# Utilizing Machine Learning as a Prediction Scheme for Network Performance Metrics of Self-Clocked Congestion Control Algorithm

Ahmed Samir Jagmagji [1,2,*], Haider Dhia Zubaydi [1,**], Sándor Molnár [1,†], and Mahmood Alzubaidi [3]

*Abstract*—**Congestion Control (CC) is a fundamental mechanism to achieve effective and equitable sharing of network facilities. As future networks evolve towards more complex paradigms, traditional CC methods are required to become more powerful and reliable. On the other hand, Machine Learning (ML) has become increasingly popular for solving challenging and sophisticated problems, and scientists have started to turn their interest from rule-based approaches to ML-based methods. This paper employs machine learning models to construct a performance evaluation scheme to predict network metrics for the Self-Clocked Rate Adaptation for Multimedia (SCReAM) algorithm. It uses a rigorous data preprocessing pipeline and a systematic application of ML methods to enhance the performance of the regression model for SCReAM's performance metrics. Also, we constructed a dataset that provides SCReAM's input parameters and output metrics, such as network queue delay, smoothed Round Trip Time (sRTT), and network throughput. Each prediction process has several phases: choosing the best initial regressor model, hyperparameter tuning, ensemble learning, stacking regressors, and utilizing the holdout data. Each model's performance was evaluated through various regression metrics; this study will mainly focus on the coefficient of determination (R2) score. The improvement between the initial best-selected model and the final improved model determined that we were able to increase R2 up to 96.64% for network throughput, 99.4% for network queue delay, and 100% for sRTT.**

*Index Terms*—**Congestion control, machine learning, optimization, prediction, SCReAM.**

## I. Introduction

Modern communication technology comprises a diverse range of services, incorporating mixed network infrastructures that employ a combination of wired, wireless, and satellite connections. Moreover, the features of these network environments depend on many limitations, such as network traffic, link capacity, and user behavior, which means that more accurate estimates are needed. Hence, it can be argued that ML is an unambiguous approach to improving our understanding of network behavior and facilitating the development of appropriate solutions. The reasons behind such an argument come from many features, such as predictive capabilities to anticipate congestion patterns, optimization through learning historical data to manage network traffic efficiently and adaptability by offering dynamic and flexible solutions to real-time changes.

Network congestion arises when the network's capacity is insufficient to accommodate excessive traffic, resulting in increased response time or, in more severe instances, network failure [1]. Therefore, it is essential to provide further consideration to the significant consequences caused by network congestion. Also, there is a notable rise in media traffic, particularly in the audiovisual domain. This can be attributed to the growth of networking applications that have been built on the structure of the transport layer, such as Voice Over IP (VoIP) and Video on Demand (VoD) [2].

Researchers have proposed learning-based CC approaches to address the previously described issues. These techniques encompass Reinforcement Learning (RL), supervised learning, and unsupervised learning techniques. RL has been demonstrated to have several advantages in effectively addressing the issue of realistic congestion in networks that exhibit dynamic and complex state spaces [3]. Hence, it may be argued that RL approaches offer advantages in congestion control due to their enhanced capacity for online learning [4]. Offline learning is appropriate for situations where assuming that others' behavior will converge and remain relatively stable is essential. In contrast, online learning facilitates a more interactive and dynamic exchange between individuals or groups striving to achieve shared objectives under optimal circumstances. Implementing ML as a networking solution is increasingly becoming possible [5].

This paper aims to enhance the efficiency of the regression model utilized as a performance evaluation scheme developed through machine learning. The purpose is to estimate crucial network metrics for SCReAM. This will be accomplished by implementing a rigorous data preprocessing pipeline and systematically applying machine learning techniques. Furthermore, the proposed scheme can be used to replace the execution of SCReAM without requiring SCReAM environment, thus reducing the resource requirements by mitigating the need to perform measurements in the live network. To implement this method, the dataset was generated from SCReAM and utilized as input for the regression model. Simultaneously, the output will be similar to the initial SCReAM. Since this work

[1] A. S. Jagmagji, H.D. Zubaydi, and S. Molnár is with the Department of Telecommunications and Media Informatics, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary. E-mail: *(ahmedsa@tmit.bme.hu), **(haider.zubaydi@tmit.bme.hu), †(molnar@tmit.bme.hu).
[2] College of Engineering, University of Mosul, Mosul, Iraq.
[3] M. Alzubaidi is with the Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khalifa University, Qatar Foundation, Doha, Qatar. (E-mail: malzubaidi@hbku.edu.qa)

DOI: 10.36244/ICJ.2024.3.1

utilized the generated dataset, it falls within the supervised learning scheme. The primary objective of this work is to concentrate on QoS measures, whereas the evaluation of QoE falls outside the scope of our studies.

This paper is organized as follows: Section II introduces the related works that proposed ML congestion control approaches to handle congestion control. Section III describes the methodology of our proposed model, including data preprocessing, model selection, tuning, improvement, and evaluation. Section IV presents the results obtained from our experiments for predicting network queue delay, sRTT, and network throughput. Finally, this paper is concluded in Section V.

## II. RELATED WORK

The primary challenge related to congestion control resides in determining the appropriate mechanism and timing for data transmission. Researchers have successfully employed ML techniques to devise robust methodologies for addressing dynamic scenarios within computer networks. Current machine learning research utilizes three main categories: supervised, unsupervised, and reinforcement learning. This section covers research efforts that contributed to managing and predicting network congestion by utilizing machine learning schemes.

A general overview has been introduced in [6] that discusses how to revolutionize CC algorithms using various ML techniques. This research discusses various ML mechanisms, such as supervised learning and RL, that utilize real-time modification of control parameters and predict network traffic to eliminate network congestion. Furthermore, the authors highlighted the possibility of managing traditional congestion control challenges such as packet loss and network latency.

Machine Learning Aided Congestion Control (MLACC) [7] is a novel approach that integrates traditional CC protocols with ML to address efficiency and fairness issues. CC parameters are dynamically adjusted based on network conditions using an RL-based framework. The results indicate that MLACC surpasses traditional congestion control approaches, balancing fair resource allocation among users and high throughput.

Another work that focuses on utilizing ML to improve the fairness of TCP congestion control algorithms is introduced in [8]. The authors argue that unfair bandwidth distribution among users occurs in traditional TCP mechanisms. Thus, an ML-based approach is proposed to ensure a fair distribution of network resources by dynamically adjusting TCP parameters. The effectiveness of this approach in various network scenarios is demonstrated in experimental results.

As discussed in [9], ML can also be employed in other network paradigms, such as Software-Defined Networking (SDN), to enhance congestion control. The proposed framework allows SDN controllers to empower the network with real-time traffic management decisions by leveraging machine learning models. This work reveals that combining ML's predicting capabilities with SDN's scalability and flexibility features enhances congestion management and network performance.

The authors in [10] argue the possibility of mastering congestion control by enabling computers to learn from heuristic designs. A detailed analysis of how heuristic-based congestion control algorithms can be employed to train ML models to produce more adaptive and robust control schemes. This study shows that ML can apply heuristics in different network conditions to improve performance.

An ablation study on leveraging Deep Reinforcement Learning (DRL) for congestion control is presented in [11]. Various components of the DRL model are systematically evaluated based on its performance. This study provides guidelines to optimize the performance of such models by identifying the most critical elements affecting DRL in managing network congestion.

A comprehensive troubleshooting solution using ML for traffic congestion control is proposed in [12]. The developed framework can suggest corrective actions by identifying the leading causes of congestion. This framework enables proactive traffic flow management by predicting potential congestion issues and analyzing traffic patterns by leveraging various ML models.

Two congestion control systems, Aurora and Custard, employ DRL techniques described in references [13] and [14]. The idea behind these plans is to use DRL to develop a way to map real-world network data to find the best transmission rate. DRL is a contemporary ML technique utilized to evaluate network conditions. This process involves multiple stages: agent training, procedure learning, and enhancing behavior through continuous environmental interaction. The network condition is characterized by the bandwidth, RTT, and loss rate, which serve as input parameters for the network agent.

In [15], the authors introduced a loss predictor that utilizes random forest, a supervised learning method, to estimate the likelihood of packet loss resulting from congestion. This methodology can predict and mitigate occurrences of packet loss, diminish the frequency of rate reduction during transmission, and attain enhanced throughput. These studies employ machine learning techniques to estimate congestion-related metrics based on passive data. Such approaches demonstrate significant potential for predicting parameter values.

For online learning schemes, the authors in [16] employed a trial-and-error methodology to determine the optimal transmitting rate. This research highlights the implementation of replacing the absolute value of RTT with a rise in RTT, as well as ensuring the fulfillment of desired network characteristics, such as fair convergence. The authors focused on investigating the impact of altering transmission rates on optimizing the environment's performance without relying on prior knowledge. Even though online learning can quickly adjust to changes in the network, its performance may sometimes drop because of its limits, which could lead to getting stuck in a local optimal [17]. Acknowledging that online learning typically involves a significant amount of time for routing convergence [18] is essential. This refers to the duration required for all routers within the network to reach a consensus on the current topology.

Although the presented works provide significant insights into various ML applications for congestion control, our work contributes by introducing a novel approach that includes the newly constructed dataset that allows predicting SCReAM's

TABLE I
COMPARISON OF RELATED WORKS ON CC WITH ML

| Reference | Focus | Key Techniques | Major Contribution | Application | Results |
|---|---|---|---|---|---|
| [6] | Adjustment and prediction of network traffic | RL, supervised learning | Enhanced packet loss management and latency | Network traffic | Adjusting control parameters in real-time |
| [7] | CC efficiency and fairness | RL | Balanced resource allocation and network throughput | Network traffic | Outperforms traditional methods |
| [8] | Fairness of bandwidth distribution | ML-based TCP parameter adjustment | Fairness of resource distribution | TCP networks | Proved effectiveness in various scenarios |
| [9] | Integration of ML and SDN | SDN controllers, ML models | Real-time traffic management | Network traffic | Improved network management and performance |
| [10] | Heuristics CC learning | Heuristic-based ML | Adaptive and robust control mechanisms | Network traffic | Improved performance in different conditions |
| [11] | DRL model evaluation | DRL | Optimization of DRL models | Network traffic | Improved performance through the identified critical components |
| [12] | Identifying traffic issues through a diagnostic framework | ML models | Proactive traffic flow management | Urban traffic | Ability to predict and fix congestion problems |
| [13] | Employing DRL to address internet CC | DRL | Demonstrates RL's ability to outperform state-of-the-art methods and presents OpenAI Gym as a test suite | Internet CC | Captures complex data traffic patterns, highlights challenges in safety, generalization, and fairness |
| [14] | Leveraging DRL for CC | DRL | Indicates that RL can efficiently improve resource allocation and manage data rates of network traffic | Traffic management | Addresses dynamic network environment issues, exceeds the performance of existing algorithms |
| [15] | Enhancing TCP CC through ML | ML | Dynamically adjusting parameters to improve TCP performance | TCP networks | Better performance compared to traditional methods |
| [16] | Online learning approach for CC | Online learning | adapting to network conditions by introducing PCC Vivace | Network traffic | Achieved high adaptability and performance |
| **Our paper** | Enhancing the performance of SCReAM and predicting its parameters | Data preprocessing, regression model, supervised learning | Predicts crucial network metrics, replaces SCReAM execution | SCReAM algorithm | Accurate prediction capabilities, reduces resource requirements |

parameters and reduces resource requirements. By leveraging supervised learning mechanisms, our work can enhance SCReAM's performance in diverse network scenarios. Table I presents a detailed comparison of related works and ours.

