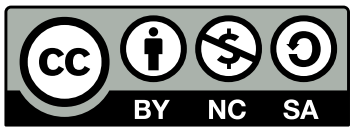
Aalto University
School of Electrical
Engineering

Master's Programme in Computer, Communication and Information Sciences

Self-attention Cell-Free MIMO

Andrei Palshin

Master's Thesis
2025



Aalto University
School of Electrical
Engineering

Author Andrei Palshin

Title Self-attention Cell-Free MIMO

Degree programme Computer, Communication and Information Sciences

Major Communications Engineering

Supervisor Prof. Sergiy A. Vorobyov

Advisor Dr. Atchutaram K. Kocharlakota

Date 17 November 2025 **Number of pages** 57 **Language** English

Abstract

In modern wireless communication, rising traffic, tighter latency budgets, and increasing network complexity place additional demands on the scalability and responsiveness of resource allocation and control across distributed networks. These demands motivate attention-based deep learning for wireless resource optimization. This thesis explores the use of a transformer-based neural network for power control in cell-free massive MIMO (CFmMIMO) systems as a case demonstration of how attention-based models can be used in wireless system optimization. In this setting, self-attention efficiently captures global dependencies among users by conditioning on large-scale fading to access points and on pilot-allocation information. The model is trained in an unsupervised manner on a high-performance computing cluster. The training data consist of simulated network snapshots represented by the large-scale fading matrix and the pilot-allocation matrix. Experiments indicate per-user spectral efficiency comparable to established optimization methods, while guaranteeing a significantly lower inference time. Results are consistent across the evaluated configurations. These findings support attention-based models as a practical and scalable solution for real-time resource optimization in CFmMIMO.

Keywords Machine Learning, Wireless Communications, Self-Attention, Transformer Neural Networks, Cell-Free Massive MIMO, Power Control

Preface

I would like to thank my supervisor, Professor Sergiy A. Vorobyov, for his guidance, patience, and steady support throughout the thesis process. His clear advice and careful feedback helped shape the direction of this work.

I am also grateful to my advisor, Dr. Atchutaram K. Kocharlakota, for his guidance on the wireless communication aspects of this work and for many helpful discussions during the project. His explanations and practical insights were essential for the simulation work.

Finally, I would like to thank my family, my friends, and everyone else who supported me during the work on this thesis.

Otaniemi, 17 November 2025

Andrei Palshin

Contents

Symbols and abbreviations

Symbols

$\mathbb{R}$	Set of real numbers
$\mathbf{x}$	Input feature vector
y	Target output
$\hat{y}$	Predicted output of the model
$\boldsymbol{\theta}$	Vector of all trainable model parameters
N	Number of training examples
$\mathcal{D}_{\text{train}}$	Training dataset
$\mathcal{D}_{\text{val}}$	Validation dataset
$\mathcal{D}_{\text{test}}$	Test dataset
$L(\boldsymbol{\theta})$	Loss function
η_t	Learning rate at optimization step t
B_t	Index set of samples
$\mathbf{Q}$	Matrix of query vectors
$\mathbf{K}$	Matrix of key vectors
$\mathbf{V}$	Matrix of value vectors
d_{model}	Model width (embedding dimension)
d_k	Dimension of query and key vectors
H	Number of attention Heads in multi-head attention
Head_i	ith attention Head in multi-head attention
$\boldsymbol{\Phi}$	Pilot-allocation matrix
M	Number of APs
K	Number of users
η_{mk}	Power-control coefficient from AP m to user k
$\boldsymbol{\eta}$	Vector collecting all power-control coefficients
ρ_d	Maximum downlink transmit power per AP

τ_p	Pilot length (number of pilot symbols)
τ_c	Coherence interval length
$\gamma_k(\boldsymbol{\eta})$	Effective downlink SINR of user k
$\mathrm{SE}_k(\boldsymbol{\eta})$	Spectral efficiency of user k

Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
ANN	Artificial Neural Network
DNN	Deep Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
MIMO	Multiple-Input Multiple-Output
CFmMIMO	Cell-Free Massive Multiple-Input Multiple-Output
AP	Access Point
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HPC	High-Performance Computing
TDD	Time-Division Duplex
SINR	Signal-to-Interference-plus-Noise Ratio
SE	Spectral Efficiency
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent
MHA	Multi-Head Attention
ReLU	Rectified Linear Unit
EPA	Equal Power Allocation
APG	Accelerated Projected Gradient

1 Introduction

Development of sixth-generation (6G) telecommunication networks introduces a wide range of technical challenges across all levels of the infrastructure [1]. More efficient solutions are required at every level of network infrastructure, from low-level energy-saving techniques to high-level architectures capable of serving hundreds of thousands of users in real time. To solve these efficiency challenges, classical algorithmic and statistical methods have been widely used in communication systems. These methods often rely on simplified models and assumptions to make the calculations easier [2]. However, as the system becomes more complex, such approaches tend to show their limitations in flexibility and do not scale well to real-world conditions [3, 4]. In many cases, these methods ignore important details to maintain computational efficiency.

One potential solution to addressing these challenges is the application of Machine Learning (ML) [5]. Today, ML methods have been applied in many areas of communication systems as these methods can achieve high accuracy in representing complex behavior without manually defining each rule.

One of the most important developments was the introduction of the attention mechanism [6]. The mechanism's ability to focus on the most relevant parts of the input data has led to the development of more efficient learning strategies and models. The attention mechanism can be applied in cell-free massive multiple-input multiple-output (CFmMIMO) systems. CFmMIMO is a wireless system where many distributed access points (APs) work together to serve all users in the area instead of splitting the

network into cells [7]. One of the key components in such systems is power control [8]. One simple strategy for power control involves allocating power equally across users or APs, but this strategy cannot flexibly adapt to changing network conditions. Better performance can be achieved with more advanced optimization-based algorithms, but they often require significantly more computational time, which is limited in modern real-time systems.

One promising direction for efficient power control is the use of transformer-based neural networks. These networks use the attention mechanism and may offer efficient power control strategies directly from data [9]. This thesis explores the use of a transformer-based neural network for power control in CFmMIMO systems as a case demonstration for attention-based models in wireless system optimization. In particular, it targets problems that would otherwise become bottlenecks in terms of computational complexity. Indeed, scalability is the main challenge in CFmMIMO. The work focuses on training models on simulated data and evaluating their performance. Results are compared with traditional methods to understand the advantages and limitations of this approach. All experiments are run using the Triton computing cluster at Aalto University.

This thesis is structured as follows. Chapter 2 introduces the fundamental concepts of ML. Chapter 3 presents the transformer architecture and the attention mechanism as its key component of a proposed design. Chapter 4 gives an overview of CFmMIMO systems, describing the system model, main challenges, and motivation for applying ML-based solutions. Chapter 5 discusses the use of attention-based models in CFmMIMO, presenting the transformer implementation and training setup, and explaining the obtained results. Finally, Chapter 6 concludes the thesis and suggests directions for future work.

2 Machine Learning

ML is a subfield of artificial intelligence that focuses on building systems capable of learning from data and making predictions or decisions without being manually programmed for an exact task. Instead of following explicit algorithmic rules, an ML model improves its performance by training on example data. Over the past decades, ML techniques have achieved significant success in different fields ranging from image and speech recognition to medical diagnosis and financial forecasting [10]. The core idea is that if we provide the model with enough well-prepared data, it can infer underlying relationships and generalize them to new, unseen examples to make accurate predictions.

There are different approaches to training in ML, the most common are:

Supervised learning is when a model is trained on labeled examples, where the correct answers are already labeled. The goal is to learn a mapping from inputs to outputs that also works for data that were not seen by the model.

Unsupervised learning deals with data that has no labels. The model tries to discover hidden structures, patterns, or groupings in the data without being told the right answers.

Reinforcement learning works in a different way. The model interacts with an environment and after every action it gets some reward or penalty. With time it learns what actions bring higher rewards and builds a strategy.

These approaches can be implemented using many different models, ranging from simple statistical methods to more complex ones. Among them, neural networks have emerged as a models family that is among the

most frequently used, forming the basis of modern deep learning.

2.1 Artificial Neural Networks

As of today, artificial neural networks (ANNs) are arguably the most popular ML models [10]. ANN is built from simple computational units called *neurons*. Each neuron receives one or more inputs, multiplies them with trainable weights, adds a bias term, and then applies a nonlinear transformation.

When many neurons are combined in parallel, they form a *layer*. A typical network consists of an input layer that receives the data, a hidden layers that transform the data into more abstract representations, and an output layer that produces the final prediction as shown in Figure 1.

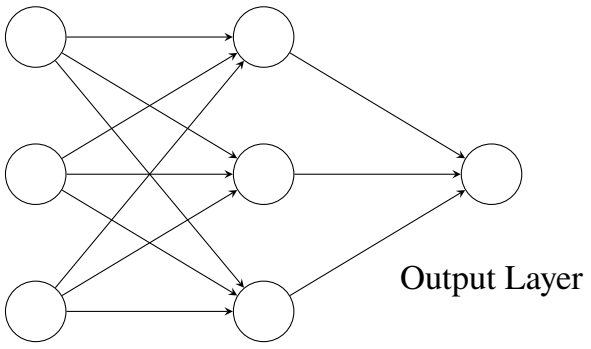


Figure 1: A simple feedforward neural network with one hidden layer.

Mathematically, the output of a single neuron can be expressed as

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{1}$$

where x_i is the ith input, w_i is the ith trainable weight, b is the bias term, and $\phi(\cdot)$ is a nonlinear activation function.

A full layer of neurons can be expressed in vector form as

$$\mathbf{h} = \phi(\mathbf{W}\mathbf{x} + \mathbf{b}),$$

where $\mathbf{x}$ is the input vector, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is the bias vector, and $\phi(\cdot)$ acts elementwise, so $\mathbf{h}$ is the output vector of the layer.

The power of ANNs comes from stacking multiple layers on top of each other. With sufficient depth and parameters, even simple architectures can approximate a wide variety of functions.

2.2 Deep Neural Networks

Deep learning refers to training ANNs with multiple hidden layers. A deep model maps an input vector to an output by repeatedly applying an affine transform followed by an elementwise nonlinearity. The depth L is the number of hidden layers.

