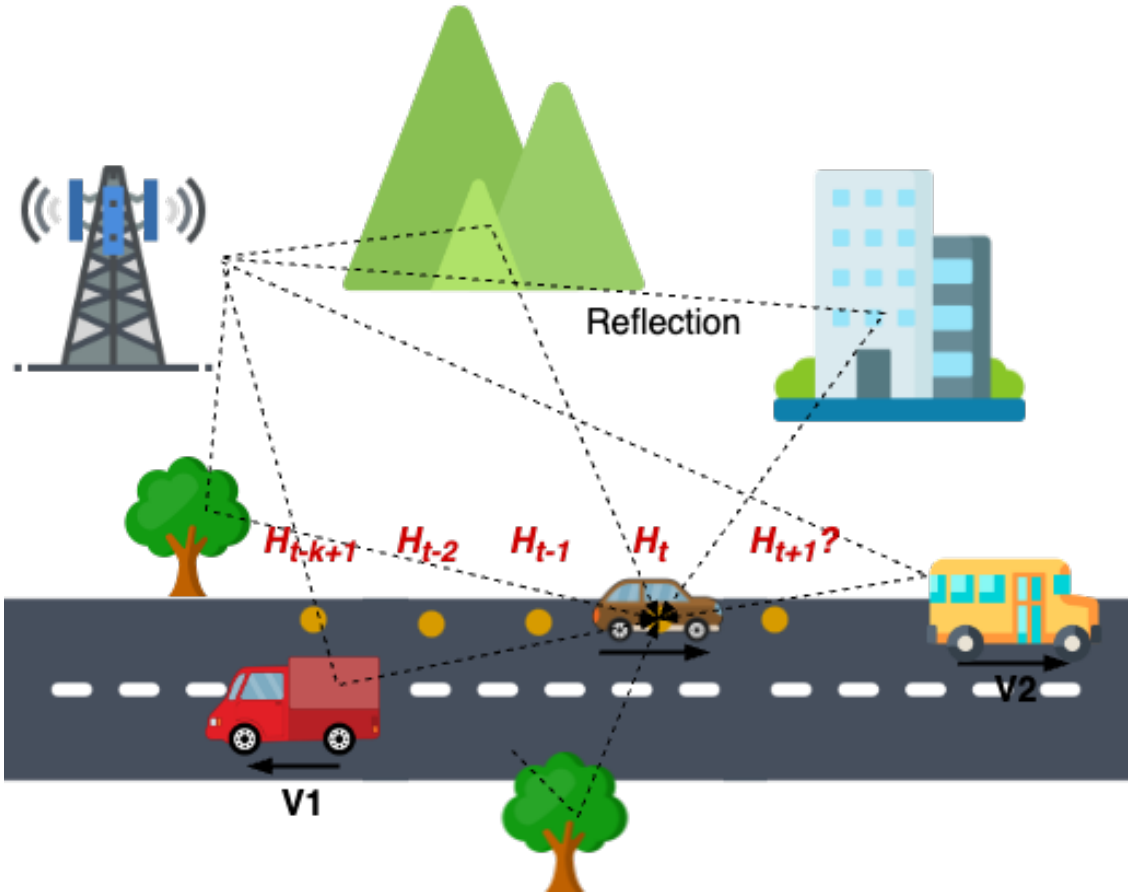


AI-Based Wireless Channel Prediction

Generalized Models for Adaptive Measurements and Predictions

Master's thesis in Master Program Information and Communication Technology

BINGCHENG CHEN

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

Master's thesis 2024

AI-Based Wireless Channel Prediction

Generalized Models for Adaptive Measurements and Predictions

BINGCHENG CHEN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communication, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2024

AI-Based Wireless Channel Prediction
Generalized Models for Adaptive Measurements and Predictions
BINGCHENG CHEN

Supervisor (Arranged Alphabetically by First Initial):
Johan Winges, Ericsson
Keerthi Kumar Nagalapur, Ericsson (∗Principal Supervisor)
Mehdi Sattari, Department of Electrical Engineering, Chalmers
Xinlin Zhang, Ericsson

Examiner:
Tommy Svensson, Department of Electrical Engineering, Chalmers

Master's Thesis 2024
Department of Electrical Engineering
Division of Communication, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A typical channel prediction scenario in which a cellular tower communicates with vehicles on a road, with the signals encountering reflections and scattering from various objects (Inspired by [1]).

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2024

AI-Based Wireless Channel Prediction
Generalized Models for Adaptive Measurements and Predictions
BINGCHENG CHEN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In wireless communication systems, channels can be time-varying, and their conditions can change due to factors such as mobility, interference, and environmental conditions, which brings challenges in maintaining communication reliability. Hence, acquiring channel state information (CSI) is a critical step in physical layer of wireless communication. However, when accurate CSI or precoder fed back by a User Equipment (UE) is used by a network for MIMO precoding, the received CSI/precoder can get outdated quickly, consequently resulting in a loss of user and system throughput due to the utilization of outdated precoders. In a feature introduced in 3GPP Rel-18, a UE can be configured to measure multiple instances of the channel, use them to predict a number of channel/precoder instances in the future and report the predictions to the network. The performance of such a scheme depends highly on the ability to predict the channel with high accuracy.

In this study, we explored AI-based methods for channel prediction to address this challenge. By utilizing a past window size of measured channel sequences, the proposed method forecasts future channel conditions. Out of all the AI models tested, the Transformer Encoder-Only model demonstrated superior performance, surpassing its counterparts and a classical non-AI based autoregressive (AR) model. We also found that, training the model with a diverse dataset with a mix of UE velocities, embedding across the time dimension within the Transformer Encoder-Only model, and constructing the model within the complex domain yields enhanced generalization capability. Furthermore, we developed a generalized model capable of using varying number of channel measurements to predict varying number of channel instances in the future.

Keywords: Artificial intelligence, Channel state information (CSI), CSI prediction, MIMO.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my superiors at Ericsson Research: Keerthi Kumar Nagalapur, Johan Winges, and Xinlin Zhang, for your guidance and support throughout my thesis. I am truly grateful for your assistance in overcoming hardware limitations, such as insufficient computational resources, RAM, and disk storage, by providing me with the upgraded devices. Your guidance has been invaluable, especially during times when I found myself unsure of the direction to take in my thesis. Our weekly meetings and discussions have been a constant source of inspiration, and I am incredibly appreciative of the time you have invested in my work.

I would also like to extend my sincere thanks to my line manager, Henrik Sahlin, for providing me with this opportunity and for being readily available to help whenever I encountered problems.

Furthermore, I am grateful to my superior Mehdi Sattari and examiner Tommy Svensson at Chalmers for your invaluable advice, information, and assistance in shaping this thesis throughout the process.

I also want to express my gratitude to OpenAI and the team behind it. You have revolutionized my study strategies and workflow, seamlessly integrating AI into my daily life, including the coding and thesis aspects of this project. Your innovation have encouraged me to approach my work with greater creativity and enthusiasm.

Last but certainly not least, I want to express my warmest gratitude to my family. You have been my unwavering shield and support throughout my entire life.

Bingcheng Chen, Gothenburg, June 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

3GPP	Third Generation Partnership Project
5G	5th Generation Mobile Network
AR	Autoregressive
BERT	Bidirectional Encoder Representations from Transformers
BS	Base Station
CNNs	Convolutional Neural Networks
CQI	Channel Quality Indicator
CSI	Channel State Information
CSI-RS	Channel State Information Reference Signal
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
FCN	Fully Connected Neural Network
FNN	Feedforward Neural Network
GPT	Generative Pre-trained Transformer
ICI	Inter-Channel Interference
LMS	Least Mean Squares
LSTM	Long Short-Term Memory
MCS	Modulation and Coding Schemes
MIMO	Multiple-Input Multiple-Output
MSE	Mean Squared Error
NMSE	Normalized Mean Squared Error
NR	New Radio
PMI	Precoding Matrix Indicator
PRB	Physical Resource Block
RI	Rank Indicator
RNNs	Recurrent Neural Networks
RRC	Radio Resource Control
SNR	Signal-to-Noise Ratio
STEM GNN	Spectral-Temporal Graph Neural Network
SU-MIMO	Single User Multiple-Input Multiple-Output
UE	User Equipment
UMa	Urban Macro
VKF	Vector Kalman Filter

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

Mobile networks has transformed many aspects of our life, including education, healthcare, online commerce, and even our intimate relationships. Each new generation of mobile networks has significantly advanced and improved various industries and personal interactions. The latest generation, 5G, is set to reshape the way businesses operate and how individuals interact with technology. According to the 2023 Ericsson Mobility Report [2], during the third quarter of 2023, 163 million 5G subscriptions were added, bringing the total to 1.4 billion. Global 5G subscriptions are forecast to exceed 5.3 billion by 2029, accounting for 58% of all mobile subscriptions at that time. Additionally, it is anticipated that by the end of 2029, the average monthly mobile data usage per smartphone will increase to 56 GB.

The growing demand for data usage has driven the need for more efficient and reliable wireless communication systems. This demand has led to the development of technologies such as massive multiple-input multiple-output (MIMO) systems. MIMO systems utilize a large number of antennas to improve spectral efficiency and increase data rates, making it as a key technology in 5G networks.

The performance of massive MIMO systems heavily relies on the availability of accurate channel state information (CSI). CSI [3] refers to the known properties of a communication channel that reveals how a signal propagates from the transmitter to the receiver. These properties include the channel's attenuation, phase shift, delay and so on. Having accurate CSI would enable the transmitter to adapt its signal processing techniques, such as precoding and beamforming, to optimize the connectivity.

However, obtaining accurate CSI is challenging, particularly in rapidly changing wireless environments, due to factors such as user mobility, interference from other devices or systems, or various environmental conditions. In 5G NR systems, both the user equipment (UE) and the base station (BS) are involved in the estimation of the CSI through a feedback mechanism. In this process, the UE first measures the downlink channel and then reports the estimated CSI back to the BS. The problem is the feedback introduces delay, so the reported CSI may be outdated when used for precoding, especially for UE with high mobility. Moreover, as the number of antennas and users increases, the overhead associated with channel estimation becomes

significant. This leads to a fundamental trade-off between the resources allocated for pilot transmission and data transmission.

In the latest 5G specification (3GPP Release-18 [4]), a new channel state information (CSI) feedback, designed for the case of user equipment (UE) mobility, has been introduced to improve both user and system throughput [5]. In the Release-18 CSI Type-II codebook, a set of N_4 precoders are predicted by a UE for a future time window based on periodically spaced channel measurements and fed back to the network using a codebook consisting of spatial, frequency and doppler domain bases. The method to be used by a UE for computing the feedback is not specified and is up to implementation.

1.2 Aim

The overall aim of this MSc thesis project is to investigate the feasibility and the computational complexity of determining channel predictions necessary for the Release-18 Type-II codebook feedback using AI/ML techniques.

1.3 Scope and Limitations

This project will exclusively explore methods based on deep neural networks, excluding other previously known approaches. The investigation will focus on the following three inquiries:

- How to setup and train a neural network for computing the channel predictions necessary for Release-18 Type-II feedback.
- Compare the performance with the classical model-based channel prediction methods.
- Optimize the trained neural network for enhancing the model's generalization ability.

1.4 Related work

Channel prediction is a well-studied area with a wide range of research papers. Various methods are proposed including the Yule-Walker equations [6], Burg's algorithm [7, 8], Least Mean Squares(LMS) [9], and the Kalman filter [10, 11, 12], based on autoregressive (AR) models. In an AR framework, a complex channel (H_t) at time (t) can be modeled as a weighted sum of its past values, allowing for the prediction of future channel states by leveraging historical data.

In recent years, researchers have expanded their approaches to incorporate deep neural networks, enhancing the accuracy of channel prediction models. [13] compares a vector Kalman filter (VKF) and a Fully Connected NN (FCN)-based channel prediction for massive MIMO systems, highlighting gains in AI-based prediction accuracy.

[14] has investigated Feedforward neural network and Recurrent neural network for channel prediction. [15] proposed 3D Complex Convolutional Neural Networks to effectively predict Channel State Information (CSI) in massive MIMO systems. A recent paper [16] used the Spectral-Temporal Graph Neural Network (STEM GNN) for 5G CSI prediction, which outperforms traditional models by leveraging spatial-temporal dynamics for enhanced system performance.

1.5 Structure of the thesis

This thesis is organized into five chapters, starting with an introduction that sets the context, objectives, and scope of the research, while also reviewing related work.

Chapter 2 provides a theoretical foundation, discussing the essentials of MIMO systems, wireless channel characterization, and the role of Channel State Information (CSI) in 5G networks. It further delves into the application of machine learning in wireless communication, reviewing various neural network architectures such as CNNs, RNNs, and the Transformer architecture.

Chapter 3 outlines the methods involved. It describes data preprocessing techniques, model development, and evaluation strategies, including metrics used to assess model performance.

Chapter 4 presents a detailed analysis of the research findings, including visualizations of channel characteristics and evaluations of model performance under different conditions. It discusses the impact of various factors on model accuracy and summarizes the key findings derived from the analysis.

Chapter 5 summarizes the contributions and suggests directions for future work.

2

Theory

This chapter covers key concepts and techniques in 4 sections, each focusing on a crucial aspect that underpins the thesis. We first introduce MIMO systems, wireless channels, and the CSI mechanism in 5G networks; these concepts are essential for understanding the context of the thesis, after that, we explore machine learning algorithms, including CNNs, RNNs, and transformers; these algorithms have demonstrated remarkable performance in a wide range of deep learning applications and are implemented in this thesis to tackle the specific challenges associated with CSI prediction.

2.1 Fundamentals of MIMO Systems

2.1.1 An Introduction to MIMO System

In the domain of wireless communications, Multiple-Input Multiple-Output (MIMO) technology has revolutionized the way data is transmitted and received over the air interface. By using multiple antennas at both the transmitter and receiver, MIMO systems allow for the simultaneous transmission and reception of multiple data streams through a technique called spatial multiplexing. This involves sending different data streams in parallel over the same frequency channel but through separate spatial paths. The capacity of a MIMO system can be mathematically expressed as follows:

$$C_{\mathrm{MIMO}} = N_{\mathrm{TX}} N_{\mathrm{RX}} \mathrm{B} log_2(1 + S/N) \tag{2.1}$$

Where C_{MIMO} represents the channel capacity of the MIMO sytem, B denotes the bandwidth, S/N is the signal-to-noise ratio, N_{TX} and N_{RX} represent the number of transmit and receive antennas, respectively. This equation highlights that the capacity of a MIMO system can be linearly increased by utilizing the product of $N_{\mathrm{TX}} N_{\mathrm{RX}}$, which corresponds to the number of transmit and receive antennas.

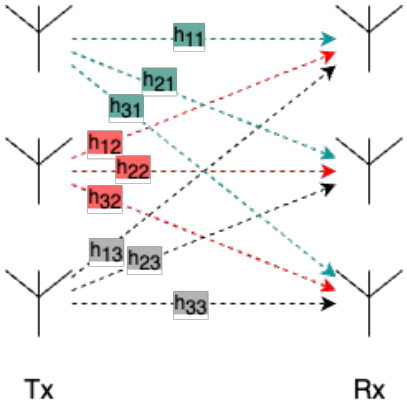


Figure 2.1: MIMO Communication System

As illustrated in Figure 2.1, In MIMO systems, a transmitter uses multiple antennas to send multiple data streams simultaneously. These data streams travel through a wireless channel, which can be represented as a matrix consisting of all the possible paths between the transmit and receive antennas. The number of paths is determined by the product of the number of transmit antennas (N_{TX}) and the number of receive antennas (N_{TX}).

$$y_t = H_t S_t + n \tag{2.2}$$

where $y_t \in \mathbb{C}^{N_{\mathrm{RX}} \times 1}$ and $S_t \in \mathbb{C}^{N_{\mathrm{TX}} \times 1}$ are the receive and transmit vectors at time t, respectively, and $H_t = \begin{bmatrix} h_{11} & h_{12} & h_{13} & \dots \\ h_{21} & h_{22} & h_{23} & \dots \\ h_{31} & h_{32} & h_{33} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}_{N_{\mathrm{RX}} \times N_{\mathrm{TX}}}$ and n are the channel matrix and the noise vector at time t, respectively.

2.1.2 Overview of Array Antenna Configurations

Figure 2.2 illustrates a two-dimensional antenna array with a configuration of $N_2 = 2$ rows and $N_1 = 8$ columns. The array is organized in a rectangular grid with uniform spacing between elements, where d_h and d_v represent the horizontal and vertical distances between adjacent antenna elements, respectively.

In this configuration, the red lines represent one polarization of the antenna elements, while the green lines represent the orthogonal polarization. By controlling the phase and amplitude of the signals sent to each element, the array can steer its main radiation beam in a desired direction. This beam steering ability is a key feature of array antennas, allowing them to adapt to changing communication needs and improve system performance.

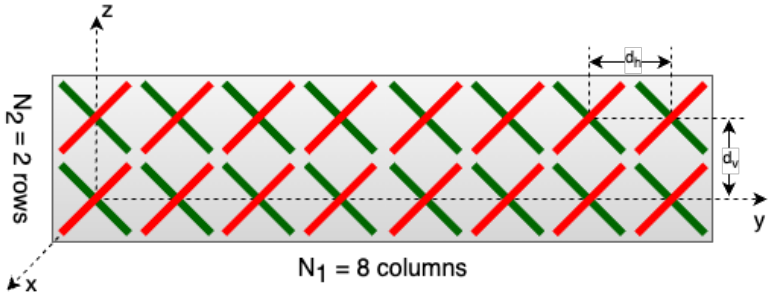


Figure 2.2: An example of Antenna Array

2.1.3 The Necessity of Precoding

However, in MIMO systems, the dimensions of the channel matrix grow with the number of transmit and receive antennas. Directly feeding back this raw channel matrix would require transmitting a significant amount of information over the feedback channel. Let's consider a specific example to illustrate the feedback overhead. Suppose the channel for frequency unit j at time t is given by $H_j(t) \in \mathbb{C}^{N_{\mathrm{RX}} \times N_{\mathrm{TX}}}$. Assume that the feedback consists of real numbers, with each real number represented using 8 bits. In this example, let's consider a scenario where $N_{\mathrm{RX}} = 4$, $N_{\mathrm{TX}} = 32$, number of frequency-units is $J = 52$. If only one time unit $T = 1$ is transmitted, the size of the feedback can be calculated as follows:

$$\begin{aligned} F_{\mathrm{OH}} &= N_{\mathrm{RX}} \times N_{\mathrm{TX}} \times J \times T \times 2 \\ &= 4 \times 32 \times 52 \times 1 \times 2 \\ &= 13312 \text{ (real values)} \\ &= 106496 \; bits \end{aligned} \tag{2.3}$$

This overhead consume a significant portion of the available bandwidth, leaving less resources for actual data transmission.

To address the issue of overhead, precoding techniques are utilized. Rather than transmitting the raw channel matrix directly, the User Equipment (UE) sends the index of a codebook entry that most accurately represents the channel's condition. The codebook consists of a pre-determined set of matrices designed to appropriately weight and combine the channel matrix prior to transmission. By selecting the most suitable codebook entry, the UE can effectively compress the channel information in a CSI report of a few hundred bits while still capturing its essential characteristics. This process significantly reduces the amount of feedback information that needs to be sent over the limited feedback channel. However, the development of the codebook above is not included within the scope of this thesis.