In this study, we will focus on supervised (offline) learning because online learning demands enormous training data sets to obtain satisfactory performance [19]. In addition, we have a fixed training dataset that does not require any real-time interaction with the environment, which is required by online learning [20]. The advantages include the concept of linear regression, which is transparent and straightforward. Normalization can also be employed as a technique to mitigate the issue of overfitting. Furthermore, stochastic gradient descent facilitates the seamless updating of linear models with incoming data. Moreover, utilizing widely recognized and appropriate categorized input data in supervised learning yields significantly higher reliability and precision than unsupervised learning. The utilization of labels can enhance performance on specific tasks. Proficient in identifying solutions for a diverse range of linear and non-linear problems, including but not limited to classification, robotics, prediction, and factory control.

## III. METHODOLOGY

Our methodology focuses on developing a regression model to predict three target variables: network queue delay, sRTT, and network bandwidth (RateTransmitted). Network queue delay is the estimated queue delay of the entire network calculated by SCReAM. In contrast, TCP endeavors to predict future round-trip times by sampling packet behavior across a connection and averaging the results, referred to as the sRTT [21]. Our methodology includes a rigorous data preprocessing pipeline and a systematic application of machine learning techniques. The goal was to improve the performance of each regression model progressively. The methodology employed in this study is depicted in Figure 1. It comprises several key components, including SCReAM, dataset generation, data pre-processing, model selection, and tuning, model improvement, and model evaluation.

### A. The SCReAM Algorithm

SCReAM was first introduced in 2014 and subsequently standardized in 2017 [22][23]. SCReAM is a congestion control algorithm that combines loss-based and delay-based techniques to create a hybrid model for managing network congestion in LTE networks. Packet conservation manages network congestion by dynamically adapting network parameters, such as transmission rate and queuing time. SCReAM adapts to variations in network conditions by adjusting its network parameters to achieve optimal performance, as determined by the assessments. Moreover, it mitigates the variations in short-term latency by employing a practical algorithm for calculating the congestion window. Additionally, the inclusion of the self-clocking feature contributes to the achievement of shorter time scale operation, hence enhancing its overall usefulness. Considering its better performance than alternative delay-based algorithms, we have selected SCReAM as the foundation for our experimental, evaluative, and analytical research [24][25].

While the initial design of SCReAM focused on its application in WebRTC, it demonstrates the potential to be utilized in several applications that require RTP streams. SCReAM is the foundation for the rate adaptation notion and other strategies that have evolved from TCP-friendly window-based and LEDBAT protocols [26]. The packet conservation principle is also incorporated into SCReAM, a crucial and fundamental concept in minimizing network congestion [27].

The critical elements of SCReAM architecture are network congestion control, media rate control, and sender transmission control, depicted in Figure 2. The sender comprises more elements, namely the UDP socket and RTP packet queue. On the other hand, the receiver consists of an RTP payload decapsulator, a de-jitter buffer (which may be optional), and a video decoder. Given that the essential SCReAM congestion management algorithm's functionalities are executed on the transmitting end, the primary goal is to illustrate its main components.

SCReAM is considered more appropriate than rate-based algorithms since it incorporates a self-clocking concept. This design feature enables the algorithm to operate within shorter intervals, precisely one round-trip time (RTT). Nevertheless, the architecture of SCReAM is characterized by its complexity due to sophisticated documentation and code, leading researchers to be reluctant to dive into or pursue studies in it. Furthermore, SCReAM incorporates numerous parameters assigned with specified values. Therefore, this study focuses on identifying and examining the key variables, which will be discussed in the subsequent part.

SCReAM can be implemented using two approaches: one involves utilizing a test application based on Windows and Visual Studio software. In contrast, the other involves using a Linux-based BW test application. The initial methodology of SCReAM involves a single transmitter and receiver built-in C++. Various auxiliary classes are employed: NetQueue, Video Encoder, and RTPQueue. Furthermore, the coordinator code, called scream_v_a, is utilized when combined with these components. The coordinator code manages and integrates multiple codes into an integrated framework. The initial methodology was employed for our experimental procedures, whereas the second strategy was utilized for the initial evaluation of SCReAM.

### B. Dataset Generation

The subsequent component of this phase is identifying and selecting potential parameters to analyze and optimize the performance of SCReAM. A comprehensive investigation was conducted on several aspects, relying on the specifications outlined in RFC8298 [23] and the coding process. Afterward, a range of parameters were initially examined. Subsequently, a comprehensive analysis is conducted on each parameter to ascertain its potential impact, ensuring its incorporation into our experimental procedures. Moreover, several parameters have yet to be considered due to their negligible impact on the results.

To ensure the effective execution of our experiments, it is essential to establish a comprehensive set of values for each parameter, thereby facilitating a clear understanding of the impact of each parameter on the overall performance. The experimentation commenced by employing a diverse set of values for each parameter. After performing ≈40,000 experiments, a range narrowing occurs exclusively in instances with negligible performance alteration. Therefore, the margin values that yield identical performance measures are removed. Our objective was to establish the default value of each parameter as the median value within the range to facilitate an accurate understanding of performance variations before and after the modifications. It is crucial to note that the maximum target bitrate initially had a default value of 20 Mbps in the algorithm. However, due to the high-speed nature of our experimental implementation, we ultimately adjusted the default value to 100 Mbps. The following parameters have been determined for the construction of our dataset:

- P1: Target value for the minimum queue delay ($QD_{low}$)
- P2: Threshold for the detection of incipient congestion ($QD_{th}$)
- P3: Maximum segment size (RTP packet size) (MSS)
- P4: Interval between media bitrate adjustments (RAI)
- P5: Minimum target bitrate in Mbps (bits per second) ($TB_{min}$)
- P6: Maximum allowed rate increase speed (RUS)
- P7: Guard factor against early congestion onset (PCG)
- P8: Guard factor against RTP queue buildup (QSF)
- P9: RTP queue delay threshold for a target rate reduction ($RQ_{th}$)
- P10: Scale factor for target rate when RTP queue delay threshold exceeds P9 (TRS)

The dataset construction procedure is as follows: Initially, a counter is established to ascertain the required number of

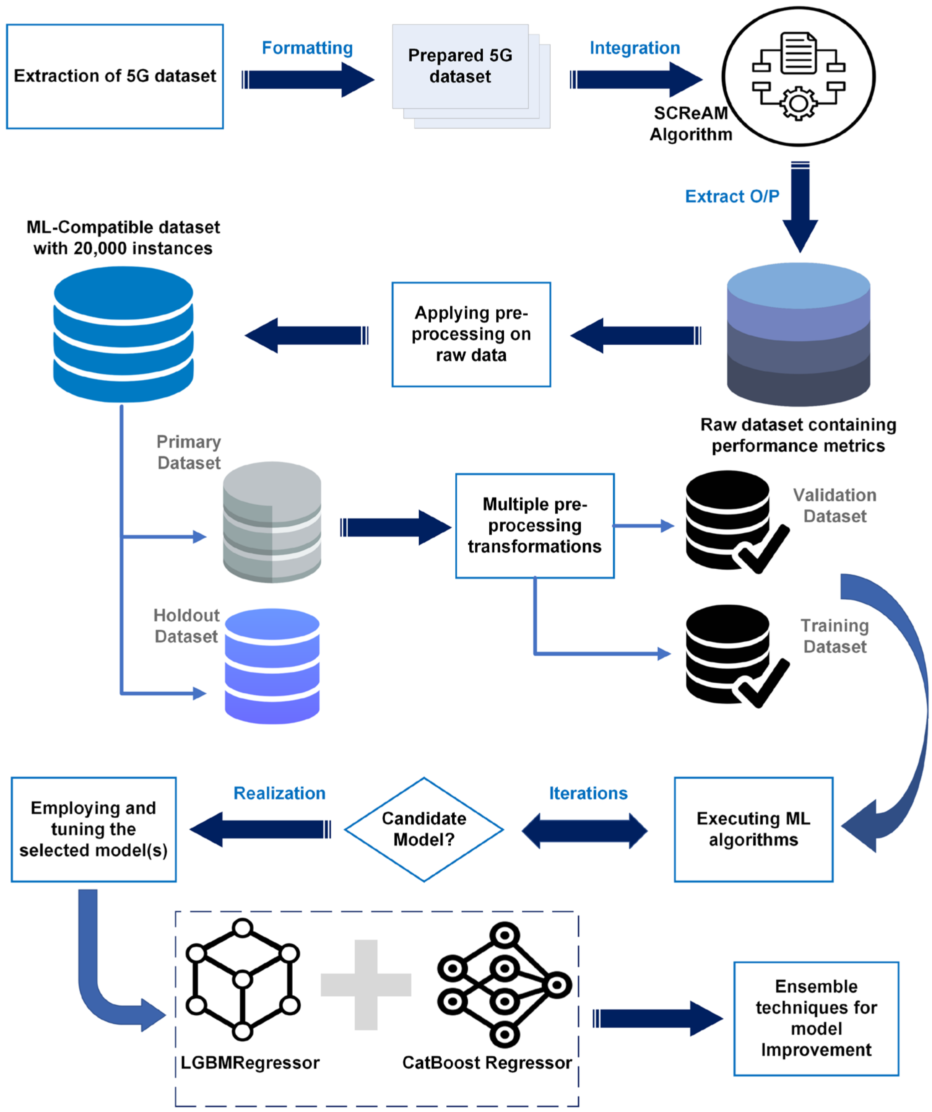


Fig. 1: Workflow of the proposed methodology.

experiments for the ML model, precisely 20,000 experiments. The algorithm that is devised tends to produce a numerical value for each parameter that falls within a predetermined range. During each experimental assessment, SCReAM is executed for 100 seconds, during which it gathers a total of 2000 data samples of network queue delay, sRTT, and network throughput. Subsequently, the mean is computed as depicted in Equation 1.

$$\overline{x} = \frac{1}{2000} \cdot \sum_{i=1}^{2000} x_{\mathrm{i}}, \tag{1}$$

Where $\overline{x}$ represents the metric average, and $x_{\mathrm{i}}$ is the value of one metric. Additional details regarding the dataset are demonstrated in the following subsection.