Mathematically, we compose multiple layers in sequence. Let $\mathbf{h}^{(0)} = \mathbf{x} \in \mathbb{R}^{d_0}$. For the lth hidden layer,

$$\mathbf{h}^{(l)} = \phi^{(l)}\big(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}\big), \qquad \mathbf{W}^{(l)} \in \mathbb{R}^{d_l \times d_{l-1}},\ \mathbf{b}^{(l)} \in \mathbb{R}^{d_l},$$

for $l = 1, \ldots, L$. The network output is

$$\widehat{\mathbf{y}} = g\big(\mathbf{W}^{(L+1)}\mathbf{h}^{(L)} + \mathbf{b}^{(L+1)}\big), \qquad \mathbf{W}^{(L+1)} \in \mathbb{R}^{d_{L+1} \times d_L},\ \mathbf{b}^{(L+1)} \in \mathbb{R}^{d_{L+1}}.$$

All trainable parameters can be collected in the set $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{L+1}$.

Without nonlinear activation functions $\phi^{(l)}$ and g, the composition of these affine transformations would reduce to a single linear map, so depth alone would not increase the expressive power of the model. With sufficient depth and number of parameters, deep models can yield compact representations for certain function families compared to shallow architectures

providing similar accuracy [11].

2.2.1 Activation Functions

In the previous subsections, the neuron output was written in the form (1), where ϕ is a nonlinear function. The activation function ϕ is applied to each pre-activation value in the layer and is the main source of nonlinearity in standard feedforward networks. Its choice influences the range of unit outputs. It also defines how easily information and gradients can flow through many layers.

An activation function ϕ maps a scalar pre-activation $z \in \mathbb{R}$ to a scalar output $\phi(z)$. Three widely used choices are

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \qquad \mathrm{ReLU}(z) = \max(0, z).$$

The sigmoid function $\sigma(z)$ maps real-valued inputs to the open interval $(0, 1)$. The hyperbolic tangent $\tanh(z)$ maps inputs to the interval $(-1, 1)$ and is zero centered. The rectified linear unit $\mathrm{ReLU}(z)$ returns zero for negative inputs and grows linearly for positive inputs. These different shapes and saturation properties are illustrated in Figure 2.

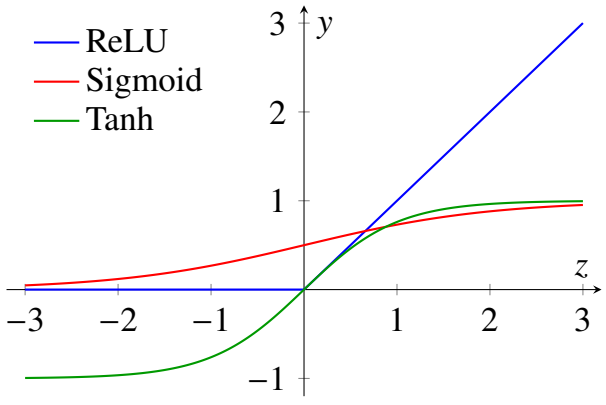


Figure 2: Common activation functions.

From an optimization point of view, these functions have different properties. For sigmoid and tanh, the derivative becomes small in the

saturation regions, which can lead to vanishing gradients in deep networks and slow learning. ReLU avoids saturation for positive inputs and keeps a constant gradient in that region, but its derivative is zero for negative inputs, so some units can become inactive if their pre-activations remain negative.

In practice, ReLU is a common default choice for hidden layers in deep feedforward and convolutional networks because it is simple, computationally cheap, and tends to train reliably. Sigmoid activations are mainly used when an output is interpreted as a probability from the interval $(0, 1)$, for example, in binary classification. The tanh function is useful when a symmetric bounded range is desired, for example, for some latent or recurrent states. Hovewer, deep networks that rely only on tanh activations can be harder to train without additional techniques.

2.2.2 Output Layers and Loss Functions

A neural network with parameters $\boldsymbol{\theta}$ defines a mapping from an input vector $\mathbf{x}$ to an output $\widehat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$. The role of the output layer is to transform the last hidden representation into a quantity that is suitable for the task, for example, a real-valued vector, a probability distribution over classes, or a set of scores.

Training is formulated as the minimization of a scalar objective (or loss) function. Given a collection of inputs $\{\mathbf{x}_i\}_{i=1}^N$ and corresponding signals $\{s_i\}_{i=1}^N$ that define the learning objective, the empirical loss function on the dataset can be written as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(f(\mathbf{x}_i; \boldsymbol{\theta}),\ s_i\big).$$

Where $\mathcal{L}$, denotes is a loss function evaluated for a single example and s_i denotes the supervision signal for that example. In supervised learning, s_i is an explicit target $\mathbf{y}_i$.

The specific form of the loss depends on the problem. For continuous targets, it is common to use a linear output layer together with a squared error loss. For regression with vector targets $\mathbf{y}_i$ and predictions $\widehat{\mathbf{y}}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$, a common choice is the mean squared error (MSE)

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{y}_i - \widehat{\mathbf{y}}_i \right\|_2^2.$$

where $\| \cdot \|_2$ denotes the Euclidean norm of a vector argument.

For multi-class classification, the output layer typically produces class scores that are converted to probabilities by a softmax transform, and the model is trained with a cross-entropy loss.

In unsupervised and self-supervised learning, the loss is defined as a differentiable objective of the model outputs and inputs. It does not rely on explicit target labels and often includes reconstruction terms, consistency terms, or penalties that encode known properties of the data or of the underlying system.

2.2.3 Training Neural Networks

Once the model $f(\mathbf{x}; \boldsymbol{\theta})$ and loss $\mathcal{L}(\boldsymbol{\theta})$ are defined, training is posed as numerical minimization with respect to $\boldsymbol{\theta}$. In deep learning, this optimization is carried out by gradient-based methods.

The dataset is split into a training set $\mathcal{D}_{\text{train}}$, a validation set $\mathcal{D}_{\text{val}}$, and a test set $\mathcal{D}_{\text{test}}$. The training set is used to update the model parameters. The validation set is used to tune hyperparameters, that is, configuration parameters of the model and the training procedure (such as the learning rate, batch size, or the number of layers) that are chosen before training instead of being learned from data. The test set is kept aside for the final performance evaluation. Input features are typically standardized featurewise using statistics computed on $\mathcal{D}_{\text{train}}$. Examples are shuffled at the beginning of

each epoch, where one epoch is a single pass through all training examples in $\mathcal{D}_{\text{train}}$.

For large datasets, exact gradients over all data are hard to compute, therefore, gradients are estimated on batches of data. At iteration t a subset $B_t \subset \{1, \ldots, N\}$ is sampled, then the batch loss

$$\mathcal{L}_{\text{batch}}(\boldsymbol{\theta}_t) = \frac{1}{|B_t|} \sum_{i \in B_t} \mathcal{L}\big(f(\mathbf{x}_i; \boldsymbol{\theta}_t),\ s_i\big)$$

is formed, and parameters are updated as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \, \nabla_{\boldsymbol{\theta}} \, \mathcal{L}_{\text{batch}}(\boldsymbol{\theta}_t),$$

where $\eta_t > 0$ is the learning rate. This procedure is referred to as stochastic gradient descent (SGD). Gradients are computed efficiently by backpropagation, which applies the chain rule through the computation graph.

In practice, the learning rate is commonly controlled by a schedule that varies with the iteration index, which helps stabilize and accelerate training. Besides plain SGD, momentum methods and adaptive optimizers such as Adam [12] are widely used to improve convergence by adjusting the effective step size per parameter.

To improve generalization, common practice includes weight decay and early stopping based on a held-out validation set. After each epoch, the current parameters are evaluated on $\mathcal{D}_{\text{val}}$. Training stops when the validation loss does not improve for a fixed patience window, and the iterate with the lowest validation loss is kept for final reporting on $\mathcal{D}_{\text{test}}$ without further tuning.

3 Transformer

Early sequence models were based on Markov chains. In a Markov chains, the next element depends only on the previous element and not on the full history of the sequence. Such models were simple and efficient, but they could not capture long-range dependencies or complex relations in the data [13]. ANNs extended sequence modelling beyond this limitation. Convolutional networks detect local patterns, but their receptive field grows slowly with depth [14]. Recurrent neural networks (RNNs) can in principle model longer dependencies, but their sequential nature makes training slow and prevents efficient parallelization. Each step depends on the previous one, creating a bottleneck that limits throughput even on modern hardware. Moreover, information must be propagated through many time steps, which often leads to vanishing or exploding gradients and unstable training [15].

The attention mechanism was first introduced to help RNNs focus on the most relevant parts of the input. Although this improved results, the processing remained sequential. The key breakthrough came with the Transformer [6], which removed both recurrence and convolution entirely. This new architecture replaced the step-by-step processing with a fully parallel encoder–decoder design. The encoder processes all input elements (called tokens) in parallel and outputs contextual representations. The decoder generates the output sequence while attending to both the encoder output and the previously generated tokens through a causal mask. Since the model has no inherent notion of order, positional information is added to token embeddings before entering the encoder.

In short term, transformers became the foundation for most state-of-

the-art scalable models, including BERT [16] and GPT [17] in language processing and Vision Transformer [18] in image analysis, showing that the same architecture can be successfully applied across very different domains. Recent studies also report promising results in wireless communications, for example in channel estimation [19], detection [20], and resource allocation [9, 21]

A comprehensive survey of transformer models [22] describes the main architectural features that make the model applicable to many different types of data. The self-attention mechanism captures relationships between all elements of the input, allowing the model to represent global dependencies. Parallel processing makes training faster and more efficient compared to recurrent architectures. In addition, the modular layer structure with attention, feed-forward, normalization, and residual connections can be reused with only small adjustments for new kinds of data. These properties together make the transformer a flexible and general architecture, which explains its rapid adoption across different domains.

3.1 Attention Mechanism

Attention provides a flexible way for a model to relate different elements of a sequence directly to each other. Instead of processing inputs in a fixed order, the model dynamically determines which parts of the input are most relevant to each output position, and different parts of the input sequence are assigned different weights depending on their relevance. As a result, the resulting representation becomes a weighted combination of all inputs, where the most informative tokens contribute more strongly to the output representation.

To illustrate internal behaviour of the attention mechanism, we visualize how each token attends to other tokens within a sentence. Figure 3 shows such example of self-attention for the token “it” in the sentence *“The signal*

was distorted after the channel changed because it was noisy."