2.1.4 MIMO Precoding Transmitter Perspective and Challenges

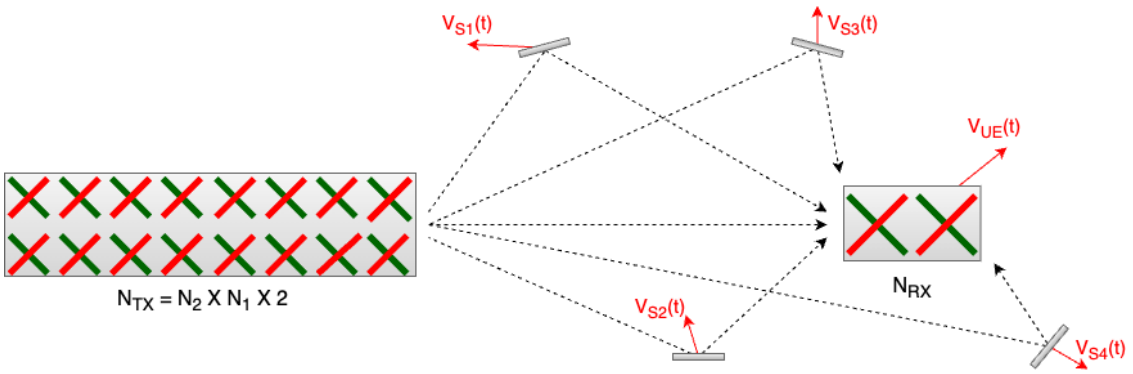


Figure 2.3: Time-varying MIMO channel

As illustrated in Figure 2.3, for a time-varying Multiple-Input Multiple-Output (MIMO) channel, with N_{TX} transmit ports and N_{RX} receive ports, the channel at time t for frequency unit/Physical Resource Block (PRB) j is denoted as $H_j(t) \in \mathbb{C}^{N_{\mathrm{RX}} \times N_{\mathrm{TX}}}$. This representation captures the complex-valued channel coefficients between transmit and receive antenna, which vary over time and frequency due to the dynamic nature of the wireless environment.

Assuming a precoder for PRB j that maps v MIMO spatial layers to N_{TX} transmit antennas, represented as $P_j \in \mathbb{C}^{N_{\mathrm{TX}} \times v}$, the received vector in a system model is given by:

$$y_j(t) = H_j(t)P_j(t)x + n \tag{2.4}$$

where $x \in \mathbb{C}^{v \times 1}$ is the transmitted vector and n is the noise-plus-interference vector.

A fundamental issue in feedback-based precoding is that,

- A UE measures a channel $H_j(t)$ at time t and determines a precoder $P_j(t)$.
- However due to a delay in feedback of $P_j(t)$ and the typically periodic and infrequent feedback, there is a mismatch if the channel changes during this delay, resulting in the following equation:

$$y_j(t+T) = H_j(t+T)P_j(t)x + n \tag{2.5}$$

- This mismatch leads to a performance degradation due to using an outdated precoder that does not accurately reflect the current channel conditions.

The main challenge now is finding a good and efficient way to predict the channel, denoted as $\hat{H}_j(t_p)$, where $t_p = t + T$, based on some past channel information, as

depicted in Figure 2.4. This prediction task is crucial for mitigating the performance loss caused by the feedback delay and ensuring that the precoder remains effective in the presence of time-varying channels.

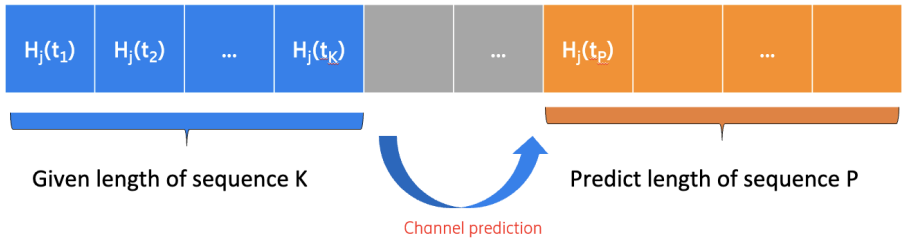
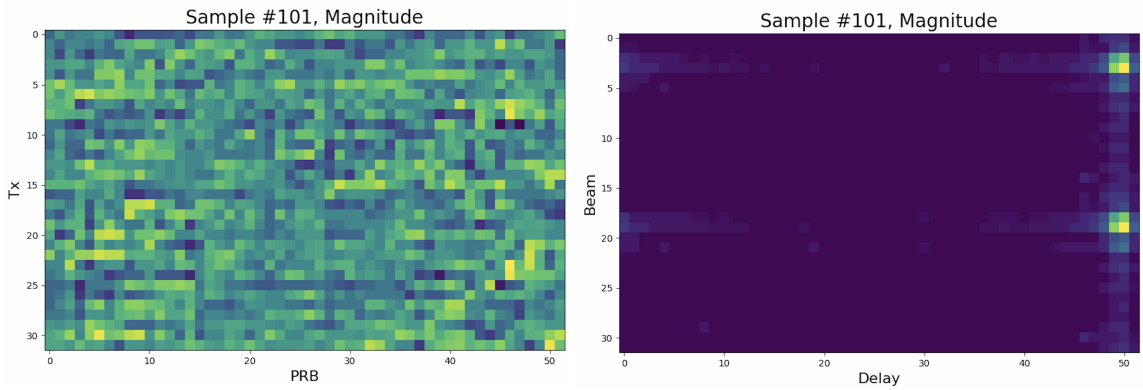


Figure 2.4: Channel prediction

2.2 Wireless Channel Characterization

2.2.1 Antenna Frequency Domain & Beam Delay Domain

The wireless channel can be analyzed from different perspectives, particularly in the antenna frequency domain and beam delay domain, Figure 2.5 illustrates a comparison of the channel representation in these two domains, highlighting their distinct characteristics and the information they provide about the wireless channel.



a) Antenna Frequency Domain b) Beam Delay Domain

Figure 2.5: Comparison of Channel Representation in Antenna Frequency Domain and Beam Delay Domain

In the antenna frequency domain, as shown in Figure 2.5(a), the vertical axis represents the transmit antennas, while the horizontal axis corresponds to the frequency domain indexed by Physical Resource Block (PRB). The magnitude of the channel matrix is depicted using a color scale, with brighter colors indicating stronger channel gains. In this domain, the channel is characterized by the complex-valued frequency response of each antenna element, which captures the amplitude and phase variations of the signal across the frequency spectrum.

Conversely, the beam delay domain, depicted in Figure 2.5(b), offers a distinct view by illustrating how the channel behaves with respect to different propagation paths, each characterized by a certain time delay and angle of arrival. The delay represents the time it takes for the signal to travel along a particular path, while the beam component represents the direction from which the signal arrives. Notably, the channel appears more sparse in the beam delay domain, with most amplitudes being small. Only the largest values are crucial for accurately representing the channel in this domain.

These two domains can be interchangeably converted. To convert a wireless channel from the antenna frequency domain to the beam delay domain, a discrete Fourier transform (DFT) can be applied across the antenna elements and the PRBs. Conversely, to convert from the beam delay domain back to the antenna frequency domain, an inverse DFT is performed across the beam indices and delays.

2.3 Channel State Information (CSI) in 5G

2.3.1 CSI Report Mechanism

In a typical 5G system, the base station (BS) and user equipment (UE) collaborate to estimate the CSI. The BS transmits reference signals, known as CSI-RS (Channel State Information Reference Signals), at predetermined time and frequency resources. The user equipment (UE) measures the received signal quality and calculates several channel state information (CSI) parameters. These parameters include the channel quality indicator (CQI), which reflects the signal-to-noise ratio (SNR), the precoding matrix indicator (PMI), which suggests the preferred precoding matrix, and the rank indicator (RI), which determines the optimal transmission rank.

Figure 2.6 illustrates the sequence flow of Channel State Information (CSI) reporting procedures in 5G [17]. First, the BS configures the CSI-RS resources and informs the UE about the sounding schedule through higher layer (RRC) configuration. After that, depending on whether it is a periodic or aperiodic CSI report, the process differs. If it is a periodic CSI report, there is no need for a low layer trigger. However, if it is an aperiodic CSI report, it relies on a low layer trigger from the BS to inform the UE to start estimating the channel. Subsequently, the BS sends a predetermined CSI related reference signal to UE. Next, the UE then measures the CSI-RS and quantizes the CSI parameters according to a predefined codebook and reports them back to the BS through the uplink channel. The BS receives the CSI reports from this UE and uses this information to make scheduling decisions, assign appropriate modulation and coding schemes (MCS), and perform precoding to mitigate interference and maximize the signal quality at the receivers.

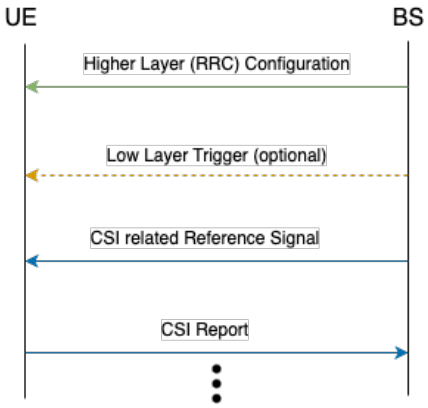


Figure 2.6: CSI Report Sequence Flow

2.4 Machine Learning with Artificial Neural Networks

Machine learning is a branch of artificial intelligence that focuses on developing algorithms that enable computers to learn on tasks without being explicitly programmed. By utilizing statistical methods and data-driven approaches, machine learning systems are expected to identify patterns, make predictions, or take actions based on the input dataset they are trained on. In the context of wireless networks, machine learning can be applied to various aspects such as channel prediction, resource allocation, signal detection, and network optimization.

2.4.1 Neural Network Overview

Neural networks [18] are a type of machine learning model inspired by the structure and function of the human brain. They consist of interconnected nodes called neurons, each neuron is a simple function, it receives input signals, processes them using an activation function, and passes the output to neurons in the next layer.

As shown in Figure 2.7, a neural network typically has three main types of layers: the input layer, one or more hidden layers, and the output layer. The input layer takes in the input features or variables. The hidden layers perform computations and transformations on the inputs using the learned connection weights and activation functions. The output layer produces the final prediction based on the processed inputs. During training, the connection weights are iteratively adjusted using optimization algorithms like gradient descent and backpropagation to minimize the difference between the predicted and ground truth outputs. This allows the neural network to learn an approximation of the underlying mapping function from inputs to outputs.

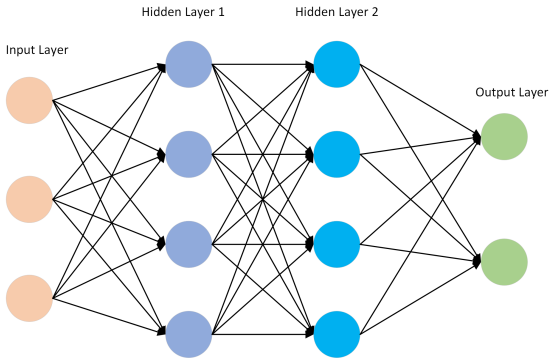


Figure 2.7: An example of neural network - A multi-layer perceptron (MLP) network, which is a class of feedforward neural network (FNN). The network comprises an input layer, two hidden layers, and an output layer. Each layer consists of nodes, also known as neurons.

2.4.2 Convolutional Neural Networks (CNNs)

Since MLPs process input data as flat vectors, ignoring spatial correlations, leading to higher number of parameters, computational costs, and overfitting risks. In response, researchers developed Convolutional Neural Networks (CNNs), which consider spatial relationships within the input data. Typically, CNNs consists of multiple building blocks, including convolutional layers, pooling layers, and fully connected layers. While CNNs were initially designed for image processing tasks [19], they have since been adapted to handle various other tasks, such as time series sequence forecasting [15, 20].

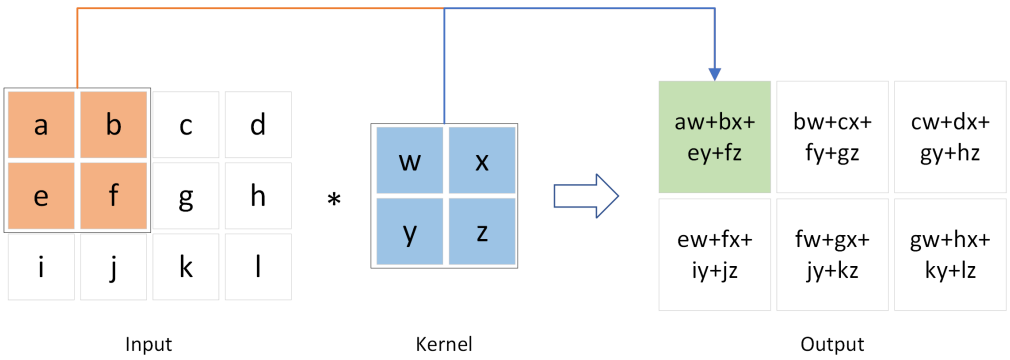


Figure 2.8: An example of 2-D convolution

Figure 2.8 illustrates the convolution operation [21], a key concept in convolutional neural networks (CNNs). The input is a 4x4 matrix containing numbers from 'a' to 'l'. The kernel, also known as the filter, is a 2x2 matrix with values 'w', 'x', 'y', and 'z'. The convolution operation involves sliding the kernel over the input matrix and computing the element-wise multiplication of the overlapping values, followed by summing up the results. The output is a 3x3 matrix, where each element is the

result of the convolution operation at the corresponding position. For example, the top-left element of the output matrix, '$aw+bx+ey+fz$', is obtained by element-wise multiplying the top-left 2x2 submatrix of the input with the kernel and summing the results. This process is repeated as the kernel moves across the entire input matrix, creating the output matrix with the resulting values.

Mathematically, in this case, the 2-D input data is represented as I and the kernel is represented as K, convolution can be written as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.6}$$

In this thesis, we will employ a classical CNN architecture, specifically the ResNet18 model proposed by He et al. [22], to predict the wireless channel. However, to apply the ResNet18 architecture to our input data, which may have different dimensions and characteristics compared to the original data the model was designed for, we need to make some modifications to the architecture. These modifications will tailor the ResNet18 model to the specific requirements of our channel prediction problem, ensuring that it can efficiently process and learn from our input data. The details of the modified ResNet18 architecture used in this thesis can be found in the Appendix A.1.

2.4.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are specifically designed to recognize patterns in sequences of data, such as text or numerical time series data. Unlike traditional feedforward neural networks, RNNs possess a unique feature - the memory to store information about previous inputs by creating loops in the network.

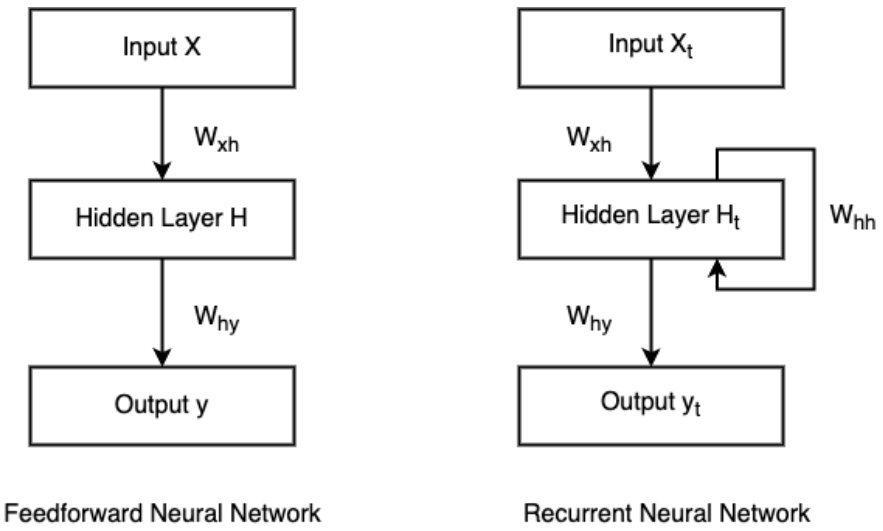
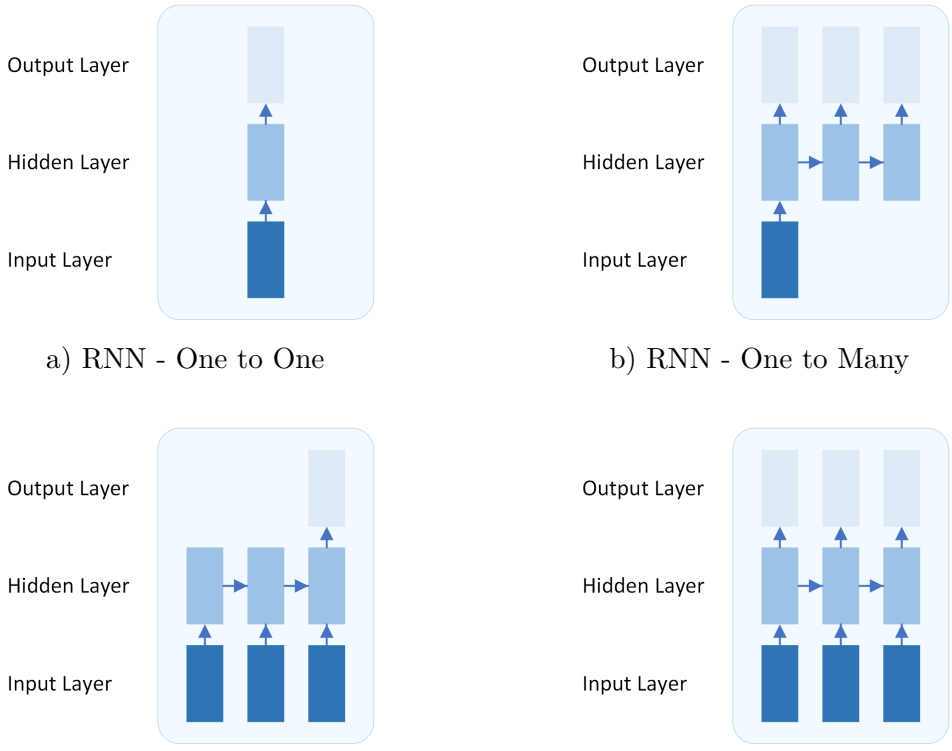


Figure 2.9: Data flow: Feedforward Neural Networks vs. Recurrent Neural Networks [23]

Figure 2.9 illustrates the key difference between Feedforward Neural Networks (FNNs) and Recurrent Neural Networks (RNNs) [23]. On the left, the FNN is depicted with an input layer (X), a hidden layer (H), and an output layer (Y). The information in an FNN flows unidirectionally from the input to the output, passing through the hidden layer. Each layer processes the information independently, without any feedback or recurrent connections.

In contrast, the RNN on the right side introduces a recurrent connection within the hidden layer. The hidden layer (H_t) not only receives the input (X_t) but also takes into account the previous hidden state (H_{t-1}) through a recurrent connection. This allows the RNN to maintain a form of "memory" or context from previous time steps, enabling it to process sequential or time-dependent data more effectively.

Figure 2.10 illustrates four different types of Recurrent Neural Networks (RNNs) based on their input-output configurations [24]: (a) "One to One" processes a single input to generate a single output, (b) "One to Many" takes a single input and produces a sequence of outputs, (c) "Many to One" processes a sequence of inputs to generate a single output, and (d) "Many to Many" handles a sequence of inputs and produces a sequence of outputs.



a) RNN - One to One

b) RNN - One to Many

c) RNN - Many to One

d) RNN - Many to Many

Figure 2.10: Different types of RNNs based on the number of inputs and outputs

2.4.3.1 Long Short-Term Memory (LSTM) Networks

Training the above RNNs can be challenging because of problems like vanishing and exploding gradients. These problems happen when the chain rule is applied through the recurrent connections during backpropagation, causing the gradients to become very small or very large. As a result, RNNs struggle to learn long-range dependencies in the input data, which limits their ability to handle long sequences.