### C. Data Preprocessing

Data preprocessing is a crucial component of the machine learning pipeline since it involves cleaning and transforming raw data into an understandable structure. This rigorous preprocessing approach forms the foundation for the subsequent data analysis and modeling, ensuring the results are reliable and replicable. The details of each model's training, fine-tuning, and final prediction for each target variable are discussed in the subsequent sections. The dataset included in our investigation initially consisted of 20,000 instances characterized by 10 attributes. Our objective was to construct predictive models for three target variables: network queue delay, sRTT, and throughput. A systematic methodology was employed for data preprocessing. The dataset was initially partitioned into two segments: a primary dataset consisting of 16,000 instances employed for model development and a holdout dataset comprising 4,000 instances reserved for the

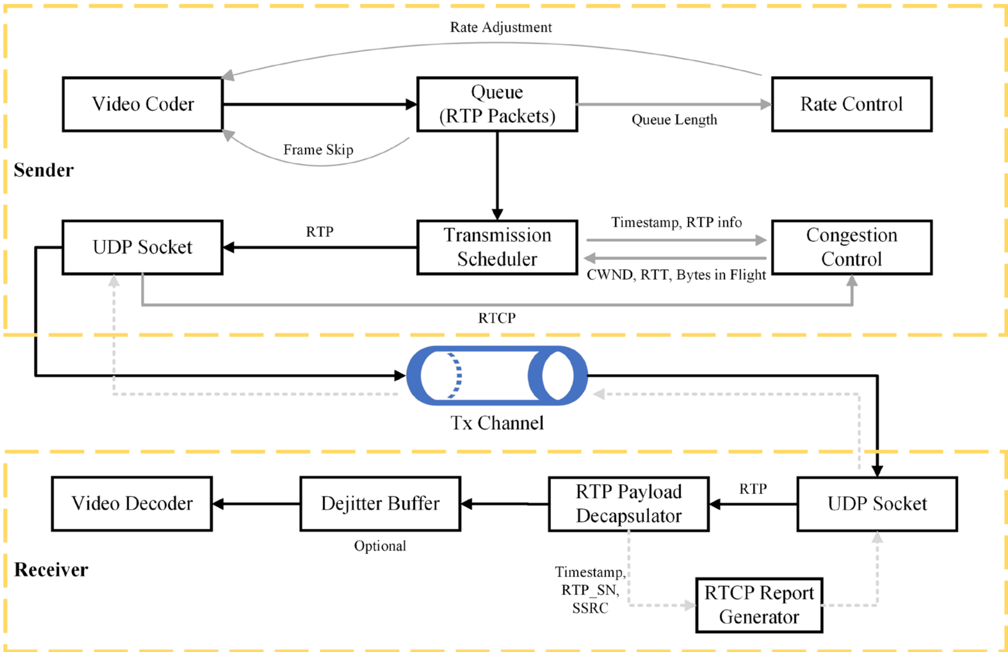


Fig. 2: SCReAM architecture (a single media source design) [21].

TABLE II
A SAMPLE FROM OUR FEATURE SET

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | Delay | sRTT | BW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **N** | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K | 20K |
| **Missing** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Mean** | 0.105 | 0.255 | 1198 | 1810 | 5.51 | 11.0 | 0.453 | 0.263 | 0.161 | 0.849 | 35.2 | 6.93 | 31.5 |
| **Median** | 0.105 | 0.257 | 1199 | 1810 | 6.0 | 11.1 | 0.45 | 0.263 | 0.161 | 0.849 | 31.4 | 6.63 | 31.1 |
| **S. Dev.** | 0.0551 | 0.142 | 174 | 851 | 2.63 | 5.18 | 0.26 | 0.137 | 0.0795 | 0.0862 | 19.2 | 1.47 | 8.54 |
| **Max.** | 0.01 | 0.01 | 900 | 327 | 1 | 2.0 | 0.0 | 0.025 | 0.025 | 0.7 | 1.92 | 4.42 | 5.92 |
| **Min.** | 0.2 | 0.5 | 1499 | 3277 | 10 | 20.0 | 0.9 | 0.5 | 0.3 | 1.00 | 201 | 22.3 | 56.8 |

final testing of the trained and validated model.

The primary dataset underwent several preprocessing transformations to adequately prepare it for model training. The primary dataset was divided into two subsets: a training set comprising 80% (12,800 instances) and a validation set including 20% (3,200 instances). Notwithstanding the transformations, the overall shape of the data remained consistent with its initial form, suggesting that no instances were removed during this stage. The primary dataset comprised ten numerical attributes and one category attribute. The aforementioned properties were assessed for the possibility of missing data. In order to address any missing data, we employed imputation techniques, specifically mean imputation for numerical features and mode imputation for categorical features. Normalization was conducted to standardize the numerical feature values to a uniform range. Min-max normalization was utilized to rescale the features from 0 to 1 to ensure that the scale of each feature is aligned (i.e., all features contribute equally to the analysis).

The Min-Max Normalization, denoted by $x_{\text{scaled}}$, can be calculated through the following Equation:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{2}$$

Where $x$ is the number before normalization, the $x_{\text{min}}$ is the smallest number in the dataset, and the $x_{\text{max}}$ is the largest number in the dataset.

The modeling process employed a 10-fold cross-validation approach generated through the K-Fold algorithm during the training and fine-tuning stages, where k denotes the number of groups into which a particular data sample is divided. Cross-validation is a resampling technique utilized to assess machine learning models on a constrained data sample by dividing it into training and testing sets to train and evaluate the data. This technique helps provide a more robust estimation of the model's performance by iteratively validating the model against different subsets of data.

We made further efforts to enhance computational efficiency during model training by utilizing all available CPU cores and, where possible, GPU resources. While we meticulously recorded all the processing steps, the experiment's logs were not saved for review. After training and validation, the final model was tested on the holdout dataset. This procedure ensures the model's performance is evaluated on unseen data, offering a more accurate predictive power evaluation. A sample from our feature set is displayed in Table II, including SCReAM parameters as input features for our model and the network metrics (target variables) for prediction. Table II also determines the number of samples N (collected data set results), missing samples, mean, median, standard deviation, minimum, and maximum values.

### D. Model Selection and Tuning

Each target variable was assigned an initial model based on previous performance considerations. The selected models were LGBM Regressor [28] for predicting network throughput, CatBoost Regressor [29] for network queue delay and sRTT. Then, each selected model is fine-tuned using Optuna, a hyperparameter optimization framework [30]. The goal was to maximize the $R^2$ score, a standard metric for regression problems, which measures the proportion of the variance in the dependent variable that is predictable from the independent variable(s). The $R^2$ score is a critical metric that is used to evaluate the performance of a regression-based machine learning model. The coefficient of determination works by measuring the amount of variance in the predictions explained by the dataset. On average, using $R^2$ in the evaluation of the ML model is one of the most effective techniques that provides powerful results [31].

$$R^2 = 1 - \frac{\text{SSR}}{\text{SST}} = 1 - \frac{\sum_i (x_i - \widehat{x}_i)^2}{\sum_i (x_i - \overline{x})^2}{}' \tag{3}$$

Where SSR is the sum squared regression, SST is the total sum of squares, and $\widehat{x}_i$ is the predicted value for $x_i$. As a percentage, it will take values between 0 and 1.

LightGBM, also known as Light Gradient Boosting Machine, gradient boosting system created by Microsoft. The LGBM Regressor is a specialized version of a gradient boosting model tailored explicitly for regression applications. This model is classified under ensemble learning techniques, notably boosting, where multiple weak learners (usually decision trees) are combined sequentially to form a powerful prediction model. It reduces the total error by optimizing based on the residuals.

Compared with other boosting algorithms, LightGBM represents one of the fastest and the most efficient algorithms as it uses a histogram-based dependent method to deal with large datasets quickly and as low as possible regarding memory space, which makes it an optimal solution for scenarios with large amounts of data.

An essential feature of LightGBM is its capability to minimize overfitting using the L1 and L2 regularization techniques integrated with the algorithm. In addition, the growing tree leaf-wise method is used as a tree-based learning algorithm to enhance accuracy, which helps mitigate loss efficiently and produce a more accurate LightGBM model.

Using an LGBM Regressor for prediction involves several steps, including data preparation, model training using a training set, and hyperparameters fine-tuning using cross-validation. Then, the trained model is optimized to start predicting the output using the unseen new data. To optimize the performance and avoid overfitting across different tasks, understanding the features used in LightGMB, such as its regularization methods and unique tree-building technique, is essential.

The most crucial benefit of LightGBM is that it efficiently handles the categorical features that will mitigate the time and effort needed for excessive preprocessing. LightGBM can be used directly with categorical data, which is different from the traditional techniques that need to be encoded and transformed into other forms before dealing with the categorical data. In addition, it can implement different methods, such as order boosting and categorical feature-numerical value transformation, resulting in enhanced performance and reduced human interaction.

LightGBM consistently generates multiple decision trees, and each tree is taught to update the previous one's error, leading to increased overall accuracy. The leaf-wise tree growth method is an essential feature in LightGBM that depends on minimizing the most significant loss in leaves, resulting in constructing deeper trees with fewer and faster leaves and precise outcomes.

CatBoost has built-in support for categorical variables, which provides a considerable advantage over models that require specific handling, such as one-hot encoding, for these types of features. CatBoost incorporates inherent mechanisms, such as depth restrictions and learning rate shrinkage, to mitigate the issue of overfitting. The framework additionally provides cross-validation techniques for optimizing hyperparameters and assessing model performance. Moreover, CatBoost effectively manages missing data, reducing preprocessing procedures.

CatBoost's design, which prioritizes efficiency and scalability, makes it well-suited for handling massive datasets. Utilizing the CatBoost Regressor for prediction generally includes preparing the data, training the model, modifying the hyperparameters, and generating predictions on unique or unobserved data. The direct handling of categorical variables, the emphasis on preventing overfitting, and the user-friendly approach to missing data makes CatBoost an attractive option for regression problems, mainly when working with heterogeneous datasets that include numerical and categorical features.

### E. Model Improvement

After the fine-tuning stage, we employed ensemble techniques to enhance the performance of each model further. Bagging methods were initially used, with results indicating slight improvements in the $R^2$ score. Subsequently, a stacking regressor was used to combine the predictions of multiple estimators to generate a final model. Stacked Regressions is a technique that creates linear combinations of various predictors

TABLE III
LIGHTGBM PARAMETERS SPECIFICATIONS – LGBM REGRESSOR

| Specifications | Value |
|---|---|
| **LGBM bagging fraction** | 0.8088 |
| **LGBM bagging freq.** | 7 |
| **LGBM device** | gpu |
| **LGBM feature fraction** | 0.6502 |
| **LGBM learning rate** | 0.0512 |
| **LGBM min. child samples** | 52 |
| **LGBM Regressor min. split gain** | 0.5451 |
| **LGBM n. estimators** | 185 |
| **LGBM n. leaves** | 32 |
| **LGBM random state** | 123 |
| **LGBM Regressor reg. alpha** | 5.8939e-07 |
| **LGBM reg. lambda** | 2.2148e-07 |

TABLE IV
LIGHTGBM PARAMETERS SPECIFICATIONS –
STACKING REGRESSOR (NETWORK THROUGHPUT)

| Specifications | Value |
|---|---|
| **Stacking Regressor cv** | 5 |
| **Stacking Regressor estimators** | LGBM |
| **LGBM device** | gpu |
| **LGBM random state** | 123 |
| **CatBoost Regressor** | catboost.core.CatBoost |
| **CatBoost Regressor object** | 0x0000017C68BE5910 |
| **Gradient Boosting Regressor** | GradientBoosting Regressor |
| **GradientBoosting random state** | 123 |
| **Stacking Regressor final estimator** | LinearRegression |
| **LinearRegression n. jobs** | -1 |
| **Stacking Regressor n. jobs** | 1 |
| **Stacking Regressor passthrough** | True |

to enhance prediction accuracy [3]. The three best models for throughput were the LGBM Regressor, CatBoost Regressor, and GradientBoosting Regressor. The best three models for Network Queue Delay and sRTT were the CatBoost Regressor, LGBM Regressor, and ExtraTrees Regressor. The stacking regressor models were then validated and evaluated using the holdout dataset, which offered a more realistic evaluation of the model's predictive performance, as it had yet to be exposed to this data during training.

