DEGREE PROJECT IN MACHINE LEARNING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2024

Traffic Prediction in 4G/5G Networks using Time Series Transformers

ROMAIN TROST

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Romain Trost (rtrost@kth.se)
Department of Intelligent Systems
KTH Royal Institute of Technology

Host Company

Global AI Accelerator – GAIA, Ericsson AB
Stockholm, Sweden

Company Supervisors

Tahar Zanouda (tahar.zanouda@ericsson.com)
Henrik Holm (henrik.holm@ericsson.com)
Global AI Accelerator – GAIA, Ericsson AB

Examiner

Pawel Herman (paherman@kth.se)
Division of Computational Science and Technology
KTH Royal Institute of Technology

Supervisor

Hossein Azizpour (azizpour@kth.se)
Divison of Robotics, Perception and Learning
KTH Royal Institute of Technology

Abstract

Predicting traffic in 4G/5G networks is a crucial task for many applications in the telecom domain such as energy saving and proactive ressource management. Current state-of-the-art methods for traffic prediction involve the use of recurrent networks such as RNN/LSTM. However, these architectures are inherently prone to forgetting parts of longer input sequences, as they place a bias on recent historical points (vanishing gradient problem). This makes these architectures unable to model the prominent long-term patterns (i.e: daily, weekly, annual...) that telecom network data often exhibits.

To address this issue, we propose novel methods for traffic prediction that leverage a class of emerging time series models known as Time Series Transformers (TSTs). These models aim to incorporate the powerful long-term modeling capabilities of transformers into the time series domain by renovating Transformer's architecture with three components: (1) an enhanced positional encoding module, (2) an efficient attention mechanism, (3) a deep decomposition architecture. Together, these components enable TSTs to capture both short and long-term dependencies in time series data.

In this work, we evaluate the forecasting performance of four different TST models trained on three diverse real-world telecom network datasets. Furthermore, we study the feasibility of employing TSTs in a practical setting, by measuring their training time, inference time, and memory costs. Based on our experiments, we find that TSTs generally outperform LSTM in terms of prediction accuracy, particularly when it comes to predicting longer sequences into the future. Moreover, TSTs typically have a shorter training time than LSTM, whilst having slightly longer inference times and significantly larger memory requirements.

Keywords

Time Series Forecasting, KPI Prediction, 4G/5G, Time Series Transformers, LSTM

Abstract

Att förutsäga trafik i 4G/5G-nätverk är en avgörande uppgift för många applikationer inom telekomdomänen som energibesparing och proaktiv resurshantering. Nuvarande toppmoderna metoder för trafikprediktion involverar användning av återkommande nätverk såsom RNN/LSTM. Emellertid är dessa arkitekturer till sin natur benägna att glömma delar av längre inmatningssekvenser, eftersom de placerar en fördom på nyare historiska punkter (problem med försvinnande gradient). Detta gör att dessa arkitekturer inte kan modellera de framträdande långsiktiga mönstren (dvs. dagligen, veckovis, årlig...) som telekomnätverksdata ofta uppvisar.

För att lösa detta problem föreslår vi nya metoder för trafikförutsägelse som utnyttjar en klass av nya tidsseriemodeller som kallas Time Series Transformers (TSTs). Dessa modeller syftar till att införliva transformatorers kraftfulla långtidsmodelleringsförmåga i tidsseriedomänen genom att renovera Transformers arkitektur med tre komponenter: (1) en förbättrad positionskodningsmodul, (2) en effektiv uppmärksamhetsmekanism, (3) en djup nedbrytning arkitektur. Tillsammans gör dessa komponenter det möjligt för TST:er att fånga både kort- och långsiktiga beroenden i tidsseriedata.

I detta arbete utvärderar vi prognostiseringsprestanda för fyra olika TST-modeller som tränats på tre olika datauppsättningar för telekomnätverk i verkliga världen. Dessutom studerar vi möjligheten att använda TST i en praktisk miljö, genom att mäta deras träningstid, slutledningstid och minneskostnader. Baserat på våra experiment finner vi att TSTs generellt överträffar LSTM när det gäller förutsägningsnoggrannhet, särskilt när det gäller att förutsäga längre sekvenser in i framtiden. Dessutom har TST:er vanligtvis en kortare träningstid än LSTM, samtidigt som de har något längre slutledningstider och betydligt större minneskrav.

Nyckelord

Tidsserieprognoser, KPI-prediktion, 4G/5G, Time Series Transformers, LSTM

Acknowledgements

I want to convey my profound gratitude to Tahar Zanouda and Henrik Holm, my supervisors at Ericsson, for their invaluable guidance, unwavering patience, and continuous support throughout the entirety of this degree project. Furthermore, I would like to extend my gratitude to my KTH supervisor, Hossein Azizpour, and KTH examiner, Pawel Herman, who provided invaluable assistance and prompt feedback, particularly during the initial and closing stages of the project, offering their expertise and guidance.

Acronyms

3GPP	3rd Generation Partnership Project
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
CNN	Convolutional Neural Network
DDA	Deep Decomposition Architecture
FEA	Frequency Enhanced Attention
FEB	Frequency Enhanced Block
FES	Frequency Enhanced Structures
FFN	Feed Forward Network
KPI	Key Performance Indicator
LSTF	Long Sequence Time Series Forecasting
LSTM	Long Short-Term Memory
LTE	Long-Term Evolution
MAE	Mean Absolute Error
ML	Machine Learning
MHA	Multi-Head Attention
MSE	Mean Squared Error
MSSC	Multi-Scale Segment Correlation
NLP	Natural Language Processing
NoCU	Number of Connected Users
RNN	Recurrent Neural Network
SC	Segment Correlation
TFT	Temporal Fusion Transformer
TST	Time Series Transformer
UE	User Equipment

Contents

Chapter 1

Introduction

The last decade has seen an unprecedented increase in demand for wireless connectivity, caused primarily by the rapid urbanisation in many parts of the world coupled with the rise of mobile and IoT device usage. To accommodate for this ever-increasing demand, telecom operators are having to undergo a comprehensive evolution in their strategies, technologies and infrastructure. One challenge in particular that operators are facing today, is in their ability of delivering fast, low-latency 4G/5G cellular coverage to large geographical areas, whilst keeping their energy footprint and costs low. Effectively tackling this challenge is essential for operators to guarantee optimal network performance, enhance user experience, and promote sustainability in the telecom industry.

A crucial part in overcoming this challenge involves optimizing the utilisation of the different resources within the network through proactive resource management. One effective way of achieving this is by employing time series forecasting methods [29]. For instance, with the ability to accurately forecast the incoming traffic of a specific cell ahead of time, operators can dynamically adjust the utilization of that cell within the network by either increasing or decreasing its capacity as needed. This method, typically referred to as traffic prediction, is commonly used in the telecom domain to optimize resource utilization and reduce energy consumption and costs.

Traffic prediction in mobile networks is a well-researched area in the telecom industry. Older works commonly tackle this problem using simple statistical methods (e.g: ARIMA [12, 45]), which have been shown to provide satisfactory forecasting capabilities whilst being easy to use and implement. However, recent advancements in

technology have enabled the execution of more complex computations on a large scale, which has opened the doors for the adoption of Machine Learning (ML) methodologies. Recent works [2, 27, 43, 46] have shown that using ML-based time series models such as Long Short-Term Memory (LSTM) [24], tend to provide superior performance for the task of traffic prediction.

Most recently, a notable development in the field of ML was the introduction of Transformer [28] by Google researchers in 2017. Transformer has demonstrated state-of-the-art performance in tackling many time series problems [18, 42, 49], which were previously dominated by the use of LSTM, particularly in the Natural Language Processing (NLP) domain. Consequently, this breakthrough has sparked numerous efforts to apply Transformer-inspired models to the time series forecasting domain, giving rise to Time Series Transformer (TST) [31].

In this work, we build and evaluate the performance of different novel TST architectures for traffic prediction. We train the models using traffic data collected from three different cells deployed in diverse locations. To gauge their performance, we compare these models against the current domain-dominant approach that is LSTM. Specifically, we study the performance of each model according to its prediction accuracy, training time, inference time and memory requirements.

1.1 Problem Statement

Optimizing the utilization of resources inside a telecom network can lead to substantial benefits for operators, such as improving energy efficiency while maintaining QoS (Quality of Service), which is a prioritized concern in the industry. For instance, inside a telecom network, each User Equipment (UE) connects to one of the network's BS's (Base Station) radio units, which we refer to as cells in this work. A UE is only able to connect to a cell if it is inside that cell's coverage area, where multiple cells can sometimes have overlapping coverage areas. However, during off-peak traffic hours, the excessive power consumption in relation to demand arises due to cells being configured to operate as if under peak traffic scenarios, causing minimal variation in the network's power consumption with respect to the number of connected UEs.

Several approaches can be employed to help improve a network's energy efficiency. One such approach could be to dynamically modify the utilization of individual cells

depending on the amount of UEs that are connected to it. Another approach could be to, during off-peak traffic hours, offload connected UEs to fewer cells, by deactivating cells that have overlapping coverage areas for example. These approaches would enable the deactivation of redundant radio capacity and reducing the overall power consumption of the network. However, these approaches require robust and accurate traffic prediction methods to avoid compromising QoS due to reducing the network capacity. Furthermore, these approaches would have to be applied on a large scale to achieve any desirable impact, which requires the underlying traffic prediction models to have low computational complexity, thereby keeping their energy and operational costs low.

The recurrent LSTM architecture has proven to perform well at capturing temporal dependencies in telecom traffic data, making it a commonly used model for traffic prediction today [2, 27]. However, Transformer-based models have recently shown their ability to outperform LSTM when it comes to many sequence modelling problems. Furthermore, TSTs have emerged and shown impressive performance in many time series forecasting problems that were previously dominated by the use of LSTM [31]. It is therefore worth investigating whether TSTs can provide improved performance over the current domain dominant method of LSTM in the context of traffic prediction in telecom. It is noteworthy however, that Transformer-based models typically have a higher number of parameters and operations than LSTM which makes them more computationally demanding. It is therefore also of interest to investigate whether their potential increased computational complexity can still allow them to be feasibly deployed at large scale in real-world scenarios. To this end, this degree project aims to answer the following research question:

How do Time-Series Transformers compare to the current state-of-the-art LSTM network when predicting traffic in telecom, and furthermore, how feasible is it to employ them in a practical setting when considering computational costs?

1.2 Purpose and Goal

The purpose of this degree project is twofold. Firstly, this project seeks to contribute to the existing knowledge repository of attention-based models in the context of time series prediction. Secondly, this project aims to advance the current research relating

to traffic prediction in telecom networks. Through these, the project aims to shed light on the applicability of TSTs in mobile traffic prediction, potentially leading to more performant traffic prediction architectures that can help promote energy efficiency and sustainability within telecom networks.

The goal of this project is to answer the research question presented in Section 1.1. This will be achieved by building and assessing different novel TST architectures trained on real-world telecom traffic data. The evaluation process will encompass various evaluation metrics, including prediction accuracy, training time, inference time and memory requirements, which will be measured and then contextualized in the form of a practical setting.

1.3 Delimitations

This project focuses on introducing and assessing novel TST models for traffic prediction in 4G and 5G telecom networks. Thus, this project solely utilizes data taken from 4G and 5G cells, and doesn't consider any other types of telecom data. Moreover, due to time and computational resource constraints, conducting an extensive hyperparameter grid search was not feasible. The primary goal of this work is not to provide an optimal model, but rather to explore and benchmark the performance of a newly emerged class of models that are novel to the specific application domain.

1.4 Outline

The structure of this degree project is as follows. Chapter 2 presents a literature review covering the current research on traffic prediction in telecom, along with an introduction to the concept of time series modelling. This chapter also contains a detailed introduction of the various models that are implemented for the experimental work (i.e: LSTM and Transformer-based models). Chapter 3 focuses on the experimental work, outlining the datasets that were used and the experiments that were performed. The experimental findings are discussed in Chapter 4, followed by the conclusions and suggestions for further work in Chapter 5.

Chapter 2

Background

2.1 Traffic Prediction in Telecom Networks

A telecom operational network consists of interconnected sites located in different regions across a large area, like a city or town. Each of these sites consists of cells that provide 4G/5G network coverage in a geographic area, often referred to as a sector coverage area. A device such as a smartphone or laptop, otherwise known as a UE, can gain access to a telecom network by connecting to one of its cells if found inside the cell's sector coverage area.

Each cell has different kinds of data that can be monitored to assess the behavior of the network, where each data type is known as a Key Performance Indicator (KPI). According to the 3rd Generation Partnership Project (3GPP), which is a global consortium that develops and maintains standards for mobile communication networks, there exists different categories of KPIs, such as accessibility, mobility, integrity, utilization and energy performance [1]. In this work, we are mainly interested in accessibility KPIs that are related to measuring services requested by users. These include measurements like:

1. **Downlink Traffic**: The amount of data that flows from a cell to the UEs.
2. **Uplink Traffic**: The amount of data that flows from the UEs to a cell.
3. **Number of Connected Users (NoCU)**: The number of UEs that are connected to a cell.

Predicting these KPIs in advance can be useful for many applications in telecom such

as energy saving and proactive resource management. For example, the amount of energy needed to power a cell is highly correlated to the amount of traffic that it experiences. Thus, being able to accurately predict a cell's traffic beforehand can help with optimizing the cell's power scheduling. The benefit of this is twofold. Firstly, knowing to provide reduced power to a cell when traffic is low can help save energy costs. Secondly, knowing to provide increased power to a cell when traffic is high can help improve the overall user experience by ensuring that no UE loses connection to the cell. With telecom networks requiring more and more energy to operate due to the growing demand for data and connectivity, the importance of having an effective method for predicting these KPIs has become increasingly apparent.

The behavior of these KPIs can vary depending on a few factors, one of which is the location in which the cell is deployed. For example, cells in urban areas might exhibit higher traffic patterns than those in rural areas due to a larger concentration of people being outside and using their smartphones. Nevertheless, these KPIs all generally follow predictable patterns, which we will dive into further detail in the next section.

2.2 Time Series Modelling

Accessibility KPIs, in essence, represent a sequence of measurements that have been taken through time. This is otherwise known as time series data, which can be formally defined as a sequence of discrete chronological data points, typically evenly spaced and known as time steps. The task of modelling time series data involves three main application domains: forecasting, anomaly detection, and classification. In our case, accessibility KPI prediction is a time series forecasting problem.

Time series forecasting is a sequence-to-sequence problem, where the goal is to use a sequence of known historical time steps to predict a sequence of unknown future time steps. Specifically, we have an input $\mathcal{X}^t = \left\{\mathbf{x}_1^t, \ldots, \mathbf{x}_{L_x}^t \mid \mathbf{x}_i^t \in \mathbb{R}^{d_x}\right\}$ at time t, and the output is to predict a corresponding sequence $\mathcal{Y}^t = \left\{\mathbf{y}_1^t, \ldots, \mathbf{y}_{L_y}^t \mid \mathbf{y}_i^t \in \mathbb{R}^{d_y}\right\}$. The input length L_x and output length L_y can have different values and typically vary from use-case to use-case. The input feature dimension d_x and output feature dimension d_y determine the type of forecasting problem we are dealing with, which can take three main forms [39]:

1. **Univariate to univariate** ($d_x = d_y = 1$): This is the simplest case, where both the input and output sequences come from the same time series.

2. **Multivariate to univariate** ($d_x > 1, d_y = 1$): In this scenario, multiple time series are used to predict one time series. This can be useful when the predicted time series is dependant on additional external factors or features. For example in our case, the traffic that a cell experiences at a given time might be highly correlated to the NoCU at that time. It is therefore feasible to assume that allowing our model to utilize both the traffic and NoCU time series as input can enable it to make more accurate traffic predictions.

3. **Multivariate to multivariate** ($d_x = d_y > 1$): This case is similar to the first one, the key difference being that a single model is used to tackle multiple univariate to univariate forecasting problems. This can result in the predicted time series potentially co-varying as a result of being correlated to the same input. This method can be very attractive in practical settings, as it offers the convenience of only having to train one model to predict multiple time series, which can lead to lower overall training times and memory costs.