Here, the model assigns the strongest attention weight to *channel*, indicating that "it" refers to the channel as the source of noise. The mechanism considers the entire sequence when forming contextual representations, and each token assigns different weights to all other tokens depending on their relevance in the given context. As a result, the model captures contextual dependencies within the sequence and forms representations that reflect the actual meaning of the sentence.

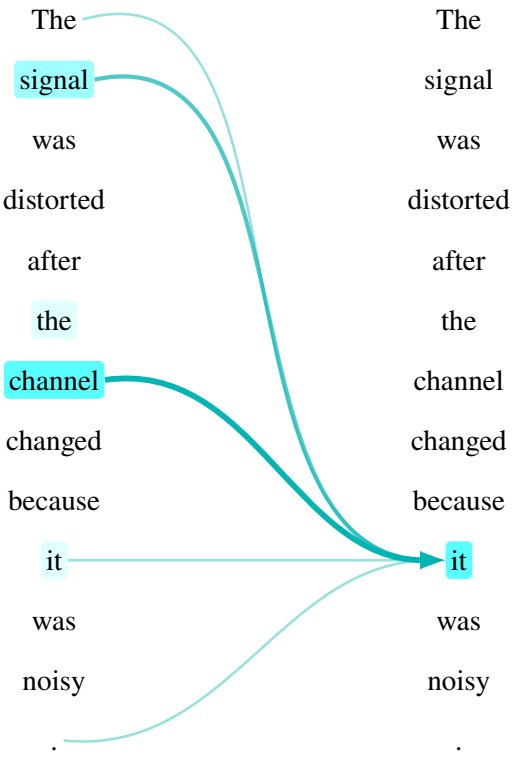


Figure 3: Self-attention for the token "it".

Figure 4 shows another example with a similar sentence structure but a different meaning. In the sentence *"The signal was distorted after the channel changed because it was weak,"* the attention weight shifts from *channel* to *signal*. This means that the model now interprets "it" as referring to the signal itself. Although both sentences share the same syntactic

structure, the distribution of attention differs, showing that the mechanism adapts to the semantic context rather than relying on fixed positional patterns.

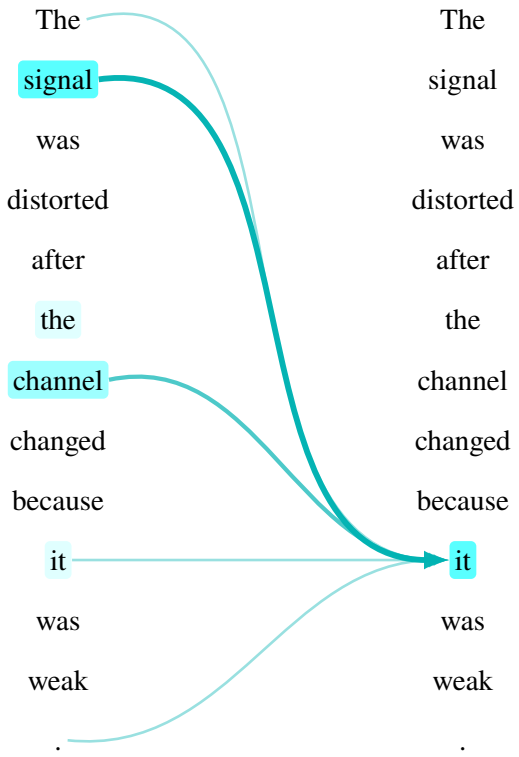


Figure 4: Self-attention for the token “it”.

Mathematically, attention can be described as a process that computes how much each element of a sequence should contribute to the representation of every other element. Each input vector $\mathbf{x}_i$ is first projected into three separate representations: a query $\mathbf{q}_i$, a key $\mathbf{k}_i$, and a value $\mathbf{v}_i$.

These are obtained using learned projection matrices:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}^K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}^V,$$

where $\mathbf{X}$ is the matrix of input embeddings and $\mathbf{W}^Q$, $\mathbf{W}^K$, and $\mathbf{W}^V$ are trainable matrices.

The attention mechanism measures the similarity between each query

and all keys using the dot product, producing a matrix of attention scores.

To prevent large values from dominating, the softmax operation is performed, that is, the scores are scaled by the key dimension d_k and normalized:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right).$$

The resulting attention weights determine how much information each token should take from the others. The output representations are then computed as a weighted sum of the value vectors:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A}\mathbf{V}.$$

Through this operation, each token becomes a context-aware combination of all other tokens, where the weights reflect their relative importance.

3.1.1 Multi-Head Attention

In practice, the model employs several attention mechanisms, called Heads, that operate in parallel. Each Head has its own set of parameters and can focus on different dependencies in the sequence. Formally, for each Head $i = 1, \ldots, h$:

$$\text{Head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q,\ \mathbf{K}\mathbf{W}_i^K,\ \mathbf{V}\mathbf{W}_i^V),$$

and the outputs of all Heads are concatenated and linearly transformed:

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{Head}_1, \ldots, \text{Head}_h)\ \mathbf{W}^O.$$

This multi-Head structure allows the model to capture diverse relationships simultaneously. While one Head may focus on local details, others can attend to long-range dependencies or global patterns in the data.

3.1.2 Positional Information

Because attention treats all tokens as a set, it does not remember their position information. To introduce positional awareness, each input embedding is combined with a positional encoding vector. A common formulation uses fixed sinusoidal encodings defined as

$$[\mathbf{PE}]_{pos,\,2i} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \qquad [\mathbf{PE}]_{pos,\,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right).$$

These encodings provide a continuous and generalizable way for the model to distinguish the position of each token and to infer relative distances within the sequence.

3.2 Transformer Architecture

The transformer architecture is built by stacking layers that combine the attention mechanism with additional components designed to improve learning stability and model capacity. It includes two main elements: the encoder and the decoder. The encoder processes the input sequence and produces contextual representations of the input. The decoder generates the output sequence by attending to these encoder representations and to the tokens produced so far. The overall block-scheme of the model is shown in Figure 5, which illustrates how information flows from the input to the output through the encoder–decoder structure.

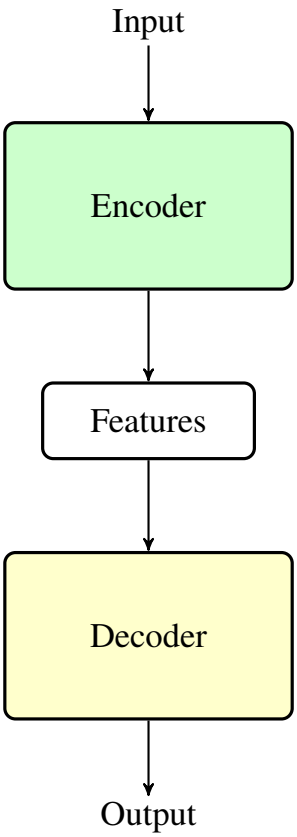


Figure 5: General encoder–decoder structure of the Transformer.

Many variants of the transformer have been developed for different types of data and learning tasks. Some models, such as BERT [16], use only the

encoder, while others, like the original Transformer for translation, combine both the encoder and the decoder. There are also decoder-only models, such as GPT [17], which rely entirely on self-attention within the decoder.

As shown in Figure 6, each encoder block consists of two main components: a multi-head attention module and a position-wise feed-forward network.

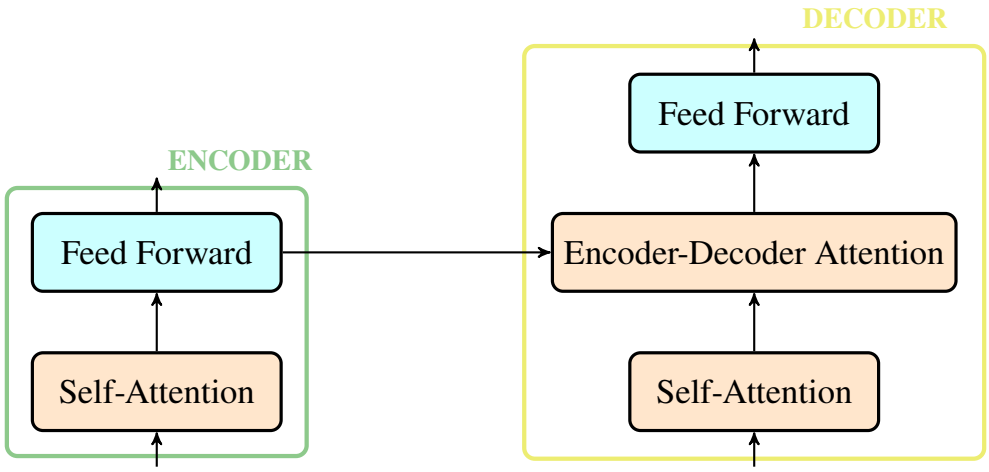


Figure 6: Overview of the Transformer encoder and decoder structure.

The encoder receives the input sequence and processes it through a stack of layers, each containing a self-attention mechanism followed by a feed-forward network. Self-attention allows the encoder to consider relationships between different positions of the input, enabling it to capture context when representing each token. The feed-forward network then refines these representations for further processing.

The decoder follows a similar structure but includes an additional encoder–decoder attention layer between its self-attention and feed-forward sublayers. This layer allows the decoder to selectively focus on the most relevant parts of the encoded input when generating the output sequence. Together, these components form the core of the Transformer architecture, where attention mechanisms ensure that both encoder and decoder operate contextually rather than positionally.

3.2.1 Encoder

The encoder is responsible for converting the input sequence into contextual representations that capture relationships between all tokens. Its key property is that all input tokens are processed in parallel, which allows the model to efficiently learn dependencies across the entire sequence.