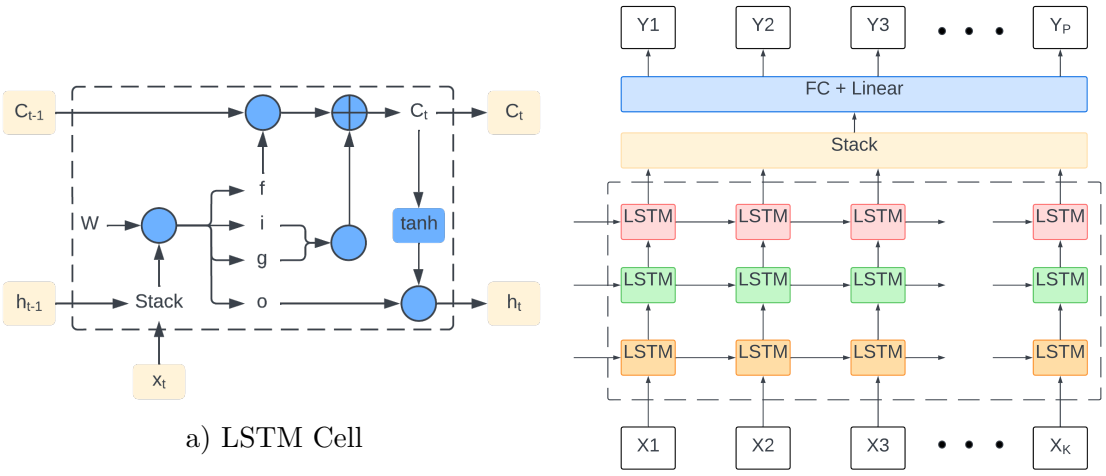
To address the limitations of traditional RNNs, Long Short-Term Memory (LSTM) [25] networks were introduced as a special kind of RNN which is capable of learning long-term dependencies. An LSTM network consists of multiple LSTM cells connected in a sequence, where each cell processes a single time step of the input data. The output of each LSTM cell is passed as input to the next cell in the sequence, allowing the network to maintain and propagate information over time.

At the heart of an LSTM network is the LSTM cell, which consists of several components, as illustrated in Figure 2.11(a) [26]. An LSTM cell has two important states: the cell state (C_t) and the hidden state (h_t). The cell state acts as the long-term memory of the cell, carrying relevant information throughout the processing of the sequence. The hidden state represents the output of the LSTM cell at each time step, based on the current input and the previous hidden state.

A LSTM cell also has four gates: Input(i), Forget(f), Modulation(g), and Output(0). The input gate (i) controls the extent to which new information is added to the cell state, while the forget gate (f) determines what information should be discarded from the cell state. The modulation gate (g) generates candidate values that could be added to the cell state, and the output gate (o) decides what information from the cell state should be output at the current time step. All gates in an LSTM cell use sigmoid activation functions to control the flow of information, except the modulation gate uses a hyperbolic tangent (tanh) activation function to generate candidate values. The mathematical formulas for the operations performed within a LSTM cell are as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.7}$$
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.8}$$
$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g) \tag{2.9}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{2.10}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot g_t \tag{2.11}$$
$$h_t = o_t \cdot \tanh(c_t) \tag{2.12}$$

where W_i, W_f, W_g, and W_o are weight matrices, b_i, b_f, b_g, and b_o are bias vectors, and $\cdot$ denotes element-wise multiplication.



a) LSTM Cell

b) LSTM - Many to Many

Figure 2.11: LSTM Cell Architecture and A LSTM Many-to-Many Network

Figure 2.11(b) illustrates a LSTM Many-to-Many network, where a sequence of inputs $(X_1, X_2, ..., X_K)$ is processed by a stack of LSTM cells to generate a corresponding sequence of outputs $(Y_1, Y_2, ..., Y_P)$. This architecture is commonly used for tasks such as sequence-to-sequence prediction, machine translation and time series forecasting.

2.4.4 The Transformer Architecture

Transformers have revolutionized the field of machine learning, first in the realm of natural language processing. Initially introduced in the paper 'Attention is All You Need' by Vaswani et al. in 2017 [27], the transformer architecture has set new standards for performance in various tasks, especially in sequence prediction. Unlike previous models that relied heavily on sequence-based processing through recurrent neural networks (RNNs) or convolutional neural networks (CNNs), transformers utilize a mechanism known as self-attention. This approach enables the model to weigh the importance of different parts of the input data, allowing for parallel processing and reducing computation time.

2.4.4.1 Self-Attention Mechanism

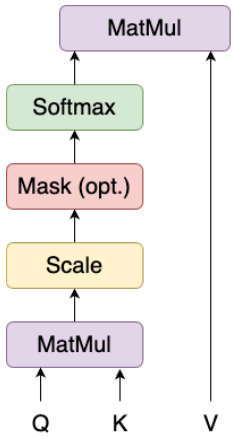


Figure 2.12: Scaled Dot-Product Attention [27]

The widely utilized self-attention mechanism is also known as Scaled Dot-Product Attention, as depicted in Figure 2.12, it starts with the input being converted into three sets of vectors: queries (Q), keys (K), and values (V). These vectors undergo a sequence of matrix operations, as shown in the figure. The self-attention in transformer architecture is typically expressed as follows.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.13}$$

In this process, the product of queries and keys is scaled by the inverse square root of the key's dimensionality (d_k), ensuring stable training behavior, and then a Softmax function is applied to derive the attention weights. These weights are then used to combine the value vectors, generating the final output. Note that the optional 'Mask' step can be used to prevent certain positions from influencing the attention, which is useful when the model needs to ignore padding or future tokens in specific tasks.

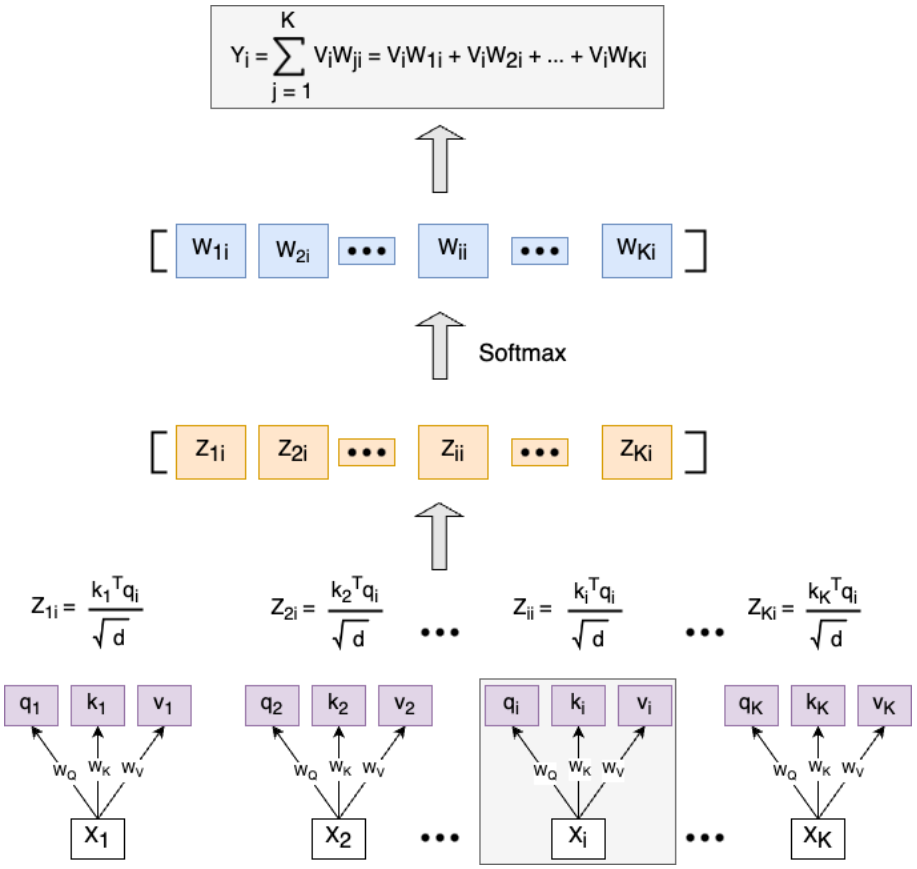


Figure 2.13: The Computation Flow of the Self-Attention Mechanism

Figure 2.13 illustrates the computation process within the self-attention mechanism [28]. At the base of the diagram, given an input sequence of vectors $(X_1, X_2, ..., X_K)$, the self-attention mechanism computes three matrices for each input vector: the query matrix q_i, the key matrix k_i and the value matrix v_i, these matrices are obtained by linearly transforming the input vectors using learnable weight matrices W_Q, W_K and W_V, respectively.

These vectors are then used to calculate the attention scores Z_{ij}, where each score is the dot product of a query q_i and a key k_j, normalized by the square root of the dimensionality d of the key vectors to maintain stable gradients during training. Subsequent to this, a softmax function is applied to the attention scores to obtain the weights W_{ij}, which represent the importance of each input's features when constructing the new representation Y_i of the i-th position.

The output vector Y_i is then computed as a weighted sum of the value vectors V_j, scaled by the softmax-normalized weights W_{ij}, effectively allowing the model to focus on the most relevant parts of the input sequence. This self-attention process enables the model to dynamically weigh the significance of different parts of the input, thus capturing contextual relationships within the data.

2.4.4.2 Transformer Encoder-Decoder Framework

As illustrated in Figure 2.14, the Transformer Encoder-Decoder model is composed of an encoder and a decoder, each consisting of a stack of identical layers. The encoder's primary function is to map an input sequence to another sequence, which the decoder then uses to generate an output sequence.

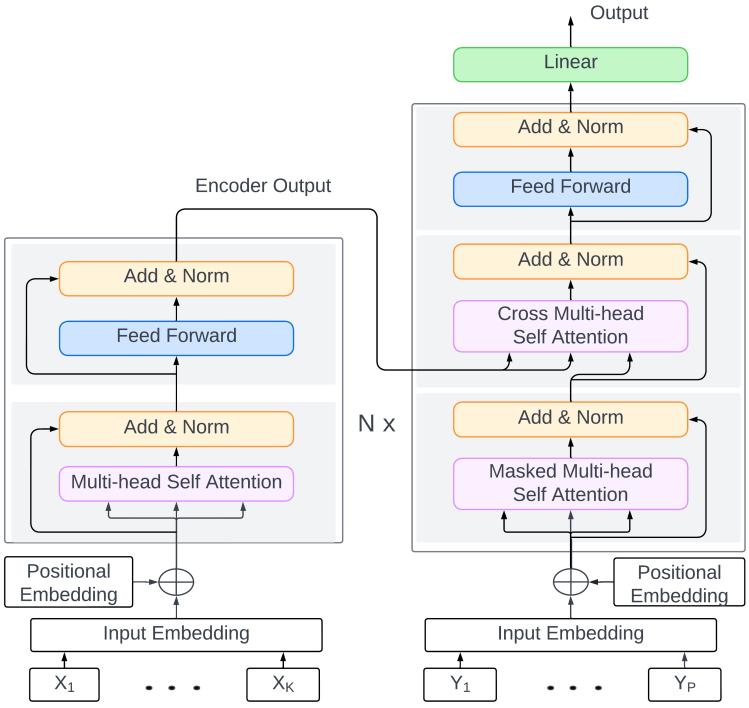


Figure 2.14: Transformer Encoder-Decoder architecture

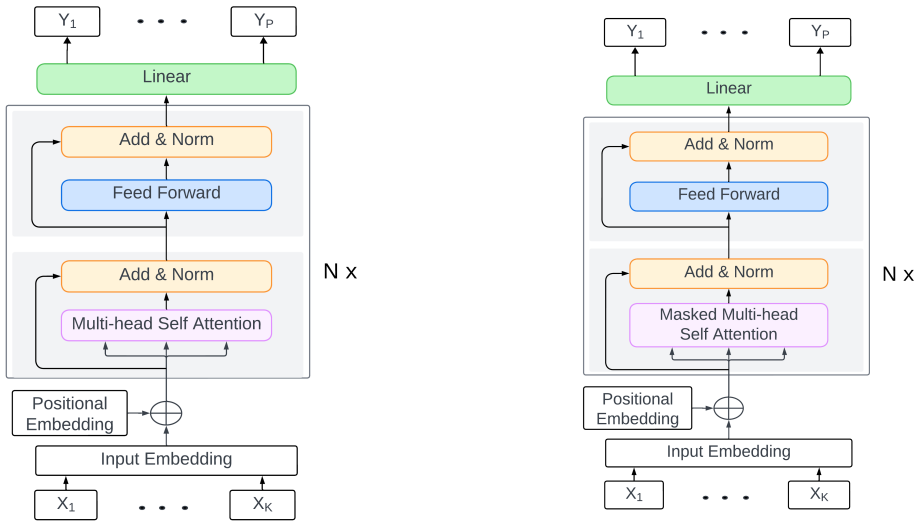
In the encoder, each layer comprises two main sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network, the multi-head self-attention mechanism allows the model to attend to different positions of the input sequence, capturing the dependencies between them, the fully connected feed-forward network consists of two linear transformations with a ReLU activation in between, helping the model to capture complex non-linear relationships in the input sequence. Each sub-layer in the encoder is followed by a residual connection and a layer normalization step. The residual connection adds the input of the sub-layer to its output, allowing the model to learn residual functions and facilitating the flow of information across layers.

On the decoder side, similar to the encoder, is composed of a stack of layers, but with an additional sub-layer that performs multi-head attention over the encoder's output. This is known as cross multi-head self-attention and it enables the decoder to focus on relevant parts of the input sequence during the generation of the output sequence. Prior to this cross-attention sub-layer, the decoder also includes a masked multi-head self-attention sub-layer which prevents positions from attending

to subsequent positions. This masking ensures that the predictions for a particular position can only depend on the known outputs at positions before it.

Both the encoder and decoder incorporate an input embedding layer to convert the input tokens into dense vector representations and positional encoding to inject information about the relative or absolute position of the tokens in the sequence.

2.4.4.3 Encoder-Only and Decoder-Only Transformer Variants



a) Transformer Encoder-Only

b) Transformer Decoder-Only

Figure 2.15: Transformer Variants: Encoder-Only and Decoder-Only

Transformer variants, such as encoder-only and decoder-only architectures, have emerged to address specific tasks and optimize performance in certain domains. The original transformer model consists of an encoder to process the input and a decoder to generate the output. However, for some tasks, an encoder-only or decoder-only model may be more appropriate. These specialized variants can offer computational efficiency and performance benefits by streamlining the architecture for the task, eliminating unnecessary components and focusing the model's capacity on the essential elements of the problem.

Encoder-only Transformers, like BERT (Bidirectional Encoder Representations from Transformers), focuses solely on understanding and generating representations of the input data. By removing the decoder component, encoder-only models are typically tailored for tasks that require a deep understanding of the input context, such as sentence classification.

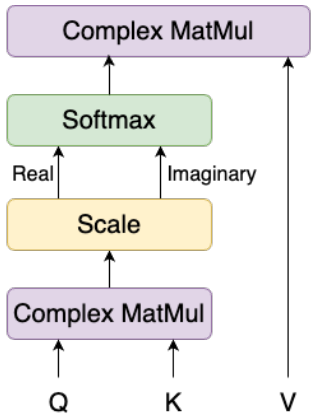
On the other hand, decoder-only Transformers, like GPT (Generative Pre-trained Transformer), is excel at generating output sequences based on a given context. In this configuration, the Transformer lacks the encoder component and relies entirely

on the decoder for both understanding the input and producing output. The self-attention in the decoder is masked to prevent the model from accessing future positions in the sequence during training, enforcing an auto-regressive property where the prediction for a particular position can only depend on previously generated positions. The decoder-only model iteratively generates each output symbol based on the accumulated sequence, thus differing from the encoder-only model, which processes the entire input in parallel.

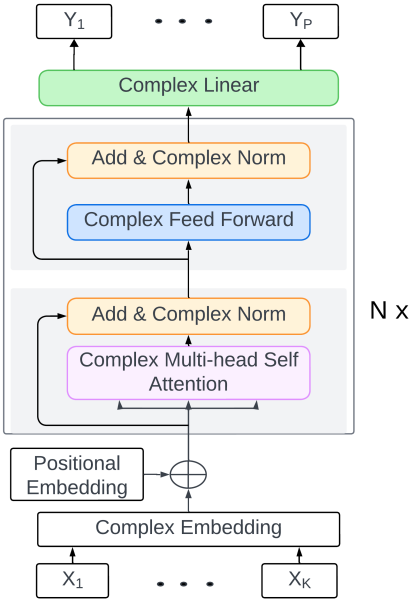
2.4.4.4 Transformer Built in Complex Domain

Transformer-based models have been successful in various fields, including natural language processing and computer vision, due to their ability to capture long-range dependencies and learn meaningful representations from input data. However, most transformer-based models are initially built to operate on real-valued data, as the majority of tasks in these domains deal with input data that is naturally represented in the real domain. However, there are certain tasks, particularly in the fields of audio and signal processing, where the input data is inherently complex-valued. In these fields, operations like the Fourier Transform play a crucial role in analyzing and processing the signals. The Fourier Transform allows for the decomposition of a signal into its frequencies, providing valuable insights into the signal's spectral content.

Given the importance of complex-valued operations in audio and signal processing tasks, there is a growing need for transformer-based models that can effectively handle complex-valued input data. In response to this need, researchers have developed transformer models that operate directly in the complex domain [29, 30]. These innovative architectures enable the models to leverage the full potential of complex-valued representations, allowing them to capture the rich information present in complex domain.



a) Self-attention in Complex Domain



b) Transformer Encoder-Only in Complex Domain

Figure 2.16: Transformer Encoder-Only in Complex Domain

Figure 2.16 illustrates the architecture of a Transformer Encoder-only model in the complex domain. Figure 2.16(a) depicts the self-attention mechanism in the complex domain, which is similar to the mechanism in the real domain but with some key differences. It begins with three complex matrices labeled Q (query), K (key), and V (value), however, which undergo complex matrix multiplication. The output is then scaled down and passed through a Softmax function to handle the real and imaginary components separately. The final step involves another complex matrix multiplication that combines the Softmax output with V, resulting in a complex representation of self-attention.

Figure 2.16(b) presents the overall structure of the Transformer Encoder-Only model in the complex domain. The architecture is similar to the Transformer Encoder-Only model in the real domain, but all components have been adapted to operate in the complex domain. The input undergoes Complex Embedding, followed by Complex Multi-head Self-attention and Complex Feedforward layers. Complex Norm Layers are used to normalize the complex-valued activations, and a final Complex Linear Layer projects the learned representations to the desired output format. By adapting each component to the complex domain, the model can effectively process and learn from complex-valued input data.

3

Methods

In this chapter, we present the main methodological framework, beginning with an outline of the overall design and approach. A detailed discussion on data collection and management follows. We then explore the data preprocessing steps employed, including beamspace transformation and feature selection and Data augmentation techniques. The development and evaluation of our model are covered next, where we discuss its design, how we train it, and the metrics we use to check its performance. After that, strategies for improving model generalization and robustness are presented, such as augmenting the training dataset diversity, incorporating side information (autocorrelation), and applying adaptive selection of hyperparameters. Finally, the model inference pipeline is described.