A key property of time series data is that each time step depends on those that came before it. It is commonly assumed that a given time step depends mostly on the recent time steps that precede it. For instance, predicting the weather at a given hour will depend more on the weather of hours that came just before it, than the weather that occurred further back in time. However, time series data commonly has a seasonality aspect. For example, the weather at a given time during the year is likely to be similar to the weather that occurred at the same time of the year prior. Time series data also often has a trend aspect, where in this example, global warming might affect the weather over time. Formally, we can describe a time series using its two key components [36]:

- **Trend**: describes the overall direction the time series follows over time.
- **Seasonality**: represents the intricate seasonal patterns (i.e: repeating patterns) that happen on lower time frames.

Accessibility KPIs in telecom exhibit seasonal patterns at different time scales that can be exploited to make more accurate forecasts. For instance, a cell's downlink traffic often contains strong daily patterns, where traffic is typically low during the night and morning when people are more likely to be indoors either working or

sleeping, and peaks in the evening when people are more likely to be outside using their smartphones. There are also weekly patterns where for example on Sundays, there are typically less people outside during the day when compared to other weekdays, which can lead to lower traffic at those times. There are also annual patterns, where traffic patterns may exhibit similar behaviour during annual holidays, or special social events such as concerts.

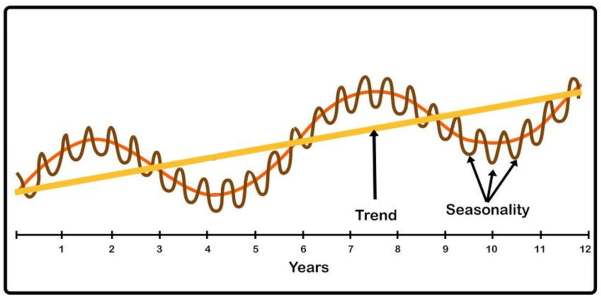


Figure 2.2.1: Seasonal and trend components in time series data. Seasonality can be present at different scales as shown by the orange and brown lines. [21]

2.2.1 Seasonal-Trend Decomposition

It is common practice when tackling time series forecasting problems to predict both the trend and seasonality components separately, before adding them back together to form the final prediction. This is achieved by first decomposing the time series into its trend and seasonality components, where the trend is extracted by applying a moving average over the time series, and the seasonality component is extracted by subtracting the original time series by its trend component. Specifically, we describe the series decomposition operation as:

$$\mathcal{X}_t, \mathcal{X}_s = \text{SeriesDecomp}(\mathcal{X})$$

with

$$\mathcal{X}_t = \text{AvgPool}(\text{Padding}(\mathcal{X}))$$
$$\mathcal{X}_s = \mathcal{X} - \mathcal{X}_t$$

where, $\mathcal{X}$ is the original time series and $\mathcal{X}_t$, $\mathcal{X}_s$ represent its trend and seasonality components, respectively. The $\text{AvgPool}(\cdot)$ applies a moving average with the padding operation to keep the series length unchanged.

2.2.2 Training Time Series Models

Deep learning models are today considered the state-of-the-art when it comes to tackling time series forecasting problems. To train and evaluate these models, the time series dataset must be split into individual data points to form a training and testing set. This is commonly achieved by sliding a fixed size window over the entire dataset, as shown in Figure 2.2.2. The earliest data points are typically used for training, and the latter ones for testing.

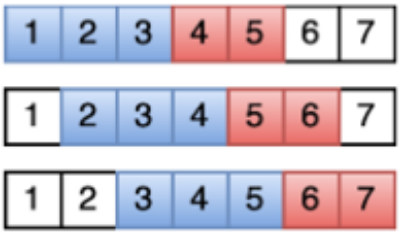


Figure 2.2.2: Sliding window approach to extract individual data points from a time series dataset. Each row corresponds to one data point, which includes its input sequence (blue) and output sequence (red). [7]

The decision on what input and output lengths to use must therefore be made before training occurs. Typically, the output length is decided first, as this value depends directly on the use-case in which we are forecasting for, particularly how frequently we want to take actions. Some applications benefit from forecasting values further into the future (i.e: sales/weather forecasting...), and some applications require more proactivity by making more frequent but smaller forecasts (i.e: stock market prediction...).

If we assume that the input sequence length is fixed, its length primarily depends on the seasonality of the data that is being used. For example, in the case of cell traffic data which exhibits strong daily patterns, it would be useful for the input sequence to be long enough (i.e: multiple days), such that the model can look back and use the traffic behaviour of previous days to make better and more informed predictions for the current day. Another factor that might affect the choice of the input length is the size of the dataset. When splitting the data using the sliding window method, the total number of samples that can be extracted is determined by: $datasetsize - (inputlength + outputlength) + 1$. Consequently, the longer the input sequence is, the fewer the amount of individual data samples can be extracted, which leads to having less data for training the forecasting model which can negatively affect performance. Therefore, choosing an

appropriate input sequence length is an important step when training deep learning forecasting models.

2.2.3 Performance Evaluation

There exists several metrics for evaluating the accuracy of time series predictions, with two of the most widely used ones being the Mean Squared Error (MSE) [38] and the Mean Absolute Error (MAE) [37].

The MSE computes the average of the squared error between the predicted time steps and the ground truth, as shown in Equation 2.1. By squaring the error, larger discrepancies carry greater significance compared to smaller ones. This characteristic renders the MSE an effective loss function for training, as it imposes harsher penalties on larger errors.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

The MAE takes the average of the absolute error between the predicted time steps and the ground truth, as shown in Equation 2.2.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.2}$$

Using both metrics together is useful to gauge the overall quality of a model's predictions, as MSE indicates how good the model is at predicting extreme points, whereas MAE gives a more objective view on how good the predictions are as it penalizes errors linearly instead of quadratically.

2.2.4 Common Time Series Models

There exists a variety of different time series models which continue to evolve to this day. An older and more traditional example is the Autoregressive (AR) model, a type of lag regression model, which is still being used today. However, there is a growing trend towards utilizing deep neural network models for making time series predictions, such as the LSTM network. The next sections provide an overview of both the AR and LSTM models.

2.2.4.1 Lag Regression Models

The earliest methods for predicting time series data involved using statistical models known as lag regression models, such as the AR model [34]. As its name implies, the AR model is ”self regressive”, meaning that it infers values based on previous values. It is defined as:

$$\hat{X}_t = b + \sum_{i=1}^{p} w_i X_{t-i} \tag{2.3}$$

where X_t represents the value at time t, p is the number of historical time steps to look at, w_i is the weight for the value i time steps back and b is a constant bias term. The weights and bias can be found using ordinary least squares.

There have been multiple variations of the AR model which incorporate the historical moving average of the time series to refine predictions, such as the Autoregressive Moving Average (ARMA) and Autoregressive Integrated Moving Average (ARIMA) models [33, 35]. Although they can prove effective for modelling simpler time series data, they tend to over reproduce the data’s mean and thus don’t perform as well when dealing with more complex time series data.

2.2.4.2 Long Short-Term Memory Network

With the advent of deep learning, it has become increasingly popular to tackle time series problems using deep learning models. The Recurrent Neural Network (RNN) [24] is a type of deep learning neural network architecture used for processing sequential data, such as text, speech, or time series data. An RNN is made up of a series of units stacked one after another, where each unit is used to process a given time step in a sequence. Each unit consists of an input neuron, an output neuron, and a hidden state. For each unit, the output depends on the current input, in addition to the hidden states of previous units. This allows the RNN to retain an internal memory of previous time steps, enabling it to learn patterns and create dependencies between different parts of a sequence.

One downside of the RNN, is that it faces a phenomenon known as the vanishing gradient problem [11]. This occurs during the backpropagation phase of the training process, where gradients get multiplied by each other as the sequence progresses back in time, causing gradients to become very small and eventually ”vanish”. This prevents

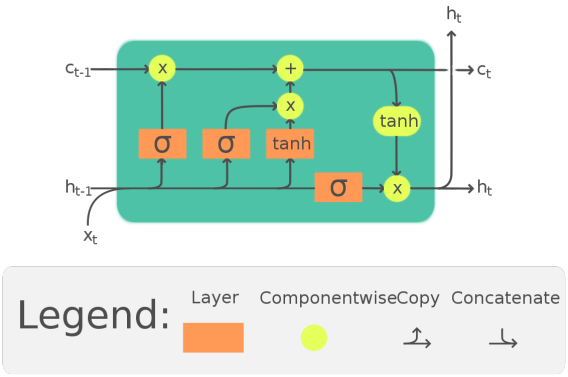


Figure 2.2.3: Illustration of an LSTM cell. Cells are stacked one after another to form a complete LSTM structure. [6]

the RNN from being able to update the weights of its earlier units when dealing with long sequences. In other words, RNN is unable to model the long-term dependencies often found in sequential data as it tends to ”forget” information from the earlier parts of a sequence.

The LSTM network, illustrated in Figure 2.2.3, renovates the RNN by introducing a gating mechanism which allows it to selectively retain or discard information over time. This enables LSTM to learn and store information for a longer period of time, making it better suited for modelling long-term dependencies. Unfortunately, LSTM does eventually fall into the vanishing gradient problem, however it is generally able to model much longer sequences than RNN [24].

2.2.5 Major Challenges

Prolonging the forecasting horizon is a critical demand for real applications, such as long-term energy consumption planning and weather early warning. However, a major challenge that time series models face today is their inability to make forecasts in the long run, namely Long Sequence Time Series Forecasting (LSTF). Existing methods are mostly designed under short-term problem settings, like predicting 48 time steps or less [17, 20, 32]. Increasing the prediction sequence length typically causes the models' prediction capacity to deteriorate. Figure 2.2.4 illustrates this with an empirical example showing the forecasting results on a real dataset, where an LSTM network is used to predict the hourly temperature of an electrical transformer station from the short-term (i.e: 12 steps) to the long-term (i.e: 480 steps). As the prediction length exceeds 48 steps (marked by the solid star), the LSTM's overall performance

deteriorates quickly, with the MSE rising to unsatisfactory levels and the inference speed dropping considerably.

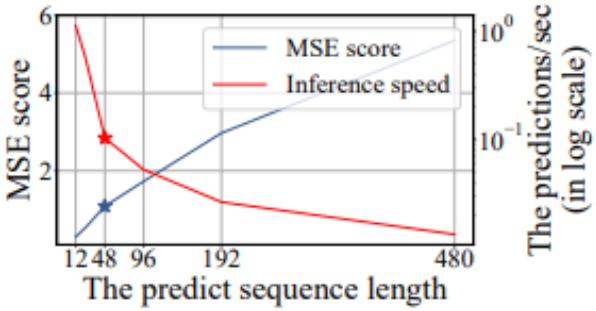


Figure 2.2.4: Empirical example showing the LSTF performance of an LSTM on a real dataset. The LSTF performance is limited due to the LSTM's prediction capacity. [48]

This calls for novel time series models that have increased prediction capacities. Recently, Transformer models have demonstrated superior performance in capturing long-term dependencies than LSTM. The next section introduces Transformer and its architectural components.

2.3 Transformer

The Transformer, developed by Google in 2017, is an architecture purposefully designed for sequence-to-sequence modelling. Unlike previous sequence-to-sequence models (RNN/LSTM), which utilize the concept of recurrence, Transformer relies solely on the attention mechanism to model dependencies between input and output. This allows Transformer to process inputs in parallel and overcome the vanishing gradient problem associated with recurrent architectures, which in turn, enables it to capture longer sequences with equal emphasis given to all points.

Although initially designed for the task of machine translation, Transformer has made significant impacts in various application domains, notably NLP [4, 8, 28] and Computer Vision [9, 22, 23]. Transformer's ability to capture long-range dependencies and interactions in sequential data has also made it attractive for time series modelling [31], achieving impressive performance in forecasting [16, 48, 49], anomaly detection [42], and classification [44] tasks.

In this section, we first give an overview of the architecture of the vanilla transformer proposed by Vaswani et al. [28]. Then, we introduce one of Transformer's biggest

limitation; its complexity. Subsequently, we summarize some of the key challenges Transformer faces when applied to the time series application domain.

2.3.1 Overview

Transformer has an encoder-decoder architecture, comprised of Multi-Head Attention (MHA) and Feed Forward Network (FFN) blocks stacked on top of each other. A graphical representation of Transformer is shown in Figure 2.3.1. The encoder is composed of 6 identical layers, each having two sub-layers. The first is a multi-head self-attention block and the second is a simple position-wise fully connected feed-forward network. The decoder is also composed of 6 identical layers, however in addition to the 2 sub-layers found in the encoder, between these it has a third one composed of a multi-head cross-attention block which performs MHA over the encoder's output. Each sub-layer includes a residual connection and layer normalisation, and can be described as:

$$x_{out} = LayerNorm(x + Sublayer(x)) \tag{2.4}$$

where x and x_{out} represent the input and output, respectively, and $Sublayer(x)$ is the sub-layer function; either MHA or FFN. We present the details of each sub-layer function below.

2.3.1.1 Multi-Head Attention

The attention mechanism is a key component of the Transformer, as it is what enables it to create dependencies between different parts of an input sequence. It uses a Query-Key-Value (**QKV**) model, where the matrices **Q** (queries), **K** (keys) and **V** (values) are the linear projection of a given input **X**, defined as:

$$\mathbf{Q} = \mathbf{X}\mathbf{W_Q}, \mathbf{K} = \mathbf{X}\mathbf{W_K}, \mathbf{V} = \mathbf{X}\mathbf{W_V} \tag{2.5}$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L \times D}$ and $\mathbf{W_{Q,K,V}} \in \mathbb{R}^{D \times D}$. L denotes the sequence length, and D is the dimension of the **Q**, **K**, **V** representations.

The attention, also called the *Scaled Dot-Product Attention*, is then computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{D}}\right)\mathbf{V} \tag{2.6}$$

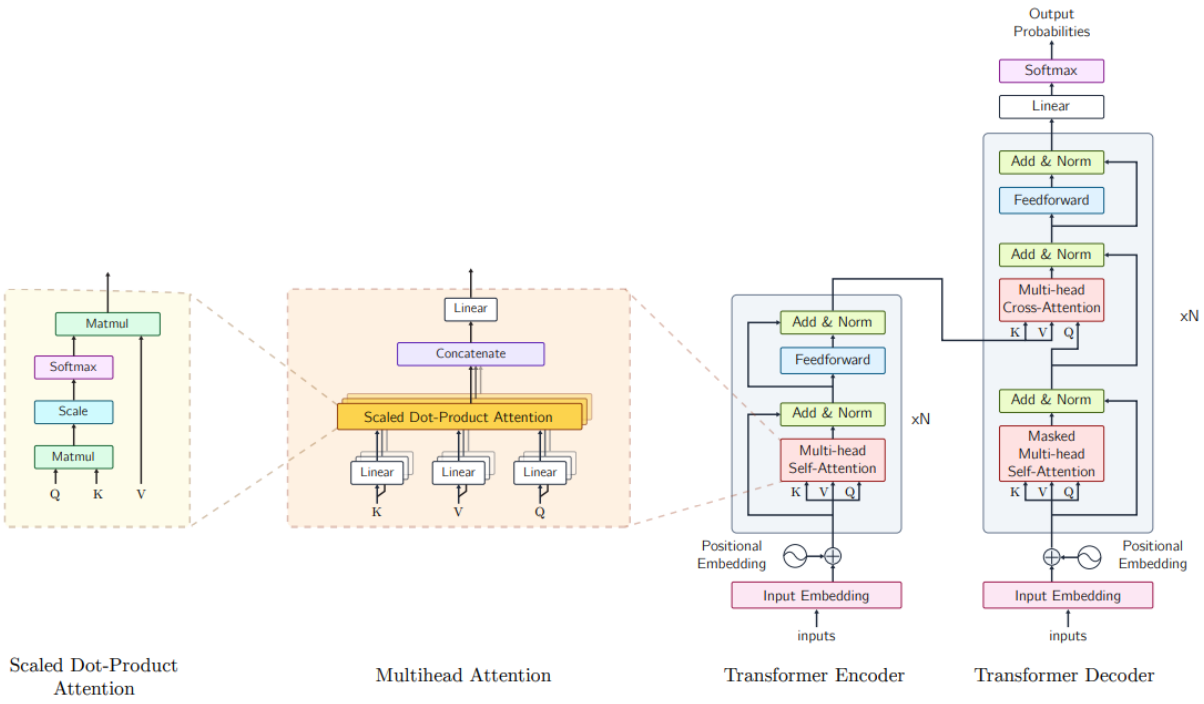


Figure 2.3.1: Structure of Transformer [28], comprised of several multi-head attention layers each employing the scaled dot-product attention.

where the scaling factor $\sqrt{D}$ is used to prevent the softmax function from being pushed into regions where it has extremely small gradients (i.e: vanishing gradient problem). In other words, the attention computes the weighted sum of the values by the compatibility between the queries and the keys.

