.000363 | 0.001161 | 0.0028 | 0.005933 | 0.011064 | 0.019922 | 0.034284 | 0.059048 |
| **13** | 0.000295 | 0.001029 | 0.00255 | 0.005496 | 0.010577 | 0.019001 | 0.033589 | 0.058326 |
| **14** | 0.000278 | 0.000941 | 0.002397 | 0.005301 | 0.010055 | 0.018906 | 0.03306 | 0.058101 |
| **15** | 0.000245 | 0.000872 | 0.002235 | 0.005012 | 0.009933 | 0.018543 | 0.032926 | 0.05813 |
| **16** | 0.000271 | 0.000833 | 0.002247 | 0.004916 | 0.009875 | 0.018395 | 0.032797 | 0.057846 |
| **17** | 0.000194 | 0.000924 | 0.002322 | 0.004975 | 0.01002 | 0.018458 | 0.032703 | 0.058458 |
| **18** | 0.000206 | 0.000883 | 0.002314 | 0.005032 | 0.010113 | 0.018623 | 0.033339 | 0.058736 |
| **19** | 0.000256 | 0.001024 | 0.00243 | 0.005135 | 0.010305 | 0.019052 | 0.033937 | 0.05903 |
| **20** | 0.000228 | 0.001064 | 0.002525 | 0.005517 | 0.010563 | 0.019373 | 0.034518 | 0.060642 |

**Table A.10:** Average Changes to BER for Neural Receiver Due to Dropout applied to 7th Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.001148 | 0.000154 | 0.000127 | 0.000254 | 0.000496 | 0.000781 | 0.001373 | 0.001366 |
| **-19** | 0.000328 | 0.000202 | 0.000192 | 0.001248 | 0.001057 | 0.001349 | 0.001827 | 0.002912 |
| **-18** | 0.000266 | 0.0004 | 0.000965 | 0.001661 | 0.001804 | 0.002464 | 0.003683 | 0.004882 |
| **-17** | 0.000287 | 0.001892 | 0.002299 | 0.003276 | 0.004234 | 0.005407 | 0.006737 | 0.008596 |
| **-16** | 0.001088 | 0.002496 | 0.003435 | 0.005323 | 0.006516 | 0.00853 | 0.010976 | 0.013572 |
| **-15** | 0.001692 | 0.002988 | 0.004481 | 0.007165 | 0.00936 | 0.012323 | 0.015084 | 0.01839 |
| **-14** | 0.001528 | 0.004041 | 0.006212 | 0.008562 | 0.010902 | 0.01501 | 0.018481 | 0.024122 |
| **-13** | 0.001965 | 0.003558 | 0.006497 | 0.009032 | 0.012469 | 0.0168 | 0.021787 | 0.028498 |
| **-12** | 0.001617 | 0.003315 | 0.006527 | 0.009416 | 0.013049 | 0.017813 | 0.024024 | 0.032292 |
| **-11** | 0.001158 | 0.003755 | 0.0056 | 0.009196 | 0.012618 | 0.018012 | 0.025297 | 0.035005 |
| **-10** | 0.001764 | 0.002869 | 0.005074 | 0.008706 | 0.013025 | 0.018524 | 0.0257 | 0.0374 |
| **-9** | 0.001076 | 0.00299 | 0.005087 | 0.008134 | 0.011738 | 0.01792 | 0.026284 | 0.03838 |
| **-8** | 0.001208 | 0.002661 | 0.004807 | 0.007403 | 0.011998 | 0.017547 | 0.025949 | 0.038739 |
| **-7** | 0.000853 | 0.002414 | 0.004536 | 0.007226 | 0.011271 | 0.016511 | 0.024818 | 0.038655 |
| **-6** | 0.001134 | 0.002441 | 0.004521 | 0.00723 | 0.010596 | 0.016333 | 0.023925 | 0.037894 |
| **-5** | 0.000987 | 0.002539 | 0.004528 | 0.007236 | 0.010835 | 0.016246 | 0.02429 | 0.037894 |
| **-4** | 0.001223 | 0.002794 | 0.00472 | 0.007667 | 0.011179 | 0.016588 | 0.024843 | 0.038523 |
| **-3** | 0.001441 | 0.003097 | 0.005198 | 0.007775 | 0.012108 | 0.017853 | 0.026024 | 0.04062 |
| **-2** | 0.001536 | 0.003441 | 0.005663 | 0.009085 | 0.013327 | 0.019311 | 0.028329 | 0.043832 |
| **-1** | 0.001599 | 0.004063 | 0.006806 | 0.010367 | 0.015158 | 0.022279 | 0.031967 | 0.048481 |
| **0** | 0.002185 | 0.004636 | 0.007845 | 0.011739 | 0.017176 | 0.024543 | 0.035864 | 0.054098 |
| **1** | 0.002273 | 0.005031 | 0.008636 | 0.013121 | 0.019122 | 0.027205 | 0.039452 | 0.059659 |
| **2** | 0.002388 | 0.005299 | 0.009031 | 0.013921 | 0.020415 | 0.02949 | 0.04286 | 0.064303 |
| **3** | 0.002355 | 0.00546 | 0.009304 | 0.014289 | 0.021338 | 0.030913 | 0.045211 | 0.068386 |
| **4** | 0.002329 | 0.00525 | 0.009068 | 0.014139 | 0.021294 | 0.031586 | 0.046947 | 0.071558 |
| **5** | 0.002065 | 0.004754 | 0.00837 | 0.013264 | 0.020682 | 0.031273 | 0.047299 | 0.073353 |
| **6** | 0.001726 | 0.003898 | 0.007423 | 0.012497 | 0.019699 | 0.030033 | 0.046977 | 0.074607 |
| **7** | 0.001246 | 0.003222 | 0.006203 | 0.010911 | 0.018053 | 0.028859 | 0.045998 | 0.074562 |
| **8** | 0.000845 | 0.002462 | 0.005164 | 0.009414 | 0.016405 | 0.027465 | 0.045065 | 0.07448 |
| **9** | 0.000638 | 0.001885 | 0.00422 | 0.008184 | 0.015001 | 0.025704 | 0.043896 | 0.073915 |
| **10** | 0.000406 | 0.001421 | 0.00351 | 0.007171 | 0.013506 | 0.024411 | 0.042381 | 0.07302 |
| **11** | 0.000381 | 0.001111 | 0.002861 | 0.00624 | 0.012455 | 0.022909 | 0.041144 | 0.072325 |
| **12** | 0.000268 | 0.0009 | 0.002371 | 0.005605 | 0.01142 | 0.021763 | 0.040288 | 0.071223 |
| **13** | 0.000202 | 0.000755 | 0.002185 | 0.005031 | 0.010799 | 0.021045 | 0.039037 | 0.070416 |
| **14** | 0.0002 | 0.000633 | 0.001903 | 0.004654 | 0.010175 | 0.020389 | 0.038314 | 0.069856 |
| **15** | 0.000192 | 0.000618 | 0.001799 | 0.004424 | 0.009809 | 0.019804 | 0.037791 | 0.069479 |
| **16** | 0.000203 | 0.0006 | 0.001717 | 0.004257 | 0.009497 | 0.01964 | 0.037537 | 0.06861 |
| **17** | 0.00021 | 0.000541 | 0.001554 | 0.004098 | 0.009371 | 0.019431 | 0.037266 | 0.068957 |
| **18** | 0.000222 | 0.000555 | 0.001686 | 0.004024 | 0.00921 | 0.019366 | 0.037251 | 0.068739 |
| **19** | 0.000209 | 0.000575 | 0.00157 | 0.004024 | 0.009314 | 0.019402 | 0.037298 | 0.068823 |
| **20** | 0.000206 | 0.000622 | 0.001599 | 0.004044 | 0.009315 | 0.019383 | 0.037374 | 0.069655 |