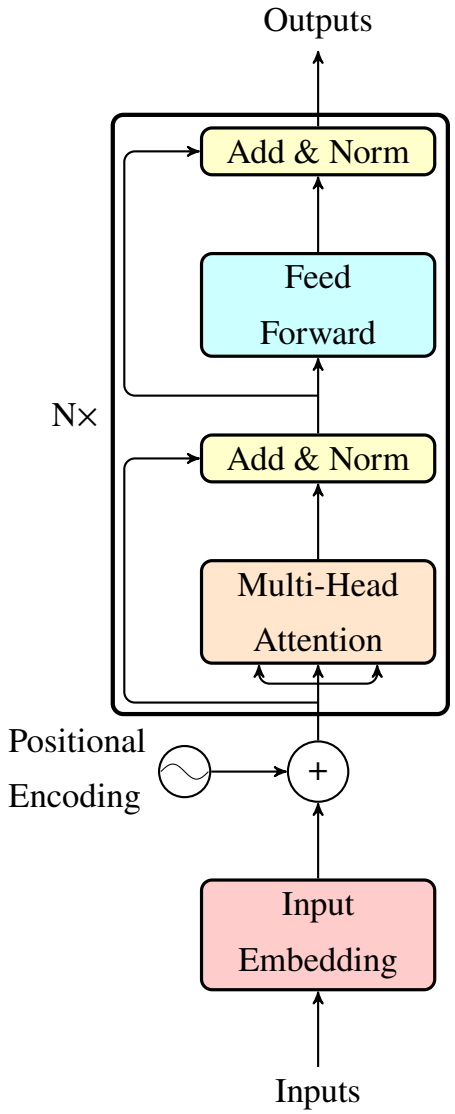


Figure 7: Structure of the Transformer encoder block.

As it can be seen from the encoder structure shown in Figure 7, the process starts with the input tokens, which are mapped into continuous

vector representations by the embedding layer.

Let us assume that we are given $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \in \mathbb{R}^{N \times d_{\text{model}}}$ which denote the matrix of token embeddings.

The next step is positional encoding. The encoder processes tokens in parallel and does not know their order by itself, so we add positional vectors $\mathbf{P} \in \mathbb{R}^{N \times d_{\text{model}}}$ elementwise:

$$\mathbf{Z} = \mathbf{X} + \mathbf{P}.$$

This addition produces position-aware representations that preserve both the semantic meaning of tokens and their order in the input.

Next comes the multi-head attention layer, whose detailed mechanism is discussed in Section 3.1.1. It allows each token to attend to all other tokens in the input sequence, enabling the model to capture contextual dependencies regardless of their distance. The output of this sublayer is wrapped with a residual connection and layer normalization:

$$\mathbf{Z}' = \text{LayerNorm}\big(\mathbf{Z} + \text{MHA}(\mathbf{Z})\big),$$

which helps preserve information and improve gradient flow during training. After that, layer normalization is applied to each token embedding independently, stabilizing the learning process and preventing issues such as exploding or vanishing gradients. It behaves the same during both training and inference, which makes it more robust than batch normalization for sequence models.

Finally, the position-wise feed-forward network applies two linear transformations with a nonlinearity between them, processing each token independently:

$$\text{FFN}(\mathbf{x}) = \phi(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2,$$

where ϕ is typically a ReLU activation. Its output is again followed by an

Add & Norm operation:

$$\mathbf{Z}'' = \text{LayerNorm}\big(\mathbf{Z}' + \text{FFN}(\mathbf{Z}')\big),$$

completing one encoder block. In the full encoder, this block is repeated N times, allowing the model to gradually refine token representations across multiple layers.

An important property of the encoder design is that the input and output of each sublayer have the same dimensionality. Each layer takes a sequence of N embeddings of size d_{model} and returns a sequence of the same shape. Because of this, encoder blocks can be stacked without any intermediate reshaping, forming a deep architecture that refines token representations layer by layer. Furthermore, all sublayers except attention layer operate independently on each token, and even attention itself is highly parallelizable. As a result, the encoder can process sequences of almost any length efficiently and in parallel. This high degree of parallelism makes the Transformer architecture significantly faster and more scalable than previously used recurrent approaches.

3.2.2 Decoder

The decoder has a structure similar to the encoder and consists of three main sublayers: masked multi-head self-attention, encoder–decoder attention, and a feed-forward network. Each sublayer is followed by residual connection and layer normalization. While the encoder reads and encodes the entire input sequence, the decoder generates the output sequence step by step, relying on both its previous outputs and the representations produced by the encoder.

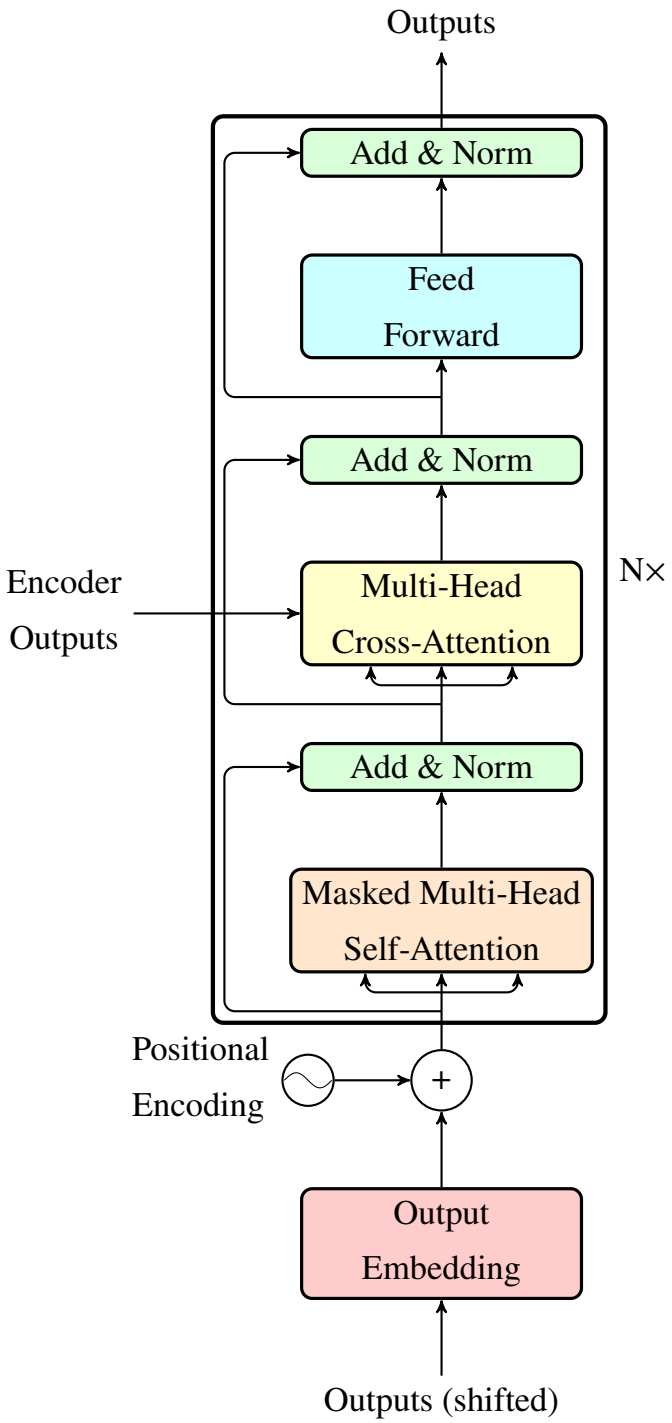


Figure 8: Structure of the Transformer decoder block.

As we can see in Figure 8, the decoder receives as input the shifted output sequence, where each token embedding is combined with positional encoding in the same way as in the encoder. The sequence is shifted so

that each position can only depend on the preceding tokens, ensuring that the model predicts the next token step by step. The resulting vectors are then processed by a masked multi-head self-attention layer. The mask allows each token to attend only to earlier tokens in the output sequence, which preserves the autoregressive property of the model and avoids using information from future positions.

Mathematically, this is achieved by applying a mask $\mathbf{M}$ to the attention scores before the softmax operation:

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d_k}} + \mathbf{M}\right),$$

where $\mathbf{M}$ is a masking matrix defined as

$$[\mathbf{M}]_{ij} = \begin{cases} 0, & j \leq i, \\ -\infty, & j > i. \end{cases}$$

This matrix restricts attention to the current and previous positions, assigning large negative values to all future positions so that their corresponding probabilities after the softmax become zero.

In the next layer, the cross-attention mechanism is the key connection between the encoder and decoder, where the queries $\mathbf{Q}$ are obtained from the decoder representations, while the keys $\mathbf{K}$ and values $\mathbf{V}$ are taken from the encoder outputs. This allows the decoder to selectively focus on relevant parts of the encoded input sequence when generating each token in the output.

Formally, it can be written as

$$\text{CrossAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{QK}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

where $\mathbf{Q} = \mathbf{YW}^Q$, $\mathbf{K} = \mathbf{HW}^K$, and $\mathbf{V} = \mathbf{HW}^V$, with $\mathbf{Y}$ denoting decoder

embeddings and $\mathbf{H}$ denoting encoder outputs.

Each of the attention layers is followed by a residual connection and layer normalization, as in the encoder. Finally, a position-wise feed-forward network refines the decoder's internal representations:

$$\mathrm{FFN}(\mathbf{x}) = \phi(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2,$$

and its output is again wrapped with residual and normalization layers. Stacking N such decoder blocks enables the model to progressively integrate information from both past outputs and encoder representations.

Despite having no recurrent connections, the decoder still generates sequences in an autoregressive manner due to masking. During training, all tokens can be processed in parallel by applying the mask matrix, while during inference, the model generates tokens one by one, feeding previously produced outputs back into the decoder.

3.2.3 Transformer structure

The full Transformer architecture can be viewed as an encoder–decoder system that maps an input sequence to an output sequence through stacked attention and feed-forward layers. Its overall structure is shown in Figure 9.