Simultaneously performing the attention function on h different linear projections of **Q**, **K** and **V** (which are referred to as heads), and then concatenating the different, learned representations, creates the MHA block. MHA allows the model to jointly attend to information from different positions from different subspaces, enhancing the learned representations. It is formulated as:

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \left[\text{head}^1, \ldots, \text{head}^h\right] \mathbf{W}_O \tag{2.7}$$

where the braces $[\cdot]$ represent the vector concatenation of the heads defines as:

$$\text{head}^{\text{i}} = \text{Attention}\left(Q\mathbf{W}_Q^i, K\mathbf{W}_K^i, V\mathbf{W}_V^i\right)$$

where the projection matrices **W** are the trainable parameters of MHA.

2.3.1.2 Feed-Forward Network

Following the MHA module is a fully connected two-layer position-wise feed-forward network (FFN), used for converting the attention's output to the next layer. It consists of two linear transformations usually with a ReLU activation in between:

$$\text{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{2.8}$$

The linear transformation applied to each position is identical, but each layer uses their own unique trainable parameters: W_1 and W_2. The input/output has dimensionality $d_{model} = 512$ and the inner-layer has dimmensionality $d_{ff} = 2048$.

2.3.1.3 Positional Encoding

As Transformer has no recurrence or convolution, it does not take into account the positioning of tokens in an input sequence. This is undesirable as the order of tokens usually carries a lot of importance in sequence processing tasks. To solve this, positional information is encoded for each token and added in the input embeddings, which is then fed to Transformer.

Vanilla Transformer uses a form of *Absolute Positional Encoding*, which assigns positions using sinusoidal functions of different frequencies:

$$\begin{aligned} PE(pos, 2i) &= \sin\left(pos/10000^{2i/D}\right) \\ PE(pos, 2i+1) &= \cos\left(pos/10000^{2i/D}\right) \end{aligned} \tag{2.9}$$

where pos is the position and i is the dimension. There have been many efforts to improve this rudimentary encoding scheme, which we present in Section 2.4.1.

2.3.2 Complexity of Transformer

One of Transformer's biggest limitations is its time and memory complexity, which can create a bottleneck hindering its performance. The complexity of Transformer comes from both the self-attention mechanism and the FFN module. The self-attention and FFN modules have a quadratic complexity of $\mathcal{O}(L^2D)$ and $\mathcal{O}(LD^2)$, respectively. So, when $L >> D$, self-attention becomes the main bottleneck, and when $L << D$, the main bottleneck is caused by the FFN module. In practice, it is much more common for sequence modelling problems to have $L >> D$, which makes self-attention the most

common cause for Transformer's high computational complexity.

2.3.3 Transformer in Time Series

The Transformer was initially designed for NLP applications, where it has shown impressive performance in modelling long sequences making it the preferred method for tackling NLP tasks such as machine translation (i.e: Google Translate [28]), question-answering (i.e: BERT [8]), spam detection...). This has generated much interest in trying to extend the impressive performance of Transformer to the time series application domain.

Transformer, in its vanilla form, often shows unsatisfactory performance when it comes to tackling time series problems [30]. This can be attributed to the fact that time series and NLP data are quite different in nature. Specifically, time series data differentiates itself from NLP data in three key ways. Firstly, time series data comes with timestamp information not found in NLP data, which can be useful for understanding the global positioning of timesteps in a sequence. Secondly, time series data is often dependant on much longer input sequences due to periodicity. For example, if a times-series exhibits strong weekly patterns, then it would be useful to refer to the values of previous weeks for making predictions in the current week, which could lead to a very long historical sequence depending on how many weeks we want to look back to. And thirdly, unlike in NLP, time series data can be decomposed into trend and seasonality components, as described in Section 2.2.1.

In line with this, the taxonomy proposed by [31] and shown in Figure 2.3.2 attributes Transformer's unimpressive performance in time series to 3 main components:

1. The **Positional Encoding** in Transformer isn't able to fully leverage time series data and the extra features that come with it (i.e: calendar timestamps).
2. Transformer's **Attention Module** has a too high complexity to effectively model and utilize long input sequences.
3. Transformer requires **Architecture-Level** innovation to be able to capture the unique behaviour of time series data (i.e: trend and seasonality).

TST is a transformer-based model that addresses these architectural limitations by making key network modifications to the vanilla Transformer in order to better exploit the unique properties of time series data. In the next section, we present novel TST

models and the architectural adjustments that set them apart from the canonical Transformer.

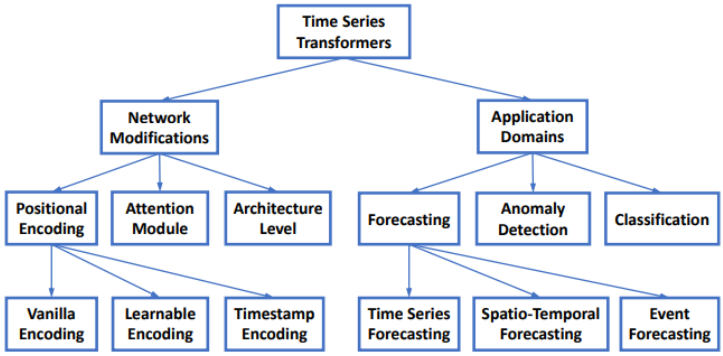


Figure 2.3.2: Taxonomy of TSTs from the perspective of network modifications and application domains [31].

2.4 Time Series Transformers

Time Series Transformers are transformer-based models that have been specifically adapted to tackle time series problems. There have been many different TST variations over recent years (see Table 2.4.1), each of which employs their own unique methods aiming to accommodate for the special challenges in time series described in the previous section.

TST Models	Release Date
LogFormer [16]	Jun. 2019
Temporal Fusion Transformer (TFT) [18]	Dec. 2019
Reformer [15]	Jan. 2019
Informer [48]	Dec. 2020
Autoformer [41]	Jun. 2021
Pyraformer [19]	Jan. 2021
ETSformer [40]	Feb. 2022
Preformer [10]	Feb. 2022
FEDformer [49]	May. 2022

Table 2.4.1: Different TST models (ordered chronologically).

Based on the current literature and some initial tests performed as part of this project, Autoformer, Preformer and FEDformer tend to consistently outperform their competition. For this reason, we focus mainly on these 3 models in this work, each of which will be presented in this section. One reason which could be indicative of their superior performance when compared to other models, is that they all

share a somewhat similar structure, with Preformer and FEDformer being heavily inspired from Autoformer's architecture. Specifically, they each make 3 key network modifications to the vanilla transformer. Namely, they employ:

1. An **Enhanced Positional Encoding** scheme, which leverages the timestamp information of the time series to extract a global representation of each time-step's position.

2. An **Efficient Attention Mechanism**, uniquely designed to effectively and efficiently model long-term dependencies in time series data.

3. A **Deep Decomposition Architecture**, which uses series-decomposition sub-modules to model and predict both the trend and seasonality components of the time series separately.

To provide a complete overview of how each of these 3 models operate, we first summarise how they tackle each of the 3 aforementioned network modifications, and subsequently, we describe their overall structure.

2.4.1 Enhanced Positional Encoding

As mentioned in Section 2.3.1.3, Transformer requires positional information to be embedded into the sequence that it wants to process. However, the vanilla encoding used in Transformer only accounts for the positioning of points relative to each other. It does not include any global positioning information which is quite prevalent in time series (i.e: time of day, day of week...), and can be useful for extracting the periodic patterns that may occur in the data. To remedy this, TSTs employ an enhanced positional encoding scheme which uses a combination of what is referred to as *learnable encoding* and *timestamp encoding*.

2.4.1.1 Learnable Encoding

This type of encoding learns appropriate positional embeddings directly from time series data. It has shown better performance than vanilla encoding thanks it its flexibility and ability to adapt to specific tasks and data. For example, TFT [18] uses an LSTM network to encode positional embeddings, aiming to better exploit the ordering of the input sequence.

2.4.1.2 Timestamp Encoding

Real-world time series data often comes with timestamp information, like calendar information (i.e: second, minute, hour, day, week...) or special events (i.e: holidays, concerts...). Including these timestamps in the positional encoding can be beneficial to help model periodic patterns at different time frames (i.e: hourly, daily, weekly...). Informer [48] leverages timestamps to capture the global properties of a time series using a learnable embedding layer, as shown in Figure 2.4.1. Autoformer [41], FEDformer [49] and Preformer [10] use a similar timestamp encoding scheme, but discard the positional encodings (Local Time Stamp), as their attention mechanism inherently captures the sequential information.

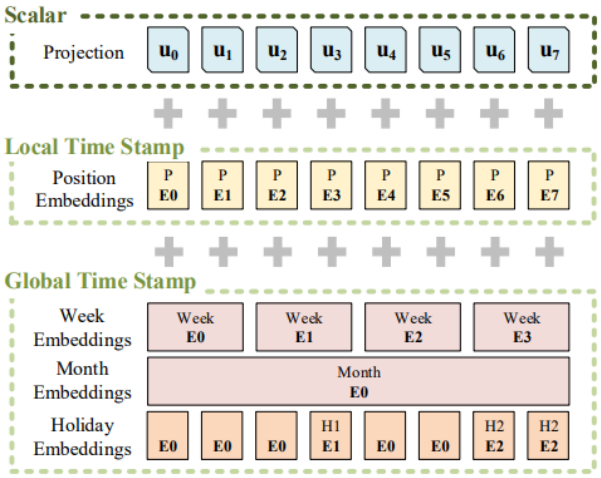


Figure 2.4.1: Input embedding used for Informer [48]. It consists of three separate parts, a scalar projection, the local time stamp (Position) and global time stamp embeddings (Minutes, Hours, Week, Month, Holiday...).

2.4.2 Efficient Attention Module

As seen in section 2.3.2, self-attention in Transformer has a quadratic complexity of $\mathcal{O}(L^2)$, which creates a computational bottleneck when dealing with long input sequences. This is because the three matrices (Q, K, V) used in self-attention are dynamically generated based on the pairwise similarity of input patterns. Obtaining these matrices thus requires a very large number of dot products, which scales quadratically with sequence length L.

Efficient attention modules in TSTs rely on the fact that not all parts of the input sequence are needed to make good predictions at a given time-step. They exploit the sequence's seasonality to only attend to its most relevant parts, reducing the

total number of dot-products needed which in turn, lowers the computational complexity.

Listed below are the efficient attention modules for Autoformer, Preformer and FEDformer, along with their time and memory complexities in Table 2.4.2:

- **Auto-Correlation**: Autoformer's attention module, which performs series-wise attention instead of point-wise.
- **Multi-Scale Segment Correlation (MSSC)**: Preformer's attention module, which performs attention on segments of a series instead of individual points.
- **Frequency Enhanced Structures (FES)**: FEDformer's attention module, which performs attention in the frequency domain instead of the time domain.

Self-Attention Methods	Training		Testing
	Time	Memory	Steps
Full-Attention (Transformer [28])	$\mathcal{O}(\mathrm{L}^2)$	$\mathcal{O}(\mathrm{L}^2)$	L
Auto-Correlation (Autoformer [41])	$\mathcal{O}(L \log L)$	$\mathcal{O}(L \log L)$	1
MSSC (Preformer [10])	$\mathcal{O}(L^2/\, L_0)$	$\mathcal{O}(L^2/\, L_0)$	1
FES (FEDformer [49])	$\mathcal{O}(L)$	$\mathcal{O}(L)$	1

Table 2.4.2: Complexity comparisons of the different attention modules in TSTs. L is the input sequence length and L_0 is the starting input sequence length unique to Preformer.

The following sections describe the operation of each attention mechanism.

2.4.2.1 Auto-Correlation

Autoformer employs a unique attention module called the *Auto-Correlation* mechanism. It is unique in that it uses series-wise connections as opposed to point-wise connections used in traditional self-attention (i.e: the queries (Q), keys (K) and values (V) are series instead of points). This is illustrated in Figure 2.4.2. Auto-Correlation works by using the autocorrelation between sub-series to discover period-based dependencies and then combines similar sub-series by time delay aggregation,

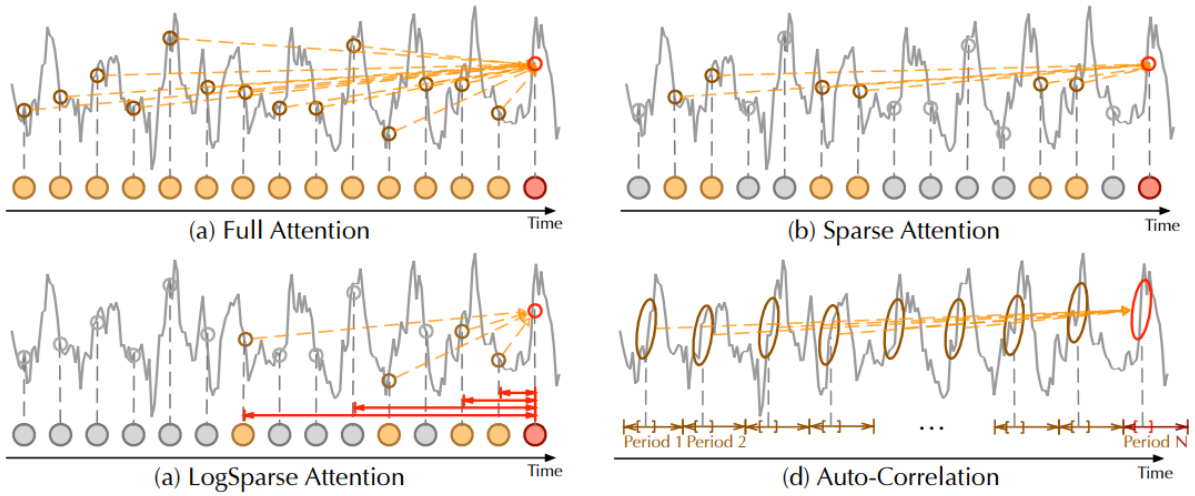


Figure 2.4.2: Auto-Correlation vs. traditional self-attention. Full Attention (a) attends to all points. Sparse Attention [15, 48] (b) attends to points based on a proposed similarity metrics. LogSparse Attention [16] attends to points with exponentially increasing intervals. Auto-Correlation [41] (d) attends to similar sub-series among underlying periods.

as shown in Figure 2.4.3. More specifically, it can be described as:

$$\begin{aligned} \tau_1, \cdots, \tau_k &= \underset{\tau \in \{1, \cdots, L\}}{\arg \operatorname{Topk}} \left(\mathcal{R}_{Q,K}(\tau) \right) \\ \widehat{\mathcal{R}}_{Q,K}\left(\tau_1\right), \cdots, \widehat{\mathcal{R}}_{Q,K}\left(\tau_k\right) &= \operatorname{SoftMax}\left(\mathcal{R}_{Q,K}\left(\tau_1\right), \cdots, \mathcal{R}_{Q,K}\left(\tau_k\right) \right) \\ \text{Auto-Correlation}\left(Q, K, V\right) &= \sum_{i=1}^{k} \operatorname{Roll}\left(V, \tau_i\right) \widehat{\mathcal{R}}_{Q,K}\left(\tau_i\right) \end{aligned} \tag{2.10}$$

where autocorrelation $\mathcal{R}_{\mathcal{X}\mathcal{X}}(\tau)$ represents the time-delay similarity between real discrete-time process $\mathcal{X}_t$ and its τ lag series $\mathcal{X}_{t-\tau}$ [5]:

$$\mathcal{R}_{\mathcal{X}\mathcal{X}}(\tau) = \lim_{L \to \infty} \frac{1}{L} \sum_{t=1}^{L} \mathcal{X}_t \mathcal{X}_{t-\tau} \tag{2.11}$$

