Multistage Predictor for Cellular Network Throughput Prediction

Killian Nolan
119409382
Data Science & Analytics

2023

Abstract

Accurate throughput prediction is an important component of many of today's communication systems such as video streaming or the scheduling of large downloads. Today's world heavily utilizes wireless communication technologies such as 4G and 5G for such applications, however throughput prediction for wireless systems presents a unique challenge for application designers due to the highly dynamic nature of wireless networks. The variability observed in wireless systems stems from a combination of scheduling challenges at the base station, changes in channel condition from interference and signal degradation as well as the mobility of connected devices. Deep learning based throughput prediction models have previously been recognised as a good approach to tackling this problem due to the their ability to handle the high dimensionality and complexity of cellular data. This project aims to expand on these previous works by exploring the potential use of multistage deep learning models in wireless throughput prediction. The goal of the multi-stage models is to better accommodate for the imbalanced nature of cellular data and improve upon a traditional deep learning model's ability to generalize.

Declaration

I hereby declare that the contents of this paper and the work surrounding it are my own. Signed - Killian Nolan.

Contents

Introduction

Total mobile network traffic has doubled over the last two years and is expected to increase a by a factor of 4 by the year 2028. This is due to growth in the number of mobile devices as well as the use of 5G mobile networks [4]. Cellular networks are highly dynamic systems. The system conditions are highly variable and depend on various constantly changing factors such as the level of interference, location, line of site etc. Throughput is heavily dependent on the current condition of the network channel and as such can vary dramatically of the course of a few seconds. This presents a challenge for both the network itself, and the mobile devices that leverage the network. The network attempts to balance its available resources between all of its connected users and applications running on the mobile devices require knowledge of the available throughput in order to function correctly.

A notable use case of throughput prediction on cellular networks relates to video streaming. The vast majority of mobile network traffic is video streaming [4]. Video streaming applications require high network throughput and good consistency in order to deliver the video quality we expect in today's world. Video streaming requires that chunks of the video be downloaded in a timely manner before they are used. The application will dynamically adjust the size of the chunk of the video it attempts to download depending on throughput predictions. For example if the throughput prediction algorithm in use predicts a low download bitrate over the next few seconds the application will schedule the download of a lower quality video chunk as lower quality means less overall data. Failure to adapt to the current network environment may result in buffering or noticeable changes in quality throughout a video for the user. Accurate throughput prediction plays a key role in maintaining the quality of user experience in these applications [8].

Adoption of currently niche technologies such as AR (augmented reality) and VR (virtual reality) is projected to increase in the near future [14]. These technologies will make heavy use of cellular networks for device mobility and ergonomics. VR and AR devices require low latency and high throughput connections to function optimally. Right now this is achieved by tethering such devices to a network via a physical wire. This limits the potential use cases and adoption of such technologies. 5G can provide the latency and throughput requirements for optimal operation of these technologies allowing them to be used in more mobile devices. As such they stand to benefit from accurate throughput prediction.

Multistage models make use of 2 or more models to solve a given problem. This can be done when the problem can be broken up into smaller pieces which can be solved separately. The models may be linked together sequentially or in parallel depending on how the problem was decomposed. The intuition behind using a multistage approach as opposed to tradition

single model approaches is that throughput in a cellular channel is inherently imbalanced. As seen in this paper, typical download throughput levels fall in the ranges above 5Mbps. User experience is disproportionally influenced by low throughput situations such as ¡5Mbps or ¡1Mbps as opposed to differences experienced outside these bounds [8]. Models trained on smaller sub-ranges of the data should make better predictions in these ranges compared to a single model trained on data related to the entire throughput range.

3.1 Background on Cellular Networks

A basic understanding of the cellular network architecture will help with understanding the dataset used in this project. Cellular networks, be that 4G, 5G or any previous generation use electromagnetic waves to carry data. The access points for cellular networks are referred to as base stations or sometimes as cell towers. Base stations are what mobile devices such as your phone or laptop will connect to for its internet connection. As the main point of contact for mobile devices, we will focus our understanding of cellular networks solely on the interaction between mobile devices and base stations.

The provider of a cellular network is allocated a frequency band which it can use to provide internet connection. Typically mobile providers would have multiple bands to leverage for different use cases. Bands spanning higher frequency ranges are capable of carrying more data (higher bitrate), however they suffer from having short range and less of an ability to penetrate common obstructions such as walls. Lower frequency bands conversely, have longer range and a better ability to penetrate obstacles but a lower bitrate. As such lower frequency bands tend to be used to serve rural areas, where the longer range and better penetration are best utilised and higher frequency waves are used in densely populated areas such as cities or towns.

When a mobile device connects to a cellular network it is allocated a sub-section of the frequency band of the provider. The width of the band allocated to the mobile device depends on the bandwidth available as well as the bitrate required for the connection. For 4G LTE, the smallest sub-band a user can be allocated 180kHz in width. From this sub-band 12 orthogonal frequencies are chosen. Orthogonal in this case means that the electromagnetic waves do not interfere with one another. Data can be encoded in each of these waves individually. As such the bitrate a user has access to is a product of the number of orthogonal waves they are allocated (bandwidth) as well as the frequency of said waves.

With the deployment of 5G comes the use of higher frequency bands (24GHz +). Because of the shorter range and lower penetration at these frequencies, more base stations will be required to cover the same area compared to before. As such, it is likely such bands will be deployed in densely populated areas such as cities or towns where they will get the most use. Mobile devices will have to jump between base stations more often in order to maintain a connection. Obstructions are also abundant in such areas which may lead to intermittent connection disruption, from simple buildings to more dynamic obstructions such as cars or even other people. There is also more electromagnetic interference due to the density of devices that leverage wireless communication technologies. Accurate throughput prediction will play a vital role in the quality of service in such environments.

3.2 Throughput Prediction for Cellular Networks

Statistical estimation of cellular channel conditions is known to be highly difficult due to the variability of such environments [15]. Throughput prediction involves the use of various metrics available to a mobile device to estimate future throughput scenarios. Data used can be both present and historical and as such this problem is essentially a time series forecasting problem. Data available to a mobile device includes metrics relating to the physical layer connection between it and the base station, metrics from neighbouring base stations as well as location data. As it stands, data relating to the overall state of the network, as as the number of users connected to a base station or the available network bandwidth is not readily available to a mobile device further adding to the difficulty in estimation. As the state of the channel depends on a large number of factors, machine learning (ML) and deep learning (DL) techniques are likely well suited for tackling this issue. ML & DL have proven themselves in other networking related problems in the past [16]. LSTM (long short-term memory) networks are recognised as effective models for time series forecasting [13]. This paper explores the use of Lstm models in a multistage architecture to preform throughput prediction.

Problem Analysis

Accurate throughput prediction in wireless networks is inherently difficult due to the highly dynamic nature of wireless cellular networks. Given the data made available by the dataset used in this project, there were still a number of challenges that had be solved before any prediction model could be considered. This section outlines the dataset, and the understanding gleaned from it and applies this understanding to the inherent challenges faced in fitting a deep learning based throughput predictor.

4.1 Outline of the Dataset

The dataset used in this paper was collected by researchers in University College Cork in and around the greater Cork City area. Data was collected using an Android network monitoring application, G-NetTrack Pro. Apple devices currently do not have any equivalent application for collecting cellular network analytics. The dataset is a collection of 135 different traces approximately 15 minutes in length on average. Traces were collected by the UCC researchers under a number of different movement patterns. The traces are divided based on the following movement patterns:

•Static: The trace was collected while the mobile devices location remained fixed. This is characteristic of a common use case for mobile devices such as watching video while seated at a desk. Such a use case presents the best case scenario for a cellular network as the connection will experience low variability in its stability.
•Car: The trace was collected while travelling in Cork city and its surrounding suburbs by car.
•Train: The trace was collected while travelling by train. These traces contain a mix of both 4G and 3G as availability for 4G networks was in urban areas only at the time these experiments took place.
•Bus: Traces collected while using public transport around Cork City.
•Pedestrian: Traces collected while walking around Cork City center using different routes.

Traces were collected at a variety of times on both weekdays and weekends in order to provide adequate depiction of congestion patterns. The dataset includes of a number of physical layer metrics, as well has GPS metrics and the upload and download bitrate. The following description of the metrics collected was taken directly from the paper [9] written by the researchers involved in the construction of this dataset. For a more in depth

understanding of the dataset I recommend reading their paper. All credit goes to them for the following description of the metrics:

•Timestamp: timestamp of sample
•Longitude and Latitude: GPS coordinates of mobile device
•Velocity: velocity in kph of mobile device
•Operatorname: cellular operator name (anonymised)
•CellId: Serving cell for mobile device
•NetworkMode: mobile communication standard (2G/3G/4G)
•RSRQ: value for RSRQ. RSRQ Represents a ratio between RSRP and Received Signal Strength Indicator (RSSI). Signal strength (signal quality) is measured across all resource elements (RE), including interference from all sources (dB).
•RSRP: value for RSRP. RSRP Represents an average power over cell-specific reference symbols carried inside distinct RE. RSRP is used for measuring cell signal strength/coverage and therefore cell selection (dBm).
•RSSI: value for RSSI. RSSI represents a received power (wide-band) including a serving cell and interference and noise from other sources. RSRQ, RSRP and RSSI are used for measuring cell strength/coverage and therefore cell selection(handover) (dBm)0
•SNR: value for signal-to-noise ratio (dB).
•CQI: value for CQI of a mobile device. CQI is a feedback provided by UE to eNodeB. It indicates data rate that could be transmitted over a channel (highest MCS with a BLER probability less than 10%), as the function of SINR and UE's receiver characteristics. Based on UE's prediction of the channel, eNodeB selects an appropriate modulation scheme and coding rate.
•DL_bitrate: download rate measured at the device (application layer) (kbit/s)
•UL_bitrate: uplink rate measured at the device (application layer) (kbit/s)
•State: state of the download process. It has two values, either I (idle, not downloading) or D (downloading)
•NRxRSRQ & NRxRSRP: RSRQ and RSRP values for the neighbouring cell.
•Cell_Longitude & Cell_Latitude: GPS coordinates of serving eNodeB. We use OpenCelliD4, the largest community open database providing GPS coordinates of cell towers.
•Distance: distance between the serving cell and mobile device in metres.

4.2 Considerations of the Dataset

As the dataset is a collection of separate traces (experiments), this must be taken into account when constructing train and test splits. The entire dataset cannot be viewed as a series of disconnected time series as traces start from different physical locations, run for different lengths of time and use the same workload as a starting point for measuring network data. As such special care must be taken in order to ensure proper construction of train-test sequences.

The dataset contains considerable missing values for some network features such as NRxRSRP & NRxRSRQ. Lstm based machine learning techniques require complete data. As such imputation had to be carried out to fill in these gaps. There are various methods of imputation such as mean imputation, max or min imputation, K nearest neighbours (knn) based imputation [1] methods and more. Some of these methods were considered in this project but it is important to note that no imputation method is perfect. Understanding of the missingness observed in the data may lead to choosing one method over the other.

Methods like knn imputation also require considerable computational power & space in memory compared to other more simple methods. This limits the usefulness of such a method to the training phase of a throughput predictor only. Knn or other complex imputation methods might not be viable for deployment directly on the mobile device due to the space and computational constraints of mobile devices. Edge computing would allow for more complicated imputation methods however there is also a time penalty incurred for utilising more complex methods.

Some features collected are not reported directly by the G-NetTrack Pro such as cell tower location data. As such these features were excluded from consideration in this project. If such information was made more readily available to mobile devices in the future it may prove useful in throughput prediction applications however this is outside the scope of the project.

While this dataset is robust in its construction and depiction of typical mobile communication system scenarios, it is not universal. Mobile network infrastructure implementations vary heavily by location. This dataset contains a mix of 2G/3G/4G networks. Older wireless communication technologies have different network environment distributions. As technology continues to progress the models will suffer from distribution shift.

4.3 Identifying Missing Values

Firstly we aimed to understand the nature of missing values included in the dataset. The models considered in this paper require complete data for training. Understanding the nature of missing values in the dataset is vital in the process of selecting methods of imputation[3]. From figure 4.1 we observed that physical layer features SNR and CQI were unreported in approximately 30% of sample observations with RSSI being unreported in 37%. Cell tower related features also exhibited missing values in approximately 30% of sample points. The degree of missingness rules out excluding observations that containing missing values from consideration.