**Table A.11:** Average Changes to BER for Neural Receiver Due to Dropout applied to 8th Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000501 | 0.000841 | 0.001554 | 0.001883 | 0.002751 | 0.002936 | 0.003075 | 0.003782 |
| **-19** | 0.000854 | 0.002275 | 0.002229 | 0.003293 | 0.004099 | 0.00482 | 0.005353 | 0.005873 |
| **-18** | 0.001396 | 0.002546 | 0.003828 | 0.004874 | 0.00602 | 0.006697 | 0.007635 | 0.00883 |
| **-17** | 0.001512 | 0.002895 | 0.004644 | 0.006192 | 0.007506 | 0.009045 | 0.010311 | 0.011439 |
| **-16** | 0.001757 | 0.003269 | 0.005029 | 0.006641 | 0.008741 | 0.010578 | 0.012305 | 0.014517 |
| **-15** | 0.001646 | 0.004057 | 0.004947 | 0.007909 | 0.009637 | 0.012231 | 0.014678 | 0.017035 |
| **-14** | 0.00209 | 0.004089 | 0.005705 | 0.007783 | 0.011103 | 0.01337 | 0.016665 | 0.01965 |
| **-13** | 0.001735 | 0.00458 | 0.006811 | 0.009466 | 0.012378 | 0.015801 | 0.018772 | 0.022856 |
| **-12** | 0.002071 | 0.004681 | 0.006955 | 0.010092 | 0.014271 | 0.017596 | 0.021766 | 0.02624 |
| **-11** | 0.002427 | 0.005276 | 0.007792 | 0.011649 | 0.016308 | 0.020623 | 0.0253 | 0.030537 |
| **-10** | 0.00293 | 0.006631 | 0.008914 | 0.013544 | 0.017483 | 0.022851 | 0.028151 | 0.035004 |
| **-9** | 0.003504 | 0.006862 | 0.00974 | 0.014277 | 0.019768 | 0.024488 | 0.030941 | 0.037893 |
| **-8** | 0.003055 | 0.006094 | 0.010799 | 0.015512 | 0.019905 | 0.025284 | 0.032596 | 0.040793 |
| **-7** | 0.003074 | 0.006257 | 0.010798 | 0.015043 | 0.020397 | 0.026667 | 0.033408 | 0.042378 |
| **-6** | 0.003077 | 0.006649 | 0.010324 | 0.014818 | 0.019627 | 0.026206 | 0.03346 | 0.0429 |
| **-5** | 0.003027 | 0.006398 | 0.009824 | 0.014953 | 0.019822 | 0.026601 | 0.033337 | 0.042988 |
| **-4** | 0.002801 | 0.006181 | 0.009678 | 0.014673 | 0.019622 | 0.025694 | 0.033275 | 0.043161 |
| **-3** | 0.002453 | 0.006104 | 0.009714 | 0.014626 | 0.019741 | 0.025771 | 0.033836 | 0.043637 |
| **-2** | 0.00304 | 0.006006 | 0.009872 | 0.014244 | 0.019707 | 0.02607 | 0.033706 | 0.04482 |
| **-1** | 0.00273 | 0.005639 | 0.00976 | 0.014071 | 0.019456 | 0.026278 | 0.034725 | 0.046064 |
| **0** | 0.002631 | 0.005738 | 0.009467 | 0.013769 | 0.019274 | 0.026542 | 0.035399 | 0.04815 |
| **1** | 0.002541 | 0.005355 | 0.009005 | 0.013578 | 0.019123 | 0.026655 | 0.036474 | 0.050142 |
| **2** | 0.002417 | 0.005244 | 0.008827 | 0.013479 | 0.019299 | 0.027007 | 0.037306 | 0.051616 |
| **3** | 0.002165 | 0.005051 | 0.008341 | 0.013195 | 0.018877 | 0.026692 | 0.037703 | 0.053048 |
| **4** | 0.002086 | 0.004528 | 0.008148 | 0.012375 | 0.018287 | 0.026701 | 0.037916 | 0.054183 |
| **5** | 0.001662 | 0.004057 | 0.007169 | 0.011638 | 0.017402 | 0.025726 | 0.037653 | 0.054863 |
| **6** | 0.001449 | 0.003514 | 0.006366 | 0.010426 | 0.016297 | 0.024737 | 0.036963 | 0.055318 |
| **7** | 0.001094 | 0.002858 | 0.005401 | 0.009451 | 0.015087 | 0.023579 | 0.036124 | 0.055482 |
| **8** | 0.00089 | 0.002207 | 0.004545 | 0.008265 | 0.013825 | 0.022662 | 0.03554 | 0.055984 |
| **9** | 0.000611 | 0.001877 | 0.0039 | 0.007392 | 0.012982 | 0.021589 | 0.035095 | 0.056421 |
| **10** | 0.00049 | 0.001607 | 0.003388 | 0.006588 | 0.012084 | 0.020932 | 0.034763 | 0.056358 |
| **11** | 0.000403 | 0.001274 | 0.002985 | 0.006051 | 0.011275 | 0.019922 | 0.034135 | 0.056457 |
| **12** | 0.000297 | 0.001115 | 0.002657 | 0.005627 | 0.010729 | 0.019444 | 0.033498 | 0.055969 |
| **13** | 0.000263 | 0.001031 | 0.002464 | 0.005367 | 0.010318 | 0.018775 | 0.032909 | 0.055861 |
| **14** | 0.00026 | 0.00095 | 0.002323 | 0.005022 | 0.01004 | 0.018356 | 0.032689 | 0.055812 |
| **15** | 0.000222 | 0.000888 | 0.002254 | 0.004928 | 0.009698 | 0.017988 | 0.032289 | 0.054892 |
| **16** | 0.000277 | 0.000857 | 0.002168 | 0.00484 | 0.009664 | 0.018032 | 0.032164 | 0.054804 |
| **17** | 0.000273 | 0.000865 | 0.002119 | 0.0047 | 0.009686 | 0.017716 | 0.031934 | 0.055238 |
| **18** | 0.000259 | 0.000884 | 0.002185 | 0.00492 | 0.009581 | 0.017984 | 0.032122 | 0.055353 |
| **19** | 0.00028 | 0.001009 | 0.002308 | 0.004998 | 0.009663 | 0.018299 | 0.03269 | 0.055447 |
| **20** | 0.000189 | 0.000935 | 0.002297 | 0.00514 | 0.009788 | 0.018435 | 0.032481 | 0.056251 |

**Table A.12:** Average Changes to BER for Neural Receiver Due to Dropout applied to Output Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.000367 | 0.000984 | 0.001374 | 0.001224 | 0.00222 | 0.002262 | 0.002646 | 0.002981 |
| **-19** | 0.000953 | 0.000955 | 0.002128 | 0.002204 | 0.00292 | 0.00363 | 0.004424 | 0.005577 |
| **-18** | 0.000774 | 0.002323 | 0.003283 | 0.004852 | 0.004676 | 0.006894 | 0.007371 | 0.008214 |
| **-17** | 0.001355 | 0.002913 | 0.004731 | 0.006641 | 0.008626 | 0.010035 | 0.01122 | 0.013893 |
| **-16** | 0.002039 | 0.004719 | 0.007109 | 0.00991 | 0.012389 | 0.014967 | 0.017972 | 0.020915 |
| **-15** | 0.003384 | 0.006576 | 0.010011 | 0.013948 | 0.017411 | 0.020907 | 0.025033 | 0.02934 |
| **-14** | 0.004575 | 0.008506 | 0.013176 | 0.018024 | 0.023481 | 0.028774 | 0.033411 | 0.038553 |
| **-13** | 0.00562 | 0.011377 | 0.017103 | 0.023625 | 0.029715 | 0.037011 | 0.043426 | 0.04973 |
| **-12** | 0.006563 | 0.013564 | 0.020949 | 0.028671 | 0.036916 | 0.045906 | 0.05435 | 0.063691 |
| **-11** | 0.007943 | 0.017234 | 0.026035 | 0.036058 | 0.046441 | 0.056452 | 0.067031 | 0.078032 |
| **-10** | 0.010675 | 0.020523 | 0.032399 | 0.043482 | 0.055723 | 0.068395 | 0.081217 | 0.094546 |
| **-9** | 0.012307 | 0.025227 | 0.038427 | 0.052627 | 0.066812 | 0.081881 | 0.096852 | 0.112391 |
| **-8** | 0.014351 | 0.02939 | 0.045812 | 0.062051 | 0.078686 | 0.096153 | 0.114232 | 0.131908 |
| **-7** | 0.016904 | 0.034689 | 0.053226 | 0.07219 | 0.092045 | 0.112098 | 0.131686 | 0.15267 |
| **-6** | 0.019581 | 0.040504 | 0.061491 | 0.083462 | 0.10546 | 0.128513 | 0.151822 | 0.174856 |
| **-5** | 0.023122 | 0.046269 | 0.070328 | 0.095078 | 0.120295 | 0.145911 | 0.171463 | 0.198064 |
| **-4** | 0.026001 | 0.052558 | 0.079341 | 0.107349 | 0.135507 | 0.164399 | 0.193277 | 0.222372 |
| **-3** | 0.029326 | 0.059192 | 0.089663 | 0.120288 | 0.151806 | 0.183374 | 0.215671 | 0.247959 |
| **-2** | 0.032678 | 0.066153 | 0.099849 | 0.134099 | 0.168545 | 0.203269 | 0.238528 | 0.273684 |
| **-1** | 0.036169 | 0.073013 | 0.11011 | 0.147354 | 0.184977 | 0.222751 | 0.261003 | 0.299143 |
| **0** | 0.039604 | 0.079357 | 0.119535 | 0.159786 | 0.200396 | 0.240996 | 0.281798 | 0.322546 |
| **1** | 0.042393 | 0.084855 | 0.127579 | 0.170709 | 0.213708 | 0.256821 | 0.300004 | 0.34334 |
| **2** | 0.044666 | 0.089627 | 0.13443 | 0.17953 | 0.224649 | 0.269908 | 0.315215 | 0.360364 |
| **3** | 0.046608 | 0.093074 | 0.139829 | 0.186463 | 0.233441 | 0.280293 | 0.327118 | 0.374031 |
| **4** | 0.04791 | 0.095781 | 0.143663 | 0.191555 | 0.239676 | 0.287736 | 0.33577 | 0.38403 |
| **5** | 0.048704 | 0.097629 | 0.146397 | 0.195098 | 0.244125 | 0.292785 | 0.341982 | 0.390865 |
| **6** | 0.049262 | 0.098617 | 0.147953 | 0.197461 | 0.246746 | 0.296181 | 0.345726 | 0.395088 |
| **7** | 0.049602 | 0.099327 | 0.148904 | 0.198648 | 0.248322 | 0.298088 | 0.347735 | 0.397522 |
| **8** | 0.04976 | 0.099593 | 0.149467 | 0.19931 | 0.24918 | 0.298981 | 0.348809 | 0.398687 |
| **9** | 0.04993 | 0.099701 | 0.149726 | 0.199632 | 0.249518 | 0.299458 | 0.349391 | 0.399322 |
| **10** | 0.049963 | 0.099886 | 0.149842 | 0.199808 | 0.249667 | 0.299735 | 0.349634 | 0.399582 |
| **11** | 0.050008 | 0.099879 | 0.149867 | 0.199835 | 0.24978 | 0.299799 | 0.349771 | 0.399683 |
| **12** | 0.04997 | 0.099906 | 0.149881 | 0.199862 | 0.249877 | 0.299802 | 0.349823 | 0.399813 |
| **13** | 0.049954 | 0.099923 | 0.149933 | 0.19992 | 0.249928 | 0.299818 | 0.349825 | 0.39979 |
| **14** | 0.049999 | 0.099918 | 0.149945 | 0.19987 | 0.249895 | 0.299896 | 0.349821 | 0.399805 |
| **15** | 0.04996 | 0.099939 | 0.149943 | 0.199922 | 0.249808 | 0.299862 | 0.349856 | 0.399779 |
| **16** | 0.049993 | 0.099987 | 0.149928 | 0.199873 | 0.249834 | 0.299811 | 0.349811 | 0.39982 |
| **17** | 0.049979 | 0.099918 | 0.149865 | 0.199935 | 0.249862 | 0.29985 | 0.349792 | 0.399724 |
| **18** | 0.049922 | 0.099926 | 0.149883 | 0.199938 | 0.249801 | 0.299779 | 0.349744 | 0.399758 |
| **19** | 0.049931 | 0.09995 | 0.149947 | 0.19984 | 0.249815 | 0.299777 | 0.349702 | 0.399735 |
| **20** | 0.049946 | 0.099838 | 0.149906 | 0.199779 | 0.249674 | 0.299778 | 0.349823 | 0.39964 |