Firstly, autocorrelation is applied between series Q and K for different time delays τ, where $\arg \operatorname{Topk}$ fetches τ for the k most correlated series: $\tau_1, \cdots, \tau_k$ (i.e: the most possible k period lengths of the input time series). Secondly, a normalised confidence score of the similarity between series Q and K for each $\tau_1, \cdots, \tau_k$ is calculated using softmax: $\widehat{\mathcal{R}}_{Q,K}\left(\tau_1\right), \cdots, \widehat{\mathcal{R}}_{Q,K}\left(\tau_k\right)$. This score represents the importance each period τ should be given when attended to by the model (i.e: the aggregation weight). Finally, the $\operatorname{Roll}(\cdot)$ operation applies a time-delay τ to the series V, which gets multiplied by the confidence score for that specific τ. This is done for each $\tau_1, \cdots, \tau_k$, where the resulting

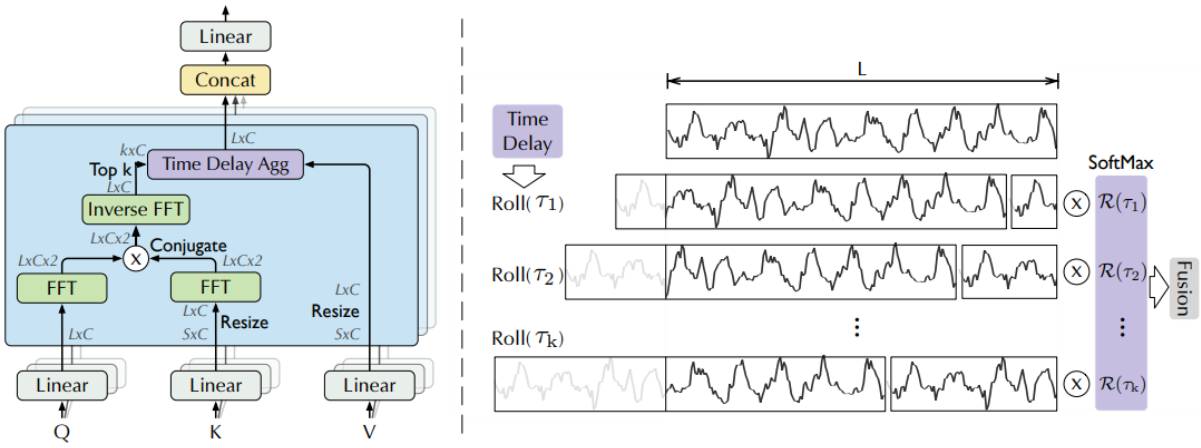


Figure 2.4.3: Auto-Correlation (left) and Time Delay Aggregation (right) [41].

k series are then aggregated together.

The original paper sets $k = \log L$, meaning $\mathcal{O}(\log L)$ series are aggregated in total. Furthermore, each series has length L, giving Auto-Correlation a computational complexity of $\mathcal{O}(L \log L)$.

2.4.2.2 Multi-Scale Segment Correlation

Segment Correlation (SC) is a key part of Preformer's attention module, which performs segment-wise attention instead of point-wise attention. First, Q, K and V are segmented into several segments having the same length L_{seg}: $\{Q_1, Q_2, \ldots, Q_m\}, \{K_1, K_2, \ldots, K_n\}, \{V_1, V_2, \ldots, V_n\}$, where $\boldsymbol{Q_i}, \boldsymbol{K_i}, \boldsymbol{V_i} \in \mathbb{R}^{L_{seg} \times D}$ and m, n denote the number of segments. Then SC is performed:

$$
\begin{aligned}
c_{ij} &= \text{Correlation}\left(Q_i, K_j\right) = \frac{1}{d \times L_{seg}} Q_i \odot K_j \\
\hat{c}_{i1}, \hat{c}_{i2}, \ldots, \hat{c}_{in} &= \text{Softmax}\left(c_{i1}, c_{i2}, \ldots, c_{in}\right) \\
Y_i &= \sum_{j=1}^{n} \hat{c}_{ij} V_j \\
\text{SC}\left(H; L_{\text{seg}}\right) &= \left[Y_1, \ldots, Y_m\right]
\end{aligned}
\tag{2.12}
$$

Here, c_{ij} is the correlation measurement between any Q_i, K_j segment pair. These measurements are then normalised using Softmax, giving the aggregation weights $\hat{c}_{i1}, \hat{c}_{i2}, \ldots, \hat{c}_{in}$ for each query segment Q_i. The output at the position of the i-th segment Y_i is the weighted sum of all the value segments $\{V_1, V_2, \ldots, V_n\}$. Finally, concatenating all the Y_i along the length dimension yields SC's output.

Multi-Scale Segment Correlation (MSSC) fuses the output of multiple SC with different L_{seg} values. This allows to capture both coarse-grained (large L_{seg}) and fine-grained (small L_{seg}) temporal dependencies. MSSC starts with a small initial segment length L_0, which increases exponentially until the max scale level $l_{max} = log_2(\frac{L}{L_0})$ is reached (i.e: $L_l = 2^l L_0$ where $l \in \{0, 1, \ldots, l_{max}\}$. As the scale level increases, the weight of the l-th level α_l is set to decrease exponentially. MSSC can be formulated as:

$$\begin{aligned} \text{MSSC}(H) &= \sum_{l=0}^{l_{\text{max}}} \alpha_l \cdot \text{SC}\left(H; 2^l L_0\right) \\ \alpha_l &= \frac{2^l}{\sum_{l=0}^{l_{\text{max}}} 2^l} \end{aligned} \tag{2.13}$$

The computational complexity of MSSC is the sum of all scales: $\mathcal{O}(L^2/L_0)$.

Preformer's cross-attention module combines a predictive paradigm with MSSC, and is called *Predictive Multi-Scale Segment Correlation (PreMSSC)*. It relies on the fact that during the decoding phase, the query for the period to be predicted is its preceding segment rather than itself. Thus, it uses the query of previous segment to predict the current segment. Furthermore, the value segment is delayed by one segment relative to the corresponding key segment. Therefore, in PreMSSC, Y_i is computed as:

$$\begin{aligned} Y_1 &= \sum_{j=1}^{n-1} \hat{c}_{(Q_m, K_j)} V_{j+1} \\ Y_i &= \sum_{j=1}^{n-1} \hat{c}_{(Q_{i-1}, K_j)} V_{j+1}, \quad i > 1 \end{aligned} \tag{2.14}$$

The overall structure of MSSC is illustrated in Figure 2.4.4.

2.4.2.3 Frequency Enhanced Structures

FEDformer substitutes both the self-attention and cross-attention blocks with the *Frequency Enhanced Block (FEB)* and *Frequency Enhanced Attention (FEA)* modules, respectively. In these *Frequency Enhanced Structures*, computations are performed in the frequency domain on a fixed set M of randomly selected modes (i.e: frequency components). By setting $M << L$ (default is $M = 64$), FEDformer achieves a linear time and memory complexity of $\mathcal{O}(L)$.

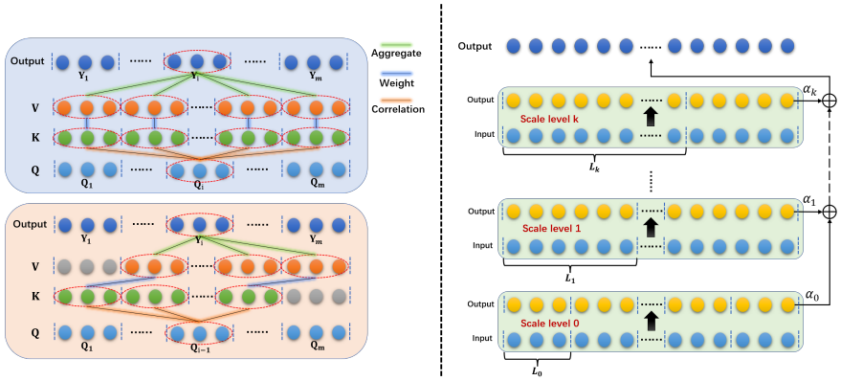


Figure 2.4.4: Segment-Correlation (upper left), predictive paradigm (bottom left) and the multi-scale architecture (right) [10].

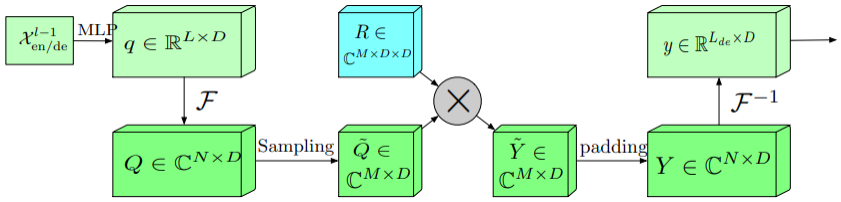


Figure 2.4.5: Frequency Enhanced Block (FEB) structure [49].

The FEB module, whose structure is shown in Figure 2.4.5, works as follows:

$$\begin{aligned} \boldsymbol{q} &= \boldsymbol{x} \cdot \boldsymbol{w} \\ \tilde{\boldsymbol{Q}} &= \text{Select}(\boldsymbol{Q}) = \text{Select}(\mathcal{F}(\boldsymbol{q})) \\ \text{FEB}\,(\boldsymbol{x}) &= \mathcal{F}^{-1}(\,\text{Padding}\,(\tilde{\boldsymbol{Q}} \odot \boldsymbol{R})) \end{aligned} \tag{2.15}$$

First, the input series $\boldsymbol{x} \in \mathbb{R}^{L \times D}$ is linearly projected on $\boldsymbol{w} \in \mathbb{R}^{D \times D}$. The result $\boldsymbol{q}$ gets converted from the time domain to the frequency domain using the Fourier transform operation $\mathcal{F}(\cdot)$, giving $\boldsymbol{Q} \in \mathbb{C}^{L \times D}$. Here, only M randomly selected modes are kept, yielding $\tilde{\boldsymbol{Q}} \in \mathbb{C}^{M \times D}$. The production operator $\odot$ is then applied between $\tilde{\boldsymbol{Q}}$ and a randomly initialized parameterized kernel $\boldsymbol{R} \in \mathbb{C}^{D \times D \times M}$, whose result is zero-padded to give it the same length as the input series, yielding $\boldsymbol{Y} \in \mathbb{C}^{L \times D}$. Finally, $\boldsymbol{Y}$ is converted back to the time domain using inverse Fourier transform, where the output is $\boldsymbol{y} \in \mathbb{R}^{L \times D}$.

The FEA module behaves similarly to cross-attention in Transformer, the key difference being that in FEDformer, the queries $\tilde{\boldsymbol{Q}}$, keys $\tilde{\boldsymbol{K}}$ and values $\tilde{\boldsymbol{V}}$ represent a random set of M frequency modes, whereas in Transformer, they represent temporal points. The structure of FEA is shown in Figure 2.4.6 and is formally described

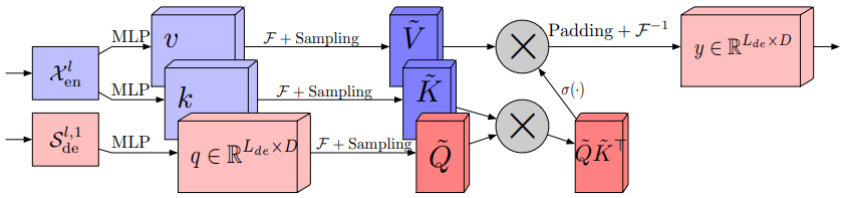


Figure 2.4.6: Frequency Enhanced Attention (FEA) structure [49].

as:

$$\begin{aligned} \tilde{\boldsymbol{Q}} &= \text{Select}(\mathcal{F}(\boldsymbol{q})) \\ \tilde{\boldsymbol{K}} &= \text{Select}(\mathcal{F}(\boldsymbol{k})) \\ \tilde{\boldsymbol{V}} &= \text{Select}(\mathcal{F}(\boldsymbol{v})) \\ \text{FEA}(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v}) &= \mathcal{F}^{-1}\left(\text{Padding}\left(\sigma\left(\tilde{\boldsymbol{Q}} \cdot \tilde{\boldsymbol{K}}^{\top}\right) \cdot \tilde{\boldsymbol{V}}\right)\right) \end{aligned} \tag{2.16}$$

where σ is the activation function, which is usually softmax of tanh.

2.4.3 Deep Decomposition Architecture

TSTs typically employ what is known as a Deep Decomposition Architecture (DDA), which unlike the vanilla transformer architecture, contains a series decomposition block as a reoccurring inner module. The series decomposition block serves to extract both the long-term stationary trend and the intricate seasonal patterns of a time series. Having it as a reoccurring inner block, instead of the common approach of simply decomposing the data once prior to model input, allows TSTs to progressively decompose the time series. The benefit of this is twofold. Firstly, it allows the attention mechanism to focus solely on modelling the seasonality part of the data, which makes up the repeating and thus predictable patterns of the time series. And secondly, it allows the model to predict the trend and seasonality components separately before combining them, which can lead to much improved results when forecasting longer sequences. The operation of the series decomposition block is described in Section 2.2.1.

2.4.4 The Models

The TSTs presented in this section all share a somewhat similar DDA structure (see previous section), where their main differentiating factor is their attention mechanisms. Specifically, their encoders and decoders all perform the same operation,

in addition to having the same inputs. For this reason, we present the different TSTs by first summarising their common components, namely their model inputs, their encoder and their decoder. Subsequently, we provide a description of each model's respective structure.

Model Inputs

The input for the TST's encoder $\mathcal{X}_{\text{en}} \in \mathbb{R}^{L \times d}$ is the original time series with length L embedded with additional time-dependant features called covariates (as explained in Section 2.4.1). As DDAs, their decoder input contains 2 components that are to be refined: the seasonal component $\mathcal{X}_{\text{des}} \in \mathbb{R}^{\left(\frac{L}{2}+O\right) \times d}$ and the trend component $\mathcal{X}_{\text{det}} \in \mathbb{R}^{\left(\frac{L}{2}+O\right) \times d}$. Each component is composed of 2 parts: the component decomposed from the latter half of encoder's input $\mathcal{X}_{\text{en}}$ with length $L/2$ to provide recent information, and placeholders with length O filled by scalars to be recursively updated with the model predictions. Since the time-dependant covariates are predetermined (i.e: time of day, day of week...), these are also embedded into the placeholders. The decoder input is formalized as follows:

$$\begin{aligned} \mathcal{X}_{\text{ens}}, \mathcal{X}_{\text{ent}} &= \text{SeriesDecomp}\left(\mathcal{X}_{\text{en } \frac{L}{2}:I}\right) \\ \mathcal{X}_{\text{des}} &= [\mathcal{X}_{\text{ens}}, \mathcal{X}_0] \\ \mathcal{X}_{\text{det}} &= [\mathcal{X}_{\text{ent}}, \mathcal{X}_{\text{Mean}}] \end{aligned} \tag{2.17}$$

where $\mathcal{X}_{\text{ens}}, \mathcal{X}_{\text{ent}} \in \mathbb{R}^{\frac{L}{2} \times d}$ are the seasonal and trend components of $\mathcal{X}_{\text{en}}$ and $\mathcal{X}_0, \mathcal{X}_{\text{Mean}} \in \mathbb{R}^{O \times d}$ are the placeholders filled with zero and the the mean of $\mathcal{X}_{\text{en}}$, respectively.