3.1 Overall Design and Approach

Figure 3.1 outlines a three-stage process for handling data and developing a neural network model. In the first stage, the 'raw dataset' undergoes preprocessing to prepare it for the model; this typically involves cleaning, normalization, and possibly feature extraction or selection. Once preprocessed, the dataset is split into three parts: 'training dataset', 'validation dataset' and 'testing dataset'. The training dataset is used to teach the model to recognize patterns and make predictions, while the validation dataset helps in tuning the model parameters and preventing overfitting.

In the second stage, the 'model' is trained using the training dataset. This involves adjusting the model's internal parameters until it performs well on the training data. The validation dataset is used throughout this process to validate the model's performance and ensure it generalizes well to new, unseen data. After the model has been trained and validated, it moves to the third stage -'model inference' Here, the model is applied to the testing dataset to evaluate its final performance. The output of this stage is the 'inference result', which indicates how well the model can predict outcomes based on new data inputs. This end-to-end process is crucial for developing robust models that can be trusted to make accurate predictions.

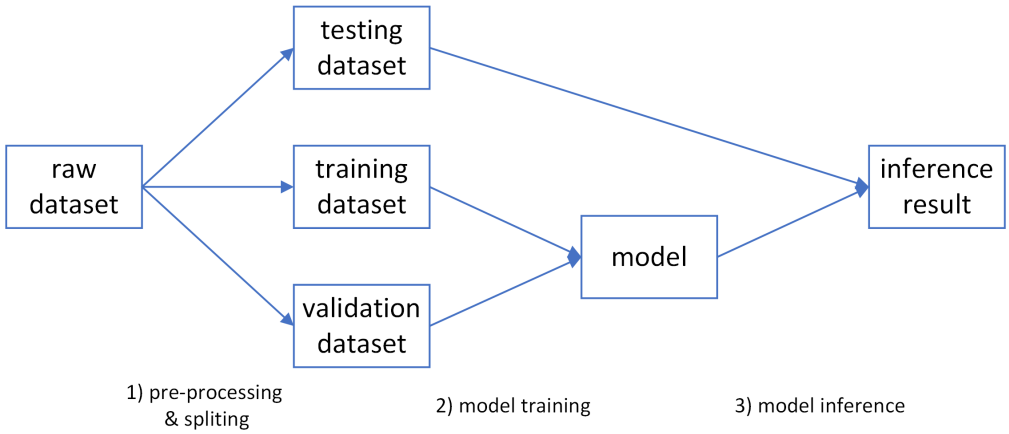


Figure 3.1: Data Preprocessing, Model Training, and Inference Workflow

Building on the workflow above, Figure 3.2 provides a more detailed look at the utilization of the training dataset within the model training phase. The training dataset, comprising millions channel feature samples labeled $[H_j(t_1), H_j(t_2), ..., H_j(t_K)]$, and target samples $[H_j(t_P), H_j(t_{P+1}), ...]$, represents the data that the model will learn from. These feature samples serve as input data to the Deep Neural Network (DNN) model for training. Before being fed into the DNN, the training data may undergo optional pre-processing steps, the pre-processed data is then passed through the DNN, which learns to map the input data to the corresponding target. As the data flows through the network, the model learns to make predictions, which are then compared to the target samples. The 'Post-processing' step, also marked as optional, could involve transforming the predictions to a different format.

The predictions made by the model are evaluated using a 'Loss Function,' such as Mean Squared Error (MSE) or Normalized Mean Squared Error (NMSE), to quantify how well the model's predictions match the actual targets. The loss function provides feedback to the neural network, which is used to adjust the model's weights and biases in the direction that reduces prediction error. This process is repeated iteratively until the model converges.

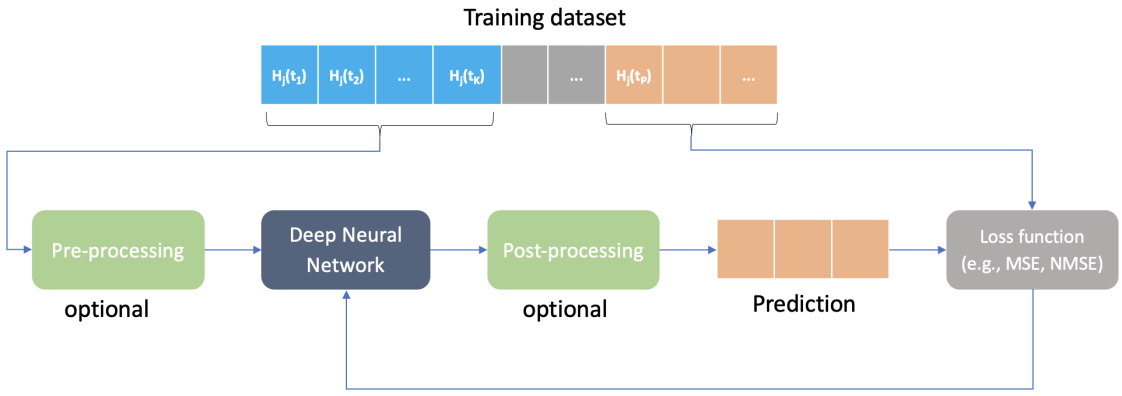


Figure 3.2: workflow of the training and validation process

3.2 Initial Benchmarking Approach

As outlined in Chapter 1, since autoregressive (AR) models is one of the classical methods for channel prediction. In this thesis, we employ the AR model in conjunction with Burg's algorithm as a benchmarking tool. In AR model, the channel prediction at a future time point $\hat{H}_{t+1}$ is calculated using the weighted sum of K prior observations. The predictive equation for the AR model is expressed as follows:

$$\hat{H}_{t+1} = \sum_{i=1}^{K} \alpha_i H_{t-i+1} \tag{3.1}$$

Where α_i is the coefficients which correspond to each of the K previous observations.

For Burg's algorithm, it is an approach used to determine the optimal coefficients in autoregressive (AR) models by minimizing the forward and backward prediction (least mean-squared) error, for more details, please refer to [7].

3.3 Data Collection and Management

3.3.1 Data Sources

The provided datasets, one for 30 km/h and another for 60 km/h User Equipment (UE) velocity, were generated through simulations using the Urban Macro (UMa) channel model. Each dataset has dimensions of (850, 150, 4, 32, 52), where:

- 850 represents the number of UE instances,
- 150 corresponds to the number of time slots,
- 4 indicates the number of receive antennas,
- 32 represents the number of transmit antennas, and
- 52 denotes the number of Physical Resource Blocks (PRBs).

The UMa channel model is a standardized model defined by 3GPP [31], it is designed to emulate the characteristics of real-world radio channels, such as multipath propagation, delay spread, and spatial correlation. Table 3.1 illustrates part parameters used for data generation. As we can see from the table, the Base Station (BS) is equipped with an array of subarrays, with a configuration of 2 rows, 8 columns, and 2 polarizations (2, 8, 2). The BS antenna elements follow the 3GPP standard. The UE is also equipped with an array, having a configuration of 1 row, 2 columns, and 2 polarizations (1, 2, 2). The carrier frequency is set to 2 GHz and the bandwidth is 10 MHz, and the inter-site distance between BS and UE is 200 meters. The UE velocity is set at two different speeds: 30 km/h and 60 km/h. By setting these parameters and using the UMa channel model, a dataset can be generated that captures the essential characteristics of the wireless channel in an urban macro-cell environment.

Table 3.1: Part of Parameters used for Dataset Generation

Parameter	Value
Scenario	Urban Macro, Outdoor UEs only
Base Station Array Configuration	2 Rows × 8 Columns × 2 Polarizations
Carrier Frequency	2 GHz
Bandwidth	10 MHz
BS Antenna Element	3GPP
UE Velocity	30 km/h and 60 km/h
UE Array Configuration	1 Rows × 2 Columns × 2 Polarizations
Inter-Site Distance	200 m

3.3.2 Data Privacy and Ethics

In this thesis, the prediction and analysis of channels inherently involve the use of simulated data without any real-world personal information, which eliminates concerns about privacy and data security.

3.4 Data Preprocessing

3.4.1 Beamspace Transformation

Prior research, as referenced in [32], has shown that channel representation in the beam domain outperforms representations in other domains. Based on this insight, we initially convert the channel H_j from the antenna domain to the beam domain, denoted as G_j. Throughout the remainder of this thesis, the beam domain will be utilized for all model training, inference and benchmarking purposes.

Consider the channel for Physical Resource Block (PRB) j represented as:

$$H_j \in \mathbb{C}^{N_{\mathrm{RX}} \times N_{\mathrm{TX}}}, j = 1, 2, ..., N_{PRB} \tag{3.2}$$

Here, suppose $W_1 \in \mathbb{C}^{N_{\mathrm{TX}} \times 2L}$ is the transformation matrix that applies the Discrete Fourier Transform (DFT) to transition the channel from the antenna domain to the beam domain and L is the number of single polarized beams, the beamspace channel for the j-th PRB is given by:

$$G_j = H_j W_1 \in \mathbb{C}^{N_{\mathrm{RX}} \times 2L}, j = 1, 2, ..., N_{PRB} \tag{3.3}$$

Let $g_j(t) \in \mathbb{C}^{2LN_{\mathrm{R}} \times 1}$ be the vectorized version of $G_j(t)$, for beamspace channel at time $t_p > t_K$, $\hat{\mathbf{g}}_j(t_p)$ is predicted using K equi-spaced measurements $[g_j(t_1), g_j(t_2), ..., g_j(t_K)] \in \mathbb{C}^{2LN_{\mathrm{R}} \times K}$.

The following is the code snippet for transforming the channel from the antenna domain to the beam domain.

```
for i in range(sample_number):
    for j in range(prb_number):
```

```
        # pick one sample of size (150, 4, 32)
        sample_picked = data_raw[i, :, :, :, j]

        # reshape the transmit dimension of sample from 32 to
    (2,8,2)
        sample_picked_reshaped = np.reshape(
            sample_picked, (150, 4, 2, 8, 2), order='F')

        # fft across the axes (-3, -2), size (150, 4, 2, 8, 2)
        sample_picked_fft = np.fft.fft2(
            sample_picked_reshaped, axes=(-3, -2))
```

3.4.2 Feature Selection

Figure 3.3 illustrates the process of extracting samples (sequences of channel) from a dataset with dimensions (850, 150, 4, 32, 52). The extraction process involves several key parameters: m, d, and δ. The parameter m represents the spacing (in slots) between consecutive measurements, while d denotes the spacing between predictions. The parameter δ indicates the spacing between the last measurement and the first prediction. The sample extraction process begins with the User Equipment (UE) estimating the channel over K slots. These K measurements are uniformly separated by m slots. Based on the collected K measurements, the UE then predicts the future channel state for P slots. In this thesis, for simplicity, the values of m, δ, and d are all set to 5, resulting in an equi-spaced configuration.

To extract samples, the measurements and predictions are progressively shifted to the right. This sliding window approach allows for the generation of multiple samples from the dataset. Initially, the values of K and P are set to 5 and 4, respectively. This configuration implies that the UE predicts the channel state for the next 4 instances based on the previous 5 instances of measurements. However, to enhance the generalization ability of the model, the restriction on K and P will be relaxed in a later section (3.6) of the thesis.

By extracting samples in this manner, the dataset is transformed into a collection of input-output pairs, where the input consists of K measurements and the output consists of P predictions. These samples serve as the foundation for training and evaluating the channel prediction model.

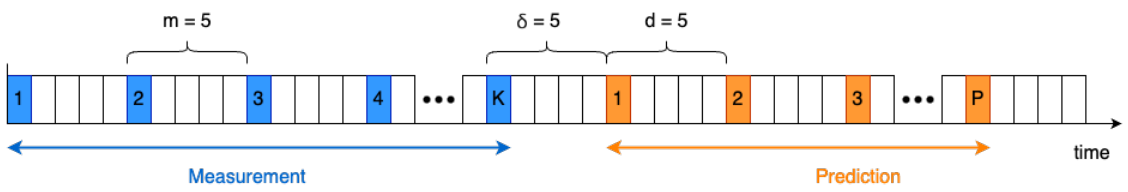


Figure 3.3: Channel Prediction: Measurement and Prediction Configuration

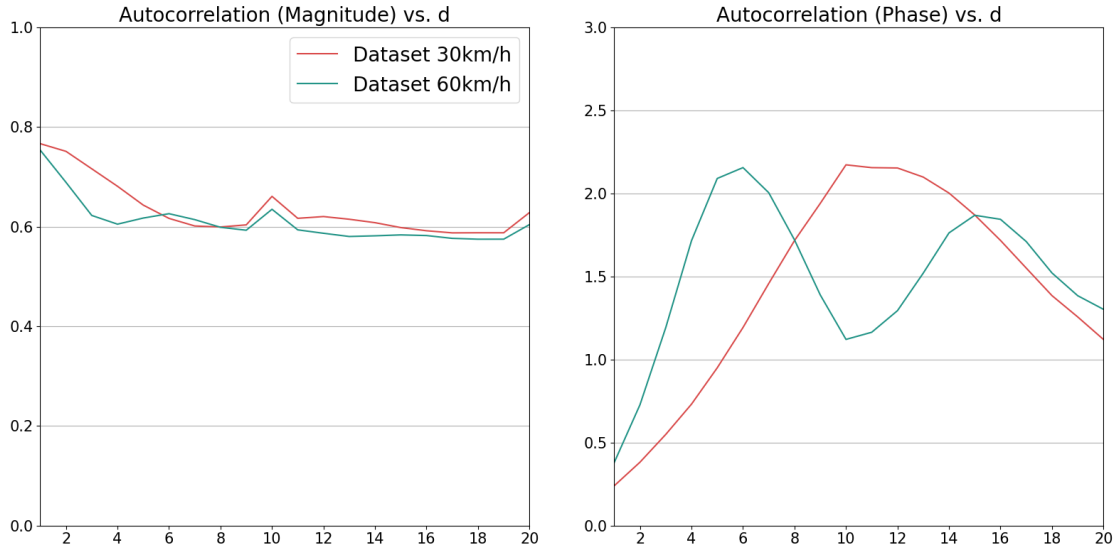
3.4.3 Data Augmentation Techniques

To enhance the generalization ability of a neural network, increasing the size and diversity of the training data is crucial. In real-world scenarios, the mobility speed of User Equipment (UE) can vary significantly, ranging from a walking speed of 3 km/h to a driving speed of 180 km/h. However, the given dataset only contains data for two specific speeds: 30 km/h and 60 km/h. To address this limitation, data augmentation techniques can be employed to generate additional data representing different velocities.

Autocorrelation is an effective method for assessing the rate how channel changes over time. The autocorrelation can be calculated using the following equation:

$$A_c(d) = \frac{\sum G_j(t+d) \cdot G_j^*(t)}{\frac{1}{2}\sum(G_j(t) \cdot G_j^*(t) + G_j(t+d) \cdot G_j^*(t+d))} \tag{3.4}$$

Where $A_c(d)$ represents the autocorrelation at a spacing of d slots, $G_j(t)$ denotes the channel at time t, and $G_j^*(t)$ is the complex conjugate of $G_j(t)$.



a) Autocorrelation (Magnitude)

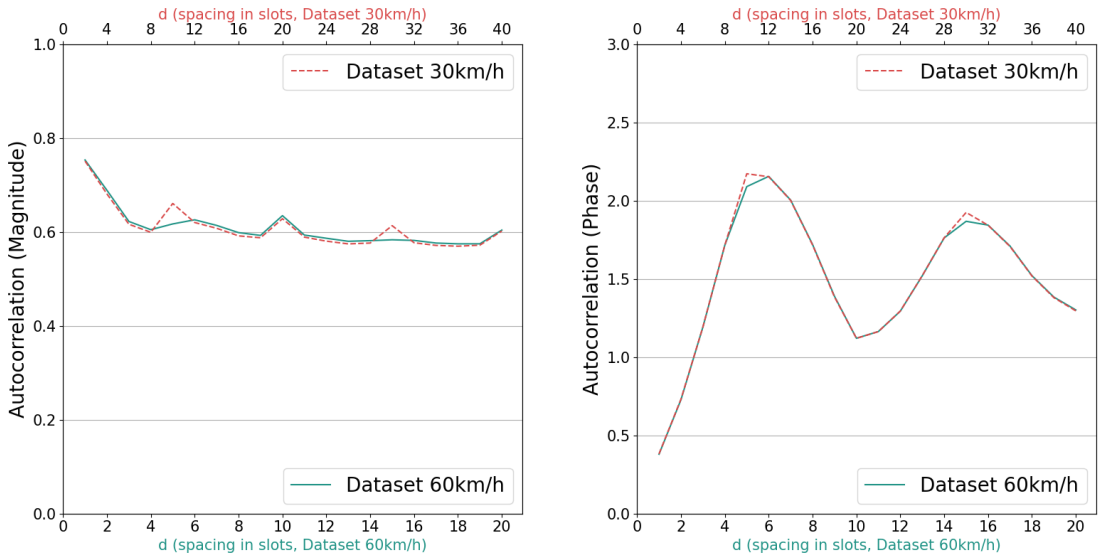
b) Autocorrelation (Phase)

Figure 3.4: Comparison of Channel Autocorrelation: 30 km/h vs. 60 km/h

Figure 3.4 illustrates the magnitude and phase autocorrelation of the channel with respect to the spacing d (in slots). The plots compare the autocorrelation behavior for two different UE speeds: 30 km/h and 60 km/h. It is evident from the figure that the autocorrelation for a velocity of 60 km/h varies more rapidly than that of 30 km/h. This observation aligns with the intuition that higher mobility speeds lead to more rapid changes in the channel over time.

Expanding on the concept of data augmentation in this context, we consider a fixed equi-spaced configuration where ($m = \delta = d = 5$), denoted as (d) for simplicity in the following sections. Suppose the current dataset corresponds to a UE velocity of (v). The question then arises: is it possible to synthesize a dataset for a doubled UE velocity, ($2v$), simply by adjusting the spacing to ($2d$)? The rationale is straightforward: assume two UEs traveling at different speeds along the same path and interacting with a base station (BS). Both UEs perform channel estimations at fixed intervals; the only distinction between them lies in the spacing of the collected data. For instance, we could potentially derive a dataset for a velocity of 120 km/h by doubling the spacing (d) from that of the 60 km/h dataset.

Figure 3.5 supports this hypothesis by showing the average autocorrelation w.r.t (d) for the 30 km/h and 60 km/h datasets, with the x-axis for the 30 km/h dataset scaled to twice that of the 60 km/h dataset. It can be observed that the autocorrelation curves for both datasets follow remarkably similar patterns, albeit with minor discrepancies. This similarity suggests that it is indeed feasible to generate data corresponding to a velocity of 60 km/h from the 30 km/h data by simply doubling the spacing (d). By extension, this approach implies that we can generate datasets for different velocities by adjusting the value of (d) accordingly.



a) Autocorrelation (Magnitude)

b) Autocorrelation (Phase)

Figure 3.5: Comparison of Channel Autocorrelation: 30 km/h vs. 60 km/h over different scale of d (spacing in slots)

3.5 Model Development and Evaluation

3.5.1 Model Architecture and Selection

Figure 3.6 provides an overview of the four main types of model architectures explored for channel prediction in this thesis. The first type is the SimpleNN (Simple Neural Network), which consists of two linear layers with a ReLU activation function between them. This simple neural network serves as a baseline for comparison with more complex architectures.

The second type is a CNN (Convolutional Neural Network), specifically ResNet18. However, since the original ResNet18 is designed for image classification with three input channels (RGB) and an output size of 1000, the architecture was modified to align with the channel prediction task. The adapted ResNet18 accepts two input channels, corresponding to the two polarizations of the channel.

The third type is the classical LSTM (Long Short-Term Memory) Many-to-Many model, which is well-suited for sequence-to-sequence predictions.