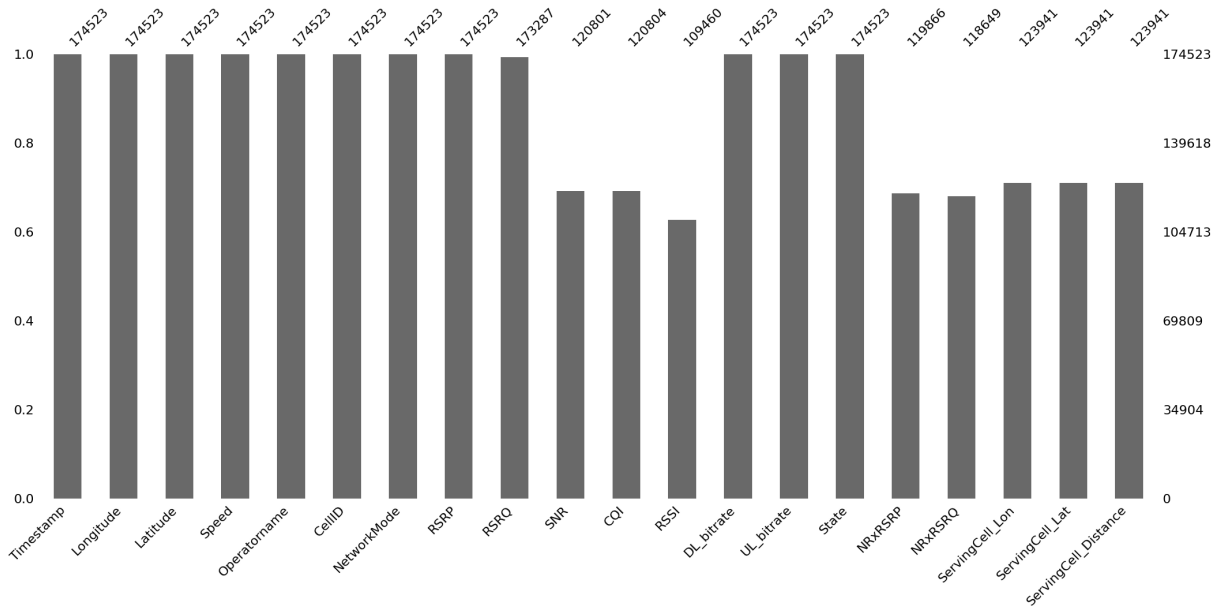

Figure 4.1: Bar chart showing the total number of missing values for each feature over all traces.

We then checked for correlation between observations of missing values in each column. From Figure 4.2 we observed a strong correlation between when the mobile device failed to report a value for the physical layer features RSSI, CQI and SNR. Geo-location data for the serving cell tower also exhibits strong positive correlation with the physical layer features. NRxRSRQ & NRxRSRP show moderate negative correlation with for missing values with the previously mentioned features. Figure 4.3 provides an alternative way to view these relations.

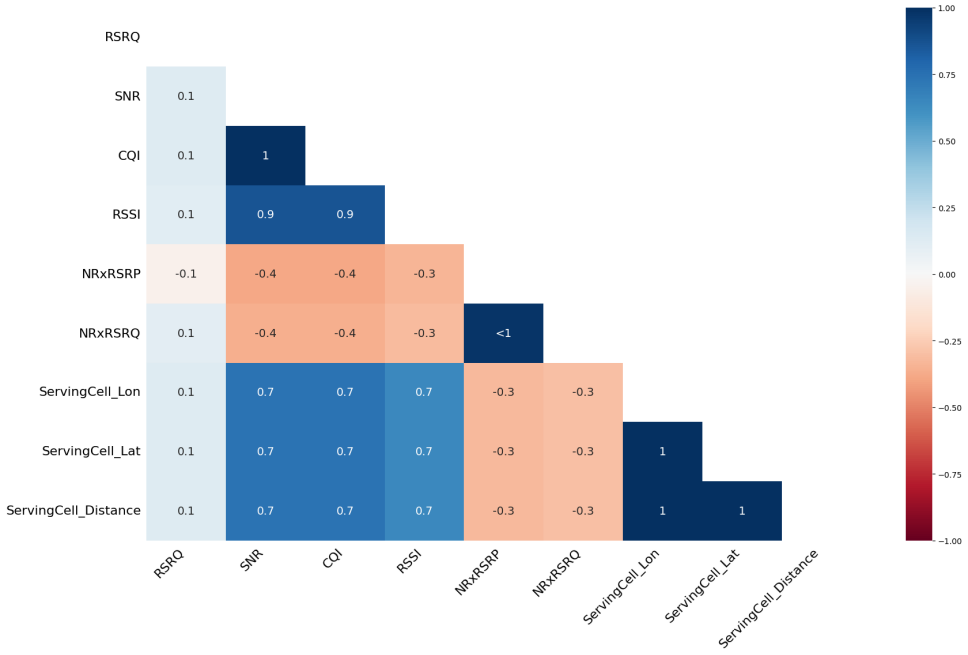

Figure 4.2: Correlation Heatmap for Missing values

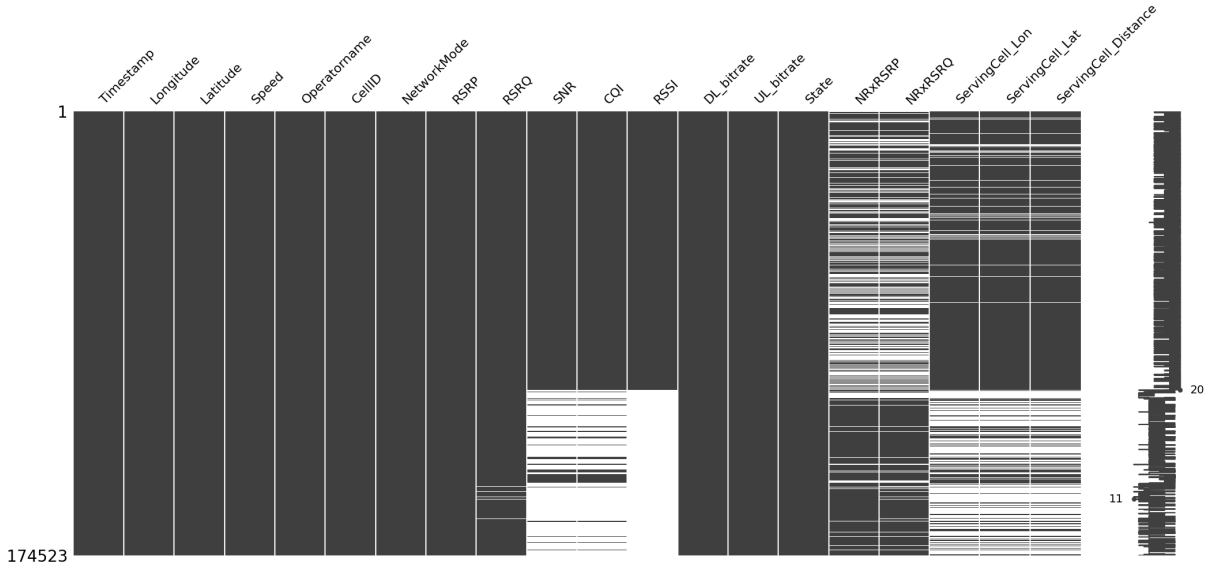

Figure 4.3: White regions indicate a missing value for that row. Diagram was sorted by RSSI to group missing rows together.

From this brief analysis we can conclude that the majority of the missingness observed in the dataset is MNAR (missing not at random). Attempting to impute these missing values using simple imputation methods such as mean imputation would have introduced bias into the analysis [3]. We concluded that figure 4.3 identifies a reasonable cause of the missing entries. The strong positive correlation between missing values in physical layer features

and the geo-location features of the serving cell suggest that the mobile device is on the edge of the current serving cell's range. This hypothesis is further reinforced by the negative correlation observed with the missing entries in neighbouring tower's physical layer features. The negative correlation suggests that neighbouring towers are more likely to be reporting values for RSRP and RSRQ when the current cell tower is not reporting values for CQI, SNR, RSSI and its geo-location data. This would make sense if the mobile device is moving into the range of a neighbouring cell tower. Assuming this hypothesis were true, it would help to inform the selection of adequate methods of imputation for each feature. This is considered in the later section 4.10.

4.4 Distribution of Features & Outlier Detection

The distribution of Dl_bitrate values across all traces is shown in 4.1 with 4.4 showing the distribution of the majority of observations.

Table 4.1: Distribution Statistics of Dl_bitrate in Mbps

Mean	Standard Dev	Median	Range
10.57	13.82	5.24	[0,168.96]

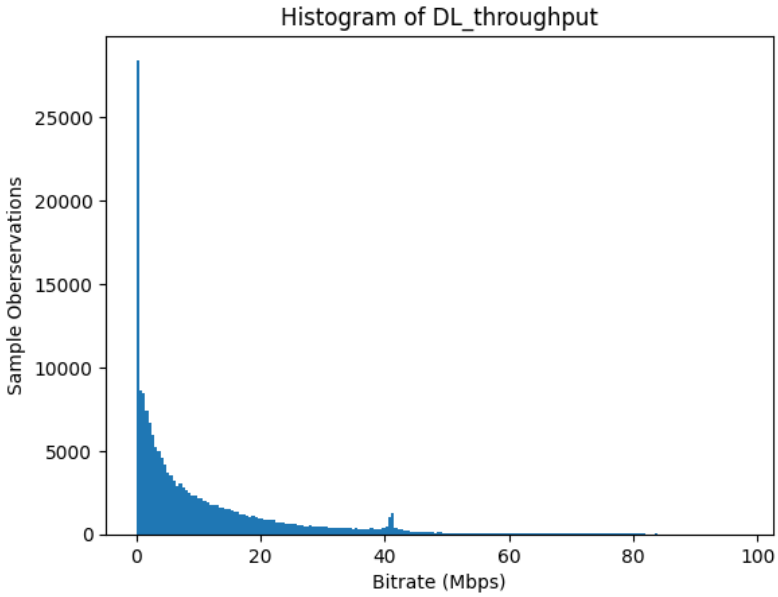

Figure 4.4: Distribution of observations of download throughput over all traces

To show the contribution of movement patterns to the variability observed in wireless cellular network we can look at the summary statistics shown in 4.2. The "train" movement pattern experienced the lowest mean throughput. This is likely due to the reduced 4G coverage along the train's route. Following this we plotted the duration of each trace and highlighted the mean in 4.5. There is one notable outlier amongst traces who's duration is much longer than the average. This trace, trace 116, was collected while travelling by train. Because the experiment runs at least until the download of the control file is complete, it

is likely that the low download throughput observed for the train mobility pattern caused this deviation.

Table 4.2: Distribution Statistics of Dl_bitrate (Mbps) per Movement Pattern

Movement Pattern	Mean	Standard Dev	Median	Range
Static	10.56	16.66	3.11	[0,95.51]
Car	13.54	14.19	9.06	[0,144.46]
Pedestrian	10.73	11.79	6.36	[0,77.42]
Bus	9.77	11.18	5.83	[0,85.81]
Train	4.87	12.25	0.49	[0,168.96]

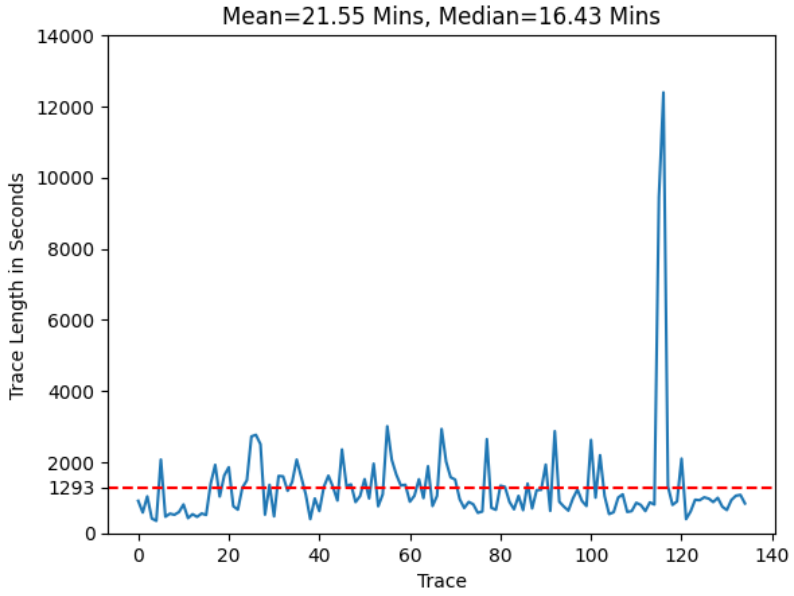

Figure 4.5: Plot of Trace Durations

The distributions of all features were checked for normality using the shapiro-wilk normality test [11]. It was found that no numeric features followed a Gaussian distribution. We then checked for outliers in each feature using the interquartile range method. X is an outlier if:

$$X < Q1 - 1.5 \times IQR \quad OR \quad X > Q3 + 1.5 \times IQR \tag{4.1}$$

A considerable number of outliers were identified in the dataset. Of the 174,523 observations, Dl_bitrate contained 13,245 outliers. Considering nature of wireless cellular networks outliers are inevitable and as such no strict measures were taken to deal with the outliers identified. It was found the expressing some physical layer features as a weighted exponential mean of the current and previous values improved predictions slightly, and as such this was the only action taken to deal with outliers.