## A.3 Monte Carlo Dropout: End to End Autoencoder

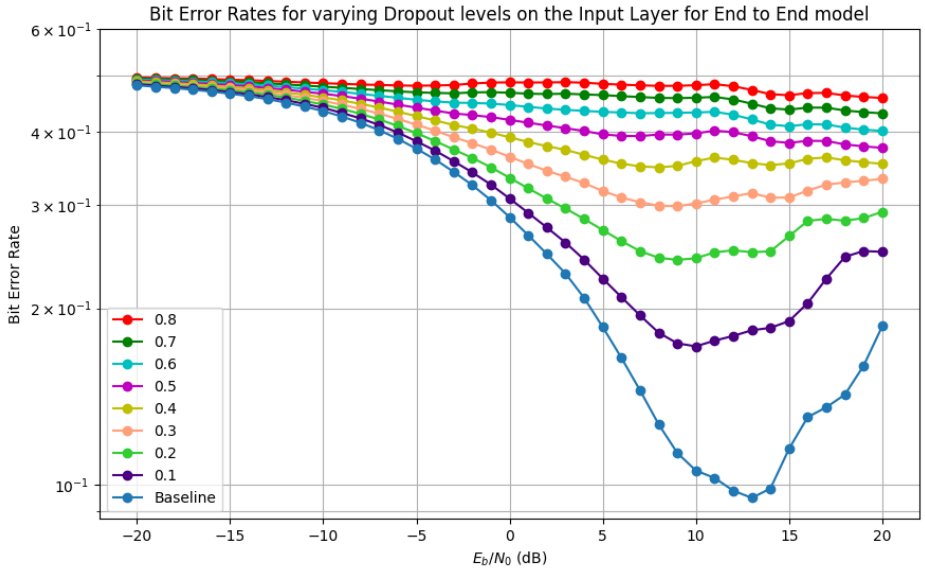



**Figure A.11:** Bit Error Rate curves for the End to End model with different dropout rates applied to its input layer.

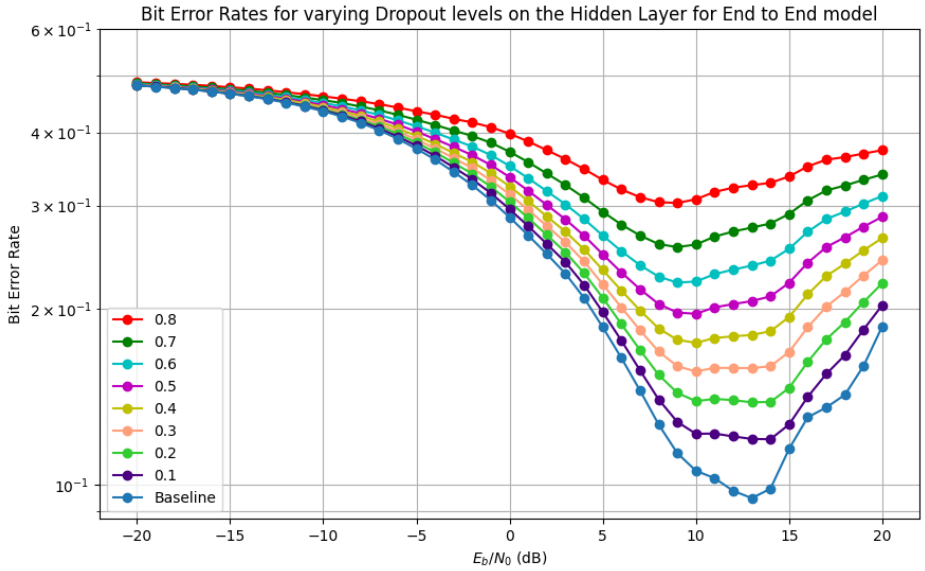



**Figure A.12:** Bit Error Rate curves for the End to End model with different dropout rates applied to its first hidden layer.

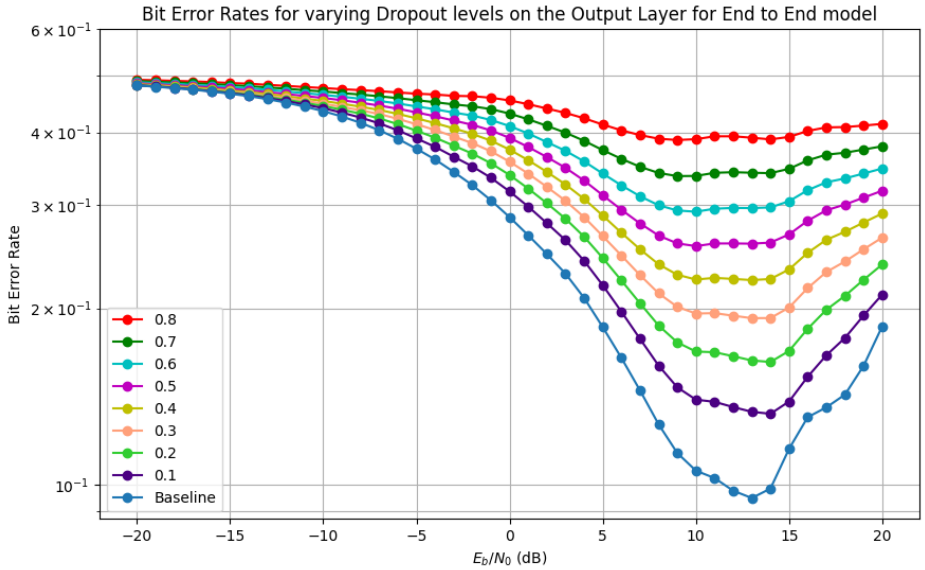



**Figure A.13:** Bit Error Rate curves for the End to End model with different dropout rates applied to its output layer.