Encoder

Each TST encoder consists of N identical layers, much like the vanilla Transformer. Each layer consists of a self-attention module and a FFN module, each of which is followed by a series decomposition module (see Autoformer's structure in Figure 2.4.7 for reference). All decomposition modules in the encoder eliminate the trend component, making the encoder focus solely on modelling the seasonality part of the input series. The encoder's output, which contains the past seasonal information, is then used to help the decoder refine prediction results during the cross-attention phase. The equations for the l-th encoder layer are summarised as

$\mathcal{X}_{\text{en}}^{l} = \text{Encoder}\left(\mathcal{X}_{\text{en}}^{l-1}\right)$, where each sub-layer is obtained as:

$$\begin{aligned} \mathcal{S}_{\text{en}}^{l,1}, \_ &= \text{SeriesDecomp}\left(\text{ Self-Attention } \left(\mathcal{X}_{\text{en}}^{l-1}\right) + \mathcal{X}_{\text{en}}^{l-1}\right) \\ \mathcal{S}_{\text{en}}^{l,2}, \_ &= \text{SeriesDecomp}\left(\text{ FeedForward } \left(\mathcal{S}_{\text{en}}^{l,1}\right) + \mathcal{S}_{\text{en}}^{l,1}\right) \end{aligned} \tag{2.18}$$

where $\mathcal{X}_{\text{en}}^{l} = \mathcal{S}_{\text{en}}^{l,2}, l \in \{1, \cdots, N\}$ denotes the output of l-th encoder layer and $\mathcal{X}_{\text{en}}^{0}$ is the positionally encoded input $\mathcal{X}_{\text{en}}$. $\mathcal{S}_{\text{en}}^{l,i} \in \{1, 2\}$ represents the seasonal component after the i-th series decomposition block in the l-th layer, and "_" is the eliminated trend part. Depending on the model, Self-Attention$(\cdot)$ can be either Auto-Correlation, MSSC or FEB.

Decoder

Each TST decoder consists of M identical layers. The decoder is divided into two sections: the accumulation structure for trend components and the stacked attention mechanism for seasonal components. Each decoder layer consists of an inner self-attention module and an encoder-decoder cross-attention module, which refine the prediction and utilize the past seasonal information, respectively. The series decomposition modules in the decoder extract the underlying trend part from the intermediate hidden variables progressively, allowing the model to continually refine its prediction. The equations for the l-th layer, given the latent variable X_{en}^{N} from the encoder, can be written as $\mathcal{X}_{\text{de}}^{l} = \text{Decoder}\left(\mathcal{X}_{\text{de}}^{l-1}, \mathcal{X}_{\text{en}}^{N}\right)$. The decoder can be formalized as:

$$\begin{aligned} \mathcal{S}_{\text{de}}^{l,1}, \mathcal{T}_{\text{de}}^{l,1} &= \text{SeriesDecomp}\left(\text{ Self-Attention } \left(\mathcal{X}_{\text{de}}^{l-1}\right) + \mathcal{X}_{\text{de}}^{l-1}\right) \\ \mathcal{S}_{\text{de}}^{l,2}, \mathcal{T}_{\text{de}}^{l,2} &= \text{SeriesDecomp}\left(\text{ Cross-Attention } \left(\mathcal{S}_{\text{de}}^{l,1}, \mathcal{X}_{\text{en}}^{N}\right) + \mathcal{S}_{\text{de}}^{l,1}\right) \\ \mathcal{S}_{\text{de}}^{l,3}, \mathcal{T}_{\text{de}}^{l,3} &= \text{SeriesDecomp}\left(\text{ FeedForward } \left(\mathcal{S}_{\text{de}}^{l,2}\right) + \mathcal{S}_{\text{de}}^{l,2}\right) \\ \mathcal{T}_{\text{de}}^{l} &= \mathcal{T}_{\text{de}}^{l-1} + \mathcal{W}_{l,1} * \mathcal{T}_{\text{de}}^{l,1} + \mathcal{W}_{l,2} * \mathcal{T}_{\text{de}}^{l,2} + \mathcal{W}_{l,3} * \mathcal{T}_{\text{de}}^{l,3} \end{aligned} \tag{2.19}$$

where $\mathcal{X}_{\text{de}}^{l} = \mathcal{S}_{\text{de}}^{l,3}, l \in \{1, \cdots, M\}$ is the output of the $l - th$ decoder layer. $\mathcal{X}_{\text{de}}^{0}$ and $\mathcal{T}_{\text{de}}^{0}$ represent the model inputs $\mathcal{X}_{\text{des}}$ and $\mathcal{X}_{\text{det}}$, respectively. $\mathcal{S}_{\text{de}}^{l,i}, \mathcal{T}_{\text{de}}^{l,i}, i \in \{1, 2, 3\}$ denote the seasonal component and trend component after the i-th series decomposition block in the l-th layer, respectively. $\mathcal{W}_{l,i}, i \in \{1, 2, 3\}$ represents the projection matrix for the $i - th$ extracted trend $\mathcal{T}_{\text{de}}^{l,i}$.

The refined prediction for both the trend and seasonal components then get summed together as $\mathcal{W}_{\mathcal{S}} * \mathcal{X}_{\text{de}}^{M} + \mathcal{T}_{\text{de}}^{M}$, giving the final prediction, where the matrix $\mathcal{W}_{\mathcal{S}}$ is used to

project the seasonal component $\mathcal{X}_{\text{de}}^{M}$ to the target dimension.

2.4.4.1 Autoformer

Autoformer, proposed by [41] in 2021, and shown in Figure 2.4.7, was the first model to renovate Transformer as a DDA. It has 2 new sub-layers. The first is the Auto-Correlation block, which uses the Auto-Correlation mechanism (see section 2.4.2.1) to replace both self and cross-attention blocks, enabling Autoformer to discover temporal dependencies at the sub-series level. The second is the series decomposition block, which follows each Auto-Correlation and FFN block, allowing the progressive decomposition of the series into trend and seasonality components.

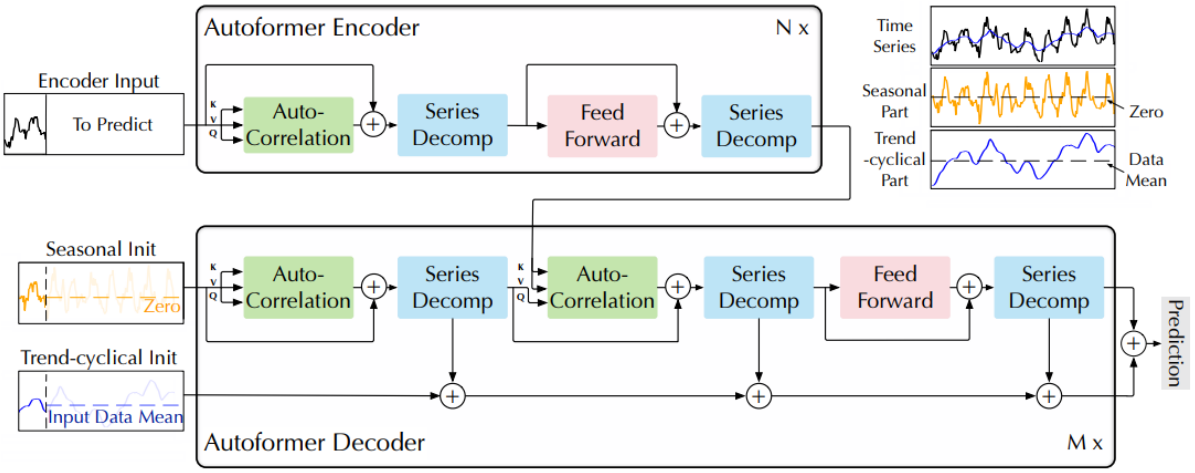


Figure 2.4.7: Autoformer structure [41].

2.4.4.2 Preformer

Preformer, developped by [10] in 2022, is heavily inspired by Autoformer's DDA structure. Different to Autoformer however, is Preformer's self and cross-attention sub-layers, which are replaced with MSSC and PreMSSC blocks (see Section 2.4.2.2), respectively. These new blocks allow Preformer to divide the time series into segments and perform segment-wise correlation-based attention to discover temporal dependencies in the data. The structure of Preformer is shown in Figure 2.4.8.

2.4.4.3 FEDformer

FEDformer, or *Frequency Enhanced Decomposed Transformer*, also employs a DDA structure and was proposed in 2022 by [49]. It uses the FEB and FEA modules (see Section 2.4.2.3) as its self and cross-attention sub-layers, respectively. This enables

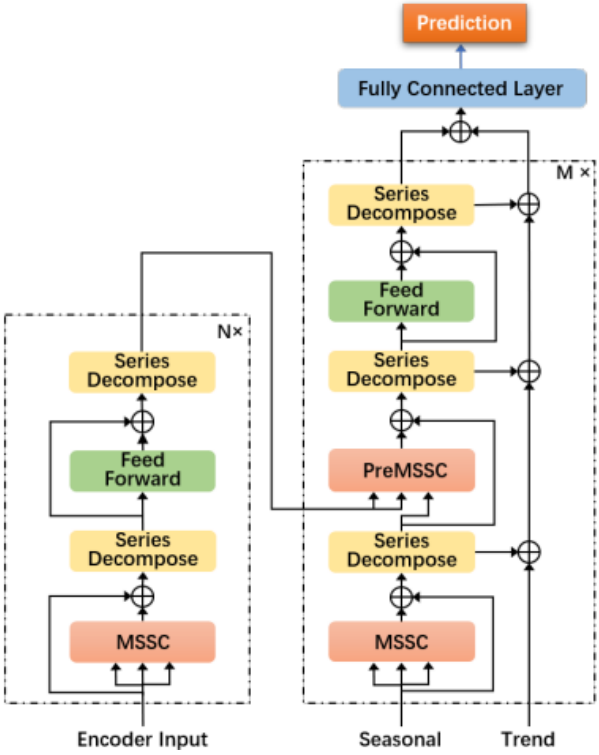


Figure 2.4.8: Preformer structure [10].

FEDformer to capture the temporal dependencies in the time series entirely from the frequency domain. Furthermore, it employs a unique series decomposition block called *Mixture of Experts Decomposition (MOEDecomp)*, which differentiates itself from traditional series decomposition by using multiple moving averages to extract the trend component instead of just using a single one. FEDformer's structure is shown in Figure 2.4.9.

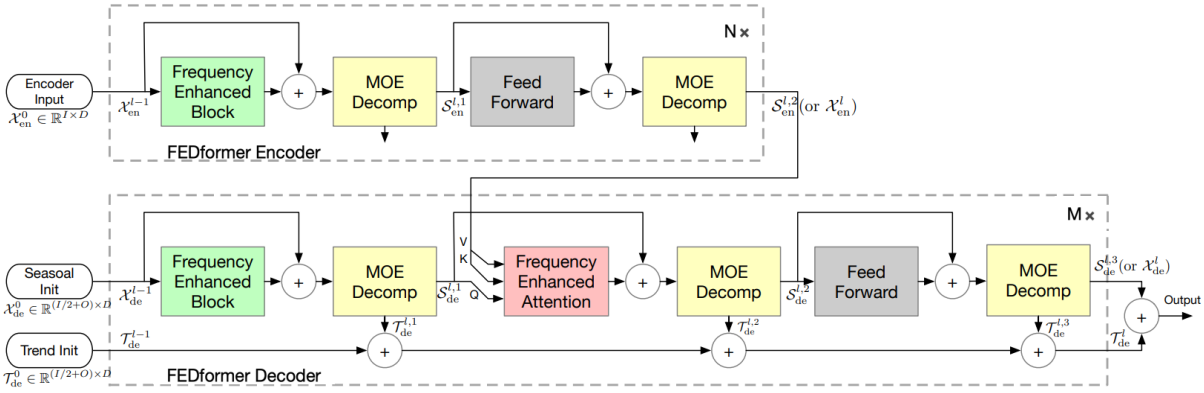


Figure 2.4.9: FEDformer structure [49].

2.5 Related Work

KPI prediction in telecom networks is a topic that has been extensively researched for the many practical benefits it can yield, as outlined in Section 2.1.

In [2], a comparative evaluation between LSTM and ARIMA is performed for the task of cellular traffic prediction. The results demonstrate that LSTM outperforms ARIMA in general, especially when the amount of available training data is large enough, and it is enhanced with additional carefully selected features. The study also reveals that, in some less complex scenarios, ARIMA can perform very closely to LSTM. [13] performs a similar study but also evaluates a FFN for traffic prediction, with results showing that LSTM achieves much lower prediction errors than ARIMA, and produces similar errors to a FFN while requiring a significantly shorter training time. [27] studies the capabilities of LSTM, FFN and ARIMA for predicting the mobile traffic of a Long-Term Evolution (LTE) base station, where the NoCU is used as the target KPI. Its empirical results show that both machine learning models outperform ARIMA, with LSTM achieving the best overall prediction accuracy. The study further shows that LSTM's prediction accuracy increases with the input sequence length K, where K was tested from 1 to 10 time steps.

In [25], a multi-modal deep learning model for predicting traffic load in 4G/5G networks is put forward, which utilizes and combines external data features like landscape and weather information to make more informed predictions. Specifically, the proposed hybrid model employs (1) an LSTM to model the temporal traffic component, (2) a Convolutional Neural Network (CNN) trained on satellite imagery data to classify the type of region in which a cell is situated, (3) weather information to model the correlations between weather and traffic behaviours. The performed experiments demonstrate that the hybrid model doesn't typically lead to better predictive performance, and that LSTM alone is already capable of modeling complex sequential patterns. Similarly, [47] proposes a multi-view ensemble learning approach for predicting mobile traffic load, which in addition to temporal patterns, captures the spatial influence and impact of external events, such as weather and holidays. Additionally, an optimisation algorithm was applied which uses the traffic predictions to put base stations to sleep for reduced energy consumption, yielding about 10% more energy savings than other more traditional methods.

Transformer was tested and compared to LSTM for mobile traffic prediction in

[30]. The results show that LSTM consistently outperforms Transformer in terms of prediction accuracy, when both models are trained for the same number of epochs. However, the results also suggest that Transformer requires less overall training time than LSTM to achieve similar performance, as when both models are given the same amount of training time, Transformer yields more accurate predictions than LSTM in 3 out of 4 evaluation cases. It is worth noting that [30] only evaluates the vanilla Transformer for traffic prediction, whereas this project focuses on evaluating novel TST architectures for said task, which, post-research, has not been done before.

Chapter 3

Methods

3.1 Dataset

In this section, we present the data that was used for the experiments. We first provide a description of the data followed by how the data was pre-processed and split prior to being fed to the models.

3.1.1 Description

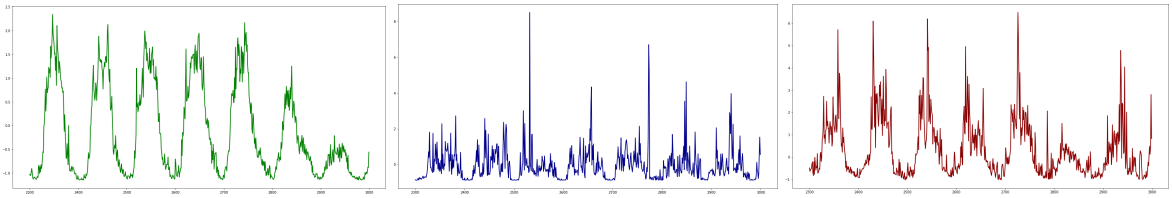
The data used in this project is real-world time series data taken from a telecom operational network in Geneva, with downlink traffic being the main KPI which we are trying to predict. Specifically, the data from three different cells was used, where the cells were selected based on their geographic deployment location, which, as outlined in Section 2.1, has a big impact on their traffic behaviour. Specifically, the 3 cells come from the following locations:

1. **A downtown area**: This cell generally experiences quite high traffic. Its traffic exhibits strong daily patterns with traffic typically being quite low during the night and rising throughout the day where it peaks around the evening time. It also displays obvious weekly patterns with traffic being much lower on weekends, with Sundays commonly showing the lowest traffic.
2. **A suburb area**: The traffic experienced by this cell consistently remains low and displays somewhat less predictable patterns than the downtown cell. The traffic tends to peak at random times depending on the day, with no clear sign of

weekly patterns. The only repeating pattern is the presence of no traffic during a few hours each night.

