Towards QoS Prediction based on Temporal Transformers for IoT Applications

Aroosa Hameed, John Violos, Aris Leivadeas, Nina Santi, Rémy Grünblatt, Nathalie Mitton

Towards QoS Prediction based on Temporal Transformers for IoT Applications

Aroosa Hameed, John Violos, Aris Leivadeas (*Senior Member, IEEE*), Nina Santi, Rémy Grünblatt, Nathalie Mitton (*Member, IEEE*)

***Abstract*—Internet of Things (IoT) devices generate a tremendous amount of time series data that is extremely dynamic, heterogeneous and time dependent. Such types of data introduce significant challenges for the real-time prediction of QoS metrics of IoT applications with different traffic characteristics. To this end, in this paper, we propose a temporal transformer model and a unified system to predict several QoS metrics of heterogeneous IoT applications when they communicate with the Edge of the network. The transformer model also leverages an attention module to provide a solution for both short-term and long-term sequence prediction of QoS metrics that allows to better extract any time dependencies. In particular, in our framework, we firstly generate a set of datasets containing real-time traffic information of five different IoT applications such as Heating, Ventilation, and Air Conditioning (HVAC), lighting, Voice over Internet Protocol (VoIP), surveillance and emergency response using the 802.15.4 access technology and the RPL routing protocol. Following, we perform the data cleaning, downsampling and pre-processing of the datasets and we construct the QoS datasets, which include four QoS metrics, namely throughput, packet delivery ratio, packet loss ratio and latency. Finally, we evaluate the transformer model through extensive experimentation using both short-term and long-term dependencies and we show that our model can guarantee a robust performance and accurate QoS prediction.**

***Index Terms*—Deep learning, Edge computing, Internet of Things, QoS prediction, time series, transformer**

I. Introduction

THE number of Internet of Things (IoT) applications have considerably increased, while generating a tremendous amount of data. According to Cisco, the number of connected devices will reach up to 14.7 billion by 2023 [1]. The devices are expected to continuously generate large volumes of data requiring extensive analysis to capture valuable information that can help in the intelligent decision making. However, the device's CPU, memory, and disk capacity restricts the data processing on the device itself. Thus, the data and the analysis processing have to be offloaded to more resource powerful platforms, such as the newly introduced Edge Computing [2]. Edge computing can facilitate the data processing very close to the source of the data, reducing thus the overall latency perceived. In this way also, the processing burden is shifted/offloaded to the Edge of the network through a process that is called task offloading [3]. However, the amount of Edge resources needed for each IoT application depends on the volume of the data generated from the IoT devices. This creates an important challenge related to the accurate workload (e.g. throughput) profiling of an IoT application.

At the same time, IoT applications consist of heterogeneous devices that send data of different contexts, with different reporting frequencies usually over a random access channel generating thus, high interference levels [4]. All these add several levels of complexity when it comes to the prediction of typical Key Performance Indicators (KPIs) in IoT. Regarding the reporting frequency, IoT devices follow very dynamic models ranging from periodic to event-based transmissions. Hence, the feature of time dependence make such data different and more challenging than traditional data. Therefore, each IoT application, when generating/offloading data, will have different instantaneous Quality of Service (QoS) behavior, which will be time dependent. Hence, it is necessary to propose an efficient model that will analyze and predict the QoS metrics using IoT time series data.

A time series data is a series of data points that are ordered by their chronological order. Time dependency is a very important feature of the IoT time series data, since data are becoming widespread in an IoT context [5]. Accordingly, the time feature is affecting the way prediction and analysis of IoT data is done. One way to predict the data at a next time step is to use the data from previous time steps in the short or long past [6]. Therefore, there is huge interest in analyzing the IoT traffic profiles by applying various machine learning techniques [7].

For example, several studies applied traditional time series algorithms or deep learning models to predict the IoT traffic behavior [8]-[23]. In the studies [8]-[17], the authors applied diverse deep learning algorithms such as Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), attention mechanism, regression techniques and stochastic gradient descent for the prediction of either specific or a set of QoS metrics. Nonetheless, various research gaps can be identified in these existing studies. Firstly, in [8]-[13] several QoS prediction mechanisms are presented, however, without considering any time dependencies. Secondly, for the works [14]-[17], only a simple traffic prediction is provided, without predicting typical QoS metrics found in an IoT context. Thirdly, no multivariate prediction of QoS is provided for the studies [11], [14] and [15], which is an important element to capture the dependencies among multiple QoS metrics. Additionally, some works, such as [18]-[23], applied deep learning specifically for the time series fore-

A. Hameed, J. Violos and A. Leivadeas are with the Department of Software and Information Technology Engineering, École de Technologie Supérieure, University of Quebec, Montreal, Canada. N. Santi and N. Mitton are with INRIA Lille-Nord, France. R. Grünblatt is with SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Évry, France E-mails: {aroosa.hammed, ioannis.violos}.1@ens.etsmtl.ca, aris.leivadeas@etsmtl.ca, {nina.santi, nathalie.mitton}.inria.fr, remy@grunblatt.org

casting task. These works proposed various learning networks such as Temporal Convolutional Network (TCN), DeepAR, LSTNet and an improved versions of LSTM as stacked LSTM and bidirectional LSTMs for time series forecasting problems. However, this set of works lacks the ability to handle both the short and long-term dependencies at the same time, while training over long sequences of data degrades the accuracy of the prediction.

To overcome the above described research gaps of the current studies, we deploy five different IoT applications to investigate four different QoS metrics. Moreover, this research also investigates the multivariate prediction of QoS metrics for each application. Last but not least, a novel model that makes an efficient use of the time features of the IoT applications and accurately predicts their QoS behavior, in a dynamic network environment, is proposed. The model also investigates the short and long input sequence dependencies without any performance degradation. To this end, the main contributions of this paper can be summarized as follows:

- We consider 5 different IoT smart building applications that present different requirements in terms of number of devices, packet length, context of message, and message frequency transmission. We deploy the applications in a real testbed [24] comprised of approximately 300 IoT devices and generate data over an IEEE 802.15.4 access network.
- We provide the predictions of four major QoS metrics such as Throughput, Packet Delivery Ratio (PDR), Packet Loss Ratio (PLR) and Latency. As multivariate time series forecasting poses a challenge of how to capture and leverage the dependencies among multiple variables, we provide both univariate and multivariate multi-step prediction for all four QoS metrics of the five IoT applications under consideration.
- We design and implement a QoS prediction mechanism based on Temporal Transformers that models temporal dependencies within input sequences consisting of IoT data and that is able to handle the long input sequences with the attention module to make prediction. The model accurately provides the multi-step QoS prediction and its temporal relation with its preceding QoS values from past observations.

The rest of the paper is organized as follows: Section II presents the related work and current limitations. Section III gives a detailed information on the challenges of IoT time series data. Section IV summarizes the real time dataset generation of the considered IoT applications. Section V presents the proposed model along with its algorithmic form and asymptotic analysis. Section VI provides the experimentation setup and illustrates the results and the efficiency of the proposed solution. Finally, Section VII concludes the paper.

II. Related Work

In the pertinent literature, there are various studies either for the prediction of IoT traffic along with the QoS metrics or for the general time series forecasting task using machine learning or deep learning approaches. Thus, in this section we divide the related work into two distinct categories: i) deep learning models for QoS prediction and ii) deep learning models for general time series forecasting.

A. Deep Learning for QoS Prediction

The authors in [8], predicted the delay using a nonlinear autoregressive exogenous (NARX) RNN following both a single-step and a multi-step ahead prediction. The prediction accuracy is measured using MSE, RMSE and MAPE metrics. However, they used a simulated dataset of an IoT environment. Furthermore, the delay metric is also predicted in [9] using a simple Deep Neural Network (DNN) consisting of forward with backward passes and also providing the analysis of hyperparameters, which presented good results such as size of training data, number of layers, number of neurons in each layer and epochs. The features utilized by this work were extracted from the application layer, MAC layer and physical layer of the network. The authors in [10] proposed a deep learning model that predicts the throughput, delay, and packet loss of an IoT communication system. The proposed model consists of three layers: The first layer includes a neural network for the Internet as it represents the transmission medium between different networks in an IoT system. The second layer consists of a number of neural network for each access network such as Wireless Sensor Networks (WSN), Radio Frequency Identification (RFID) network and Mobile Ad-hoc Network (MANET) in an IoT system. This layer predicts the individual performance of each network. The third layer comprises the last neural network model which is used to predict the final performance of the entire IoT system. The work in [11] attempted to predict the throughput using a Convolutional Neural Network (CNN) with the target vectorization technique as their throughput distribution was centralized and concentrated on several values. This is why and in order to mitigate this centralized distribution they resorted to a vectorization technique. However, the dataset was generated from a simulated factory scenario.

Fan et al. [14] proposed a deep learning based Recurrent Neural Network (RNN) model using an attention mechanism for the IoT data processing at the Edge. All input time series were fed into the RNN and attention network to calculate the extrinsic correlations and to provide the final prediction. The proposed model, called UrbanEdge, used four different datasets such as traffic volume, building occupancy, electricity and Air Quality Index (AQI) consisting of time series based sensor readings. The results proved that the proposed UrbanEdge model outperforms several baseline methods such as Autoregressive Integrated Moving Average (ARIMA), Vector Autoregression (VAR), LSTM and Sequence-to-Sequence (Seq2Seq). However, there is the vanishing gradient problem for the training of the RNN and the model also requires a high bandwidth for the transfer of the monitoring metrics.

The authors in [15], proposed EdgeLSTM, which is an Edge-based deep learning system that utilizes grid LSTM along with Support Vector Machine (SVM). The pipeline of this framework followed a data processing, a hyperparameter selection, and a construction of multi-class SVM models to be

trained using four different datasets. The output was to get the results for four different tasks such as data prediction, network maintenance, anomaly detection and mobility management. Abdellah et al. [16] performed the prediction of throughput of IoT traffic in a 5G communication network using an LSTM network. The dataset is generated using an IoT traffic generator. The features of the dataset includes the timestamp, bytes count and packets count. Finally, the authors in [17] proposed the forecasting of IoT traffic by using a stochastic gradient descent algorithm and a neural network architecture called gaNET. The dataset used in the paper consists of features such as obfuscated mobile identification and timestamp of records.

There are also few recent studies that applied regression based approaches [12], [13], to predict throughput and packet delivery ratio (PDR), since regression based techniques tend to be a light weight alternative for the prediction of QoS metrics. However, most of the IoT data used for the QoS prediction consist of time series sequences which are better predicted using deep learning approaches, such as Recurrent Neural Networks (RNN) or Long Short-Term Memory (LSTM) networks, that are specifically designed for handling time series data.

B. Deep Learning for Time Series Forecasting

Regarding the time series data forecasting, various neural network based methods are developed for sequence-to-sequence learning. Specifically, RNNs are well suited for the time series forecasting as they consist of a memory cell that can be used to recall things from the past. However, as explained before, the vanishing gradient problem persists over the longer time series sequences. A variant of RNN is LSTM [18] that uses a gating mechanism for controlling an access to memory cell and mitigates the vanishing gradient problem. There is also a stacked LSTM model [19] for the time series prediction. This model stacks LSTM layers on top of each other to learn longer dependencies. Another extension to LSTM is the bidirectional LSTM [20] in which two models are trained. The first model is used for learning the input sequence and the second learns the reverse of that sequence.

Furthermore, a Temporal Convolutional Network (TCN) which combines the dilations and residual connections with the causal convolutions needed for autoregressive prediction, was proposed in [21]. The authors showed that TCN performed better than RNN models for time series forecasting tasks. Salinas et al. [22] proposed a model called DeepAR for probabilistic forecasting using autoregressive recurrent networks that learns from historical data of all time series in the dataset and provides the forecasting results. Another deep learning model for multivariate time series forecasting, was proposed in [23] called Long- and Short-term Time-series Network (LSTNet). This work combined the convolutional layer along with recurrent layer to learn both local patterns and long-term dependencies among multi-dimensional input variables. It also incorporated the autoregressive linear model along with a non-linear model to make the framework more robust for the time series which violate scale changes.

C. Limitations of the Related Work

As stated in Section I, the limitations of the above mentioned works can be summarized as follows:

- Most of the studies provide the prediction of the IoT traffic type and do not predict the QoS attributes [14]-[17]. There are only few studies that provide the QoS prediction [8]-[13]. However, these works have not thoroughly examined the actual prediction task with respect to time, especially in emerging IoT application scenarios.
- Some of the existing studies provide the prediction of IoT traffic or QoS attributes as a univariate forecast [11], [14] and [15]. However, multivariate prediction can capture and use the dependencies among multiple variables to predict the future QoS at a specific time step.
- The existing studies based on neural networks are mostly designed for a short-term sequence prediction setting [18]-[22]. Specifically, RNN based models have the vanishing gradient problem which prevents the training over long sequences of data.