Lastly, the fourth type explores various transformer architectures, including encoder-decoder, encoder-only, and decoder-only models. For the encoder-only and decoder-only variants, both real and complex domain implementations were investigated to assess their performance in channel prediction tasks.

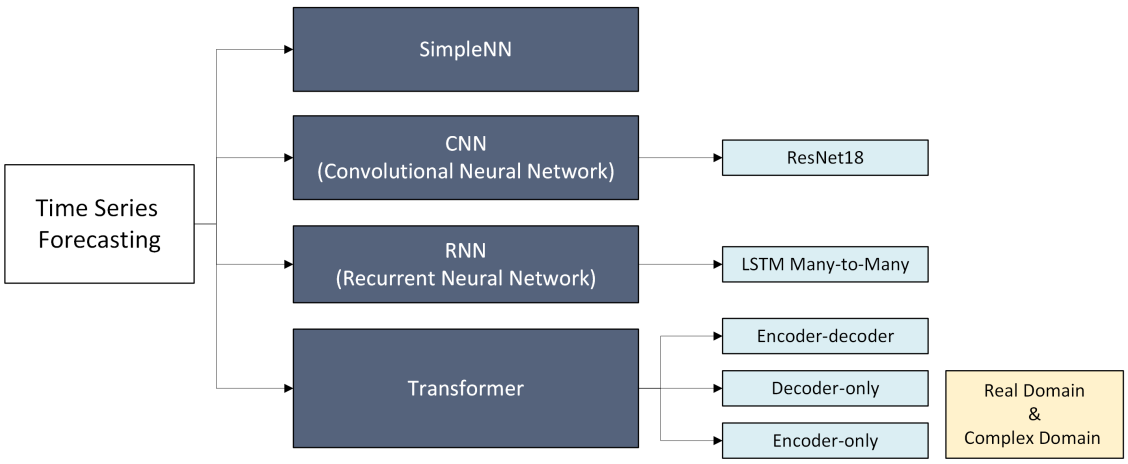


Figure 3.6: Model Architecture and Selection

3.5.2 Training and Validation Strategy

To train and evaluate the performance of the model, the dataset consisting of 850 UE instances is first divided into two groups randomly. A total of 800 UE instances are allocated for training and validation, while the remaining 50 UE instances are reserved for inference purposes.

From the 800 UE instances designated for training and validation, samples are extracted using the above described section. After sample extraction, the data is further split into two subsets: 95% of the samples are used for training the model, while the remaining 5% are utilized for validation. By following this training and validation strategy, the model's performance can be effectively evaluated, and its generalization capabilities can be assessed.

The overall strategy for model training involves exploring all four different architectures to assess their effectiveness in the channel prediction task, and each model is trained from scratch. After evaluating the performance of each model, the best-performing model is selected for further investigation and enhancement.

3.5.3 Model Evaluation Metrics

In this thesis, the Normalized Mean Squared Error (NMSE) is employed as the loss function and evaluation metric to assess the performance of different channel prediction models. NMSE is a widely used measure that quantifies the difference between the predicted and true values, its mathematical formulation is as follows:

$$\text{NMSE} = \mathbb{E}\left[\frac{||G_{true} - G_{predict}||_F^2}{||G_{true}||_F^2}\right] \tag{3.5}$$

where G_{true} represents the ground truth channel matrix in the beam domain and $G_{predict}$ denotes the corresponding predicted channel matrix obtained from the model. A lower NMSE value indicates better prediction accuracy, as it suggests that the predicted values are closer to the ground truth. Conversely, a higher NMSE value implies larger discrepancies between the predictions and the true values, indicating poorer prediction performance.

3.6 Enhancing Model Generalization and Robustness

3.6.1 Augmentation of Training Dataset Diversity

As described in section 3.4.3, Table 3.2 presents the results of the data augmentation process. We first set $K = 5, P = 4$, indicating that 5 previous slots of channel information are provided to predict the future 4 time slots. To synthesize datasets with varying velocities ranging from 6 km/h to 192 km/h, we adjust the spacing d from 1 to 16. This approach allows for the generation of a diverse dataset that encompasses a wide range of velocities. However, due to the limited total length of time slots in the raw dataset (150 slots), it is not feasible to extract samples for d values larger than 16.

It is important to note that during the creation of the new mixed dataset, two synthesized velocities, namely 12 km/h and 120 km/h, are intentionally omitted from the training dataset. By excluding these specific velocities, we can utilize them in the testing datasets to better assess the generalization ability of the trained model.

Table 3.2: Synthesized Data from Dataset 30km/h and 60km/h

d (spacing in slots)	Dataset 30km/h	Dataset 60km/h
1	6 km/h	12 km/h (skip)
2	12 km/h (skip)	24 km/h
3	18 km/h	36 km/h
4	24 km/h	48 km/h
5	30 km/h	60 km/h
6	36 km/h	72 km/h
7	42 km/h	84 km/h
8	48 km/h	96 km/h
9	54 km/h	108 km/h
10	60 km/h	120 km/h (skip)
11	66 km/h	132 km/h
12	72 km/h	144 km/h
13	78 km/h	156 km/h
14	84 km/h	168 km/h
15	90 km/h	180 km/h
16	96 km/h	192 km/h

3.6.2 Incorporation of Side Information (Autocorrelation)

Incorporating side information is another approach to enhance the model's generalization and robustness. In this case, we explore the possibility of leveraging the autocorrelation between slots of the channel as additional input to the model. Autocorrelation captures the temporal dependencies and patterns within the channel

data, providing valuable insights into how the channel evolves over time.

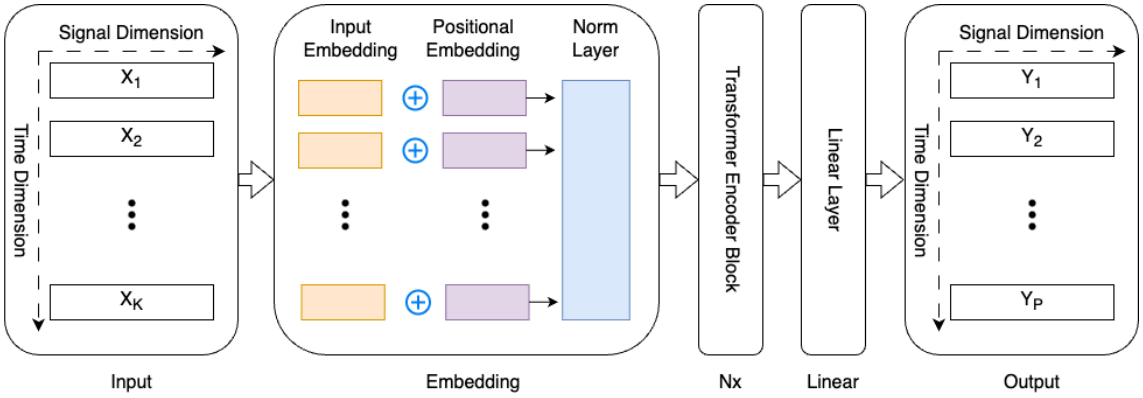


Figure 3.7: Transformer Encoder-Only without Adding Side Information

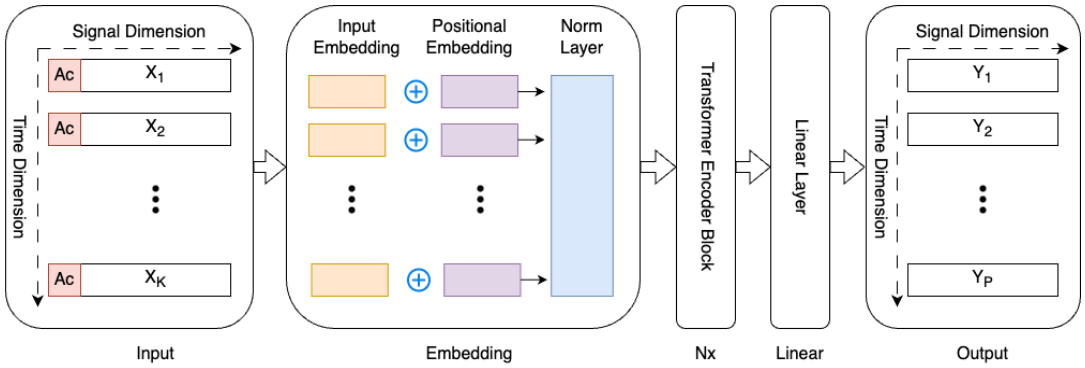
Among the architectures discussed earlier, the transformer encoder-only model demonstrates the best performance in the channel prediction task. Building upon its success, we investigate the potential benefits of incorporating autocorrelation as side information into this model. Figure 3.7 illustrates the transformer encoder-only model without side information. In this setup, K slots of channel data are fed into the network to predict the next P slots.

The process of incorporating autocorrelation as side information involves preprocessing the channel data to compute the autocorrelation coefficients. These coefficients are then combined with the channel data to provide additional temporal context to the model. In this study, the autocorrelation coefficient is calculated by taking the average value of the autocorrelation between each channel at time t and its corresponding value at the previous time step $t-1$, the resulting average autocorrelation is a complex number.

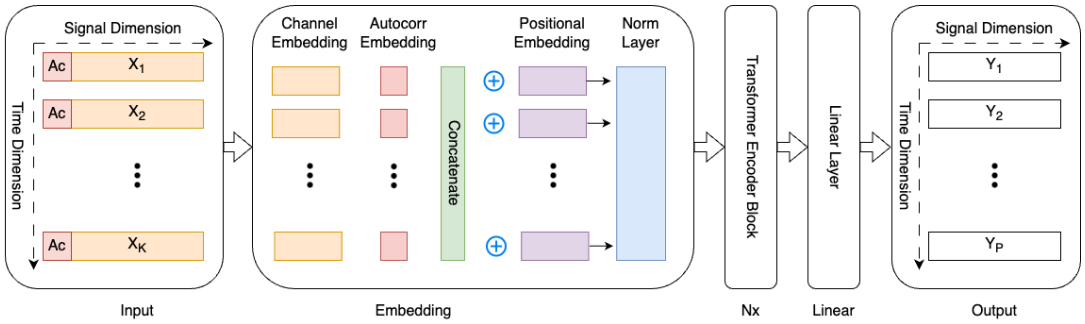
To explore different approaches for adding autocorrelation as side information into the transformer encoder-only model, we investigate four methods:

(a) Autocorrelation concatenated to the input channel data.
(b) Autocorrelation added using a separate embedding layer and then concatenated with embedded channel.
(c) Autocorrelation added using separate embedding layer and then added to the embedded channel.
(d) Autocorrelation added using separate embedding and then concatenated with the output of the transformer block (and thus not included in the self attention).

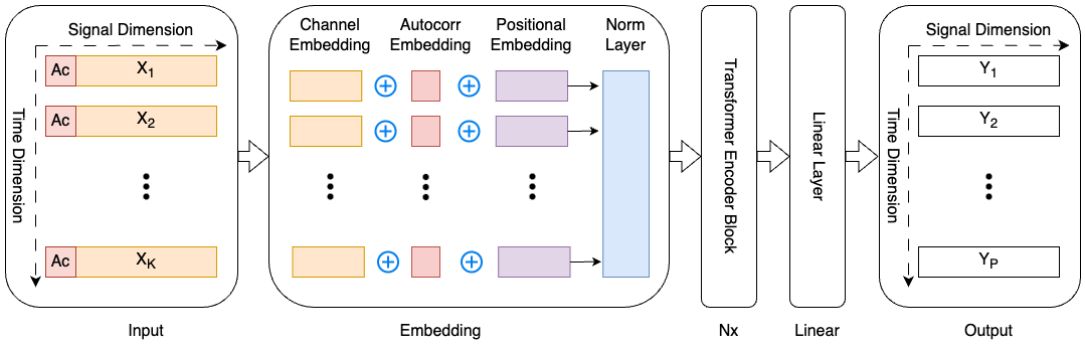
The corresponding model architectures are shown in figure 3.8 (a)-(d).



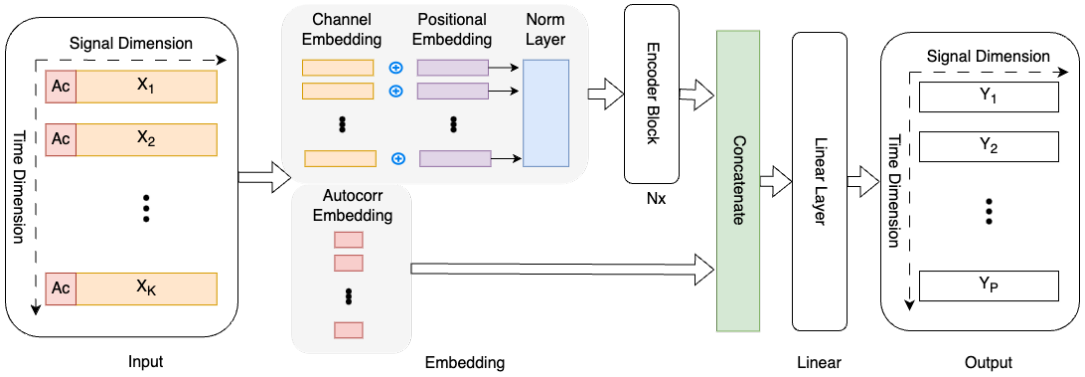
(a) Autocorrelation concatenated with input



(b) Autocorrelation in Embedding Layer, then Concatenated



(c) Autocorrelation in Embedding Layer, then Added



(d) Autocorrelation in Embedding, Concatenated Post-transformer

Figure 3.8: Four Different Methods of Incorporating Autocorrelation as Side Information in the Transformer Encoder-Only Model

3.6.3 Adaptive Selection of Hyperparameters K and P

In previous sections, we fixed number of measurements K to 5 and number of prediction P to 4. However, to train a model with better generalization ability, it is beneficial to investigate the impact of using flexible values for K and P.

As illustrated in Figure 3.9, To achieve this flexibility, we introduce a range for K, allowing it to vary from 5 to 12. This means that the model can accept input sequences of different lengths within this range. To ensure consistent input dimensions during training and inference, we employ a padding technique. Specifically, for each input sequence, we pad the left side with $0 + 0j$ (complex zero) values until the sequence reaches the maximum length of $K_{\text{MAX}} = 12$. By incorporating this adaptive selection of K, the transformer encoder-only model learns to handle different amounts of historical channel information.

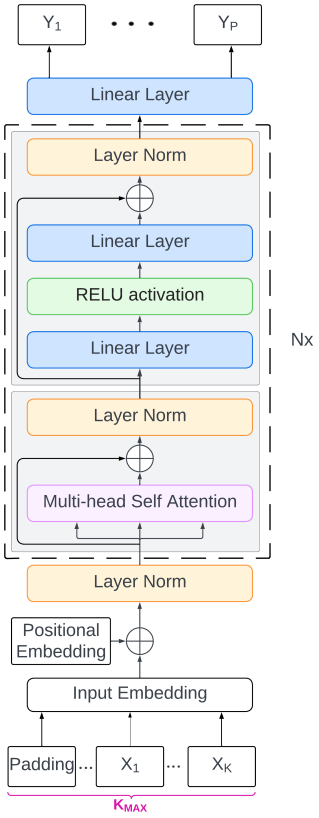


Figure 3.9: Transformer Encoder-Only: Adaptive Length of K

3.7 Model Inference

3.7.1 Inference Pipeline

The general inference pipeline for different models follows the same process as the evaluation pipeline when the number of measurements K and the number of predictions P are consistent with the training dataset. However, when performing

inference with a larger number of predictions, some adjustments are necessary. The following paragraphs explain how to perform inference in such scenarios for transformer encoder-only model.

The inference pipeline of the transformer encoder-only model, as illustrated in Figure 3.10, consists of two rounds. In this setup, we set input sequence length K_{MAX} to 12 and the output sequence length to $2P$, which is doubled compared to the training process. In the first round, the model takes an input sequence of length K to generates an output sequence of length P, where the input sequence represents the historical channel information and the output sequence is the predicted future channel.

In the second inference round, the model leverages the concept of a sliding window to generate predictions for the subsequent time steps. The input sequence is updated by shifting the window to the right, incorporating the previously predicted values. Specifically, the new input sequence becomes $[X_{P+1}, ..., X_K, Y_1, ..., Y_P]$, where the first $K - P$ values are the actual channel measurements from the previous time steps, and the last P values are the predicted channel values from the first inference round. The model processes this input and generates a new output sequence of length P. By concatenating the output of two round, we can get the final channel prediction.

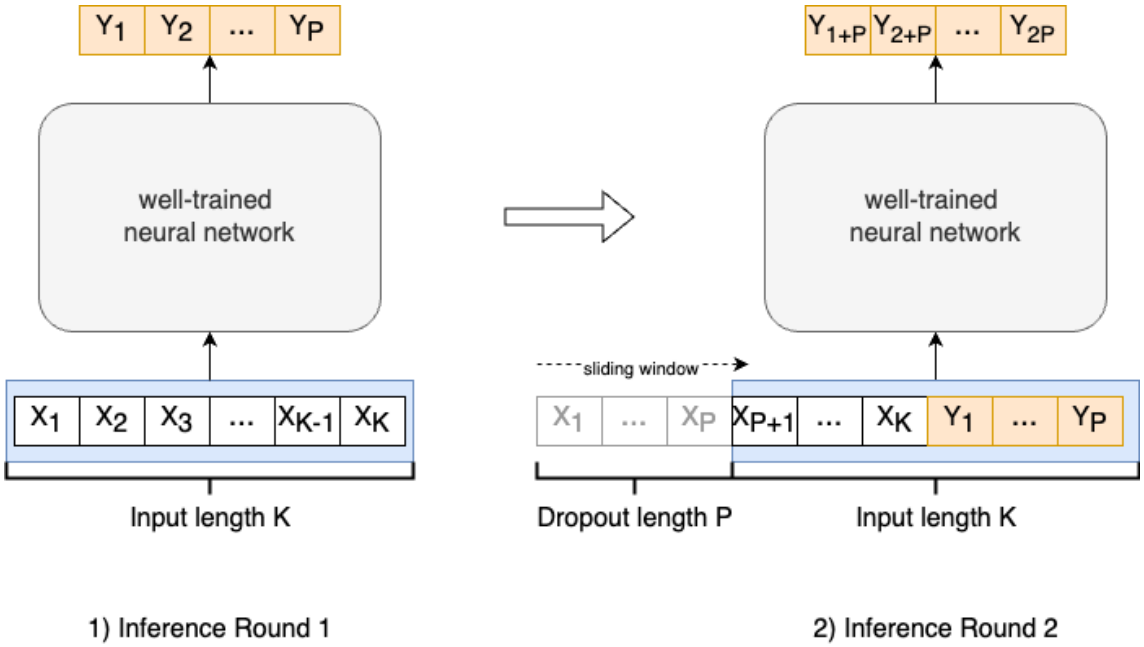


Figure 3.10: Model Inference Pipline

4

Results and Analysis

In this chapter, the results and analysis of the research are presented. The chapter begins with a visualization of wireless channel characteristics, followed by an evaluation of the model performance metrics, including an analysis of training loss and validation loss. A comparative analysis of inference performance is then conducted, examining temporal vs. signal feature embedding efficacy, performance on variable speed datasets, domain-specific model performance, the impact of autocorrelation on model accuracy, adaptive hyperparameters K and P, and single vs. multiple receive antennae performance. The chapter concludes with a summary of the key findings.