3. **An airport**: This cell experiences quite erratic traffic behaviour. Its traffic level varies a lot depending on the day, and displays no obvious sign of weekly patterns. This pattern irregularity could be due to traffic behavior being reliant on the airports flight activity which can be quite random and fluctuate a lot depending on the day and time of year.

Each cell also experiences high spikes of traffic that seem to occur randomly and for very short durations. The data for each cell consists of 20640 data points with a time granularity of 15 minutes (i.e: each time step is separated by a 15 minute interval) which spans from 2022-04-01 to 2022-11-01, representing a total of 7 months worth of data. The traffic data for each cell can be visualised in Figure 3.1.1, where each data point represents the traffic that occurred at that particular timestamp.



(a) Downtown (b) Suburb (c) Airport

Figure 3.1.1: One week normalised traffic data for each cell (taken from 2022-04-25 to 2022-05-01).

3.1.2 Pre-processing

The raw data contains several undesirable attributes that may have arisen from the data collection process or a malfunctioning cell. Hence, before being fed to the models, the data needed to undergo cleaning and pre-processing. Specifically, the data pre-processing phase involved three steps, which are described next.

The first two steps involved removing any duplicate time steps and replacing any missing time steps. The missing time steps only account for roughly 0.32% of the entire dataset. Since the missing time steps always represent small gaps, *mean interpolation* was used to replace the data points, which essentially replaces each missing data point with the average of the data points directly neighbouring it. These steps are crucial to ensure that the positional information encoded into each time step is correct, so as to ensure that the TSTs don't mistake for consecutive time steps, time steps that

are in fact duplicates or far away from each other, which could negatively affect their performance.

As the third step, the data is normalised to ensure that all input features are on the same scale. Specifically, each data feature gets normalized to a mean of 0 and a standard deviation of 1. For each data feature x, its normalized representation x^* is computed by:

$$x^* = \frac{x - \text{mean}(x)}{\text{std}(x)} \tag{3.1}$$

3.1.3 Data Splitting

Individual data points are extracted from the dataset using the sliding window approach described in Section 2.2.2. The data is then split such that the first 70% is used for training the models, and the last 30% is used for testing the models. The data split in terms of time periods is shown in the table below.

	Description
City	Geneva
Training period	April 1st 2022 - August 28th 2022
Testing period	August 29th 2022 - November 1st 2022
Training weeks	21
Testing weeks	9

Table 3.1.1: Experiment data split in terms of time periods.

3.2 Model Benchmarking

In this section, we present a comprehensive overview of the entire model benchmarking process. This encompasses model selection, the selection of hyperparameters for training, the evaluation metrics employed, as well as implementation details and hardware configuration.

3.2.1 Model Selection

Four different TST models and one LSTM baseline model were selected and built according to their relative performance in the current literature on 6 common time series benchmark datasets [10, 41, 48, 49]. The four TST models include the 3 models

presented in Section 2.4.4 in addition to the vanilla transformer which implements the enhanced positional encoding scheme as described in Section 2.4.1. We include the vanilla transformer to evaluate the performance discrepancy between the vanilla attention and efficient attention mechanisms.

3.2.2 Training Hyperparameters

Each model was trained with the default hyperparameters proposed in their respective papers. It's important to note that while these hyperparameters may not be optimal for the specific datasets utilized in this work, conducting an exhaustive hyperparameter grid search was not feasible within the constraints of time and computational resources. Nevertheless, the parameter settings for both the TSTs and LSTM are presented in the Tables 3.2.1 and 3.2.2, respectively. All models are trained and tested with the MSE loss, using the ADAM [14] optimizer with a decaying learning rate. The batch size is set to 32. Other important hyperparameters such as the input and prediction sequence lengths are discussed in Section 3.3 where we describe the experiments that were performed in this work.

Parameter	**Value**	**Description**
d_model	512	Model dimmensionality representing the vector dimension for each encoded data point.
n_heads	8	Number of attention heads.
e_layers	2	Number of encoder layers.
d_layers	1	Number of decoder layers.
d_ff	2048	Dimmensionnality of the inner layer of the FFN modules.
$dropout$	0.05	Dropout value which controls regularization.
$activation$	$GELU$	Activation function.
$train\_epochs$	10	Number of training epochs.
$learning\_rate$	0.0001	Initial learning rate.
$modes$	64	Number of frequency modes for FEDformer.

Table 3.2.1: TST hyperparameter configuration.

Parameter	Value	Description
num_layers	2	Number of LSTM units.
$hidden\_size$	128	Number of hidden layers.
num_epochs	20	Number of training epochs.
$learning\_rate$	0.001	Initial learning rate.

Table 3.2.2: LSTM hyperparameter configuration.

3.2.3 Evaluation Metrics

To answer the research question provided in Section 1.1, we use a set of metrics to evaluate the performance of each model in different areas. The research questions are fundamentally focused on two core aspects. The first is to investigate whether TST models can provide better predictive performance than LSTM, and the second is to examine the feasibility of using and deploying TSTs in a practical setting where computational ressources might be more limited. To answer the former, we calculate each models prediction accuracy using the MSE and MAE scores (see Section 2.2.3). To answer the latter, we measure each models training time, inference time, and memory costs.

3.2.4 Implementation Details

All practical implementations were built in Python 3.9. The TST models were implemented using the code from their respective papers, which was made publicly available by the authors. The LSTM network was built using the PyTorch library. The code repository for this project was carefully organised such that a single shell script is used to simultaneously run all of the experiments described in the next section.

The hardware setup consisted of a machine with one NVIDIA Tesla T4 GPU, four CPUs, and a memory of 16GB.

3.3 Experiments

To gauge and compare the performance of each model, a set of different experiments were conducted where each model was trained under different scenarios consisting of four configurable parameters. These parameters, in addition to their tested values, are shown in Table 3.3.1. The first configurable parameter is the dataset on which the

model is trained, with comprehensive descriptions for each dataset outlined in Section 3.1.1. We alter this parameter primarily to assess the robustness of each model, or in other words, to understand how they perform when presented with different kinds of data exhibiting diverse characteristics. The second parameter we consider is the selection of input features. As discussed in Section 2.2, the NoCU to a given cell is strongly correlated with its downlink traffic, making it of particular interest to evaluate whether incorporating NoCU as an additional input feature can lead to improved traffic predictions. Accordingly, we consider both univariate and multivariate forms of input data in our experiments. The third parameter relates to the input sequence length. Adjusting this parameter is useful in determining whether increasing its value leads to better predictions by the TSTs, which were designed with attention mechanisms for this specific purpose. We choose values ranging up to a week as some of the datasets used to train the models exhibit weekly patterns. Finally, the fourth parameter concerns the output sequence length. We vary this parameter to investigate the forecasting capabilities of the models, with the aim of identifying specific scenarios and use cases in which TSTs may be particularly effective.

Dataset	**Input Features**	**Input Length**	**Output Length**
Downtown	Traffic (*univariate*)	48 (*12 hours*)	4 (*1 hour*)
Suburb	Traffic + NoCU (*multivariate*)	96 (*1 day*)	16 (*4 hours*)
Airport		672 (*1 week*)	96 (*1 day*)

Table 3.3.1: Configurable parameters for the performed experiments.

It is worth noting that each model is trained on an individual cell basis, or in other words, each model is trained to predict the traffic of one cell only. This means that we don't consider the multivariate to multivariate forecasting scenario described in Section 2.2. Furthermore, to allow for a fair comparison, each model is trained and evaluated on the exact same number of data samples. Finally, each experiment is repeated 3 times to account for different model initializations.

Chapter 4

Results and Analysis

In this section, we present and analyse the results of our experiments detailed in the prior chapter. This section is structured around the various evaluation metrics that we aim to comprehend from the different models.

4.1 Predictive Performance

To present the results of each model's predictive performance, we divide this section into each of the different prediction horizons that were tested, as each of these provide their own unique insights for the behaviour of each model.

4.1.1 One-hour Horizon

The one-hour horizon forecasting results using the univariate input data are displayed in Table 4.1.1. The main takeaway from these results, is that Transformer produces the lowest MSE score out of all models, although it is somewhat marginal when compared to LSTM. Specifically, Transformer (L = 672) achieves an average MSE of 0.339, which is around 5.57% lower than the best performing LSTM (L = 48), which achieves an average MSE of just 0.359. Interestingly however, LSTM outperforms the three TSTs with efficient attention mechanisms, producing very slight accuracy improvements over Preformer and FEDformer, while significantly outperforming Autoformer. This suggests that LSTM still remains quite competitive for predicting traffic under this short prediction horizon. To validate the robustness of these findings, the ANOVA statistical significance test [26] was conducted on the three repeated runs for the best

results (underlined in the table), revealing a p-value of 0.023 (< 0.05). This outcome signifies the reliability of the results, indicating that they are not merely a product of chance.

Model	Metric	Downtown			Suburb			Airport			**Average**		
		48	96	672	48	96	672	48	96	672	48	96	672
LSTM	MSE	**0.106**	0.109	0.111	0.682	0.687	0.715	0.289	0.294	0.288	0.359	0.363	0.371
	MAE	0.226	0.231	0.229	0.520	0.502	0.523	0.346	0.353	0.347	0.364	0.362	0.367
Transformer	MSE	0.111	0.110	0.108	0.656	0.648	**0.647**	0.265	**0.262**	0.264	0.344	0.340	**0.339**
	MAE	0.229	0.224	**0.221**	**0.451**	0.468	0.457	0.328	0.335	**0.322**	0.336	0.342	**0.333**
Autoformer	MSE	0.427	0.274	0.347	0.710	0.700	0.685	0.381	0.407	0.368	0.506	0.460	0.467
	MAE	0.476	0.389	0.454	0.512	0.508	0.501	0.436	0.466	0.446	0.474	0.454	0.467
Preformer	MSE	0.123	0.149	0.169	0.686	0.681	0.688	0.289	0.303	0.311	0.366	0.377	0.389
	MAE	0.247	0.279	0.306	0.485	0.475	0.486	0.357	0.377	0.382	0.363	0.377	0.391
FEDformer	MSE	0.130	0.119	0.188	0.687	0.681	0.680	0.287	0.280	0.317	0.368	0.360	0.395
	MAE	0.254	0.241	0.320	0.479	0.477	0.492	0.361	0.353	0.400	0.364	0.357	0.404

Table 4.1.1: One-hour forecasting univariate results.

To support these results, Figure 4.1.1 shows the 1-hour predictions of LSTM, Transformer and FEDformer on the last 4 days of the downtown dataset.

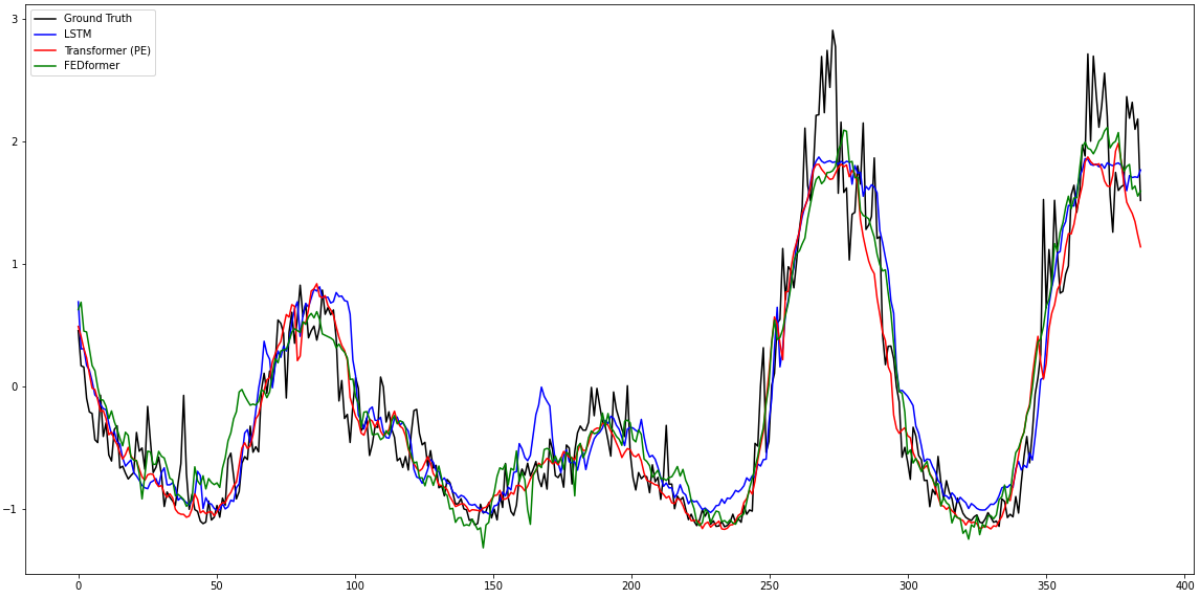


Figure 4.1.1: LSTM (blue) vs. Transformer (red) vs. FEDformer (green) predictions on the downtown dataset. Each point is predicted 4 steps (1 hour) beforehand. Four days of data are displayed, showcasing both low and high traffic days.

4.1.2 Four-hour Horizon

Table 4.1.2 shows the four-hour forecasting results, where again, Transformer (L = 672) provides the best average prediction accuracy with an MSE of 0.366, which is 9.63% lower than the best MSE score of 0.405 achieved by LSTM (L = 96).

Interestingly, this accuracy improvement of Transformer w.r.t LSTM is larger than for the one-hour horizon, indicating a potential trend where as the forecasting horizon increases, so does the predictive performance improvement of TSTs w.r.t to LSTM. To support this, Preformer and FEDformer both provide better average MSE scores than LSTM for the four-hour horizon, which wasn't the case for the one-hour horizon. Applying the ANOVA test to these results gives a p-value of 0.019 (< 0.05), confirming the reliability of the findings.

Model	Metric	Downtown			Suburb			Airport			**Average**		
		48	96	672	48	96	672	48	96	672	48	96	672
LSTM	MSE	0.151	0.153	0.146	0.778	0.754	0.826	0.324	0.307	0.323	0.418	0.405	0.432
	MAE	0.277	0.280	0.272	0.575	0.553	0.596	0.382	0.364	0.377	0.411	0.399	0.415
Transformer	MSE	0.133	0.126	**0.125**	0.693	0.687	**0.685**	0.302	**0.286**	0.288	0.376	0.367	**0.366**
	MAE	0.255	0.247	**0.241**	**0.456**	0.468	0.487	0.340	0.355	**0.333**	**0.350**	0.356	0.354
Autoformer	MSE	0.343	0.419	0.387	0.786	0.715	0.706	0.439	0.379	0.430	0.522	0.504	0.507
	MAE	0.443	0.490	0.454	0.559	0.505	0.504	0.468	0.426	0.495	0.49	0.473	0.484
Preformer	MSE	0.153	0.163	0.201	0.730	0.725	0.719	0.334	0.328	0.333	0.404	0.405	0.417
	MAE	0.284	0.298	0.339	0.513	0.505	0.501	0.388	0.394	0.404	0.395	0.399	0.414
FEDformer	MSE	0.184	0.142	0.181	0.721	0.696	0.709	0.332	0.302	0.339	0.412	0.380	0.409
	MAE	0.310	0.265	0.313	0.502	0.487	0.502	0.393	0.363	0.405	0.401	0.371	0.406

Table 4.1.2: Four-hour forecasting univariate results.

Figure 4.1.1 shows the four-hour predictions of LSTM, Transformer and FEDformer on the last 4 days of the downtown dataset. The predictions show that the TSTs are typically better at making distinctive predictions for both low and high traffic days, whereas LSTM tends to slighty overshoot its predictions for low traffic days. This suggests that LSTM is not as effective as anticipating these change in daily patterns when compared to TSTs.