In this work, we solve the above mentioned challenges as follows: (i) Firstly, we provide the detailed prediction of four QoS metrics such as throughput, packet delivery ratio (PDR), packet loss ratio (PLR) and latency for five heterogeneous IoT applications such as HVAC, VoIP, lighting, surveillance and emergency application; (ii) Secondly, we provide the multistep prediction of each QoS in both univariate and multivariate settings; (iii) Thirdly, to overcome the vanishing gradient problem in the training of long QoS data sequences, we are introducing a temporal transformer architecture. To the best of our knowledge, this is the first work which provides a transformer based QoS prediction for IoT applications.

III. Problem Formulation of QoS Prediction

In this section, we describe and formulate the QoS prediction problem, when we have multiple QoS metrics such as throughput, PDR, PLR and latency to be predicted and when IoT devices belonging to different IoT applications communicate with an Edge infrastructure. In particular, the IoT applications are represented by the set $A = \{a_1, a_2, a_3, a_4, a_5\}$ where a_1 represents the first IoT application, a_2 represents the second IoT application and so on. Similarly, the set $D = \{d^1_{a_1}, d^2_{a_2}, ..., d^m_{a_i}\}$ represents the data generated by each IoT application where $d^1_{a_1}$ represents the first dataset in the set D and it is generated by the IoT application a_1. The $d^m_{a_i}$ denotes the m^{th} dataset generated and it is for the i^{th} IoT application where $m <= 5$ and $i <= 5$ as data is generated for five different IoT applications. Furthermore, each network dataset generated for an i^{th} IoT application is constituted by a sending and receiving information which is denoted as $D = \{(u^1_{a_1}, s^1_{a_1}), (u^2_{a_2}, s^2_{a_2}), (u^3_{a_3}, s^3_{a_3}), (u^4_{a_4}, s^4_{a_4}), (u^5_{a_5}, s^5_{a_5})\}$ where $(u^1_{a_1}, s^1_{a_1})$ represents the pair of sending and receiving information for IoT application a_1. More specifically, $U = \{u^1_{a_1}, u^2_{a_1}, ..., u^j_{a_1}\}$ denotes the set of the transmitting information by the IoT devices of the IoT application a_1. Similarly, $S = \{s^1_{a_1}, s^2_{a_1}, ..., s^j_{a_1}\}$ represents the set of the receiving information at the Edge server side, where $s^j_{a_1}$ is the j^{th} receiving information of the i^{th} IoT application.

Regarding the features used, the set UF denotes the features related to the transmitting data in the network by the IoT devices as: $UF = \{u_{f_1}, u_{f_2}, u_{f_3}, u_{f_4}, u_{f_5}, u_{f_6}\}$ where u_{f_1} denotes the timestamp at which the packet is sent; u_{f_2} is the sensor node ID that is sending the packet; u_{f_3} represents the size of the UDP payload in bytes; u_{f_4} is the IPv6 destination address (we use an 802.15.4 access network with 6LoWPAN); u_{f_5} is the destination port; u_{f_6} is the actual payload in a hexadecimal format. In a similar way, the set SF represents the features related to the receiving information at the Edge server side and is further expressed as $SF = \{s_{f_1}, s_{f_2}, s_{f_3}, s_{f_4}\}$, where s_{f_1} represents the timestamp at which the packet is received; s_{f_2} is the IPv6 address from which the packet originates; s_{f_3} denotes the receiver port on which the packet has been received; and s_{f_4} is the hexadecimal payload of the packet. Given the sets UF and SF, we computed the QoS datasets for each IoT application. The throughput is represented as $Q = \{q_1^1, q_2^2, ..., q_i^t\}$ where q_i^t is the i^{th} throughput value at timestamp t, such that $0 < t < T$, where T represents the total timestamps for which data are generated. The packet deliver ratio is represented as $P = \{p_1^1, p_2^2, ..., p_i^t\}$ where p_i^t is the i^{th} PDR value at timestamp t. The packet loss ratio is denoted as $E = \{e_1^1, e_2^2, ..., e_i^t\}$, where e_i^t is the i^{th} PLR value at timestamp t. Lastly, the latency is denoted as $L = \{l_1^1, l_2^2, ..., l_i^t\}$, where l_i^t is the i^{th} latency value at timestamp t.

In the Time Series Forecasting (TSF) setting, let $X = \{x^1, x^2, ..., x^N\}^T$ represent the multivariate QoS time series with N variables, T as timestamp and $X \in \mathbb{R}^{T \times N}$. When $N = 1$ it becomes a univariate time series problem which can be represented, for the throughput Q for example, as the i^{th} univariate QoS time series, given as $X_i^T = \{x_1^1, x_2^2, ..., x_i^t\} \in Q^T$ where x_i^t is the i^{th} value of the QoS metric collected at a timestamp t. Given the X and a fixed window size τ, with $\tau \in \mathbb{N}$, this time series is split into a fixed length input as $X = \{(x_1^t, x_2^{t+1}, .., x_\tau^{t+\tau}), (x_1^{t+1}, x_2^{t+2}, .., x_\tau^{t+\tau}), ..., (x_1^{t+i}, x_2^{t+i+1}, .., x_\tau^{k+\tau})\}$ such that $0 < t < T$, $\forall i \in \mathbb{N}$ and $k = T - \tau$.

Given the input time sequence as $\{x_1^t, x_2^{t+1}, .., x_\tau^{t+\tau}\} \subset X$, we consider the task of predicting either only one step ahead value, such as to predict the value of $x_{\tau+1}^{t+\tau+1}$ or multistep values i.e., h number of future values of QoS as $\tilde{X} = \{\tilde{x}_1^t, \tilde{x}_2^{t+1}, ..., \tilde{x}_{h-1}^{T-1}, \tilde{x}_h^T\}$, with $h \in \mathbb{N}$ and $\tilde{x}_1^t$ trying to predict the value of $x_{\tau+1}^{t+\tau+1}$, and so on. Thus, the goal is to learn a precise forecasting model as $M : X_{t,\tau(i)} \rightarrow \tilde{X}_{t,h(i+\tau)}$ by minimizing some loss function.

IV. Edge Computing Infrastructure and Dataset Construction

A. Applications and Edge Computing Infrastructure

Five different IoT applications and their respective datasets are considered in this work. These applications are: **1) Emergency Response:** The emergency system is used to monitor the critical areas of the building such as gas pipes or fire alarms. If a situation occurs where the pipelines reach high pressure, which may cause an explosion, then the IoT devices at a specific location will detect this and send an alert with relevant contextual information to a control system to remedy the situation. **2) Heating, Ventilation and Air Conditioning (HVAC):** The HVAC system provides various handling systems inside the building by controlling factors such as temperature, humidity etc., in order to provide the necessary comfort and indoor air quality to the occupants. **3) Surveillance:** The surveillance systems involve cameras, monitoring and sensor devices that are used to provide the required physical security at a specific location. **4) Voice over IP (VoIP):** The VoIP systems are used for providing automatic help desks or interactive voice recognition. **5) Lighting:** The lighting systems can be used to provide information regarding room occupancy, while also reducing the total energy consumption of the building.

All of the above applications coexist in the same building and generate data at the same time. This can create a very dynamic environment, especially when a random access channel is considered that can create QoS uncertainties due to interference and re-transmissions. For each of the IoT applications, the experiment involves three types of entities, or nodes, namely:

1) Server: This entity (node) represents a UDP server which collects and receives all of the information regarding the packet exchanges in the network. For all of the experiments, one central server is used, which is accessible through the internet via an IPv6 connection.
2) Border Routers: The sensor nodes are connected to the internet via border routers which have two interfaces. The first interface is connected to the internet and the second is connected to the sensors network, using the 802.15.4 as an access protocol and the IPv6 Routing Protocol for Low power and Lossy Networs (RPL) as the routing protocol. More specifically, the border routers are the roots of the RPL's Destination Oriented Directed Acyclic Graphs (DODAGs) with a role similar to the ISP "box" for residential users that have an interface connected to the Internet and another providing Wi-Fi connectivity. For the experiment purposes, the total number of border routers is kept constant for each of the individual application, however it may vary as it is a modifiable parameter.
3) Sensors: The sensors are nodes that are used to generate data following a specific distribution, as shown in Table I, according to the five IoT applications mentioned earlier. The sensor data are transmitted to the server using the 802.15.4 technology via the RPL routing mechanism. Further, each sensor can also be used to relay packets to border routers, if it lies on the shortest path between a sensor and the DODAG root. Each sensor can have several DODAG parents, creating multiple possible paths to the border routers.

We have defined a heterogeneous set of parameters for each IoT application to perform the data generation experiments. These parameters include the number of sensors, number of border routers, duration, packet length in bytes, generation type of packets, lambda value of their generation type and time period in seconds, as shown in Table I. The only common

TABLE I
EXPERIMENTATION'S PARAMETERS

Scenario	No. of sensors	No. of routers	Duration (s)	Packet Length (B)	Generation Type	Lambda	Period (s)
Surveillance	10	3	10090	127	Exponential	196.74	—
Emergency Response	40	5	10090	127	Hybrid	0.0333	30.0
HVAC	100	5	10090	60	Periodic	—	260.0
Lighting	100	5	10090	30	Exponential	0.00208	—
VoIP	10	1	10090	127	Hybrid	15.74	0.063532

parameter among the five applications is the duration of the experimentation, since the applications coexist at the same time. The generation type represents the distribution according to which application data are generated. If it is exponential, as for surveillance and lighting applications, then the packets generated by each node follow an exponential distribution using the parameter Lambda. If the generation type is Periodic i.e., for HVAC, then the packets are generated periodically according to the Period parameter. If the generation type is hybrid i.e., for emergency response and VoIP applications, then data follow a hybrid generation according to an exponential distribution that follows a specific Lambda value and a periodic pattern. This behavior creates another level of QoS uncertainty that can lead to considerable traffic fluctuations, as well as spectrum and resource requirements. More details regarding the testbed and the dataset generation can be found in [25].

B. Feature Engineering

The dataset generated for the five different IoT applications provide the receiving and transmitting information of the packets within the network. Each application has its own database with UDP and server tables. The UDP table contains information about packets as they are transmitted by the sensors and the Server table contains information about packets as they are received by the server. The raw features are highlighted in Table II.

TABLE II
DESCRIPTION OF RAW FEATURES IN DATASET

Data	Feature	Description
Transmitting data (UDP)	node_name	name of sensor node
	timestamp	time at which the packet is sent
	payload_size	size of the UDP payload, in bytes
	dest_address	destination IPv6 address
	dest_port	contains the destination port
	payload	hexadecimal identifier of the packet
Receiving Data (Server)	timestamp	time at which the packet is received
	IPv6_address	source IPv6 address
	receiver_port	port on which the packet is received
	payload	hexadecimal identifier of the packet

In order to extract the most useful features from the given raw data, we engineered several features as described below:

1) Timestamp: It is the time that is associated with each packet in the network. Initially, data were collected and added to the raw dataset at a nanosecond granularity. However, we changed the granularity of the dataset from 1 nanosecond to 5 milliseconds, to better capture the QoS metrics fluctuations. For example, it was not always possible to calculate the QoS metrics for each nanosecond as in most of the nanosecond timestamps we did not have any sending or receiving packets in the network that was causing the generation of many null values for the QoS datasets. Thus, each of the below described features are computed for a time interval t of 5 milliseconds without however losing significant information.
2) $time_{first\_pack}$: It is the time at which the first packet is transmitted in a specific time interval of 5 ms.
3) $time_{last\_pack}$: It is the time at which the last packet is transmitted to the server in a specific time interval of 5 ms.
4) $total_{trans\_pack}$: It is the total number of packets transmitted by a node during a specific time interval of 5 ms.
5) $total_{rec\_pack}$: It is the total number of packets received by the server during a specific time interval of 5 ms.
6) Packet Delivery Ratio (PDR): It is the ratio of the received packets to the transmitted packets per node for every 5 ms and it is given as:

$$PDR = \frac{total_{rec\_pack}}{total_{trans\_pack}} * 100 \tag{1}$$

7) Packet Loss Ratio (PLR): It is the ratio of the lost packets to the received packets at the server side and it is given as:

$$PLR = \frac{total_{loss\_pack}}{total_{rec\_pack}} * 100 \tag{2}$$

8) Throughput: It is the rate of the total number of received packets (or their size) over a time period of 5 ms:

$$Throughput = \frac{total_{rec\_pack}}{time_{last\_pack} - time_{first\_pack}} \tag{3}$$

9) Transmission Latency: It is the average time taken by a transmitted packet to be successfully received at the receiving side over a time period of 5 ms and is given as:

$$Latency = \frac{\sum_{t\_pack}(time_{rec\_pack} - time_{trans\_pack})}{total_{trans\_pack}} \tag{4}$$

C. Data Preprocessing

Each application dataset is stored in a SQLite3 database and compressed with the zstd compression algorithm. We firstly decompress the dataset and read the sql table in the .csv format. Then we engineer the QoS related features and

create a second QoS dataset for each of the IoT applications. However, before the QoS datasets are fed to our proposed transformer models for training or validation purposes, several preprocessing operations are applied to refine their quality and thereby the QoS forecasting performance. In particular, we remove any outliers that are caused by some unseemly situations in the datasets. There are also some missing values in the QoS dataset because it may occur that no packets are transmitted and received for some time intervals. For instance, the HVAC and lighting applications are generating packets with very low frequencies, as can be seen in Table I. For the particular applications, the missing values are filled by average values of their respective features.