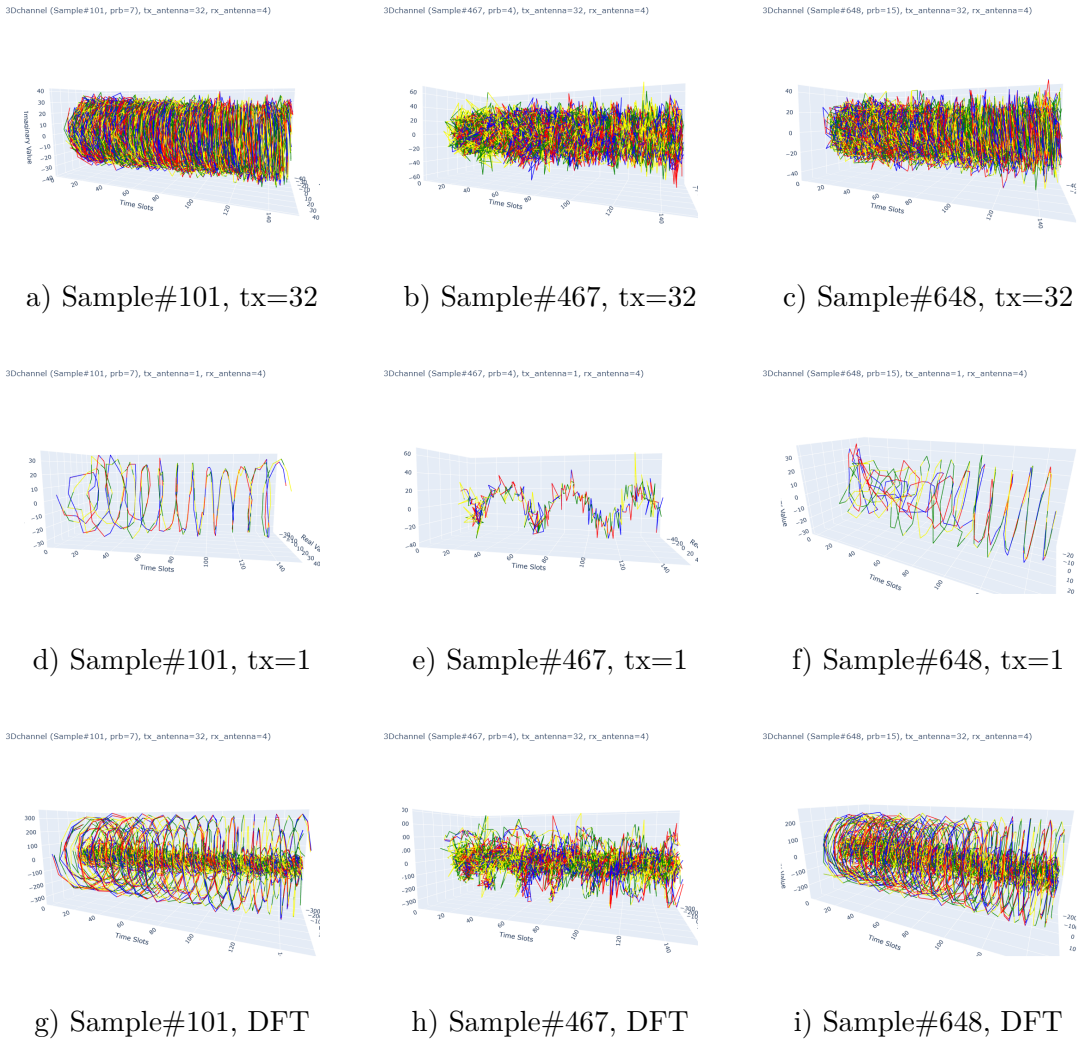
4.1 Visualization of Wireless Channel Characteristics

4.1.1 Channel Representation in 3D Space

Figure 4.1 provides a detailed visual representation of the MIMO channel characteristics in three dimensions. In our MIMO scenario, the transmitter (tx) is equipped with 32 antennas, and the receiver (rx) has 4 antennas. Out of the 44,200 samples from 850 UE instances, each with 52 PRBs, we randomly select three samples for visualization. Since the receiver has 4 antennas, we represent the channel response for each Rx antenna using four distinct colors: red, yellow, green, and blue. Each time slot yields a (32, 4) matrix where each element is a complex number. The 3D plots are thus constructed with the real parts represented along the x-axis, the imaginary parts along the y-axis, and the temporal dimension along the z-axis, displaying 150 time slots for each sample. For each tx-rx antenna pair, a line plot illustrates the temporal evolution of the signal.

The first row of subplots (a), (b), and (c) displays the three chosen samples in the antenna frequency domain. Each receive antenna collects 32 transmit signals that vary over time, resulting in 32 lines per color. As observed from Figure 4.1(a), (b), and (c), although each channel sample exhibits distinct characteristics, a pattern of rotation in the 3D space is discernible across time. To provide a clearer observation, subplots (d), (e), and (f) extract a single randomly selected tx antenna from the 32 available. It is evident that each sample presents a different shape. In subplots (g), (h), and (i), we convert the channel from the antenna domain to the beam domain. The converted channel still exhibits a rotating behavior in the 3D

space, similar to the antenna domain. However, the key difference is that the channel in the beam domain appears more sparse, with fewer significant components. This sparsity can be attributed to the fact that the beam domain representation focuses on the dominant propagation paths.



a) Sample#101, tx=32 b) Sample#467, tx=32 c) Sample#648, tx=32

d) Sample#101, tx=1 e) Sample#467, tx=1 f) Sample#648, tx=1

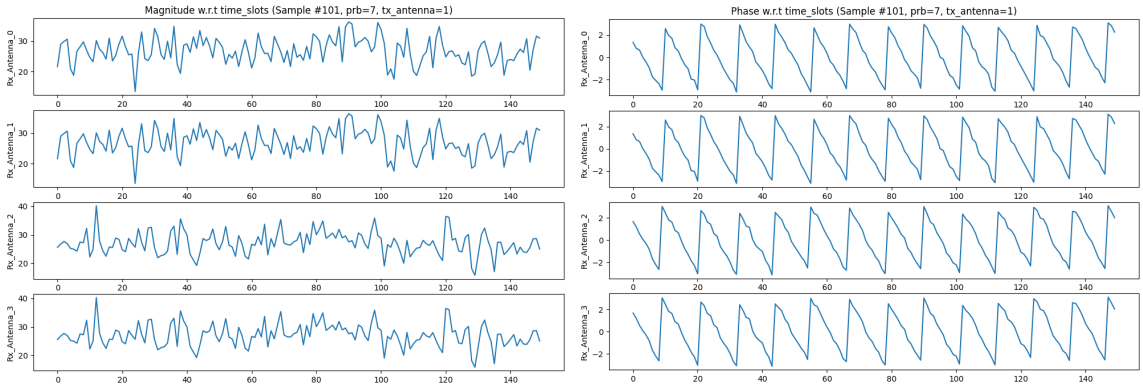
g) Sample#101, DFT h) Sample#467, DFT i) Sample#648, DFT

Figure 4.1: Visualization of Channel in Antenna Domain and Beam Domain: Randomly Selected 3 Samples

4.1.2 Channel Representation in 2D Space

Figure 4.2 and Figure 4.3 presents channel characteristics of Sample #101 in 2 dimension. Plots for the other two samples, #467 and #648, are provided in the Appendix 1 for further reference and comparison. Figure 4.2(a) shows the magnitude of the channel with respect to time slots for each of the 4 receive antennas. The magnitude varies over time, exhibiting a fluctuating pattern. This variation can be attributed to the dynamic nature of the wireless channel, influenced by factors such as multipath propagation, mobility, and environmental changes.

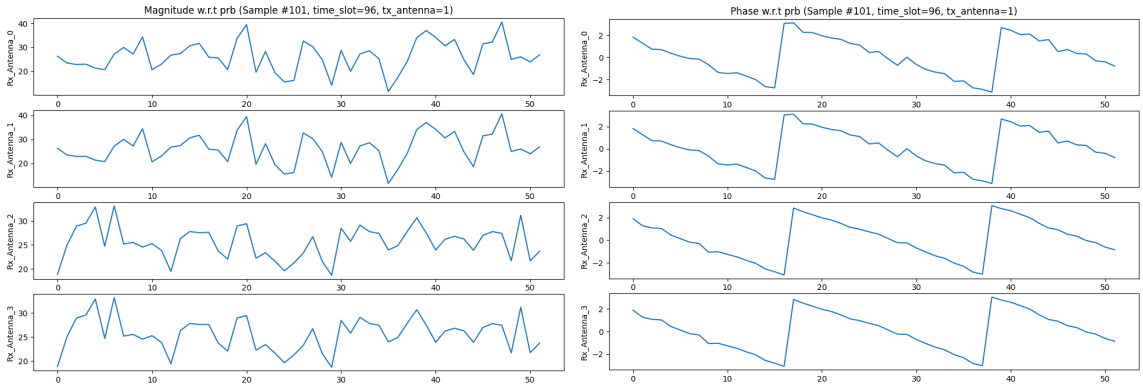
Figure 4.2(b) presents the phase of the channel with respect to time. we observe a distinct sawtooth-like pattern for each receive antenna. This pattern is a result of the phase wrapping in the range of $-\pi$ to π radians.



a) Magnitude w.r.t Time b) Phase w.r.t Time

Figure 4.2: Visualization of Channel (Sample #101): Magnitude and Phase w.r.t Time

Figure 4.3 (a)(b) shows the magnitude and phase of the channel with respect to the Physical Resource Blocks (PRBs) for each receive antenna. The magnitude of the channel varies across the PRBs, highlighting the frequency-selective nature of the channel, which means different PRBs experience different levels of attenuation.



a) Magnitude w.r.t PRB b) Phase w.r.t PRB

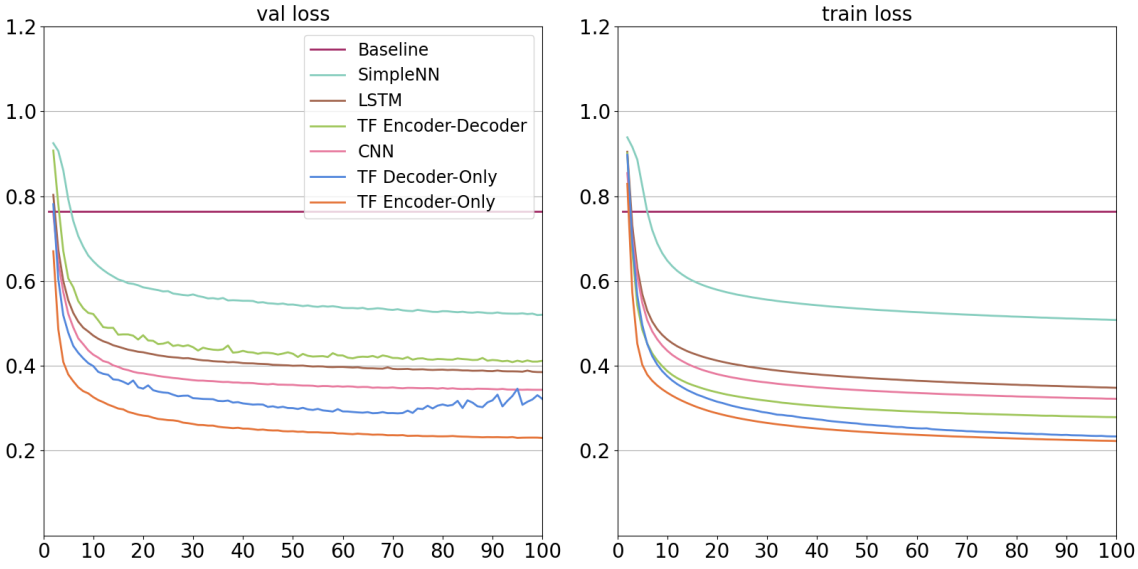
Figure 4.3: Visualization of Channel (Sample #101): Magnitude and Phase w.r.t PRB

4.2 Model Performance Metrics

4.2.1 Analysis of Training Loss & Validation Loss

Figure 4.4 presents a comparative analysis of the validation loss and training loss for various models over the training process, utilizing the average Normalized Mean Squared Error (NMSE) as the loss function. The hyperparameter K is set to 5 and P to 4 in this setting, indicating that the previous 5 slots of channel information are fed into the models to predict the future 4 slots. Additionally, the number of receive antennas is set to 1 (rx = 1), and all models are built in real domain in this section. The training dataset consists of only 60km/h data with an equi-spaced configuration of $d = 5$ without autocorrelation information, to focus on selecting the best-performing model for a single velocity scenario. It is also important to note that all models are trained and inferenced using a single GPU card - NVIDIA Geforce RTX 3090.

Figure 4.4(a) illustrates the validation loss for the different models, and Table 4.1 presents the trainable parameters, training duration, and inference latency for different AI models (for more details about the models' settings, please refer to Appendix A.1). We observe that the Transformer Encoder-Only model achieves the lowest validation loss, relatively fast training duration and inference latency, demonstrating its superior performance compared to the other models. The Transformer Decoder-Only model follows closely, exhibiting the second-best performance. The CNN-Resnet18, LSTM, Transformer Encoder-Decoder, and SimpleNN models also outperform the non-AI based (AR) model baseline in this setup.



a) Validation Loss

b) Training Loss

Figure 4.4: Comparison of Validation Loss and Training Loss for Different Models across Epochs ($K = 5$, $P = 4$, rx=1, Trained with 60km/h Channel Data)

Table 4.1: Trainable Parameters, Training Duration and Inference Latency for various AI Models

Model	Trainable Parameters	Training Time (1.5 million Samples, 100 Epochs)	Inference Time (160000 Samples)
SimpleNN	2.3 million	1h 52min	/
LSTM	2.2 million	7h 53min	/
Transformer Encoder-Decoder	2.3 million	20h 4min	/
CNN - Resnet18	11.3 million	17h 12min	/
Transformer Decoder-Only	2.3 million	6h 5min	47s
Transformer Encoder-Only	2.4 million	7h 19min	23s

In the subsequent sections, we will focus on the best-performing model, the Transformer Encoder-Only, and explore various methods to enhance its generalization ability. By leveraging techniques such as different embedding techniques, data augmentation, and incorporate side information, we aim to improve the model's performance and robustness across different velocity scenarios.

4.3 Comparative Analysis of Inference Performance

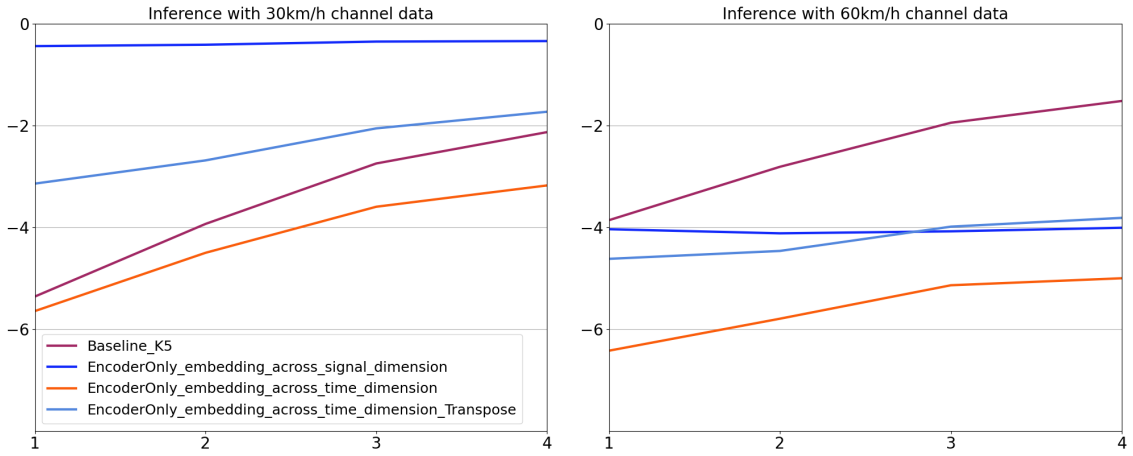
4.3.1 Temporal vs. Signal Feature Embedding Efficacy

To enhance the model's generalization ability across multiple UE velocities, different methods of embedding were tested on the transformer encoder-only model's performance, as illustrated in Figure 4.5. The model's performance is measured by the Normalized Mean Squared Error (NMSE) in decibels (dB) versus the predicted slot. The model is trained exclusively with 60km/h channel data, and the plots compare its inference performance when tested with both 30km/h and 60km/h channel data.

Figure 4.5(b) presents the inference results with 60km/h channel data, which is the same velocity used during training. All embedding methods show improved performance compared to the baseline. However, the encoder-only model with embedding across the time dimension significantly outperforms the other two embedding approaches.

In Figure 4.5(a), the inference results with 30km/h channel data are shown. This velocity is not included in the training dataset and serves as a good metric to assess the model's generalization ability. The encoder-only model with embedding across the time dimension still surpasses the performance of the other methods, including the baseline. However, the embedding across the signal dimension and embedding across the time dimension with transpose perform worse than the baseline, indicating that these two models have limited generalization capability.

These results highlight the importance of selecting an appropriate embedding method for the transformer encoder-only model. Embedding across the time dimension consistently outperforms other methods.



a) Inference with 30km/h

b) Inference with 60km/h

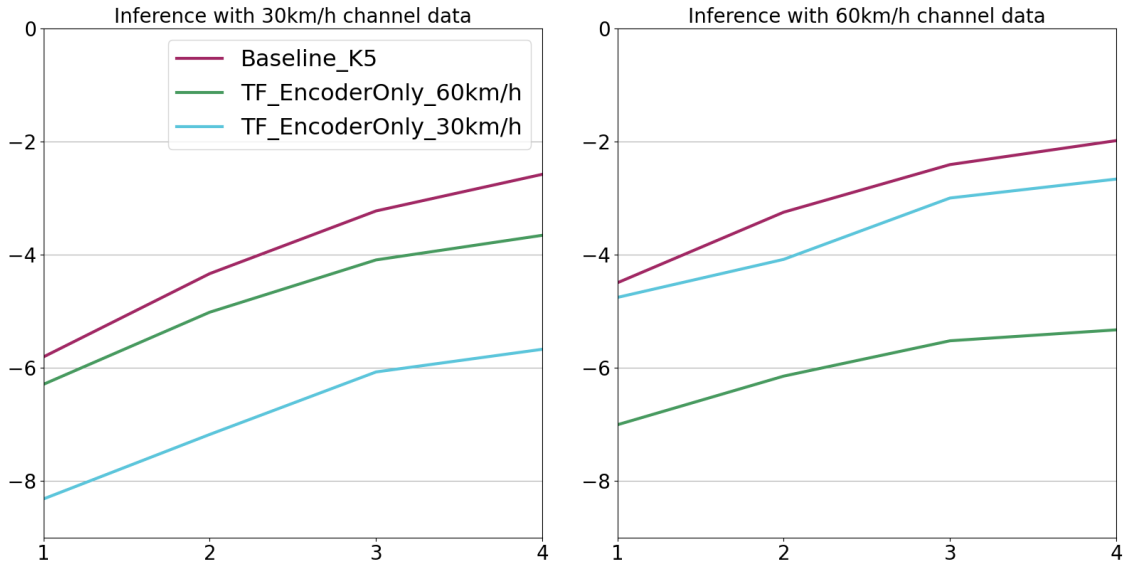
Figure 4.5: Comparison of NMSE(dB) vs. Predicted Slot for Different Embedding Methods on Transformer Encoder-Only Model ($K = 5$, $P = 4$, rx=1, Trained with 60km/h Channel Data)

4.3.2 Performance on Variable Speed Datasets

4.3.2.1 Dataset at 60 km/h vs. 30 km/h

Figure 4.6 presents a comparison of the Normalized Mean Squared Error (NMSE) in decibels (dB) versus the predicted slot for the same Encoder-Only model (embedding across the time dimension, as described in the previous section) but trained with different channel data velocities: 30km/h and 60km/h. It is evident that the model trained with 30km/h data exhibits better performance when inferring on 30km/h data, and similarly, the model trained with 60km/h channel data performs better when inferring on 60km/h data. This observation suggests that the model learns to recognize specific patterns associated with a particular velocity.