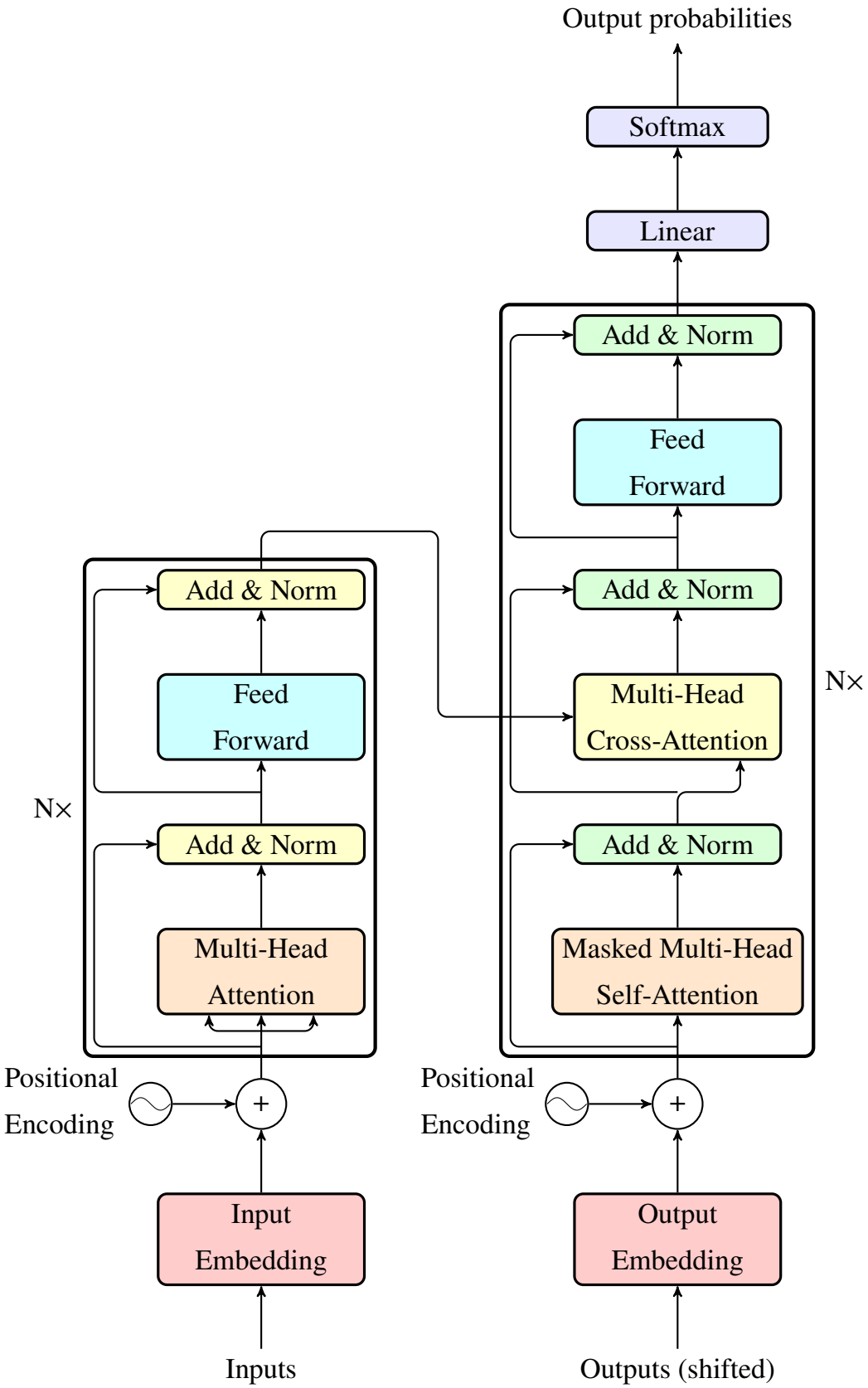


Figure 9: Transformer architecture

On the encoder side, the model receives a sequence of input tokens, such as words or generic symbols. Each token is first mapped to a dense vector through an embedding layer, and the resulting matrix $\mathbf{X}$ collects these token embeddings for a sequence of length N. Since the attention mechanism itself is permutation invariant with respect to the token order, positional encodings $\mathbf{P} \in \mathbb{R}^{N \times d_{\text{model}}}$ are added to the embeddings to provide information about token positions,

$$\mathbf{H}^{(0)} = \mathbf{X} + \mathbf{P}.$$

This position-aware representation is then processed by a stack of N_{enc} identical encoder layers. Each encoder layer applies multi-Head self-attention followed by a position-wise feed-forward network, with residual connections and layer normalization around both sublayers [6]. Conceptually, the encoder stack refines the same sequence representation layer by layer and produces the final encoder output

$$\mathbf{H}^{\text{enc}} = \mathbf{H}^{(N_{\text{enc}})},$$

which serves as a set of context-dependent features for the decoder.

The decoder operates on the output side. During training, it receives the ground-truth target sequence shifted by one position, so that the token at time step t is predicted based on all target tokens up to step $t-1$. These tokens are embedded and combined with positional encodings in the same way as in the encoder, which yields an initial decoder state $\mathbf{Y}^{(0)} \in \mathbb{R}^{T \times d_{\text{model}}}$ for a sequence of length T. Each decoder layer then performs three operations. First, masked self-attention allows each position to attend only to previous positions in the output sequence, which enforces the autoregressive property and avoids access to future targets. Second, cross-attention uses the current decoder representation as queries and the encoder output $\mathbf{H}^{\text{enc}}$ as keys and values. This mechanism allows the decoder to focus on those parts of the encoded input that are most relevant for generating the next output token. Third, a position-wise feed-forward network refines the decoder representation,

again wrapped with residual connections and layer normalization.

After N_{dec} decoder layers, the final decoder representation $\mathbf{Y}^{(N_{\text{dec}})}$ is mapped to output logits by a linear projection and converted into probabilities by a softmax function,

$$\hat{\mathbf{y}}_t = \text{softmax}\big(\mathbf{Y}_t^{(N_{\text{dec}})}\mathbf{W}_{\text{out}} + \mathbf{b}_{\text{out}}\big),$$

where $\hat{\mathbf{y}}_t$ is the predicted distribution for position t. During training, all positions in the encoder and the decoder can be processed in parallel since attention and feed-forward layers act on all tokens in a sequence at the same time, while masking in the decoder guarantees the correct causal structure. During inference, the decoder generates tokens one by one and feeds previously generated outputs back as its own input.

Overall, the Transformer can be viewed as an encoder that builds a contextual representation of the input sequence and a decoder that generates the output sequence by combining its own history with this encoder representation through attention.

4 Cell-Free MIMO

In modern wireless communications, rising traffic, tighter latency budgets, and increasing network complexity place additional demands on the scalability and responsiveness of resource allocation and control across distributed networks. Classical cellular architectures partition the service area into disjoint cells, each served by a base station that mainly processes the signals of users located within its own cell. This cell-centric design simplifies resource allocation and frequency planning, but it also introduces hard cell boundaries and strong inter-cell interference, especially for users located near the cell edges. As the network becomes more dense and heterogeneous, these effects make it difficult to guarantee uniform quality of service across all users [1].

Massive MIMO has been proposed as a key technology to address some of these limitations [23]. In a classical massive MIMO deployment, each base station is equipped with a large number of co-located antennas and serves many users simultaneously on the same time–frequency resources by spatial multiplexing. This architecture improves spectral and energy efficiency by exploiting array gain and spatial diversity. However, it still relies on the concept of cells and therefore inherits the basic drawbacks of cellular networks, such as cell-edge interference and performance variations between users in favorable and unfavorable locations [7].

CFmMIMO has been introduced as an alternative network architecture that removes the notion of cells [7, 24]. In CFmMIMO, a large number of geographically distributed APs are connected to a central processing unit (CPU) and jointly serve all users in the coverage area on the same

time-frequency resources. Under the canonical CFmMIMO model, each AP can participate in the transmission to every user, so the whole deployment acts as a single large cooperative antenna array. From the user perspective, the network appears as one cooperative entity rather than a set of competing cells. For large deployments, a common variant is user-centric CFmMIMO [25], where each user is served only by a small set of nearby APs with strong large-scale fading. This reduces fronthaul and processing load while keeping most of the macro diversity and near-uniform service quality of the canonical cell-free model.

This distributed architecture provides several important benefits. First, the large number of APs and their distributed placement create a high degree of macro diversity. Even if some APs experience deep fading or shadowing, other nearby APs can still provide strong links, which improve the reliability of the received signal. Second, by allowing coherent joint transmission from multiple APs, CFmMIMO can reduce inter-user interference and provide more uniform spectral efficiency across the coverage area, including locations that correspond to cell edges in a traditional network. Third, CFmMIMO can improve energy efficiency by enabling flexible power allocation. An illustrative comparison between a classical cellular layout and a CFmMIMO deployment is shown in Fig. 10.

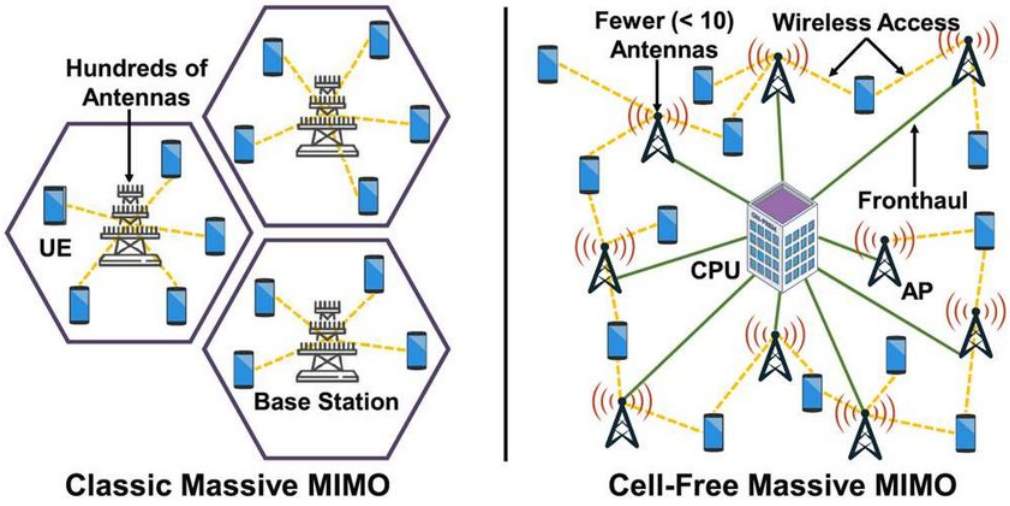


Figure 10: Conventional cellular network and cell-free massive MIMO network [26].

To fully use these advantages, CFmMIMO systems are usually operated in time-division duplexing (TDD) mode [7]. Users send uplink pilot signals, APs estimate the channels, and the same channel estimates are reused for downlink precoding and power control. Depending on the available fronthaul capacity and processing capabilities, these operations can be implemented in a fully centralized manner at the CPU or partly distributed manner across the APs.