4.1.3 One-day Horizon

The one-day forecasting horizon results, shown in Table 4.1.3, further support the idea that TSTs increasingly outperform LSTM as we extend the prediction horizon. Indeed, Transformer (L = 96) provides the best accuracy with an average MSE score of 0.389, a 11.79% improvement over the 0.441 MSE score achieved by the best LSTM (L = 96), showing the largest improvement out of all prediction horizons. Furthermore, we again find that Preformer and FEDformer outperfrom LSTM, with Autoformer showing the highest average MSE and MAE scores out of all the models. In this instance, the ANOVA test gives a p-value of 0.011 (< 0.05), confirming the reliability of the findings.

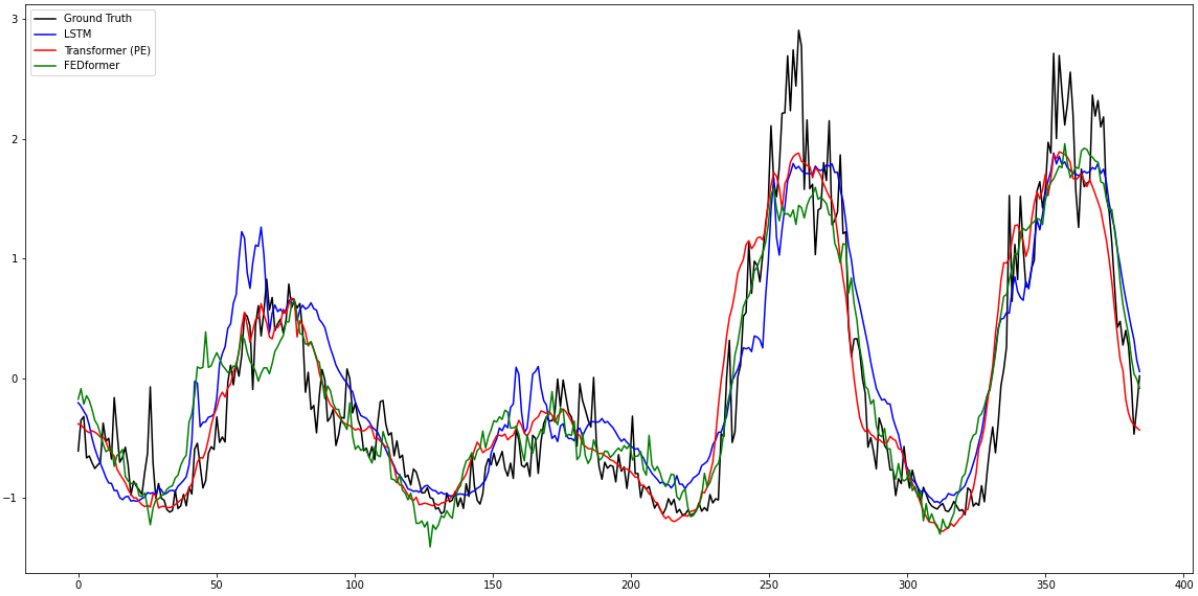


Figure 4.1.2: LSTM (blue) vs. Transformer (red) vs. FEDformer (green) predictions on the downtown dataset. Each point is predicted 16 steps (4 hours) beforehand. Four days of data are displayed, showcasing both low and high traffic days.

Model	Metric	Downtown			Suburb			Airport			**Average**		
		48	96	672	48	96	672	48	96	672	48	96	672
LSTM	MSE	0.244	0.236	0.455	0.752	0.745	0.830	0.361	0.343	0.359	0.452	0.441	0.548
	MAE	0.340	0.336	0.504	0.540	0.531	0.580	0.395	0.385	0.396	0.425	0.417	0.493
Transformer	MSE	0.150	**0.140**	0.152	0.722	0.706	0.725	0.343	0.320	**0.305**	0.405	**0.389**	0.394
	MAE	0.271	**0.254**	0.280	0.515	0.507	0.486	0.356	0.353	**0.339**	0.381	0.371	**0.368**
Autoformer	MSE	0.565	0.383	0.705	0.744	0.711	0.732	0.506	0.525	0.550	0.605	0.539	0.662
	MAE	0.575	0.461	0.623	0.519	**0.485**	0.504	0.502	0.527	0.537	0.532	0.491	0.554
Preformer	MSE	0.187	0.196	0.159	0.711	0.715	0.723	0.387	0.382	0.366	0.428	0.431	0.416
	MAE	0.316	0.327	0.280	0.496	0.489	0.493	0.419	0.428	0.412	0.410	0.414	0.395
FEDformer	MSE	0.210	0.183	0.219	0.713	0.694	**0.687**	0.354	0.332	0.335	0.425	0.403	0.413
	MAE	0.335	0.309	0.353	0.501	0.491	0.490	0.408	0.391	0.403	0.414	0.397	0.415

Table 4.1.3: One-day forecasting univariate results. Displayed scores represent the average for the 3 repetitions.

Figure 4.1.3 plots the best one-day predictions achieved by LSTM, Transformer and Preformer on the last week of the downtown test data. It is clear to see that, when predicting one day ahead, TSTs do a much better job at anticipating the lower traffic days that occur on weekends when compared to LSTM, with LSTM commonly overshooting its traffic predictions for those days. This indicates that TSTs are able to more effectively model these long-term weekly patterns when compared to LSTM.

One interesting observation is that Transformer consistently outperforms the three TSTs with efficient attention mechanisms. The reason behind this is unclear, however could be attributed to one of two possible explanations. Firstly, upon examining

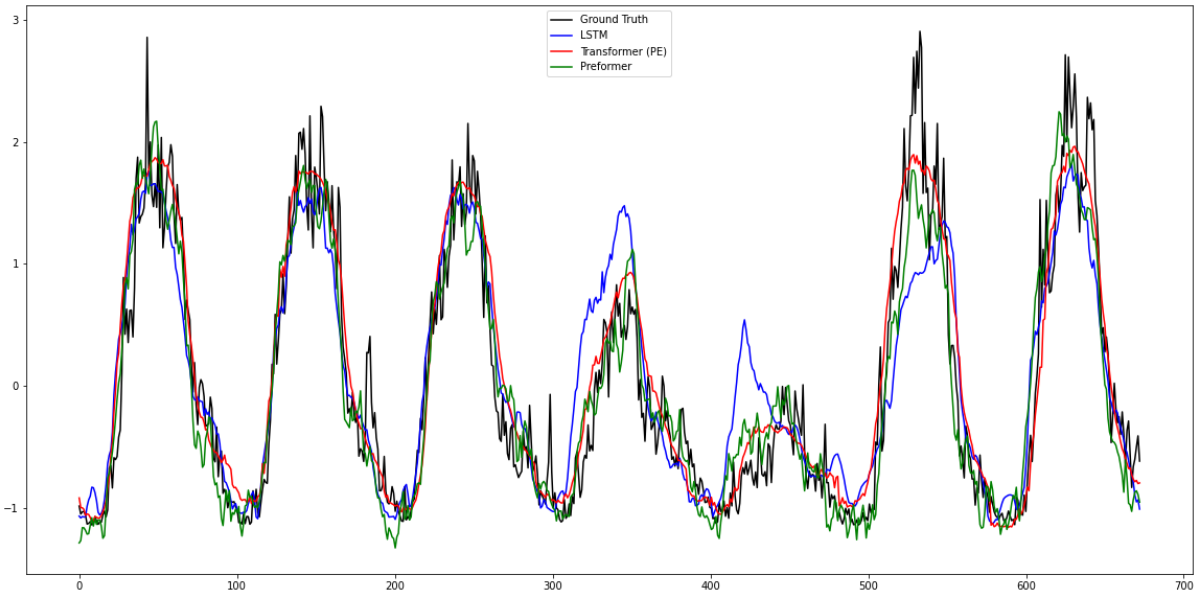


Figure 4.1.3: LSTM (blue) vs. Transformer (red) vs. Preformer (green) predictions on the downtown dataset. Each point is predicted 96 steps (one day) beforehand. One full week of data is displayed, showcasing both low and high traffic days.

the predictions shown in Figure 4.1.3, it becomes apparent that Transformer tends to make more conservative forecasts, primarily predicting the long-term trend of the traffic without giving much consideration to the short-term fluctuations. In contrast, Preformer displays a tendency to try and forecast both the trends and intricate short-term traffic fluctuations that are subject to many transient effects, making them particularly challenging to predict one day in advance, and hence, could be the cause of Preformer's slightly higher MAE and MSE scores when compared to Transformer. Another potential explanation could be that by reducing the complexity of Transformer's attention mechanism, this causes the TSTs that employ these efficient attention mechanisms to overlook certain critical data points from the input sequence that could help improve their predictions. This hypothesis is reinforced by the fact that increasing the input sequence length does not always result in better predictions, with L = 96 frequently yielding the most accurate forecasts, and L = 672 only sometimes providing very marginal improvements. In other words, while the goal of improving the efficiency of the vanilla attention mechanism is to better utilize longer input sequences, and since using these longer sequences does not commonly lead to improved predictions, it can be inferred that these attention mechanisms are being optimized for not much added benefit. Therefore, we can conclude that further research is needed in optimizing these attention mechanisms, such that they can more effectively and consistently harness long input sequences from time series data.

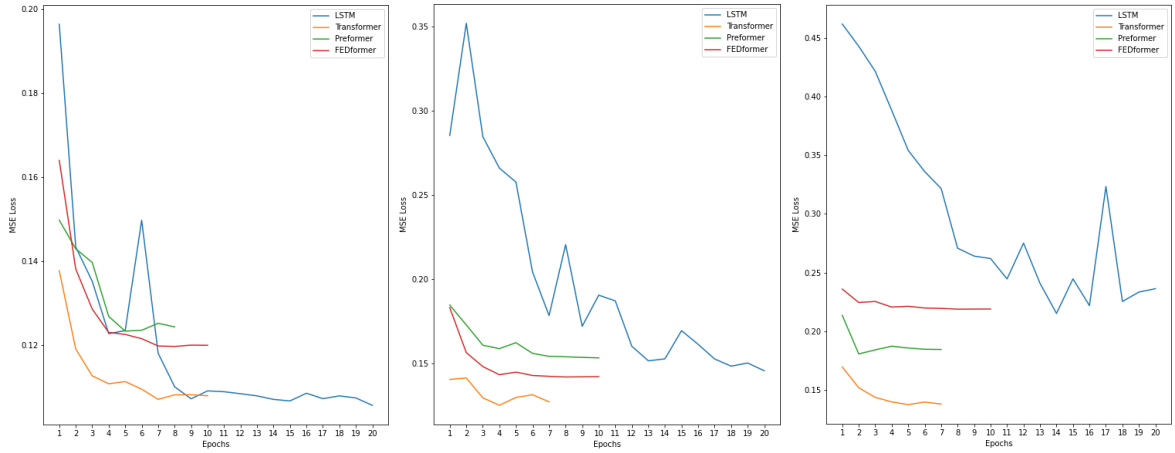
4.2 Training Time

The aim of this section is to offer a broad overview of the achievable test accuracy when training these models within a specific timeframe. While it's important to highlight that basing the decision solely on test accuracy for determining the number of epochs to train the models is insufficient (with the evolution of test accuracy being more crucial), it remains intriguing to examine the performance-to-training-time ratio of these models. Table 4.2.1 shows the training time per epoch for each model, which increases with the input sequence length L. Figure 4.2.1 shows the evolution of MSE test loss by training epoch for each best model on the downtown dataset. Together, they show that although TSTs require more training time per epoch when compared to LSTM, they typically converge after much fewer epochs, which results in TSTs usually requiring less overall training time than LSTM before converging.

Model	Training time (s)		
	48	96	672
LSTM	13	24	155
Transformer	29	43	260
Autoformer	46	72	375
Preformer	45	73	402
FEDformer	90	180	760

Table 4.2.1: Training time per epoch results per input sequence length.

For the one-hour horizon, LSTM achieves its best MSE score of 0.106 with L = 48 in 20 epochs (260s), while Transformer achieves its best MSE score of 0.108 with L = 672 in 7 epochs (1820s). However, by reducing L to 48, Transformer achieves a slightly increased MSE score of 0.111 in a much reduced time of 203s (7 epochs), giving a 4.7% higher MSE score than LSTM's best MSE score while being trained for roughly the same amount of time. For the four-hour horizon, LSTM achieves its best MSE score of 0.146 with L = 672 in 20 epochs (3100s) whilst achieving a very comparable MSE score of 0.151 with L = 48 in only 260s. Transformer produces its best MSE score of 0.125 with L = 672 in 4 epochs (1040s), but is still able to achieve a MSE score of 0.133 in just 116s (4 epochs) when L is reduced to 48, which still remains 8.9% lower than LSTM's best score. For the one-day horizon, Transformer achieves its best MSE of 0.140 with L = 96 after 4 epochs (172s), while LSTM achieves its best MSE of 0.236 with L = 96 after 14 epochs (336s). Interestingly, the same Transformer achieves an MSE of 0.169 after



(a) Horizon = 4 (1 hour) (b) Horizon = 16 (4 hours) (c) Horizon = 96 (1 day)

Figure 4.2.1: Test MSE loss vs. epoch for each best performing model on the downtown dataset. Autoformer is left out for its inferior performance. TSTs typically converge much more quickly than LSTM in terms of epochs. Early stopping was used for TSTs if their test loss didn't see a decrease in three epochs.

just one epoch of training (43s), which is better than LSTM's best achieved score.

Overall, it is clear that the improvement in prediction performance-to-training-time ratio of TSTs relative to LSTM increases as the forecasting horizon gets longer. Specifically, for a one-hour forecasting horizon, LSTM is able to achieve marginally better performance than TSTs when trained for the same amount of time. However, when the forecasting horizon is extended to one day, TSTs require a considerably shorter training time to achieve a significantly superior performance when compared to LSTM, which cannot achieve the same level of performance even with an extensive period of training.

4.3 Inference Time and Memory Costs

Table 4.3.1 shows both the respective memory costs and inference times of each model. Specifically, the inference times represents the average inference time of each model over 1000 datapoints from the test set, with the standard deviation also shown. The memory cost represents the amount of memory needed to store the checkpoint for each model (i.e: its parameter and weight values).

Concerning memory costs, we see that TSTs generally take up significantly more memory than LSTM, which is unsurprising considering the much larger number of parameters found in these models. Naturally, this makes them impractical to use in

Model	Number of parameters	Memory cost (MB)	Inference time (ms)
LSTM	211 552	1.6	342.21 $\pm$39.74
Transformer	10 519 553	60.0	683.38 $\pm$52.41
Autoformer	10 506 257	60.0	769.90 $\pm$63.02
Preformer	10 504 219	60.0	738.12 $\pm$54.13
FEDformer	16 797 713	100.0	1047.61 $\pm$51.75

Table 4.3.1: Inference time (CPU) and memory cost results.

scenarios where memory capacity is more limited, and consequently, they are more likely to require some sort of cloud service to be deployed to. When it comes to inference times, TSTs generally have a 2 to 3 times larger value when compared to LSTM, making them not very suited for critical real-time applications (i.e: when the forecasting horizon is on a seconds or sub-seconds time-frame). It's important to emphasize that these figures serve as a general indication of the performance one might anticipate with these models. However, actual results can fluctuate based on the available hardware resources and the efficiency of architecture implementation. Despite this, since the models employed here adhere to their original configurations from the respective research papers and are subject to the same hardware resources, these results should offer a reliable set of expectations.

4.4 Associated Costs: Practical Example

In this section, we provide some context to help clarify the implications of the aforementioned results by presenting a practical example to illustrate the potential costs associated with TSTs compared to LSTM when deploying these models in practice.

Let's assume a telecom operator has 1000 cells deployed and would like to predict their individual traffic levels four hours beforehand to know when to put each cell into energy saving mode, which will help save energy costs. To achieve this, they plan to deploy either an LSTM or Transformer model to an external cloud provider such as AWS or Azure. The estimated costs for the different machine learning services of these cloud providers, which are rough approximations based on Microsoft Azure's pricing web page [3], are as follows:

- Model Training: 0.9$ per hour.
- Model Storage: 0.02$ per GB per month.
- Model Inferences: 0.0001$ per second.

Using these costs in conjunction with the results obtained for each model in the previous sections, we can calculate the monthly expenses that the telecom operator would incur for deploying the LSTM and Transformer models. These monthly costs are shown in Table 4.4.1.