**Table A.13:** Average Changes to BER for End to End Due to Dropout applied to Input Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.002042 | 0.003751 | 0.005479 | 0.007626 | 0.009453 | 0.011102 | 0.013043 | 0.014921 |
| **-19** | 0.002019 | 0.004287 | 0.006042 | 0.008269 | 0.010413 | 0.012534 | 0.014682 | 0.016815 |
| **-18** | 0.00208 | 0.004668 | 0.00671 | 0.009291 | 0.011675 | 0.013845 | 0.016531 | 0.018746 |
| **-17** | 0.002644 | 0.004857 | 0.007419 | 0.010093 | 0.0128 | 0.015657 | 0.018448 | 0.021203 |
| **-16** | 0.002761 | 0.005748 | 0.008599 | 0.011447 | 0.014295 | 0.017651 | 0.020686 | 0.023784 |
| **-15** | 0.00306 | 0.006361 | 0.009657 | 0.012524 | 0.01615 | 0.019523 | 0.023011 | 0.026243 |
| **-14** | 0.003291 | 0.006903 | 0.010556 | 0.01446 | 0.018283 | 0.021793 | 0.025869 | 0.029896 |
| **-13** | 0.003734 | 0.00799 | 0.012054 | 0.016174 | 0.020371 | 0.024998 | 0.029243 | 0.033755 |
| **-12** | 0.004208 | 0.008958 | 0.013385 | 0.018304 | 0.023437 | 0.028197 | 0.033357 | 0.038486 |
| **-11** | 0.004832 | 0.010024 | 0.0154 | 0.020987 | 0.026643 | 0.032381 | 0.038174 | 0.044048 |
| **-10** | 0.005555 | 0.011387 | 0.017708 | 0.024066 | 0.030772 | 0.037277 | 0.043853 | 0.050478 |
| **-9** | 0.006078 | 0.012968 | 0.020315 | 0.027892 | 0.035652 | 0.043274 | 0.050949 | 0.058435 |
| **-8** | 0.007246 | 0.014973 | 0.023304 | 0.032088 | 0.041361 | 0.050101 | 0.059033 | 0.067534 |
| **-7** | 0.008393 | 0.017319 | 0.027214 | 0.037653 | 0.048129 | 0.058405 | 0.068823 | 0.078674 |
| **-6** | 0.009691 | 0.020325 | 0.031742 | 0.043829 | 0.056069 | 0.068344 | 0.080039 | 0.09113 |
| **-5** | 0.011082 | 0.023671 | 0.037048 | 0.051313 | 0.065731 | 0.079565 | 0.093142 | 0.105465 |
| **-4** | 0.012742 | 0.027033 | 0.043172 | 0.059669 | 0.076213 | 0.092361 | 0.107669 | 0.121554 |
| **-3** | 0.014634 | 0.031452 | 0.049914 | 0.0694 | 0.08854 | 0.107187 | 0.12403 | 0.13947 |
| **-2** | 0.01668 | 0.036658 | 0.058354 | 0.080927 | 0.103133 | 0.124098 | 0.143061 | 0.159833 |
| **-1** | 0.019407 | 0.042443 | 0.067539 | 0.093394 | 0.118524 | 0.141818 | 0.162413 | 0.180473 |
| **0** | 0.021765 | 0.047914 | 0.076858 | 0.10577 | 0.133586 | 0.158864 | 0.180974 | 0.20043 |
| **1** | 0.024487 | 0.054086 | 0.086288 | 0.118317 | 0.148617 | 0.175351 | 0.198941 | 0.219923 |
| **2** | 0.027198 | 0.060239 | 0.09592 | 0.130929 | 0.162969 | 0.191435 | 0.217144 | 0.238918 |
| **3** | 0.029742 | 0.066924 | 0.106366 | 0.143951 | 0.177092 | 0.207544 | 0.236244 | 0.257835 |
| **4** | 0.033825 | 0.075943 | 0.119029 | 0.157525 | 0.193585 | 0.226796 | 0.255826 | 0.277847 |
| **5** | 0.039063 | 0.086353 | 0.131698 | 0.17341 | 0.210918 | 0.247614 | 0.277229 | 0.29818 |
| **6** | 0.044443 | 0.096079 | 0.144734 | 0.18895 | 0.229681 | 0.267821 | 0.296502 | 0.317825 |
| **7** | 0.049708 | 0.105661 | 0.158831 | 0.204931 | 0.249221 | 0.28627 | 0.314646 | 0.336332 |
| **8** | 0.055019 | 0.117392 | 0.172864 | 0.221645 | 0.269771 | 0.304834 | 0.331244 | 0.353507 |
| **9** | 0.06107 | 0.12882 | 0.185763 | 0.237214 | 0.283083 | 0.318513 | 0.344418 | 0.367029 |
| **10** | 0.0666 | 0.137997 | 0.196542 | 0.251375 | 0.292593 | 0.326836 | 0.352596 | 0.375834 |
| **11** | 0.073545 | 0.146712 | 0.204177 | 0.259556 | 0.300015 | 0.330972 | 0.357114 | 0.38078 |
| **12** | 0.081824 | 0.153751 | 0.213142 | 0.26131 | 0.302539 | 0.331054 | 0.357558 | 0.381951 |
| **13** | 0.088631 | 0.154596 | 0.220068 | 0.259404 | 0.298966 | 0.325883 | 0.352753 | 0.37767 |
| **14** | 0.086803 | 0.151923 | 0.21071 | 0.253045 | 0.287723 | 0.313701 | 0.341068 | 0.366312 |
| **15** | 0.075131 | 0.150691 | 0.194321 | 0.238788 | 0.268479 | 0.294572 | 0.322235 | 0.347678 |
| **16** | 0.07384 | 0.151791 | 0.187615 | 0.229994 | 0.256403 | 0.282429 | 0.310376 | 0.336407 |
| **17** | 0.089265 | 0.149 | 0.189952 | 0.226829 | 0.250983 | 0.277038 | 0.305232 | 0.331588 |
| **18** | 0.102525 | 0.139858 | 0.18584 | 0.215767 | 0.238832 | 0.264879 | 0.293267 | 0.31982 |
| **19** | 0.091056 | 0.125739 | 0.17088 | 0.195863 | 0.218839 | 0.244691 | 0.273414 | 0.300136 |
| **20** | 0.063368 | 0.105976 | 0.146488 | 0.166959 | 0.189733 | 0.215577 | 0.244457 | 0.271375 |