The cell-free architecture also introduces several challenges [24]. The first challenge is scalability. As the number of APs and users grows, the requirements on fronthaul and backhaul capacity, synchronization, and computational resources at the CPU become significant. Efficient algorithms are needed for channel estimation, scheduling, and power control that can operate with limited signaling overhead and processing time. A second challenge is pilot contamination. The number of orthogonal pilot sequences is limited by the coherence interval, so pilots must be reused across users. In CFmMIMO, pilot reuse creates coherent interference across the entire distributed network, which directly affects channel estimation quality and the effectiveness of downlink beamforming. A third challenge is the design

of robust and efficient power control.

Because of these benefits and despite the challenges, which hopefully can be resolved, CFmMIMO is considered a strong candidate for beyond-5G and 6G deployments [1]. At the same time, the increasing network complexity and strict performance requirements motivate attention-based deep learning for wireless resource optimization. In this thesis, power control in CFmMIMO is studied as a representative resource allocation problem.

4.1 Power Control in Cell-Free Massive MIMO

Downlink power control is one of the central design problems in CFmMIMO networks [7, 8]. The power control coefficients determine how the available transmit power at the APs is distributed across users. This distribution affects both the strength of the desired signal and the level of interference for each user and therefore has a direct impact on spectral efficiency and fairness.

Let $\eta_{mk} \geq 0$ denote the power control coefficient used by AP m when transmitting to user k. These coefficients are chosen under per-AP power constraints. If ρ_d denotes the maximum downlink power at each AP, the constraints can be written as

$$\sum_{k=1}^{K} \eta_{mk} \leq \rho_d, \quad m = 1, \ldots, M.$$

The collection of all coefficients can be represented by a vector $\boldsymbol{\eta} \in \mathbb{R}^{MK}$ that describes how the total downlink power is allocated across all AP and user pairs. In practical CFmMIMO designs, $\boldsymbol{\eta}$ is typically optimized based on large-scale fading, pilot allocation, and noise variance, while small-scale fading is averaged out in the performance expressions [7].

Under standard CFmMIMO assumptions with TDD operation and uplink

pilots, the effective downlink signal-to-interference-plus-noise ratio (SINR) of each user can be expressed in closed form as a function of $\boldsymbol{\eta}$ and the large-scale fading parameters. The corresponding spectral efficiency of user k can be written as

$$\mathrm{SE}_k(\boldsymbol{\eta}) = \left(1 - \frac{\tau_p}{\tau_c}\right) \log_2\big(1 + \gamma_k(\boldsymbol{\eta})\big),$$

where τ_p is the pilot length, τ_c is the coherence interval, and $\gamma_k(\boldsymbol{\eta})$ is the effective SINR that captures the useful signal, interference from other users, pilot contamination, and noise. The exact form of $\gamma_k(\boldsymbol{\eta})$ depends on the adopted CFmMIMO system model, but it only involves statistical channel information [7].

Different performance metrics (objective functions) can be defined based on the collection $\{\mathrm{SE}_k(\boldsymbol{\eta})\}_{k=1}^{K}$. A common design target in CFmMIMO is max-min fairness, where the goal is to maximize the minimum user spectral efficiency in order to provide a more uniform quality of service. The corresponding optimization problem can be written as

$$\begin{aligned} \max_{\boldsymbol{\eta}} \quad & \min_{k} \ \mathrm{SE}_k(\boldsymbol{\eta}) \\ \text{subject to} \quad & \sum_{k=1}^{K} \eta_{mk} \leq \rho_d, \quad m = 1, \ldots, M, \\ & \eta_{mk} \geq 0, \quad m = 1, \ldots, M, \ k = 1, \ldots, K. \end{aligned}$$

This problem is non-convex and large scale. The power control coefficients for all AP and user pairs are coupled through the interference terms in $\gamma_k(\boldsymbol{\eta})$, which makes it difficult to optimize each user independently.

Classical solutions rely on iterative optimization algorithms. Equal power allocation (EPA) can be considered as a simple baseline that distributes the available transmit power equally across the served users and satisfies the per-AP power constraints. EPA has negligible computational cost. Accelerated projected gradient (APG) methods, in contrast, iteratively

update $\boldsymbol{\eta}$ along gradient directions of a smooth objective and project the updates onto the feasible set defined by the power constraints [8, 27]. APG-based solvers can achieve high-quality max-min solutions, but their runtime grows with the number of APs, the number of users, and the required accuracy, which makes APG unacceptably expensive for large CFmMIMO networks.

An alternative view is to interpret power control as a high-dimensional mapping from network parameters to power coefficients. Each CFmMIMO snapshot is described by quantities such as the large-scale fading matrix, the pilot allocation, and the noise and power parameters. The power control problem then consists of mapping this snapshot to a vector of coefficients $\boldsymbol{\eta}$ that yields high and balanced SEs. This perspective motivates deep learning models that learn such a mapping from generated CFmMIMO snapshots and approximate near-optimal power control with low inference time.

5 Attention in Cell-Free MIMO

ML methods have recently been explored for several design problems in CFmMIMO, such as user association, pilot allocation, channel estimation, and power control [1]. Among these tasks, downlink power control is one of the most critical and computationally demanding. The goal is to determine power control coefficients $\boldsymbol{\eta}$ that depend on the global interference structure in the network and on per-AP power constraints. In practice, these coefficients must be recomputed many times as users move, channels change, and pilot allocations are updated. This repeated optimization can become a bottleneck when latency and energy constraints are tight [8].

A single CFmMIMO snapshot already contains rich structure. The large-scale fading between APs and users describes which links are strong or weak. The pilot allocation pattern determines which users create coherent interference through pilot reuse. Together, these quantities define a global interference structure where a change in the power of one user can affect many others through shared APs and shared pilots. A data-driven model for power control should therefore be able to represent such non-local dependencies and generalize across different numbers of users and APs.

Self-attention-based architectures are well suited for this type of structured data [21]. They operate on a set of input feature vectors and, for each element in the set, form a new representation as a learned weighted combination of all other elements. In the CFmMIMO setting, each user can be represented by a feature vector that summarizes its large-scale fading to the APs, its pilot assignment, and possibly other scenario parameters. The attention mechanism then learns which other users are most relevant for the

power control decision of a given user. Since the same attention module is applied to all users, the model is naturally invariant to permutations of user indices and can be scaled to different network sizes with moderate changes in architecture.

5.1 Data

The model is trained and evaluated on data generated by Monte Carlo simulation [28] of a TDD CFmMIMO system [21]. Each training sample consists of the large-scale fading matrix $\mathbf{B} \in \mathbb{R}^{M \times K}$ and a pilot-allocation mask $\mathbf{\Phi}$ that identifies users sharing the same pilot. The learning setup is unsupervised, so no target power coefficients are stored, and the objective function is computed directly from the analytical system model during training. Both the training and test snapshots are obtained from the same Monte Carlo simulator, but the test set is generated independently and is used only for reporting the resulting performance.

To test the scalability and adaptability of the transformer across different dataset sizes and problem dimensions, three configurations are considered. These configurations range from a small, contamination-free setting to larger ones where pilot reuse is required. They cover cases with few users and APs, as well as cases with about a hundred APs and several tens of users.

Here, M denotes the number of APs and K denotes the number of users.

- **C0 (small, no pilot reuse).** A simple scenario with M=10 and K=4. Pilots are orthogonal, so pilot contamination does not appear.
- **C1 (medium, no pilot reuse).** M=100, K=20. Pilots remain orthogonal. Included as a main training configuration.
- **C2 (large, with pilot reuse).** M=100, K=40. Since K exceeds the pilot pool, pilots are reused, which induces pilot contamination. Included

in training to evaluate robustness under pilot contamination.

For each configuration, AP positions are fixed and user locations are drawn uniformly. Large-scale fading for all pairs of APs and users is computed from geometry using a standard path-loss model with log-normal shadowing. Pilots are drawn from an orthogonal pool. When K exceeds the pool size, pilots are reused, which induces pilot contamination. The inputs to the network are the large-scale fading matrix $\mathbf{B}$ and the pilot-allocation matrix $\mathbf{\Phi}$.

In training, the order of millions of independent samples per training configuration is used, while evaluation relies on a fresh set of a few thousand samples. All samples are generated offline and used during training.

5.2 Training Setup

Training is carried out on the Aalto Triton high-performance computing cluster [29]. Triton uses the Slurm scheduler [30] to allocate CPUs, memory, GPUs, and wall time to batch jobs. The experiments are run on GPU nodes with research-grade NVIDIA accelerators, and a high-throughput parallel scratch filesystem is used for temporary data, checkpoints, and logs [31].

Jobs are launched with Slurm batch scripts [32]. Each script activates a fixed software environment and executes the training driver with explicit flags for the chosen configuration. Data for each configuration are prepared offline and stored on shared storage, then read during training. Checkpoints and logs are written to scratch to enable recovery and inspection. If maintenance or node draining interrupts a job, training resumes from the latest checkpoint.

In practice, the training code runs on a single GPU per job. Slurm allocations use one CPU core with 16 GB of memory per task. Parameter sweeps are organized as Slurm job arrays when independent runs are

available.

5.3 Training

The training process is based on the dataset and unsupervised setup defined in Subsection 5.1, where the optimization objective is derived directly from the system model rather than from labeled data. Each mini-batch consists of a pair $(\mathbf{B}, \mathbf{\Phi})$ that contains one large-scale fading snapshot and the corresponding pilot allocation. The first step after setting up the Triton training environment is to run Configuration C0 on a small subset of snapshots in order to verify data loading, objective evaluation, gradient backpropagation, and model checkpoint saving.

Once these initial test runs confirm that the training setup is functioning correctly, a series of longer training runs is carried out to identify a stable region of hyperparameters. In these runs, the following hyperparameters are varied:

- the number of Transformer layers between one and six,
- the hidden width parameter M_2, for example between 8 and 60 times the number of attention heads,
- the learning rate and its stepwise scheduling,
- the number of epochs on datasets of $4 \cdot 10^4$ to $4 \cdot 10^5$ snapshots.