### *F. Model Configurations*

This part describes the configurations used to optimize the model, including Optuna, LightGBM parameters specifications, and the Stacking Regressor. A *Hyperparameter* is an exterior configuration parameter engineers use to control machine learning training. The number of nodes and layers in a neural network and the number of branches in a decision tree are illustrative examples of hyperparameters. Hyperparameters define essential model properties such as architecture, learning speed, and ML model complexity. Training a machine learning model using multiple sets of variables, analyzing the performance of each set, and selecting an optimal set that produces the best performance are called Hyperparameter tuning. Ensemble learning integrates multiple machine learning models, known as weak learners, into a single problem. The idea is that combining these weak learners can create strong learners. Stacking regressions is a technique that combines various predictors linearly to enhance the accuracy of predictions [32].

The Optuna configuration settings used in the optimization process and the LightGBM core parameters with their values

TABLE V
STACKING REGRESSOR PARAMETERS SPECIFICATIONS –
STACKING REGRESSOR (NETWORK QUEUE DELAY)

| Specifications | Value |
|---|---|
| **Stacking Regressor cv** | 5 |
| **Stacking Regressor estimators** | CatBoost Regressor |
| **catboost.core.CatBoost Regressor object** | 0x0000017C806D57C0 |
| **Light Gradient Boosting Machine** | LGBM Regressor |
| **LGBM Regressor device** | gpu |
| **LGBM Regressor random state** | 123 |
| **Extra Trees Regressor** | ExtraTrees Regressor |
| **ExtraTrees Regressor n. jobs** | -1 |
| **ExtraTrees Regressor random state** | 123 |
| **Stacking Regressor final estimator** | LinearRegression |
| **LinearRegression n. jobs** | -1 |
| **Stacking Regressor n. jobs** | 1 |
| **Stacking Regressor passthrough** | True |

TABLE VI
STACKING REGRESSOR PARAMETERS SPECIFICATIONS –
STACKING REGRESSOR (SRTT)

| Specifications | Value |
|---|---|
| **Stacking Regressor cv** | 5 |
| **Stacking Regressor estimators** | CatBoost Regressor |
| **catboost.core.CatBoost Regressor object** | 0x0000017CE14A8F40 |
| **Light Gradient Boosting Machine** | LGBM Regressor |
| **LGBM Regressor device** | gpu |
| **LGBM Regressor random state** | 123 |
| **Extra Trees Regressor** | ExtraTrees Regressor |
| **ExtraTrees Regressor n. jobs** | -1 |
| **ExtraTrees Regressor random state** | 123 |
| **Stacking Regressor final estimator** | LinearRegression |
| **LinearRegression n. jobs** | -1 |
| **Stacking Regressor n. jobs** | 1 |
| **Stacking Regressor passthrough** | True |

are described in Table III. The core parameters mentioned are ranking application parameters, including bagging fraction, bagging frequency, processing device type, learning rate logarithmic value, number of boosting estimators, and maximum number of leaves in one tree. There are also several learning control parameters, including minimum child samples per leaf, minimal gain to perform split, and two regularization parameters at the regression analysis level ($\alpha$ and $\lambda$) [33].

We used a stacking regressor to improve the result by stacking the best three regression models (LGBM Regressor, CatBoost Regressor, GradientBoosting Regressor) with the configuration settings for network throughput, network queue delay, and sRTT presented in Tables IV, V, and VI respectively.

Where $cv$ specifies the number of the cross-validation's splitting strategy, random_state specifies the value we set to get the same values in train and test datasets whenever we run the stacking regressor code. Linear regression is the final result estimator, assuming the relation between the input and output variables is linear. The n_jobs=-1 means that all the CPU cores will be used during the simulation while specifying the number; for example, n_jobs=1 will specify the exact number of cores. Finally, the boolean value for the pass-through option shows that when it is set to false, the estimators' predictions will only be used to train the final_estimator. In contrast, true value means the final_estimator is trained on the predictions and the original training data.

### G. Model Evaluation

Each model's performance was evaluated through various metrics, such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), coefficient of determination ($R^2$), Root Mean Squared Logarithmic Error (RMSLE) and Mean Absolute Percentage Error (MAPE). $R^2$ will be the main focus of our discussions. Mean Absolute Error (MAE) is a metric used to measure the average absolute difference between the predicted and actual values. It represents the average magnitude of the errors without considering their direction and is calculated by summing up the absolute differences and dividing by the total number of samples.

RMSE is a commonly used metric to evaluate the performance of a predictive model and measure the square root of the average squared differences between the predicted and actual values. It is calculated by taking the square root of the average of the squared differences between predicted and actual values. RMSLE is a metric commonly used in tasks where the predicted variable and actual values span a wide range and are skewed. It calculates the RMS of the logarithmic differences between the predicted and actual values. It is calculated by taking the square root of the average of the squared logarithmic differences between predicted and actual values.

MAPE is a metric used to measure the average percentage difference between predicted and actual values. MAPE is commonly used in forecasting and demand planning tasks. However, it has some limitations, such as being sensitive to zero values and unbounded, meaning it can produce infinite values. To visualize and better comprehend the performance of each model, we also generated residual and prediction error plots. Additionally, scatter plots were used to compare the actual and predicted values of the target variables.

The methodology applied in this study ensures a robust and comprehensive approach toward model development and evaluation, aiding the reliable and replicable prediction of the target variables.

## IV. Results and Discussion

In this study, we performed rigorous data preprocessing and utilized various machine-learning models to predict three target attributes: network queue delay, sRTT, and network throughput. We employed diverse techniques to enhance the model's prediction quality, including hyperparameter tuning, ensemble learning, and stacking regressors.

As previously mentioned, we calculated five different outputs (MAE, RMSE, $R^2$, RMSLE, and MAPE) values for each metric. Further experiments are performed during four subsequent stages. The reason for involving each stage in this phase is as follows:

- **Fine-tuning with Optuna:** During this stage, hyperparameter optimization occurs. Optuna examines various possible combinations to identify the most suitable set of hyperparameters. This process leads to a better fitting between the model and data, which improves $R^2$.
- **Applying the bagging method:** This method ensures more reliable and stable predictions by reducing variance and overfitting. It calculates the average prediction values of multiple models trained on different training data subsets. As a result, it will increase the model's ability to identify the underlying data patterns, which in turn increases $R^2$.
- **Deployment of stacking regressor:** Multiple base models can be trained simultaneously and later achieve a meta-model that combines the strengths of their predictions, mitigating their weaknesses. The realized meta-model can eventually enhance $R^2$ by capturing more complex relationships by learning to assign proper weights to the model's prediction values.
- **Using holdout data:** The final stage of our improvements handles the unseen data and ensures that the stacking regressor generalizes well to them. Capturing the underlying data distribution is indicated by the model's performance on the holdout data. Better performance leads to higher $R^2$ values. This step realizes the accuracy and robustness of the model.

### A. Summary of Improvements: Tabular Data

#### A.1 Network Throughput

We implemented multiple regression models and applied a comparative analysis to demonstrate the best initial prediction performance. Table VII indicates that LightGBM outperformed all other models. It achieved the highest $R^2$ (0.7805) and lowest MAE, RMSE, RMSLE, and MAPE values. It is expected that LightGBM will outperform other models due to its speed, accuracy, and capability to capture complex data patterns.

A minor improvement in $R^2$ value was observed when fine-tuning the model, as shown in Table VIII, where $R^2$ increased to 0.7814. The improvement process indicates that the hyperparameters are close to optimal values. It is noticed that there are standard deviations for MAE and RMSE, indicating a moderate variability in the folds.

As demonstrated in Table IX, a slight improvement in the $R^2$ value with a performance gain of 0.32%, where the bagging fits several independent models and averaged their predictions to get a lower variance model. By applying the bagging method, we realized a more consistent performance and lower standard deviations across different folds compared to Table VIII.

Subsequently, a stacking regressor was deployed, utilizing the three best regression models (LGBM Regressor, CatBoost Regressor, and GradientBoosting Regressor), resulting in further performance gain of 0.397%, as shown in Table X. This combination produces a linear regression scheme that accurately predicts target variables with approximately low errors across different folds. Based on the standard deviation (std) values across multiple subsets of data, the performance of this model is reasonably stable. Among all tested methods, stacking resulted in the lowest variability, which indicates its efficiency in improving the consistency and accuracy of the model's predictions.

Finally, we utilized the holdout data (4000 rows) to evaluate the performance. As demonstrated in Table XI, which exhibits

TABLE VII
INITIAL PREDICTION OF NETWORK THROUGHPUT

| | Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| **LightGBM** | Light Gradient Boosting Machine | 3.1541 | 3.9967 | 0.7805 | 0.1311 | 0.1074 |
| **CatBoost** | CatBoost Regressor | 3.1763 | 4.0272 | 0.7771 | 0.1317 | 0.1077 |
| **GBR** | Gradient Boosting Regressor | 3.3041 | 4.1592 | 0.7623 | 0.1352 | 0.1122 |
| **RF** | Random Forest Regressor | 3.3348 | 4.1945 | 0.7581 | 0.1383 | 0.1144 |
| **XGBoost** | Extreme Gradient Boosting | 3.3235 | 4.2104 | 0.7564 | 0.1385 | 0.1129 |
| **ET** | Extra Trees Regressor | 3.3771 | 4.2454 | 0.7523 | 0.1391 | 0.1156 |
| **Ada** | AdaBoost Regressor | 3.9693 | 4.8661 | 0.6745 | 0.1629 | 0.1417 |
| **Ridge** | Ridge Regression | 4.2362 | 5.3658 | 0.6040 | 0.1738 | 0.1456 |
| **LR** | Linear Regression | 4.2360 | 5.3658 | 0.6040 | 0.1738 | 0.1456 |
| **BR** | Bayesian Ridge | 4.2361 | 5.3658 | 0.6040 | 0.1738 | 0.1456 |
| **LAR** | Least Angle Regression | 4.2360 | 5.3658 | 0.6040 | 0.1738 | 0.1456 |
| **Huber** | Huber Regressor | 4.2241 | 5.3736 | 0.6029 | 0.1732 | 0.1440 |
| **KNN** | K Neighbors Regressor | 4.4880 | 5.6626 | 0.5591 | 0.1884 | 0.1596 |
| **DT** | Decision Tree Regressor | 4.7010 | 5.9827 | 0.5078 | 0.1960 | 0.1599 |
| **PAR** | Passive Aggressive Regressor | 4.7495 | 5.9758 | 0.5061 | 0.1934 | 0.1678 |
| **OMP** | Orthogonal Matching Pursuit | 5.0106 | 6.3742 | 0.4416 | 0.2058 | 0.1753 |
| **Lasso** | Lasso Regression | 5.8857 | 7.2623 | 0.2755 | 0.2406 | 0.2125 |
| **LLAR** | Lasso Least Angle Regression | 5.8857 | 7.2623 | 0.2755 | 0.2406 | 0.2125 |
| **EN** | Elastic Net | 6.6885 | 8.1526 | 0.0870 | 0.2689 | 0.2426 |
| **Dummy** | Dummy Regressor | 7.0298 | 8.5365 | -0.0011 | 0.2807 | 0.2551 |