**Table A.14:** Average Changes to BER for End to End Due to Dropout applied to Hidden Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 5.68173E-05 | 0.000177 | 0.000511 | 0.000921 | 0.001347 | 0.002637 | 0.004027 | 0.006038 |
| **-19** | 0.000253218 | 0.000431 | 0.000795 | 0.000959 | 0.001788 | 0.003036 | 0.004543 | 0.006941 |
| **-18** | 0.000426501 | 0.000585 | 0.000893 | 0.001634 | 0.002367 | 0.003572 | 0.00555 | 0.008089 |
| **-17** | 0.000251698 | 0.000506 | 0.001278 | 0.0017 | 0.002832 | 0.004212 | 0.006136 | 0.00918 |
| **-16** | 0.000183716 | 0.000761 | 0.001167 | 0.002046 | 0.003296 | 0.004831 | 0.007196 | 0.010757 |
| **-15** | 0.000119427 | 0.000943 | 0.001448 | 0.002508 | 0.004056 | 0.005933 | 0.008451 | 0.012017 |
| **-14** | 0.000355979 | 0.001047 | 0.001903 | 0.003133 | 0.004615 | 0.006927 | 0.009807 | 0.014022 |
| **-13** | 0.000655989 | 0.001514 | 0.002264 | 0.003561 | 0.005462 | 0.008356 | 0.011631 | 0.016205 |
| **-12** | 0.000794705 | 0.001835 | 0.003057 | 0.004841 | 0.006832 | 0.009612 | 0.013537 | 0.018849 |
| **-11** | 0.001226482 | 0.002389 | 0.003875 | 0.005861 | 0.008558 | 0.011755 | 0.016182 | 0.022141 |
| **-10** | 0.00134381 | 0.003074 | 0.004885 | 0.007216 | 0.010414 | 0.014336 | 0.019177 | 0.026086 |
| **-9** | 0.001967234 | 0.003792 | 0.006409 | 0.009003 | 0.012666 | 0.01729 | 0.023207 | 0.030888 |
| **-8** | 0.002381102 | 0.004895 | 0.007875 | 0.011531 | 0.015661 | 0.020807 | 0.027631 | 0.036881 |
| **-7** | 0.002855737 | 0.006138 | 0.009717 | 0.013836 | 0.018589 | 0.025042 | 0.032854 | 0.043618 |
| **-6** | 0.003487146 | 0.00757 | 0.011514 | 0.016552 | 0.0223 | 0.029456 | 0.038671 | 0.05107 |
| **-5** | 0.004595691 | 0.009048 | 0.013752 | 0.019468 | 0.026252 | 0.034707 | 0.045295 | 0.059535 |
| **-4** | 0.005174179 | 0.010535 | 0.016222 | 0.022701 | 0.030188 | 0.040116 | 0.0524 | 0.069294 |
| **-3** | 0.006264857 | 0.012403 | 0.019111 | 0.026393 | 0.035479 | 0.046489 | 0.060573 | 0.079451 |
| **-2** | 0.007587489 | 0.014862 | 0.022537 | 0.030943 | 0.040647 | 0.05332 | 0.069399 | 0.091039 |
| **-1** | 0.008647982 | 0.016746 | 0.025017 | 0.034439 | 0.045618 | 0.059597 | 0.077849 | 0.102035 |
| **0** | 0.009196503 | 0.01799 | 0.026943 | 0.036998 | 0.048954 | 0.064075 | 0.084188 | 0.111503 |
| **1** | 0.009782181 | 0.018767 | 0.028095 | 0.038515 | 0.051092 | 0.067167 | 0.088934 | 0.119121 |
| **2** | 0.0100243 | 0.01929 | 0.028902 | 0.039493 | 0.052745 | 0.069503 | 0.092324 | 0.125153 |
| **3** | 0.010584148 | 0.020139 | 0.030164 | 0.041203 | 0.054584 | 0.071929 | 0.095695 | 0.130591 |
| **4** | 0.010994513 | 0.021419 | 0.032194 | 0.043883 | 0.05785 | 0.075869 | 0.100777 | 0.137718 |
| **5** | 0.011357134 | 0.022768 | 0.034118 | 0.046885 | 0.061724 | 0.080694 | 0.10682 | 0.145895 |
| **6** | 0.011653066 | 0.023482 | 0.036177 | 0.049974 | 0.065823 | 0.085974 | 0.113543 | 0.154801 |
| **7** | 0.011948675 | 0.024983 | 0.038746 | 0.053588 | 0.070918 | 0.09232 | 0.121701 | 0.165099 |
| **8** | 0.013099334 | 0.027347 | 0.042102 | 0.058228 | 0.076825 | 0.099996 | 0.130954 | 0.177173 |
| **9** | 0.014792971 | 0.030297 | 0.046189 | 0.063576 | 0.0836 | 0.108251 | 0.141429 | 0.189716 |
| **10** | 0.016552969 | 0.033294 | 0.050493 | 0.069099 | 0.090282 | 0.116741 | 0.151601 | 0.201753 |
| **11** | 0.019666902 | 0.037547 | 0.055738 | 0.07546 | 0.098275 | 0.126338 | 0.162763 | 0.213767 |
| **12** | 0.023299183 | 0.041846 | 0.060672 | 0.081654 | 0.105876 | 0.135443 | 0.173154 | 0.224137 |
| **13** | 0.024851593 | 0.043417 | 0.063317 | 0.08537 | 0.111065 | 0.141896 | 0.180109 | 0.23005 |
| **14** | 0.021155385 | 0.040043 | 0.060922 | 0.084466 | 0.111359 | 0.143002 | 0.180967 | 0.22943 |
| **15** | 0.011616079 | 0.031167 | 0.053448 | 0.078271 | 0.106082 | 0.137795 | 0.175028 | 0.221102 |
| **16** | 0.010894142 | 0.031729 | 0.05539 | 0.081054 | 0.109025 | 0.140058 | 0.175706 | 0.219427 |
| **17** | 0.019117035 | 0.041704 | 0.065988 | 0.091793 | 0.119177 | 0.148944 | 0.18272 | 0.223676 |
| **18** | 0.024020925 | 0.047149 | 0.071072 | 0.09608 | 0.122223 | 0.1503 | 0.181999 | 0.220384 |
| **19** | 0.024529549 | 0.045608 | 0.068106 | 0.091814 | 0.116341 | 0.142614 | 0.172238 | 0.208325 |
| **20** | 0.016189926 | 0.034382 | 0.055425 | 0.077472 | 0.100421 | 0.124863 | 0.15246 | 0.18638 |

**Table A.15:** Average Changes to BER for End to End Due to Dropout applied to Output Layer

| | **Dropout(%)** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **10** | **20** | **30** | **40** | **50** | **60** | **70** | **80** |
| **-20** | 0.00022 | 0.000766 | 0.001301 | 0.002355 | 0.003991 | 0.006098 | 0.00818 | 0.010659 |
| **-19** | 0.000299 | 0.000914 | 0.001634 | 0.003077 | 0.004651 | 0.006805 | 0.009369 | 0.012297 |
| **-18** | 0.000656 | 0.001032 | 0.001944 | 0.003463 | 0.005496 | 0.008047 | 0.01088 | 0.013817 |
| **-17** | 0.000527 | 0.00136 | 0.002561 | 0.00425 | 0.006443 | 0.00898 | 0.012248 | 0.015671 |
| **-16** | 0.000637 | 0.001848 | 0.003099 | 0.005037 | 0.007684 | 0.010755 | 0.014185 | 0.01787 |
| **-15** | 0.001114 | 0.002249 | 0.003922 | 0.005948 | 0.009177 | 0.012458 | 0.016126 | 0.019998 |
| **-14** | 0.001395 | 0.002973 | 0.004946 | 0.00741 | 0.010692 | 0.01442 | 0.018462 | 0.023181 |
| **-13** | 0.001884 | 0.003826 | 0.006122 | 0.009258 | 0.012748 | 0.017118 | 0.021516 | 0.026427 |
| **-12** | 0.002677 | 0.005111 | 0.007793 | 0.011491 | 0.015457 | 0.02012 | 0.025393 | 0.030636 |
| **-11** | 0.003758 | 0.006642 | 0.010219 | 0.014101 | 0.018862 | 0.024122 | 0.029442 | 0.035472 |
| **-10** | 0.005037 | 0.00891 | 0.012946 | 0.017727 | 0.023046 | 0.028705 | 0.03498 | 0.041382 |
| **-9** | 0.006567 | 0.011543 | 0.016425 | 0.022027 | 0.028172 | 0.034503 | 0.041103 | 0.048376 |
| **-8** | 0.008681 | 0.015226 | 0.021037 | 0.02717 | 0.034158 | 0.041284 | 0.048858 | 0.056997 |
| **-7** | 0.011031 | 0.019126 | 0.026474 | 0.033436 | 0.041069 | 0.049192 | 0.057638 | 0.066949 |
| **-6** | 0.013233 | 0.023166 | 0.031899 | 0.040182 | 0.049025 | 0.058145 | 0.067636 | 0.077989 |
| **-5** | 0.015507 | 0.02756 | 0.037854 | 0.047671 | 0.057658 | 0.067919 | 0.07886 | 0.090703 |
| **-4** | 0.018383 | 0.032224 | 0.044238 | 0.055299 | 0.066672 | 0.078548 | 0.090875 | 0.104714 |
| **-3** | 0.021066 | 0.037276 | 0.051075 | 0.064007 | 0.076706 | 0.089982 | 0.104133 | 0.119421 |
| **-2** | 0.024915 | 0.042947 | 0.058449 | 0.072578 | 0.087044 | 0.102429 | 0.118299 | 0.13619 |
| **-1** | 0.028324 | 0.048011 | 0.065086 | 0.080956 | 0.096826 | 0.113768 | 0.132133 | 0.152186 |
| **0** | 0.030493 | 0.051695 | 0.069897 | 0.087261 | 0.104907 | 0.123542 | 0.144178 | 0.167051 |
| **1** | 0.031874 | 0.053804 | 0.073212 | 0.091624 | 0.110773 | 0.13134 | 0.154243 | 0.180051 |
| **2** | 0.03198 | 0.054828 | 0.074893 | 0.094498 | 0.115056 | 0.137135 | 0.162554 | 0.19149 |
| **3** | 0.03223 | 0.055283 | 0.076058 | 0.096374 | 0.118444 | 0.142499 | 0.170049 | 0.20234 |
| **4** | 0.032754 | 0.056758 | 0.077985 | 0.09931 | 0.122313 | 0.148234 | 0.17835 | 0.214286 |
| **5** | 0.033284 | 0.058072 | 0.080039 | 0.102424 | 0.126447 | 0.154227 | 0.187095 | 0.226521 |
| **6** | 0.032845 | 0.05863 | 0.08168 | 0.104701 | 0.13024 | 0.159968 | 0.195667 | 0.238518 |
| **7** | 0.032589 | 0.059244 | 0.083166 | 0.107576 | 0.13461 | 0.166121 | 0.204379 | 0.250775 |
| **8** | 0.032944 | 0.060253 | 0.085511 | 0.111184 | 0.139731 | 0.17344 | 0.214105 | 0.263335 |
| **9** | 0.033386 | 0.061772 | 0.087991 | 0.11513 | 0.145064 | 0.180846 | 0.223467 | 0.274816 |
| **10** | 0.034055 | 0.063152 | 0.090414 | 0.118477 | 0.149999 | 0.187006 | 0.23125 | 0.283719 |
| **11** | 0.035883 | 0.065751 | 0.093731 | 0.122944 | 0.155627 | 0.193494 | 0.238449 | 0.291132 |
| **12** | 0.038049 | 0.068092 | 0.096739 | 0.127002 | 0.160672 | 0.199416 | 0.24426 | 0.29614 |
| **13** | 0.0383 | 0.068122 | 0.097629 | 0.128769 | 0.163208 | 0.201998 | 0.246294 | 0.296707 |
| **14** | 0.033683 | 0.063558 | 0.094136 | 0.126111 | 0.160713 | 0.19917 | 0.242158 | 0.290593 |
| **15** | 0.02355 | 0.05408 | 0.085582 | 0.11807 | 0.152514 | 0.189874 | 0.231328 | 0.277675 |
| **16** | 0.022504 | 0.05424 | 0.086318 | 0.118586 | 0.152076 | 0.188144 | 0.227771 | 0.271772 |
| **17** | 0.030828 | 0.063462 | 0.095327 | 0.126544 | 0.158799 | 0.1931 | 0.230809 | 0.272299 |
| **18** | 0.035618 | 0.067719 | 0.098112 | 0.12793 | 0.158525 | 0.191086 | 0.226601 | 0.265839 |
| **19** | 0.034891 | 0.064257 | 0.09282 | 0.120872 | 0.14972 | 0.180503 | 0.214164 | 0.251078 |
| **20** | 0.024817 | 0.0513 | 0.077838 | 0.10411 | 0.131366 | 0.160424 | 0.192101 | 0.22704 |

## A.4 FGSM: Neural Receiver

**Table A.16:** Average Changes to BER for Neural Receiver Due to FGSM

| | **Epsilon Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
| **-20** | 0.007404 | 0.013964 | 0.020218 | 0.025155 | 0.031074 | 0.034822 | 0.039623 | 0.043796 |
| **-19** | 0.007806 | 0.01482 | 0.020796 | 0.026328 | 0.031649 | 0.037071 | 0.041061 | 0.044732 |
| **-18** | 0.008119 | 0.015283 | 0.021563 | 0.027402 | 0.033069 | 0.037618 | 0.040697 | 0.046319 |
| **-17** | 0.008477 | 0.015596 | 0.022956 | 0.02891 | 0.03416 | 0.040568 | 0.044143 | 0.048651 |
| **-16** | 0.009112 | 0.017267 | 0.023544 | 0.030721 | 0.036747 | 0.040166 | 0.0457 | 0.050507 |
| **-15** | 0.009738 | 0.018574 | 0.026858 | 0.034207 | 0.040164 | 0.045957 | 0.051476 | 0.053515 |
| **-14** | 0.01041 | 0.020651 | 0.029507 | 0.037571 | 0.044295 | 0.05101 | 0.055023 | 0.061917 |
| **-13** | 0.012314 | 0.023126 | 0.032739 | 0.041988 | 0.049845 | 0.057378 | 0.062855 | 0.069931 |
| **-12** | 0.013667 | 0.025817 | 0.037809 | 0.048492 | 0.055509 | 0.065964 | 0.072723 | 0.078144 |
| **-11** | 0.01581 | 0.028991 | 0.042027 | 0.05339 | 0.064188 | 0.07245 | 0.082766 | 0.090727 |
| **-10** | 0.017993 | 0.034146 | 0.049511 | 0.060275 | 0.071263 | 0.081527 | 0.092636 | 0.099495 |
| **-9** | 0.019625 | 0.036945 | 0.054214 | 0.067668 | 0.081677 | 0.088915 | 0.102088 | 0.110309 |
| **-8** | 0.021235 | 0.039765 | 0.058762 | 0.072374 | 0.086719 | 0.099584 | 0.112493 | 0.120626 |
| **-7** | 0.02316 | 0.043599 | 0.061679 | 0.078096 | 0.094589 | 0.107637 | 0.119512 | 0.128049 |
| **-6** | 0.025527 | 0.046495 | 0.066205 | 0.082176 | 0.100646 | 0.112564 | 0.125567 | 0.139495 |
| **-5** | 0.027416 | 0.050251 | 0.069154 | 0.08506 | 0.101876 | 0.119439 | 0.132023 | 0.145342 |
| **-4** | 0.029051 | 0.052396 | 0.069856 | 0.087879 | 0.104119 | 0.121451 | 0.137728 | 0.153479 |
| **-3** | 0.030091 | 0.052135 | 0.070606 | 0.089924 | 0.106348 | 0.124215 | 0.144667 | 0.157735 |
| **-2** | 0.029684 | 0.050444 | 0.068275 | 0.087288 | 0.107586 | 0.12707 | 0.146626 | 0.16394 |
| **-1** | 0.028523 | 0.046129 | 0.064439 | 0.084479 | 0.1067 | 0.129923 | 0.150953 | 0.171853 |
| **0** | 0.025311 | 0.041654 | 0.062731 | 0.084269 | 0.107476 | 0.128926 | 0.15137 | 0.173394 |
| **1** | 0.021406 | 0.037288 | 0.059159 | 0.08387 | 0.108011 | 0.135529 | 0.15725 | 0.179146 |
| **2** | 0.017818 | 0.033243 | 0.054657 | 0.082337 | 0.110552 | 0.136233 | 0.163336 | 0.186295 |
| **3** | 0.014015 | 0.029212 | 0.051801 | 0.078744 | 0.107553 | 0.134198 | 0.164579 | 0.186767 |
| **4** | 0.010921 | 0.026017 | 0.049418 | 0.077765 | 0.109615 | 0.138936 | 0.161809 | 0.187969 |
| **5** | 0.008317 | 0.021569 | 0.044616 | 0.075269 | 0.102553 | 0.135742 | 0.163126 | 0.190566 |
| **6** | 0.005771 | 0.018342 | 0.0408 | 0.068533 | 0.102877 | 0.136299 | 0.160325 | 0.19388 |
| **7** | 0.004177 | 0.014433 | 0.036147 | 0.060742 | 0.096703 | 0.127606 | 0.159314 | 0.19266 |
| **8** | 0.002979 | 0.010832 | 0.029266 | 0.053442 | 0.092528 | 0.124664 | 0.155317 | 0.188085 |
| **9** | 0.002057 | 0.009564 | 0.026367 | 0.05326 | 0.0903 | 0.120626 | 0.152825 | 0.184385 |
| **10** | 0.001488 | 0.006481 | 0.023474 | 0.045564 | 0.083961 | 0.110336 | 0.151418 | 0.187962 |
| **11** | 0.000941 | 0.007509 | 0.021886 | 0.043271 | 0.086317 | 0.125787 | 0.143298 | 0.182778 |
| **12** | 0.000936 | 0.007252 | 0.021728 | 0.046771 | 0.080853 | 0.11634 | 0.161331 | 0.183782 |
| **13** | 0.000787 | 0.005115 | 0.018473 | 0.049708 | 0.089011 | 0.127316 | 0.155887 | 0.187613 |
| **14** | 0.000659 | 0.004224 | 0.017692 | 0.053335 | 0.087535 | 0.131043 | 0.159089 | 0.195407 |
| **15** | 0.001894 | 0.005885 | 0.016064 | 0.049674 | 0.089544 | 0.127826 | 0.158456 | 0.194144 |
| **16** | 0.000994 | 0.005924 | 0.017319 | 0.052741 | 0.093004 | 0.124267 | 0.162388 | 0.194823 |
| **17** | 0.000646 | 0.007063 | 0.017062 | 0.057871 | 0.094064 | 0.135043 | 0.174812 | 0.196086 |
| **18** | 0.000808 | 0.007174 | 0.018475 | 0.055208 | 0.095193 | 0.134295 | 0.164623 | 0.212213 |
| **19** | 0.001983 | 0.008128 | 0.0229 | 0.051961 | 0.104369 | 0.13991 | 0.17576 | 0.202306 |
| **20** | 0.001504 | 0.007138 | 0.020164 | 0.056139 | 0.101972 | 0.145966 | 0.1773 | 0.206317 |

**Table A.17:** Average Changes to Prediction Confidence for Neural Receiver Due to FGSM

| | **Epsilon Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
| **-20** | 0.035736 | 0.072011 | 0.106386 | 0.141716 | 0.177313 | 0.210074 | 0.247809 | 0.276894 |
| **-19** | 0.03802 | 0.074488 | 0.111233 | 0.148243 | 0.183448 | 0.222304 | 0.259234 | 0.295524 |
| **-18** | 0.039712 | 0.079972 | 0.120903 | 0.159778 | 0.198647 | 0.237279 | 0.275022 | 0.308247 |
| **-17** | 0.043349 | 0.086869 | 0.128947 | 0.17239 | 0.21343 | 0.254359 | 0.298052 | 0.339843 |
| **-16** | 0.047642 | 0.094113 | 0.143398 | 0.188581 | 0.232785 | 0.278233 | 0.324292 | 0.367178 |
| **-15** | 0.053115 | 0.104951 | 0.156992 | 0.207938 | 0.256741 | 0.305735 | 0.356209 | 0.410984 |
| **-14** | 0.058992 | 0.117509 | 0.174105 | 0.235046 | 0.288803 | 0.340852 | 0.390467 | 0.445601 |
| **-13** | 0.06649 | 0.133596 | 0.196873 | 0.26066 | 0.319719 | 0.379368 | 0.440524 | 0.50714 |
| **-12** | 0.075773 | 0.148621 | 0.222174 | 0.292408 | 0.360945 | 0.43134 | 0.495973 | 0.557604 |
| **-11** | 0.084773 | 0.171208 | 0.253555 | 0.32995 | 0.40274 | 0.485147 | 0.550944 | 0.615524 |
| **-10** | 0.097919 | 0.190564 | 0.279057 | 0.374279 | 0.454576 | 0.538449 | 0.612556 | 0.68073 |
| **-9** | 0.110266 | 0.216146 | 0.31856 | 0.417336 | 0.516205 | 0.600974 | 0.687228 | 0.765046 |
| **-8** | 0.123069 | 0.239384 | 0.357335 | 0.464213 | 0.567145 | 0.659174 | 0.764882 | 0.836312 |
| **-7** | 0.139218 | 0.270145 | 0.39569 | 0.51849 | 0.629305 | 0.739422 | 0.837664 | 0.922695 |
| **-6** | 0.15535 | 0.305492 | 0.449475 | 0.577796 | 0.697789 | 0.819275 | 0.929996 | 1.029176 |
| **-5** | 0.176685 | 0.339259 | 0.50071 | 0.649529 | 0.789729 | 0.917246 | 1.034832 | 1.145692 |
| **-4** | 0.204702 | 0.397192 | 0.583452 | 0.736413 | 0.904273 | 1.037733 | 1.194063 | 1.316287 |
| **-3** | 0.239581 | 0.457461 | 0.685852 | 0.881315 | 1.044091 | 1.227943 | 1.382753 | 1.523983 |
| **-2** | 0.285661 | 0.562265 | 0.820269 | 1.057344 | 1.251352 | 1.439907 | 1.631269 | 1.852167 |
| **-1** | 0.355523 | 0.685212 | 0.995111 | 1.292429 | 1.508166 | 1.761576 | 1.969902 | 2.137168 |
| **0** | 0.434998 | 0.839712 | 1.239445 | 1.54554 | 1.88781 | 2.148131 | 2.424528 | 2.60056 |
| **1** | 0.525434 | 1.0006 | 1.472058 | 1.86456 | 2.22795 | 2.522243 | 2.830817 | 3.067641 |
| **2** | 0.615438 | 1.166883 | 1.69236 | 2.173108 | 2.570143 | 2.986324 | 3.282964 | 3.576533 |
| **3** | 0.707858 | 1.394212 | 1.991076 | 2.4972 | 2.943142 | 3.372906 | 3.782082 | 4.088865 |
| **4** | 0.839199 | 1.596943 | 2.248179 | 2.847302 | 3.378114 | 3.865302 | 4.233888 | 4.620678 |
| **5** | 0.958439 | 1.798015 | 2.579132 | 3.307736 | 3.863135 | 4.321296 | 4.776269 | 5.212969 |
| **6** | 1.089223 | 2.045058 | 2.881142 | 3.594339 | 4.261584 | 4.891966 | 5.326192 | 5.929914 |
| **7** | 1.216685 | 2.262254 | 3.204224 | 3.992812 | 4.749484 | 5.441424 | 5.956753 | 6.503576 |
| **8** | 1.315465 | 2.468162 | 3.485371 | 4.423963 | 5.164505 | 5.939209 | 6.681333 | 7.219769 |
| **9** | 1.328178 | 2.605996 | 3.660298 | 4.687263 | 5.64431 | 6.359511 | 7.164647 | 7.760785 |
| **10** | 1.356884 | 2.610681 | 3.850864 | 4.952113 | 5.942072 | 6.857104 | 7.760212 | 8.189072 |
| **11** | 1.380227 | 2.662348 | 3.883268 | 5.091587 | 6.136686 | 7.024518 | 7.87096 | 8.684822 |
| **12** | 1.362854 | 2.702855 | 3.939573 | 5.215706 | 6.211768 | 7.282264 | 8.09088 | 8.912025 |
| **13** | 1.390454 | 2.69587 | 3.928696 | 5.192624 | 6.320302 | 7.361215 | 8.334358 | 9.081624 |
| **14** | 1.423659 | 2.747424 | 3.974759 | 5.150024 | 6.254113 | 7.335784 | 8.414747 | 9.170728 |
| **15** | 1.402889 | 2.728841 | 3.976065 | 5.167489 | 6.301169 | 7.460137 | 8.395815 | 9.174944 |
| **16** | 1.426602 | 2.754762 | 4.019296 | 5.20769 | 6.237952 | 7.41856 | 8.369213 | 9.220868 |
| **17** | 1.43403 | 2.784503 | 3.962819 | 5.193556 | 6.363837 | 7.412468 | 8.357891 | 9.232536 |
| **18** | 1.430716 | 2.74177 | 4.101672 | 5.208396 | 6.273557 | 7.385919 | 8.292755 | 9.311301 |
| **19** | 1.433082 | 2.782577 | 4.019177 | 5.258334 | 6.338202 | 7.335022 | 8.296944 | 9.127494 |
| **20** | 1.42382 | 2.804508 | 4.012204 | 5.232384 | 6.301556 | 7.365746 | 8.227826 | 9.061571 |

## A.5 FGSM: End to End Autoencoder

**Table A.18:** Average Changes to BER for End to End Due to FGSM

| | **Epsilon Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
| **-20** | 0.009272 | 0.018754 | 0.02244 | 0.024954 | 0.032786 | 0.039347 | 0.041111 | 0.044055 |
| **-19** | 0.010418 | 0.018038 | 0.025863 | 0.029934 | 0.035767 | 0.04219 | 0.047922 | 0.052711 |
| **-18** | 0.006356 | 0.01771 | 0.029129 | 0.036697 | 0.045058 | 0.051315 | 0.059696 | 0.065294 |
| **-17** | 0.010686 | 0.018184 | 0.028338 | 0.036032 | 0.042767 | 0.048776 | 0.055738 | 0.060251 |
| **-16** | 0.013391 | 0.018599 | 0.024984 | 0.030562 | 0.036672 | 0.042701 | 0.046929 | 0.05013 |
| **-15** | 0.021603 | 0.0325 | 0.041047 | 0.048556 | 0.059209 | 0.067089 | 0.071947 | 0.074603 |
| **-14** | 0.018975 | 0.030465 | 0.040273 | 0.051144 | 0.060837 | 0.067755 | 0.072391 | 0.073724 |
| **-13** | 0.017372 | 0.029396 | 0.041292 | 0.057493 | 0.070511 | 0.077967 | 0.083863 | 0.087475 |
| **-12** | 0.016035 | 0.031125 | 0.045129 | 0.064891 | 0.085623 | 0.096097 | 0.104183 | 0.108404 |
| **-11** | 0.017202 | 0.030491 | 0.045668 | 0.06404 | 0.087221 | 0.104343 | 0.112586 | 0.117258 |
| **-10** | 0.019126 | 0.03 | 0.04719 | 0.065769 | 0.084043 | 0.104897 | 0.11321 | 0.118658 |
| **-9** | 0.021903 | 0.032097 | 0.049766 | 0.071852 | 0.088897 | 0.111842 | 0.121422 | 0.127289 |
| **-8** | 0.024896 | 0.035301 | 0.051692 | 0.074274 | 0.093224 | 0.117702 | 0.129117 | 0.133991 |
| **-7** | 0.027914 | 0.038519 | 0.054593 | 0.076578 | 0.096672 | 0.122337 | 0.136635 | 0.141162 |
| **-6** | 0.030456 | 0.041742 | 0.056156 | 0.076205 | 0.096999 | 0.121846 | 0.139991 | 0.145407 |
| **-5** | 0.033459 | 0.044696 | 0.057642 | 0.075625 | 0.096382 | 0.120113 | 0.139135 | 0.147946 |
| **-4** | 0.03527 | 0.048068 | 0.059493 | 0.075085 | 0.094877 | 0.116979 | 0.13636 | 0.149523 |
| **-3** | 0.035922 | 0.053199 | 0.064495 | 0.078174 | 0.09606 | 0.116593 | 0.136105 | 0.151487 |
| **-2** | 0.035646 | 0.055332 | 0.066938 | 0.078808 | 0.094771 | 0.113735 | 0.132532 | 0.148978 |
| **-1** | 0.034704 | 0.056038 | 0.068355 | 0.07929 | 0.092529 | 0.108384 | 0.125715 | 0.142114 |
| **0** | 0.033386 | 0.053697 | 0.066986 | 0.078868 | 0.091389 | 0.104735 | 0.119142 | 0.133351 |
| **1** | 0.032606 | 0.052551 | 0.065339 | 0.076278 | 0.087783 | 0.099834 | 0.111929 | 0.123522 |
| **2** | 0.030589 | 0.0487 | 0.060056 | 0.069638 | 0.079376 | 0.089087 | 0.09834 | 0.107686 |
| **3** | 0.028633 | 0.045434 | 0.056199 | 0.065156 | 0.073102 | 0.080333 | 0.087135 | 0.093491 |
| **4** | 0.026833 | 0.042525 | 0.052657 | 0.060702 | 0.067327 | 0.072168 | 0.077059 | 0.081336 |
| **5** | 0.025767 | 0.041198 | 0.050807 | 0.058162 | 0.063803 | 0.068014 | 0.07167 | 0.074807 |
| **6** | 0.024323 | 0.039047 | 0.047951 | 0.054278 | 0.058972 | 0.06253 | 0.065431 | 0.067477 |
| **7** | 0.023063 | 0.036912 | 0.044912 | 0.050106 | 0.053593 | 0.056029 | 0.058014 | 0.059163 |
| **8** | 0.021568 | 0.034322 | 0.041561 | 0.045729 | 0.048269 | 0.049673 | 0.050391 | 0.050498 |
| **9** | 0.019875 | 0.03137 | 0.037859 | 0.041325 | 0.04301 | 0.043766 | 0.043633 | 0.04334 |
| **10** | 0.018199 | 0.028554 | 0.034135 | 0.037072 | 0.038293 | 0.038381 | 0.037972 | 0.037342 |
| **11** | 0.016594 | 0.025837 | 0.030895 | 0.03352 | 0.034492 | 0.03455 | 0.033864 | 0.032881 |
| **12** | 0.015095 | 0.023402 | 0.028024 | 0.030595 | 0.031623 | 0.031518 | 0.03097 | 0.029939 |
| **13** | 0.013739 | 0.021225 | 0.025494 | 0.02778 | 0.028858 | 0.028836 | 0.028201 | 0.027219 |
| **14** | 0.012417 | 0.01915 | 0.0231 | 0.025179 | 0.025988 | 0.026217 | 0.025565 | 0.024885 |
| **15** | 0.011263 | 0.017217 | 0.020611 | 0.02244 | 0.023233 | 0.023298 | 0.023003 | 0.022386 |
| **16** | 0.010078 | 0.015358 | 0.018288 | 0.01988 | 0.020497 | 0.020662 | 0.020442 | 0.01982 |
| **17** | 0.008865 | 0.013431 | 0.015959 | 0.017247 | 0.017716 | 0.01784 | 0.01765 | 0.017211 |
| **18** | 0.007727 | 0.011613 | 0.013712 | 0.01488 | 0.015344 | 0.015233 | 0.015097 | 0.014672 |
| **19** | 0.006744 | 0.010042 | 0.011819 | 0.012758 | 0.013106 | 0.013086 | 0.012855 | 0.012594 |
| **20** | 0.005856 | 0.008695 | 0.010108 | 0.010909 | 0.011216 | 0.011177 | 0.011057 | 0.010754 |