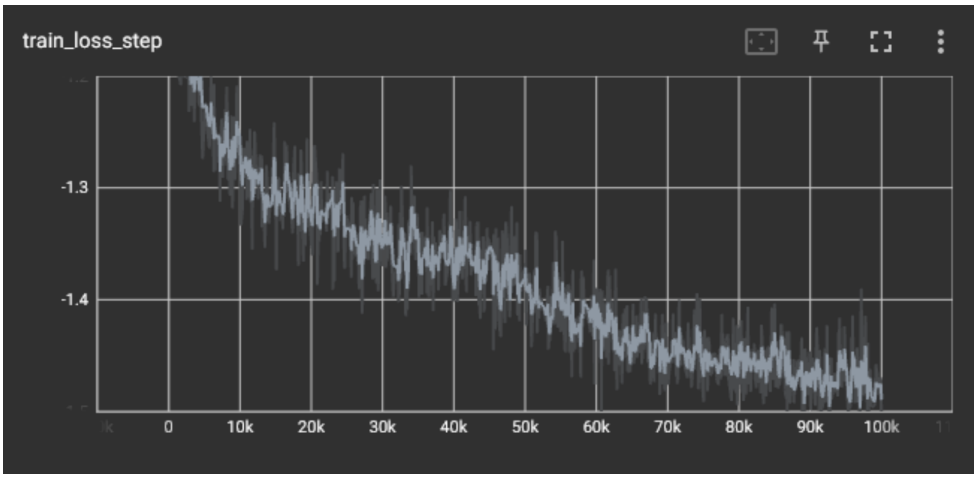
The corresponding performance curves show that three layers are sufficient. Deeper models do not provide consistent gains compared to the three-layer architecture, and changing the learning rate mainly affects the speed of convergence rather than the final value of the objective. Increasing the number of epochs beyond the default does not produce clear benefits. These observations indicate that the dominant factors are the model width

M_2 and the amount of training data rather than the learning rate schedule or the depth beyond three layers.

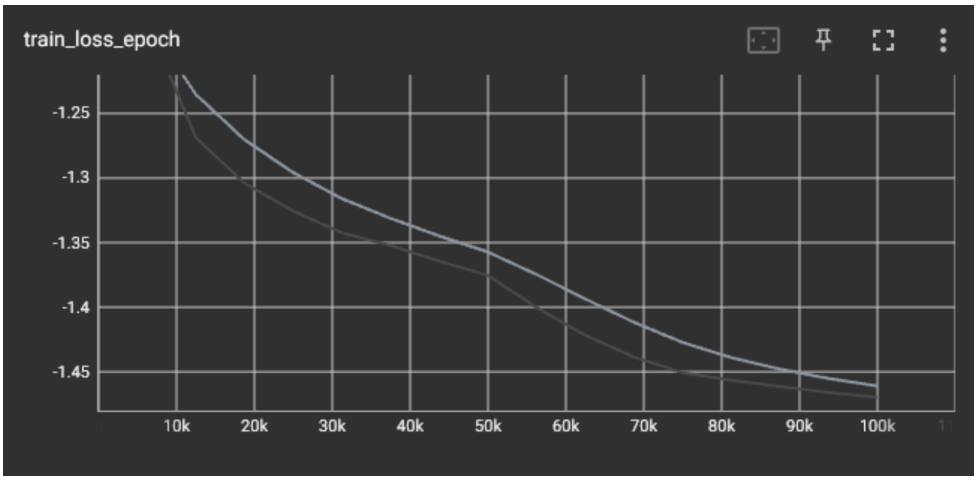
Based on these observations, the search focuses on the combination of M_2 and the number of independent snapshots. Several configurations with M_2 between 20 and 60 times the number of heads and with training set sizes between $4 \cdot 10^5$ and $3.2 \cdot 10^6$ snapshots are evaluated across configurations C0, C1, and C2. Increasing the training set size consistently improves the distribution of the per-user spectral efficiency, while wider models yield smaller but still visible gains once M_2 reaches the range of 40 to 60 times the number of heads.

After extensive experimentation with different hyperparameter settings, a single configuration that consistently performs best among the tested options is selected. The model uses three Transformer layers and a width parameter $M_2 = 50H$, where H is the number of attention heads. It is trained on $3.2 \cdot 10^6$ snapshots with a batch size of 256. The optimizer uses a fixed learning rate with the parameter `VARYING_STEP_SIZE` set to `false`. On Triton, a single training run of this configuration requires a Slurm time allocation of about 45 hours.

The behaviour of the loss curves during training is illustrated in Figures 11 and 12. For Configuration C0 (small, no pilot reuse), the step-wise training loss is noisy at the level of individual optimization steps, whereas the epoch-averaged loss is smooth and consistently decreasing. This indicates that the unsupervised objective and the gradient computation are numerically stable.



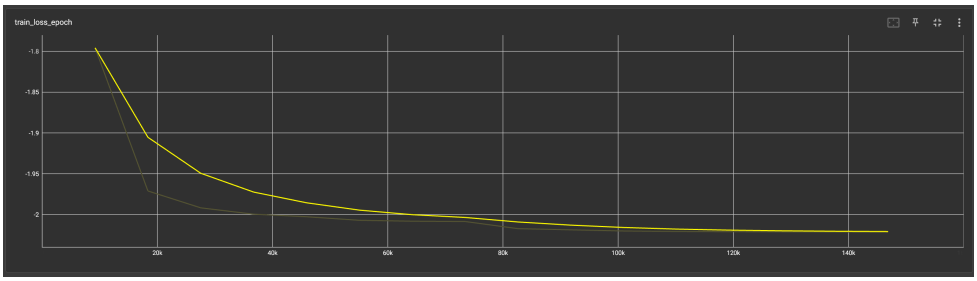
(a) Stepwise training loss.



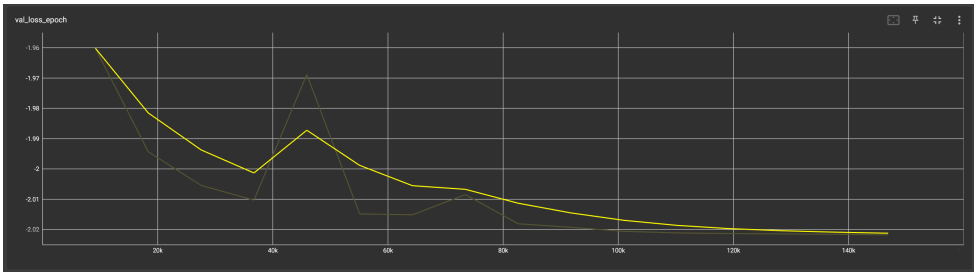
(b) Epoch averaged training loss.

Figure 11: Training loss for Configuration C0 (small, no pilot reuse).

For the final configuration used in the experiments, Figure 12 shows the epoch-averaged training and validation losses. Both losses decrease during training and then gradually level off. They remain close to each other over the entire run. This suggests that overfitting is limited for the chosen training length.



(a) Epoch averaged training loss.



(b) Epoch averaged validation loss.

Figure 12: Training and validation losses for the final configuration.

After these design choices are fixed, Configuration C1 is trained on the main CFmMIMO setup without pilot reuse, followed by Configuration C2, which uses the same architecture and hyperparameters but includes pilot reuse and stronger interference. For each configuration, the model is trained once and evaluated on an independent validation split and on the large test set used to compute the per-user SEs.

5.4 Results

To compare the obtained results with established optimization methods, the Transformer model is evaluated against EPA and APG. The evaluation focuses on the per-user SE achieved on the test sets for Configurations C1 and C2, while Configuration C0 is used mainly for debugging and hyperparameter selection, and therefore, C0 is not included in the final figures.

For each configuration and method, per-user SEs are computed on independent test snapshots and summarized in the form of quantile curves

as a function of the user percentile. Low percentiles (towards the left of the curves) correspond to the weakest users in the network, such as those with poor channel conditions or strong interference, whereas high percentiles correspond to users with favourable propagation conditions.

Figure 13 shows the quantile curves for Configuration C1, which represents a medium-size CFmMIMO setup without pilot reuse. The Transformer model achieves per-user SEs that are very close to those of APG across almost all percentiles, while both methods clearly outperform the EPA baseline.

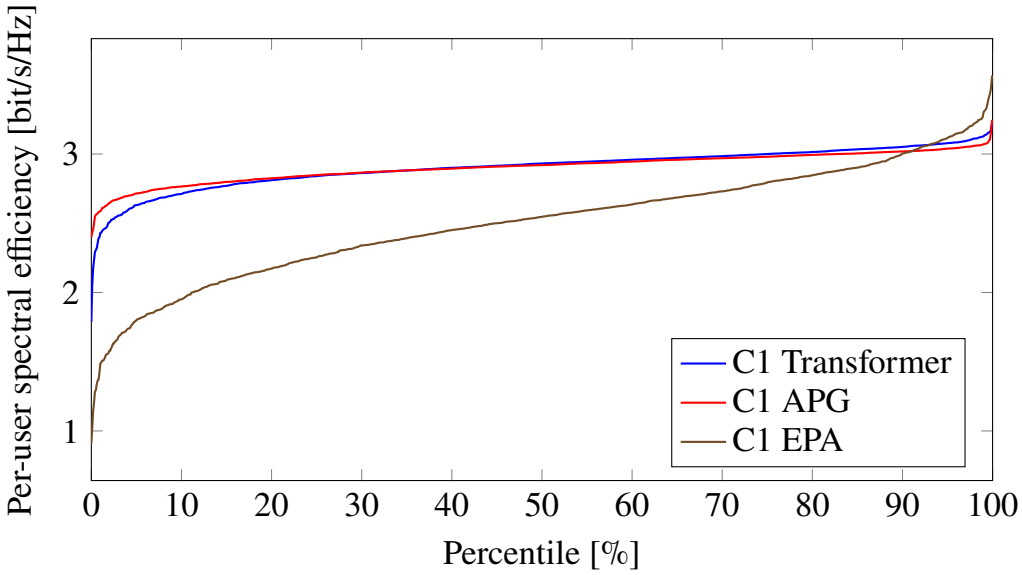


Figure 13: Quantile curves by percentile, Configuration C1.

Figure 14 shows the corresponding curves of per-user SEs for Configuration C2 with pilot reuse and stronger interference. As expected, pilot contamination reduces the absolute per-user SEs for all methods tested compared to SEs for Configuration C1, but the relative ordering remains the same. The Transformer and APG curves stay close over the entire range of percentiles and both provide a clear improvement over EPA.