TABLE VIII
FINE-TUNING THE MODEL USING OPTUNA HYPERPARAMETER OPTIMIZATION FRAMEWORK (NETWORK THROUGHPUT)

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 3.0699 | 3.8816 | 0.7950 | 0.1261 | 0.1037 |
| **1** | 3.2399 | 4.1004 | 0.7730 | 0.1346 | 0.1103 |
| **2** | 3.1861 | 4.0244 | 0.7806 | 0.1299 | 0.1073 |
| **3** | 3.1018 | 3.9078 | 0.7767 | 0.1305 | 0.1080 |
| **4** | 3.2980 | 4.1863 | 0.7624 | 0.1376 | 0.1110 |
| **5** | 3.0754 | 3.8856 | 0.7850 | 0.1265 | 0.1044 |
| **6** | 3.1419 | 3.9216 | 0.7917 | 0.1282 | 0.1068 |
| **7** | 3.1594 | 4.0331 | 0.7860 | 0.1331 | 0.1082 |
| **8** | 3.0894 | 3.9551 | 0.7800 | 0.1281 | 0.1030 |
| **9** | 3.1522 | 3.9895 | 0.7832 | 0.1334 | 0.1100 |
| **Mean** | 3.1514 | 3.9885 | 0.7814 | 0.1308 | 0.1073 |
| **Std** | 0.0703 | 0.0944 | 0.0089 | 0.0036 | 0.0027 |

TABLE IX
BOOSTING THE MODEL'S PERFORMANCE BY APPLYING THE ENSEMBLE MODEL WITH BAGGING METHOD FOR NETWORK THROUGHPUT

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 3.0789 | 3.8773 | 0.7954 | 0.1263 | 0.1042 |
| **1** | 3.2117 | 4.0543 | 0.7781 | 0.1332 | 0.1095 |
| **2** | 3.1782 | 4.0129 | 0.7819 | 0.1298 | 0.1073 |
| **3** | 3.1114 | 3.9120 | 0.7762 | 0.1311 | 0.1087 |
| **4** | 3.2795 | 4.1621 | 0.7651 | 0.1364 | 0.1105 |
| **5** | 3.0912 | 3.8833 | 0.7852 | 0.1263 | 0.1049 |
| **6** | 3.1455 | 3.9302 | 0.7908 | 0.1283 | 0.1068 |
| **7** | 3.1372 | 3.9782 | 0.7918 | 0.1315 | 0.1078 |
| **8** | 3.1239 | 3.9693 | 0.7784 | 0.1285 | 0.1042 |
| **9** | 3.1449 | 3.9582 | 0.7866 | 0.1324 | 0.1099 |
| **Mean** | 3.1502 | 3.9738 | 0.7830 | 0.1304 | 0.1074 |
| **Std** | 0.0568 | 0.0818 | 0.0085 | 0.0030 | 0.0022 |

a proper approximation to the actual values, we achieved a performance gain of 0.78%. This confirms that the model has robust stability and predictive capabilities and can generalize properly to unseen data, maintaining a high $R^2$ value and low error rates.

Although the percentage of performance gains is relatively small, they reflect a significant performance improvement and impactful enhancement in the model's predictive capabilities.

TABLE X
DEPLOYMENT OF THE STACKING REGRESSOR BY UTILIZING THE THREE BEST REGRESSION MODELS (LGBM REGRESSOR, CATBOOST REGRESSOR, AND GRADIENTBOOSTING REGRESSOR) FOR NETWORK THROUGHPUT

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 3.0672 | 3.8710 | 0.7961 | 0.1260 | 0.1034 |
| **1** | 3.1789 | 4.0474 | 0.7788 | 0.1325 | 0.1077 |
| **2** | 3.1605 | 4.0033 | 0.7829 | 0.1296 | 0.1065 |
| **3** | 3.0487 | 3.8751 | 0.7804 | 0.1295 | 0.1060 |
| **4** | 3.2392 | 4.1453 | 0.7670 | 0.1349 | 0.1084 |
| **5** | 3.1006 | 3.9183 | 0.7813 | 0.1272 | 0.1048 |
| **6** | 3.0906 | 3.8900 | 0.7951 | 0.1266 | 0.1044 |
| **7** | 3.1404 | 3.9928 | 0.7903 | 0.1318 | 0.1075 |
| **8** | 3.1016 | 3.9701 | 0.7783 | 0.1281 | 0.1027 |
| **9** | 3.1259 | 3.9649 | 0.7859 | 0.1321 | 0.1089 |
| **Mean** | 3.1254 | 3.9678 | 0.7836 | 0.1298 | 0.1060 |
| **Std** | 0.0538 | 0.0814 | 0.0082 | 0.0028 | 0.0020 |

TABLE XI
PREDICTION OF NETWORK THROUGHPUT USING THE HOLDOUT DATA

| Fold | Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| **0** | Stacking Regressor | 3.1061 | 3.9364 | 0.7866 | 0.1304 | 0.1060 |

*A.2 Network Queue Delay*

Among twenty models, the CatBoost regressor was the top model that offered the best possible initial performance for predicting the network queue delay regarding $R^2$ (0.6935) as shown in Table XII.

Optimization techniques such as scikit-learn [34], scikit-optimize [35], and optuna [34] were utilized during the fine-tuning process. However, the results in Table XIII show that implementing the mentioned techniques along with the CatBoost regressor results in performance degradation, as indicated by the 2.90% drop in $R^2$. These negative impacts on model performance imply that hyperparameter choices could have generalized better across the cross-validation folds related to many possible problems, such as data variability, unbalanced data, or hyperparameter sensitivity to some values. In addition, the overfitting problems, noise, and minor fluctuations that do not reflect the basic patterns in the data

TABLE XII
PREDICTION OF NETWORK THROUGHPUT USING THE HOLDOUT DATA

| Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **CatBoost Regressor** | 8.0329 | 10.6084 | 0.6935 | 0.3172 | 0.2861 |
| **Light Gradient Boosting Machine** | 8.0742 | 10.6451 | 0.6914 | 0.3198 | 0.2908 |
| **Extra Trees Regressor** | 8.2559 | 10.8737 | 0.6778 | 0.3285 | 0.3038 |
| **Random Forest Regressor** | 8.2610 | 10.8897 | 0.6769 | 0.3273 | 0.3015 |
| **Extreme Gradient Boosting** | 8.5795 | 11.2772 | 0.6536 | 0.3435 | 0.3074 |
| **Gradient Boosting Regressor** | 8.5462 | 11.3707 | 0.6480 | 0.3364 | 0.3057 |
| **K Neighbors Regressor** | 10.0073 | 13.1048 | 0.5321 | 0.3959 | 0.3837 |
| **Ridge Regression** | 10.4444 | 13.7138 | 0.4878 | 0.4348 | 0.3917 |
| **Linear Regression** | 10.4454 | 13.7138 | 0.4878 | 0.4351 | 0.3917 |
| **Bayesian Ridge** | 10.4445 | 13.7138 | 0.4878 | 0.4349 | 0.3917 |
| **Least Angle Regression** | 10.4454 | 13.7138 | 0.4878 | 0.4351 | 0.3917 |
| **Huber Regressor** | 10.3333 | 13.8146 | 0.4804 | 0.4156 | 0.3781 |
| **Passive Aggressive Regressor** | 10.8379 | 14.5355 | 0.4218 | 0.4567 | 0.3740 |
| **AdaBoost Regressor** | 12.3687 | 15.0337 | 0.3839 | 0.4919 | 0.5649 |
| **Lasso Regression** | 11.6777 | 15.7865 | 0.3219 | 0.4498 | 0.4595 |
| **Lasso Least Angle Regression** | 11.6777 | 15.7865 | 0.3219 | 0.4498 | 0.4595 |
| **Decision Tree Regressor** | 11.7606 | 15.7883 | 0.3200 | 0.4562 | 0.4032 |
| **Orthogonal Matching Pursuit** | 12.3014 | 16.6105 | 0.2490 | 0.4713 | 0.4855 |
| **Elastic Net** | 13.7800 | 18.2671 | 0.0921 | 0.5235 | 0.5597 |
| **Dummy Regressor** | 14.5472 | 19.1787 | -0.0009 | 0.5480 | 0.5923 |

TABLE XIII
FINE-TUNING THE CATBOOST REGRESSOR USING THE SCIKIT-LEARN, SCIKIT-OPTIMIZE, AND OPTUNA HYPERPARAMETER OPTIMIZATION TECHNIQUES FOR NETWORK QUEUE DELAY

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 8.4458 | 11.1594 | 0.6766 | 0.3340 | 0.2995 |
| **1** | 7.9667 | 10.4933 | 0.7068 | 0.3215 | 0.2886 |
| **2** | 8.2689 | 10.8947 | 0.6689 | 0.3198 | 0.2937 |
| **3** | 7.9070 | 10.1855 | 0.6783 | 0.3144 | 0.2839 |
| **4** | 8.1358 | 10.9003 | 0.6818 | 0.3199 | 0.2854 |
| **5** | 8.6920 | 11.6559 | 0.6617 | 0.3288 | 0.2986 |
| **6** | 8.5120 | 11.3981 | 0.6458 | 0.3434 | 0.3068 |
| **7** | 8.1917 | 10.7129 | 0.6848 | 0.3288 | 0.3036 |
| **8** | 8.5099 | 11.2010 | 0.6709 | 0.3272 | 0.2854 |
| **9** | 8.3884 | 10.9274 | 0.6579 | 0.3382 | 0.3154 |
| **Mean** | 8.3018 | 10.9529 | 0.6734 | 0.3276 | 0.2961 |
| **Std** | 0.2402 | 0.4076 | 0.0159 | 0.0086 | 0.01 |

TABLE XIV
BOOSTING THE MODEL'S PERFORMANCE BY APPLYING THE ENSEMBLE MODEL WITH BAGGING METHOD FOR NETWORK QUEUE DELAY

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 8.1623 | 10.7612 | 0.6993 | 0.3182 | 0.2870 |
| **1** | 7.6904 | 10.1587 | 0.7252 | 0.3126 | 0.2817 |
| **2** | 8.0452 | 10.5875 | 0.6873 | 0.3149 | 0.2871 |
| **3** | 7.6154 | 9.8059 | 0.7018 | 0.3024 | 0.2745 |
| **4** | 7.9544 | 10.6952 | 0.6937 | 0.3121 | 0.2775 |
| **5** | 8.3519 | 11.2046 | 0.6874 | 0.3184 | 0.2879 |
| **6** | 8.2851 | 11.0772 | 0.6655 | 0.3244 | 0.2951 |
| **7** | 7.9299 | 10.3298 | 0.7069 | 0.3171 | 0.2941 |
| **8** | 8.1255 | 10.6942 | 0.7001 | 0.3136 | 0.2743 |
| **9** | 8.1274 | 10.6497 | 0.6750 | 0.3308 | 0.3085 |
| **Mean** | 8.0288 | 10.5964 | 0.6942 | 0.3165 | 0.2868 |
| **Std** | 0.2254 | 0.3922 | 0.0159 | 0.0072 | 0.0101 |

might be caused by several factors, such as poor generalization and increased error on test data, which fail to make accurate predictions on unseen data and affect its generalization ability.