**Table A.19:** Average Changes to Prediction Confidence for End to End Due to FGSM

| | **Epsilon Value** | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $E_b/N_0$ **(dB)** | **0.1** | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** |
| **-20** | 0.278014 | 0.552132 | 0.823202 | 1.09119 | 1.359076 | 1.629352 | 1.904544 | 2.191594 |
| **-19** | 0.259024 | 0.518275 | 0.782205 | 1.05504 | 1.328221 | 1.607104 | 1.886792 | 2.174139 |
| **-18** | 0.265748 | 0.532298 | 0.797769 | 1.065023 | 1.334387 | 1.602045 | 1.881416 | 2.161383 |
| **-17** | 0.291155 | 0.588378 | 0.89064 | 1.194644 | 1.500643 | 1.812308 | 2.126005 | 2.434671 |
| **-16** | 0.275629 | 0.556082 | 0.836967 | 1.1203 | 1.402483 | 1.684859 | 1.956823 | 2.231605 |
| **-15** | 0.35956 | 0.719956 | 1.078089 | 1.429254 | 1.783101 | 2.120091 | 2.439649 | 2.750283 |
| **-14** | 0.334437 | 0.667073 | 0.99778 | 1.32074 | 1.633371 | 1.938634 | 2.240217 | 2.532307 |
| **-13** | 0.342829 | 0.679257 | 1.00855 | 1.335253 | 1.657786 | 1.966058 | 2.271829 | 2.575555 |
| **-12** | 0.387526 | 0.761536 | 1.128485 | 1.489706 | 1.845982 | 2.196768 | 2.535479 | 2.861016 |
| **-11** | 0.407173 | 0.802275 | 1.187496 | 1.562494 | 1.933683 | 2.299412 | 2.65697 | 2.9948 |
| **-10** | 0.39353 | 0.779732 | 1.150166 | 1.511695 | 1.868793 | 2.221699 | 2.568713 | 2.90431 |
| **-9** | 0.403355 | 0.799652 | 1.180754 | 1.550949 | 1.912401 | 2.271069 | 2.628984 | 2.971005 |
| **-8** | 0.409441 | 0.804519 | 1.189897 | 1.561223 | 1.927258 | 2.29201 | 2.651719 | 3.007113 |
| **-7** | 0.404378 | 0.796778 | 1.178804 | 1.54445 | 1.904639 | 2.267729 | 2.630229 | 2.982761 |
| **-6** | 0.381999 | 0.752756 | 1.114689 | 1.466083 | 1.813315 | 2.157399 | 2.498856 | 2.836578 |
| **-5** | 0.351098 | 0.693241 | 1.027122 | 1.358527 | 1.688851 | 2.019633 | 2.341287 | 2.663656 |
| **-4** | 0.321299 | 0.633726 | 0.944667 | 1.259295 | 1.574734 | 1.892338 | 2.198359 | 2.505417 |
| **-3** | 0.305673 | 0.606536 | 0.908208 | 1.213848 | 1.524781 | 1.834121 | 2.141447 | 2.450773 |
| **-2** | 0.289048 | 0.571554 | 0.857095 | 1.146432 | 1.437675 | 1.734156 | 2.032475 | 2.335512 |
| **-1** | 0.266952 | 0.527233 | 0.788594 | 1.053558 | 1.325151 | 1.599172 | 1.879349 | 2.167425 |
| **0** | 0.246894 | 0.485599 | 0.722734 | 0.963917 | 1.210126 | 1.463692 | 1.723319 | 1.99106 |
| **1** | 0.229567 | 0.450987 | 0.66921 | 0.888355 | 1.113343 | 1.347176 | 1.587361 | 1.838649 |
| **2** | 0.20947 | 0.411175 | 0.609123 | 0.807462 | 1.009653 | 1.224548 | 1.443904 | 1.675178 |
| **3** | 0.189423 | 0.370726 | 0.546597 | 0.722268 | 0.904448 | 1.093604 | 1.290129 | 1.49811 |
| **4** | 0.173229 | 0.338392 | 0.49886 | 0.657187 | 0.821885 | 0.9927 | 1.170553 | 1.360423 |
| **5** | 0.163835 | 0.320865 | 0.472791 | 0.622285 | 0.774084 | 0.933159 | 1.099647 | 1.273937 |
| **6** | 0.15641 | 0.307009 | 0.452389 | 0.595938 | 0.740943 | 0.891556 | 1.049532 | 1.214438 |
| **7** | 0.148374 | 0.292074 | 0.431715 | 0.569475 | 0.708297 | 0.85162 | 1.002334 | 1.158359 |
| **8** | 0.140835 | 0.277907 | 0.41125 | 0.544311 | 0.676756 | 0.812585 | 0.954541 | 1.098504 |
| **9** | 0.132586 | 0.26136 | 0.387632 | 0.513386 | 0.637856 | 0.766372 | 0.896992 | 1.032609 |
| **10** | 0.123155 | 0.243275 | 0.360745 | 0.476257 | 0.592688 | 0.711319 | 0.831162 | 0.956339 |
| **11** | 0.114179 | 0.225077 | 0.333459 | 0.441684 | 0.549444 | 0.658379 | 0.769075 | 0.881419 |
| **12** | 0.106132 | 0.208874 | 0.309307 | 0.409065 | 0.508372 | 0.609696 | 0.712434 | 0.815178 |
| **13** | 0.098044 | 0.192895 | 0.285641 | 0.377385 | 0.468882 | 0.562512 | 0.653901 | 0.751403 |
| **14** | 0.090482 | 0.178219 | 0.263961 | 0.348198 | 0.432253 | 0.518429 | 0.602218 | 0.691077 |
| **15** | 0.083028 | 0.164131 | 0.242715 | 0.320127 | 0.397621 | 0.476282 | 0.556367 | 0.635802 |
| **16** | 0.07589 | 0.149909 | 0.222392 | 0.293883 | 0.365099 | 0.436291 | 0.508844 | 0.582795 |
| **17** | 0.068429 | 0.135651 | 0.201365 | 0.265363 | 0.3314 | 0.395125 | 0.461018 | 0.527118 |
| **18** | 0.061373 | 0.121504 | 0.180206 | 0.238449 | 0.296672 | 0.353599 | 0.413012 | 0.472878 |
| **19** | 0.054342 | 0.10763 | 0.160059 | 0.212605 | 0.263155 | 0.314435 | 0.366081 | 0.416983 |
| **20** | 0.047867 | 0.095113 | 0.141086 | 0.186951 | 0.231908 | 0.277852 | 0.322767 | 0.367866 |