4.5 Sequence construction

The models were trained on input, target sequences constructed using a rolling window over the observed time series data. Sequences were created on a trace by trace basis, i.e. there is no 1 sequence that contains observations from 2 different traces.

For a given pair of input and target, the input sequence is the observed values of the predictor features over the past p seconds. The target is the observed $DL\_bitrate$ of the next k seconds. The variables p and k are referred to as the history and horizon window respectively. A single input sequence takes the shape of a [n x f] matrix where f is the no of features used to predict the target. A single target sequence takes the shape of a [k x 1] matrix as we are only interested in predicting $DL\_bitrate$. For example in the univariate case we used a history window of 10 seconds, a horizon window of 5 seconds. Previous values of $DL\_bitrate$ were used to predict the throughput horizon. E.g.

$$\begin{aligned} input &: [10 \times 1] \\ target &: [5 \times 1] \end{aligned} \tag{4.2}$$

4.6 Choice of History & Horizon Windows

A typical input for an Lstm model used in forecasting takes the shape of: $[n, p, q]$ where n is the number of examples, p is the length of the history window and q is the number of features. The output will then be in the form of $[n, k, s]$ where k is the length of the horizon window and s is the number of target variables the model predicts, typically s=1. Fig 4.6 illustrates the construction of a single example in the univariate case.

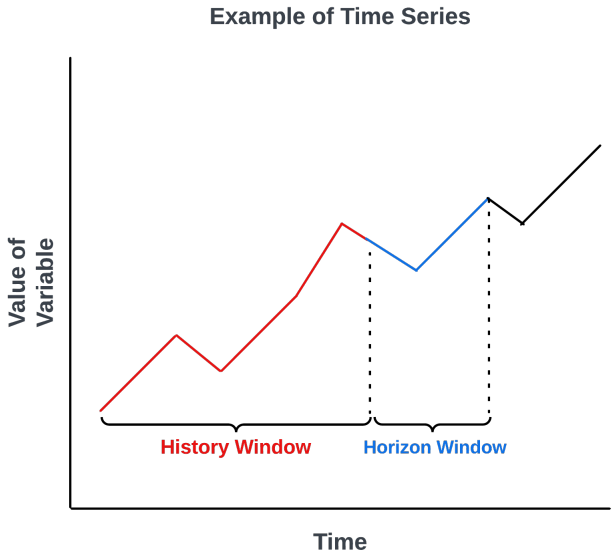

Figure 4.6: Graph showing how a history and horizon window can be constructed for a feature in time series data

There are many methods for identifying the optimal length of the horizon and history windows. Well studied methods such as the ACF (auto correlation function) or PACF

(partial auto correlation function) [10] are useful for identifying temporal dependencies in the univariate case. The multivariate case is more complex as the target depends not only on previous values of itself, but previous values of other features. Still ACF provided a useful insight into the choices of history windows to consider. An ACF of ¡0.4 indicates that including values at the time lag does not contribute greatly to predicting the current value. An ACF ¿0.8 indicates that a particular lag is very relavant when predicting the current and future value of a metric.

Looking at figure 4.7 which shows the average ACF of the download throughput over the previous minute we observed that significant contributions to the prediction of current values falls off after 26 seconds. The red shaded region shows the standard deviation of the ACF observed over all traces. Intuitively this also suggests that for a horizon window of 5 seconds, 21 seconds would be an adequate maximum bound to consider for the history window.

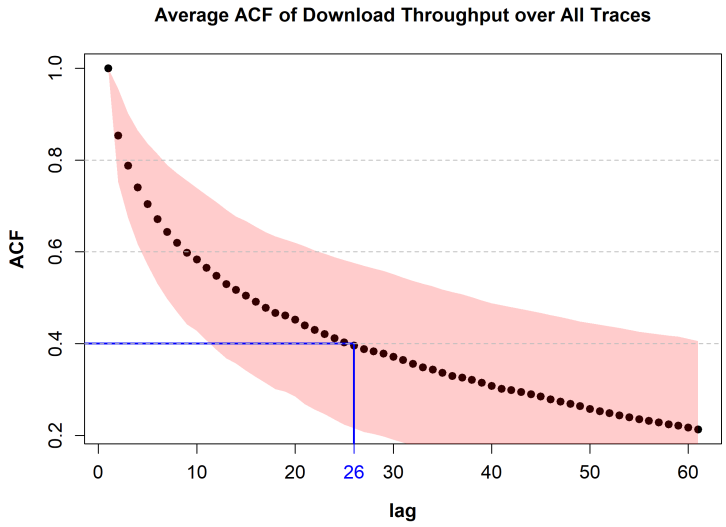

Figure 4.7: This figure shows the average value of the auto correlation function for download throughput over all traces

A selection of different combinations of horizon and history windows were considered to evaluate the impact each has on the throughput horizon predictions, however it was found that longer horizon lengths than 15 seconds caused model performance to decrease. An important note is that the choice of history and horizon window also impacts the choice of model. The model used to test these varying windows was tuned for a window size of 10 for history and 5 for horizon. It is possible that tuning the hyper-parameters for a model with longer windows may lead to increased performance however due to computational limitations this could not be explored within the scope of the project. The horizon window is more application dependent. The granularity of data available to us through G-NetTrack Pro is one second. In general more time sensitive applications such as web-conferencing would benefit a smaller horizon window. This incurs the cost of more computational power as the throughput prediction model must be run more frequently. Inversely, less time sensitive applications could use a longer horizon window. For throughput prediction, if the average horizon throughput is the main variable of interest then a longer horizon length

may actually benefit predictions as the average will be calculated over a greater number of throughput predictions [8].

4.7 Selection of train/test Splits

Typical train/test splits divide the dataset in a given ratio, usually 80:20 for train and test respectively. This is difficult to reproduce with this dataset as the dataset is not a single continuous time series but a collection of 135 distinct time series experiments. Train and test can be created by dividing the dataset based on the number of traces however, as seen in **??** the traces are of varying length making it difficult to split the dataset based on solo on the count. Another issue is that the training set must contain an adequate count of examples for all 3 classes, low, medium and high. As the classes are imbalanced, blindly partitioning the dataset based on traces may lead to situations where all examples of a single class ends up in either the training dataset or the testing dataset. This is especially the true for low throughput examples which have the least representation in the dataset. Both the baseline model and the multistage architectures must use the same train and test in order to be comparable. Failure to provide adequate examples of each class in the training data would result in the multistage models performing poorly. Instead, division of the traces was viewed an an optimisation problem. For each trace, the number of observations for each class of low, medium and high throughput was calculated. The number of observations of each class will vary based on the chosen horizon length and as such, different choices of horizon length will require a different train and test partition. The output of this process is as follows: The class counts per trace were then used to calculate a percentage of the

Table 4.3: Example of Label Counting Output

Trace No.	Low Sequence Count	Medium Sequence Count	High Sequence Count
0	10	30	100
1	4	32	70
2	0	60	130

total number of observations in each class over all traces contained within each trace. These percentages are then used partition the 135 traces in a given ratio (4:1 was used) in order to achieve the desired distribution of sequences in the train/test sets. Ideally the train set would include:
- 80% of all low samples, 80% of all medium samples and 80% of all high samples
- Test would be the remainder i.e 20% of low, medium and high

In practice it is difficult to find a split of traces that achieves the perfect ratio between low/medium/high for train and test. Instead a large number (10000) of potential splits are created and their class % are measured. The split that is chosen is the train/test split with the lowest sum of differences between % training low, medium, high, that is to say, train/test split chosen has the closest distribution to the desired distribution outlined above. The preprocessing code provides a summary of the distribution of the train/test split it finds. For example, most of the analysis in this paper focuses on a history window

of 10 seconds and a horizon window of 5 seconds. The distribution of the train / test split for this selection is shown in table 4.4.

Table 4.4: Train/Test Class Distribution

train low	train medium	train high	test low	test medium	test high	distribution diff
0.864 66	0.836 30	0.837 40	0.135 34	0.163 70	0.162 60	0.056 72

4.8 Identifying Bounds for Mutlistage Model Classes

Two multistage approaches for modelling throughput prediction were considered in this paper. Each of the two architectures however share the same base models. Both multistage architectures proposed make use of a simple classifier that aims to predict a horizon throughput as one of 3 distinct situations, low throughput, medium throughput or high throughput. The choice to divide the problem based on the value of the download throughput comes from the fact that traces typically reported throughput above certain bounds (5Mbps, 1Mbps etc) far more often than below them. This imbalance leads to a single stage model frequently overestimating in low/medium throughput situations. Overestimating in these low or medium throughput scenarios leads to a noticeable decrease in the perceived quality of experience in applications such as video streaming [8] and is one of the main motivations of the multistage approach.

The choice of the number of bounds (and subsequently models) and how to classify a given sequence of input data is arbitrary. For the purposes of this paper, dividing the data into one of three bounds sufficed. The sequences were divided by horizon download throughput into the three following ranges:

•Low: $mean[y] < 1Mbps$
•Medium: $1Mpbs \leq mean[y] \leq 5Mpbs$
•High: $mean[y] > 5Mbps$

For forecasting, Lstm models are typically trained on x, y pairs where x is historic data of the predictor features over the past p seconds and y is the true value of the target variable of the next k seconds. The average download throughput of y was used to classify a given trace as an example of high, medium or low throughput.

The bounds 1Mbps and 5Mbps were chosen based on domain knowledge. Popular video streaming applications provide guidelines for the required download throughput for given video quality. As of the time of writing, 720p video requires 3Mbps on Netlfix whereas Youtube and Amazon Prime Video require at least 5Mbps of download throughput. While the adoption of AV1 hardware encoding may lower the required bitrate for HD video, use of the codec is still a while off mass market adoption and providers could choose to maintain the current bitrate while providing better quality source content.

4.9 Feature Selection

Feature selection is an important step in any machine learning problem. Good feature selection is known to improve loss, increase runtime performance and improve the understandability of predictions [5]. Feature selection for multivariate time series is still an evolving field. The challenge is that the a selected feature set must capture the correlation between the target variable and predictors as well as between the target variable and lagged values of the predictors.

Feature selection is important for the analysis of multistage vs single stage throughput predictors explored this paper as it is an effective method in reducing the number of parameters

of a given model. This is especially important when tight constraints are put on model size (in memory) and speed of inference, as is the case for models intended for mobile devices.

The number of feature included in the dataset is relatively small at 19 by current deep learning standards. This allows for methods such as exhaustive search to be viable, however hardware limitations prevented us from carrying out such an analysis due to time constraints. Geo-location data related to the serving cell tower was excluded from consideration. This data is currently not readily available on mobile devices and had to be gathered from a third party source .