It is also worth noting that even when the transformer encoder-only model is trained with a different velocity than the inference data, it still outperforms the baseline model in both scenarios. This suggests that the transformer architecture has a higher capacity to capture the underlying patterns in the channel data compared to the baseline approach.



a) Inference with 30km/h

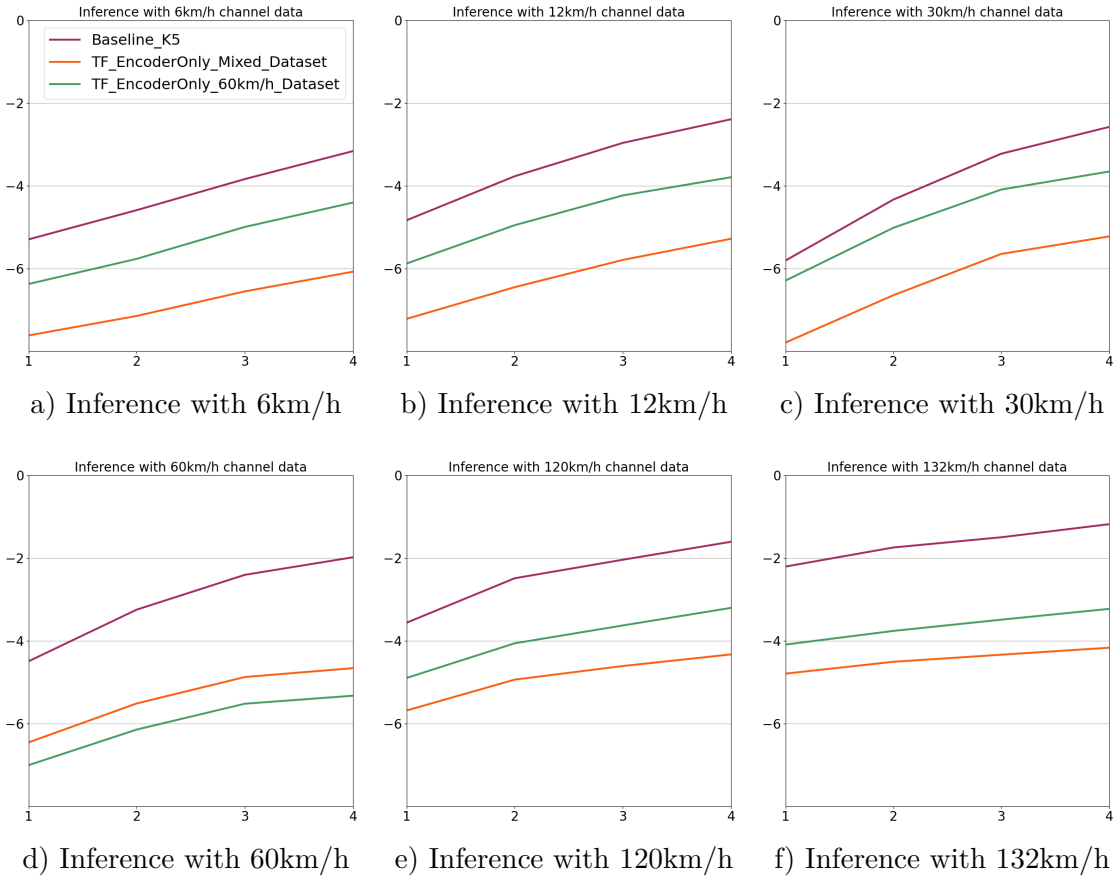
b) Inference with 60km/h

Figure 4.6: Comparison of NMSE(dB) vs. Predicted Slot for Training with Different Velocities (30km/h vs. 60km/h) on Transformer Encoder-Only Model ($K = 5$, $P = 4$, rx=1)

4.3.2.2 Fixed Speed vs. Mixed Speed Datasets

Figure 4.7 extends the analysis from the previous section, where the model was trained with a single velocity dataset, to training with multi-velocity data (referred to as mixed dataset). The mixed dataset is a synthesized dataset containing velocities ranging from 6km/h to 192km/h, as described in Table 3.2, generated using data augmentation techniques.

Across all inference velocities, the transformer encoder-only model trained with the mixed dataset consistently outperforms both the baseline model and the transformer encoder-only model trained with the 60km/h dataset (except for inference with 60km/h). It is important to note that the training dataset does not contain 12km/h and 120km/h channel data; however, the model still demonstrates good generalization ability on these two velocities. Overall, it is evident that the model trained with a mixed dataset significantly improves its generalization ability. This approach enables the model and capture the underlying patterns in the channel data across various velocity conditions.



a) Inference with 6km/h b) Inference with 12km/h c) Inference with 30km/h

d) Inference with 60km/h e) Inference with 120km/h f) Inference with 132km/h

Figure 4.7: NMSE(dB) vs. Predicted Slot: Impact of Training with Mixed Dataset on Model Performance ($K = 5$, $P = 4$, rx=1)

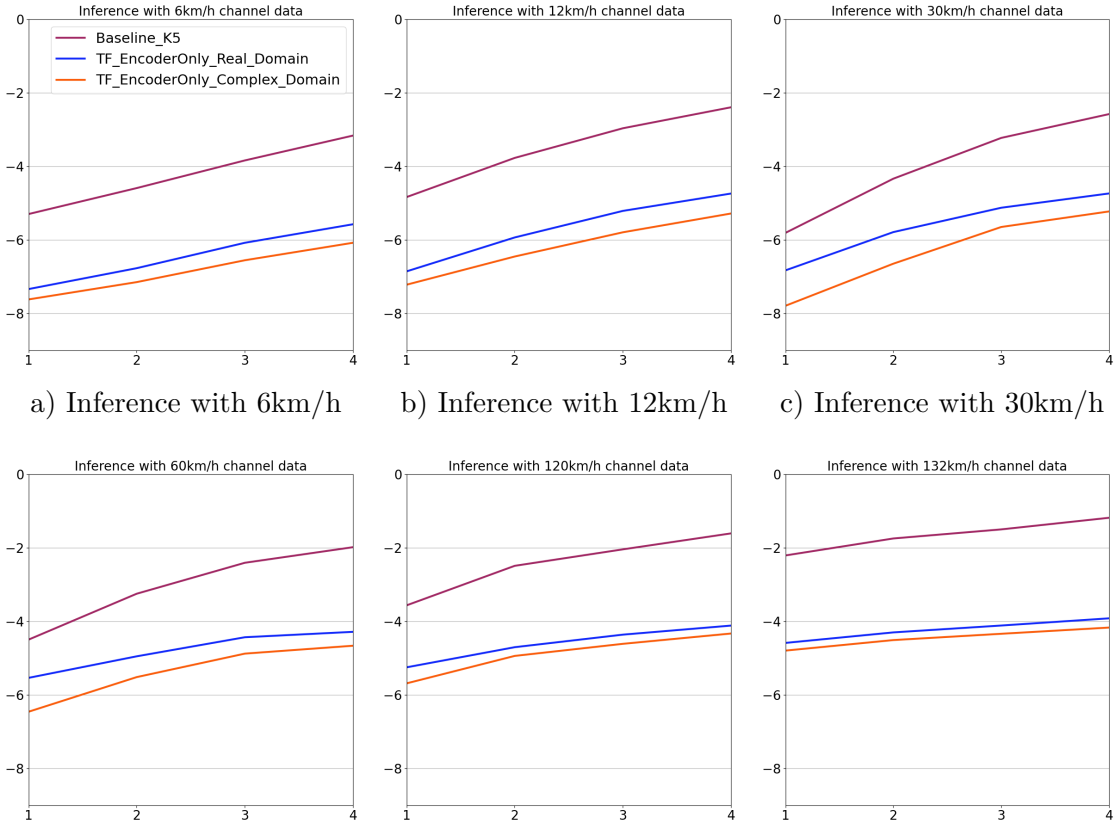
4.3.3 Domain-Specific Model Performance

4.3.3.1 Complex vs. Real Domain Model Comparison

Figure 4.8 presents a comparison of the Normalized Mean Squared Error (NMSE) in decibels (dB) versus the predicted slot for Encoder-Only model built on real domain and on the complex domain. Across all inference velocities, both models exhibit similar behavior and outperform the baseline. However, the Encoder-Only model built on the complex domain consistently surpasses the performance of the model built on the real domain. This observation suggests that the complex domain model is more effective in capturing the characteristics of the channel data. The parameters used for training both models can be found in Table 4.2. Note that although the model built in the complex domain has fewer trainable parameters compared to the model in the real domain (0.5 million vs. 2.4 million), however, each weight in the complex domain model is a complex number, which introduces additional computational complexity compared to real numbers.

Table 4.2: Configuration, Training Duration and Inference Latency for Model Training: Real Domain vs. Complex Domain

Transformer Encoder-Only Model	**Real Domain**	**Complex Domain**
Trainable Parameters	2.4 million	0.5 million
Input size	64	32
Embedding Dimension	64	32
Number of Attention Heads	8	8
Feedforward Hidden Layer Size	2048	2048
Number of Encoder Block	5	3
Dropout Rate	0.01	0.01
Training Time (3.5 million Samples, 100 Epochs)	23h 30min	1d 3h 46min
Inference Time (560000 Samples)	1min 11s	1min 50s



a) Inference with 6km/h b) Inference with 12km/h c) Inference with 30km/h

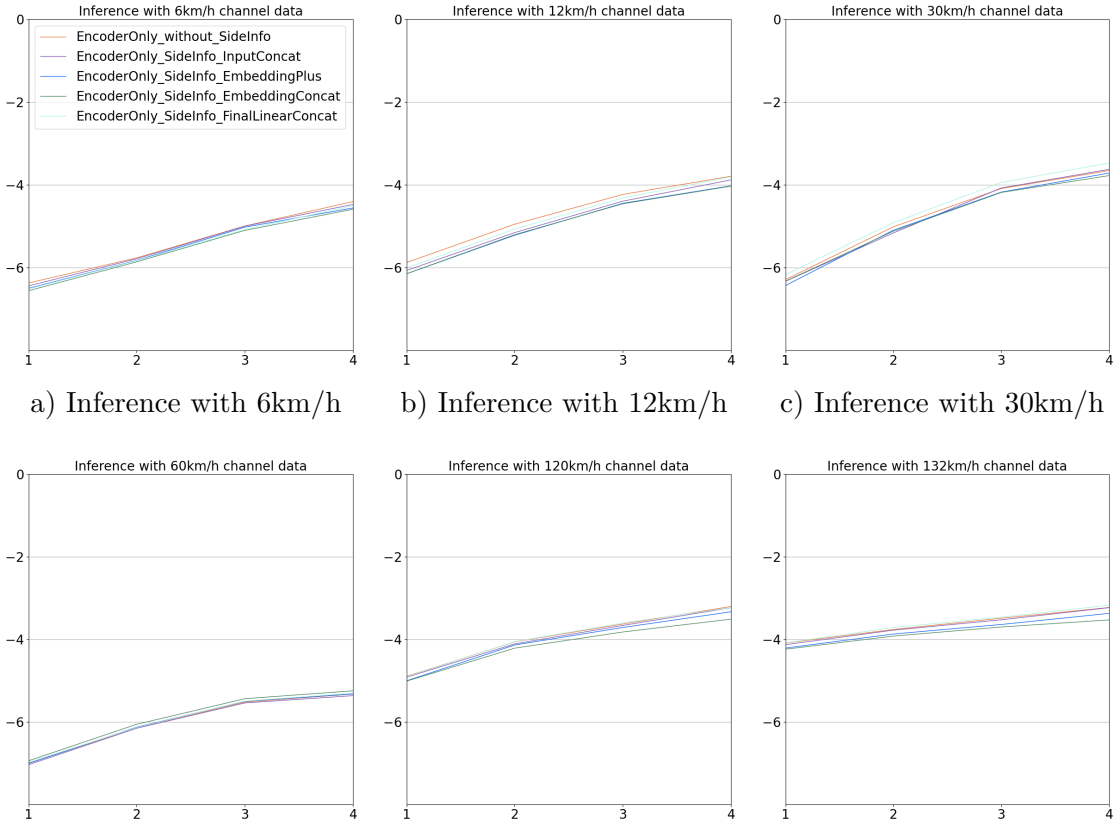
d) Inference with 60km/h e) Inference with 120km/h f) Inference with 132km/h

Figure 4.8: NMSE(dB) vs. Predicted Slot: Comparison of Encoder-Only Models Built on Real Domain and Complex Domain ($K = 5$, $P = 4$, rx=1)

4.3.4 Impact of Autocorrelation on Model Accuracy

Figure 4.9 illustrates the impact of adding side information - autocorrelation, into the model training process, as described in Chapter 3, Section 3.6.2. Four distinct

methods of incorporating autocorrelation are explored: (1) concatenate autocorrelation with the input, (2) concatenate autocorrelation in the embedding layer, (3) add autocorrelation in the embedding layer, and (4) concatenate autocorrelation in the last linear layer. However, as evident from the figure, the impact of these four methods is very limited, with only a marginal improvement in the overall performance. This might be because the autocorrelation is not informative enough to significantly enhance the model's prediction, or the model has already learned to capture the relevant patterns without explicitly being fed the autocorrelation.



a) Inference with 6km/h b) Inference with 12km/h c) Inference with 30km/h

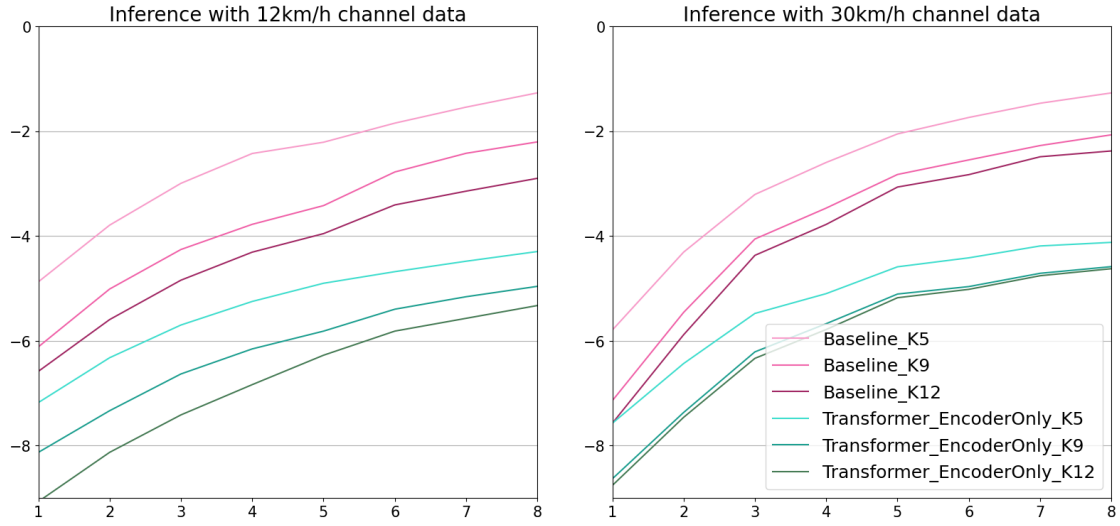
d) Inference with 60km/h e) Inference with 120km/h f) Inference with 132km/h

Figure 4.9: NMSE(dB) vs. Predicted Slot: Impact of Adding Side Information-Autocorrelation on Model's Performance ($K = 5$, $P = 4$, rx=1, Trained with 60km/h Channel Data)

4.3.5 Adaptive Hyperparameters K and P

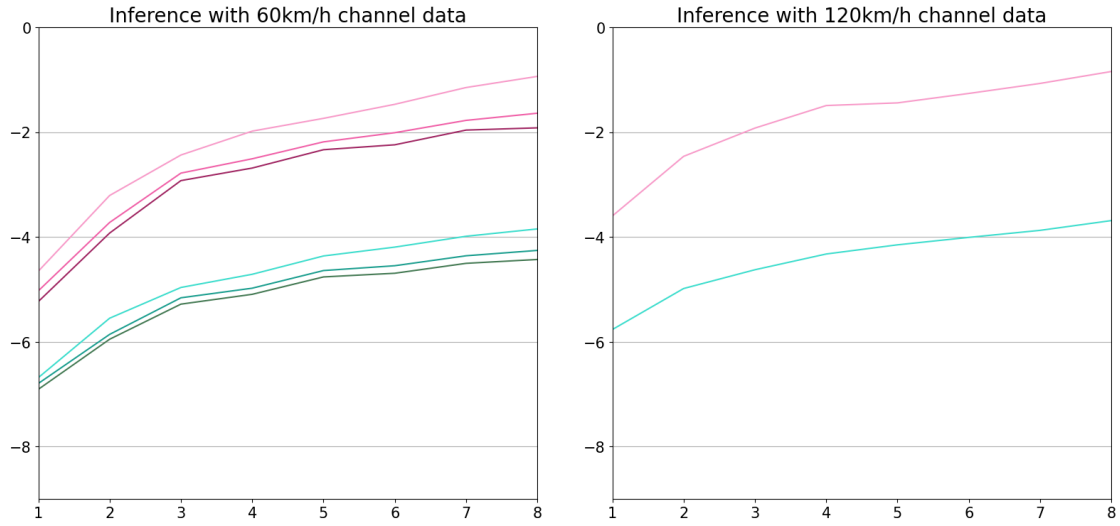
Figure 4.10 illustrates the results of enhancing model's generalization ability by making number of measurements K and number of prediction P adaptive. As described in Chapter 3 section 3.6 and 3.7, the Encoder-Only model is trained with varying value of K, ranging from 5 to 12 (excluding $K = 9$ in the training dataset to better assess its generalization ability during inference). The input fed into the model is padded to maximum length of $K_{\text{MAX}} = 12$, the number of prediction P is fixed to 4 during training, however, when inference, we extend P to 8, refer to Figure 3.10.