As depicted in Table XIV, the results are improved when the ensemble model is incorporated, and $R^2$ slightly increased compared to the initial value. However, it can be noticed that $R^2$ increased by 3.09% compared to the previous stage, as the variance is reduced by averaging the predictions of multiple models trained on different subsets of data.

The results of incorporating a stacking regressor are presented in Table XV. $R^2$ increased by 0.79% and 0.69% compared to the initial and ensemble method values, respectively. Such an increase in $R^2$ indicates that combined predictive power enhanced the stacked model and improved generalization and accuracy. When the holdout data is used, the results demonstrated in Table XVI imply that this method achieved the highest $R^2$ value (0.7127), Where the performance gain is 2.77% and 1.96% compared to the initial and stacking regressor values, respectively. The combined predictive power of utilized models allowed more generalization to unseen data. Each stage demonstrated a progressive improvement across all metrics (MAE, RMSE, RMSLE, and MAPE) according to the results in each table.

TABLE XV
DEPLOYMENT OF THE STACKING REGRESSOR BY UTILIZING THE THREE BEST REGRESSION MODELS (CATBOOST REGRESSOR, LGBM REGRESSOR', EXTRATREES REGRESSOR) FOR FOR NETWORK QUEUE DELAY

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 8.0436 | 10.6636 | 0.7047 | 0.3152 | 0.2814 |
| **1** | 7.6167 | 10.1162 | 0.7275 | 0.3083 | 0.2756 |
| **2** | 7.9972 | 10.5298 | 0.6907 | 0.3143 | 0.2860 |
| **3** | 7.5196 | 9.6959 | 0.7085 | 0.2990 | 0.2686 |
| **4** | 7.8449 | 10.6083 | 0.6986 | 0.3112 | 0.2724 |
| **5** | 8.2504 | 11.0529 | 0.6958 | 0.3149 | 0.2828 |
| **6** | 8.2262 | 10.9737 | 0.6717 | 0.3213 | 0.2920 |
| **7** | 7.8528 | 10.2175 | 0.7133 | 0.3177 | 0.2931 |
| **8** | 8.0982 | 10.6163 | 0.7044 | 0.3126 | 0.2729 |
| **9** | 8.1028 | 10.6499 | 0.6750 | 0.3302 | 0.3069 |
| **Mean** | 7.9552 | 10.5124 | 0.6990 | 0.3145 | 0.2831 |
| **Std** | 0.2323 | 0.3844 | 0.0160 | 0.0077 | 0.0111 |

TABLE XVI
PREDICTION OF NETWORK QUEUE DELAY USING THE HOLDOUT DATA

| Fold | Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| **0** | Stacking Regressor | 7.8795 | 10.4409 | 0.7127 | 0.3109 | 0.2813 |

### *A.3 sRTT*

In the initial prediction of sRTT, the CatBoost regressor provided the best performance (in terms of MAE, MAPE, and $R^2$) compared to the other tested regression models, as shown in Table XVII.

TABLE XVII
INITIAL PREDICTION OF sRTT

| Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **CatBoost Regressor** | 0.6115 | 0.8652 | 0.6495 | 0.0977 | 0.0853 |
| **Light Gradient Boosting Machine** | 8.0742 | 10.6451 | 0.6914 | 0.3198 | 0.2908 |
| **Extra Trees Regressor** | 8.2559 | 10.8737 | 0.6778 | 0.3285 | 0.3038 |
| **Random Forest Regressor** | 8.2610 | 10.8897 | 0.6769 | 0.3273 | 0.3015 |
| **Extreme Gradient Boosting** | 8.5795 | 11.2772 | 0.6536 | 0.3435 | 0.3074 |
| **Gradient Boosting Regressor** | 8.5462 | 11.3707 | 0.6480 | 0.3364 | 0.3057 |
| **K Neighbors Regressor** | 10.0073 | 13.1048 | 0.5321 | 0.3959 | 0.3837 |
| **Ridge Regression** | 10.4444 | 13.7138 | 0.4878 | 0.4348 | 0.3917 |
| **Linear Regression** | 10.4454 | 13.7138 | 0.4878 | 0.4351 | 0.3917 |
| **Bayesian Ridge** | 10.4445 | 13.7138 | 0.4878 | 0.4349 | 0.3917 |
| **Least Angle Regression** | 10.4454 | 13.7138 | 0.4878 | 0.4351 | 0.3917 |
| **Huber Regressor** | 10.3333 | 13.8146 | 0.4804 | 0.4156 | 0.3781 |
| **Passive Aggressive Regressor** | 10.8379 | 14.5355 | 0.4218 | 0.4567 | 0.3740 |
| **AdaBoost Regressor** | 12.3687 | 15.0337 | 0.3839 | 0.4919 | 0.5649 |
| **Lasso Regression** | 11.6777 | 15.7865 | 0.3219 | 0.4498 | 0.4595 |
| **Lasso Least Angle Regression** | 11.6777 | 15.7865 | 0.3219 | 0.4498 | 0.4595 |
| **Decision Tree Regressor** | 11.7606 | 15.7883 | 0.3200 | 0.4562 | 0.4032 |
| **Orthogonal Matching Pursuit** | 12.3014 | 16.6105 | 0.2490 | 0.4713 | 0.4855 |
| **Elastic Net** | 13.7800 | 18.2671 | 0.0921 | 0.5235 | 0.5597 |
| **Dummy Regressor** | 14.5472 | 19.1787 | -0.0009 | 0.5480 | 0.5923 |

TABLE XVIII
FINE-TUNING THE CATBOOST REGRESSOR USING THE SCIKIT-LEARN, SCIKIT-OPTIMIZE, AND OPTUNA HYPERPARAMETER OPTIMIZATION TECHNIQUES FOR sRTT

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 0.8619 | 1.1640 | 0.3796 | 0.1326 | 0.1226 |
| **1** | 0.8455 | 1.1465 | 0.3971 | 0.1310 | 0.1199 |
| **2** | 0.8427 | 1.1433 | 0.3840 | 0.1306 | 0.1194 |
| **3** | 0.8041 | 1.0681 | 0.3977 | 0.1245 | 0.1152 |
| **4** | 0.8407 | 1.1999 | 0.3627 | 0.1317 | 0.1187 |
| **5** | 0.8640 | 1.2722 | 0.3648 | 0.1355 | 0.1196 |
| **6** | 0.7882 | 1.0941 | 0.3824 | 0.1247 | 0.1125 |
| **7** | 0.8288 | 1.1383 | 0.3763 | 0.1300 | 0.1195 |
| **8** | 0.8306 | 1.1673 | 0.3853 | 0.1294 | 0.1174 |
| **9** | 0.8373 | 1.1182 | 0.3719 | 0.1297 | 0.1210 |
| **Mean** | 0.8344 | 1.1512 | 0.3802 | 0.1300 | 0.1186 |
| **Std** | 0.0223 | 0.0538 | 0.0112 | 0.0032 | 0.0027 |

TABLE XIX
BOOSTING THE MODEL'S PERFORMANCE BY APPLYING THE ENSEMBLE MODEL WITH BAGGING METHOD FOR sRTT

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 0.6218 | 0.8768 | 0.6480 | 0.0992 | 0.0868 |
| **1** | 0.6213 | 0.8602 | 0.6606 | 0.0973 | 0.0860 |
| **2** | 0.6122 | 0.8653 | 0.6471 | 0.0983 | 0.0851 |
| **3** | 0.5819 | 0.7895 | 0.6709 | 0.0928 | 0.0825 |
| **4** | 0.6056 | 0.8927 | 0.6472 | 0.0976 | 0.0838 |
| **5** | 0.6570 | 0.9687 | 0.6317 | 0.1037 | 0.0893 |
| **6** | 0.6175 | 0.8522 | 0.6253 | 0.0973 | 0.0869 |
| **7** | 0.6055 | 0.8447 | 0.6565 | 0.0965 | 0.0856 |
| **8** | 0.5916 | 0.8443 | 0.6784 | 0.0940 | 0.0824 |
| **9** | 0.6162 | 0.8652 | 0.6239 | 0.0993 | 0.0872 |
| **Mean** | 0.6131 | 0.8660 | 0.6490 | 0.0976 | 0.0856 |
| **Std** | 0.0191 | 0.0429 | 0.0174 | 0.0028 | 0.0021 |

The results of the fine-tuning process are presented in Table XVIII, it shows that $R^2$ is significantly decreased by -41.47% compared to the initial value. This is caused by sub-optimal parameter selection or an overfitting issue. As demonstrated in Table XIX, applying the bagging method does not seem to affect $R^2$, where the values are almost similar. However, it indicates a significant increase (70.67%) compared to the previous stage value, which indicates the effectiveness of this stage in enhancing robustness and reducing variance.

Later, when the stacking regressor is deployed as depicted in Table XX, $R^2$ increased by 0.63% and 0.71% compared to the initial and previous stage values, respectively. This indicates that the performance gains are marginal but consistent, leveraging multiple models' strengths. Finally, the highest achieved $R^2$ is 0.6560 when the holdout data is employed, as depicted in Table XXI. $R^2$ increased by 1% and 0.37% compared to the initial and previous stage values, respectively.

TABLE XX
DEPLOYMENT OF THE STACKING REGRESSOR BY UTILIZING THE THREE BEST REGRESSION MODELS (CATBOOST REGRESSOR, LGBM REGRESSOR', EXTRATREES REGRESSOR) FOR sRTT

| Fold | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|
| **0** | 0.6180 | 0.8680 | 0.6550 | 0.0986 | 0.0862 |
| **1** | 0.6221 | 0.8665 | 0.6556 | 0.0975 | 0.0858 |
| **2** | 0.6061 | 0.8603 | 0.6512 | 0.0976 | 0.0842 |
| **3** | 0.5711 | 0.7751 | 0.6828 | 0.0914 | 0.0810 |
| **4** | 0.6027 | 0.8845 | 0.6537 | 0.0973 | 0.0833 |
| **5** | 0.6528 | 0.9592 | 0.6389 | 0.1031 | 0.0888 |
| **6** | 0.6130 | 0.8513 | 0.6261 | 0.0970 | 0.0860 |
| **7** | 0.6027 | 0.8353 | 0.6642 | 0.0960 | 0.0853 |
| **8** | 0.5835 | 0.8321 | 0.6876 | 0.0929 | 0.0813 |
| **9** | 0.6118 | 0.8692 | 0.6205 | 0.0993 | 0.0866 |
| **Mean** | 0.6084 | 0.8601 | 0.6536 | 0.0971 | 0.0848 |
| **Std** | 0.0209 | 00.0439 | 0.0205 | 0.0031 | 0.0023 |

TABLE XXI
PREDICTION OF sRTT USING THE HOLDOUT DATA

| Fold | Model | MAE | RMSE | $R^2$ | RMSLE | MAPE |
|---|---|---|---|---|---|---|
| **0** | Stacking Regressor | 0.6107 | 0.8673 | 0.6560 | 0.0971 | 0.0848 |

*A.4 Predicted Performance Metrics*

A sample of predicted network throughput, network queue delay, and sRTT is presented in Table XXII. It shows the network throughput (BW), network queue delay (NQD), sRTT, their corresponding predicted values, and parameter sets.

The predicted values of network throughput are relatively close to the actual values, which indicates that the model can effectively learn from the given features. However, the distinctions between the actual and predicted values normally occur in predictive modeling. These differences can be analyzed and utilized for further research to improve the model.

TABLE XXII
SAMPLE OF PREDICTED PERFORMANCE METRICS

| | P1 | P2 | P3 | P4 | P5 | P6 | P7 | P8 | P9 | P10 | BW | P-BW | NQD | P-NQD | sRTT | P-sRTT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **10650** | 0.083 | 0.014 | 1016 | 1752 | 5 | 9.5 | 0.10 | 0.223 | 0.266 | 0.764 | 25.965 | 31.357 | 27.25 | 29.74 | 6.240 | 6.240 |
| **2041** | 0.054 | 0.424 | 1212 | 3223 | 6 | 9.1 | 0.79 | 0.133 | 0.094 | 0.733 | 29.714 | 31.558 | 23.41 | 23.27 | 6.233 | 6.410 |
| **8668** | 0.063 | 0.188 | 1175 | 1396 | 9 | 9.5 | 0.05 | 0.122 | 0.069 | 0.924 | 26.323 | 29.110 | 23.59 | 32.93 | 6.311 | 7.149 |
| **1114** | 0.136 | 0.491 | 944 | 2461 | 4 | 14.5 | 0.72 | 0.163 | 0.223 | 0.788 | 44.374 | 42.88 | 49.88 | 36.84 | 7.153 | 6.250 |
| **13902** | 0.067 | 0.127 | 1470 | 2443 | 8 | 6.7 | 0.42 | 0.31 | 0.252 | 0.742 | 33.585 | 28.889 | 42.08 | 25.81 | 7.271 | 6.228 |

Based on the samples given in the table, the achieved accuracy is 79.23% (ID number: 10650), 93.8% (ID number: 2041), 89.39% (ID number: 8668), 96.64% (ID number: 1114), and 86% (ID number: 13902).

For network queue delay, it can be observed that the $R^2$ value increased by 2.76% compared to Table XII. By comparing the predicted and actual values, we can notice that the prediction accuracy is as follows: Based on the samples given in the table, the achieved accuracy is 90.86% (ID number: 10650), 99.4% (ID number: 2041), 60.4% (ID number: 8668), 73.85% (ID number: 1114), and 61.33% (ID number: 13902).

In terms of sRTT, by comparing the predicted and actual values, we can notice that the prediction accuracy is as follows: Based on the samples given in the table, the achieved accuracy is 100% (ID number: 10650), 97.17% (ID number: 2041), 86.72% (ID number: 8668), 87.37% (ID number: 1114), and 85.42% (ID number: 13902).

### B. Prediction Insights: Visual Analysis

#### B.1 Residuals Plot

This part describes the residuals plot of the stacking regressor for network throughput, network queue delay, and sRTT demonstrated in Figures 3, 4, and 5, respectively. Blue points represent training data, while green points represent testing data. Also, the density of residuals is shown on the right side of each figure.

Figure 3 illustrates that the residuals are mainly distributed around the x-axis at zero, indicating potential improvement, and the model does not have a significant bias. The model provided a consistent performance as the residuals are spread relatively uniformly across the margins of predicted values. Furthermore, the model shows a proper fit for testing and training data, even though the predicted $R^2$ value is less than the training value by 6.7%, which usually occurs due to overfitting. Although the testing value of $R^2$ is lower than the training value, it still assures fairly efficient prediction.

For network queue delay, the residuals are centered around zero. A noticeable spread of residuals is seen at higher predicted values, which denotes that the model might struggle at higher delay predictions. The predicted value of $R^2$ is lower than the trained data by 16.7%, suggesting potential overfitting.

For sRTT, a training $R^2$ value of 0.857 indicates that the training data fits the model. However, testing $R^2$ is lower by 24.6%, which might refer to overfitting, performance issues on unseen data, or the model is not well generalized for sRTT predictions. Similar to network queue delay, the model has difficulties with high sRTT predictions, which are observed through a wider spread of residuals at high prediction values.

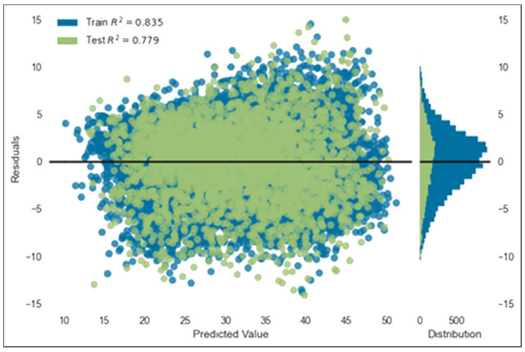


Fig. 3: Residuals plot of the stacking regressor model (network throughput)

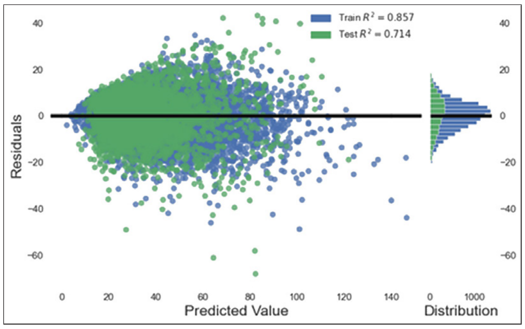


Fig. 4: Residuals plot of the stacking regressor model (network queue delay)

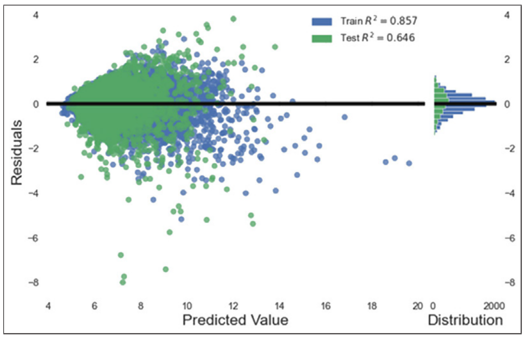


Fig. 5: Residuals plot of the stacking regressor model (sRTT)

#### B.2 Prediction Error Plot

Prediction error for network throughput, network queue delay, and sRTT demonstrated in Figures 6, 7, and 8, respectively. The best-fit line describes the median prediction trend, while the identity represents the variance of predicted values compared to the actual values. Predictions are accurate when the best fit and identity lines are closer.

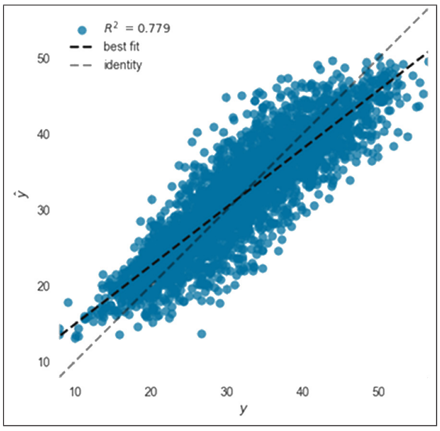


Fig. 6: Prediction error plot of the stacking regressor (network throughput)

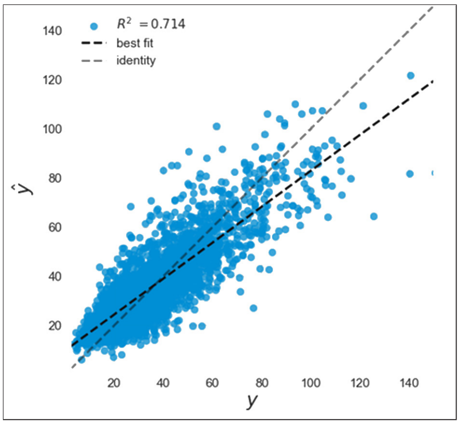


Fig. 7: Prediction error plot of the stacking regressor (network queue delay)

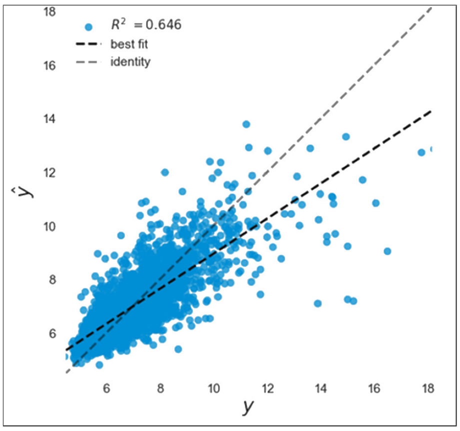


Fig. 8: Prediction error plot of the stacking regressor (sRTT)

For network throughput, although most points cluster around the best-fit line, a small linear trend is noticed. However, the $R^2$ value of 0.779 implies that our prediction accuracy is good and suggests a decent model fit. A little deviation from the identity line is noticed at high predicted values, which denotes possible difficulties in predicting high throughput.

$R^2$ value of 0.714 indicates a moderate accuracy level for network queue delay. Like the previous case, the model presents some limitations in predicting higher values. The model overestimates or underestimates delay in some scenarios, shown through points distant from the identity line. The model delivered weaker prediction capabilities when predicting sRTT values based on the achieved $R^2$ value (0.646). At the lower part, the predictions are closer to the actual values; however, when values increase, the predictions deviate from the actual results, which suggests that the model's reliability and consistency decrease at specific parts.

### *B.3 Scatter Plot*

Figures 9, 10, and 11 depict the scatter plot of the network throughput, network queue delay, and sRTT, respectively. The x-axis represents the experiment number and the y-axis denotes the output metric value. The actual values are in red, and the predicted ones are in blue.

The dense clustering of actual and predicted measurements for network throughput demonstrates a good performance of the utilized model. As the measurements spread further at higher values, higher variance in prediction accuracy is carried out, which means that the prediction model operates more efficiently at lower values. The model can generalize well for measuring network throughput while maintaining a consistent accuracy as no significant deviations are displayed.

For network queue delay, the displayed data reveals that most predictions fall at the lower end, along with the actual values, which implies that accuracy is higher in this range. However, there is a wide variance in the actual values that the model could not capture, indicating that the utilized model is not sensitive to such outliers, or the prediction range is insufficient.

Compared to the previous cases, the prediction performance for the sRTT is lower because the alignment is less accurate, and the actual values have more variation, while the predicted values are more concentrated around a particular range. Overall, the predicted values are clustered below the actual values, which shows some underestimation in some cases.

## V. CONCLUSION

This study presents a rigorous and systematic scheme that led to the development of robust machine-learning models utilized in SCReAM for predicting the network throughput, network queue delay, and sRTT. Despite facing challenges, the final models demonstrated promising results, implying their potential utility in future applications.

The ML models leveraged our constructed dataset, resulting in enhanced prediction capabilities. The coefficient of determination $R^2$ is used as one of the numerical performance metrics to evaluate the models. Several regression models were used to predict the network metrics for the SCReAM algorithm, followed by a comparative analysis to find the best initial prediction performance.