Firstly to explore the correlation between features. Fig 4.8 shows the average correlation matrix over all traces. The features: RSRQ, RSSI, RSRP, NRxRSRP and NRxRSRQ are measured on a logarithmic scale and as such were transformed using the following formula before calculating the correlation matrix:

$$Y = exp(|X|) \tag{4.3}$$

NetworkMode_{X} is the one-hot-encoded transform of the NetworkMode feature, the same is true for State_I and State_DBefore performing this transform correlation between DL_bitrate and these features was not observed. Understanding this matrix requires some careful thought. Firstly, this is a time series problem, DL_bitrate will be used to predict itself. Knowing this, the observed strong correlation between DL_bitrate and UL_bitrate is misleading in regards to feature selection. This matrix suggests that including UL_bitrate might actually be redundant. SNR and CQI are good candidates for inclusion in the optimal feature subset. RSRQ and NRxRSRP are also good candidates for inclusion. RSRQ has strong correlation with both CQI and SNR suggesting that one of either SNR or CQI could be also be dropped should constraints call for it. Geo-location data had no correlation with DL_bitrate which is to be expected as the dataset was constructed from mobile devices in a relatively small geographic area. The NetworkMode variables were one-hot encoded and as such correlation is less applicable for these features. This matrix suggests that a good feature selection would include:

-DL_bitrate
-SNR
-CQI
-NRxRSRP
-RSRQ

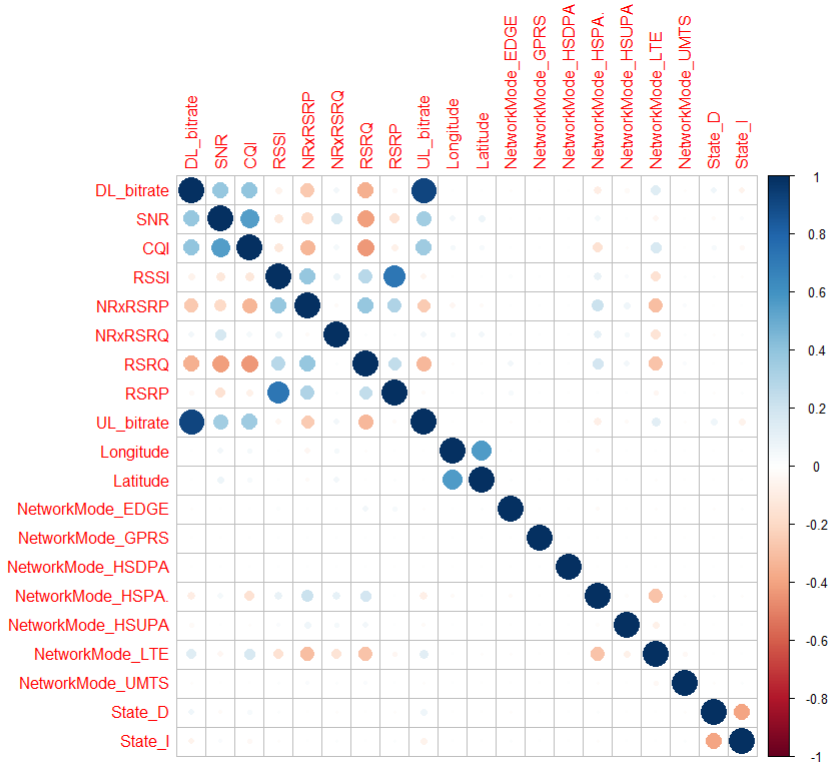

Figure 4.8: Correlation matrix showing the average correlation between features over all traces

Next permutation feature importance was considered. An Lstm was fit on all network related features, as well as location data for the mobile device. The Lstm used was the standard Lstm identfied in section 5.1 The metric we chose to compare the results of this test is the mean absolute percentage error (MAPE) given by the following formula:

$$MAPE = \frac{1}{n} \times \frac{|true - predicted|}{true} \times 100 \tag{4.4}$$

where n is the number of observations in the test set.

The procedure for permutation feature importance is as follows:

- Given a target variable Y, with predictor variables $X = \{X_1, X_2, X_3, ...X_n\}$
- Create a train/test split of the dataset
- Fit a model on the training data using all features in X
- Compute the loss of the model on the test set to use as a baseline
- Randomly shuffle the values of one feature X_k in the test set and compute the loss with just this feature's values shuffled. Do this $\forall X_k \in X$.
- Compared the performance of the model on the test set with the performance of the model on each test set with a shuffled feature
- Rank the features based on the difference in performance
- Use this ranked list to identify possible features to eliminate

The results of this analysis can be seen in figure 4.9. In this case the model relied heavily on the NetworkMode feature. RSRQ was once again important in improving prediction performance.

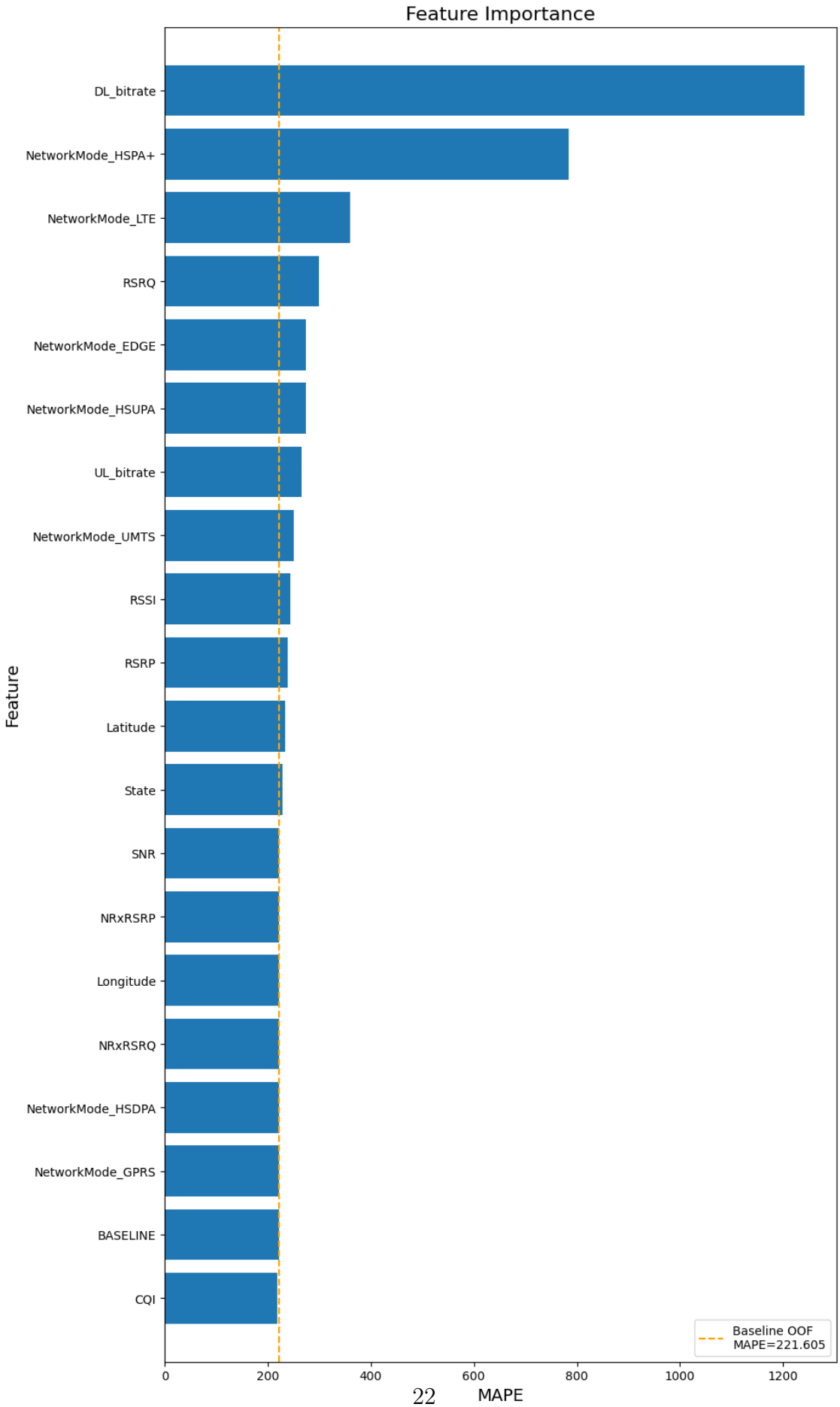

Figure 4.9: Bar Chart showing Difference in Model Performance with Permuted Feature Column.

4.10 Imputation & Scaling

In section 4.3 we observed that the majority of missing values can be categorised as MNAR (missing not at random). The following imputation steps were considered:

- SNR: Imputed with the minimum observed value
- CQI: Imputed with the minimum observed value
- RSSI: Imputed with the minimum observed value
- ServingCell_Lon: Imputed using forward fill
- ServingCell_Lat: Imputed using forward fill
- ServingCell_Distance: Imputed using forward fill
- RSRQ: Imputed using KNNImputer(k=5) from sklearn
- RSRP: Imputed using KNNImputer(k=5) from sklearn
- NRxRSRQ: Imputed using KNNImputer(k=5) from sklearn
- NRxRSRP: Imputed using KNNImputer(k=5) from sklearn

This method of imputation was compared with using the K-nearest-neighbours algorithm on all numeric features and ultimately the choice between these two methods of imputation made little to no difference in the model's performance.

As observed in section 4.4 the distributions of features did not follow a normal distribution. Z-score based scaling works best when the data is already normally distributed. As such we chose to use min-max scaling in the range of (0,1) to preserve the actual feature distributions. Categorical features were one-hot-encoded.

Design

The focus of this paper is to explore the differences between a single model and a multistage architecture for throughput prediction in wireless networks. The choice of machine learning or deep learning model must therefore be consistent across all models considered to avoid confusing the issue with that of comparing one ML (machine learning) or DL (deep learning) method of regression with another. Lstm (long short-term memory) networks are recognised as effective models for time series forecasting [13], with deep learning models in general performing well in networking related tasks [16]. As such any individual model considered in this paper is some form of Lstm deep network. Initial comparisons of multistage vs a single model ensure that all Lstm models have the same hyperparameters. That is to say, all models have the same number of nodes, layers, the same dropout chance, activation function, optimizer, etc. The only difference between the individual models considered will be that the classifier model has a different output layer to the traditional regression models as it aims to predict the class of the input sequence as opposed to the DL_bitrate horizon. In total there are 5 individual Lstm models that had to be trained with brief description given in 5.1

Table 5.1: Lstm Deep Models Used

Model	Training Data	Outputs
Baseline	All TP examples	DL_bitrate prediction horizon
Low	Only Low TP examples	DL_bitrate prediction horizon
Medium	Only Medium TP examples	DL_bitrate prediction horizon
High	Only High TP examples	DL_bitrate prediction horizon
Classifier	All labelled TP examples	Class probability table

In 6.4 the number of parameters of the Lstm models were altered in various ways in order to achieve memory size parity between the single baseline throughput predictor and the multistage predictors. This is an important point to explore as adoption of a multistage approach vs a single model on mobile devices will require that both methods have comparable performance on hardware. A multistage approach could only be considered more optimal if the required memory and inference time were comparable. By design a multistage approach will be bigger than a single model, as such the total number of parameters of both approaches should be equalised for further comparison.

5.1 Model Tuning

A hyperparameter is a parameter whose value is set before the learning process begins. In machine learning, a hyperparameter is a configuration variable that is used to control the training process, as opposed to the parameters of the model such as weights and biases, which are learned from the data. For Lstm deep models there are a number of hyperparameters than can be altered by the user, a few of note are:

- Choice of optimiser
- Learning rate
- Number of hidden nodes
- Number of layers
- Inclusion of dropout layers
- Dropout chance

Hyperparameter tuning is the practice of optimising the hyperparameters of a model in order to achieve better model performance. The impact of hyperparameters is wide reaching. The choice of hyperparameters not only affects the predictive performance of the model and model size, but also impacts less obvious metrics such as inference latency [6]. For models intended for low power devices such as mobile devices and IoT (internet of things) devices, hyperparameter tuning is even more important, as it can influence the battery consumption of the device [6]. The issue is with hyperparameter tuning is that the choice of hyperparameters leads to an infinite set of possible models. There are many methods for tuning deep learning models. As the models were implemented in Tensorflow, we made use of KerasTuner [7], a tuning framework that allows for a low code solution to this issue. Random search was used to explore the possible set of combinations of hyperparameters as it is known to perform better than similar tools such as grid search [2].

It must be noted that due to hardware and time limitations, only 10 such combinations of hyperparameters were considered for each model. The model was trained on each set of hyperparameters 3 times, in an attempt to account for different initialisation states leading to differences in model performance. This process was by no means extensive and given more time and computational power, a larger set of hyperparameters should be considered.