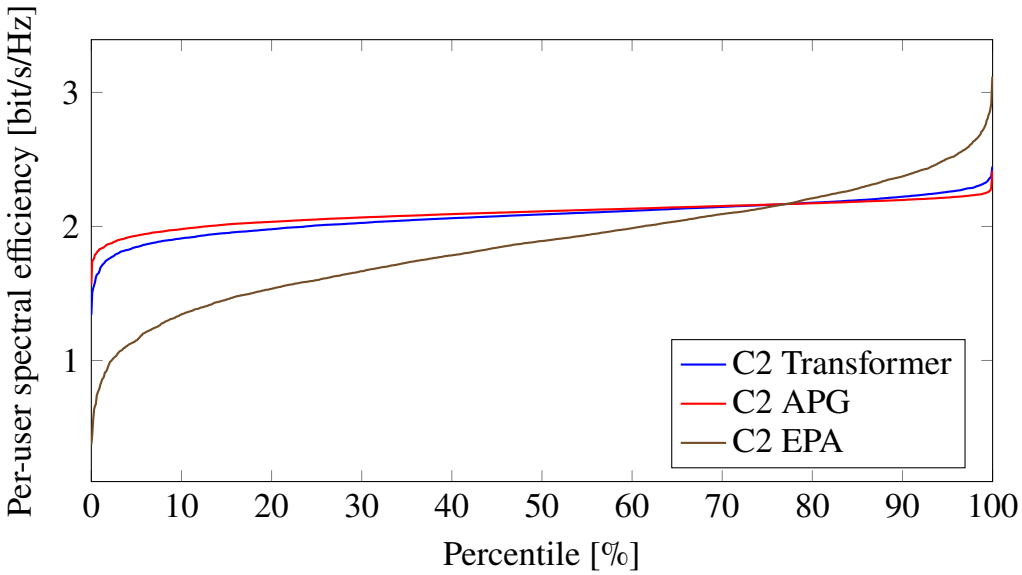


Figure 14: Quantile curves by percentile, Configuration C2.

Overall, the results for Configurations C1 and C2 show that the Transformer model achieves per-user SEs that are comparable to those of APG over a wide range of user percentiles, while both methods significantly outperform EPA. At the same time, the Transformer computes power-control coefficients in a single forward pass, so its computational cost per snapshot is much lower than that of iterative optimization with APG.

6 Conclusions

This thesis has investigated a transformer-based self-attention model for downlink power control in CFmMIMO systems. The model was trained in an unsupervised manner on a generated dataset that consists of CFmMIMO network snapshots. Experimental results show that the model is able to achieve per-user SE that is comparable to established optimization-based methods, while providing a significantly lower inference time.

The transformer architecture is capable of exploiting large-scale fading information and pilot allocation patterns through self-attention, which allows it to capture the global interference structure in each snapshot. Even though training the model and tuning its hyperparameters require many iterations and a noticeable amount of computation time on a GPU cluster, this cost is incurred only offline. Once training is completed, the controller maps each new snapshot to power control coefficients with a single forward pass and with predictable latency. These findings support attention-based models as a practical and scalable solution for real-time resource optimization in CFmMIMO systems.

6.1 Future research directions

The present work focuses on power control in a limited set of simulated CFmMIMO scenarios and relies on a standard analytical CFmMIMO model based on large-scale fading and pilot-based channel estimation, without explicit hardware impairments. Since the transformer architecture is flexible with respect to the input representation, a natural next step is to extend the same attention-based modelling approach to a broader set of CFmMIMO

tasks. In particular, similar architectures could be adapted to pilot allocation, user association, and channel estimation, or to joint designs where power control is learned together with one of these tasks.

Another direction is to explore alternative model designs, training objectives, and data generation strategies. Variants of the transformer with modified input preprocessing, different depth or width, or hybrid architectures that combine attention with other neural network modules could be evaluated. On the data side, the current work relies entirely on synthetic snapshots with a fixed set of assumptions. Future work could investigate more diverse synthetic scenarios that better cover realistic propagation conditions and network configurations. It is also relevant to combine such synthetic data with a limited amount of real-world measurements.

References

[1] W. Saad, M. Bennis, and M. Chen, "A Vision of 6G Wireless Systems: Applications, Trends, Technologies, and Open Research Problems" *IEEE Network*, vol. 34, no. 3, pp. 134–142, 2020.

[2] Y.-F. Liu, T.-H. Chang, M. Hong, Z. Wu, A. M.-C. So, E. A. Jorswieck, and W. Yu, "A Survey of Recent Advances in Optimization Methods for Wireless Communications," *IEEE J. Sel. Areas Commun.*, 2024.

[3] E. Björnson, L. Sanguinetti, and J. Hoydis, "Scalable Cell-Free Massive MIMO Systems," *IEEE Trans. Commun.*, vol. 68, no. 7, pp. 4247–4261, 2020.

[4] L. Sanguinetti, E. Björnson, and J. Hoydis, "Towards Massive MIMO 2.0: Understanding Spatial Correlation, Interference Suppression, and Hardware Limitations," *Found. Trends Signal Process.*, vol. 14, no. 3–4, pp. 162–472, 2020.

[5] Q. Mao, F. Hu, and Q. Hao, "Deep Learning for Intelligent Wireless Networks: A Comprehensive Survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2595–2621, 2018.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Proc. *Advances in Neural Inf. Process. Systems (NeurIPS)*, vol. 30, 2017, pp. 5998–6008.

[7] H. Q. Ngo, A. Ashikhmin, H. Yang, E. G. Larsson, and T. L. Marzetta, "Cell-Free Massive MIMO Versus Small Cells," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1834–1850, 2017.

[8] M. Farooq, L. Sanguinetti, and E. Björnson, "Utility Maximization for Large-Scale Cell-Free Massive MIMO Downlink," *IEEE Trans. Commun.*, vol. 69, no. 10, pp. 6774–6788, 2021.

[9] A. K. Kocharlakota, S. A. Vorobyov, and R. W. Heath Jr, "Attention Neural Network for Downlink Cell-Free Massive MIMO Power Control," in *Proc. 56th Asilomar Conf. Signals, Systems, and Computers*, 2022, pp. 738–743.

[10] N. Maslej, L. Fattorini, R. Perrault, J. Zhang, E. Brynjolfsson, J. Etchemendy, T. I. Lyons, and F.-F. Li, "AI Index Report 2025," *Stanford Institute for Human-Centered Artificial Intelligence (HAI),* Stanford University, Tech. Rep., 2025. [Online]. Available: https://aiindex.stanford.edu/report/

[11] R. Eldan and O. Shamir, "The Power of Depth for Feedforward Neural Networks," in *Proc. 29th Conf. Learning Theory (COLT)*, 2016, pp. 907–940.

[12] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2015.

[13] M. Mohri, F. C. N. Pereira, and M. Riley, "Weighted Finite-State Transducers in Speech Recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[14] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4905–4913, 2016.

[15] Z. C. Lipton, "A Critical Review of Recurrent Neural Networks for Sequence Learning," arXiv preprint arXiv:1506.00019, 2015.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.

[17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," *OpenAI Technical Report*, 2019.

[18] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, and others, "An Image Is Worth 16×16 Words: Transformers for Image Recognition at Scale," *International Conference on Learning Representations (ICLR)*, 2021.

[19] D. Luan and J. Thompson, "Channelformer: Attention Based Neural Solution for Wireless Channel Estimation and Effective Online Training," *IEEE Trans. Wireless Commun.*, vol. 22, no. 11, pp. 8243–8257, 2023.

[20] Y. Yang, L. Zhang, Y. Wang, J. Xu, and X. Li, "Transformer Learning-Based Efficient MIMO Detection Method," *J. Electr. Comput. Eng.*, vol. 88, pp. 102032, 2025.

[21] A. K. Kocharlakota, S. A. Vorobyov, and R. W. Heath Jr, "Pilot Contamination Aware Transformer for Downlink Power Control in Cell-Free Massive MIMO Networks," arXiv preprint arXiv:2411.19020, 2024.

[22] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A Survey of Transformers," *AI Open*, vol. 3, pp. 111–132, 2022.

[23] E. Björnson, L. Sanguinetti, H. Wymeersch, J. Hoydis, and T. L. Marzetta, "Massive MIMO is a Reality — What is Next? Five

Promising Research Directions for Antenna Arrays," *arXiv preprint arXiv:1902.07678*, 2019.

[24] J. Kassam, D. Castanheira, A. Silva, R. Dinis, and A. Gameiro, "A Review on Cell-Free Massive MIMO Systems," *Electronics*, vol. 12, no. 4, p. 1001, 2023.

[25] S. Buzzi and C. D'Andrea, "Cell-Free Massive MIMO: User-Centric Approach," *IEEE Wireless Commun. Lett.*, vol. 6, no. 6, pp. 706–709, 2017.

[26] E. A. Kadir, R. Shubair, S. K. Abdul Rahim, M. Himdi, M. R. Kamarudin, and S. L. Rosa, "B5G and 6G: Next Generation Wireless Communications Technologies, Demand and Challenges," in *Proc. 2021 Int. Congr. Adv. Technol. Eng. (ICOTEN)*, 2021, pp. 1–6, doi: 10.1109/ICOTEN52080.2021.9493470.

[27] V. Mai, N. Dao, and B. An, "Energy Efficiency Maximization in Large-Scale Cell-Free Massive MIMO: A Projected Gradient Approach," *IEEE Trans. Wireless Commun.*, vol. 21, no. 10, pp. 8725–8739, 2022.

[28] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, 2nd ed., Springer, 2004.

[29] Aalto Scientific Computing, "Triton cluster," *Aalto Scientific Computing (ASC),* Aalto University, Online documentation, 2025. [Online]. Available: https://scicomp.aalto.fi/triton/

[30] Aalto Scientific Computing, "Slurm on Triton," *Aalto Scientific Computing (ASC),* Aalto University, Online documentation, 2025. [Online]. Available: https://scicomp.aalto.fi/triton/tut/slurm/

[31] S. A. Weil, A. Leung, S. A. Brandt, E. L. Miller, and C. Maltzahn, "RADOS and Beyond: A Framework for Building Distributed Storage

Systems," *USENIX Conference on File and Storage Technologies (FAST)*, 2014.

[32] SchedMD LLC, "sbatch — Slurm Workload Manager," *SchedMD LLC,* Online documentation, 2025. [Online]. Available: https://slurm.schedmd.com/sbatch.html