	LSTM	**Transformer**
Training monthly cost	Training time per model: $260\,seconds$ Total training time: $260 \times 1\,000 = 72.2\,hours$ Total cost: $72.2 \times 0.9 =$ **64.98$**	Training time per model: $116\,seconds$ Total training time: $116 \times 1\,000 = 32.2\,hours$ Total cost: $32.2 \times 0.9 =$ **28.98$**
Storage monthly cost	Storage per model: $1.6\,MB$ Total storage: $0.0016 \times 1\,000 = 1.6\,GB$ Total cost: $1.6 \times 0.02 =$ **0.032$**	Storage per model: $60\,MB$ Total storage: $0.06 \times 1\,000 = 60\,GB$ Total cost: $60 \times 0.02 =$ **1.2$**
Inference monthly cost	Inference time per model: $0.342\,seconds$ Number of inferences: $(4 \times 24 \times 30) \times 1\,000 = 2\,880\,000$ Total inference time: $2\,880\,000 \times 0.342 = 984\,960\,seconds$ Total cost: $984\,960 \times 0.0001 =$ **98.5$**	Inference time per model: $0.683\,seconds$ Number of inferences: $(4 \times 24 \times 30) \times 1\,000 = 2\,880\,000$ Total inference time: $2\,880\,000 \times 0.683 = 1\,967\,040\,seconds$ Total cost: $2\,880\,000 \times 0.0001 =$ **196.7$**
Total monthly cost	**163.51$**	**226.88$**

Table 4.4.1: Associated monthly costs of LSTM vs. Transformer trained on a four-hour forecasting horizon, showing both the cheaper and costlier model per machine learning service.

Based on this example, we observe that Transformer is 63.368$ or 38.75% costlier per month than LSTM, whilst providing 12% better prediction accuracy. This suggests that although TSTs typically provide superior predictive accuracy than LSTM, their deployment is costlier than that of LSTM due to their larger memory requirements and longer inference times. Therefore, when deciding to use TSTs over LSTM for KPI prediction, careful consideration should be placed on whether the added benefits of improved predictive accuracy outweigh the downside of higher deployment costs. In the above example, for instance, the amount of money that could be saved through the better power scheduling of each cell should outweigh the extra deployment costs, resulting in overall cost savings.

Chapter 5

Conclusions

In this Section, we discuss the conclusions that can be made from the work undertook in this project, and follow by proposing valid future work.

5.1 Discussion

In this project, four different novel Time Series Transformer architectures have been built, trained and evaluated for the task of predicting 4G and 5G traffic, thereby advancing the current knowledge repository of applying ML models for traffic prediction in telecom networks.

The findings show that TSTs exhibit superior performance compared to the traditional domain-dominant method, LSTM, in specific scenarios of traffic prediction requirements. Notably, TSTs display a substantial advantage in terms of prediction accuracy for longer prediction horizons. However, for shorter prediction horizons, TSTs do not show any significant improvement and yield comparable results to LSTM. It is important to note however that these results can vary depending on the datasets used, as well as the hyperparameter tuning process, which was not performed in this work due to time constraints. The advantage of TSTs is further reflected in their overall training time, as they require considerably less time to train than LSTM as the prediction horizon increases to achieve commendable results. This advantage can be attributed to the highly complex nature of TSTs, resulting from their efficient attention modules, enhanced positional encoding modules, and deep decomposition architectures. However, the increased complexity also leads to larger

memory requirements and longer inference times for TSTs when compared to LSTM, with our findings indicating that TSTs consist of a significantly larger number of model parameters than LSTM. Furthermore, it should be noted that the time and memory metrics can be affected by how efficiently the architectures are implemented in addition to what kind of hardware resources are available.

This work also offered valuable insights into the performance of TSTs and LSTM through a practical example. The example illustrated that while TSTs exhibit superior accuracy in traffic predictions, their deployment entails higher memory costs and longer inference times, leading to increased overall costs when compared to LSTM. This finding highlights the importance of operators carefully evaluating the trade-off between predictive accuracy and deployment costs when deciding on the most suitable model for their specific needs.

Overall, our findings emphasize the potential of Time Series Transformers as a promising alternative to LSTM, particularly in situations where accurate long-term predictions are essential. By considering both the strengths and limitations of TSTs and LSTM, industry professionals can make informed decisions to optimize their forecasting strategies and maximize the overall effectiveness of their operational processes.

5.2 Limitations & Future Work

A natural continuation to this degree project could involve conducting hyperparameter tuning for both the TST and LSTM models, which would provide a more extensive investigation into the performance of these models. This task was not feasible within the time constraints of this project, as the sheer amount of models and experiments required for a satisfactory set of results made it impractical to undertake during the short time frame of this project.

Another compelling direction would be to apply this project's findings to develop an optimal traffic prediction model for a specific use-case in the telecom industry. For instance, integrating the TST model into an existing forecasting pipeline, previously reliant on models like LSTM, presents an intriguing opportunity. Additionally, refining the TST models through techniques such as hyperparameter optimization or enhancing the positional encoding module to be better adapted to specific traffic

data could be explored. Another avenue worth exploring involves augmenting the traffic data with external sources, such as weather data [25], in conjunction with the TST model, to assess whether this combination can yield an optimal traffic prediction model.

Finally, another area for future research lies in assessing the TSTs' capacity for generalization across various cells spread across different geographic locations. This research topic could center on the development of techniques aimed at broadening the utility of TSTs for network-wide predictions, as opposed to solely operating at the cell level. This topic could significantly enhance our understanding of deploying TSTs on a large scale, a pivotal concern for numerous telecom operators.

Bibliography

[1] 3GPP. *The 3rd Generation Partnership Project*. URL: https://www.3gpp.org (visited on 03/14/2023).

[2] Azari, Amin, Papapetrou, Panagiotis, Denic, Stojan, and Peters, Gunnar. "Cellular traffic prediction and classification: A comparative evaluation of LSTM and ARIMA". In: *Discovery Science: 22nd International Conference, DS 2019, Split, Croatia, October 28–30, 2019, Proceedings 22*. Springer. 2019, pp. 129–144.

[3] Azure, Microsoft. *Pricing - Azure Machine Learning*. URL: https://azure.microsoft.com/en-us/pricing/details/machine-learning/?cdn=disable.

[4] Brown, Tom B., Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, Agarwal, Sandhini, Herbert-Voss, Ariel, Krueger, Gretchen, Henighan, Tom, Child, Rewon, Ramesh, Aditya, Ziegler, Daniel M., Wu, Jeffrey, Winter, Clemens, Hesse, Christopher, Chen, Mark, Sigler, Eric, Litwin, Mateusz, Gray, Scott, Chess, Benjamin, Clark, Jack, Berner, Christopher, McCandlish, Sam, Radford, Alec, Sutskever, Ilya, and Amodei, Dario. "Language Models are Few-Shot Learners". In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[5] Chatfield, Chris. *The analysis of time series: an introduction*. Chapman and hall/CRC, 2003.

[6] Chevalier, Guillaume. *The LSTM Cell*. URL: https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg (visited on 03/14/2023).

[7] Dao, Erik. "Incorporating Sparse Attention Mechanism into Transformer for Object Detection in Images". In: (2022).

[8] Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton, and Toutanova, Kristina. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[9] Dosovitskiy, Alexey, Beyer, Lucas, Kolesnikov, Alexander, Weissenborn, Dirk, Zhai, Xiaohua, Unterthiner, Thomas, Dehghani, Mostafa, Minderer, Matthias, Heigold, Georg, Gelly, Sylvain, Uszkoreit, Jakob, and Houlsby, Neil. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

[10] Du, Dazhao, Su, Bing, and Wei, Zhewei. “Preformer: Predictive Transformer with Multi-Scale Segment-wise Correlations for Long-Term Time Series Forecasting”. In: *arXiv preprint arXiv:2202.11356* (2022).

[11] Engati. *Vanishing gradient problem*. URL: https://www.engati.com/glossary/vanishing-gradient-problem#:~:text=Vanishing%20gradient%20problem%20is%20a,layers%20to%20the%20earlier%20layers. (visited on 06/12/2023).

[12] Guo, Jia, Peng, Yu, Peng, Xiyuan, Chen, Qiang, Yu, Jiang, and Dai, Yufeng. “Traffic forecasting for mobile networks with multiplicative seasonal ARIMA models”. In: *2009 9th International Conference on Electronic Measurement & Instruments*. IEEE. 2009, pp. 3–377.

[13] Jaffry, Shan. “Cellular traffic prediction with recurrent neural network”. In: *arXiv preprint arXiv:2003.02807* (2020).

[14] Kingma, Diederik P and Ba, Jimmy. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[15] Kitaev, Nikita, Kaiser, Łukasz, and Levskaya, Anselm. “Reformer: The efficient transformer”. In: *arXiv preprint arXiv:2001.04451* (2020).

[16] Li, Shiyang, Jin, Xiaoyong, Xuan, Yao, Zhou, Xiyou, Chen, Wenhu, Wang, Yu-Xiang, and Yan, Xifeng. “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting”. In: *Advances in neural information processing systems* 32 (2019).

[17] Li, Yaguang, Yu, Rose, Shahabi, Cyrus, and Liu, Yan. “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting”. In: *arXiv preprint arXiv:1707.01926* (2017).

[18] Lim, Bryan, Arık, Sercan Ö, Loeff, Nicolas, and Pfister, Tomas. “Temporal fusion transformers for interpretable multi-horizon time series forecasting”. In: *International Journal of Forecasting* 37.4 (2021), pp. 1748–1764.

[19] Liu, Shizhan, Yu, Hang, Liao, Cong, Li, Jianguo, Lin, Weiyao, Liu, Alex X, and Dustdar, Schahram. “Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting”. In: *International Conference on Learning Representations*. 2021.

[20] Liu, Yeqi, Gong, Chuanyang, Yang, Ling, and Chen, Yingyi. “DSTP-RNN: A dual-stage two-phase attention-based recurrent neural network for long-term and multivariate time series prediction”. In: *Expert Systems with Applications* 143 (2020), p. 113082.

[21] Ogre51. *In Time Series Forecasting, What Do You Think Is the Difference Between Seasonality and Cyclicity?* Month Year. URL: `https://ogre51.medium.com/in-time-series-forecasting-what-do-you-think-is-the-difference-between-seasonality-and-cyclicity-f4e8d9523d24`.

[22] Parmar, Niki, Vaswani, Ashish, Uszkoreit, Jakob, Kaiser, Lukasz, Shazeer, Noam, and Ku, Alexander. “Image Transformer”. In: *CoRR* abs/1802.05751 (2018). arXiv: `1802.05751`. URL: `http://arxiv.org/abs/1802.05751`.

[23] Rao, Yongming, Zhao, Wenliang, Zhu, Zheng, Lu, Jiwen, and Zhou, Jie. “Global Filter Networks for Image Classification”. In: *CoRR* abs/2107.00645 (2021). arXiv: `2107.00645`. URL: `https://arxiv.org/abs/2107.00645`.

[24] Sherstinsky, Alex. “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306.

[25] Shibli, Ali. “Hybrid Deep Learning Model for Cellular Network Traffic Prediction, KTH MSc Thesis”. In: (2022).

[26] St, Lars, Wold, Svante, et al. “Analysis of variance (ANOVA)”. In: *Chemometrics and intelligent laboratory systems* 6.4 (1989), pp. 259–272.

[27] Trinh, Hoang Duy, Giupponi, Lorenza, and Dini, Paolo. “Mobile traffic prediction from raw data using LSTM networks”. In: *2018 IEEE 29th annual international symposium on personal, indoor and mobile radio communications (PIMRC)*. IEEE. 2018, pp. 1827–1832.

[28] Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, and Polosukhin, Illia. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[29] Wang, Shaowei and Ran, Chen. “Rethinking cellular network planning and optimization”. In: *IEEE Wireless Communications* 23.2 (2016), pp. 118–125.

[30] Wass, Daniel. “Transformer learning for traffic prediction in mobile networks, KTH MSc Thesis”. In: (2021).

[31] Wen, Qingsong, Zhou, Tian, Zhang, Chaoli, Chen, Weiqi, Ma, Ziqing, Yan, Junchi, and Sun, Liang. “Transformers in time series: A survey”. In: *arXiv preprint arXiv:2202.07125* (2022).

[32] Wen, Ruofeng, Torkkola, Kari, Narayanaswamy, Balakrishnan, and Madeka, Dhruv. “A multi-horizon quantile recurrent forecaster”. In: *arXiv preprint arXiv:1711.11053* (2017).

[33] Wikipedia. *Autoregressive integrated moving average*. URL: https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average (visited on 03/14/2023).

[34] Wikipedia. *Autoregressive model*. URL: https://en.wikipedia.org/wiki/Autoregressive_model (visited on 03/14/2023).

[35] Wikipedia. *Autoregressive–moving-average model*. URL: https://en.wikipedia.org/wiki/Autoregressive-moving-average_model (visited on 03/14/2023).

[36] Wikipedia. *Decomposition of time series*. URL: https://en.wikipedia.org/wiki/Decomposition_of_time_series (visited on 06/11/2023).

[37] Wikipedia. *Mean absolute error*. URL: https://en.wikipedia.org/wiki/Mean_absolute_error (visited on 06/12/2023).

[38] Wikipedia. *Mean squared error*. URL: https://en.wikipedia.org/wiki/Mean_squared_error (visited on 06/12/2023).

[39] Wikipedia. *Time series*. URL: `https://en.wikipedia.org/wiki/Time_series` (visited on 06/11/2023).

[40] Woo, Gerald, Liu, Chenghao, Sahoo, Doyen, Kumar, Akshat, and Hoi, Steven. "Etsformer: Exponential smoothing transformers for time-series forecasting". In: *arXiv preprint arXiv:2202.01381* (2022).

[41] Wu, Haixu, Xu, Jiehui, Wang, Jianmin, and Long, Mingsheng. "Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 22419–22430.

[42] Xu, Jiehui, Wu, Haixu, Wang, Jianmin, and Long, Mingsheng. "Anomaly transformer: Time series anomaly detection with association discrepancy". In: *arXiv preprint arXiv:2110.02642* (2021).

[43] Xu, Mingxing, Dai, Wenrui, Liu, Chunmiao, Gao, Xing, Lin, Weiyao, Qi, Guo-Jun, and Xiong, Hongkai. "Spatial-temporal transformer networks for traffic flow forecasting". In: *arXiv preprint arXiv:2001.02908* (2020).

[44] Yang, Chao-Han Huck, Tsai, Yun-Yun, and Chen, Pin-Yu. "Voice2series: Reprogramming acoustic models for time series classification". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11808–11819.

[45] Yu, Yanhua, Wang, Jun, Song, Meina, and Song, Junde. "Network traffic prediction and result analysis based on seasonal ARIMA and correlation coefficient". In: *2010 International Conference on Intelligent System Design and Engineering Application*. Vol. 1. IEEE. 2010, pp. 980–983.

[46] Zerveas,
George, Jayaraman, Srideepika, Patel, Dhaval, Bhamidipaty, Anuradha, and Eickhoff, Carsten. "A transformer-based framework for multivariate time series representation learning". In: *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 2021, pp. 2114–2124.

[47] Zhang, Sheng, Zhao, Shenglin, Yuan, Mingxuan, Zeng, Jia, Yao, Jianguo, Lyu, Michael R, and King, Irwin. "Traffic prediction based power saving in cellular networks: A machine learning method". In: *Proceedings of the 25th ACM SIGSPATIAL international conference on advances in geographic information systems*. 2017, pp. 1–10.

[48] Zhou, Haoyi, Zhang, Shanghang, Peng, Jieqi, Zhang, Shuai, Li, Jianxin, Xiong, Hui, and Zhang, Wancai. “Informer: Beyond efficient transformer for long sequence time-series forecasting”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 11106–11115.

[49] Zhou, Tian, Ma, Ziqing, Wen, Qingsong, Wang, Xue, Sun, Liang, and Jin, Rong. “FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting”. In: *arXiv preprint arXiv:2201.12740* (2022).