5.2 Multistage One

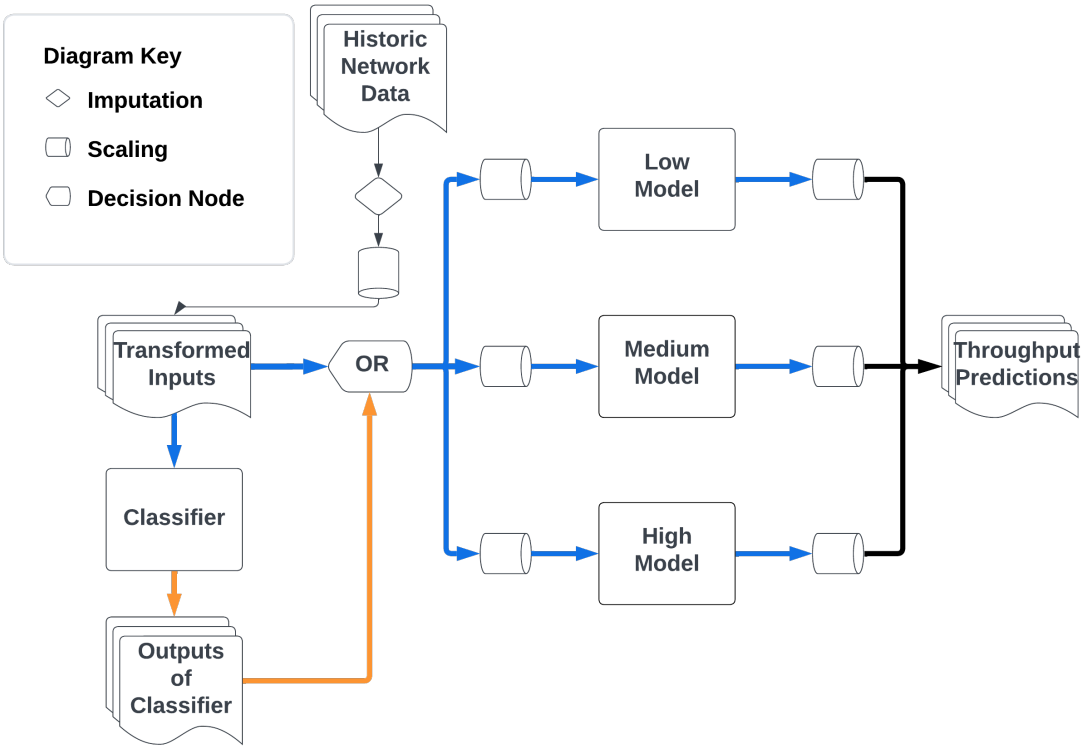

Figure 5.1: Multistage One model.

The multistage throughput predictor herein referred to by "multistage one" uses a classifier to predict the most likely throughput class of the horizon for a given history sequence. As previously discussed, three classes are considered, high, medium and low, which describe the group the horizon throughput based on sub-ranges. Refer back to section 4.8 for a detailed description on the construction of these classes from the dataset. Based on the classifier's decision, the input is passed to one of the three regression models, each trained exclusively on examples of sequences of their respective class. As only one model is chosen, performance of this architecture is heavily dependent on correct classification of the horizon throughput scenario by the classifier. Misclassification of an input sequence will lead to the wrong regression model being chosen for throughput prediction negating the advantage of the multistage approach vs a single model entirely. Figure 5.2 shows the higher level design of the multistage one model.

5.3 Multistage All

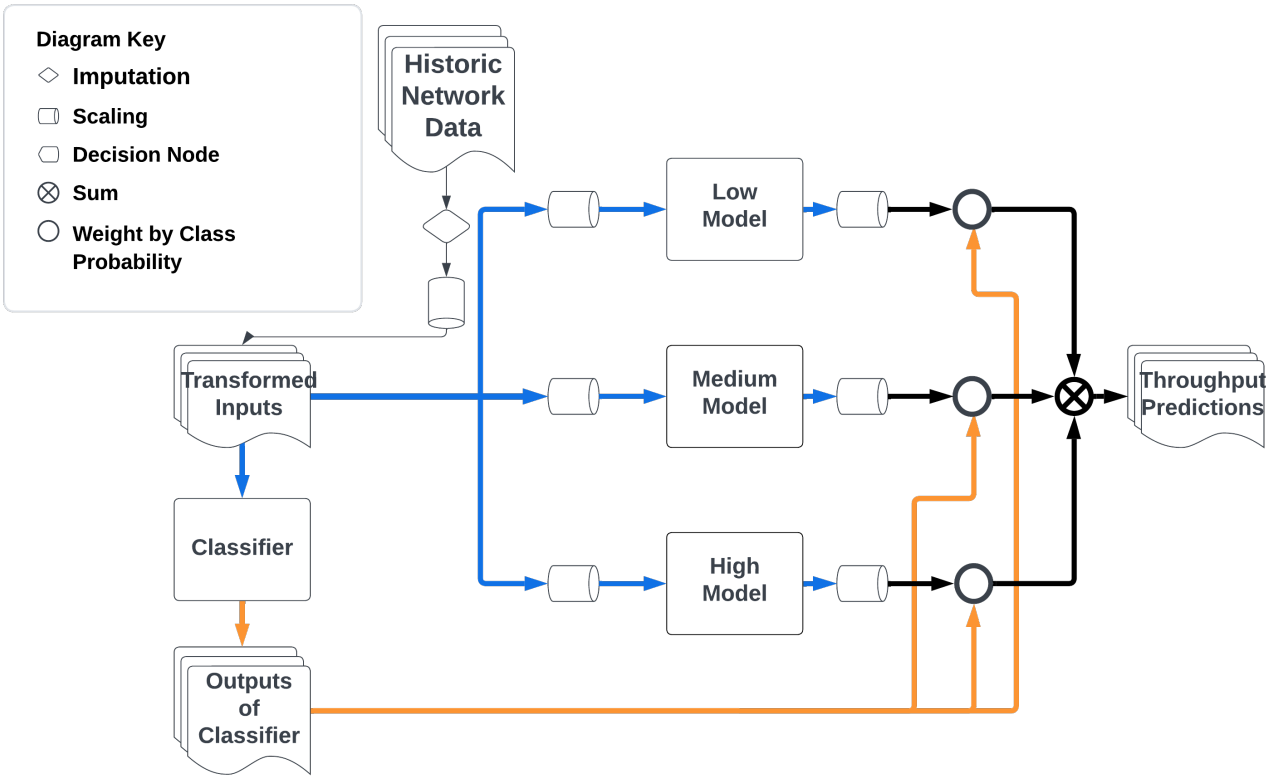

Figure 5.2: Multistage All model.

The multistage throughput predictor herein referred to by "multistage all" uses a classifier to predict the probability of each of the classes of the horizon for a given history sequence. Unlike the multistage one model, the input sequence is then passed to all three regression models. The outputs of the regression models are then weighted by the class probability vector from the classifier and summed to create the final prediction. This is essentially an ensemble model. Ensemble models are known to perform better than using a single model [12]. This model should perform better in situations where the classifier made less definite predictions on the input sequence's class. In theory, sequences on the border of two classes are better accommodated for by this model compared to the multistage one model. Figure 5.3 shows the higher level design of the multistage all model.

Results

6.1 Testing Framework

To properly test the multistage approaches we first had to prove the rational to construct the multistage models based on the division of the range of DL_bitrate. To do this we first test the individual regression models on their respective test sets and compare this to the baseline model's performance on the same test set. This shows that a model trained on a restricted range will be better at predicting values within that range. We then compare the performance of the multistage one, multistage all and baseline model on the complete test set. The complete test set is the baseline model's test set or the sum of the low, medium and high model's test sets. This shows actual performance of the multistage approaches as test sequences will be passed through the entire multistage frameworks for inference. We then considered the multistage models performance on the low and medium test sets in particular vs the baselines performance to identify in where difference in overall performance stemmed from. Results were calculated on the actual scale of the data in order to improve the understandability. The following metrics were considered in the comparison:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- Mean Absolute Percent Error (MAPE)
- Residual boxplots
- Absolute Percent Error boxplots

Relative error metrics such as MAPE are the most important in this application as they take into account that errors in lower throughput situations are more critical than errors in high throughput situations. I.e. A difference of 2Mbps in the prediction vs the true throughput when the predicted value is 152Mbps and the true value is 150Mbps is far less important than a difference in 2Mbps when the predicted throughput is 3Mbps and the true throughput is 1Mbps. All tests were run on the same system. Specs are as follows:

- Cpu : Ryzen 5 1600 (6 cores / 12 threads) 3.6Ghz
- GPU : Gtx 1050ti
- Ram : 32GB cl 3200

6.2 Constricted Throughput Models Vs Baseline Model

In order to prove the validity of constructing multi-stage models from a collection of models trained on constricted throughput classes (low, medium and high), we first proved that the models trained exclusively on examples of a respective throughput class perform significantly better than a baseline model trained on examples from all classes. As such the models considered in this section all share the same hyper-parameters identified from the tuning described in 5.1. The following features were used as inputs for the models in this analysis: DL_bitrate, RSRQ, RSRP, RSSI, SNR, CQI, State, NetworkMode, Ul_bitrate, NRxRSRP, NRxRSRQ, Longitude, Latitude

A history window of 10 seconds and a horizon window of 5 seconds were used.

Figure 6.1 shows the residuals of the baseline model vs a model trained exclusively on low throughput examples on a test set consisting of low throughput examples. The objective of this plot is to show the difference in how the two models deal with outliers. The baseline model heavily overestimated the download throughput horizon in outlier situations. For applications such as video streaming the baseline model would cause the user to experience buffering, as the application expects considerably more throughput than what is available to it. Figure 6.2 shows the distribution of residuals between the 5th and 95th percentiles. The baseline model overestimates more frequently as shown by the larger bias. It is also less accurate in predicting throughput in these low throughput scenarios compared to low only model as seen by the larger spread observed in the boxplots at each horizon time step.

We then considered the absolute percentage error of the predictions. The results are shown in 6.3 and 6.4. The low only model shows a considerable performance improvement over the baseline model in both the handling of outliers, as well as in the general case for 90% of the data. The table 6.2 provides the summary statistics for this analysis.

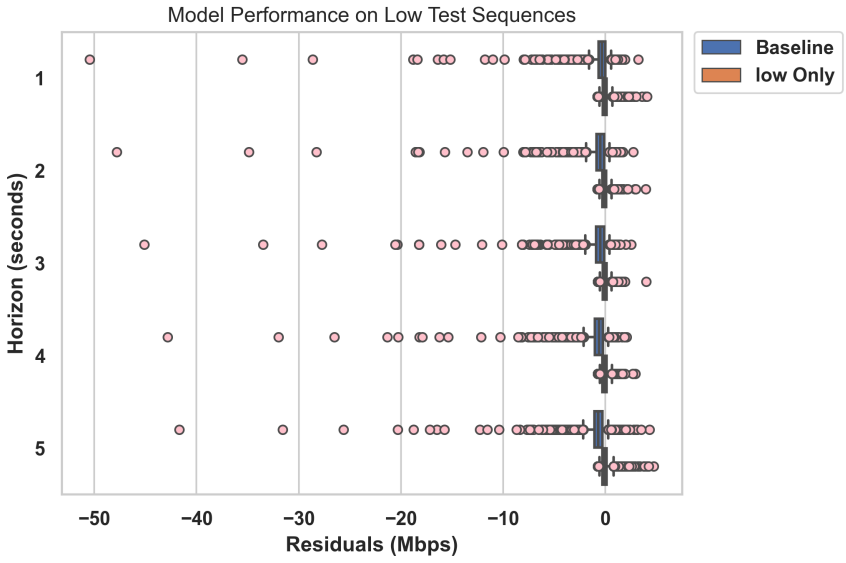

Figure 6.1: Used to show the effect of outliers

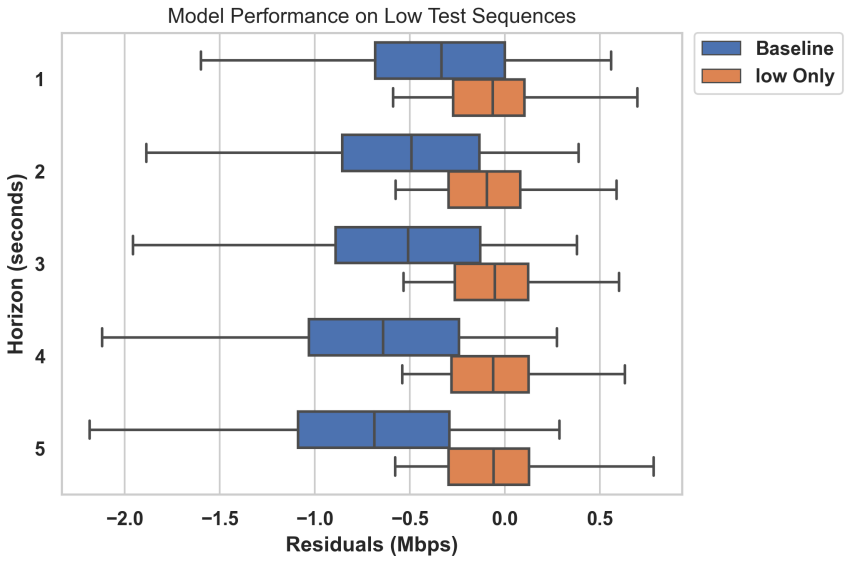

Figure 6.2: Whiskers depict the 5th and 95th percentiles

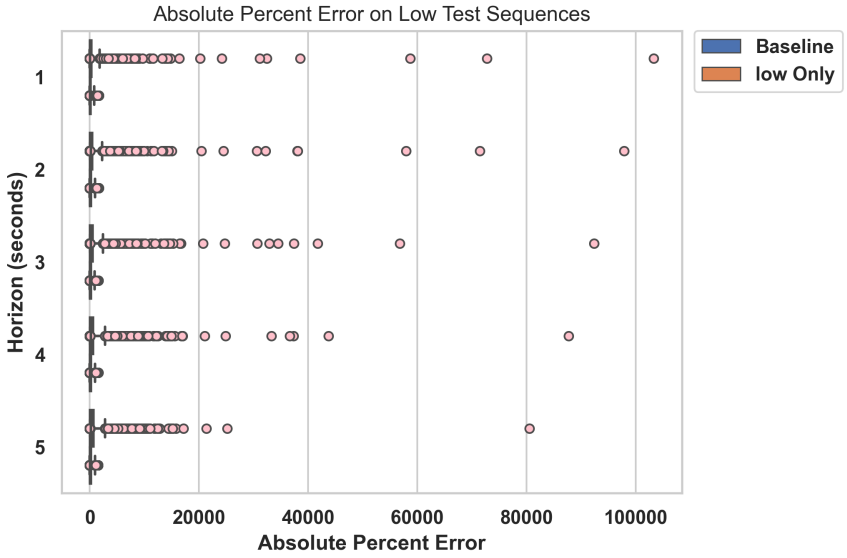

Figure 6.3: Used to show the effect of outliers

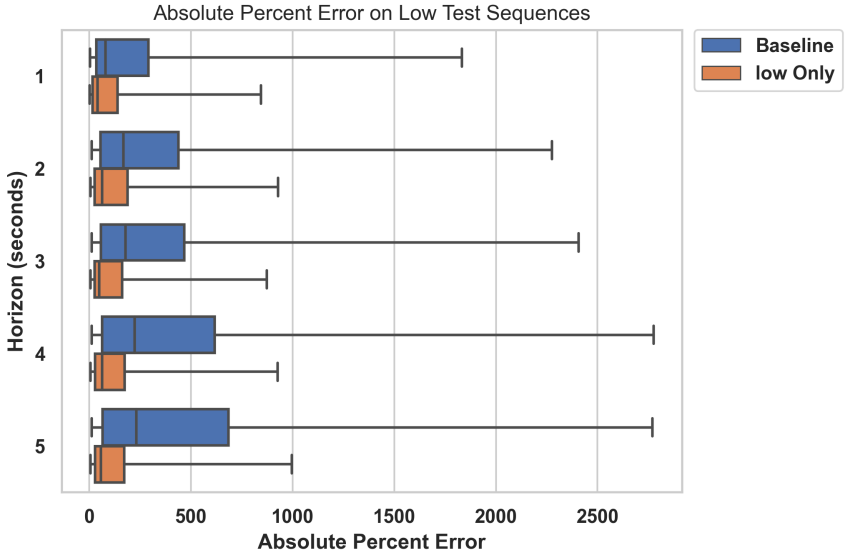

Figure 6.4: Whiskers depict the 5th and 95th percentiles

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	-0.672	1.425	674.087	2.482	0.777
low Only	-0.042	0.396	185.417	0.159	0.285

We expected similar results in the medium throughput scenario as class membership for medium is also relatively restrictive as shown in 4.8. Observations were in line with the expectation, figures 6.5, 6.6, 6.7, 6.8 align with what was confirmed in the low throughput case. The difference between the baseline and medium only model was lessened compared to low scenario due to the less strict restrictions for a throughput example to be classified as medium, as opposed to low. The table 6.2 provides summary statistics for the medium case. Again we observed the decreased bias, -0.03 vs -0.436 Mbps and improved relative performance with a decrease of 65.384 in MAPE compared to the baseline model.

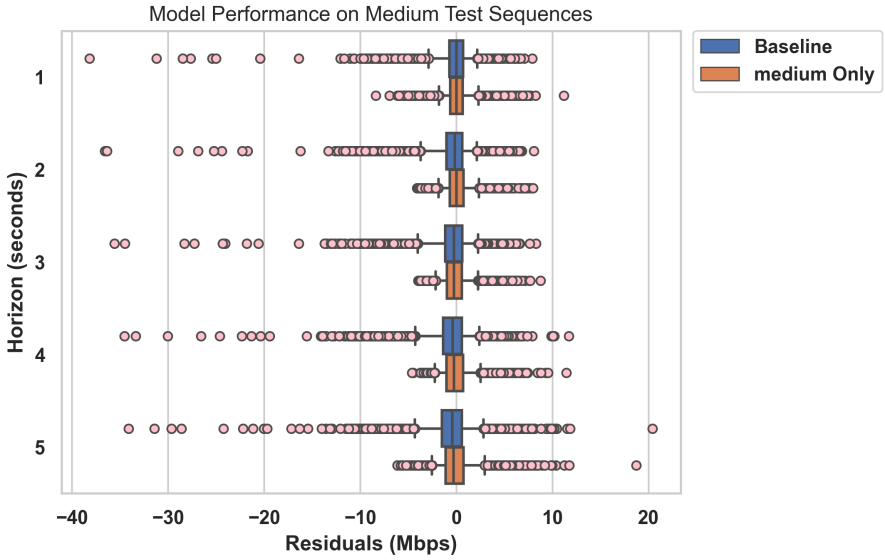

Figure 6.5: Used to show the effect of outliers

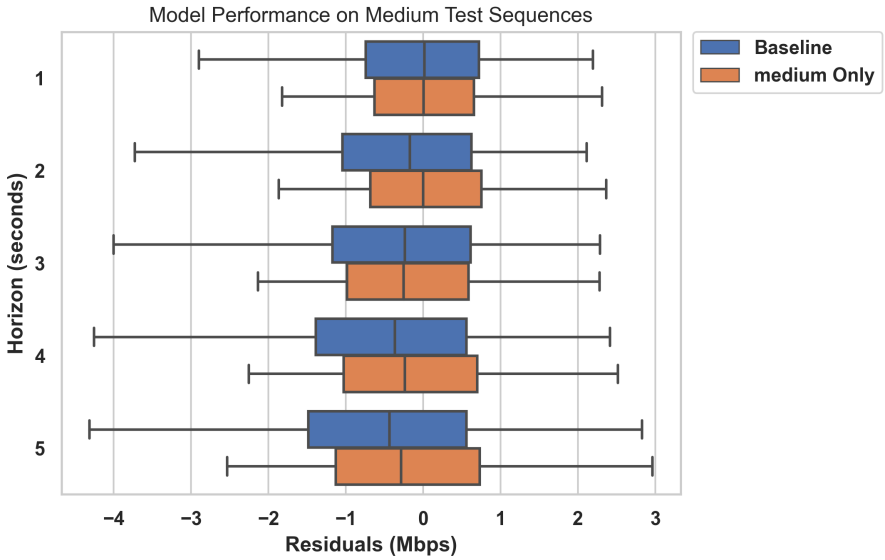

Figure 6.6: Whiskers depict the 5th and 95th percentiles

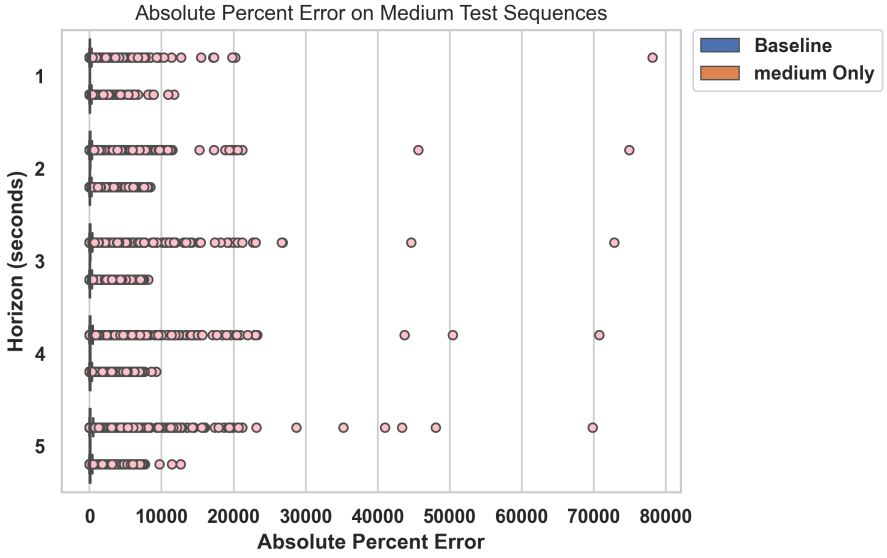

Figure 6.7: Used to show the effect of outliers

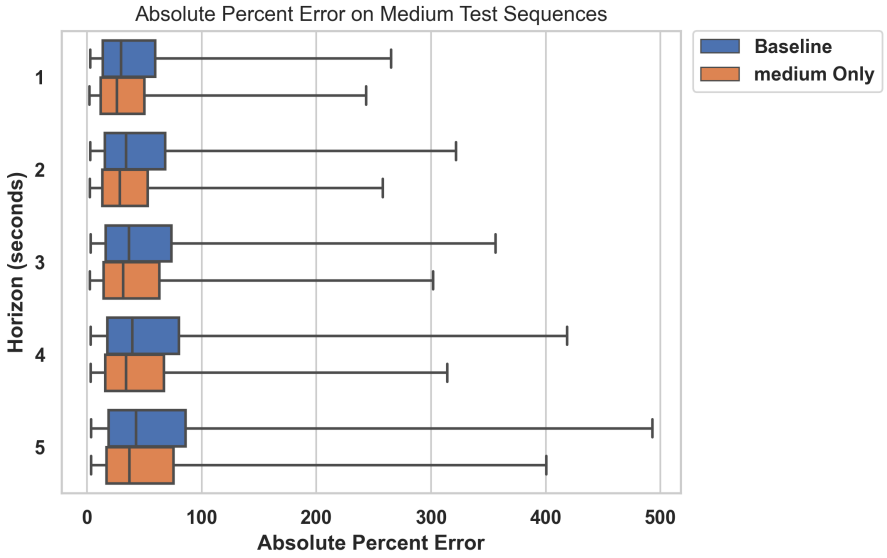

Figure 6.8: Whiskers depict the 5th and 95th percentiles

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	-0.436	2.171	198.599	4.903	1.386
medium Only	-0.03	1.44	133.215	2.074	1.061

Finally we look at the model trained exclusively on high throughput examples vs the baseline. This comparison is the most likely to show the smallest difference in model performance as membership of the high throughput class is the least restrictive and also most numerous in the dataset. Indeed we observed the two models performing roughly on par with only a difference of roughly 7 in MAPE. The boxplot 6.11 shows near identical performance on all horizon time steps.

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	1.735	6.587	97.855	46.398	4.432
high Only	0.991	6.636	104.708	45.015	4.281

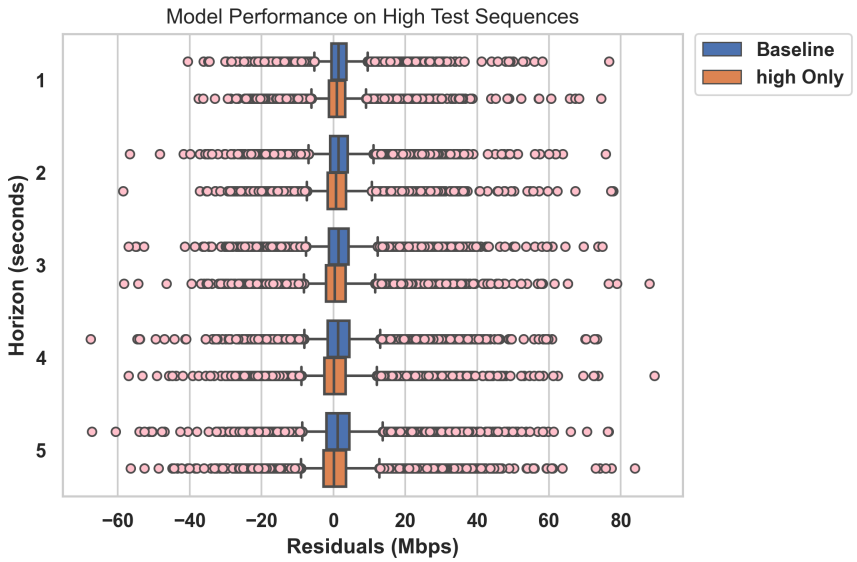

Figure 6.9: Used to show the effect of outliers

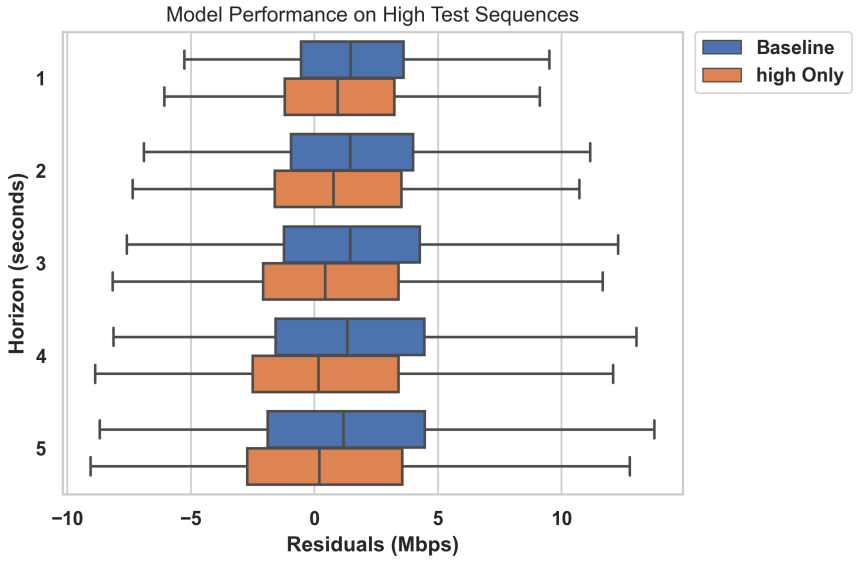

Figure 6.10: Whiskers depict the 5th and 95th percentiles

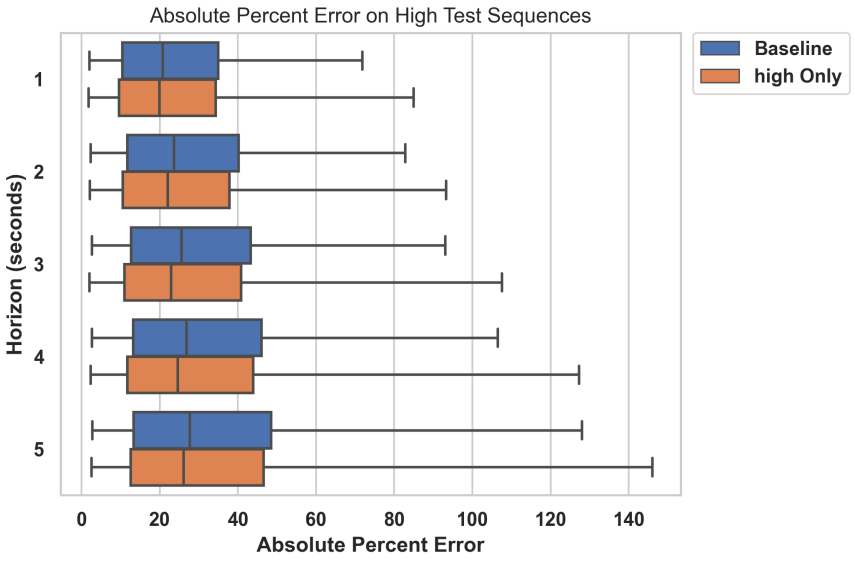

Figure 6.11: Whiskers depict the 5th and 95th percentiles

6.3 Multistage Vs Baseline

Haven proven the rational behind the potential for multi-stage models in throughput prediction we then compared the 2 multi-stage models: multi-stage one & multi-stage all to the baseline (single stage) model. The models used in this analysis are those described in 6.2. As such the 2 multistage models are considerable larger (approximately 5.9Mb) than the baseline model (approximately 1.48Mb). The aim of this section is to first show that a multistage approach to throughput prediction can compete with a "single stage" approach. Comparison between multistage and a single stage deep model where size in Mb of both approaches is equalised is explored in the later section 6.4.

The residuals (excluding outliers for clarity) on the complete test set (the baseline model's test set) are shown in figure 6.12. Considering just this boxplot, one might conclude that the multistage approaches offer no meaningful advantage over a single model. However, as previously discussed, relative error measures such as MAPE are a more telling metric of actual real-world performance differences. Looking at figure 6.13 we observe a notable decrease in the 95th percentile in absolute percentage error for the multstage models compared to the baseline model. The multistage one model in particular shows improved performance vs the baseline. The table 6.3 compiles the summary statistics of this analysis. Both multistage models provided a considerable decrease in MAPE compared to the baseline model. The area where the largest's impact was observed was specifically on cases of low throughput. In 6.14 we observe the considerable difference in the distribution of absolute percentage error values of the predictions across all horizon steps.

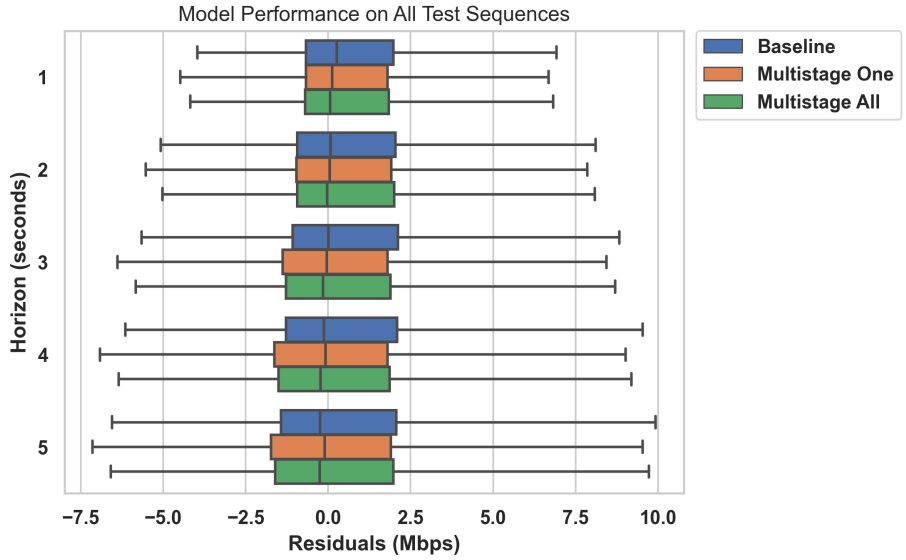

Figure 6.12: Whiskers depict the 5th and 95th percentiles

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	0.7	5.145	221.605	26.966	2.935
Multistage One	0.505	5.212	184.644	27.421	2.942
Multistage All	0.583	5.18	196.031	27.169	2.915

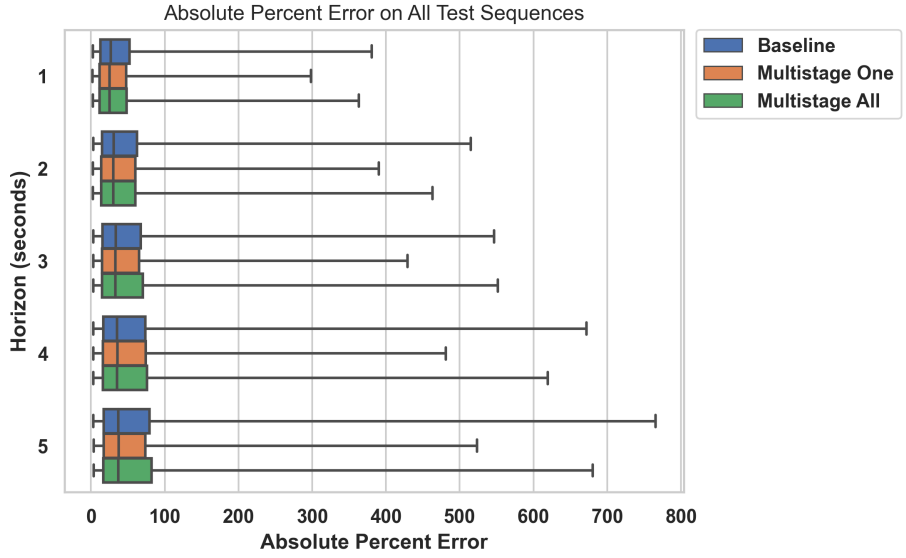

Figure 6.13: Whiskers depict the 5th and 95th percentiles

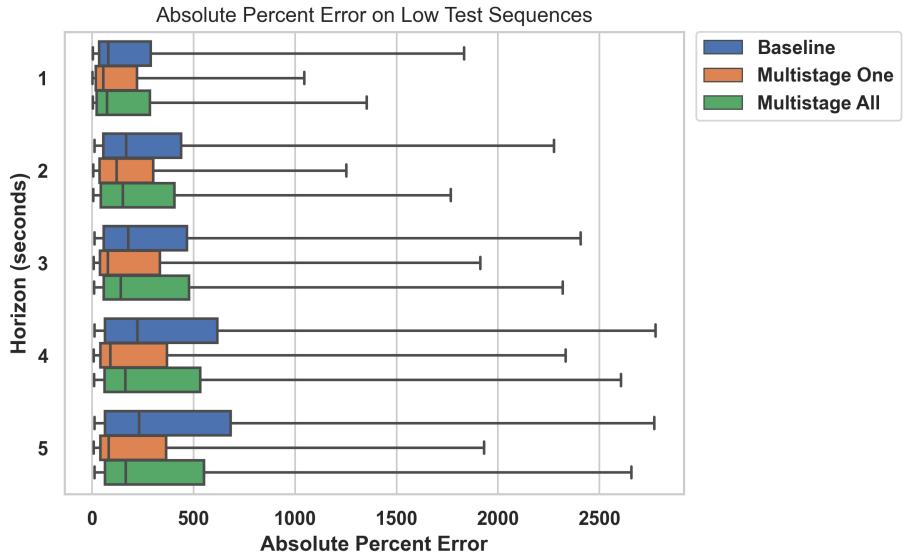

Figure 6.14: Whiskers depict the 5th and 95th percentiles

The accuracy of the classifier will have a large impact on the performance of both multistage models. Figure 6.15 show the confusion matrix for the classifier used in both models.Reducing the misclassification rate of low throughput examples as medium is the most likely area to consider for improvement. Many attempts were made to improve the accuracy of the classifier. Training with the classifier with either up-sampled data or the use of class weights showed no discernible difference from run to run variance. Down-sampling was initially tested, however it lead to worse performance on average compared to either weighted classes or up-sampling. Future multistage approaches in this field should explore the use of alternative model types for a classifier to see if any offer a noticeable improvement.

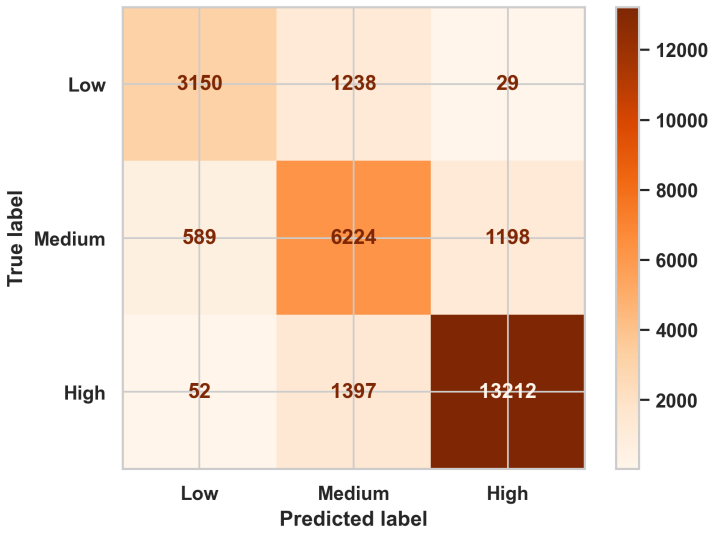

Figure 6.15: Accuracy 0.83

6.4 Equalised for Model Size

The multistaqe approach has proven to be effect in reducing the absolute percentage error of throughput predictions. The final consideration for this analysis is that of size contraints. In order for a multistage approach to be viable for consideration, it needs to be comparable in size to a single stage approach. A 5Mb model vs a 1.5Mb model is an unfair comparison in the mobile setting as mobile devices have limited computational power, battery life and memory space. Larger models use more of the available computational resources, take up more space in memory and use more power in general. In this section we explore limiting the total size of the models and compare them against a similarly sized single stage baseline model. Intuitively this means that the individual Lstm models used to construct the "multistage one" and "multistage all" will have less parameters than the model used by the baseline.

Finding the optimal solution for both the multistage models and the baseline models for a restricted hyper-parameter range would require an extensive exploration of the hyper-parameter space via model-tuning. Due to the aforementioned hardware and time limitation in 5.1 this is outside of the scope that was able to be covered in this project.

Instead educated adjustments were made to the existing best performing model identified in 5.1 in order to reach the desired size. As well as tuning the hyper-parameters of the models, a reduced feature set was also used to achieve the required size models. The features included were those identified as potential good features for predicting the throughput in 4.9. These were:

-DL_bitrate
-RSRQ
-SNR
-NRxRSRP
-State
-NetworkMode

A target size of 1.5Mb was chosen to compare model performance with similar parameter counts. The choice is arbitrary, however it was decided that the models should no be greater than 5Mb in size.

Actual sizes varied slightly around the 1.5Mb target however total trainable parameter count fell within 1% difference for all models. Figures 6.16 and 6.17 show the residuals of the 1.5Mb models on the entire test set. When comparing the raw residuals, model performance is roughly comparable at this target size, however again differenced can be found in the absolute percentage errors of the models as shown in 6.18. From the table 6.4 we see that the multistage one model provides a measurable improvement over the baseline model in MAPE with a value of 187.894 vs 199.345.

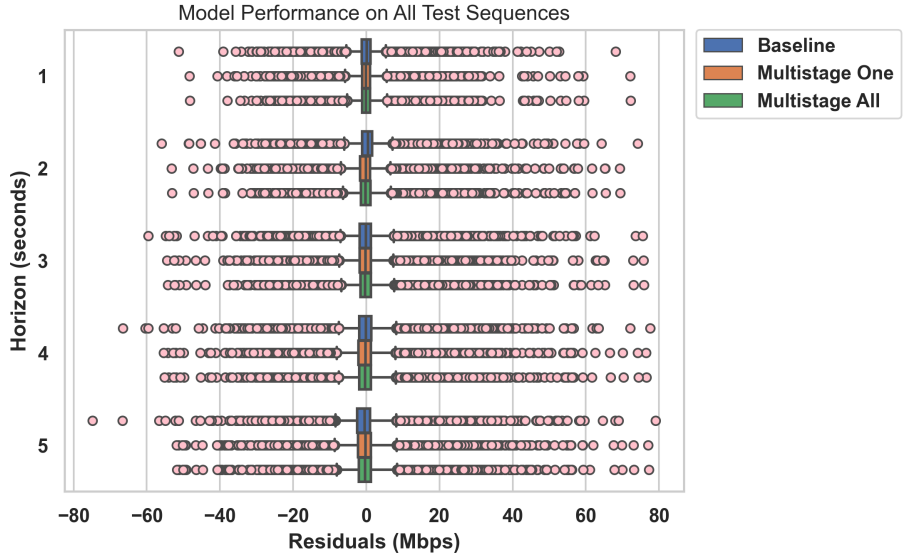

Figure 6.16: Whiskers depict the 5th and 95th percentiles

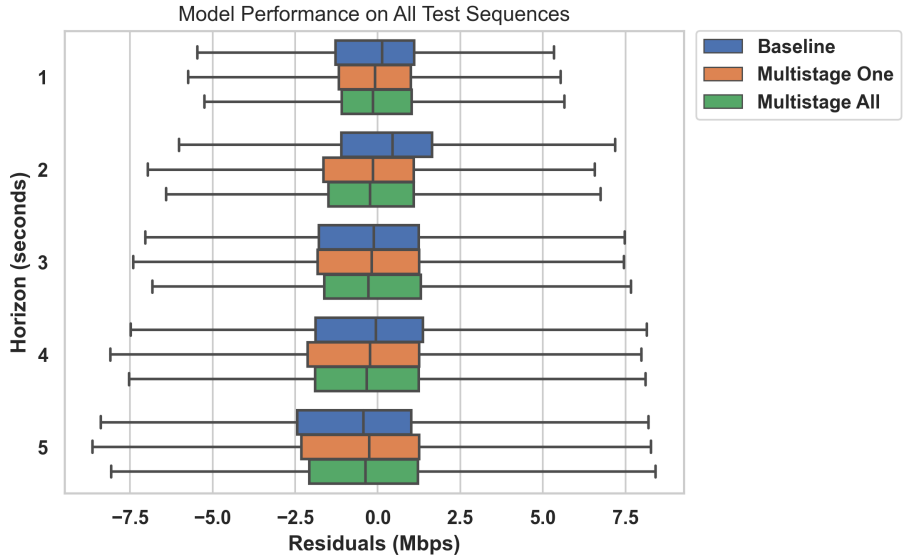

Figure 6.17: Whiskers depict the 5th and 95th percentiles

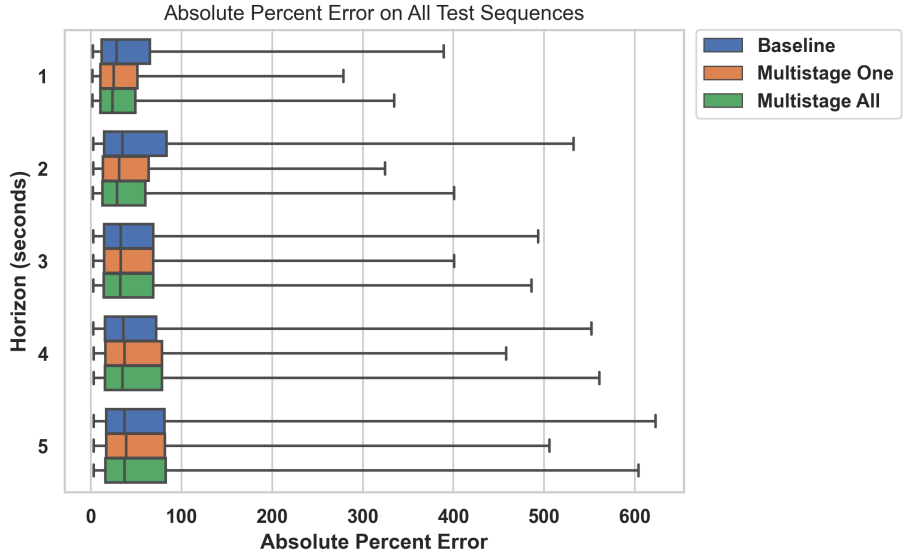

Figure 6.18: Whiskers depict the 5th and 95th percentiles

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	0.001	5.118	199.345	26.193	2.91
Multistage One	-0.176	5.193	187.894	26.999	2.953
Multistage All	-0.06	5.103	194.868	26.045	2.865

Again performance in low throughput scenarios is of particular interest to us for its correlation with improving the quality of experience for users in many applications. In figure 6.19 we observed large reduction in the absolute percentage error for the multistage one model vs the baseline model. Summary statistics for all three models exclusively tested on low throughput examples can be seen in table 6.4.

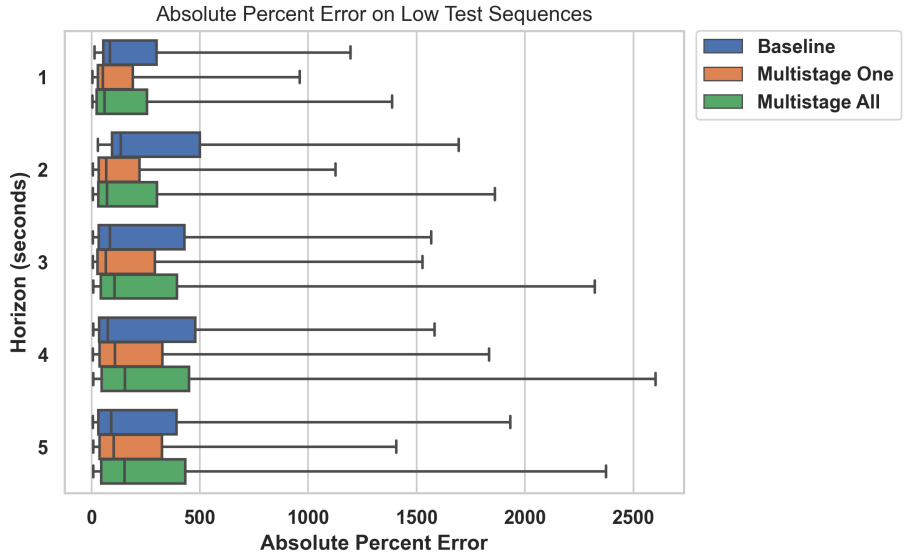

Figure 6.19: Whiskers depict the 5th and 95th percentiles

Model	Mean Resids (Mbps)	Resids std (Mbps)	MAPE	MSE (Mbps)	MAE (Mbps)
Baseline	-0.158	1.415	518.546	2.028	0.641
Multistage One	-0.404	1.391	429.679	2.099	0.596
Multistage All	-0.536	1.233	498.816	1.808	0.622

Conclusions

The usefulness of multistage deep models in throughput prediction has been shown in this paper, with multistage models providing a noticable increase in relativistic measures of model performance such as percentage based errors. The multistage one architecture proposed in this paper proved to be the best of the 3 methods considered for throughput prediction in wireless cellular networks. The variability in cellular network channels still poses a challenge for throughput prediction algorithms with much room left for improvement. Now that the multistage approach has been proven viable for consideration in both non size-constricted and size-constricted deployments, the possibility exists to explore new multistage architectures to tackle the problem of wireless throughput prediction. More work still needs to be done on building and testing these models for deployment on mobile devices, as well optimisation of the sub-models used to construct a multistage approach.

Bibliography

[1] Gustavo EAPA Batista, Maria Carolina Monard, et al. A study of k-nearest neighbour as an imputation method. *His*, 87(251-260):48, 2002.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

[3] A. Rogier T. Donders, Geert J.M.G. van der Heijden, Theo Stijnen, and Karel G.M. Moons. Review: A gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*, 59(10):1087–1091, 2006.

[4] Ericsson. Ericsson mobility report. Technical report, Ericsson, 2022.

[5] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[6] Lizhi Liao, Heng Li, Weiyi Shang, and Lei Ma. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Trans. Softw. Eng. Methodol.*, 31(3), apr 2022.

[7] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[8] Darijo Raca. *Improving video streaming experience through network measurements and analysis*. PhD thesis, 2019.

[9] Darijo Raca, Jason Quinlan, Ahmed Zahran, and Cormac Sreenan. Beyond throughput: a 4g lte dataset with channel and context metrics. pages 460–465, 06 2018.

[10] Fred L. Ramsey. Characterization of the partial autocorrelation function. *The Annals of Statistics*, 2(6):1296–1301, 1974.

[11] Nornadiah Mohd Razali, Yap Bee Wah, et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics*, 2(1):21–33, 2011.

[12] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *WIREs Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[13] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1394–1401, 2018.

[14] Statista. Statista ar and vr report. Technical report, Statista, 2022.

[15] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. Adaptive congestion control for unpredictable cellular networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, page 509–522, New York, NY, USA, 2015. Association for Computing Machinery.

[16] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys and Tutorials*, 21(3):2224–2287, 2019.