As evident from Figure 4.10, the Encoder-Only model outperforms the baseline when inferring across all velocities. As number of measurements K increases, the performance of both the baseline and the Transformer Encoder-Only model improves. This improvement can be attributed to the availability of more information about the channel conditions, enabling the model to achieve higher performance.



a) Inference with 12km/h

b) Inference with 30km/h



c) Inference with 60km/h

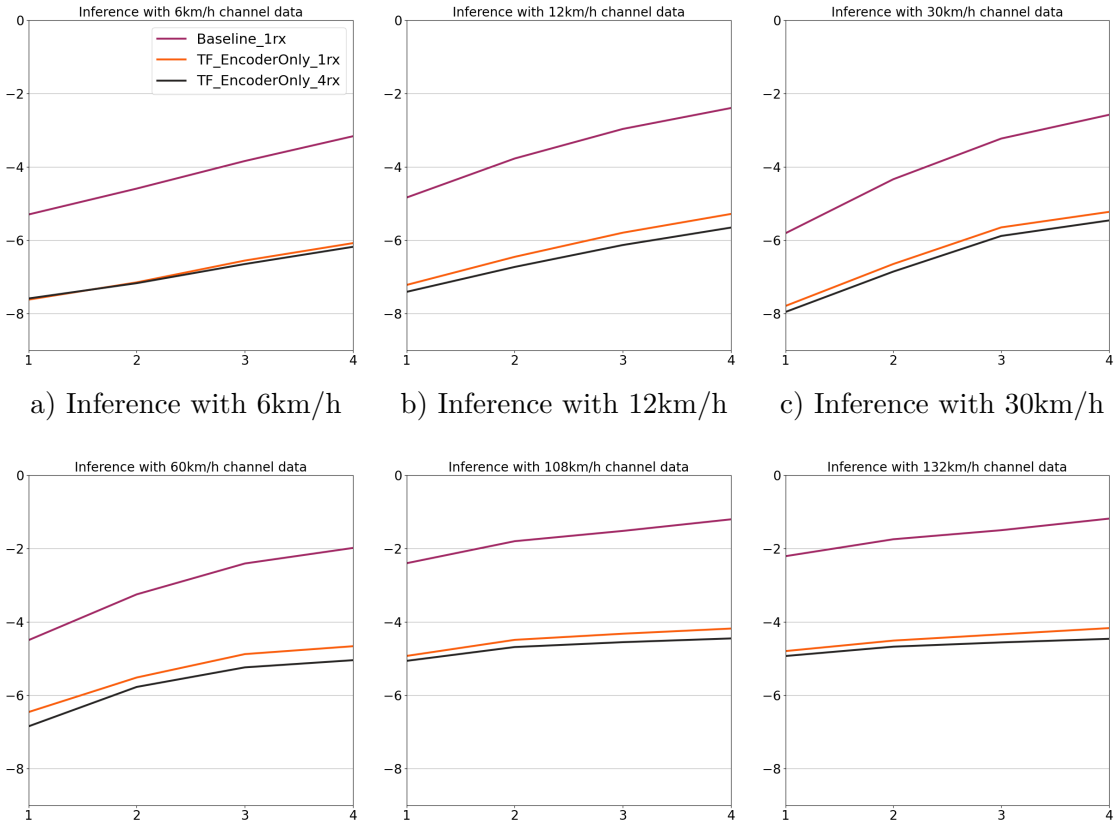
d) Inference with 120km/h

Figure 4.10: NMSE(dB) vs. Predicted Slot: Encoder-Only Model with Adaptive Value of K and P (rx=1, Trained with Mixed Channel Data)

4.3.6 Single vs. Multiple Receive Antennae Performance

Figure 4.11 illustrates the impact of predicting multiple receive antennas simultaneously compared to predicting a single receive antenna individually. Both models

were trained using the same architecture - the Transformer Encoder-Only model in the complex domain. The results demonstrate that the model trained to predict 4 receive antennas outperforms the model predicting a single receive antenna. This improvement in performance can be attributed to the model's ability to recognize and exploit the patterns and correlations between the received signals at different antennas. These patterns provide additional information about the channel, leading to enhanced prediction accuracy.



a) Inference with 6km/h b) Inference with 12km/h c) Inference with 30km/h

d) Inference with 60km/h e) Inference with 108km/h f) Inference with 132km/h

Figure 4.11: NMSE(dB) vs. Predicted Slot: Performance Comparison of Single and Multiple Receive Antenna Prediction ($K = 5$, $P = 4$, Trained with Mixed Dataset)

4.4 Summary of Key Findings

Based on the discussions presented in this chapter, we have obtained the following key findings regarding the use of AI for channel prediction:

1. The Transformer Encoder-Only model achieves the highest performance among all the models explored. This architecture demonstrates superior ability in capturing and learning the complex patterns present in wireless channel data.

2. For the Transformer Encoder-Only model, embedding across the time dimension produces better performance compared to other embedding approaches, allowing the model to learn more informative representations of the channel data.

3. Training the model with a mixed dataset significantly improves its generalization ability, enabling it to perform well on unseen data.

4. The Transformer Encoder-Only model built on the complex domain is more effective in capturing the patterns of the wireless channel compared to models operating in the real domain.

5. The impact of adding autocorrelation as side information has a very limited improvement in the model's performance.

6. It is feasible to train a model with an adaptive number of measurements and predictions that outperforms the baseline. By allowing the model to dynamically adjust the number of input measurements and output predictions, it can better adapt to varying channel dynamics.

7. Training a model to predict multiple antennas simultaneously provides performance gains compared to predicting each receive antenna individually.

5
Conclusion

In this thesis, we have explored the application of AI-based methods for channel prediction in SU-MIMO systems. First, the Transformer Encoder-Only model achieved the top performance, outperforming its counterparts. Moreover, when the model was trained with a diverse mixed velocity dataset, it exhibited enhanced generalization capabilities, suggesting that such a strategy is critical for robust performance.

Furthermore, the thesis has revealed that embedding across the time dimension within the Transformer Encoder-Only model yields superior results compared to alternative embedding methods. Additionally, constructing the model within the complex domain has proven to be more effective in capturing the patterns of the wireless channel compared to models built in the real domain, although training and inference in the complex domain are more computationally costly. The incorporation of autocorrelation as side information in model training process offered marginal performance improvements, suggesting that exploring other methods of adding side information might be needed to leverage this domain knowledge. The feasibility of training models with an adaptive number of measurements and predictions has been established, highlighting the model's generalization ability. Lastly, training a model to simultaneously predict multiple receive antennas presents performance gains compared with predicting each receive antenna individually, however, such model may need to be re-trained for UEs with different number of antennas.

5.1 Future Work

One limitation of the thesis is that it does not consider the complexity of the proposed models. This is important when implementing these models on User Equipment (UE) with limited computing power. The models are more accurate but could be too complex for real-time use on devices with limited hardware capabilities. This opens up a path for future research, which could aim to decrease the complexity of the model while keep the accuracy. These could include network pruning, quantization or low-rank factorization techniques.

Future research could also explore more architectures beyond those in this thesis. Emerging architectures like Mamba [33] may offer a better balance between performance and computation cost. Another promising direction is to apply transfer learning, fine tuning pre-trained models in other domains like language processing, computer vision or audio. These models may have learned how to catch patterns in

channel-like data. Additionally, the dataset used for training and inference in this thesis was generated solely by the Urban Macro (UMa) channel model. To develop a more generalized model, future research could explore mixing datasets from different channel models. Finally, there is potential in further exploring how side information can be used to improve model predictions. These directions could all make AI-based channel prediction more practical and effective for next-generation wireless networks.

Bibliography

[1] Xuanfan Shen, Guodong Sun, Xuewu Dai, and Shaohua Wan. Ekf/ukf-based channel estimation for robust and reliable communications in v2v and iiot. *EURASIP Journal on Wireless Communications and Networking*, 2019, 06 2019.

[2] Ericsson AB. Ericsson Mobility Report November 2023 — ericsson.com. `https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2023`, 2023. [Accessed 17-05-2024].

[3] Wikipedia contributors. Channel state information — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Channel_state_information&oldid=1193736602`, 2024. [Online; accessed 16-May-2024].

[4] 3GPP ETSI TS 138 214 v17.1.0. 5G; NR; Physical layer procedures for data, 2022. [Accessed 16-05-2024].

[5] Venkatesh Ramireddy, Marcus Grossmann, Markus Landmann, and Giovanni Del Galdo. Enhancements on type-ii 5g new radio codebooks for ue mobility scenarios. *IEEE Communications Standards Magazine*, 6(1):35–40, 2022.

[6] Maya Kallas, Paul Honeine, Cédric Richard, Francis Clovis, and Hassan Amoud. Prediction of time series using yule-walker equations with kernels. 03 2012.

[7] Kazys Kazlauskas. The burg algorithm with extrapolation for improving the frequency estimation. *Informatica*, 22(2):177–188, 2011.

[8] Steven M Kay. *Modern spectral estimation : Theory and application.* Prentice Hall, n.d.

[9] S. Hu, H. Hallen, and A. Duel-Hallen. Physical channel modeling, adaptive prediction and transmitter diversity for flat fading mobile channel. In *1999 2nd IEEE Workshop on Signal Processing Advances in Wireless Communications (Cat. No.99EX304)*, pages 387–390, 1999.

[10] Rikke Apelfröjd. Channel estimation and prediction for 5g applications. *Uppsala University*, 2018.

[11] Daniel Aronsson. Channel estimation and prediction for mimo ofdm systems: Key design and performance aspects of kalman-based algorithms. *Uppsala University*, 2011.

[12] Hwanjin Kim, Sucheol Kim, Hyeongtaek Lee, Chulhee Jang, Yongyun Choi, and Junil Choi. Massive mimo channel prediction: Kalman filtering vs. machine learning. *IEEE Transactions on Communications*, 69(1):518–528, 2021.

[13] Hwanjin Kim, Sucheol Kim, Hyeongtaek Lee, Chulhee Jang, Yongyun Choi, and Junil Choi. Massive mimo channel prediction: Kalman filtering vs. machine learning. *IEEE Transactions on Communications*, 69(1):518–528, 2021.

[14] Wei Jiang and Hans D. Schotten. Neural network-based fading channel prediction: A comprehensive overview. *IEEE Access*, 7:118112–118124, 2019.

[15] Daoud Burghal, Yang Li, Pranav Madadi, Yeqing Hu, Jeongho Jeon, Joonyoung Cho, Andreas F. Molisch, and Jianzhong Zhang. Enhanced ai-based csi prediction solutions for massive mimo in 5g and 6g systems. *IEEE Access*, 11:117810–117825, 2023.

[16] Sharan Mourya, Pavan Reddy, SaiDhiraj Amuru, and Kiran Kumar Kuchi. Spectral temporal graph neural network for massive mimo csi prediction. *IEEE Wireless Communications Letters*, pages 1–1, 2024.

[17] ShareTechnote. 5G | ShareTechnote — sharetechnote.com. `https://www.sharetechnote.com/html/5G/5G_CSI_Report.html`, 2023. [Accessed 17-05-2024].

[18] Enzo Grossi and Massimo Buscema. Introduction to artificial neural networks. *European journal of gastroenterology hepatology*, 19:1046–54, 01 2008.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Artemios-Anargyros Semenoglou, Evangelos Spiliotis, and Vassilis Assimakopoulos. Image-based time series forecasting: A deep convolutional neural network approach. *Neural Networks*, 157, 10 2022.

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

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[23] Robin M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 2019.

[24] Introduction to Recurrent Neural Network - GeeksforGeeks. `https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/`. [Accessed 17-05-2024].

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[26] Wikipedia contributors. Long short-term memory — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=1220425877`, 2024. [Online; accessed 17-May-2024].

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[28] Sebastian Raschka. Understanding and Coding the Self-Attention Mechanism of Large Language Models From Scratch — sebastianraschka.com. `https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html`. [Accessed 17-05-2024].

[29] Muqiao Yang, Martin Q. Ma, Dongyu Li, Yao-Hung Hubert Tsai, and Ruslan Salakhutdinov. Complex transformer: A framework for modeling complex-valued sequence. *CoRR*, abs/1910.10202, 2019.

[30] Florian Eilers and Xiaoyi Jiang. Building blocks for a complex-valued transformer architecture. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, June 2023.

[31] 3GPP ETSI TR 138 901 v16.1.0. 5G; Study on channel model for frequencies from 0.5 to 100 ghz, 2020. [Accessed 16-05-2024].

[32] Changwei Lv, Jia-Chin Lin, and Zhaocheng Yang. Channel prediction for millimeter wave mimo-ofdm communications in rapidly time-varying frequency-selective fading channels. *IEEE Access*, 7:15183–15195, 2019.

[33] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

A

Appendix 1

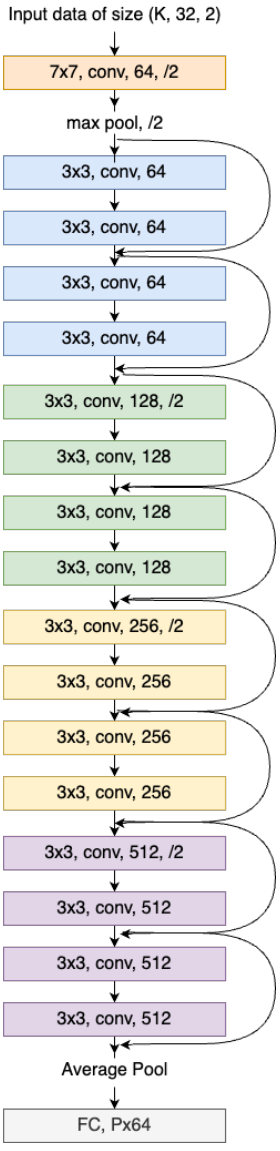
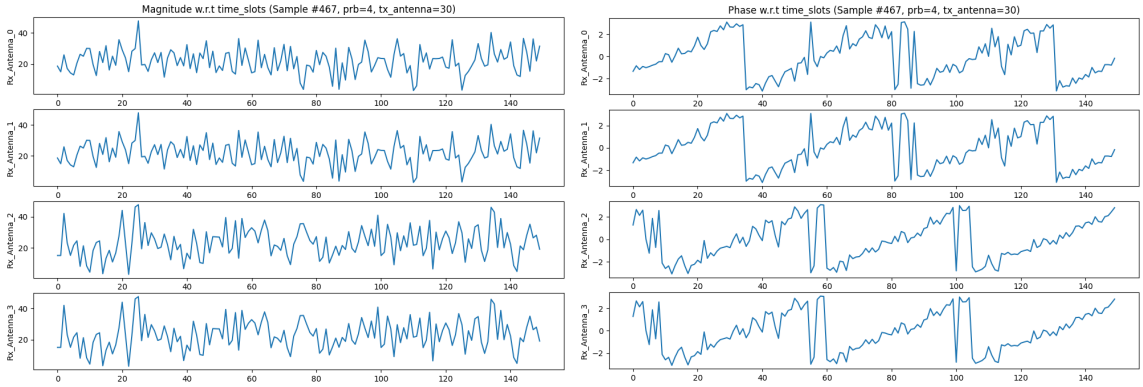
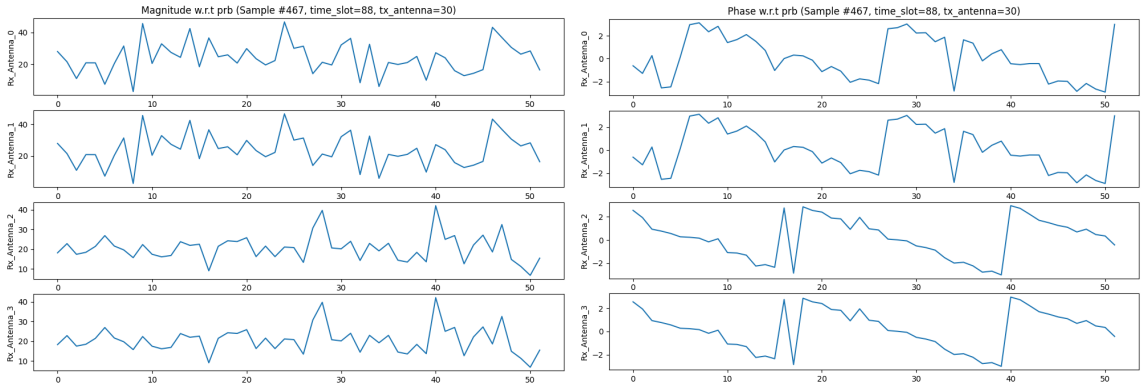


Figure A.1: ResNet18 Architecture (Modified for Channel Prediction)



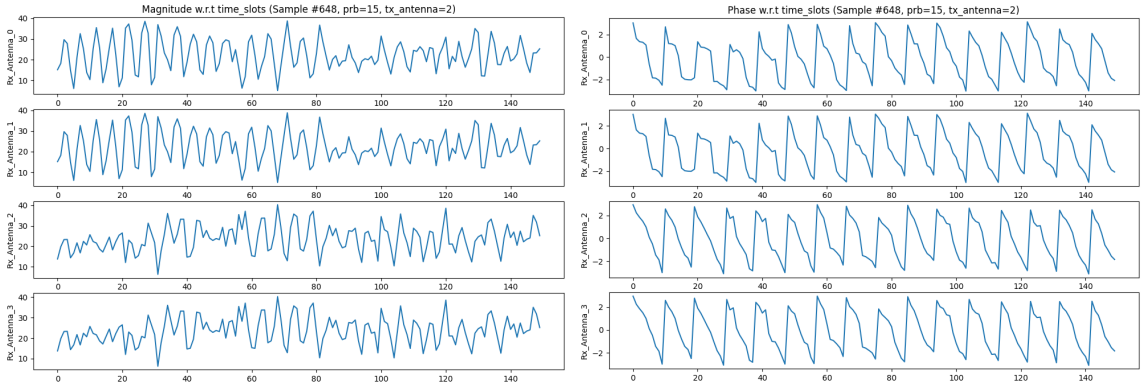
a) Magnitude w.r.t Time b) Phase w.r.t Time

Figure A.2: Visualization of Channel (Sample #467): Magnitude and Phase w.r.t Time



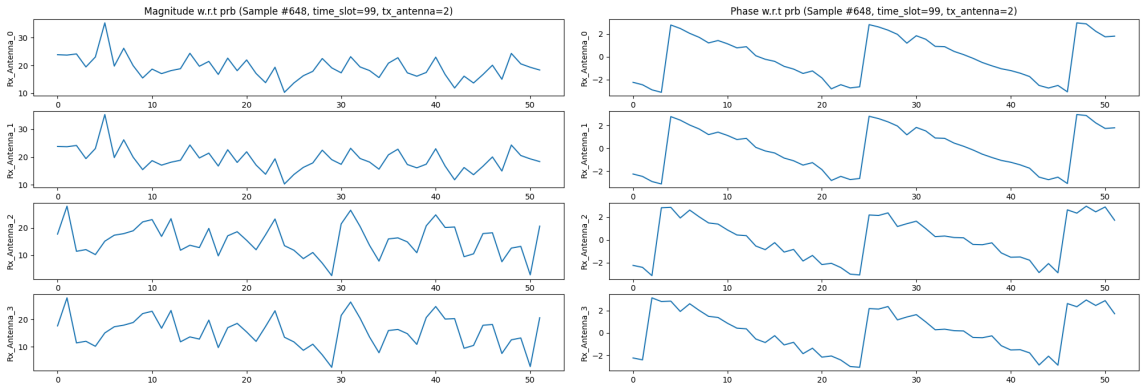
a) Magnitude w.r.t PRB b) Phase w.r.t PRB

Figure A.3: Visualization of Channel (Sample #467): Magnitude and Phase w.r.t PRB



a) Magnitude w.r.t Time b) Phase w.r.t Time

Figure A.4: Visualization of Channel (Sample #648): Magnitude and Phase w.r.t Time



a) Magnitude w.r.t PRB b) Phase w.r.t PRB

Figure A.5: Visualization of Channel (Sample #648): Magnitude and Phase w.r.t PRB

Table A.1: Parameters Settings for Different Architectures

Model	**SimpleNN**	**LSTM**	**CNN Resnet18**	**Transformer Encoder-Decoder**	**Transformer EncoderOnly**	**Transformer DecoderOnly**
Trainable Parameters	2.3m	2.2m	11.3m	2.3m	2.4m	2.3m
Dropout Rate	-	-	-	-	0.01	0.01
Input Size	320	-	-	64	64	64
Embedding Dimension	-	-	-	64	64	128
Number of Attention Heads	-	-	-	8	8	8
Number of Encoder Block	-	-	-	4	-	-
Number of Decoder Block	-	-	-	4	5	4
Number of Layers	2	4	18	-	-	-
Feedforward Size	-	-	-	2048	2048	2048
Hidden Size	4096	256	-	-	-	-
Learning Rate	1e-4	1e-3	1e-4	1e-4	1e-4	1e-4
Optimizer	Adam					

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY