Machine Learning for Performance Prediction of Data Distribution Service (DDS)

Kaleem Peeroo, Peter Popov,

Vladimir Stankovic, Tillman Weyde

Technical Report

CITY UNIVERSITY
LONDON

School of Science and Technology

Department of Computer Science

October 2023

Contents

List of Figures

List of Tables

Abstract

Data Distribution Service (DDS) is a specification of networking middleware used in real-time mission-critical systems such as autonomous vehicles, energy management systems, and air traffic control. It follows the publish-subscribe communication pattern and offers a set of Quality of Service (QoS) parameters, allowing the users to align the data communication to the needs of the application.

Configuring DDS to achieve the required performance is a daunting task, given the large space of QoS parameters. Evaluation of performance levels by running experimental performance tests with a real DDS system for different QoS configurations can be complex and require substantial time and resources.

This report provides a detailed overview on our recent work on the use of machine learning (ML) models to predict the performance of DDS under different configurations. This is done by measuring the performance of some configurations and using the observations to train an ML model. The trained model can then be used to predict the performance of DDS under other system configurations. Since the prediction is computationally inexpensive, we can predict the performance of many different configurations to find a suitable one for given requirements. To our knowledge, we are the first to have applied this approach to DDS performance evaluation.

We used linear regression and random forests as ML methods. We selected four performance metrics, and for each of them, we trained a random forests model and

tuned its hyperparameters. We tested the final models on unseen system configurations. The random forests models show good predictive accuracy and outperform the predictions obtained with the linear regression ones. Five of the eleven random forests-based models have a coefficient of determination greater than 0.9 for unseen system configurations.

We conclude that the proposed ML models offer a way of evaluating the likely performance of a range of configurations much more cheaply than relying on experimentation and is thus a useful tool to guide system design.

Chapter 1

Introduction

1.1 Data Distribution Service

The Data Distribution Service (DDS) serves as middleware in systems that require real-time, mission-critical operations, such as in energy management and air traffic control. This system operates on a publish-subscribe model, in which the communicating entities, encompassing both publishers and subscribers, are collectively known as participants. In this model, publishers distribute data via DataWriters, each responsible for generating samples pertaining to a singular Topic. These Topics act as distinct categories of data. A publisher may employ several DataWriters to disseminate diverse samples across various topics. On the other end, subscribers utilise DataReaders to receive these samples, with each DataReader dedicated to a specific topic. To encompass a broader range of topics, subscribers may deploy multiple DataReaders, enabling them to access a variety of data samples from different topics.

1.1.1 Quality of Service

DDS employs Quality of Service (QoS) policies to delineate the end-to-end communication, many of which are amendable in real-time. These QoS settings are configured individually for each DataWriter and DataReader, incorporating Request-Offered policies. This implies that a DataWriter proposes a certain level of service, whilst a DataReader solicits a service level. Communication is feasible when the service level offered by the DataWriter is equivalent to or surpasses that requested by the DataReader, ensuring policy compatibility.

Two notable QoS parameters are Reliability and Durability. Reliability governs the acknowledgment status of a received message, offering two choices: reliable or best effort. Under a reliable setting, it is imperative for subscribers to confirm message receipt. In such scenarios, if the publisher opts for reliability, it awaits acknowledgment for a predetermined duration before resending the message. Conversely, a best-effort approach signifies message transmission without assurance of receipt.

Durability pertains to the publishers' capacity to dispatch previously sent messages to late-joining subscribers. It encompasses four tiers: firstly, not retaining or dispatching past samples; secondly, preserving and transmitting samples provided the DataWriter remains active; thirdly, utilising a memory-based service for saving and dispatching; and lastly, employing a disk-based service for the same purpose.

The DDS specification is built upon the Real-Time Publish-Subscribe (RTPS) specification and mentioned within the RTPS specification are several QoS policies that are most likely to impact the performance of the communication:

1. Durability

2. Presentation

3. Liveliness

4. Time Based Filter

5. Reliability

6. Destination Order

7. Writer Data Lifecycle

In addition to QoS parameters, other non-QoS factors can also potentially influence performance. These include the length of the data in transmitted messages, the number of publishers and subscribers, as well as the employment of multicast in communication.

Multicast is a network communication methodology, where a single sender addresses multiple specific recipients. This approach enhances bandwidth efficiency by circumventing the need to replicate packets. Rather, a solitary packet dispatched by the sender is duplicated and distributed by the routers and other network components involved in the transmission process.

Collectively, DDS encompasses 22 QoS parameters and 4 non-QoS parameters, each playing a pivotal role in the system's overall functionality.

1.2 Performance Evaluation and Prediction

Performance evaluation, though a non-functional requirement, holds significant importance in system assessment. The advantages of conducting performance evaluations are manifold. Firstly, it aids in determining the scalability of a system, reflecting its efficiency and capability to handle increased loads. Secondly, performance assessment can illuminate areas suitable for optimisation, particularly in reducing operational costs. This includes refining deployment strategies to minimise hardware

requirements and cut down on data transfer overheads. Furthermore, evaluating a system's performance allows for a comparative analysis with alternative solutions, helping to pinpoint the most effective system for specific workloads.

The predominant method for evaluating system performance is through experimental means. This involves conducting real-time performance tests on an operational system, a process that demands both time and computational resources. Ideally, these tests should cover a variety of workloads and conditions, potentially leading to a considerable number of tests and, consequently, significant resource consumption. It is also worth noting that longer-duration performance tests can yield deeper insights into system behaviour, though the cumulative time investment can be substantial across numerous tests.

An alternative to experimental evaluation is the model-based approach. This method entails developing and fine-tuning statistical or machine learning models that learn from historical data patterns. Such models can then forecast the performance of systems under untested workloads. This approach is pivotal for optimising settings to achieve peak performance, allowing for the simulation of different scenarios in a fraction of the time and resource expenditure required for experimental evaluations. Moreover, model-based evaluation is instrumental in efficient resource allocation, for instance, determining the most effective settings for a given bandwidth to maximise performance.

Our research focuses on employing Machine Learning models to predict the performance of DDS systems influenced by various settings, including both QoS and non-QoS parameters.

1.3 Perftest

Perftest, developed by RTI, a DDS vendor, serves as a performance testing tool distinct from standard DDS publishers and subscribers. It involves a Perftest publisher and a Perftest subscriber, each incorporating a publisher and a subscriber within their application structure. The Perftest publisher employs its DDS publisher to send continuous sample data to the DDS subscriber within the Perftest subscriber application. Periodically, it transmits a special sample, prompting a measurement packet from the Perftest subscriber. At this juncture, the Perftest publisher marks the time. Upon receiving this request, the Perftest subscriber responds using its DDS publisher, sending the measurement packet back. The Perftest publisher, upon receipt, logs another timestamp, allowing the calculation of the round-trip time.

Perftest operates in two modes: throughput and latency. In the throughput mode, the operation is as previously described. Conversely, in latency mode, each sent sample is a measurement packet, enabling the assessment of latency without additional load, as opposed to the throughput test where latency is gauged under a specific load.

Originally tailored for testing the performance of RTI's Connext DDS implementation, Perftest's versatility allows for its application in evaluating other DDS implementations. For this study, Perftest was utilised to gather data on the RTI Connext application, which subsequently aided in training a machine learning model.

1.4 Machine Learning

Machine learning, a branch of artificial intelligence, concentrates on crafting algorithms and models that empower computers to learn from data and make informed decisions without specific programming directives. This concept revolves around

the notion that computers can autonomously enhance their task performance by gleaning experience from data. In our research, we have explored the use of random forests to forecast the performance of DDS. Additionally, linear regression has been employed as a baseline model for comparative analysis.

1.4.1 Linear Regression

Linear regression stands as a prevalent statistical technique for modelling the correlation between a dependent variable (the target) and one or multiple independent variables (predictors or features). In our context, the target is the performance, while the features encompass the values of both QoS and non-QoS parameters. This method presupposes a linear interrelation between the variables, implying that variations in the features consistently influence the target. Linear regression calculates the coefficients for each independent variable, signifying the intensity and tendency of their relationship with the dependent variable. Its high interpretability stems from the way these coefficients elucidate the effect of changes in the independent variables on the target.

1.4.2 Random Forests

Random forests, an ensemble learning technique, enhances prediction accuracy by amalgamating the outputs of multiple individual models. Constructed using Decision Trees, random forests are trained with a randomly chosen subset of the training data. Not every feature is taken into account for decision-making at each node split within a tree. For prediction purposes, each tree in the forest contributes its own forecast, with the final prediction typically being an average of these individual predictions. The ensemble nature of random forests, coupled with the randomness in both sampling and feature selection, significantly mitigates the risk of overfitting,

thereby augmenting their robustness against noisy data. This attribute renders random forests particularly effective for predicting the performance of distributed systems, including DDS. Additionally, like linear regression, random forests can ascertain feature importance, highlighting which features exert the most influence on the model's predictions.

1.4.3 Model Evaluation

Numerous metrics are available for assessing the performance of a model. In our study, we have chosen to concentrate specifically on two of these metrics: Root Mean Square Error (RMSE) and the coefficient of determination, denoted as R^2.

RMSE

The Root Mean Squared Error (RMSE) measures the average size of the errors or residuals between predicted and actual observed values. The formula for RMSE is given by:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_{\text{predicted},i} - y_{\text{actual},i})^2}$$

This calculation involves taking the square root of the average of the squared differences between the predicted and actual values, as demonstrated in the formula above. RMSE is expressed in the same units as the dependent variable, enhancing its interpretability in relation to the problem at hand. A lower RMSE value signifies a more accurate model, as it indicates that the predictions are closer to the actual observations. However, RMSE is particularly sensitive to outliers, as the squaring of errors disproportionately emphasises larger discrepancies, which can markedly affect the RMSE value.

Coefficient of Determination

The coefficient of determination, denoted as R^2, is a statistical measure used in regression analysis to assess the fit quality of a model. It usually ranges between 0 and 1, indicating the fraction of the variance in the dependent variable that is accounted for by the independent variables. The R^2 formula is as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_{\text{actual},i} - y_{\text{predicted},i})^2}{\sum_{i=1}^{n}(y_{\text{actual},i} - \bar{y})^2}$$

This metric can be interpreted as the ratio of the explained variance (the sum of squares of the fitted values) to the total variance (the sum of squares of the actual values). A higher R^2 value signifies that a greater portion of the variance in the dependent variable is explained by the model, implying a better fit. However, it is important to note that R^2 does not convey the direction or clinical significance of the relationship. Even with a high R^2, scrutinizing the individual coefficients and the context of the data remains crucial, which is why RMSE is also considered in our analysis.

Chapter 2

Related Work

This study pioneers the application of Machine Learning (ML) models for predicting Data Distribution Service (DDS) performance. A thorough literature review found no direct precedents in this specific area. However, expanding the scope of our search revealed relevant studies under broader terms in adjacent fields.

The first pertinent field is *Performance Prediction of Distributed Systems*. Here, studies like (Bouloukakis et al. 2018) have utilised Stochastic Petri Nets (SPNs) for assessing the performance of publish-subscribe systems, while (Sachs et al. 2013) applied them in the context of message-oriented middleware.

The second relevant area is *Performance Prediction of Configurable Systems using Machine Learning*. Numerous studies have focused on predicting the performance of systems with adjustable parameters, where different configurations impact overall performance. These investigations predominantly leverage ML techniques, but typically they forecast a *single* statistical figure representing a chosen performance metric, such as the mean value.

In works like (Bao et al. 2018), a random forests-based comparative model is employed to identify the optimal system configuration for maximising throughput.

(Guo et al. 2018) and (Siegmund et al. 2015) integrate ML methods and sampling strategies to predict performance in configurable systems. These strategies select training data based on configuration positions within a multi-dimensional space defined by the configuration options' values. (Ha et al. 2019*a*) employs a deep feed-forward neural network to predict performance across different configurations, representing the performance with a singular value. An innovative approach is seen in (Shu et al. 2020), where adversarial learning via Generative Adversarial Networks (GANs) is applied to predict and optimise performance with fewer samples than traditional techniques. Moreover, (Zhang et al. 2015) and (Ha et al. 2019*b*) explore performance prediction in systems with solely Boolean configuration options.

Our work draws substantial inspiration from (Rausch & Sanders 2020), which mirrors our process conceptually. The ML methodologies we adopted are based on insights from (Grebhahn et al. 2019), a comprehensive survey reviewing ML techniques and sampling strategies for performance prediction in configurable systems. This survey advocates using three primary ML methods: random forests, multiple linear regression, and classification and regression trees.

Chapter 3

Method

This work can be broken down into 4 main parts:

1. Experimental Setup (Testbed)

2. Dataset Creation

3. Configuration Space Exploration

4. Model Training and Evaluation

Section 3.1 looks at the machines involved in the performance tests, the tools, the network setup, and how the data is collected. Section 3.2 gives an overview of how the raw data collected from the tests was processed into a dataset for the exploration of the configuration space to take place. Section 3.3 takes this dataset and explores the configuration space to split up the data in preparation for the model training. Finally, section 3.4 looks at how the ML training took place and the various metrics used for evaluating the ML models.

3.1 Experimental Setup (Testbed)

Perftest serves as the foundational tool for executing DDS performance tests to collect data. Yet, it exhibits limitations in terms of automation. Primarily suited for a limited number of tests, Perftest requires setting up publisher and subscriber applications through command line instructions, with options provided as command line parameters. To overcome these constraints, we have developed a supplementary tool, named AutoPerf. This tool enhances Perftest by facilitating the execution of numerous tests across multiple machines, offering flexibility in test duration and enabling pre-set variations in parameter options.

3.1.1 AutoPerf

To initiate test campaigns using AutoPerf, it is essential first to populate the configuration file with the requisite settings prior to executing the tool. The configuration file serves as the hub for defining each campaign. It allows for the definition of multiple campaigns, which can be executed sequentially, thus providing considerable flexibility. The concept revolves around pre-defining all the tests intended for execution before starting the tool, enabling a set-and-forget approach. For every campaign, the user is required to specify the following settings:

- `use_random_combination_generation`
- `random_combination_count`
- `custom_tests_file`
- `settings`
- `machines`

The two important options to focus on are `settings` and `machines`. `settings` contain the QoS and non-QoS parameters and the values they should take. These include:

- Duration (seconds)
- Data Length (bytes)
- Publisher Count
- Subscriber Count
- Reliability Usage
- Multicast Usage
- Durability Level
- Latency Count Value

Below is an example of the settings configuration and the values it could take:

```
"settings": {
    "duration_s": [600],
    "datalen_bytes": [1, 100, 1000, 10000],
    "pub_count": [1, 5, 10],
    "sub_count": [1, 5, 10],
    "reliability": [true, false],
    "use_multicast": [true, false],
    "durability": [0, 1, 2, 3],
    "latency_count": [1000]
}
```

The `machine` configuration in AutoPerf delineates details about the machines utilised in the test, functioning as slaves in the setup. The machine running AutoPerf interacts with these configured machines in a slave-controller relationship, with the latter being the slaves. This configuration encompasses various details such as machine names, host IP addresses, locations of SSH keys, usernames for authentication, home directory paths, Perftest file paths, and participant allocations. The participant allocation can be specified as "pub", "sub", or "both". A setting of "pub" restricts the machine to hosting only publishers, and similarly, "sub" for subscribers. Conversely, a "both" value allows a machine to accommodate both publishers and subscribers. This flexibility is crucial, particularly for scenarios where an equal distribution of publishers and subscribers across machines is desired.

AutoPerf operates in two distinct modes: Preset Combination Generation (PCG) and Random Combination Generation (RCG). In this research, both modes were employed to facilitate comprehensive data collection.

Preset Combination Generation (PCG)

PCG works by taking all of the settings and their values and applying the Cartesian product to produce all possible combinations of values for all settings. For example, in the scenario with the following settings:

```
"settings": {
    "duration_s": [600],
    "datalen_bytes": [100],
    "pub_count": [1, 25],
    "sub_count": [1, 25],
    "reliability": [true],
```

```
    "use_multicast": [true],
    "durability": [0],
    "latency_count": [100]
}
```

When applying the Cartesian product, the produced combinations would be:

- `600SEC_100B_1P_1S_REL_MC_0DUR_100LC`
- `600SEC_100B_1P_25S_REL_MC_0DUR_100LC`
- `600SEC_100B_25P_1S_REL_MC_0DUR_100LC`
- `600SEC_100B_25P_25S_REL_MC_0DUR_100LC`

Notice how there are four combinations as a result of two `pub_count` options per `sub_count` option. Each combination is treated as it's own test and is run for the prescribed duration (in this case, 600 seconds).

Random Combination Generation (RCG)

RCG works by defining a minimum and maximum value for each setting that is numeric. For non-numeric settings, you define all possible options. The tool will then pick a random value between the minimum and maximum values and treat that as a random combination and its test to run. For example, with the following settings:

```
"settings": {
    "duration_s": [600],
    "datalen_bytes": [100, 128_000],
    "pub_count": [1, 25],
```

```
    "sub_count": [1, 25],
    "reliability": [true, false],
    "use_multicast": [true, false],
    "durability": [0, 3],
    "latency_count": [100]
}
```

A random value would be defined for `datalen_bytes`, `pub_count`, `sub_count`, and `durability`. In the case of `reliability` and `use_multicast`, a value of "true" or "false" is randomly chosen. A valid combination as a result of random selection could be `600SEC_34353B_7P_3S_BE_MC_2DUR_100LC`. This process is repeated for a number of tests defined by the value of the `random_combination_count`. The case for repeating the same random values for all parameters is incredibly small. However, we have still accounted for this by keeping track of all generated combinations and regenerating a new combination where the exact values of the combination have already been seen.

Perftest only lets the user run an executable for each Perftest publisher or Perftest subscriber application. The configuration parameters passed into the executable command must match for the publisher and subscriber applications to communicate. Any configuration mismatch will result in an error, and the test won't run. Autoperf has been created to deal with this by checking that the parameter values match between applications before deploying them. Once the parameter values have been confirmed to be valid, autoperf generates bash files consisting of multiple executable commands glued together. Each slave machine has its own defined bash file, and the publisher and subscriber applications are allocated as evenly as possible onto each machine. Therefore, each bash file only contains the publisher and subscriber

applications that have been allocated to it.

When running a test, autoperf starts multiple different processes to execute the scripts remotely on each machine and also records the statuses of the machines during execution to keep track of which machines have failed during run time. In the case of a failure, the test is recorded as failed, and the tool attempts to rerun the failed test up to 3 times before moving on to the next combination. At the end of each test, autoperf moves all data related to that test, including the status of the machines, to the campaign folder. Once the campaign of multiple tests is finished, autoperf compresses the results to allow for easy transmission.

3.1.2 Testbed

All of the data collected for this work was carried out by 3 Raspberry Pis. As mentioned in the previous section, the use of Autoperf requires the usage of the slave-controller pattern. Therefore, 1 of the 3 Raspberry Pis was the controller while the other 2 were slaves. The controller machine was Version 4 Model B released in 2018, while both slaves were Version 3 Model B Version 1.2, released in 2015. All three machines were connected in a closed LAN environment via an Ethernet connection. The controller machine executed autoperf, allowing the slaves to communicate with themselves.

3.2 Dataset Creation

This section is split into two further sections. The first looks at how much data was collected for each operational mode of autoperf. The second section looks at how the raw data collected from the tests is processed for the modelling phase.

3.2.1 Data Acquisition

Initially, we used PCG to capture a grid from the various parameter options. Below are the values used for the setting configuration:

```
"settings": {
    "duration_s": [600],
    "datalen_bytes": [100, 1_000, 32_000, 128_000],
    "pub_count": [1, 5, 10, 25],
    "sub_count": [1, 5, 10, 25],
    "reliability": [true, false],
    "use_multicast": [true, false],
    "durability": [0, 1, 2, 3],
    "latency_count": [100]
}
```

This collectively results in a total of 1,024 different combinations. However, only 633 tests were successful - a 62% success rate. We suspect a large failure rate due to the hardware limitations. The majority of the tests with more than 10 publishers would fail. Of all PCG tests that were run successfully, the median and average values for the publisher count were 5 and 6, respectively, indicating that tests with more publishers would most likely fail. The median and average values for the subscriber count were 5 and 9 respectively indicating that the tests with more subscribers more often succeeded compared to the higher number of publishers.

The rest of the data was collected using RCG mode. A total of 952 tests were successfully run in this mode. Figures 3.2 and 3.3 show the histograms of the publisher and subscriber counts for all RCG tests. These diagrams confirm the above hypothesis that tests with higher numbers of publishers (the figure indicates

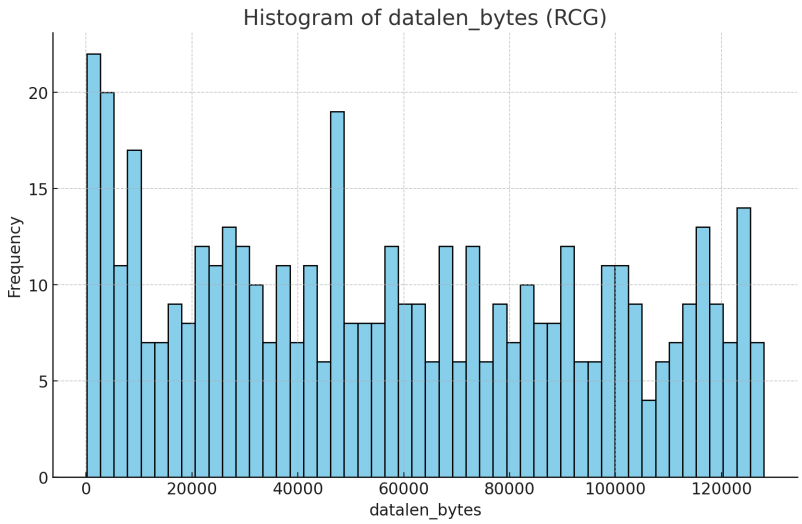

Figure 3.1: Histogram of the values of data length (bytes) for RCG data.

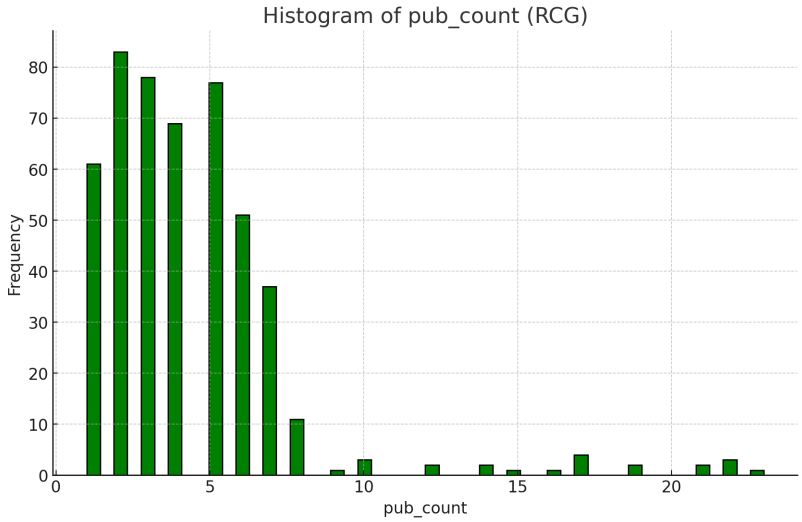

Figure 3.2: Histogram of the values of publisher count for RCG data.

any value greater than 10) are more likely to fail.

In total, 1,585 tests were successfully run (633 in PCG mode and 952 in RCG mode). Figures 3.4, 3.5, and 3.6 show the histograms for the data length (bytes), publisher count, and subscriber count. There are peaks that correspond to the values ran in PCG mode indicating that there is more data to learn on for the options specified in PCG mode.

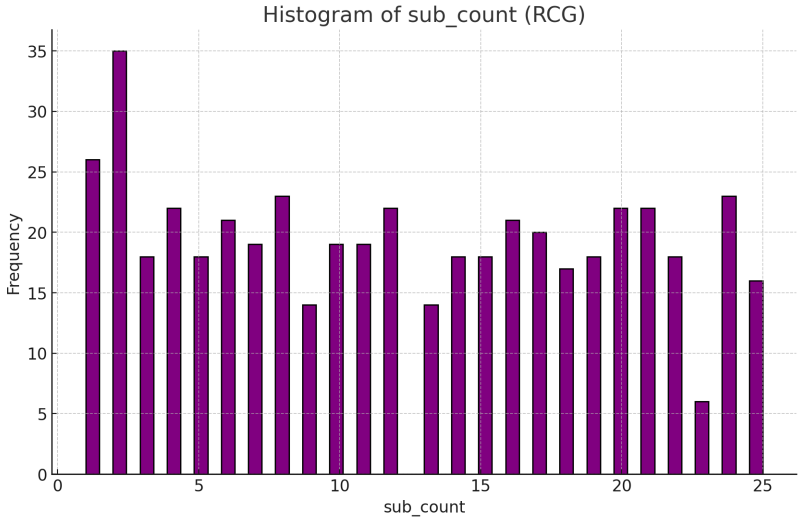

Figure 3.3: Histogram of the values of subscriber count for RCG data.

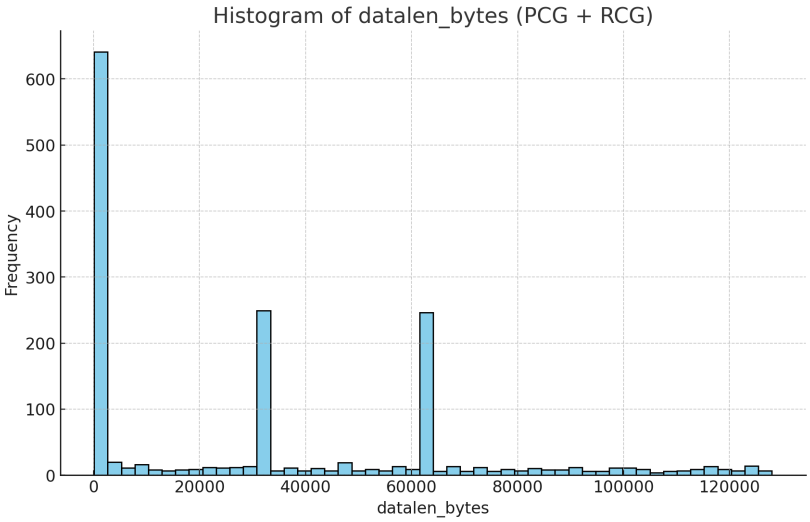

Figure 3.4: Histogram of the values of data length (bytes) for all data (PCG and RCG).

3.2.2 Data Processing

Each test involves x publishers and y subscribers, which means that in total, $y + 1$ files are generated: only the first publisher records the latency and one file for each subscriber. The files generated by subscribers record the following metrics:

- **Total Samples**: Total samples received by the subscriber.
- **Samples/s**: Sample rate. Number of samples received per second by the subscriber.

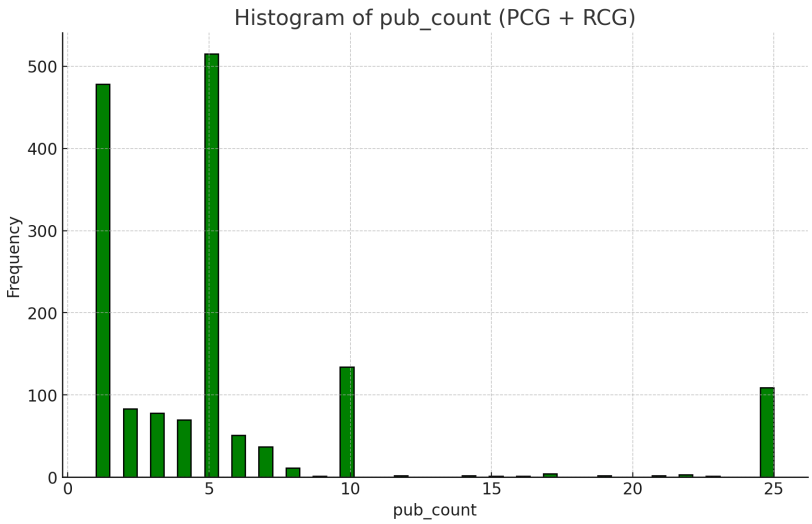

Figure 3.5: Histogram of the values of publisher count for all data (PCG and RCG).

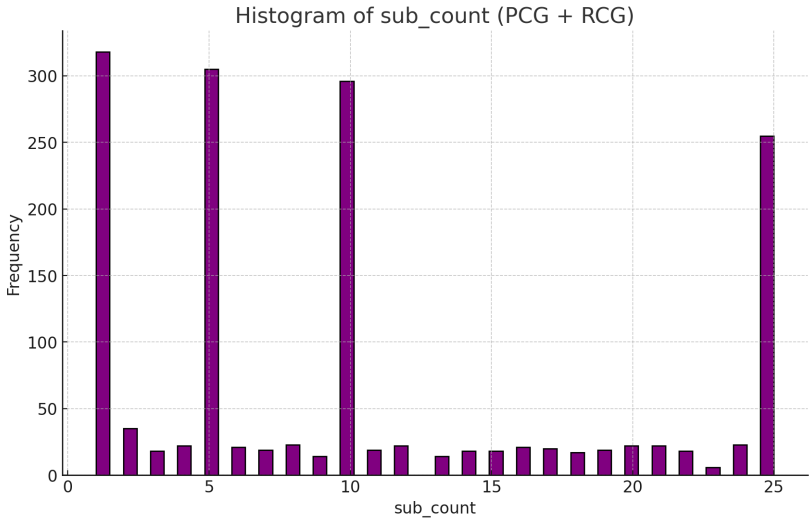

Figure 3.6: Histogram of the values of subscriber count for all data (PCG and RCG).

- **Avg Samples/s**: Moving sample rate average.
- **Mbps**: Throughput of the subscriber.
- **Avg Mbps**: Moving throughput average.
- **Lost Samples**: Number of samples lost.
- **Lost Samples (%)**: Percentages of all samples lost.

The publisher file generated only by the first publisher records the following metrics:

- **Latency (us)**: One-way latency (round trip time divided by two).
- **Ave (us)**: Moving latency average.
- **Std (us)**: Moving latency standard deviation.
- **Min (us)**: Minimum latency observed throughout entire test.
- **Max (us)**: Maximum latency observed throughout entire test.

The first step of the data processing pipeline involves extracting the important columns from the files. For the publisher, this is simply extracting the latency column. For the subscriber files, we extracted the throughput, sample rate, number of lost samples, and lost samples as a percentage. The lost samples percentage is used to calculated the percentage of samples received from the total samples count. Each of the columns from the subscriber has the ID of the subscriber prepended to the name e.g. "mbps" becomes "sub_1_mbps". All columns are then placed into a single file. This reduces all files from a single test into a single file containing the latency column from the publisher and throughput, sample rate, number of lost samples, lost samples percentage, and received samples percentage for each subscriber. Therefore, each test is reduced to a single file and each campaign is reduced to a single folder containing a CSV file per test.

To make the subscriber-related data comparable between tests, the data was summarised using aggregation and averaging over all subscribers. This means that for each metric from the subscriber e.g. throughput, two new columns would be generated, one for the averaged value and one for the total over all subscribers. Therefore, all subscriber related columns are reduced to just two per metric meaning that the final version of the data for an individual test had the following columns:

- Latency (us)

- Avg. Throughput (Mbps)
- Total Throughput (Mbps)
- Avg. Sample Rate
- Total Sample Rate
- Avg. Received Samples Count
- Total Received Samples Count
- Avg. Lost Samples Count
- Total Lost Samples Count
- Avg. Samples Received Count
- Total Samples Received Count
- Avg. Samples Received (%)
- Avg. Lost Samples (%)

At this point, the processed data was ready for transient analysis.

3.2.3 Transient Analysis

We visually inspected the data to look for transient periods at the start of the test. This was done for both the latency and throughput.

We randomly sampled 100 tests and plotted the data in a 10x10 grid. We plotted a vertical dashed line at the 10% for reference. We then identified any plots that showed visibly different performance at the start of the test (preferably within the first 10%). We counted the number of tests that showed this behaviour and noted the numbers. In the case of latency, the average number of tests out of the 100 that

showed transient behaviour at the start of the test was 25. In the case of throughput, this number was 61 out of 100. Figures 3.7 and 3.8 show examples of the 10x10 grid of data and which tests were deemed to show transient behaviour.

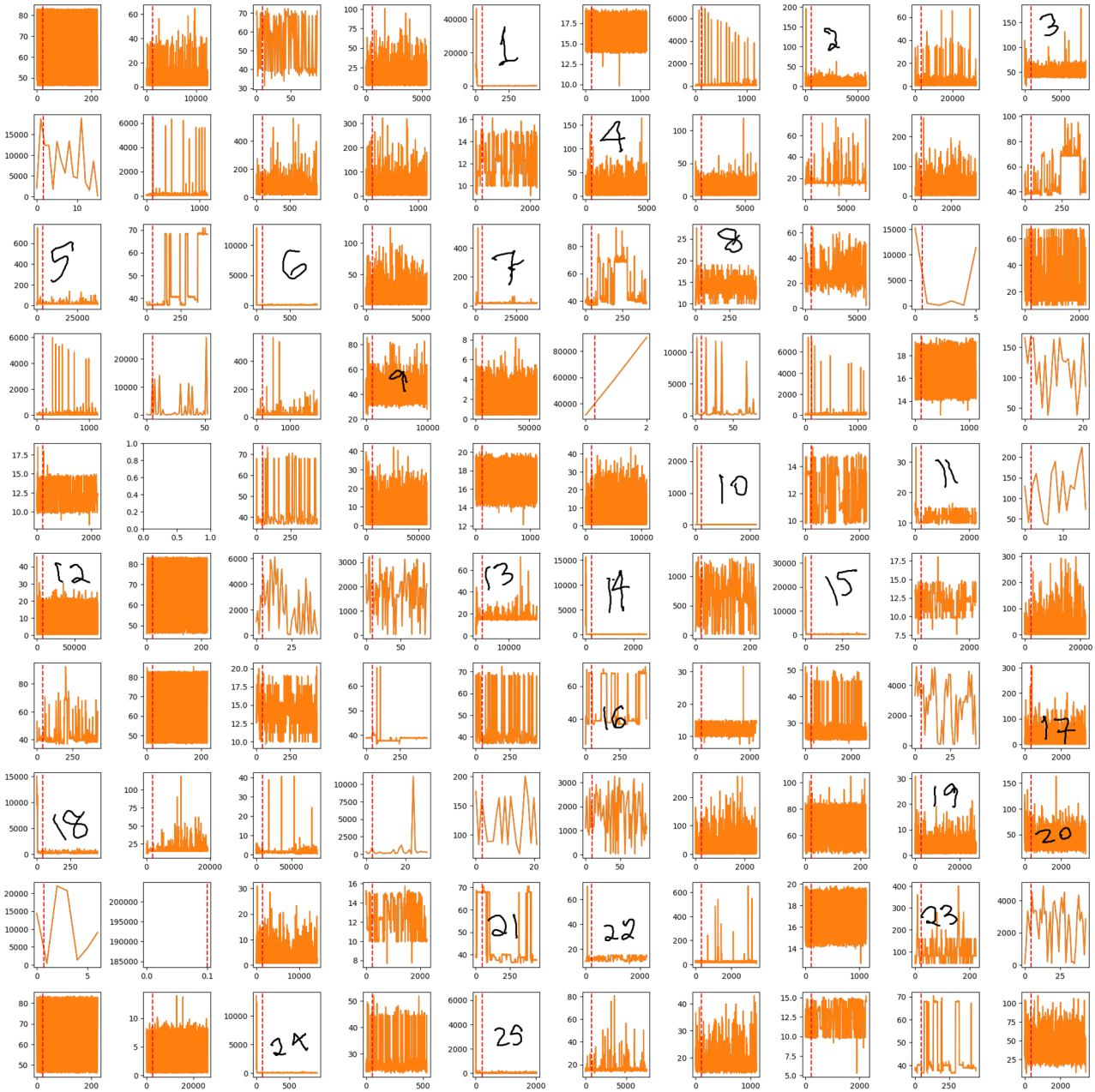

Figure 3.7: Example of visual inspection of latency data.

It is essential to remove the transient period as this often clouds the true behaviour of the system during the performance test. We concluded that many of the tests showed transient periods, especially in the case of the throughput. We, there-

fore, decided to cut out the first 20% of the data to be sure that all transient periods do not affect the overall performance behaviour (despite the transient periods only showing up within the first 10%).

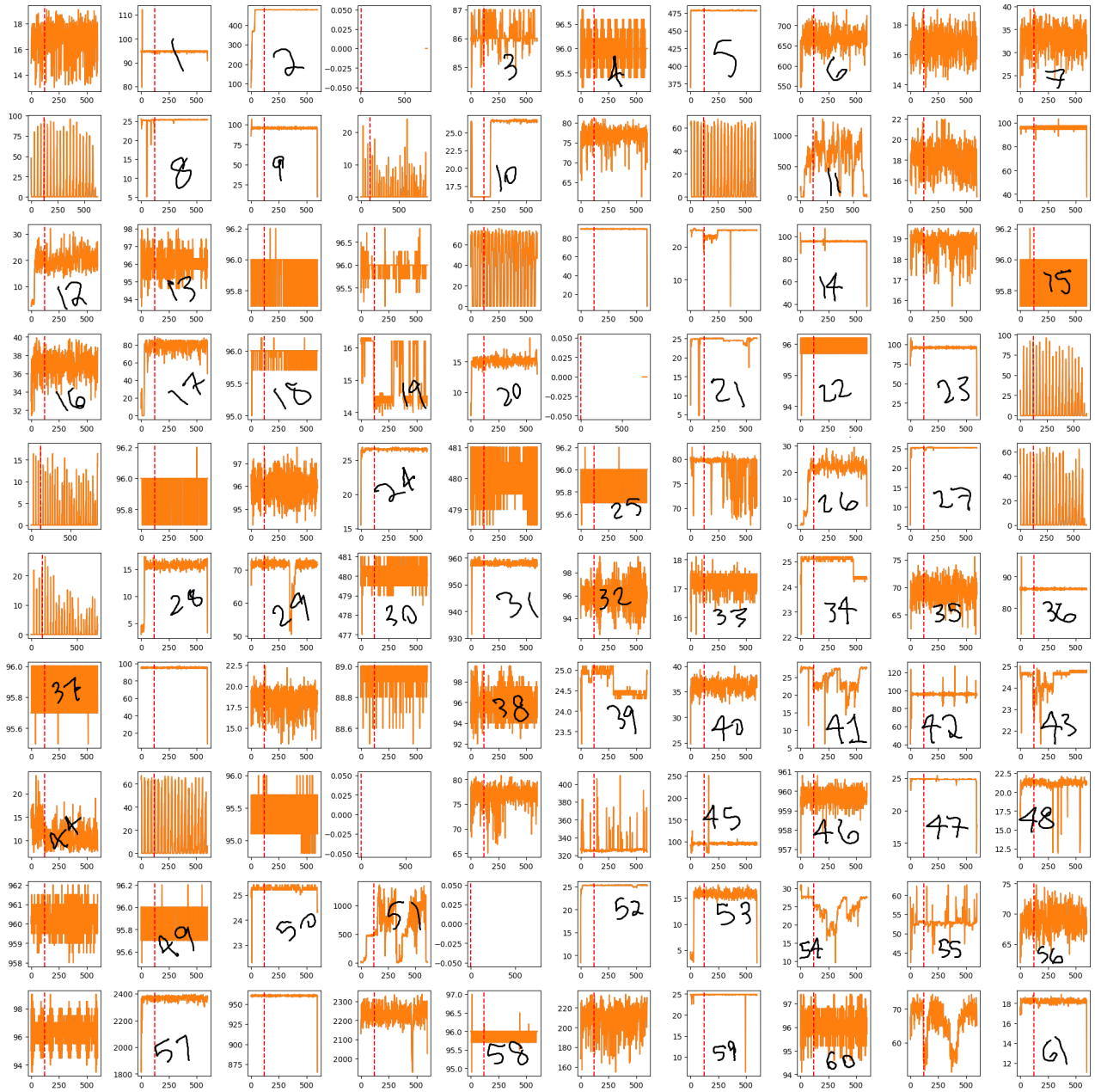

Figure 3.8: Example of visual inspection of throughput data.

3.2.4 Dataset Processing

After removing the first 20% of all data, it was time to reduce the files per test into rows per test. This section describes the creation of a single CSV file capturing the performance of every test for each row.

When predicting the performance of a system, it is usual to predict a single value - the average. In some cases, researchers also predict the standard deviation or variance. However, our aim is to best capture the performance and ideally by capturing the distribution of the data. There is one end of the spectrum which is to predict a single value to capture the performance and the other end of the spectrum is to predict the entire distribution. We have opted to take an approach somewhere in the middle. Models theoretically perform worse in the case where there are more targets to predict. Therefore, we have chosen to recreate the distributions according to various statistics gathered from the data. These statistics include: mean, standard deviation, minimum, maximum, and the following quantiles:

```
1, 2, 5, 10, 20, 25, 30, 40, 50, 60, 70, 75, 80, 90, 95, 99
```

We wanted to capture the worst case performances. In the case of the latency this would be the highest values which reside in the tail of the distribution while for the throughput, the worst values would be the lowest in magnitude and would reside in the head of the distribution. This is why the first, second, fifth, ninety-fifth, and ninety-ninth percentiles have specifically been included in the statistics used to recreate the distribution of the performance.

Therefore, for each metric column produced from the previous data processing steps, we gathered the mentioned statistics. This means that we reduce a single column of a metric down to 20 numbers and given that there are 13 metrics, a total of 260 columns are created to capture the distribution of the metrics per

test. All of this data was combined with the parameter settings used for the test including the duration in seconds, data length in bytes, publisher count, subscriber count, reliability, multicast, latency count, and durability. All Boolean columns e.g. reliability were parsed into integers with 0 representing False and 1 representing True and categorical columns e.g durability had one-hot encoding applied to them meaning that the durability column would be split into 4 columns because of the 4 options. A value of 1 would mean that that durability specific option would be used e.g if a durability value of 3 was used in the test then a 0 would be shown for all durability columns except for "durability_3" which would have a value of 1.

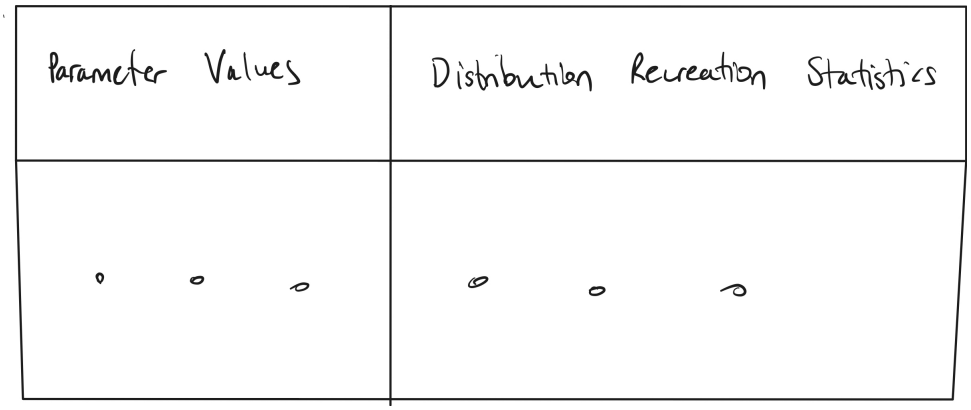

Figure 3.9: The columns that make up the dataset.

Figure 3.9 visually shows how the columns are split in the dataset where the parameter values are the duration in seconds, data length in bytes, publisher count, subscriber count, reliability, multicast, latency count, and durability and the distribution recreations statistics are the 19 statistics gathered for every metric e.g. latency, throughput, etc. We therefore end up with one CSV file where each row represents a single test with the parameter values of the test in the first few columns and the distribution recreation statistics per metric for all other columns. In total, this dataset has 1,585 rows and is ready for an exploration of the configuration space.

3.3 Configuration Space Exploration

There are two techniques for predicting values: interpolation and extrapolation. Interpolation involves the estimation of values within the range of known data points whilst extrapolation involves the estimation of values outside of the range. Ideally, the perfect scenario would be a model that can accurately extrapolate. This would mean that one could collect data of configurations that are not too resource-intensive to predict configurations that could be too resource-intensive. Therefore, we had to define the range differentiating between interpolation and extrapolation.

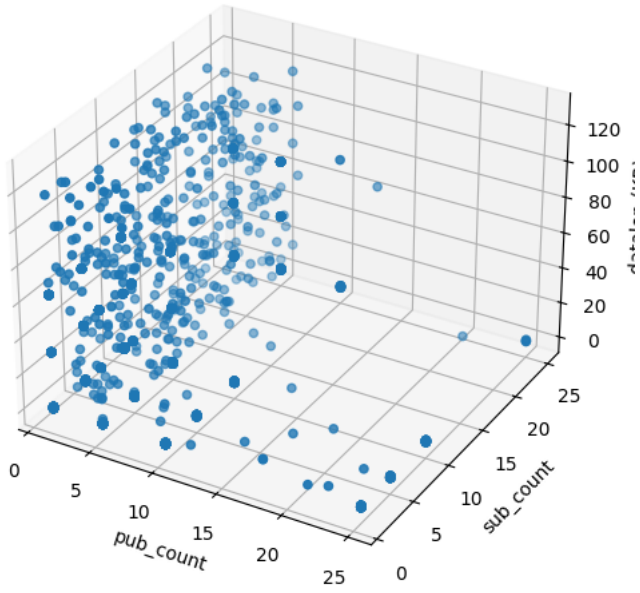

Figure 3.10: 3D scatter plot of the values for publisher/subscriber count and data length (bytes).

Of the six parameters, 3 took numerical values and we created a 3-dimensional grid according to these parameters: publisher count, subscriber count, and data length. Figure 3.10 shows a 3D scatter plot, while figure 3.11 shows three 2D scatter plots where the axes show all variations of combinations of the 3D plot.

Ideally, the data should be split in an 80-20 ratio, with 80% of the data used for training and the other 20% used for testing. Then, for interpolation, the model

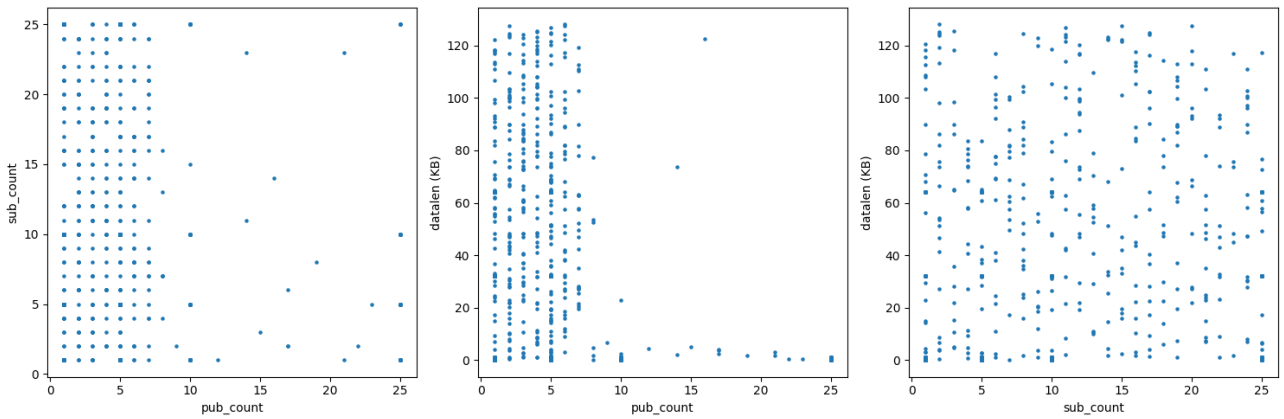

Figure 3.11: All variations of the 3D scatter plot as seen in 2D.

would train and test within the range while in the case of extrapolation, the model would train within the range and test outside of the range as shown in Figure 3.12.

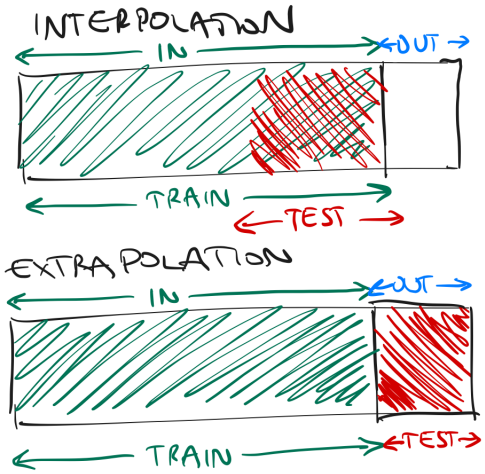

Figure 3.12: How the data is allocated per train/test phase for interpolation and extrapolation.

It should be noted that in either case (interpolation or extrapolation), the amount of data used for training and testing should remain the same to allow for a fair comparison.

To define the range that also fits with the 80-20 split, we started off by defining values for the 3 parameters as shown below:

```
pub_counts = [5, 10, 15, 20, 25]
sub_counts = [5, 10, 15, 20, 25]
```

```
datalen_bytes = [40_000, 60_000, 80_000, 100_000, 120_000]
```

The Cartesian product would then be applied to all 3 lists, producing all possible combinations of values for the 3 parameters. For each combination, the number of tests with values lower than or equal to the chosen values were counted as well as the opposite case: tests with values greater. The inside and outside percentages were also calculated to find the best match of 80% inside and 20% outside. There were over hundred different combinations with various inside and outside percentages. We therefore ordered the rows of data according to the highest inside percentage first before narrowing the range of inside values between 75% to 85%. This resulted in 6 combinations as shown in Table 3.1.

pub count	sub count	datalen	in count	out count	in percentage	out percentage
20	25	80000	1235	235	84.0136	15.9864
15	25	80000	1231	239	83.7415	16.2585
10	25	80000	1227	243	83.4694	16.5306
25	20	120000	1145	325	77.8912	22.1088
5	25	120000	1125	345	76.5306	23.4694
25	20	100000	1104	366	75.102	24.898

Table 3.1: All combinations of values for publisher count, subscriber count, and data length alongside the allocation of data inside and outside the range.

This was narrowed down further to look for percentages between 76% and 84% (inclusive). This eliminated the top and bottom combinations in Table 3.1. On top of this, we decided to eliminate the case with 5 publishers as this value was too low to appropriately spread the data across the parameter as well as the case of 25 publishers which covered all publisher values. A split across all values of all parameters would have been ideal, though this was not achievable because the final two cases are shown in Table 3.2. The closest percentage to the 80-20 split is the combination of 10 publishers, 25 subscribers, and 80KB for the data length.

That concludes the explanation on how the configuration space was reduced to

pub count	sub count	datalen	in count	out count	in percentage	out percentage
15	25	80000	1231	239	83.7415	16.2585
10	25	80000	1227	243	83.4694	16.5306

Table 3.2: The final two combinations of values for publisher count, subscriber count and data length when looking for the ideal split for interpolation and extrapolation.

the 3 numerical parameters to find the closest values for all 3 parameters that would suit the 80-20 data split for training and testing. The final split is visually shown in Figure 3.13 showing the values of the 3 parameters for the boundary, the number of tests inside (green), the number of tests outside (blue) and the percentage of tests inside and outside.

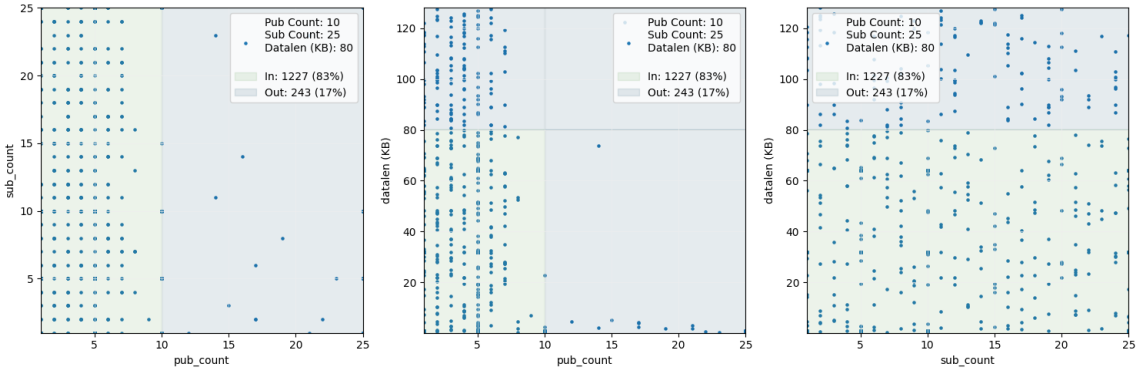

Figure 3.13: Chosen values for the 3 numerical parameters and allocated tests within (green) and outside (blue) of the range.

The aim was to split the dataset into a training and testing set where the training set takes up 80% of the data and the testing set takes up the rest. Ideally we would have liked to split the numerical data accordingly i.e. if the publishers ranged from 1 to 100 then train on 1 to 80 publishers and test on 81 to 100 publishers. However, this could not be achieved because of how the data is spread within the configuration space. The tests are not equally spread throughout the space and allocating an 80-20 split according to the values of the axes is therefore not feasible. Thus, we have opted for the best option regarding the parameter values (10 publishers, 25 subscribers and 80KB).

3.4 Model Training and Evaluation

The training of the model was carried out separately for interpolation and extrapolation. In both cases, the input parameters were standardised and the target variables were transformed (though we did experiment with the raw values for both the input parameters and the target variables). This section will cover the various standardisation procedures applied to the input parameters, transformation functions applied to the target variables, and evaluation metrics recorded.

3.4.1 Standardisation

Standardisation in the context of ML and statistics refers to the process of rescaling or transforming features. The usual method of standardisation is known as "z-score normalisation" and rescales the data to have a mean of 0 and a standard deviation of 1. The standardisation methods used in the model training include no standardisation, z-score, min-max, and robust.

Z-Score

$$\text{standardised\_X} = \frac{X - \mu}{\sigma}$$

For z-score normalisation, one must first calculate the mean and standard deviation of the feature. Then, for each data point of the feature, the mean is subtracted from the data point and the result is divided by the standard deviation of the feature. This results in the data having a mean of 0 and a standard deviation of 1.

Min-Max

$$\text{Min-Max\_Scaled\_X} = \frac{(X - \text{min\_val})}{\text{max\_val} - \text{min\_val}} \cdot (b - a) + a$$

Min-max scaling works by first calculating the minimum and maximum values of the target. For each data point the minimum value is subtracted from the data point and the result is divided by the range (maximum minus minimum) of the feature. This scales the data to a specified range, typically this lies between 0 and 1 though this can be customised according to the desired conditions.

Robust

$$\text{Robust\_Scaled\_X} = \frac{X - M}{IQR}$$

In the case of robust scaling, the median and interquartile range of the feature are calculated. The median is subtracted from every data point in the future and the result of this subtraction is divided by the interquartile range. Robust scaling is less sensitive to outliers compared to min-max scaling and z-score normalisation. It brings the data into a consistent range based on the median and interquartile range.

Overall, standardisation is a useful preprocessing technique for several reasons. Firstly, standardisation equalises the scales of all features since all features are scaled to have a mean of 0 and standard deviation of 1. This makes all features directly comparable and prevents features with larger numeric values such as the data length from dominating those with smaller values e.g. publisher count and subscriber count. Secondly, many ML algorithms, particularly those based on gradient descent such as linear regression, converge faster when the input features are standardised because it helps them reach their optimal solution faster, reducing training time. There are many other reasons but one of the main reasons to use robust scaling is because it is less affected by outliers compared to min-max scaling. This is because outliers have a small impact on the mean and standard deviation and so they don't disproportionately influence the scaling. In our work we have only applied the mentioned

standardisation methods to the numerical input parameters: publisher/subscriber count and data length.

3.4.2 Transformation

Transformations in machine learning refer to the process of applying mathematical or statistical operations to the data to modify its features or structure. These transformations can be crucial for improving model performance, handling non-linear relationships, and making the data more suitable for specific algorithms. We mainly focused on using variations of the log transformations as we noticed when analysing the data that there were often large values that would impact the modelling. When applying the log transformation, these larger values have as equal of an impact on the modelling as the smaller values. We explored the following transformation functions: log, log10, log2, log1p, and square root.

log, log2, and log10

$$log(x)$$

This transformation applies the natural logarithm function to each data point and compresses larger values more than smaller ones. This is useful when dealing with data that exhibits exponential growth or decay because it stabilises the variance of data with a multiplicative effect. It can transform data with exponential relationships into linear ones, making it suitable for linear models such as the one we use - linear regression. $log_2(x)$ is similar to the log transformation but uses base 2 logarithms while $log_{10}(x)$ uses base 10.

log1p

$$log(1 + x)$$

This transformation adds 1 to each data point before applying the natural logarithm and is often used when dealing with data containing zero values. This is the case with the number of lost samples when reliability is being used, and that is why we have opted to use this transformation.

Square Root

The square root transformation takes the square root of each data point, which also compresses larger values more than smaller ones, similar to the log transformations. This is useful for dealing with data with non-constant variance, a characteristic often seen in the performance of distributed systems. Like the log transformation, the square root transformation can also make data more linear and suitable for linear models such as linear regression.

3.4.3 Training

Linear regression was chosen as a baseline for comparison, whilst random forests was the selected model to be used as the first step in using ML with DDS for performance prediction. Regardless, both models went through the same training process. For both models, the inputs were standardised and the outputs were transformed for all combinations of standardisation on inputs and transformation on outputs, e.g. for z-score on inputs, all transformations were used (log, log2, log10, log1p and square root).

Before processing is applied to the inputs and outputs, the dataset is split into the training and test splits. For interpolation, this involves randomly sampling 80% of the data within the range for training and using the remaining 20% of the data within the range as testing. Given that there are 1,585 tests in total and we defined the boundary at 10 publishers, 25 subscribers, and 80KB, the number of tests inside

the boundary is 1,227 whilst outside is 243. Therefore, in the case of interpolation 80% of 1,227 which is 982, tests are used for training and the remaining 20% (245) are for testing.

In the case of extrapolation, training is done with a randomly sampled proportion of 80% of the data within the range - the same case as interpolation. However, the key difference with extrapolation is the testing data. For the case of extrapolation, 245 tests are used for testing. However, there are only 243 tests outside of the boundary. Therefore, for any random set of training data for extrapolation, the testing data always consists of the same 243 tests outside the boundary.

Models are trained on a single DDS performance metric, so there is one model per DDS performance metric e.g. number of lost samples. After allocating the training and testing datasets, the standardisation is applied to the inputs and the transformations to the outputs. The model is then trained and the error metrics are calculated on the predictions. It is critical to highlight that the error metrics are calculated for each statistic used to recreate the distribution of the metric, e.g. 5^{th} percentile of latency. This means that all evaluation metrics mentioned later e.g. Root Mean Squared Error, are applied for each statistic, e.g. 60^{th} percentile. On top of this, the error metrics are averaged over all outputs. We should mention that when transformations are applied to the outputs, the transformations are reversed before calculating the error metrics.

After the model has been trained, all data associated with that model are written to files including the model name, model type, training example count, testing example count, input variables, output variables, DDS performance metric of interest, standardisation function, transformation function, and errors per output variable. This data is then used later on for comparing all models to find the ones with the

best performance.

3.4.4 Evaluation Metrics

Evaluation metrics in machine learning are used to assess the performance of models and quantify how well they are making predictions. These metrics help you understand how effectively your model is performing on a given task. We have recorded the following metrics in our works: Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Median Absolute Error (MedAE), and Explained Variance.

Mean Squared Error

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

Mean Squared Error (MSE) is calculated as the average of the squared differences between predicted and actual values and penalises large errors, making it sensitive to outliers. This is useful for tasks where you want to quantify the magnitude of prediction errors and penalise larger errors more heavily. In our work, we focus on using the Root Mean Squared Error (RMSE), which is the squared root of the MSE. The reason for this decision is that the RMSE is more interpretable because it is in the same units as the variable.

Mean Absolute Error

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

This is calculated as the average of the absolute differences between predicted and actual values, is less sensitive to outliers compared to MSE and is useful when you want a straightforward measure of prediction accuracy and do not want to heavily

penalise outliers.

Mean Absolute Percentage Error

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{|y_i - \hat{y}_i|}{|y_i|}\right) \times 100\%$$

This is calculated as the average absolute percentage differences between predicted and actual values. This is useful when you want to express prediction errors as a percentage of the true values, making it easier to interpret.

Median Absolute Error

$$\text{MedAE} = \text{median}\left(|y_i - \hat{y}_i|\right)$$

This is calculated as the median of the absolute differences between predicted and actual values and is less affected by outliers than MAE. It is useful when the data dataset may contain outliers and a robust measure of prediction error is needed.

Explained Variance

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

This metric measures the proportion of the variance in the dependent variable explained by the independent variables. While R^2 and Explained Variance are often used interchangeably, they typically refer to the same concept: measuring the goodness of fit of a regression model.

Chapter 4

Results

To recap, one model (linear regression or random forests) is trained for a specific DDS performance metric (e.g., throughput) in interpolation and extrapolation. For each model, the error metrics are taken for the training and testing phases. Therefore, for each DDS performance metric, there are two possible models (linear regression or random forests) trained under two possible situations (interpolation or extrapolation) with two specific error metrics of focus (RMSE and R^2) for the two phases (training and testing). This means every model trained on a specific DDS performance model has $2 \times 2 = 2^2 = 4$ variations. All source code and data including error metrics over the DDS metric distribution for all models can be found in Appendix A.

Metric	R2 Train LR Int	R2 Test LR Int	R2 LR Int % Diff	R2 Train LR Ext	R2 Test LR Ext	R2 LR Ext % Diff	R2 Train RF Int	R2 Test RF Int	R2 RF Int % Diff	R2 Train RF Ext	R2 Test RF Ext	R2 RF Ext % Diff
Avg Lost Samples	0.198	0.159	-20%	0.083	0.277	234%	0.994	0.982	-1%	0.996	0.699	-30%
Avg Lost Samples (%)	0.251	0.207	-18%	0.146	0.237	62%	0.984	0.933	-5%	0.986	0.907	-8%
Avg Received Samples	0.387	0.453	17%	0.397	0.028	-93%	0.998	0.999	0%	0.999	0.78	-22%
Avg Received Samples (%)	0.251	0.201	-20%	0.24	-0.023	-110%	0.986	0.893	-9%	0.988	0.908	-8%
Avg Sample Rate	0.494	0.534	8%	0.431	0.363	-16%	0.994	0.969	-3%	0.994	0.952	-4%
Avg Throughput (Mbps)	0.497	0.536	8%	0.511	0.192	-62%	0.982	0.94	-4%	0.986	0.921	-7%
Latency (us)	0.083	0.225	171%	0.117	0.029	-75%	0.723	0.636	-12%	0.734	0.188	-74%
Total Lost Samples	-0.268	-62.23	-23120%	0.009	-0.187	-2178%	0.113	-1.211	-1172%	0.311	-1.816	-684%
Total Received Samples	0.313	0.241	-23%	0.275	-0.082	-130%	0.984	0.922	-6%	0.965	0.427	-56%
Total Sample Rate	0.52	0.259	-50%	0.434	-1.548	-457%	0.958	0.885	-8%	0.898	0.711	-21%
Total Throughput (Mbps)	0.333	0.37	11%	0.522	0.466	-11%	0.987	0.939	-5%	0.988	0.901	-9%

Table 4.1: R2 values during training and testing for Linear Regression (LR) and Random Forests (RF) for interpolation (int) and extrapolation (ext).

Table 4.1 shows all R^2 scores averaged for the model trained for a specific DDS performance metric, while Table 4.2 shows the same for RMSE.

Metric	RMSE Train LR Int	RMSE Test LR Int	RMSE Train LR Ext	RMSE Test LR Ext	RMSE Train RF Int	RMSE Test RF Int	RMSE Train RF Ext	RMSE Test RF Ext
Avg Lost Samples	898,871.4	988,578.1	980,807.9	1,044,115.3	77,933.5	142,382.6	59,083.9	123,808.7
Avg Lost Samples (%)	14.4	13.7	14.5	13.2	1.9	3.2	1.6	4.4
Avg Received Samples	866,112.6	717,621.7	835,644.4	1,082,257.3	28,213.8	31,133.4	25,825.6	261,104.7
Avg Received Samples (%)	14.4	13.8	13.7	15.6	1.8	4.2	1.5	4.4
Avg Sample Rate	4,704.5	4,026.4	4,711.9	6,796.4	202.0	379.2	205.8	1,126.8
Avg Throughput (Mbps)	25.9	26.4	26.2	30.8	1.6	5.3	1.3	8.6
Latency (us)	3,045,001.4	1,454,728.3	2,848,990.0	9,509,031.8	1,139,337.4	771,102.4	916,733.7	7,903,234.4
Total Lost Samples	7,337,539.8	31,204.0	4,131,444.1	2,691,201.6	7,304,408.1	76,398.2	5,656,594.0	183,124.5
Total Received Samples	756,761.8	791,921.4	712,238.7	1,295,118.4	109,964.1	235,794.3	163,318.2	753,747.7
Total Sample Rate	1,839.6	2,018.7	1,745.8	4,918.2	454.0	615.0	651.0	2,206.5
Total Throughput (Mbps)	326.6	361.5	319.2	278.1	23.1	47.3	23.9	149.6

Table 4.2: RMSE values during training and testing for Linear Regression (LR) and Random Forests (RF) for interpolation (int) and extrapolation (ext).

The analysis of the results will initially break down into looking at linear regression and random forests. We analyse interpolation versus extrapolation for each model by analysing the R^2 and RMSE.

4.1 Linear Regression

4.1.1 Interpolation

In the context of our performance analysis, it is evident that linear regression models may not be the most suitable choice for modelling performance metrics. Among the 12 models under consideration, only 9 exhibited R^2 scores exceeding the threshold of 0.2, highlighting that the remaining three models performed inadequately. The highest R^2 score recorded in our analysis was 0.54, achieved by the model focusing on the average throughput metric. Another model, specifically designed to capture variations in the average sample rate, demonstrated relatively promising results with an R^2 score of 0.53. Further inspection reveals three additional models surpassed the 0.4 R^2 threshold, while four models exceeded 0.3. In summary, our comprehensive evaluation indicates that the overall performance of these linear regression models can be characterised as sub-optimal for the given dataset and metrics.

4.1.2 Extrapolation

Our comprehensive performance evaluation shows that the overall performance of the models under consideration falls short of expectations. The observed performance is notably inferior to that achieved by interpolation methods. Specifically, only 4 of the considered models managed to attain R^2 scores exceeding the threshold of 0.2. Among these models, a mere two could surpass a more stringent threshold of 0.3. Notably, the highest-performing model in our analysis was associated with the

total throughput metric, achieving an R^2 score of 0.47. Our rigorous evaluation underscores the sub-optimal performance of the models in question, which fails to meet the desired standards.

Metric	**R2 LR Int**	**R2 LR Ext**	**R2 LR % Diff (Int vs Ext)**
Avg Lost Samples	0.159	0.277	74%
Avg Lost Samples (%)	0.207	0.237	14%
Avg Received Samples	0.453	0.028	-94%
Avg Received Samples (%)	0.201	-0.023	-111%
Avg Sample Rate	0.534	0.363	-32%
Avg Throughput (Mbps)	0.536	0.192	-64%
Latency (us)	0.225	0.029	-87%
Total Lost Samples	-62.23	-0.187	100%
Total Received Samples	0.241	-0.082	-134%
Total Sample Rate	0.259	-1.548	-698%
Total Throughput (Mbps)	0.37	0.466	26%

Table 4.3: Interpolation vs extrapolation for linear regression focusing on the r-squared metric.

Metric	**RMSE LR Int**	**RMSE LR Ext**	**RMSE LR % Diff (Int vs Ext)**
Avg Lost Samples	988,578.1	1,044,115.3	6%
Avg Lost Samples (%)	13.7	13.2	**-4%**
Avg Received Samples	717,621.7	1,082,257.3	51%
Avg Received Samples (%)	13.8	15.6	13%
Avg Sample Rate	4,026.4	6,796.4	69%
Avg Throughput (Mbps)	26.4	30.8	17%
Latency (us)	1,454,728.3	9,509,031.8	554%
Total Lost Samples	31,204.0	2,691,201.6	8525%
Total Received Samples	791,921.4	1,295,118.4	64%
Total Sample Rate	2,018.7	4,918.2	144%
Total Throughput (Mbps)	361.5	278.1	**-23%**

Table 4.4: Interpolation vs extrapolation for linear regression focusing on the RMSE metric.

Table 4.3 compares R^2 scores between interpolation and extrapolation techniques in linear regression. The focus here lies on instances where extrapolation exhibits superior performance, resulting in a positive percentage difference.

In four specific cases, extrapolation outperforms interpolation: Average Lost

Samples, Average Lost Samples (%), Total Lost Samples, and Total Throughput.

Regarding the Lost Samples category, it's worth noting that a significant portion of the data consists of zeroes, making extrapolation advantageous. In this scenario, the model predicts zero values due to the high frequency of zeroes in the data. This is evident in the R^2 scores, where interpolation achieves 0.16, while extrapolation notably improves to 0.28, as observed in Average Lost Samples.

An intriguing case arises with Total Throughput. Interpolation yields an R^2 score of 0.37, while extrapolation achieves an impressive 0.47, marking a noteworthy 26% improvement. This suggests that linear regression has the potential for effective extrapolation, albeit with room for improvement in R^2 performance.

Notably, the R^2 scores for extrapolation, while showing improvement in these cases, still indicate room for enhancement. Further refinement and optimisation of the linear regression model to better accommodate extrapolation scenarios may be warranted.

In examining the RMSE, we find that in the case of Total Throughput, the RMSE decreases from 362 to 278, implying a 23% reduction. This reduction in RMSE reinforces that linear regression holds promise for extrapolation but underscores the need for continued improvement.

The results encourage consideration of improving the existing linear regression model to enhance its extrapolation capabilities. A thorough investigation into the model's limitations and potential enhancements could provide valuable insights into its suitability for extrapolation tasks.

4.1.3 Summary and Conclusion

The comprehensive analysis of linear regression models across 12 performance metrics yields mixed results, with significant implications for interpolation and extrap-

olation methods. The R^2 scores indicate that linear regression may not be ideally suited for modelling these specific performance metrics, with only a subset of models surpassing minimal performance thresholds. The highest R^2 score observed was 0.54 for average throughput in interpolation, but overall, the models generally showed sub-optimal performance.

Extrapolation results were notably inferior to interpolation, with only a few models achieving R^2 scores above 0.2, and even fewer surpassing 0.3. However, metrics like Total Throughput exhibited better performance in extrapolation than interpolation, suggesting potential areas where linear regression might be more effective.

The analysis of RMSE metrics further complements these findings, with significant variations observed between interpolation and extrapolation. In some cases, such as Total Throughput, a notable decrease in RMSE during extrapolation was observed, indicating better predictive accuracy.

These findings highlight the nuanced and context-specific nature of linear regression's applicability to performance metric prediction. While certain metrics show promise, particularly in extrapolation, the overall performance suggests a need for refinement and optimisation of the models. Future work should focus on enhancing linear regression techniques for specific metrics, especially those that have shown potential in extrapolation scenarios. This would involve a more detailed investigation into the underlying factors influencing model performance and incorporating more sophisticated modelling approaches.

4.2 Random Forests

4.2.1 Interpolation

In the evaluation of the 11 models, one model fell within the 0.6 to 0.7 range in terms of performance, specifically in relation to latency. Two models scored between 0.8 and 0.9, namely the average received samples percentage and the total sample rate. Notably, seven models demonstrated very good performance, each scoring higher than 0.9.

The model with the lowest performance registered a score of -1.2, associated with the total lost samples. This poor performance is likely attributable to the predominance of zero values in the dataset. Conversely, the best-performing model achieved a near-perfect score of 0.999, specifically in the category of predicting average received samples.

Overall, an impressive 9 out of the 11 models displayed incredibly good performance in terms of the R^2 metric. Only one model was considered mediocre, and the worst-performing model was significantly poor. The nine models that exhibited superior performance are as follows: average lost samples, average lost samples percentage, average received samples, average received samples percentage, average sample rate, average throughput, total received samples, total sample rate, and total throughput.

The root mean square error (RMSE) is challenging to interpret in isolation, requiring a comparison with the average value of the metric for meaningful insights. To address this, the RMSE has been divided by the mean for all metrics under consideration, producing the relative RMSE (RRMSE).

The lowest relative error observed was 4% for the average received samples percentage, indicating a high degree of accuracy for this model. On the other end of

Metric	R2 RF Int	R2 RF Ext	R2 RF % Diff (Int vs Ext)
Avg Lost Samples	0.982	0.699	-29%
Avg Lost Samples (%)	0.933	0.907	-3%
Avg Received Samples	0.999	0.78	-22%
Avg Received Samples (%)	0.893	0.908	2%
Avg Sample Rate	0.969	0.952	-2%
Avg Throughput (Mbps)	0.94	0.921	-2%
Latency (us)	0.636	0.188	-70%
Total Lost Samples	-1.211	-1.816	-50%
Total Received Samples	0.922	0.427	-54%
Total Sample Rate	0.885	0.711	-20%
Total Throughput (Mbps)	0.939	0.901	-4%

Table 4.5: Interpolation vs extrapolation for random forests focusing on the r-squared metric.

the spectrum, the worst relative error was found in latency, at a significant 64%.

In terms of relative RMSE (RRMSE), the performance varied across models: Two models had an RRMSE of less than 10%, specifically average received samples and average received samples percentage. Two models fell within the 10 to 20% range, namely average sample rate and average throughput. Three models exhibited RRMSEs between 20 to 30%, including total lost samples, total sample rate, and total throughput. One model was in the 30 to 40% range, specifically total received samples. Finally, two models had their RRMSEs between 50 to 60%, which were average lost samples and average lost samples percentage.

The most outstanding model in this analysis was the average received samples model, boasting an R^2 of 0.999 and a remarkably low RRMSE of 4%. This reflects its exceptional accuracy and precision. Additionally, several other models demonstrated very good overall performance. The average received samples percentage model, with an R^2 of 0.89 and an RRMSE of 4%, performed almost as well as the top model, albeit with a slightly lower R^2 score. The average sample rate model showed a strong correlation with an R^2 of 0.95, though its RRMSE was higher at 33%. Similarly, the average throughput model recorded an R^2 of 0.94 and an RRMSE of 13%, indicating

a high level of accuracy with a moderate error relative to the mean. Finally, the total throughput model also presented a strong predictive capability with an R^2 of 0.94, but a somewhat higher RRMSE of 21%, reflecting a reasonable level of error in relation to the mean.