Finally, the features of each application dataset is normalized in a particular range using the min-max normalization given as:

$$X\_normalized = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5}$$

where x is the original QoS value of the metric/feature under consideration (e.g. Throughput, PDR, PLR and Latency), x_{min} represents the minimum value of that feature and x_{max} denotes its maximum value. Thus, the normalized data lie in the range from 0 to 1.

V. Proposed Temporal Transformer Framework

This section discusses the overview of the proposed temporal transformer for the QoS time series prediction between the IoT devices and the Edge server. Following, the next paragraphs discuss the details of the proposed model and present the description of each of its modules.

A. Overview of Proposed Framework

Given the ability of temporal transformer models to get the time dependencies of a dataset, we proposed a framework which adopts the benefits of the particular model to process and estimate the QoS metrics for IoT applications in an edge environment. In the proposed framework as shown in Fig. 1, we first generate the real IoT data for five different applications as discussed already in the section IV.A. Then, our second step is to take all of these raw datasets and engineer the new useful features as discussed in section IV.B. Then we process these data by performing data cleaning, data down-sampling and data normalization. Then the new pre-processed QoS datasets for the five IoT applications are divided into training sets, validation sets, and testing sets. The total experimentation duration is lasted about one week. The training sets contain the data generated in the first five days, while the both of validation and testing sets contain one day data. The training and validation datasets are used to construct the optimal transformer network by selecting the appropriate hyperparameters. Finally, after the temporal transformer model is trained, the QoS metric prediction results are obtained by using the testing dataset.

B. Temporal Transformers

The base of our proposed temporal transformer lies in the transformer encoder architecture which was initially proposed in 2017 for machine translation tasks [26] [27]. However, we do not use the decoder part of the base transformer for the following reasons. Firstly, the decoder module in the transformer architecture is suitable when the output sequence length is not predefined such as for generative tasks e.g., machine translation in Natural Language Processing (NLP) or summarization tasks. In contrast, in this work, the task is to predict the future throughput, PDR, PLR or latency in defined time steps. Secondly, using only the encoder part makes the proposed work suitable for solving several types of problems for IoT applications, such as classification, regression and generative tasks. Finally, the main purpose of the proposed temporal transformer is to learn the short as well as the long-term dependency of the Throughput, PDR, PLR and latency with the time domain. Thus, in our case, the temporal transformer consists of temporal inputs, positional embedding and encoder modules, while the QoS prediction will be the final output.

1) Input and Output of the Temporal Transformer: As mentioned earlier, we are solving both the univariate and multivariate QoS prediction. Therefore, the input to the transformer in these two cases will be different according to the number of the sequential values to be predicted, as described in Section III. For the temporal transformer input, a rolling window strategy is applied for the QoS metric prediction. In case of a univariate prediction, the individual sequence of either throughput, PDR, PLR or latency is taken as series. In contrast for the multivariate prediction, all possible features along with their timestamps are inserted as series input. Following, the series are divided into a number of observations with a length that is specified by the selected window size and they are shifted iteratively with a step size of 1.

Fig. 2 illustrates the process of sampling the univariate input. There are two parameters that are used to control the rolling window strategy: i) the rolling window size which is 8, as each of the rolling window sample has a length of 8 data samples; ii) the number of steps to be forecasted which is basically a forecast horizon, which in the particular example is 3. Given the rolling window samples as an input to the temporal transformer, the model can predict the QoS metrics of the forecast horizon based on the windows of the previous samples. It is to be noted that the window size and forecast horizon parameters used in Fig. 2 were selected for illustration purposes.

In the above example, a univariate prediction is performed. This means that if throughput is the targeted QoS metric to be predicted, the rolling window samples will contain only throughput series along with their timestamps. In case of multivariate prediction, the throughput will be predicted based on the previous time steps of all involved features, namely the total transmitted messages, total received messages, PDR, PLR, latency and throughput itself. This means that the windowing samples are created using multiple features. However, the output generated by the transformer model will be the forecast throughput value. The same procedure will be applied for other QoS metric predictions, such as, PDR, PLR and latency.

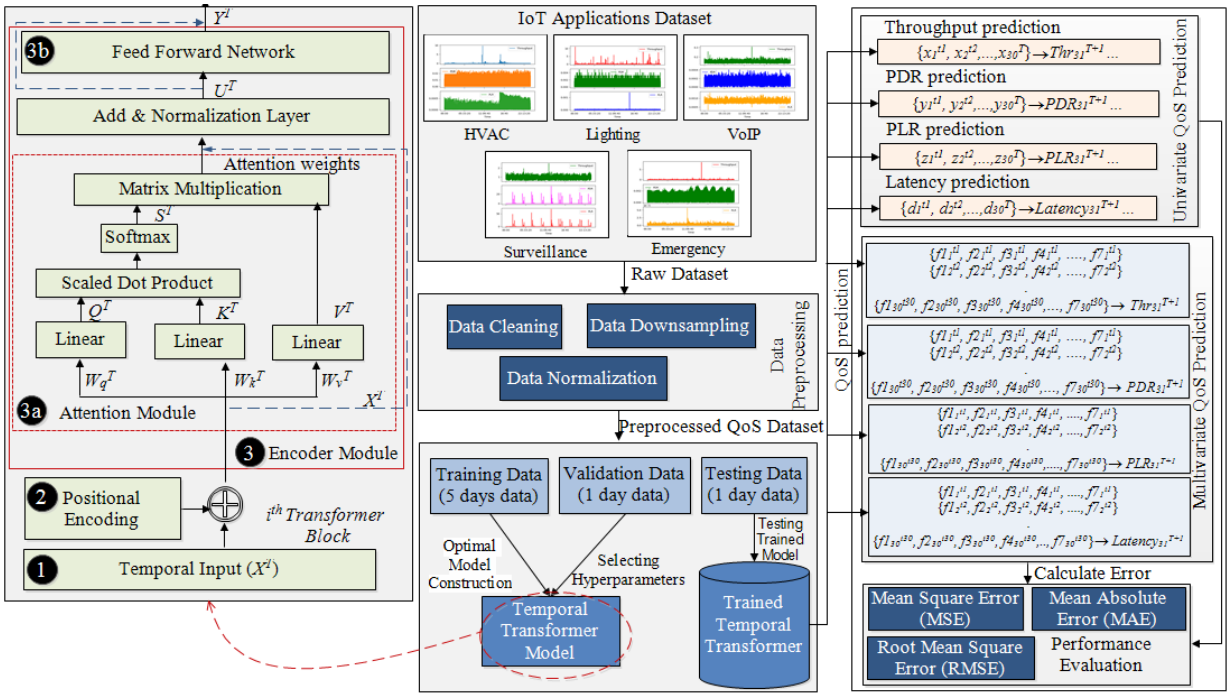


Fig. 1. Overview of Proposed System for QoS Prediction

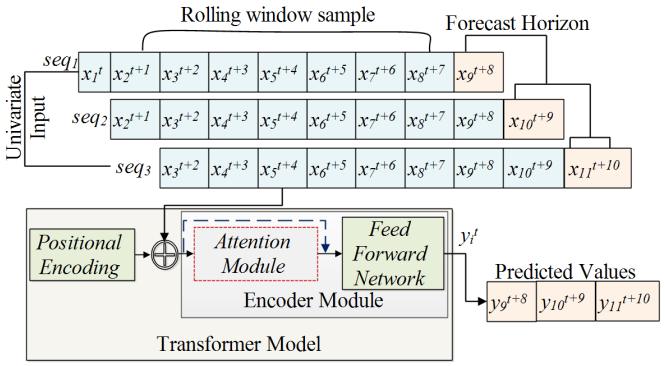


Fig. 2. Overview of univariate input and output of prediction

2) QoS Positional Encoding: The position and order of the input sequence are very important elements for the QoS prediction. Therefore, RNNs (such as LSTM) take the order of sequence inherently. The transformer on the other hand, lies on the attention mechanism in order to learn the long-term dependencies and to speed up the training time. In the attention mechanism, the attention scores are computed for all of the time steps as we will discuss in the next subsection. In case the time steps are not distinguished, the attention scores will be the same for all of the time steps. Hence, we need to incorporate the positional information of the time steps before giving the input to the transformer.

The positional encoding is the dimensional vector generated for each time step that describes the position information in the input sequence. In this work, we applied the sinusoidal positional encoding because the positional encoding provided by this scheme is fixed for each time step and no additional weights are required to be trained. The sinusoidal encoding is described as follows:

$$PE_{pos,2i} = sin(\frac{pos}{10000^{(2i/d_{mod})}}), 0 < pos < N-1 \quad (6)$$

$$PE_{pos,2i+1} = cos(\frac{pos}{10000^{(2i+1/d_{mod})}}), 0 < pos < N-1 \quad (7)$$

where PE denotes the positional encoding. pos is the position index of the time step of the input sequence and its range lies between 0 and N, which is the length of the input sequence. $2i$ represents the even dimensions of d_{mod} and $2i+1$ denotes the odd dimensions of the d_{mod}, which is the dense vector of each input time step provided by the input layer.

The positional encoding of each input sequence is added position-wise with the output of the input layer as shown in Fig. 2. This is then passed to the encoder module of the temporal transformer.

3) Encoder Module: The encoder module consists of a stack of encoders, and all are identical to each other in term of their architecture. The input of the encoder is firstly passed to the multi-head attention module that looks at the QoS values such as x_1^t and x_2^{t+1} in the input sequence seq_1 as shown in Fig. 2. It then provides the attention scores between these two QoS values and continues with the same way for other QoS values in all other input sequences. These attention scores are forwarded to the Add & Normalization layers, as shown in Fig. 1. These layers are used to stabilize the hidden states dynamics of the network and to reduce the training times. Finally, the output of the normalization layer is fed to the feed forward network. Each of the layers and sub-layers in the

encoder module also have residual connections. We provide more details for the encoder module, in the rest of this Section.

a) Multi-head attention: The main part of the transformer architecture is the Multi-Head Attention (MHA) mechanism. The attention is based on the scaled dot product that is used to compute the weights among the throughput, PDR, PLR or latency values in the input sequence as shown in Fig. 1 and it is computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (8)$$

Traditionally, Q, K and V represent the query, key and value in the attention mechanism. In this work, Q implies a certain value of QoS such as throughput, PDR, PLR or latency within the input sequence at a specific time step. K represents another QoS value within the input sequence, and V is the impact of the relation between the two QoS values within the same input sequence at their specific time steps and positions. Finally, the d_k represents the dimension of the key.

In this work, by using the scaled dot product between Q and K, the attention scores are obtained between various QoS values and then compressed with the softmax functionality. Lastly, the matrix multiplication (dot product) with V is performed. The above described attention process is performed multiple times i.e., with a multi-head attention as shown in Eq. (9).

$$h_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \quad (9)$$

In the above equation, h_i represents the i^{th} number of attention heads, with $i \in \mathbb{R}$; W_i^Q is the linear transformation of the query of the i^{th} attention head; W_i^K is the linear transformation of the key of the i^{th} attention head and W_i^V is the linear transformation of the value of the i^{th} attention head.

Following, the concatenation of multiple attentions is done by using Eq. (10) in order to represent the importance between two QoS values in terms of their correlations.

$$MultiHead(Q, K, V) = concat(h_1, h_2, ..., h_n)W^0 \quad (10)$$

where $concat$ represents the concatenation operation of the attention heads; n denotes the total number of heads, where $n \in \mathbb{R}$, and W^0 is the linear transformation of the concatenated output.

b) Feed Forward Neural Network: Finally, the last component is the Feed Forward Network (FFN), which consists of the linear transformations and the conv1D layer with the Rectified Linear Unit (ReLU) activation function. The FFN is given as:

$$FFN(x) = Relu(0, xW_1 + b_1)W_2 + b_2 \quad (11)$$

where W_1 and W_2 are the weights; b_1 and b_2 are the biases; and x is the output of the multi-head attention which is normalized by the Add & Normalization layer. The result of the Feed Forward Network along with the output of the Add & Normalization layer provides the final prediction result using a simple Dense (output) layer.

Algorithm 1 QoS Prediction Algorithm

Input: QoS training data set such as: $\{d_1^t, d_2^{t+1}, .., d_\tau^{t+\tau}\} \subset D^{train}$, validation data set D^{val} and testing data set D^{test}
1. set $m \leftarrow build\_model(D^{train}, D^{val})$
2. set $tuner \leftarrow RandomSearch(m, obj, max_{tr}, search_{tr})$
// obj is the objective of tuner which is to increase the validation accuracy; max_{tr} are the maximum trials and $search_{tr}$ are the search trials.
3. set $model \leftarrow BestModel(tuner, num_m)$
// num_m is the number of models search by the tuner.
4. set $history \leftarrow model.fit(D^{train}, D^{val}, epochs)$
5. set $\tilde{Y} \leftarrow model.predict(D^{test})$
Output: Future values of QoS as $\tilde{Y} = \{y_1^t, y_2^{t+1}, ..., y_{h-1}^{T-1}, y_h^T\}$

C. Algorithm Description

Our proposed QoS prediction algorithm (Algorithm 1) consists of either univariate or multivariate inputs that can be a QoS dataset in form of training data, validation data and testing data. The first step is to build the transformer model using the $build\_model()$ function, which takes the training and validation data as input. Following, the random search is performed with the keras tuner to search the number of models, using the RandomSearch() function, which takes the transformer model as an object, the search objective, the max trials allowed and the number of trials per search as an input. Then, the $BestModel()$ function takes the $tuner$ object and the total number of search models by the tuner as input and it returns the best model which has the highest validation accuracy across all models given by the $RandomSearch()$ function. Lastly, the best selected model is trained for a specific number of epochs using the $fit()$ function and the final prediction of the QoS values are provided as $\tilde{Y}$ using the $predict()$ function.

Algorithm 2 depicts the temporal transformer model and it consists of the three main modules described above: 1) **INPUT_EMBEDDING**, which takes as an input the training dataset, the sequence length of the input and the dimension used to represent the input sequence vector. This module is used to take the input into a specific tensor shape for the transformer along with providing the positional encoding of the time series input as well. In this module, firstly the input layer is applied, which instantiates a tensor for the temporal input sequence of the training dataset so that the input sequence is passed to the transformer model. Following, the $positional\_encoding()$ function provides the position value for each of the input in the input sequence and lastly the Add() layer of keras is used to provide the addition of the input along with their position values. This layer also returns as an output the emb_{res}, which are the embedding results. 2) **ENCODER_MODULE** consists of two main procedures namely, Multi-Head Attention and Feed Forward Network. The MHA procedure is from line 4 to line 7 and the FFN procedure is from line 8 to line 10. In MHA, firstly, the normalization layer is applied to normalize the embedding results, which are passed to the next layer which is the $MHA()$ layer that

Algorithm 2 Temporal Transformer Algorithm

INPUT_EMBEDDING(D^{train}, pos, dim)
// D^{train} is the training dataset instances, pos is the input sequence length and dim is the dimension representation.
1. set $input \leftarrow Input\_Layer(D^{train})$
2. set $pos \leftarrow Positional\_encoding(pos, dim)$
3. set $emb_{res} \leftarrow Add(input, pos)$
Output: emb_{res} ▷ Module 1

ENCODER_MODULE(emb_{res}, h_s, num_h,d_{rate}, fil, k_s, act)
// emb_{res} is the output of Module 1; h_s is the size of the head; num_h is the number of heads used; d_{rate} is the dropout rate; fil is the number of filters; k_s is the kernel size and act is the activation function.
4. set $x \leftarrow layer\_norm(emb_{res})$
5. set $x \leftarrow MHA(h_s, num_h, d_{rate}, x)$
6. set $x \leftarrow dropout(x)$
7. set $res \leftarrow x{+}input$ ▷ MHA

8. set $x \leftarrow layer\_Norm(res)$
9. set $x \leftarrow layer\_Conv1D(fil, k\_s, act, x)$
10. set $x \leftarrow dropout(d\_rate, x)$
Output:(x, res) ▷ Module 2

OUTPUT_MODULE(x, res)
// x is the output of Module 2 and res is the results of MHA module within the Module 2.
11. set $x \leftarrow layer\_GlobalAvgPooling1D(x)$
12. set $x \leftarrow layer\_Dense(x)$
13. set $x \leftarrow Add(x, res)$
14. set $x \leftarrow layer\_Norm(x)$
Output: x ▷ Module 3

also takes as an input the size of the head, the number of heads and the dropout rate and it returns the attention scores. Following the dropout function is applied using the dropout layer of keras and then the residual connection is computed by adding the output from the dropout layer with the initial input. Next, is the FFN which takes as input the residual connection values res and it passes them to the normalization layer. The results of the normalization layer along with the filters, kernel dimensions and activation function are passed to the Conv1D layer and the final dropout is performed. 3) **OUTPUT_MODULE** is used to provide the final prediction of the dataset. It takes the previous layer output i.e., x along with the residual connection value i.e., res as an input. Firstly, the x is passed to the $GlobalAvgPooling1D()$ layer, which is used specifically for the temporal data and it takes the average among all time steps. Then, the output is passed to the Dense() layer, the Add() layer, and the layer_norm() functions, in order to get the predicted values of QoS as an output.

D. Complexity Analysis

Proposition 1: The computational complexity of Algorithm 1 is $O(n^2d)$.

Proof: Line 1 of Algorithm 1 uses the $build\_model()$ function, which is the temporal transformer model and its time complexity is $O(n^2d)$ as it is represented and proved by the *proposition 3*. Following, line 2 takes $O(n)$ as $RandomSearch()$ searched all n number of models for the worst scenario and line 3 takes a constant amount of time i.e., $O(1)$. Next, the $model.fit()$ function in line 4 takes $O(t)$ time in the worst case, where t represents the length of the training dataset which is always more than the validation dataset. Lastly, line 5 predicts the QoS for a given testing dataset in $O(n)$ times. Hence, the overall complexity of Algorithm 1 is: $O(n^2d) + O(n) + O(1) + O(t) + O(n) = O(n^2d)$.

Proposition 2: The computational complexity of INPUT_EMBEDDING is $O(nd)$.

Proof: In the *INPUT_EMBEDDING* module of Algorithm 2, line 1 is a simple assignment statement, as the input layer is used to instantiate the tensor of size D^{train} and it takes $O(1)$. The computational complexity of line 2 depends on the length of the input sequence say n and the dimension representation of the input sequence say d and thus, it takes $O(nd)$. Lastly, line 3 is performing an addition operation using the Add() layer. Its complexity depends on the number of input sequences and the length of tensor provided by line 1. Since the $Add()$ layer takes as input a list of tensors, which all have the same shape, and returns a single tensor, the number of input sequences and the length of tensor provided by line 1 are of the same length and thus the complexity is $O(n)$. Accordingly, the overall time complexity of *INPUT_EMBEDDING* module is linear i.e., $O(1) + O(nd) + O(n) = O(nd)$.

Proposition 3: The computational complexity of ENCODER_MODULE is $O(n^2d)$.

Proof: The computational complexity of the encoder module depends on the MHA and FFN. The complexity of MHA procedure is $O(n^2d)$. Line 4 is the normalization of the previous layer and takes $O(n)$. Line 5 takes $O(n^2d)$ since it performs the dot product in the self attention mechanism of an n by d matrix multiplied by a d by n matrix. resulting in an $O(n^2d)$ complexity. Lines 6 and 7 takes $O(n)$ time each because line 6 is applying a dropout operation to n number of neurons and line 7 is performing an addition operation which is performed in $O(n)$ time. Next, lines 8-10 depend on the number of filters, kernel size and previous layer outputs and thus, in the worst case scenario these lines will exhibit a complexity of $O(n) + O(nd) + O(n) = O(nd)$. Lastly, we will have N number of encoder modules which are executed in parallel to perform the computations. Hence, the overall complexity of *ENCODER_MODULE* is $O(n^2d) + O(nd) = O(n^2d)$.

Accordingly, the overall complexity of the proposed temporal transformer model depends on the complexity of its three modules. As we have proved, module 1 gives a complexity of $O(nd)$ and module 2 gives a complexity of $O(n^2d)$. The *OUTPUT_MODULE* (module 3) presents a linear complexity of $O(n)$ as all layers in lines 11-14 depend on the length of the output of the previous layer and perform basic operations such as average, activation, addition and normalization

which take in the worst case $O(n)$ time. Thus, the time complexity of Algorithm 2 is represented in terms of n as: $O(nd) + O(n^2d) + O(n) = O(n^2d)$.

E. Implementation Cost

The implementation cost of the proposed framework can be divided into three parts; the model infrastructure, the data support and the deployment cost. The model infrastructure cost includes the physical resources required to run the proposed model at the edge and provide timely and accurate QoS predictions. A commodity computer has sufficient computing power, memory, and storage for the inference, data preprocessing and the parameter storage of the temporal transformer. Similar is also the answer for the metering process in the UDP server that collects the information of packet exchanges in the network. Both services can be deployed and run in the same commodity computer. Regarding the networking requirements, these are limited to the transfer of some kilobytes of monitoring data per minute between border routers and the UDP server. This is an insignificant overhead in the edge infrastructure.

Data support costs concern the costs of developing a data pull script with the corresponding preprocessing modules such as data cleaning, down-sampling and normalization. This is a one-time cost incurred by a data engineer to develop an extract-transform-load pipeline in order to extract the measurements and provide them in the appropriate format to the temporal transformer. The deployment cost concerns the labor cost of a data engineer to deploy the model in the commodity computer that runs at the edge. This labor cost also includes all the configurations, testing and preparation steps needed to install and run the operating system, various software, the python modules, the dependencies and establish the communication with the rest of the infrastructure.

To add up the three types of costs and calculate the total implementation cost, we begin with the cost of model infrastructure that comes down to a commodity computer which is approximately \$1.000 [1]. In the implementation cost we should also add the electricity cost which is approximately \$160.16 per year. [2] and the maintenance cost which ranges from \$40 to \$90 per hour for the work of a technician [3]. The data support cost is significantly higher due to the work of the data engineer. We estimate a senior data engineer can implement the proposed model, the data preprocessing and the extract-transform-load process in one man-month which results in a cost close to \$9.649 [4]. The deployment cost is reduced to the manual work of a network engineer that will integrate and run the python scripts in the edge infrastructure. This work is calculated to last approximately one week and costs \$1.665 [5]. Last but not least, we should not underestimate the training cost of the temporal transformer. Google cloud incurs a charge that begins from \$0.218 per training hour for a general purpose machine with 4GB of RAM. [6]

[1]https://www.amazon.com/Workstation-Pc/s?k=Workstation+Pc

[2]https://www.pcmag.com/how-to/power-hungry-pc-how-much-electricity-computer-consumes

[3]https://www.thumbtack.com/p/computer-repair-prices

[4]https://www.indeed.com/career/data-engineer/salaries

[5]https://www.indeed.com/career/network-engineer/salaries

VI. Performance Evaluation

A. Model Implementation and Frameworks

1) Evaluation setup: Each dataset is zero-mean normalized and standardized. Under the time series prediction settings, we forecast the four following QoS metrics as: (i) Throughput; (ii) PDR; (iii) PLR and (iv) Latency. Additionally, the prediction is performed in two time series settings as: (i) Univariate and (ii) Multivariate. The window size for both settings is set to be 30. The total data generation lasted seven days. All of the five datasets are divided into three parts as follows: i) training dataset, which contains the first five days of data; ii) validation dataset, which contains the sixth day data and iii) testing dataset which contains the seventh day data. All of the models were trained and tested on two compute clusters offered by Compute Canada namely, Cedar and Beluga. For the Beluga cluster, we trained, validated and tested the models on a NVIDIA V100 with 16GB GPU and for the cedar cluster, we utilized the NVIDIA P100 with 16GB GPU respectively.

2) Evaluation Metrics: We used three metrics to measure the prediction performance of our proposed method against all of the baseline methods as described below, namely the Root Mean Square Error (RMSE), Mean Square Error (MSE) and Mean Absolute Error (MAE). For all of these metrics a smaller value indicates a better prediction performance. MAE is the sum of the absolute value of differences between the actual QoS values represented as y_j and predicted QoS values represented as $\overline{y}_j$, divided by the total number of QoS predictions as defined below:

$$MAE = \frac{1}{n}\sum_{n=1}^{n}|y_j - \overline{y}_j| \tag{12}$$

MSE is an average of the squared errors between the predicted QoS values and the targeted (actual) QoS values divided by the total number of QoS predictions. RMSE is the square root of MSE as given below:

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(y_j - \overline{y}_j)^2 \tag{13}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{n=1}^{n}(y_j - \overline{y}_j)^2} \tag{14}$$

3) Baselines: For comparison purposes, we evaluate our proposed model against the most popular deep learning models that are appropriate for time series prediction, as presented in Section II.B. The baseline models are the following: i) **Multi-layer Perceptron (MLP)** is a feed forward network, which consists of an input layer, an output layer and multiple hidden layers. This network is fully connected, which means the identical units in each layer called neurons are connected to every neuron in the next layer in a network, ii) **stacked LSTM** is composed of multiple LSTM layers that are stacked in a

[6]https://cloud.google.com/vertex-ai/pricing

TABLE III
HYPERPARAMETERS USED IN ALL METHODS FOR THE UNIVARIATE THROUGHPUT PREDICTION ACROSS ALL DATASETS

Models	Hyperparameters	Min. value	Max. value	Best selected value				
				HVAC	VoIP	Lighting	Emergency	Surveillance
MLP	Number of neurons	8	512	64	392	288	40	504
	Dropout rate	0	0.5	0.1	0.3	0.2	0.1	0.1
	Learning rate	1e-2	1e-4	0.01	0.0001	0.001	0.001	0.0001
Stacked LSTM	Number of neurons	8	128	24	72	104	96	16
	Dropout rate	0	0.5	0.4	0.001	0.1	0.4	0.4
	Learning rate	1e-2	1e-4	0.001	0.0001	0.01	0.0001	0.0001
	Number of layers	2	6	3	2	4	3	5
Bidirectional LSTM	Number of neurons	8	512	32	16	24	64	352
	Dropout rate	0	0.5	0.5	0.4	0.1	0.1	0.2
	Learning rate	1e-2	1e-4	0.01	0.01	0.01	0.001	0.001
Temporal Transformer	head size	4	256	28	32	128	4	2
	Number of heads	4	32	6	18	24	4	3
	Dropout rate	0	0.5	0.2	0.5	0.2	0.5	0.2
	Number of transformer blocks	4	16	16	8	4	4	2
	Linear layer neurons	4	128	96	84	52	64	116
	Linear layer dropout	0	0.5	0.2	0.5	0.2	0.5	0.2
	Filter dimensions	4	64	96	28	52	64	8
	Number of attention layers	1	15	4	5	2	2	3

multi-layer and a fully connected architecture. The stacking of LSTM is done in such a way that the result of each LSTM layer is used as an input for the subsequent LSTM layer in the stack, iii) **Bidirectional LSTM** is a combination of a bidirectional RNN with an LSTM network. In this particular architecture, the input sequence is processed in a forward as well as in a backward direction in each of the network layers. The details of how MLP, stacked LSTM and bidirectional LSTM work is provided in the Appendix of this document. iv) **LSTNet** is a multivariate time series prediction framework proposed in [23], that models the short and long-term temporal patterns with Deep Neural Networks. This particular model uses the Convolution Neural Network and the Recurrent Neural Network along with the auto regressive component for the extraction of the short-term local dependency patterns among variables and the long-term patterns for time series patterns. To compare our proposed framework with this existing LSTNet model, we have used the same configuration that the authors provided in term of their architecture.

For the univariate prediction, we used the MLP, stacked LSTM and bidirectional LSTM as our baseline methods and for the multivariate prediction, we used the stacked LSTM, bidirectional LSTM and LSTNet as baseline methods. We have used only one method from the literature i.e., LSTNet because to the best of our knowledge there is no other existing method that can provide the QoS prediction, while handling the long-term dependencies at the same time in an edge computing environment. In contrast, LSTNet was designed specifically for time series forecasting while providing a multivariate prediction. Furthermore, the MLP did not provide good accuracy in case of a multivariate prediction and we have excluded it for the second part of the evaluation. Finally, it should be noted that we have also considered some traditional time series methods such as Autoregressive Integrated Moving Average (ARIMA), Simple Exponential Smoothing (SES) and Prophet. However, all these forecasting techniques presented a poor accuracy performance and therefore, we decided not to include them in our performance evaluation.

4) Hyper-parameter Tuning: For the hyper-parameter search and tuning, we performed a random search of the search space using the keras tuner. In particular, for all methods and all datasets, the input length of the input time series sequence is set as 30. In other words, the rolling window sample is set to be 30, which we believe is a sufficient value for long-term prediction. The hyperparameters that were searched for the baseline models consist of the number of neurons, dropout rate, learning rate and number of layers. For the stacked LSTM, the number of neurons were selected from the range 8 to 128 with a step of 8. For the MLP and bidirectional LSTM, the number of neurons were selected between 8 and 512, with the same step. For the dropout rate, the value is taken from the $\{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$ range with the default value set to be 0.5 whereas, the learning rate was selected from the $\{$1e-2, 1e-3, 1e-4$\}$ set for all baseline methods. Additionally, the number of layers was selected between 2 to 6 for the stacked LSTM. Lastly, for the baseline method found from the literature i.e., LSTNet, we used the already provided hyperparameters in [23].

For the proposed temporal transformer model, we have fine-tuned the following hyperparameters: head size, number of heads, dropout rate, number of transformer blocks, number of neurons for the linear layers, dropout rate for the linear layers, filter dimensions and number of attention layers. The search space set for each of the hyperparameters is set as follows. For the head size, the minimum value was set at 4 and the maximum at 256 with a step size of 4. For the number of heads, an optimal value was found within the range of 4 to 32 with a step of 2. The dropout rate was selected between 0 and 0.5 with a step size of 0.1 and the number of transformer blocks was chosen from the range $\{4, 8, 12, 16\}$. For the linear layer, which is included as part of the transformer architecture, the number of neurons was selected between 4 and 128 with a step of 8 and their dropout rate was chosen between 0 and 0.5 with a step of 0.1.

Regarding the neural network optimizer, the Adam optimizer was used for all baseline methods and for our trans-

TABLE IV
STATISTICAL CHARACTERISTICS OF QOS DATASETS FOR ALL IOT APPLICATIONS

QoS	Throughput			PDR			PLR			Latency		
Metrics	Mean	S.D	Median	Mean	S.D	Median	Mean	S.D	Median	Mean	S.D	Median
HVAC	0.253958	0.204743	0.190762	0.331391	0.390711	0.000033	0.232914	0.237775	0.124958	0.044852	0.058928	0.029333
VoIP	0.323644	0.108589	0.304021	0.532645	0.207119	0.548382	0.501071	0.060243	0.494923	0.004661	0.001512	0.004562
Lighting	0.164938	0.185021	0.094540	0.100024	0.226347	0.000000	0.038439	0.101123	0.000000	0.041879	0.075346	0.011554
Emergency	0.061513	0.061542	0.043207	0.258011	0.226774	0.173908	0.134196	0.127102	0.085104	0.066475	0.073212	0.031021
Surveillance	0.337204	0.167844	0.330451	0.290765	0.126342	0.294597	0.079721	0.128798	0.035209	0.380338	4.870949	0.000468

former model. As random search is performed to select the best values for the hyperparameters, the total number of trials considered for this search is 5 with an epoch value of 100. Finally, keras tuner selected the best trial that gave the best set of hyperparameters for all of the application datasets. Table III summarizes the hyperparameters and the best selected value from keras tuner for all five application datasets. It is to be noted that the same hyperparameters with the same corresponding search range were used for both univariate and multivariate prediction. However, due to space constraints and illustration purposes, Table III provides the hyper-parameter tuning of the univariate prediction. The hyper-parameter tuning for the multivariate prediction is provided in the Appendix of this document.

B. Explanatory Data Analysis

In this part, we provide the explanatory analysis of the applications' datasets along with their properties. The statistical properties of each dataset are presented in Table IV. In Fig. 3, the density plots for each of the QoS metrics within each dataset are also presented. The density plots are used to observe the distribution of the datasets with a continuous interval. For the emergency application, we have a positively skewed distribution for all four QoS metrics and this is because the mean in the datasets of throughput, PDR, PLR and latency are greater than their median values. For the HVAC application, the throughput, PLR and latency also exhibit a skewed distribution and more specifically a right skewness however, PDR presents a multi-modal distribution as it has three different peaks. For the lighting application, the throughput and latency both datasets are rightly skewed, but PDR and PLR are both multi-modal datasets. For the surveillance application, the throughput is multi-modal with more than 12 modes, PDR exhibits a normal distribution, PLR and latency both are rightly skewed. Lastly, for the VoIP application, we have a normal distribution for all of the three QoS metrics i.e., throughput, PDR and PLR, however, the latency dataset is slightly skewed towards right as the mean in latency data i.e., 0.004661 is slightly higher than the median value i.e., 0.004562.

C. Results

1) Univariate time series forecasting: For the univariate TSF, we included a representative range of the 5 IoT datasets to ensure the diversity and applicability of our transformer model with respect to the dimensionality and length of the time series samples, as well as the number of samples. Table V shows the MAE, MSE and RMSE achieved by the baseline methods and transformer model. As it can be seen, the transformer model worked well for the throughput prediction as compared to the other models across all datasets. We have also plotted the MSE and MAE values of all methods in Figs. 4 and 5 to better illustrate the results. It should be noted that the y axis of both figures goes from large values towards small values and we also include the data points for the transformer model to better position its efficiency.

Our first observation, is that all applied models give the least values for all error metrics for the emergency application followed by the lighting application. In contrast, for the surveillance application, the models achieve higher error values followed by the VoIP and HVAC applications. The main reason for having less accurate results for surveillance, VoIP and HVAC applications is that the datasets of these applications contain several extreme values also known as outliers. Hence, as deep learning models do not learn easily such extreme values, such behavior can cause performance degradation. We can also detect the outliers from the statistical properties of the datasets as shown in Table IV. For instance, for the surveillance application, the throughput dataset has a standard deviation value of 0.167844 and a mean value of 0.337204. This is because the more extreme outliers exist in the dataset, the more the standard deviation is affected with respect to the mean value. Similarly, for the VoIP and HVAC applications, the standard deviations are also highly affected as they appear to be 0.108589 and 0.204743 respectively, while their corresponding mean values are 0.323644 and 0.253958.

In contrast, for the lighting and emergency applications, such kind of extreme values appear more frequent and cannot be considered outliers, as the outliers by their nature are rare events that happen in a dataset. Therefore, the deep learning models adapt better to those frequent extreme events to some extend and produce better performance for the lighting and emergency datasets as compared to the other application datasets.

To better understand which model is able to capture this behavior more accurately, we shift our focus on Figs. 4 and 5. It becomes apparent that the temporal transformer provides the least error in the prediction of throughput values as compared to all other algorithms and for all datasets. This happens for the following two reasons: (i) For a longer input window size, also called input sequence length, i.e., 30 in this work, the prediction ability of the deep learning models decreases, which leads to a rise in the error metrics. This also reveals a real problem faced by the time series forecasting. However, our transformer model is well suited for solving such long

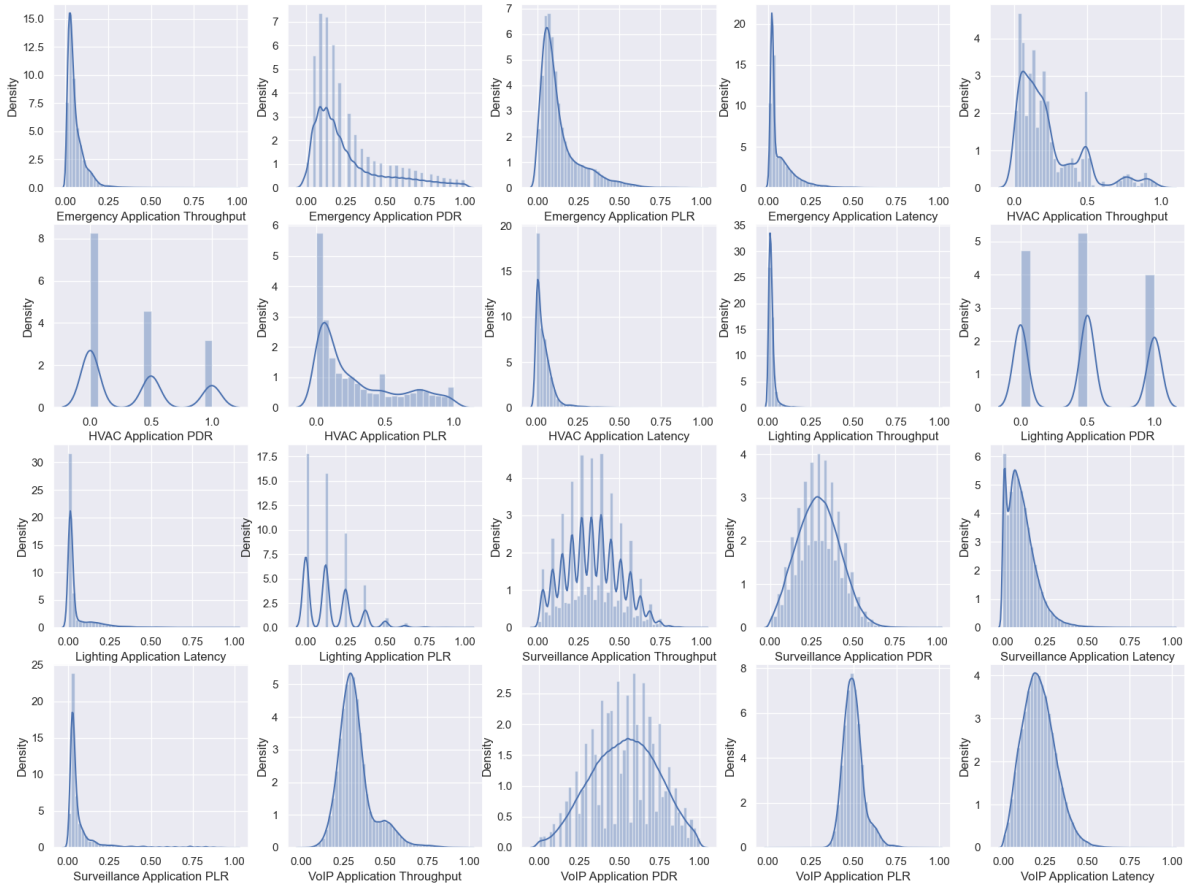


Fig. 3. Probability distribution plots of QoS data for all IoT applications

TABLE V
UNIVARIATE FORECASTING RESULTS FOR THROUGHPUT, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	MLP			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	3.91e-3	2.11e-5	4.60e-3	3.56e-3	2e-5	4.48e-3	4.57e-3	2.82e-5	5.31e-3	**2.57e-3**	**1.17e-5**	**3.42e-3**
VoIP	4.20e-3	3.49e-5	5.91e-3	2.89e-3	1.53e-5	3.92e-3	2.87e-3	1.52e-5	3.9e-3	**1.67e-3**	**4.27e-6**	**2.07e-3**
Lighting	1.62e-3	6.13e-6	2.47e-3	1.83e-3	6.67e-6	2.58e-3	1.87e-3	6.52e-6	2.55e-3	**9.63e-4**	**1.49e-6**	**1.22e-3**
Emergency	1.29e-3	6.86e-6	2.62e-3	1.3e-3	6.88e-6	2.62e-3	1.28e-3	6.87e-6	2.62e-3	**1.43e-4**	**30e-8**	**1.73e-4**
Surveillance	6.22e-2	6.92e-3	8.32e-2	2.76e-2	2.03e-3	4.5e-2	1.28e-2	7.74e-4	2.78e-2	**1.26e-2**	**7.72e-4**	**2.78e-2**

TABLE VI
UNIVARIATE FORECASTING RESULTS FOR PDR, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	MLP			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	4.41e-3	2.60e-5	5.10e-3	4.40e-3	2.65e-5	5.15e-3	4.35e-3	**2.54e-5**	**5.04e-3**	**4.15e-3**	2.73e-5	5.23e-3
VoIP	2.71e-5	1.0e-9	3.16e-5	2.31e-5	1.0e-9	2.93e-5	2.43e-5	9.47e-10	3.08e-5	**2.31e-5**	**8.55e-10**	**2.93e-5**
Lighting	**2.47e-4**	1.40e-7	3.74e-4	2.64e-4	1.44e-7	3.80e-4	2.65e-4	1.37e-7	3.70e-4	2.63e-4	**1.37e-7**	**3.70e-4**
Emergency	1.32e-4	3.0e-8	1.73e-4	1.32e-4	3.0e-8	1.73e-4	1.31e-4	2.9e-8	1.71e-4	**8.06e-5**	**9.0e-9**	**9.73e-5**
Surveillance	3.20e-5	2.0e-9	3.89e-5	2.05e-5	1.0e-9	2.65e-5	3.40e-5	1.96e-9	4.43e-5	**2.0e-5**	**6.96e-10**	**2.53e-5**

TABLE VII
UNIVARIATE FORECASTING RESULTS FOR PLR, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	MLP			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	2.76e-5	1.22e-9	3.50e-5	2.36e-5	8.37e-10	2.89e-5	2.72e-5	1.16e-9	3.41e-5	**2.27e-5**	**8.10e-10**	**2.84e-5**
VoIP	2.39e-5	1.0e-9	3.32e-5	2.61e-5	1.30e-9	3.60e-5	1.77e-5	1.0e-9	2.42e-5	**1.70e-5**	**5.82e-10**	**2.41e-5**
Lighting	**2.74e-6**	3.74e-11	6.12e-6	3.80e-6	3.37e-11	5.81e-6	3.86e-6	3.32e-11	5.76e-6	3.75e-6	**3.32e-11**	**5.76e-6**
Emergency	1.83e-12	5.40e-24	2.32e-12	2.16e-12	7.5e-24	2.74e-12	1.88e-12	5.3e-24	2.32e-12	**1.80e-12**	**5.2e-24**	**2.30e-12**
Surveillance	2.33e-3	5.46e-6	2.34e-3	2.67e-5	1.65e-9	4.06e-5	4.76e-4	2.45e-7	4.96e-4	**2.57e-5**	**1.39e-9**	**3.73e-5**

TABLE VIII
UNIVARIATE FORECASTING RESULTS FOR LATENCY, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	MLP			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	2.47e-02	9.07e-04	3.01e-02	2.60e-02	9.26e-4	3.04e-02	2.31e-02	7.27e-04	2.70e-02	**3.36e-03**	**1.56e-05**	**3.95e-03**
VoIP	9.69E-01	5.58e+02	2.36e+01	1.34e-03	2.75e-06	1.66e-03	1.42e-03	3.14e-06	1.77e-03	**1.27e-03**	**2.28e-6**	**1.51e-03**
Lighting	4.45e-02	4.09e-03	6.40e-02	4.44e-02	4.04e-03	6.36e-02	4.4166e-2	4.04e-03	6.36e-02	**2.34e-02**	**1.09e-03**	**3.30e-02**
Emergency	4.63e-02	3.73e-03	6.10e-02	4.90e-02	4.07e-03	6.38e-02	4.84e-02	4.06e-03	6.37e-02	**3.86e-02**	**2.67e-03**	**5.17e-02**
Surveillance	2.74e-04	1.20e-07	3.46e-04	2.76e-04	2.35e-06	1.53e-03	2.63e-04	1.15e-07	3.39e-04	**1.54e-04**	**3.16e-08**	**1.78e-04**

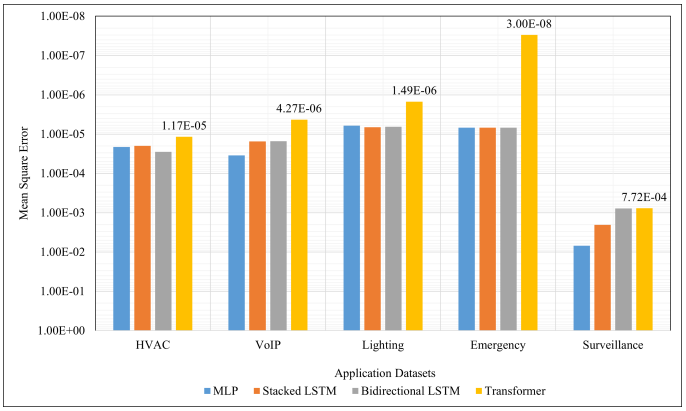


Fig. 4. MSE of univariate throughput prediction across all datasets

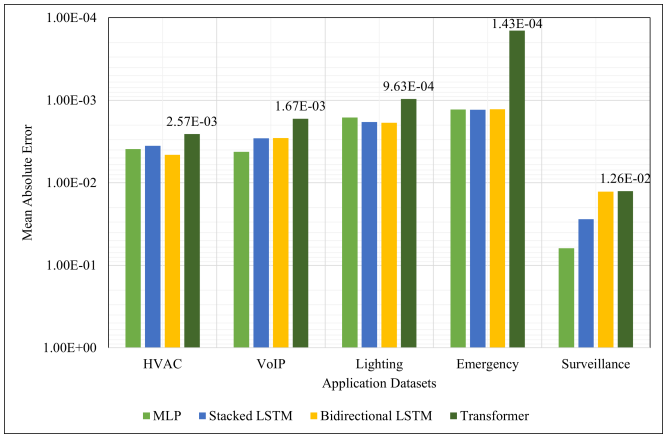


Fig. 5. MAE of univariate throughput prediction across all datasets

sequence dependency problems and thus, exhibiting a superior performance for the throughput prediction; (ii) The attention mechanism in the transformer architecture allows to learn the relation of temporal and positional features to specific throughput values at each timestamp and emphasizes on their importance.

Following, the results for the PDR prediction are presented in Table VI. As it can be seen, once more the transformer model performed better for almost all of the applications. Nonetheless, there are two applications for which other models also provide promising results and these are: (i) for the HVAC dataset the bidirectional LSTM provides the least MSE and RMSE values as 2.54e-5 and 5.04e-3. The reason that the transformer could not match these values are probably because our model tried to learn the outliers and this had an impact on the relation between the features as provided by the attention module of the transformer, which can lead to higher errors than the bidirectional LSTM model. At the same time, MSE and RMSE are more sensitive to the outliers as the squaring of high errors will lead to lower performance; (ii) for the lighting application, MLP provides the least MAE value i.e., 2.47e-4, however, its MSE and RMSE are also affected by the outliers. Nonetheless, the impact of the outliers for the particular application was less on the transformer model which led to the least attained MSE and RMSE values.

Next for illustration purposes, in Fig. 6 we also plot the predicted values (orange curves) and the collected true values (blue curves) for the PDR dataset of the surveillance application. In order to not further increase the length of the paper, we have just selected the surveillance application as it has more fluctuations and presents a more interesting behavior for the QoS metrics prediction. From the figure, we notice that the PDR data is usually noisy which means that we have peaks and troughs (i.e., fall of data points in downward direction). This means that the PDR of the surveillance application is sometimes higher and sometimes very lower than the normal pattern. This is because of the exponential distribution pattern of the application and the high network contention, since the rest of the IoT devices belonging to other applications may transmit at the same time. From this, we can deduct that the peaks and troughs are not normal patterns of the dataset and therefore, it is not necessary that all peaks and troughs appear the one after another by following a specified and periodic behavior. Given this type of fluctuating dataset, we see from

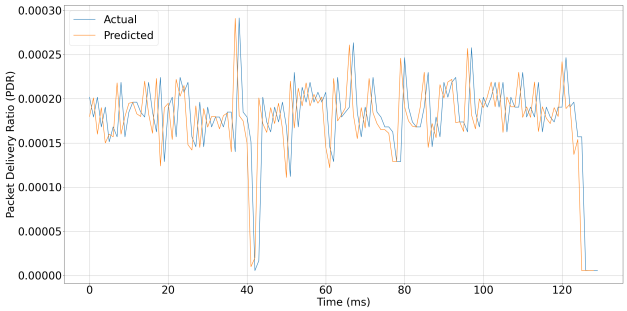


Fig. 6. PDR prediction for surveillance application

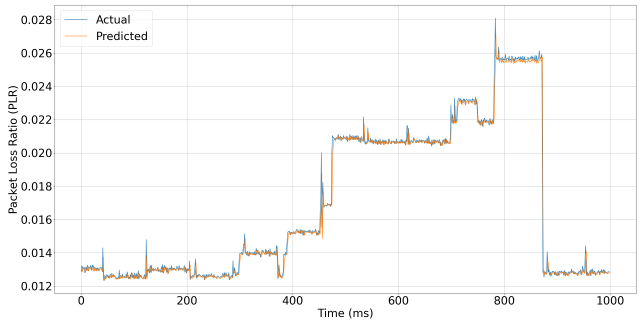


Fig. 7. PLR prediction for surveillance application

the figure that the the transformer model predicts the peaks and troughs of data adequately and this is mainly because of the attention module within the transformer that learns very well about the temporal and positional features (i.e., at which timestamp certain PDR values appear in the input sequence) of the time series dataset over the long input sequences.

Following, we provide the univariate PLR results for the five IoT applications in Table VII. As it can be seen, the transformer model provides the least error values for almost all datasets for this particular QoS metric as well. However, two particular cases are drawn from these results: 1) for the lighting application, the MLP model provides the least MAE, yet, MSE and RMSE are higher than the transformer model and the reason for such behavior is the same as the one explained for the PDR case; 2) for the emergency application, all algorithms provide the best accuracy performance with respect to the other four applications. The reason for this is that the particular dataset is not affected by outliers as the standard deviation value i.e., 0.127102 does not deviate a lot from the mean value i.e., 0.134196. Nonetheless, the transformer model provides the best performance and for these types of applications.

Once more, we plot the actual vs. predicted values for the PLR data of the surveillance application only in Fig. 7, as it has more fluctuating patterns compared to the other applications. In general, we can see that the transformer model can capture very well the general behavior of the PLR dataset. There is only just a small difference between the actual and predicted values when there are small PLR spikes as noticed at the 50ms, 150ms, 470ms, 550ms and 790ms time instances. These spikes can by attributed to high network contention time instances which can lead to an increased packet loss. Nonetheless, the transformer was able to closely follow the unusual fluctuations between the time period from 450 ms to 900 ms. This is due to the fact that the particular model can capture the time series features with long-term time dependency easily.

Following, we provide the univariate latency results for the five IoT applications in Table VIII. As it can be seen, the temporal transformer model performs better for all of the datasets and in terms of all error metrics as compared to the baseline methods. From Table VIII, we have the following observations: (1) The latency datasets of all applications are positive (right) skewed. The distribution is right skewed because of the lower bound in the dataset. So if the lower bound of the dataset is extremely low relative to the rest of the data, then this will cause the data to be skewed right. The lower bound for an application reveal that lower latency is experienced during the transmission of the packets. Furthermore, the emergency application followed by lighting and HVAC have more extreme smaller values for latency as their standard deviations is less distant from their mean value than the surveillance and VoIP applications. However, this does not affect the performance of the proposed temporal transformer model and it always outer-performs the baseline methods for all skewed datasets in term of all error metrics. (2) The second observation is that the second best model is the bidirectional LSTM as it performed well for 3 out of 5 applications after the transformer model. The reason is that the particular model is able to learn the input sequence in both forward and backward direction. However, for the proposed transformer model the dependencies among input sequence are better learned using the attention module of the model.

Overall, our proposed temporal transformer model achieves the best performance on 18 out of 20 settings for MAE, on 19 out of 20 settings for MSE, and on 19 out of 20 settings for the RMSE case. Notably, for the throughput prediction, the transformer can increase the performance by 28% for HVAC, 42% for VoIP, 41% for lighting, 89% for emergency and 2% for the surveillance applications from the second best performing model in terms of MAE. Furthermore, for the MSE, we noticed an improvement of up to 96% and for the RMSE, we noticed an improvement of up to 93%. For the PDR prediction, the transformer model enhanced the performance by decreasing the MAE by 5% for HVAC, 0.43% for VoIP, 38% for emergency and 2% for the surveillance application from the second best performing baseline method, except the lighting application in which the MLP improved the error rate by 6% in comparison to the transformers for the reasons we discussed above. Moreover, for the PLR prediction, the transformers can reduce the MAE by 2% to 4% for the four applications, but once more the MLP shows a slightly better performance for the lighting applications. Finally, for the latency predicted, the transformers provided an improvement of 85% for HVAC, 5% for VoIP, 47% for lighting, 17% for the emergency and 41% for the surveillance application than the second best performing model in term of MAE. Additionally, the proposed transformer provides 17% to 98% improvement

TABLE IX
MULTIVARIATE FORECASTING RESULTS FOR THROUGHPUT, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	LSTNet			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	8.74e-2	7.63e-1	8.73e-1	4.34e-3	2.81e-5	5.30e-3	3.38e-3	**1.17e-5**	**3.42e-3**	**3.35e-3**	1.71e-5	4.14e-3
VoIP	4.10e-2	3.91e-1	6.25e-1	4.27e-3	3.32e-5	5.76e-3	2.91e-3	1.55e-5	3.94e-3	**2.87e-3**	**1.48e-5**	**3.85e-3**
Lighting	4.46e-2	6.88e-1	8.29e-1	1.90e-3	7.18e-6	2.68e-3	1.89e-3	7.17e-6	2.68e-3	**1.85e-3**	**7.10e-6**	**2.66e-3**
Emergency	9.65e-2	1.45e-1	3.81e-1	1.64e-3	7.71e-6	2.78e-3	1.26e-3	6.72e-6	2.59e-3	**1.23e-3**	**6.67e-6**	**2.50e-3**
Surveillance	8.60e-3	9.14e-2	3.02e-1	1.28e-2	7.73e-4	2.78e-2	4.29e-3	3.65e-5	6.04e-3	**2.91e-3**	**1.55e-5**	**3.93e-3**

TABLE X
MULTIVARIATE FORECASTING RESULTS FOR PDR, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	LSTNet			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	4.37e-2	5.07e-1	7.12e-1	3.07e-4	1.16e-7	3.41e-4	3.04e-4	**1.15e-7**	**3.39e-4**	**2.94e-4**	1.31e-7	3.62e-4
VoIP	2.84e-2	3.52e-1	5.93e-1	2.64e-5	1.12e-9	3.35e-5	2.31e-5	8.58e-10	2.92e-5	**2.31e-5**	**8.57e-10**	**2.90e-5**
Lighting	2.03e-2	6.54e-1	8.09e-1	2.70e-4	1.37e-7	3.70e-4	2.65e-4	**1.37e-7**	**3.70e-4**	**2.52e-4**	1.40e-7	3.74e-4
Emergency	3.39e-2	6.58e-1	8.11e-1	1.32e-4	2.96e-8	1.72e-4	1.64e-4	4.57e-8	2.14e-4	**1.31e-4**	**2.80e-8**	**1.67e-4**
Surveillance	1.63e-2	2.11e-1	4.59e-1	2.08e-5	7.28e-10	2.70e-5	3.72e-5	2.29e-9	4.79e-5	**2.06e-5**	**7.20e-10**	**2.69e-5**

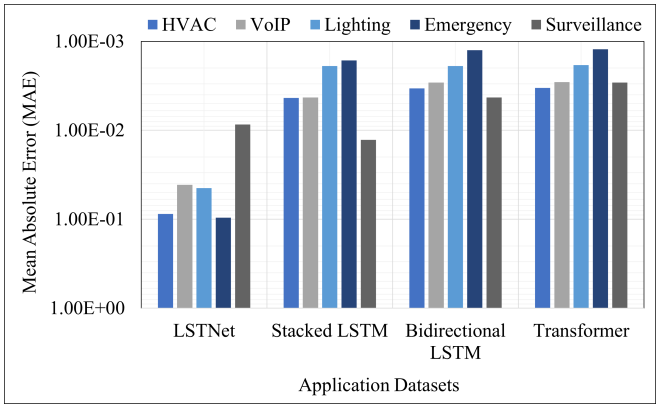


Fig. 8. MAE of multivariate throughput prediction across all datasets

in term of MSE and 9% to 85% improvement in terms of the RMSE metric.

2) Multivariate time series forecasting: In this part of the section, we present the obtained results under the multivariate setting. Regarding the multivariate throughput prediction, the prediction results are provided in Table IX. To better illustrate these results w.r.t. MAE, we plot them as well in Fig. 8. Similar to the univariate setting, the scale for MAE is logarithmic and goes from high i.e., 1.00E+00 to small values i.e., 1.00E-03. From this plot, it is shown that the LSTNet method provides the worst performance i.e., the highest MAE for all of the applications and this is because the particular method is unable to deal with the dynamic periodic patterns or the non-periodic patterns of our datasets. However, the bidirectional LSTM presents a good performance, similar to the one of our proposed temporal transformer model. Specifically, the transformer model provides 1% improvement for HVAC and VoIP application, 2% improvement for lighting and emergency applications and a noticeably 32% improvement for the surveillance application as compared to the best performing baseline method. The reason for the major improvement in the surveillance application dataset is that the surveillance application has the long-term fluctuating patterns and our transformer model is the most suitable approach for capturing and predicting this long-term behavior.

Similarly, Table X shows that the temporal transformer achieves the least MAE values for all applications in terms of PDR. However, there are two cases for which bidirectional LSTM achieves the least performance in terms of MSE and RMSE values and these are for the lighting and HVAC applications. There are several reasons for this. Firstly, such application datasets contain extreme values for specific timestamps. Secondly, the PDR data of these two applications are smaller compared to the other applications and the transformer requires a larger number of training samples compared to the other baseline methods. Thirdly, the good performance of the bidirectional LSTM can be attributed to the fact that it runs the given input sequence in two ways from past to future and future to past. Thus, it is able to better learn even for datasets that have smaller number of training samples. However, the transformers can closely follow the performance of the bidirectional LSTM even in these situations. This can be corroborated by Fig. 9, which presents the MAE metric for all applications and it can be concluded that the transformer performed consistently well, followed by the stacked LSTM for the emergency and surveillance applications and by the bidirectional LSTM for the HVAC, VoIP and lighting applications. Specifically, the proposed temporal transformers can lead to a decrease in the MAE error that ranges from 1% to 5% as compared to the second best baseline method.

Moreover, we provide the results of the PLR prediction in Table XI. Over again, the transformer model is the most dominant approach. Only for the VoIP application the stacked LSTM presents a better performance in terms of MSE and RMSE, however the transformer model provides the least MAE. This is because the stacked LSTM can also learn complicated nonlinear dependencies between time steps and between multiple time series. These types of dependencies can be easily produced when irregular network conditions are surfaced due to interference and available bandwidth reduction

TABLE XI
MULTIVARIATE FORECASTING RESULTS FOR PLR, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	LSTNet			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	5.24e-2	4.75e-1	6.89e-1	3.92e-5	2.09e-9	4.57e-5	4.73e-5	4.0e-9	6.33e-5	**3.89e-5**	**2.09e-9**	**4.57e-5**
VoIP	3.77e-2	3.87e-1	6.22e-1	2.21e-5	**3.37e-5**	**1.14e-9**	5.04e-5	5.93e-9	7.71e-5	**2.20e-5**	3.39e-5	1.15e-9
Lighting	4.03e-2	6.85e-1	8.28e-1	3.94e-6	3.33e-11	5.77e-6	4.14e-6	3.55e-11	5.96e-6	**3.84e-6**	**3.32e-11**	**5.76e-6**
Emergency	2.93e-2	6.27e-1	7.91e-1	1.90e-12	5.68e-24	2.38e-12	1.88e-12	6.27e-24	2.50e-12	**1.87e-12**	**5.64e-24**	**2.37e-12**
Surveillance	8.6e-3	1.06e-1	3.26e-1	2.44e-4	6.07e-7	2.46e-4	1.12e-3	2.0e-6	1.42e-3	**1.89e-5**	**8.34e-10**	**2.89e-5**

TABLE XII
MULTIVARIATE FORECASTING RESULTS FOR LATENCY, BEST RESULTS ARE HIGHLIGHTED IN BOLD

Methods	LSTNet			Stacked LSTM			Bidirectional LSTM			Temporal Transformer		
Metrics	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE	MAE	MSE	RMSE
HVAC	8.40e-2	2.81e0	1.66e0	2.56e-2	9.28e-4	3.05e-2	2.32e-2	7.81e-4	2.79e-2	**2.27e-2**	**7.33e-4**	**2.71e-2**
VoIP	2.41e-2	2.68e-1	5.052e-1	1.35e-3	2.853e-6	1.69e-3	1.35e-3	2.85e-6	1.69e-3	**9.24e-4**	**1.18e-6**	**1.09e-3**
Lighting	7.13e-2	1.52e0	1.23e0	3.55e-2	2.56e-3	5.06e-2	3.63e-2	2.56e-3	5.06e-2	**2.27e-2**	**1.07e-3**	**3.27e-2**
Emergency	4.18e-2	6.65e-1	8.10e-1	6.35e-2	6.75e-3	8.22e-2	4.77e-02	3.88e-03	6.23e-02	**3.88e-02**	**2.26e-03**	**4.75e-02**
Surveillance	2.41e-02	2.68e-01	5.05e-01	2.53e-04	1.16e-07	3.40e-04	2.49e-04	9.66e-08	3.11e-04	**1.56e-04**	**3.26e-08**	**1.81e-04**

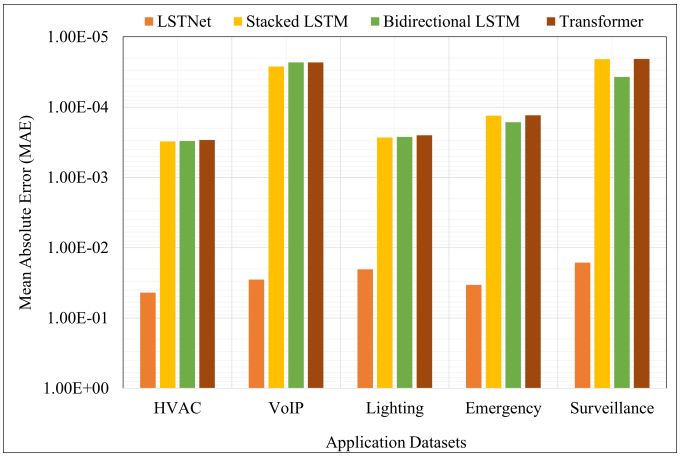


Fig. 9. MAE of multivariate PDR prediction across all datasets

in the IoT networks.

Lastly, Table XII presents the results for the multivariate latency QoS for all of the applications. It can be seen that the proposed model outer-performs all the baselines for all applications and in terms of all metrics. The second best performing baseline method is bidirectional as it gives reasonable results for 4 out of 5 applications. Once more, the LSTNet method shows poor performance compared to the rest of the methods and this is because it is unable to capture all the dependencies among input sequences and other QoS features in the datasets.

To conclude, regarding MAE, there is 1% to 92% improvements provided by our transformer model. Furthermore, for latency, there is 2% to 37% improvement in term of MAE, 6% to 66.25% in term of MSE and 3% to 42% in term of RMSE provided by our proposed temporal transformer model compared with the second best performing baseline method. Finally, our proposed transformer model achieves the best performance on 20 out of 20 settings for the MAE case, and on 16 out of 20 settings for the MSE and RMSE respectively, for the multivariate forecasting task.

Regarding the impact of the problem setting as either univariate or multivariate on the prediction of the QoS metrics, we observed that our proposed model performed better in the univariate setting than the multivariate. This is because there are only 4 univariate cases and 8 multivariate cases in which our proposed transformer model performed worse than the other models. It is to be noted that multivariate models are good to model interesting inter-dependencies however, in the expense of an additional complexity. One of the reason for this behavior is that some IoT application's QoS dataset may include outliers which can more adversely affect the multivariate than the univariate forecasts. Moreover, it is easier to spot and control outliers in the univariate context. Also, the QoS datasets showed a nonlinear behavior w.r.t. time thus, the univariate setting can handle the non-linearities more properly than the multivariate model. Therefore, it is better to use the univariate setting for predicting each of the individual QoS in real IoT application scenario.

VII. CONCLUSION

In this work, we investigated the QoS prediction problem by formulating it as a univariate and multivariate time series forecasting problem. A new framework was introduced that promotes an efficient QoS prediction for a number of coexisting and heterogeneous IoT applications that stress the IoT access network creating several levels of QoS uncertainty. We firstly generated five different real time datasets for HVAC, lighting, VoIP, surveillance and emergency response applications. Following, we presented a novel transformer-based architecture, which learns temporal representations and their complex dependencies in a long-term fashion, for the prediction of four important QOS metrics, namely, throughput, PDR, PLR and latency. The transformer architecture leverages the attention mechanism, which is effective at modelling time series. Finally, we performed an extensive experimental evaluation in which we proved that our proposed temporal transformer achieves superior performance for almost all of the five IoT applications and for both univariate and multivariate

settings, as compared with several competitive time series baseline methods.

As future work, we aim to explore alternative attention techniques, such as sparse attention or compressed attention and investigate their impact on the accuracy achieved. Furthermore, we would like to predict several key QoS metrics, when mobile IoT devices are considered by the applications, thus creating another level of uncertainty in the overall communication.

ACKNOWLEDGMENT

This work was supported in part by the CHIST-ERA-2018-DRUID-NET project "Edge Computing Resource Allocation for Dynamic Networks".

REFERENCES

[1] "Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper", Cisco, 2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed: Feb. 2022].

[2] A. Pratap, A. Kumar and M. Kumar, "Analyzing the Need of Edge Computing for Internet of Things (IoT)", in Proceedings of Second International Conference on Computing, Communications, and Cyber-Security, Singapore, 2021, pp. 203-212.

[3] F. Saeik, et al., "Task Offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions," Computer Networks, vol. 195. pp. 108177, 2021.

[4] D. Dechouniotis, et al., "Edge computing resource allocation for dynamic networks: The DRUID-NET vision and perspective," Sensors, vol. 20, no. 8, pp: 2191, 2020.

[5] M. Marjani et al., "Big IoT data analytics: Architecture, opportunities, and open research challenges," IEEE Access, vol. 5, pp. 5247–5261, 2017.

[6] D. Bisht and M. Ram, Recent advances in time series forecasting. CRC Press, 2021, p. 240.

[7] A. Hameed, J. Violos and A. Leivadeas, "A Deep Learning Approach for IoT Traffic Multi-Classification in a Smart-City Scenario," in IEEE Access, vol. 10, pp. 21193-21210, 2022, doi: 10.1109/ACCESS.2022.3153331.

[8] A. R. Abdellah, O. Abdulkareem Mahmood, and A. Koucheryavy, "Delay prediction in iot using machine learning approach," in Int. Con. on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2020, pp. 275–279.

[9] M. Ateeq, F. Ishmanov, M. K. Afzal, and M. Naeem, "Predicting delay in iot using deep learning: A multiparametric approach," IEEE Access, vol. 7, pp. 62 022–62 031, 2019.

[10] O. Said and A. Tolba, "Accurate performance prediction of IoT communication systems for smart cities: An efficient deep learning based solution", Sustainable Cities and Society, vol. 69, p. 102830, 2021. Available: 10.1016/j.scs.2021.102830.

[11] Y. Hou et al., "A Study of Throughput Prediction using Convolutional Neural Network over Factory Environment," 2021 23rd International Conference on Advanced Communication Technology (ICACT), 2021, pp. 429-434, doi: 10.23919/ICACT51234.2021.9370905.

[12] M. Ateeq, F. Ishmanov, M. Afzal and M. Naeem, "Multi-Parametric Analysis of Reliability and Energy Consumption in IoT: A Deep Learning Approach", Sensors, vol. 19, no. 2, p. 309, 2019. Available: 10.3390/s19020309.

[13] A. Hameed, J. Violos, N. Santi, A. Leivadeas and N. Mitton, "A Machine Learning Regression Approach for Throughput Estimation in an IoT Environment," 2021 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics), 2021, pp. 29-36, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics53846.2021.00020.

[14] X. Fan, C. Xiang, L. Gong, X. He, C. Chen and X. Huang, "UrbanEdge: deep learning empowered edge computing for urban IoT time series prediction", in ACM TURC '19: Proceedings of the ACM Turing Celebration Conference, China, 2019, pp. 1-6.

[15] D. Wu, H. Xu, Z. Jiang, W. Yu, X. Wei and J. Lu, "EdgeLSTM: Towards Deep and Sequential Edge Computing for IoT Applications", IEEE/ACM Transactions on Networking, vol. 29, no. 4, pp. 1895-1908, 2021. Available: 10.1109/tnet.2021.3075468.

[16] A. R. Abdellah, V. Artem, A. Muthanna, D. Gallyamov, and A. Koucheryavy, "Deep learning for iot traffic prediction based on edge computing," in Distributed Computer and Communication Networks: Control, Computation, Communications, 2020, pp. 18–29.

[17] M. Lopez-Martin, B. Carro, and A. Sanchez-Esguevillas, "Neural network architecture based on gradient boosting for iot traffic prediction," Future Generation Computer Systems, vol. 100, pp. 656–673, 2019.

[18] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in Neural Computation, vol. 9, no. 8, pp. 1735-1780, 15 Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Proc. NIPS, 2014, pp. 3104–3112.

[20] H. Cheng, Z. Xie, L. Wu, Z. Yu and R. Li, "Data prediction model in wireless sensor networks based on bidirectional LSTM", EURASIP Journal on Wireless Communications and Networking, vol. 2019, no. 1, 2019. Available: 10.1186/s13638-019-1511-4.

[21] S. Bai, J. Zico Kolter, V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." arXiv preprint arXiv:1803.01271 (2018).

[22] D. Salinas, V. Flunkert, J. Gasthaus and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks", International Journal of Forecasting, vol. 36, no. 3, pp. 1181-1191, 2020. Available: 10.1016/j.ijforecast.2019.07.001.

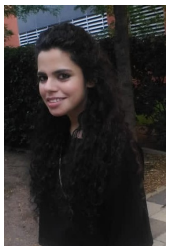
[23] G. Lai, W. Chang, Y. Yang and H. Liu, "Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks", in SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, New York, NY, United States, 2018, pp. 95-104.

[24] C. Adjih et al., "Fit IoT-lab: A large scale open experimental IoT testbed," in IEEE World Forum on IoT (WF-IoT), 2015, pp. 459–464.

[25] N. Santi, et al., "Automated and Reproducible Application Traces Generation for IoT Applications," in Proc. of the 17th Symp. on QoS and Security for Wireless and Mobile Networks (Q2SWinet), 2021, pp. 17-24.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017, pp. 6000–6010.

[27] Zerveas, G., Jayaraman, S., Patel, D., Bhamidipaty, A. and Eickhoff, C., 2021. A Transformer-based Framework for Multivariate Time Series Representation Learning. In: KDD '21: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. ACM, pp.2114–2124.



Aroosa Hameed is currently a Ph.D. student in the Dept. of Software and Information Technology Engineering at Ecole de technologie superieure (ETS) in Montreal. She received her master's degree in computer science from Quaid-i-Azam University, Islamabad, Pakistan, in 2018. Her main research interests include Internet of Things (IoT), traffic analytics, IoT services, IoT security, and machine learning.

John Violos is research associate in the Dept. of Software Engineering and Information Technology at ETS. His previous positions were research associate at National Technical University of Athens, sessional lecturer at Harokopio University of Athens and visiting lecturer at National and Kapodistrian University of Athens. He was a member in the European Commission's Digital Single Market working group on the code of conduct for switching and porting data between cloud service providers. His research interests include Deep Learning, Machine Learning, Cloud and Edge computing.

Aris Leivadeas (S'12-M'15-SM'21) is currently an Associate Professor with the Department of Software and Information Technology Engineering at the École de technologie Supérieure (ETS), Montreal, Canada. From 2015 to 2018 he was a postdoctoral fellow in the Department of Systems and Computer Engineering, at Carleton University, Ottawa Canada. In parallel, Aris worked as an intern at Ericsson and collaborated with Cisco in Ottawa, Canada. He received his diploma in Electrical and Computer Engineering from the University of Patras in 2008, the M.Sc. degree in Engineering from King's College London in 2009, and the Ph.D degree in Electrical and Computer Engineering from the National Technical University of Athens in 2015. His research interests include Network Function Virtualization, Cloud and Edge Computing, IoT, and network optimization and management. He received the best paper award in ACM ICPE'18 and IEEE iThings'21 and the best presentation award in IEEE HPSR'20.

Nina Santi is a PhD student under the supervision of Nathalie Mitton in the Inria FUN team. Their focus is on small computing devices like electronic tags and sensor networks. She has received the MSc degrees in Computer Science from University of Lille, France, in 2020.

Rémy Grünblatt is an associate professor in Computer Networks at Télécom Sud-Paris in the Très Haut Débit (THD) team, in the Telecommunications Networks and Services department. After obtaining his master's degree in Computer Science from the École Normale Supérieure de Lyon (ENS de Lyon) in 2018, he began his thesis in the Grenoble - Rhône-Alpes research center from the National Institute for Research in Computer Science and Automation (INRIA) on the subject "From WiFi Performance Evaluation to Controlled Mobility in Drone Networks", under the supervision of Isabelle Guérin-Lassous and Olivier Simonin. After his thesis, he spent 6 months in the "FUN" Inria team in Lille / Villeneuve d'Ascq, where he worked with Nathalie Mitton on edge computing resource allocation for dynamic networks. His research topics focus on networks, especially wireless and mobile networks, as well as distributed and autonomous systems such as robots or drone networks or IOT networks. He is also very interested in privacy and security, especially when dealing with the aforementioned topics.

Nathalie Mitton received the MSc and PhD. degrees in Computer Science from INSA Lyon in 2003 and 2006 respectively. She received her Habilitation à diriger des recherches (HDR) in 2011 from Université Lille 1. She is currently an Inria full researcher since 2006 and from 2012, she is the scientific head of the Inria FUN team which is focused on small computing devices like electronic tags and sensor networks. Her research interests focus on self-organization from PHY to routing for wireless constrained networks. She has published her research in more than 30 international revues and more than 100 international conferences. She is involved in the setup of the FIT IoT LAB platform (http://fit-equipex.fr/, https://www.iot-lab.info), the H2020 CyberSANE and VESSEDIA projects and in several program and organization committees such as Infocom 2021 & 2020 & 2019, PerCom 2020 & 2019, DCOSS 2021 & 2020 & 2019, Adhocnow (since 2015), ICC (since 2015), Globecom (since 2017), VTC (since 2016), etc. She also supervises several PhD students and engineers.