Among the tested models, the LightGBM and CatBoost regressors significantly outperformed others in predicting performance metrics. Fine-tuning with Optuna and ensemble

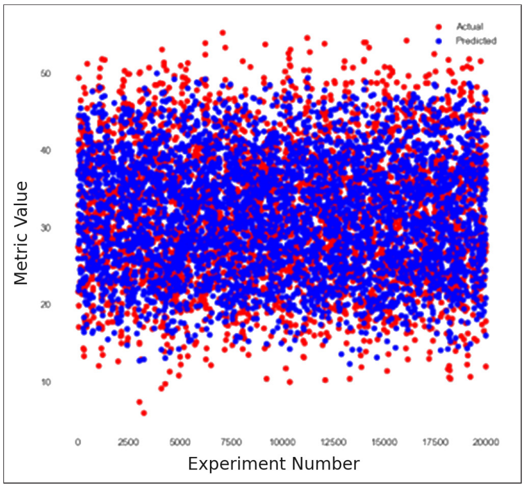


Fig. 9: Scatter plot comparing the actual and predicted values (network throughput (Mbps))

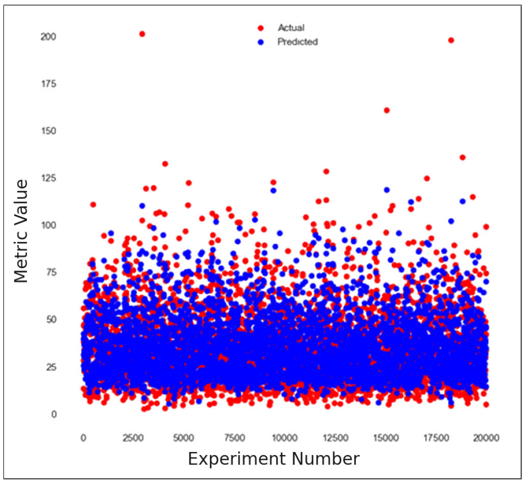


Fig. 10: Scatter plot comparing the actual and predicted values (network queue delay (ms))

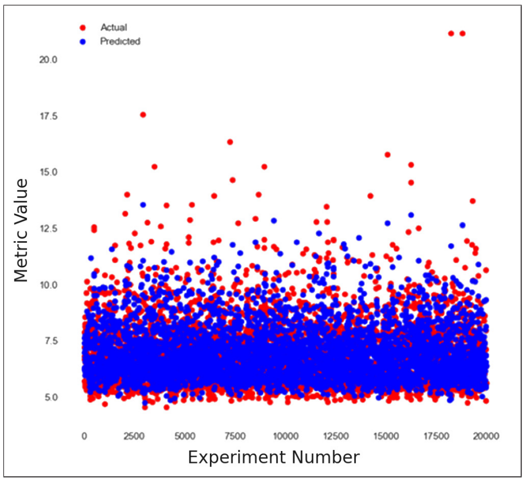


Fig. 11: Scatter plot comparing the actual and predicted values (sRTT (ms))

methods significantly improved the prediction accuracy of R2$^2$, indicating the effectiveness of these techniques. The achieved accuracy for network throughput ranges from 79.23% to 96.64%. For network queue delay, the prediction accuracy is from 60.4% to 99.4%. While ranging from 85.42% to 100% for sRTT.

Our work demonstrated an effective scheme for detecting several performance metrics based on the given features. Although we were able to improve the accuracy (or $R^2$) when integrating further methods, some experiments led to decreased $R^2$ value, which can be exploited and improved in future research. Furthermore, the model's performance can be further improved with more extensive hyperparameter tuning. Advanced ensemble techniques can also be employed to increase the accuracy of the prediction model.

It is essential to acknowledge that relatively minor differences in performance between the top ensemble methods can be caused by noise rather than actual performance improvements. Thus, in the future, it is crucial to perform rigorous statistical significance tests to determine if such differences fall within the expected variability caused by a random change.

## REFERENCES

[1] V. Kushwaha and R. Gupta, "Congestion control for high-speed wired network: A systematic literature review," *Journal of Network and Computer Applications*, vol. 45, pp. 62–78, 2014.

[2] B. Subramani and E. Chandra, "A Survey on Congestion Control," *Global Journal of Computer Science and Technology*, 2010.

[3] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," in *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE, 2018, pp. 1871–1879.

[4] K. Xiao, S. Mao, and J. K. Tugnait, "TCP-Drinc: Smart Congestion Control Based on Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 11 892–11 904, 2019.

[5] K. Winstein and H. Balakrishnan, "TCP ex Machina: Computer-Generated Congestion Control," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 123–134, 2013.

[6] W. Wei, H. Gu, and B. Li, "Congestion control: A renaissance with machine learning," *IEEE network*, vol. 35, no. 4, pp. 262–269, 2021.

[7] A. Elbery, Y. Lian, and G. Li, "Toward Fair and Efficient Congestion Control: Machine Learning Aided Congestion Control (MLACC)," in *Proceedings of the 7th Asia-Pacific Workshop on Networking*, 2023, pp. 88–94.

[8] S.-J. Seo and Y.-Z. Cho, "Fairness Enhancement of TCP Congestion Control Using Reinforcement Learning," in *2022 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. IEEE, 2022, pp. 288–291.

[9] A. Sha, S. Madhan, S. Neemkar, V. B. C. Varma, and L. S. Nair, "Machine learning integrated software defined networking architecture for congestion control," in *2023 International Conference On Distributed Computing And Electrical Circuits And Electronics (ICDCECE)*. IEEE, 2023, pp. 1–5.

[10] C.-Y. Yen, S. Abbasloo, and H. J. Chao, "Computers can learn from the heuristic designs and master internet congestion control," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 255–274.

[11] H. Naqvi and B. Anggorojati, "Ablation study of deep reinforcement learning congestion control in cellular network settings," in *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*. IEEE, 2022, pp. 80–85.

[12] V. Tong, S. Souihi, H. A. Tran, and A. Mellouk, "Troubleshooting solution for traffic congestion control," *Journal of Network and Computer Applications*, p. 103 923, 2024.

[13] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A Deep Reinforcement Learning Perspective on Internet Congestion Control," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3050–3059.

[14] N. Jay, N. H. Rotman, P. Godfrey, M. Schapira, and A. Tamar, "*Internet Congestion Control via Deep Reinforcement Learning*," *arXiv preprint arXiv:1810.03259*, 2018.

[15] Y. Kong, H. Zang, and X. Ma, "Improving TCP Congestion Control with Machine Intelligence," in *Proceedings of the 2018 Workshop on Network Meets AI & ML*, 2018, pp. 60–66.

[16] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-Learning Congestion Control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 343–356.

[17] M. Schapira and K. Winstein, "Congestion-Control Throwdown," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, 2017, pp. 122–128.

[18] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[19] A. Farchi, M. Bocquet, P. Laloyaux, M. Bonavita, and Q. Malartic, "A comparison of combined data assimilation and machine learning methods for offline and online model error correction," *Journal of Computational Science*, vol. 55, p. 101 468, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750321001435

[20] J. Schrittwieser, T. Hubert, A. Mandhane, M. Barekatain, I. Antonoglou, and D. Silver, "Online and Offline Reinforcement Learning by Planning with a Learned Model," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 27 580–27 591. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/e8258e5140317ff36c7f8225a3bf9590-Paper.pdf

[21] H. D. Zubaydi, A. S. Jagmagji, and S. Molnár, "Experimental Analysis and Optimization Approach of Self-Clocked Rate Adaptation for Multimedia Congestion Control Algorithm in Emulated 5G Environment," *Sensors*, vol. 23, no. 22, p. 9148, 2023.

[22] I. Johansson, "Self-clocked rate adaptation for conversational video in LTE," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, 2014, pp. 51–56.

[23] I. Johansson and Z. S. (2017)., "IETF RFC8298," [Online]. Available: https://www.rfc-editor.org/rfc/rfc8298.html.

[24] A. S. Jagmagji, H. D. Zubaydi, and S. Molnar, "Exploration and Evaluation of Self-Clocked Rate Adaptation for Multimedia (SCReAM) Congestion Control Algorithm in 5G Networks," in *2022 45th International Conference on Telecommunications and Signal Processing (TSP)*. IEEE, 2022, pp. 230–237.

[25] H. D. Zubaydi, A. S. Jagmagji, and S. Molnár, "Squeezing the Most Out of Congestion Window for Self-Clocked Rate Adaptation Algorithm in a 5G Environment," in *2023 17th International Conference on Telecommunications (ConTEL)*. IEEE, 2023, pp. 1–8.

[26] S. H. Choi and M. Handley, "Fairer TCP-Friendly Congestion Control Protocol for Multimedia Streaming Applications," in *Proceedings of the 2007 ACM CoNEXT conference*, 2007, pp. 1–2.

[27] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM computer communication review*, vol. 18, no. 4, pp. 314–329, 1988.

[28] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," *Advances in neural information processing systems*, vol. 30, 2017.

[29] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," *Advances in neural information processing systems*, vol. 31, 2018.

[30] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[31] M. Spüler, A. Sarasola-Sanz, N. Birbaumer, W. Rosenstiel, and A. Ramos-Murguialday, "Comparing metrics to evaluate performance of regression methods for decoding of neural signals," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2015, pp. 1083–1086.

[32] L. Breiman, "Stacked Regressions," Machine learning, vol. 24, pp. 49–64, 1996.

[33] L. (2023)., "Welcome to LightGBM's documentation," [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Parameters.html.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] T. Head, M. Pak, I. Shcherbatyi, K. Lomax, T. Fan, A. V. del Moral, B. Bossan, Z. Vinícius, and A. Popov, "Scikit-Optimize: Efficient and user-friendly optimization library in Python," 2018. [Online]. Available: https://github.com/scikit-optimize/scikit-optimize/tree/v0.5.2

**Ahmed Samir Jagmagji** received his B.Eng. degree in Computer Technology Engineering from the Technical College of Mosul, Iraq, in 2005 and his M.Sc. degree in Computer Engineering from the University of Missouri – Columbia, United States, in 2016. He is currently a Ph.D. candidate in the Telecommunications And Media Informatics department, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, Hungary. His research interests include Computer Networking, the Internet of Things (IoT), Eldercare technology, and Congestion Control Algorithms.

**Haider Dhia Zubaydi** received a B.Sc. degree in Information and Communication Engineering from the University of Baghdad, Iraq, in 2014 and an M.Sc. in Internet Engineering from the National Advanced IPv6 Center at Universiti Sains Malaysia in 2018. He is pursuing a Ph.D. in Computer Engineering at the High-Speed Networks Laboratory (HSN LAB) at the Budapest University of Technology and Economics, Hungary. His research interests include network security, SDN, blockchain technology, and congestion control algorithms.

**Sándor Molnár** received his M.Sc., Ph.D. and Habilitation in Electrical Engineering and Computer Science from the Budapest University of Technology and Economics (BME), Budapest, Hungary, in 1991, 1996 and 2013, respectively. In 1995 he joined the Department of Telecommunications and Media Informatics, BME. He is now an Associate Professor and the principal investigator of the tele-traffic research program of the High-Speed Networks Laboratory.

**Mahmood Alzubaidi** completed his Master's degree in 2018 in Internet Engineering from the National Advanced IPv6 Center at Universiti Sains Malaysia. He then pursued and received his PhD in 2023 from Hamad Bin Khalifa University, Qatar, where he continues to contribute to the field as a researcher. His research interests are broad, spanning across the Internet of Things (IoT), deep learning, and machine learning.