4.2.2 Extrapolation

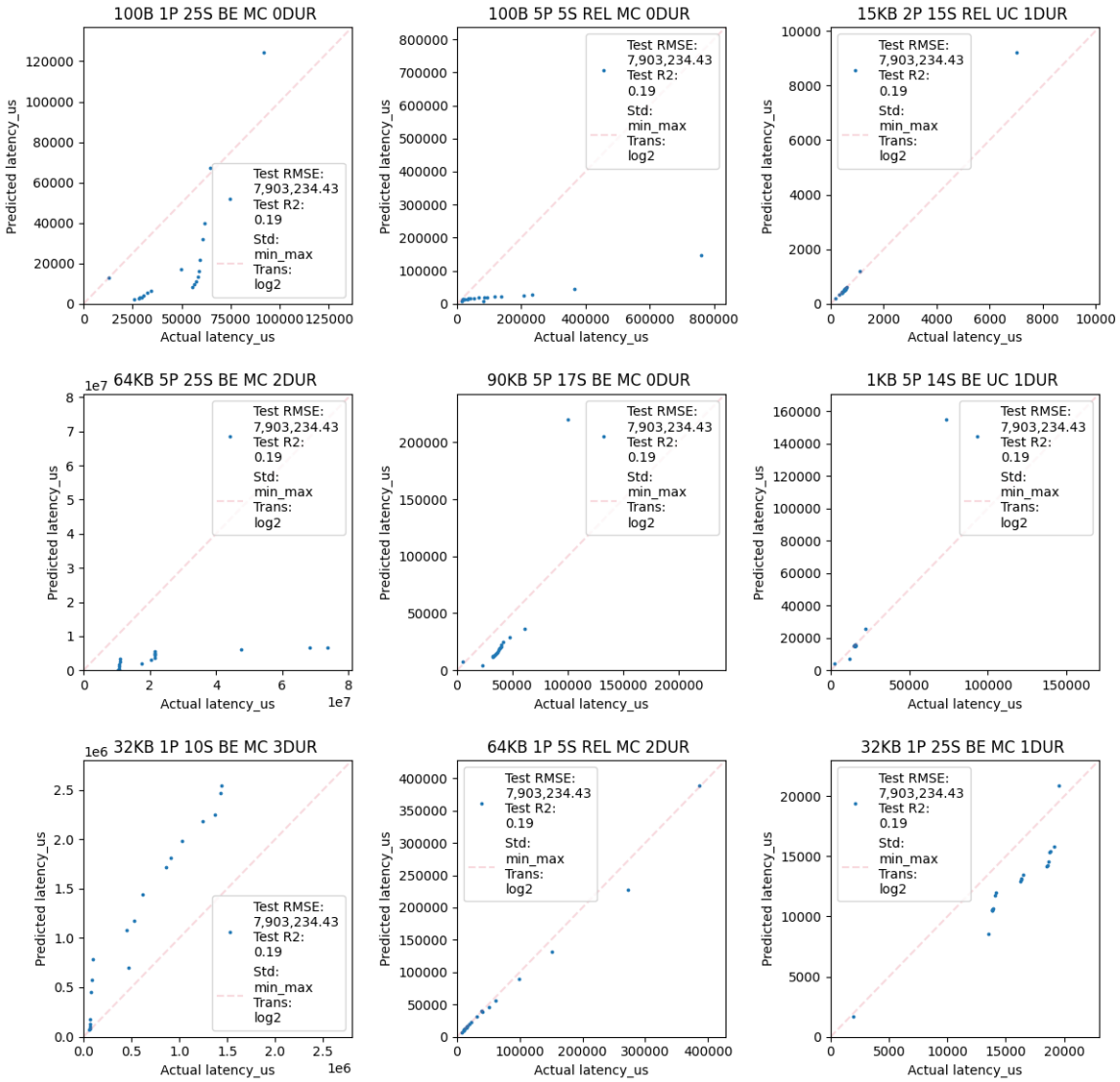

Figure 4.1: Scatter plots of predicted values on y-axis and actual values on x-axis for 9 randomly picked tests using the random forests model to extrapolate for latency.

In the evaluation of a total of 11 models, the performance varied significantly across different metrics. Five models scored between 0.9 and 1, demonstrating high

efficacy, while two models were in the 0.7 to 0.8 range. The average received samples was noted at 0.78, and the total received samples were considerably lower at 0.43. Latency, with a value of 0.19, was identified as particularly poor. Scatter plots shown in 4.1 indicated a trend where poor predictions often significantly underestimated actual values. Notably, 5 out of the 11 models excelled in extrapolation, a critical aspect for predictive analysis. These models included average lost samples percentage, average received samples percentage, average sample rate, average throughput, and total throughput, highlighting that percentage-based metrics are generally more reliable for extrapolation, especially in throughput-related data where percentage values are more insightful than absolute figures.

Metric	**RMSE RF Int**	**RMSE RF Ext**	**RMSE RF % Diff (Int vs Ext)**
Avg Lost Samples	142,382.6	123,808.7	-13%
Avg Lost Samples (%)	3.2	4.4	38%
Avg Received Samples	31,133.4	261,104.7	739%
Avg Received Samples (%)	4.2	4.4	5%
Avg Sample Rate	379.2	1,126.8	197%
Avg Throughput (Mbps)	5.3	8.6	62%
Latency (us)	771,102.4	7,903,234.4	925%
Total Lost Samples	76,398.2	183,124.5	140%
Total Received Samples	235,794.3	753,747.7	220%
Total Sample Rate	615.0	2,206.5	259%
Total Throughput (Mbps)	47.3	149.6	216%

Table 4.6: Interpolation vs extrapolation for random forests focusing on the RMSE metric.

Focusing on RRMSE, the average received samples percentage model emerged as the best, with an RRMSE of 5%. The average throughput model followed with an RRMSE of 22%, and the average sample rate had an RRMSE of 33%. Two models each had an RRMSE of 47% and 60-70%, respectively, while the total sample rate's RRMSE was disappointingly high at 94%, and the total received samples even higher at 108%. Latency fared the worst with an RRMSE of 654%, possibly due to cases

where predictions were significantly off, resulting in high errors that skewed the average. To further understand this, an analysis of relative errors is necessary.

Moreover, the best performing model in terms of R^2 was the average received samples percentage, with an R^2 of 0.91 and an RRMSE of 5%. Other models showing commendable performance included the average sample rate with an R^2 of 0.95 and RRMSE of 33%, and the average throughput with an R^2 of 0.92 and RRMSE of 22%. In stark contrast, latency was the worst-performing model with an R^2 of 0.19 and an RRMSE of 654%, and the total lost samples model also fared poorly with an R^2 of -1.82 and an RRMSE of 69%.

4.2.3 Summary and Conclusion

In summary, the comprehensive evaluation of 11 Random Forest models revealed a diverse range of performances across different metrics. The analysis highlighted a clear distinction in model capabilities, particularly in terms of R^2 scores and RRMSE. Models like the average received samples and average received samples percentage demonstrated exceptional accuracy, with high R^2 scores and low RRMSEs, indicating their robustness in both interpolation and extrapolation scenarios. Conversely, models such as latency and total lost samples performed poorly, characterised by low R^2 values and high RRMSEs, suggesting significant room for improvement in these areas.

The findings also underline the importance of considering both the R^2 metric and RRMSE for a holistic understanding of model performance. While some models showed high R^2 scores, their relative errors were significantly higher, emphasising the need for a balanced approach in model evaluation. The analysis of scatter plots further enriched these insights, revealing trends in prediction accuracy and the tendency of some models to underestimate values in certain scenarios.

Overall, this evaluation of Random Forest models underscores the effectiveness of certain models in handling complex data predictions, especially in cases where percentage values are more relevant than absolute numbers. However, it also points to the challenges faced by models in accurately predicting metrics with a high variance or a significant number of zero values. These insights are invaluable for guiding future improvements and optimisations in model development.

Chapter 5

Conclusion

In conclusion, this study marks the inaugural introduction of a machine learning-based method to predict Data Distribution Service (DDS) performance, taking into account both Quality of Service (QoS) and non-QoS parameters. A comparative analysis between linear regression and random forests for both interpolation and extrapolation was conducted. Random forests emerged as the superior model, especially for interpolation, exhibiting high performance as indicated by R^2 values exceeding 0.9. For extrapolation, the models showed varied performance, with some achieving close parallels to interpolation models in terms of R^2 and (RMSE). The high efficacy of the random forests interpolation model is further underscored by its ability to predict latency with an R^2 of 0.64 and RMSE of approximately 770,000, indicating mixed accuracy. Despite a lower R^2, the latency model can still yield sufficiently accurate predictions. Machine learning models present a cost-effective approach for estimating DDS performance in untested conditions. However, exceptional performance across all DDS metrics is not assured. Reconstructing DDS metric distributions aids in understanding performance behaviour, particularly focusing on specific parts of the distribution. One limitation of this study is its reliance

on a DDS performance benchmark, which may not perfectly mirror real-life applications. The methodology is versatile, applicable to various workloads, and requires only the collection of data and training of machine learning models. Future work will explore the use of neural networks, diverse sampling strategies, and examine the impact of configuration locations on prediction accuracy.

Appendix A

Source Code and Data

All source code and data used in this report can be found in the following GitHub repository: https://github.com/Blitz3r123/Machine-Learning-for-Performance-Prediction-of-DDS-Oc tab=readme-ov-file.

Bibliography

Bao, L., Liu, X., Xin Liu, Xin Liu, Ziheng, X. & Fang, B. (2018), 'AutoConfig: automatic configuration tuning for distributed message systems', pp. 29–40. MAG ID: 2888705583.

Bouloukakis, G., Kattepur, A., Georgantas, N. & Issarny, V. (2018), 'Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols', pp. 176–186. MAG ID: 2904555933.

Data Distribution Service (DDS) (n.d.).

Grebhahn, A., Siegmund, N. & Apel, S. (2019), 'Predicting Performance of Software Configurations: There is no Silver Bullet.', *arXiv: Software Engineering* . MAG ID: 2991131477.

Guo, J., Yang, D., Siegmund, N., Apel, S., Sarkar, A., Valov, P., Czarnecki, K., Wasowski, A., Huiqun Yu, Huiqun Yu & Yu, H. (2018), 'Data-efficient performance learning for configurable systems', *Empirical Software Engineering* **23**(3), 1826–1867. MAG ID: 2769879317.

Ha, H., Ha, H. & Zhang, H. (2019*a*), 'DeepPerf: performance prediction for configurable software with deep sparse neural network', pp. 1095–1106. MAG ID: 2954141573.

Ha, H., Ha, H. & Zhang, H. (2019*b*), 'Performance-Influence Model for Highly Configurable Software with Fourier Learning and Lasso Regression', pp. 470–480. MAG ID: 2992673542.

Innovations, R.-T. (n.d.), 'Connext Product Suite for Intelligent Distributed Systems'.

URL: *https://www.rti.com/products*

Macenski, S., Foote, T., Gerkey, B., Lalancette, C. & Woodall, W. (2022), 'Robot operating system 2: Design, architecture, and uses in the wild', *Science Robotics* **7**(66).

URL: *https://www.science.org/doi/abs/10.1126/scirobotics.abm6074*

Rausch, M. & Sanders, W. H. (2020), Sensitivity Analysis and Uncertainty Quantification of State-Based Discrete-Event Simulation Models Through a Stacked Ensemble of Metamodels, *in* M. Gribaudo, D. N. Jansen & A. Remke, eds, 'Quantitative Evaluation of Systems', Vol. 12289, Springer International Publishing, Cham, pp. 276–293. Series Title: Lecture Notes in Computer Science.

URL: *http://link.springer.com/10.1007/978-3-030-59854-9*$_2$0

RTI Perftest (2023). original-date: 2016-08-23T20:27:26Z.

URL: *https://github.com/rticommunity/rtiperftest*

Sachs, K., Kounev, S. & Buchmann, A. (2013), 'Performance modeling and analysis of message-oriented event-driven systems', *Software & Systems Modeling* **12**(4), 705–729.

URL: *http://link.springer.com/10.1007/s10270-012-0228-1*

Shu, Y., Sui, Y., Hongyu Zhang, Zhang, H. & Xu, G. (2020), 'Perf-AL: Performance

Prediction for Configurable Software through Adversarial Learning'. MAG ID: 3094340071.

Siegmund, N., Grebhahn, A., Apel, S. & Kästner, C. (2015), Performance-influence models for highly configurable systems, *in* 'Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering', ACM, Bergamo Italy, pp. 284–294.

URL: *https://dl.acm.org/doi/10.1145/2786805.2786845*

sklearn.preprocessing.MinMaxScaler (n.d.).

URL: *https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html*

sklearn.preprocessing.RobustScaler (n.d.).

URL: *https://scikit-learn/stable/modules/generated/sklearn.preprocessing.RobustScaler.html*

sklearn.preprocessing.StandardScaler (n.d.).

URL: *https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html*

Zhang, Y., Guo, J., Blais, E. & Czarnecki, K. (2015), Performance Prediction of Configurable Software Systems by Fourier Learning (T), *in* '2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)', IEEE, Lincoln, NE, USA, pp. 365–373.

URL: *http://ieeexplore.ieee.org/document/7372025/*