IoT Traffic Modelling and QoS Prediction Framework using Advanced Deep Learning

by

Aroosa Hameed

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE IN PARTIAL FULFILLMENT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, AUGUST 8, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

FOREWORD

This doctoral dissertation unfolds as a comprehensive compilation of scholarly articles. Over the course of this research study, three journal articles were written and submitted for review. Their presentation within this thesis maintains high consistency with their original form to adhere to the structure and design of the articles as they were published. However, minor adjustments were undertaken, such as modifications to figures and their positioning to conform to the Ecole de Technologie Superieure's dissertation format. The first chapter provides the research objectives, contributions, and the methodologies that we used in this research. Next, a separate chapter reviews the current and highly relevant literature that aligns with the research problems tackled in this doctoral study. Chapters 2, 3, and 4 present the articles that have been published and those that have been submitted for review. Following these chapters, the conclusion about these papers and their future research directions is provided.

ACKNOWLEDGEMENTS

First of all, I would like to express my profound gratitude to my supervisor, Prof. Aris Leivadeas, whose invaluable guidance and unwavering support have been pillars of my academic journey. His profound knowledge, dedication, and consistent encouragement have played an instrumental role in shaping my research. I am truly fortunate to have this opportunity to work under his supervision. His belief in my abilities has been a constant source of motivation, which has inspired me to push boundaries and strive for excellence. Further, I extend my sincere thanks to my thesis committee members: Prof. Christian Desrosiers, Prof. Kim Khoa Nguyen and Prof. Petros Spachos for evaluating this thesis and participating in the defense. I am also grateful to all of my colleagues at the Laboratoire en architecture de systèmes informatiques (LASI) for their insightful discussions. Also, I would like to thank all my cherished friends, both in and outside of Canada for their continual motivation during my PhD. Above all, I owe an immense debt of gratitude to my parents whose love is immeasurable, and this thesis would not have existed without their support. Finally, I would like to thank the Fonds de recherche du Québec (FRQNT) and DRUID-NET research project for the financial support throughout my PhD journey.

Modélisation du trafic IdO et cadre de prédiction de la qualité de service à l'aide de l'apprentissage profond avancé

Aroosa Hameed

RÉSUMÉ

L'Internet des Objets (IdO) désigne des milliards d'appareils physiques connectés à l'internet dans le monde, qui collectent et partagent tous des données. L'IdO englobe une grande variété d'appareils tels que les appareils domestiques intelligents (thermostats, réfrigérateurs et systèmes de sécurité), les machines industrielles, les moniteurs de santé portables et même les technologies des villes intelligentes. L'expansion des dispositifs IdO apporte de nombreux avantages, mais pose également des défis liés aux caractéristiques de ressources limitées de beaucoup de ces dispositifs. Ces limitations, notamment la puissance de calcul, la mémoire et l'efficacité énergétique, peuvent avoir un impact sur les performances et la réactivité, en particulier dans le contexte d'applications cruciales et sensibles au facteur temps. L'avènement d'informatique en périphérie du réseau (PdR) offre une solution prometteuse à ces défis. En plaçant les ressources informatiques à proximité des dispositifs IdO, le PdR peut minimiser la latence et permettre un traitement efficace des données. Pourtant, la nature imprévisible de la génération de données IdO, compte tenu de la diversité des types d'appareils et de leurs cycles d'activité, introduit son propre ensemble de complexités. En outre, l'utilisation de communications sans fil et de bandes de fréquences sans licence aggrave les problèmes liés à l'efficacité de la transmission des données, aux interférences et à la capacité limitée, ce qui peut entraîner une augmentation du temps de latence au sein des réseaux IdO. À ce cadre de communication complexe s'ajoute la connectivité réseau variable des appareils mobiles IdO tels que les drones ou les robots, qui contribue aux complications existantes. Il devient donc impératif de prévoir les caractéristiques variables dans le temps des appareils IdO pour mieux gérer les ressources et assurer la qualité de service (QdS). Le regroupement d'appareils IdO similaires peut améliorer les estimations de la charge de travail et faciliter l'allocation des ressources. En outre, les modèles de prédiction de la QoS qui intègrent des données de séries temporelles sur la génération de paquets et les caractéristiques du réseau peuvent contribuer à une meilleure compréhension des exigences et des limites de la plateforme IdO.

Dans cette thèse, nous considérons d'abord la catégorisation du trafic provenant des dispositifs IdO en classes distinctes. Cette approche est destinée à faciliter la gestion et la compréhension de la gamme variée de données provenant de divers dispositifs IdO dans un environnement de ville intelligente. Nous modélisons le problème d'identification du trafic des dispositifs IdO comme un problème d'apprentissage classification multiple et proposons un modèle d'apprentissage en deux étapes comme solution. Afin de capturer les nuances du comportement des appareils IdO, nous suggérons un ensemble de caractéristiques étendu, qui comprend des caractéristiques au niveau du flux, du paquet et de l'appareil. Ces caractéristiques décrivent les caractéristiques et le comportement des péripheriques IdO dans le contexte d'une ville intelligente. De plus, nous proposons un algorithme de prétraitement de pondération personnalisé pour déterminer la contribution des caractéristiques des données de trafic au processus de classification. Ensuite, un mécanisme de sélection des caractéristiques basé sur l'ANOVA et la méthode de corrélation basée sur le coefficient de Pearson sont utilisés pour une compréhension plus approfondie des caractéristiques du trafic (caractéristiques). Enfin, nous proposons un algorithme d'apprentissage innovant en deux étapes, qui utilise la régression logistique à l'étape 0 pour catégoriser initialement les données, et un réseau de perceptron multicouche (MLP) à l'étape 1 pour affiner la classification. Ce processus en deux étapes permet une classification plus précise des dispositifs IdO, comme cela a été démontré avec deux ensembles de données réels différents sur le trafic IdO.

Le deuxième aspect du travail présenté dans cette thèse concerne la prédiction efficace de diverses mesures de QdS, telles que le débit, le taux de livraison des paquets, le taux de perte de paquets et la latence pour un ensemble diversifié d'applications IdO. Nous proposons un modèle de transformateur temporel au sein d'un système unifié, conçu pour prédire les mesures de qualité de service typiques. Cette prédiction est modélisée comme un problème de prévision de séries temporelles, utilisant à la fois des configurations univariées et multivariées. Nous générons un ensemble de données composé d'informations sur le trafic en temps réel provenant de cinq applications IdO distinctes : Chauffage, ventilation et climatisation (HVAC), éclairage intelligent, voix sur protocole Internet (VoIP) pour un assistant virtuel, surveillance et intervention d'urgence. Le modèle de transformateur temporel proposé tire parti d'un module d'attention pour prédire les séquences de QdS à court et à long terme, ce qui permet d'extraire plus efficacement les dépendances temporelles. Ce deuxième aspect de la thèse se concentre uniquement sur la prédiction des métriques de QdS par application IdO avec des nœuds de capteurs statiques.

La troisième partie de cette thèse étend la direction susmentionnée et concerne la prédiction des métriques de qualité de service dans un cadre où les passerelles IdO sont des robots mobiles. À cette fin, nous mettons en œuvre trois applications IdO dans un environnement réel sous huit configurations de réseau distinctes qui tiennent compte de la mobilité des robots, de la puissance de transmission des dispositifs IdO et des fréquences de canal spécifiques à l'application utilisée. Pour prédire le comportement de la qualité de service de diverses mesures dans un environnement aussi dynamique, nous proposons un transformateur temporel espacé fédéré (FeD-TST). Ce cadre permet aux clients locaux de former des modèles individuels avec leur ensemble de données de qualité de service unique pour chaque configuration de réseau, puis de mettre à jour un modèle global associé par le biais de l'amalgame des modèles locaux. Pour chaque client, un modèle transformateur temporel espacé est formé, qui comprend plusieurs composants, notamment les modules d'encodage et de décodage, le module d'attention clairsemée multi-têtes et le module d'attention clairsemée masquée. Enfin, le cadre génère des prévisions univariées ou multivariées, dont l'efficacité est ensuite évaluée sur la base d'un ensemble de données de test. Par conséquent, la principale contribution de cette thèse peut être résumée en fournissant aux dispositifs IdO des mécanismes de classification du trafic et de prédiction de la qualité de service pour un ensemble d'applications IdO. Les méthodologies que nous proposons tiennent compte des différents modèles de génération de données IdO, de la mobilité des appareils et des réseaux d'accès au sein d'un écosystème IdO, tandis que des évaluations expérimentales approfondies indiquent que nos approches proposées sont plus performantes que les techniques de pointe actuelles en termes de performance et d’efficacité. En outre, nos approches proposées peuvent fournir des informations tangibles sur les besoins en ressources et les limitations rencontrées au niveau de l’accès au réseau IdO et au sein de l’infrastructure périphérique.

Mots-clés: Classification des dispositifs IdO, Prédiction de la qualité de service, Informatique en périphérie du réseau, Apprentissage profond, Prévision des séries temporelles, Apprentissage fédéré

IoT Traffic Modelling and QoS Prediction Framework using Advanced Deep Learning

Aroosa Hameed

ABSTRACT

The Internet of Things (IoT) refers to the billions of physical devices around the world that are connected to the internet, all collecting and sharing data. IoT includes a wide variety of devices such as smart home devices like thermostats, refrigerators, and security systems, industrial machines, wearable health monitors, and even smart city technologies. The expansion of IoT devices brings numerous benefits, yet it also poses challenges arising from the resource-limited characteristics of many such devices. These limitations, including constrained computing power, memory, and energy efficiency, can impact performance and responsiveness, particularly in the context of crucial, time-sensitive applications. The advent of Edge Computing (EC) offers a promising solution to these challenges. By situating computational resources in proximity to IoT devices, EC can minimize latency and enable efficient data processing. Nonetheless, the unpredictable nature of IoT data generation, given the diversity of device types and their activity cycles, introduces its own set of complexities. Moreover, the employment of wireless communication and unlicensed frequency bands compounds issues related to data transmission efficiency, interference, and limited capacity, potentially leading to increased latency within IoT networks. Adding to this intricate communication framework is the variable network connectivity of mobile IoT devices such as drones or robots, which contributes to the existing complications. Therefore, it becomes imperative to predict the time-varying characteristics of IoT devices to better manage resources and ensure Quality of Service (QoS). Grouping similar IoT devices can improve workload estimates and facilitate resource allocation. Moreover, QoS prediction models that incorporate time series data of packet generation and network characteristics can contribute to a better understanding of the IoT platform's requirements and limitations.

In this thesis, we first consider the categorization of traffic from IoT devices into distinct classes. This approach is intended to facilitate the managing and understanding of the diverse range of data coming from various IoT devices within a smart city environment. We model the IoT devices traffic identification problem as a multi-classification learning problem, and propose a two stage learning model as a solution. In order to capture the nuances of IoT device behavior, we suggest an extended feature set, which includes flow, packet, and device-level features. These features describe the characteristics and behavior of IoT devices in a smart city context. We further propose a custom weighting pre-processing algorithm to determine the contribution of traffic data features to the classification process. Subsequently, an ANOVA-based feature selection mechanism and the Pearson's coefficient-based correlation method are employed for a more profound understanding of the traffic characteristics (features). Lastly, we propose an innovative, two-stage learning algorithm, which utilizes logistic regression at stage 0 to initially categorize the data, and a Multi-Layer Perceptron (MLP) network at stage 1 to refine the classification. This two-stage process allows for more accurate classification of IoT devices, as demonstrated with two different real IoT traffic datasets.

The second aspect of the work presented in this thesis involves the efficient prediction of various QoS metrics, such as throughput, Packet Delivery ratio (PDR), Packet Loss Ratio (PLR) and latency for a diverse set of IoT applications. We propose a temporal transformer model within a unified system, designed to predict typical QoS metrics. This prediction is model as a time series forecasting problem, utilizing both univariate and multivariate setups. We generate a dataset comprised of real-time traffic information from five distinct IoT applications: Heating, Ventilation, and Air Conditioning (HVAC), smart lighting, Voice over Internet Protocol (VoIP) for a virtual assistant, surveillance, and emergency response. The proposed temporal transformer model leverages an attention module to predict both short-term and long-term QoS sequences, thereby more effectively extracting time dependencies. This second aspect of the thesis focuses solely on predicting QoS metrics per IoT application with static sensor nodes.

The third part of this thesis extends the aforementioned direction and concerns the prediction of QoS metrics in a setting where the IoT gateways are mobile robots. To this end, we implement three IoT applications in a real-world environment under eight distinctive network configurations that consider robot mobility, IoT device transmission power, and the application-specific channel frequencies used. To predict the QoS behavior of various metrics in such a dynamic environment, we propose a Federated Temporal Sparse Transformer (FeD-TST) framework. This framework enables local clients to train individual models with their unique QoS dataset for each network configuration, subsequently updating an associated global model through the amalgamation of local models. For each client, a Temporal Sparse Transformer model is trained, which comprises of several components, including the encoder and decoder modules, the multi-head sparse attention module, and the masked sparse attention module. Finally, the framework generates univariate or multivariate forecasts, and the effectiveness of which is then evaluated based on a test dataset.

Hence, the major contribution of this particular thesis can be summarized as providing IoT devices with traffic classification and QoS prediction mechanisms for an array of IoT applications. Our proposed methodologies take into account the varied patterns of IoT data generation, device mobility, and access networks within an IoT ecosystem, while extensive experimental evaluations indicate that our proposed approaches outperform current state-of-the-art techniques in terms of performance and efficiency. Additionally, our proposed approaches can provide tangible insights into the resource requirements and limitations encountered at the access level of the IoT network and within the edge infrastructure.

Keywords: IoT device classification, QoS prediction, edge computing, deep learning, time series forecasting, federated learning

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
AQI	Air Quality Index
ANOVA	Analysis of Variance
ABC	Artificial Bee Colony
ANN	Artificial Neural Networks
ARIMA	Auto Regressive Integrated Moving Average
BiLSTM	Bidirectional Long Short-Term Memory
BoW	Bag-of-Word
CBR	Case Based Reasoning
CF	Collaborative Filtering
CMMPP	Coupled Markov Modulated Poisson Processes
CNN	Convolutional Neural Network
DT	Decision Tree
DL	Deep Learning
DNN	Deep Neural Network
DODAGs	Destination Oriented Directed Acyclic Graphs
DFP	Device FingerPrinting
DEQP2	Distributed Edge Quality of Service Prediction

DEDP	Distributed Edge Differential Privacy
DNS	Domain Name System
DF	Do not Fragment Flag
DHCP	Dynamic Host Configuration Protocol
ESN	Echo State Network
EC	Edge Computing
ECNs	Edge Computing Networks
Edge-PMAM	Edge QoS forecasting with Public Model and Attention Mechanism
FNR	False Negative Rate
FPR	False Positive Rate
FL	Federated Learning
FFN	Feed Forward Network
FeD-TST	Federated Temporal Sparse Transformer
FC	Fully Connected
GRU	Gated Recurrent Unit
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
GTID	Georgia Tech ID
GB	Gradient Boosting
HVAC	Heating, Ventilation, and Air Conditioning
HE	Homomorphic Encryption

IAT	Inter Arrival Time
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
IID	Independent and Identically Distributed
ISP	Internet Service Provider
KNN	k-Nearest Neighbors
KT	Keras Tuner
KPIs	Key Performance Indicators
LPWAN	Low Power Wide Area Networks
LSTM	Long Short-Term Memory
LR	Logistic Regression
LSH	Locality-Sensitive Hashing
LSTNet	Long-and Short-term Time-series Network
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSP	Multi-Step Prediction
MLP-ANN	Multi-Layer Perceptron Artificial Neural Network
MSS	Maximum Segment Size
M2M	Machine-to-Machine

MEC	Multi-Access Edge Computing
MAPE	Mean Absolute Percentage Error
MSE	Mean Square Error
MANET	Mobile Ad-hoc Network
MHA	Multi-Head Attention
NARX	Nonlinear Auto Regressive Exogenous
NB	Naive Bayes
NOP	No Option
OP	Output Layer
PDR	Packet Delivery Ratio
PLR	Packet Loss Ratio
PCAP	Packet CAPture
PE	Positional Encoding
QoS	Quality of Service
RF	Random Forest
RFID	Radio Frequency Identification
RNN	Reccurrent Neural Network
RNCF	Recurrent Neural Network-based Collaborative Filtering
ReLU	Rectified Linear Units
RFE	Recursive Feature Elimination

RELU	Rectified Linear Unit
RL	Reinforcement Learning
RPL	Routing Protocol for Low power and Lossy Network
RMSE	Root Mean Square Error
SMPC	Secure Multi-Party Computation
Seq2Seq	Sequence-to-Sequence
SES	Simple Exponential Smoothing
SSP	single-Step Prediction
SDN	Software-Defined Networking
SQL	Structured Query Language
SVM	Support Vector Machines
TCN	Temporal Convolutional Network
TST	Temporal Sparse Transformer
TSF	Time Series Forecasting
TTL	Time-to-Live
Trainlm	Levenberg-Marquardt
Traincgf	Conjugate Gradient with Fletcher-Reeves updates
Trainrp	Resilient Backpropagation
TAN	Topology-Aware Neural
TCP	Transmission Control Protocol

TLS	Transport Layer Security
UDP	User Datagram Protocol
VAR	Vector AutoRegression
VoIP	Voice over Internet Protocol
WS	Window Size
WSO	Window Size Option
WSN	Wireless Sensor Networks
xgBOOST	eXtreme Gradient Boosting
Zstd	Zstandard
5G	Fifth-Generation

INTRODUCTION

0.1 Context and Motivation

The Internet of Things (IoT) refers to a network of physical devices, vehicles, appliances, and other objects that are embedded with sensors, software, and connectivity capabilities, allowing them to collect and exchange data over the Internet. These devices can interact with their environment, communicate with each other, and often operate autonomously. This paradigm provides a great benefit to the general public by creating a system of interrelated computing, sensing, and communication, that facilitates and improves every aspect of our daily life. It comprises of everyday objects ranging from simple sensors and actuators to lights, cameras, door locks, thermostats, power switches and household appliances, with shipments projected to reach nearly 20 billion by 2020 (Nordrum, 2016). IoT is foreseen to reach 500 billion devices that are connected to the Internet by 2030 (Cisco Systems, 2017), while the global mobile traffic is expected to grow 325 EB per month in 2028 (Ericsson, 2022).

The proliferation of IoT plays a crucial role in collecting and transmitting data, however at the same time it creates several challenges. The IoT devices consisting of local nodes such as sensors and actuators are typically designed with resource limitations due to factors such as cost, size, power consumption, and network connectivity. These constraints can limit their computational power, memory capacity, energy efficiency, and communication capabilities. As a result, when it comes to demanding or time-sensitive applications, such as those in critical infrastructures (i.e., healthcare, industrial automation, transportation, etc.) the local nodes may struggle to provide the required level of performance and responsiveness. For instance, mission-critical applications often necessitate real-time data processing and rapid response times. However, the limited computational resources and processing capabilities of the IoT devices may hinder their ability to handle complex computations or analyze data in real-time. This can result in delays or latency issues, which are unacceptable for applications that rely on immediate decision-making or control actions. Thus, the local nodes cannot meet and cannot guarantee the required high performance and/or fulfilment of time constraints, for mission-critical IoT-enabled applications.

Edge Computing (EC) arises as a promising solution because it is a distributed computing paradigm that brings computational and communication resources closer to the edge of the network. By deploying resources near to IoT devices and endpoints, EC offers faster and more responsive services, reducing latency and enabling efficient data processing. It complements the resource limitations of IoT devices, making it ideal for the mission-critical applications. Additionally, EC facilitates computational offloading, allowing the transfer of computational task to a more powerful edge nodes, resulting in faster data transmissions and reducing latency. Yet, this approach brings its own challenges such as resource estimation and in the context of IoT, this becomes a difficult task because IoT devices generate data randomly (i.e., various rates and times) due to their diverse types and their dynamic activity cycles. For example, a weather sensor is designed to monitor environmental conditions such as temperature, humidity, and rainfall. It typically transmits data only when there is a significant change in the weather conditions i.e., if the temperature suddenly rises or falls beyond a certain threshold, the sensor will send data to indicate this change. Similarly, a smart fridge is equipped with sensors and connectivity features to provide various functionalities such as temperature monitoring, inventory management, and food quality tracking. It periodically transfers smaller amounts of data at regular intervals. For instance, it might send notifications about items that need to be replenished. In contrast, a security camera is responsible for capturing and monitoring video footage for surveillance purposes. It continuously captures and streams high volumes of video data in real-time. The camera sends a constant stream of data to ensure continuous monitoring and recording of the surroundings. As a result of this diversity, the data generated by IoT devices is highly variable in both size and frequency. Also the IoT devices affiliated with different applications adhere to varying data generation modes as well, such as poll-based, periodic, or event-driven. Depending on the mode type and amount of data being generated at any given moment, the processing resources needed could be fluctuating significantly.

Furthermore, IoT access networks typically rely on wireless communication and the use of unlicensed bands further increases the potential for interference from other devices operating in the same frequency range. Due to the wireless and lossy nature of IoT access networks, there is also a possibility of reduced data transmission efficiency. Unstable connections or interference can lead to data packets being lost or delayed, impacting the overall efficiency of the transmission process. Also in the scenarios where a significant number of IoT devices are connected to the network, the limited capacity of the access network becomes a concern. When a large number of devices are active simultaneously, the network may struggle to accommodate and efficiently handle the increased load, resulting in potential delays and reduced performance. This can lead to increased communication delays in IoT access networks. The constrained nature of these networks, combined with potential interference and limited capacity, can further exacerbate delays in transmitting data between devices and the EC infrastructure.

Moreover, some IoT devices, such as robots, drones, or other mobile systems, are designed to move and operate in various locations. This introduces an additional layer of complexity in the data generation and transmission process compared to stationary devices. For example, in an emergency response scenario, drones or mobile robots are often deployed to capture aerial footage, survey affected areas, and assess the extent of damage or danger. As drones move across the scene, their network connectivity may change depending on their distance from access points or obstructions in the environment, such as buildings or natural obstacles. These changes in network connectivity can impact the quality and reliability of the communication and result in delays or disruptions in data transmission. This can impact the timeliness and accuracy of the information relayed to emergency responders. For instance, in a search and rescue operation, a drone providing live video feed from a remote location may experience intermittent connectivity or reduced video quality due to its mobility, making it challenging for responders to make informed decisions or assess the situation accurately. With device mobility, the QoS can change rapidly, leading to fluctuations in data transfer rates, latency, or even occasional disconnections. This variability in QoS can affect the overall effectiveness of different IoT applications or services. Thus, the constant changes in communication quality and network accessibility due to device mobility further complicate the communication of the IoT devices with the rest of the infrastructure (i.e., EC).

Therefore, the ability to predict the time-varying characteristics of IoT devices, such as their activity and signaling patterns, becomes increasingly important. These patterns could give us an insight into when and how much data will be generated and transferred, aiding in better resource management techniques. Moreover, grouping or classifying similar IoT devices together allows for more accurate workload estimations and ensures a certain level of Quality of Service (QoS). For instance, devices of the same type likely have similar data generation and transmission patterns. By understanding these patterns, we can predict the network traffic more efficiently. In addition to this, categorizing IoT devices also aids in anticipating the resources needs both at the access level of the IoT network e.g., spectrum requirements for the data transmission and within the edge infrastructure e.g., computational and communication resources. Furthermore, the QoS requirements for IoT applications can vary extensively depending on their specific function and nature. Therefore, developing an efficient QoS prediction model is crucial. This model should incorporate the time series data of packet generation by IoT devices (i.e., when packets are transmitted/received) as well as the network characteristics of the access network (i.e., frequency channel, etc.). Such a model can provide insights into the expected QoS based on past and current trends, thereby aiding in understanding the requirements and the limitations of an IoT platform.

0.2 Problem Statement

It is extremely difficult to design an effective traffic classification mechanisms for a diverse range of IoT devices and accurately predict the QoS requirements for various IoT applications within a dynamic IoT environment due to the following challenges as:

1. The diverse data generation patterns exhibited by IoT devices can be highly unpredictable and random in nature. Different types of IoT devices, such as sensors, actuators, cameras, or wearable, have varying data generation patterns and characteristics. Additionally, these devices often operate on dynamic cycles, meaning their activity levels can vary over time. For example, a sensor may generate data periodically, while a camera may capture images only when motion is detected. This dynamic and random nature of IoT data generation makes it difficult to categorize the IoT devices and their traffic characteristics.
2. IoT devices rely on wireless access networks to establish connectivity and exchange data with the edge infrastructure or the cloud. However, wireless networks which may consist of various technologies such as WiFi, cellular networks, or Low Power Wide Area Networks (LPWAN) inherent ambiguity and are subject to various dynamic factors that can affect the quality of the connection and the available resources. These factors include signal strength variations, interference from other devices, and network congestion caused by the simultaneous operation of multiple devices. Failing to account for the dynamic nature of the wireless access network can lead to sub-optimal resource allocation decisions. For example, if the resource allocation algorithm assumes a static and ideal network condition, it may allocate resources based on inaccurate or outdated information. This can result in inefficient utilization of resources, degraded network performance, and reduced QoS for the IoT applications.
3. The IoT gateways or devices, including robots, drones, or other mobile devices, have the ability to move or change their positions within the network environment. This mobility introduces additional complexities and considerations for resource estimation. When

IoT gateways or devices are mobile, their movement can affect the performance of the communication. The changing positions and locations of these mobile devices can result in variations in signal strength, network connectivity, and communication quality.

The above challenges can lead to IoT traffic generation with time-varying characteristics and Quality of Service (QoS) behavior for different IoT application. Hence, it is important to propose solutions to solve these challenges at three different levels, namely, i) device, ii) application and iii) mobility, such as:

1. At the device level, it is important to accurately understand the volume, frequency, and characteristics of the IoT data generated by different IoT devices. Therefore, it is required to classify IoT devices into different categories (i.e. hubs, cameras, air quality sensors etc.) according to several network features (i.e., inter-arrival time, network and link layer information, etc.). Without a clear understanding of the data patterns and activity levels, it becomes difficult to classify the IoT devices.
2. At the application level, it is important to take into consideration that the heterogeneous devices for different applications send data of different contexts, with different reporting frequencies usually over a random access channel generating thus, high interference levels. Furthermore, it is also necessary to take into account wireless access network related factors such as signal strength fluctuations, interference levels, and network congestion levels. Therefore, it becomes challenging to accurately predict typical QoS metrics for each IoT application. In reality, it is expected that the QoS behavior of an IoT application, when transmitting data, will be time-dependent showing a dynamic change in the values of key Quality of Service (QoS) metrics. Hence, it is necessary to propose an efficient model that will analyze and predict the QoS metrics using the time series data generated by the IoT devices.
3. At the mobility level, it is expected that some IoT devices or gateways will be mobile (i.e., robots, drones, etc.). This mobility adds an additional layer of complexity towards the

prediction of the traffic characteristics of the IoT applications. Specifically, the mobility of IoT devices may introduce signal loss, interference, or weak connectivity, which can in turn degrade the reliability of communication. Unreliable communication can lead to packet loss, increased latency, or disrupted data transmission, impacting the QoS of applications that depend on real-time or continuous data exchange. Therefore, it is important to propose an efficient QoS forecasting model, which will use the time series data of how packets are generated by the IoT devices (i.e., when packets are transmitted/received) along with the network characteristics of the access network (i.e., frequency channel, etc.) and the mobility factor of the IoT devices.

0.3 Research Objectives

According to the challenges identified above, the main objective of this research is to design a system that can accurately classify IoT traffic into different categories or types and develop a framework for predicting the Quality of Service in IoT networks. To achieve the required target, we decompose the main problem into several sub-problems. Each of the sub-problems is solved separately and the combination of these solutions will allow thus to attain the main objective. In particular in this thesis, we focus on the following three sub-objectives:

1. Classifying the IoT devices according to their network and traffic characteristics.
2. Predicting the QoS metrics of different IoT applications in a heterogeneous and highly fluctuating communication environment with respect to different requirements in terms of number of devices, packet length, context of message, and message frequency transmission.
3. Predicting the QoS metrics of various IoT applications in a dynamic wireless environment by analyzing the impact of mobility, transmission power, and channel allocation in the prediction accuracy.

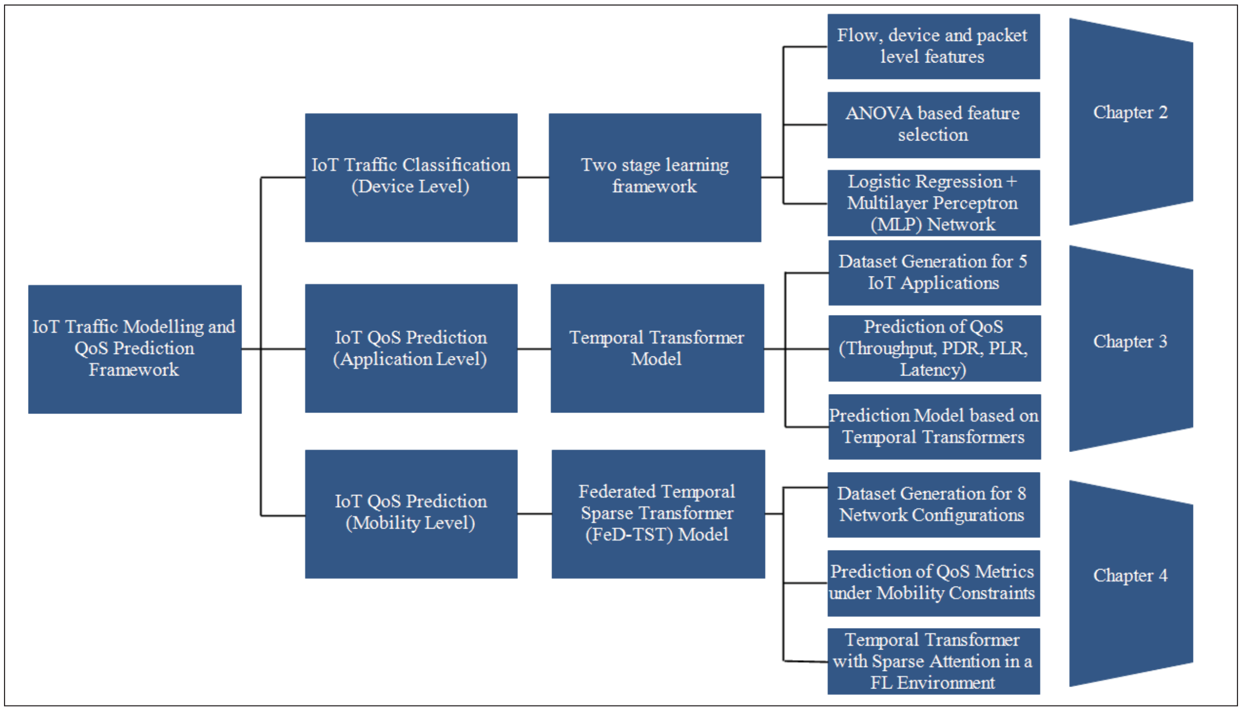

IoT Traffic Modelling and QoS Prediction Framework
IoT Traffic Classification (Device Level)
IoT QoS Prediction (Application Level)
IoT QoS Prediction (Mobility Level)
Two stage learning framework
Temporal Transformer Model
Federated Temporal Sparse Transformer (FeD-TST) Model
Flow, device and packet level features
ANOVA based feature selection
Logistic Regression + Multilayer Perceptron (MLP) Network
Dataset Generation for 5 IoT Applications
Prediction of QoS (Throughput, PDR, PLR, Latency)
Prediction Model based on Temporal Transformers
Dataset Generation for 8 Network Configurations
Prediction of QoS Metrics under Mobility Constraints
Temporal Transformer with Sparse Attention in a FL Environment
Chapter 2
Chapter 3
Chapter 4

Figure 0.1 Paradigm of the thesis contributions

0.4 Contributions

In a nutshell, the core contribution of the thesis is to propose an efficient traffic classification and QoS prediction framework for different IoT applications with their specific needs and preferences. In order to address the significant and challenging issues introduced above, the structure of the thesis contributions is depicted in Figure 0.1 and is elaborated in more details in the remainder of this section:

The first contribution of the thesis is the classification model at the device level, which focuses on categorizing the IoT devices' traffic into different classes. To do so, we propose a two stage learning model in order to categorize the IoT traffic and devices connected within a smart city environment. We model the IoT traffic categorization problem as a multi classification learning problem. Then we provide an extended feature set including, flow, packet and device level features to characterize the IoT devices in the context of a smart environment. After this, a custom weighting based preprocessing algorithm is presented to determine the importance of the IoT data values. Then the useful insights into traffic characteristics is provided using the ANOVA based feature selection mechanism and the Pearson's coefficient based correlation method. Finally, a novel, two-stage learning algorithm is proposed that incorporates logistic regression at stage 0 and Multi Layer Perceptron (MLP) network at the stage 1 to solve the multi classification problem and we demonstrate its ability to accurately categorize the IoT devices for two different datasets.

The second contribution is an efficient prediction of several QoS metrics during the transferring of data of several IoT applications. Specifically, a temporal transformer model and a unified system is proposed in order to predict typical QoS metrics of heterogeneous IoT applications when they communicate with the Edge of the network. Firstly, we model the QoS prediction problem as time series forecasting problem using both univariate and multivariate settings. Secondly, a set of datasets is generated that contains the real-time traffic information of five different IoT applications such as Heating, Ventilation, and Air Conditioning (HVAC), smart lighting, Voice over Internet Protocol (VoIP) for a virtual assistant, surveillance and emergency response. The applications communicate over a 802.15.4 access technology using the RPL routing protocol. Following, we present the data cleaning, down-sampling and pre-processing of the datasets and how the QoS datasets are constructed, which include four QoS metrics, namely throughput, packet delivery ratio, packet loss ratio and latency. Thirdly, we propose the temporal transformer model which leverages an attention module to provide a solution for both short-term and long-term QoS sequence prediction by allowing to better extract any time dependencies. Following, we provide the architectural details of temporal transformer model consisting of input/output module, QoS positional embedding module and encoder module. Finally, the performance evaluation of the transformer model through extensive experimentation using both short-term and long-term dependencies is performed and show that our transformer based model can guarantee a robust performance and accurate QoS prediction. It is worth mentioning that the second contribution only considers QoS metrics prediction per IoT application with static sensor nodes. The more general multi-parameter network scenario is part of the following contribution.

The third contribution of this thesis is to provide a secure data ownership while performing the QoS forecasting with respect to mobile IoT gateways using the Federated Learning (FL) approach (at the mobility level). Firstly, we deploy three IoT applications in a real testbed under eight different network configurations with varying parameters including the mobility of the gateways, the transmission power of the IoT devices and the channel frequency used per each application. Following, the generated datasets of these network configurations are pre-processed using normalization, down sampling and cleaning procedures and are prepared as univariate or multivariate inputs. Secondly, we propose the FeDerated Temporal Sparse Transformer (FeD-TST) framework, which allows local clients to train their local models with their own QoS dataset for each network configuration; subsequently, an associated global model can be updated through the aggregation of the local models. In particular each client is trained using the Temporal Sparse Transformer (TST) model which involves several components, such as the encoder module, the decoder module, the multi-head sparse attention module and a masked sparse attention module. Both the encoder and decoder modules consist of multiple identical sub-layers that are stacked to each other. Finally, a univariate or multivariate forecasts is produced along with the evaluation based on the test dataset.

0.5 Methodology

In this thesis, we adhered to a standardized methodological approach for all the technical contributions made. The methodology is depicted in the Figure 0.2 which consists of several steps. The first step is that each technical contribution began with a comprehensive review of the existing literature. We thoroughly surveyed pertinent scholarly articles, books, and other resources relevant to the topic at hand, helping us to understand the current state of knowledge and identify gaps in the field. Then we formulated the problem pertaining to each objective listed in Section 0.3. This involved formulating IoT traffic categorization as a multi classification problem and QoS prediction as a time series forecasting problem. Following this, we moved to datasets generation step which served as the backbone of our research. For our first contribution i.e., IoT traffic/devices classification, we relied on two open-source datasets consisting of IoT network traces. For the second and third contributions i.e., QoS prediction, we generated two original datasets published in (Santi *et al.*, 2021) and (IoT-LAB, 2022) which are tailor-made to address specific research needs. The datasets are generated in the FIT IoT-LAB which is an open testbed that allows large-scale experiments and provides openly programmable IoT nodes located on several sites. The process of gathering data for these original datasets was rigorous to ensure the reliability and validity of our findings.

After collecting the datasets, the data preprocessing was performed for preparing and refining the raw data into a usable format, including addressing issues such as missing values, outliers, and inconsistencies. Next, we proposed deep learning based solutions to address the defined problems. Following this, the next step involved training the optimal proposed model and the purpose of this step was to adjust the model's complexity to find the best balance between under-fitting and overfitting. If the model did not achieve an optimal solution, it underwent a hyperparameters tuning phase. The model was then trained with the selected hyperparameters until the desired level of performance was achieved. Upon achieving an optimally trained model, we passed the test dataset to the model for evaluation. This phase involved quantifying the model's performance using appropriate metrics and comparing the model's outputs against actual values.

After the evaluation phase, we interpreted the results. This interpretation provided context to the numerical findings, helped us understand the implications of the model's performance, and offered insights into the problem we are investigating. Finally, the findings from our research is disseminated including a detailed discussion of the results, implications, potential future research directions, and a conclusion summarizing the thesis overall contributions.

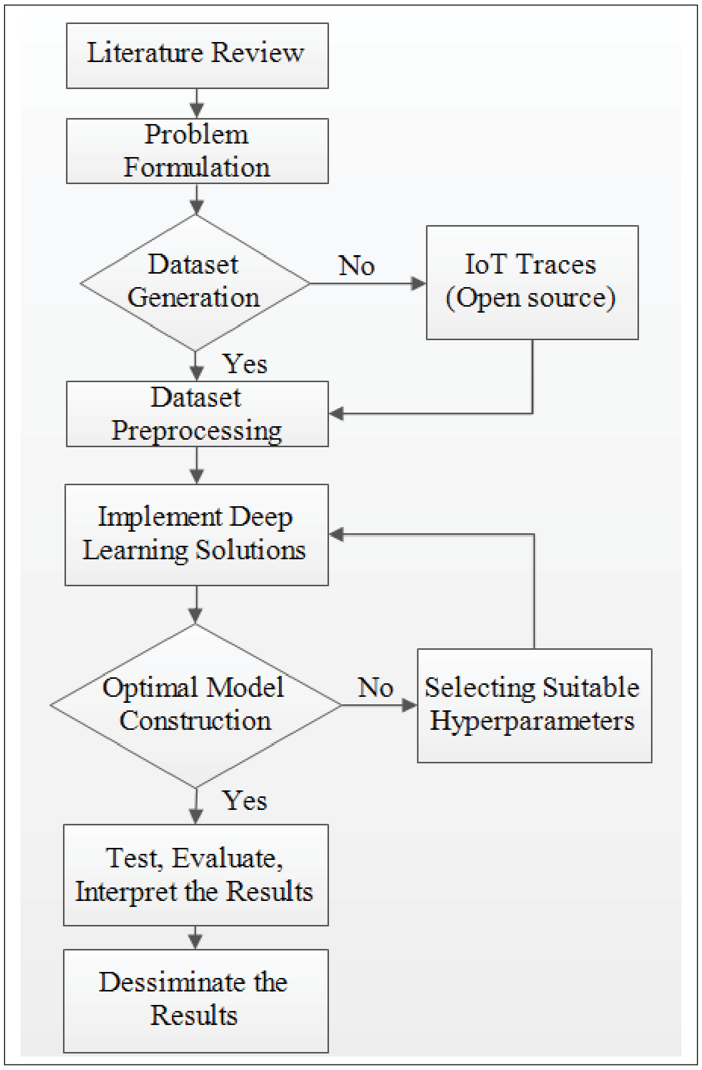


Figure 0.2 The Thesis Methodology

0.6 Publications

The following publications appears in the chronological order of their submission:

1. **Aroosa Hameed**, John Violos, Nina Santi, Aris Leivadeas, Nathalie Mitton, "FeD-TST: Federated Temporal Sparse Transformers for QoS prediction in Dynamic IoT Networks," (Under review with IEEE Transactions on Network and Service Management)
2. **Aroosa Hameed**, John Violos, Aris Leivadeas, Nina Santi, Rémy Grünblatt, and Nathalie Mitton, "Toward QoS Prediction Based on Temporal Transformers for IoT Applications," in IEEE Transactions on Network and Service Management, vol. 19, no. 4, pp. 4010-4027, Dec. 2022, doi: 10.1109/TNSM.2022.3217170.
3. **Aroosa Hameed**, John Violos and Aris 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.
4. Faical Sawadogo, John Violos, **Aroosa Hameed** and Aris Leivadeas, "An Unsupervised Machine Learning Approach for IoT Device Categorization," 2022 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Athens, Greece, 2022, pp. 25-30, doi: 10.1109/MeditCom55741.2022.9928766.
5. **Aroosa Hameed**, John Violos, Nina Santi, Aris Leivadeas, Nathalie Mitton, "A Machine Learning Regression Approach for Throughput Estimation in an IoT Environment," 2021 IEEE International Conferences on Internet of Things (iThings), Melbourne, Australia, 2021, pp. 29-36, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics53846.2021.00020.
6. Stylianos Tsanakas, **Aroosa Hameed**, John Violos, and Aris Leivadeas, "An Innovative Neuro-Genetic Algorithm and Geometric Loss Function for Mobility Prediction." 2021 in Proceedings of the 19th ACM International Symposium on Mobility Management and Wireless Access (MobiWac '21). Association for Computing Machinery, New York, NY, USA, 25–32. https://doi.org/10.1145/3479241.3486706
7. Nina Santi, Rémy Grünblatt, Brandon Foubert, **Aroosa Hameed**, John Violos, Aris Leivadeas, and Nathalie Mitton, "Automated and Reproducible Application Traces Genera-

tion for IoT Applications." 2021 in Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet '21). Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/3479242.3487321

8. **Aroosa Hameed** and Aris Leivadeas, "IoT Traffic Multi-Classification Using Network and Statistical Features in a Smart Environment," 2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Pisa, Italy, 2020, pp. 1-7, doi: 10.1109/CAMAD50429.2020.9209311.

0.7 Thesis Organization

The rest of this document is organized as follows. First, we provide the related work for each contribution in Chapter 1. Then, each of the core sections focuses on one of the aforementioned contributions: Chapter 2 is for the classification model of the IoT traffic at the device level using various feature sets. Chapter 3 presents the QoS prediction at the application level, where each application consists of a heterogeneous data generation distribution but with static sensor nodes. Chapter 4 provides the last contribution of the thesis, which is the QoS prediction when mobility and various network configurations are involved, using a more distributed and privacy aware FL approach and temporal sparse transformers. Finally, the conclusion chapter summarizes our findings and provides an outlook on current and future work.

CHAPTER 1

RELATED WORK

In this chapter, we review the state-of-the-art approaches relevant to various aspects subject to this thesis. We organize the discussion by outlining the pertinent literature associated with each of our contributions. Firstly, we delve into the literature review related to the multi classification of IoT devices and their traffic in a smart environment. Secondly, we survey a variety of mechanisms for predicting or forecasting QoS for IoT applications at the Edge. Thirdly, we present the literature corresponding to QoS predictions in a distributed context. Finally, we conclude the chapter with a concise discussion emphasizing the gaps in the state-of-the-art and the originality of our research. However, a thorough analysis of the state-of-the-art mechanisms is reserved for each separate technical chapter, where we compare our propositions with existing works relevant to each contribution.

1.1 IoT Device Traffic Multi Classification in a Smart City Scenario

The literature focusing on IoT traffic and device classification, forms a significant part of the foundation for this thesis. It is noteworthy that most of the existing research in this area leans towards the application of device fingerprinting or various machine learning approaches. Device fingerprinting typically involves identifying unique attributes or ’fingerprints’ of IoT devices to aid in their classification. On the other hand, machine learning-based approaches rely on algorithms that can learn from and make decisions based on data. Both strategies have shown potential in enhancing the classification accuracy and efficiency of IoT traffic and devices, thereby contributing significantly to the evolution of smart environments.

The authors in (Pinheiro, de M. Bezerra, Burgardt & Campelo, 2019) developed an IoT Device Fingerprinting (DFP) approach that can effectively differentiate between various kinds of network traffic, specifically IoT and non-IoT traffic traces. Additionally, the model can identify individual IoT devices using statistics of packet length such as mean, standard deviation, and the total number of bytes transmitted for a 1 second window as time duration. For this, the proposed approach employed five different models such as k-Nearest Neighbors (kNN), Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM) and a Majority Voting. After the performance evaluation, the authors concluded that the random forest classification model achieved the best results in categorizing 21 IoT and 7 non-IoT devices as compared to other models. However, this approach relies on a single feature i.e., the packet length, to distinguish IoT traffic from encrypted traffic traces. While this simplicity can be an advantage in terms of computational cost, it may not be sufficient to capture the complexity and diversity of IoT traffic characteristics. The risk is that the model might perform poorly when faced with more complex or varied scenarios that go beyond what the single feature can capture.

In another work by (Hui *et al.*, 2022), authors provide a DFP model which differentiates the IoT and non-IoT devices. The particular DFP model aimed to identify five distinct categories of IoT devices by analyzing network traffic features across three levels: packet-level that includes the packet number and size, traffic-level i.e., flow volume, and the mobility-level which consists of location information and mobility entropy. They calculated 22 statistical features which were employed to train seven different classifiers. However, the Random Forest classifier exhibited a classification accuracy exceeding 95% in distinguishing devices based on the provided features. However, as the number of IoT devices grows, the scalability of the system could be a concern. The proposed work does not address how the classifiers would perform with a significantly larger number of devices, or how they would manage real-time classification of a growing number of devices.

The researchers in (Chowdhury, Aneja, Aneja & Abas, 2020) opted to extract 86 features solely from the TCP/IP packet header information. Firstly, the authors assessed the variability, stability and suitability of features and assigned score to each of feature. Then the features selection is performed using the user defined threshold value. Finally, two machine learning models are applied such as J48 and PART to classify the IoT devices using the selected feature sets. However, the selection of features based on a user-defined threshold value could lead to sub-optimal choices. If the threshold is set too high, important features might be excluded. If it is too low, the model might become too complex, leading to overfitting and poor generalization to new data.

The authors in (Fan *et al.*, 2020) introduced a semi-supervised model that derived feature vectors of 219 dimensions from four attributes such as time interval, traffic volume, protocols (TCP, UDP, ICMP, DNS, and DHCP), and Transport Layer Security (TLS) related features. These features were computed from traffic traces and served as distinctive fingerprints for device identification. After this, the proposed model utilized a Convolutional Neural Network (CNN) architecture and achieved a remarkable accuracy on the UNSW dataset. However, the approach uses feature vectors of 219 dimensions, which may make the model computationally intensive and prone to the "curse of dimensionality," where the amount of data needed grows exponentially with the number of features.

The researchers in (Bai, Yao, Kanhere, Wang & Yang, 2018a) divided network traffic data into 5-minute segments to compute six statistical characteristics from four network traffic aspects: traffic volume, packet length, network protocols, and network traffic directions. The goal was to classify the semantic category of IoT devices. An innovative classification model, which is a cascade of Long Short-Term Memory (LSTM) and CNN, was developed for categorizing four categories of devices. The researchers reported an average accuracy of 74.8% when utilizing the UNSW dataset (Sivanathan *et al.*, 2018), considering only 15 IoT devices. However, the choice of 5-minute segments for traffic data might be arbitrary and may not capture all relevant patterns in the data. For example, some important network events could span shorter or longer periods, and this might affect the accuracy of the classification model.

In the study (Aneja, Aneja & Islam, 2018), the authors focused on DFP by exploiting the Inter Arrival Time (IAT), which represents the time difference between the receipt of two successive packets. It was noted that IAT is distinct for each device, given the variations in hardware and software used. Their research proposed an innovative concept by generating graphs of IAT for packets, each graph representing 100 IATs, and subsequently processing these graphs for device identification. A CNN was employed to identify the devices, and the approach achieved an accuracy rate of 86.7%. However, the experiments conducted as part of this work only considered two devices. This is a very small sample size given the diversity and number of IoT devices in practical settings. As a result, the ability to generalize the findings of this proposed work to a wider variety of devices is questionable. Given that IoT devices can greatly vary in terms of their network behavior, hardware, and software configurations, it is important to validate the approach with a larger and more diverse set of devices to ensure its robustness and applicability in real-world settings.

Furthermore, the authors in (Aneja, Aneja, Bhargava & Chowdhury, 2022) proposed a method that involved the use of a statistical analysis of IAT values to create device fingerprints, which facilitated the identification of network-connected devices through wire-side network traffic traces. They used IAT values from a sequence of 1000 packets to generate these unique device fingerprints. Utilizing a Residual Network (ResNet)-50 CNN model, their approach attained a high level of precision, with an accuracy exceeding 97.7% in pinpointing individual devices in the Georgia Tech ID (GTID) dataset. However, generating IAT values for a sequence of 1000 packets for each device could become computationally expensive, especially with an increasing number of devices and this may limit the scalability of the approach.

Next, in the work by Kotak & Elovici (2021), the authors put forth a deep learning (DL)-based model aimed at categorizing network-connected devices. They transformed TCP payload data into unique fingerprints, represented as 28x28 pixel grey-scale images. This conversion process entailed transforming a packet capture (PCAP) file (excluding the header) into a binary file with hexadecimal values, which was then converted into an image. This process was performed using data from a single TCP session to create these fingerprints. They then applied their CNN-based model to network traffic traces from ten devices, encompassing both IoT and non-IoT devices, in order to evaluate its classification capabilities. However, this approach relies on the TCP payload data for creating fingerprints and in many practical scenarios, the payload data might be encrypted, limiting the applicability of this approach. Moreover, the payload data can vary significantly for different sessions of the same device, which might affect the model's performance.

Finally, Sivanathan *et al.* (2019) equipped a smart environment with 28 diverse IoT devices and over a span of six months, the authors gathered and compiled the traffic traces from this setup. Then, the authors offered insights into the inherent network traffic features, utilizing statistical properties such as activity cycles, port numbers, signaling patterns, and cipher suites. Finally, they devised a multi-stage, machine learning-based classification algorithm in which the Naive Bayes algorithm is employed at the stage 1 and the random forest is employed at the stage 2. The authors illustrate the model's capacity to identify specific IoT devices based on their network activity. However, the use of Naive Bayes in the first stage assumed that the features are independent of each other and its performance can suffer if the naive assumption of independence does not hold in the data.

1.2 IoT QoS Prediction per Application based on Temporal Transformer

In an pertinent literature, there is considerable focus on the QoS prediction techniques as well, especially those that leverage machine and deep learning models, within the context of IoT and web services. Scholars have advanced the field by developing cutting-edge methods for QoS prediction, utilizing a combination of diverse CF approaches and machine learning strategies. Therefore, in the rest of this Section, we present the most significant related studies regarding QoS prediction.

For the CF based QoS approaches, Jia *et al.* (2022) predicts the response time and throughput by introducing a prediction model that integrates both local and global location data of various services and employed a CF based MLP to capture complex, high-dimensional non-linear associations between users and web services. Moreover, they also utilize the dot product to aid in the understanding of simpler, low-dimensional linear relationships. However, MLP requires manual feature selection, which can be time-consuming and needs domain expertise. Next, the authors in (Li, Wu, Chen, He & Hsu, 2022b) proposed a Topology-Aware Neural (TAN) model, which leveraged the network topology among the users and services by using IP addresses and autonomous systems, while it also used graph convolution to capture the cross correlation among them and employed the BiLSTM for QoS prediction i.e., throughput, response time and reliability. For this, firstly they projected features to a shared latent space, which is then used as an input attribute to the model. The model is structured to embody the invocation process by capturing path features and end-cross features via a designated path modeling layer and an implicit cross-modeling layer. Following this, a gating layer integrates these features and forwards them to the prediction layer, the role of which is to estimate unrecorded QoS values. However, in the real-world application of this model, it might often be the case that data such as IPs are not available for all users or services. This could be due to privacy considerations, incomplete records, or other factors. As a result, the TAN model might only be applicable to a subset of the total users or services, thereby limiting its utility. Furthermore, the need to remove a substantial number of users/services from the experiment due to missing data could introduce bias or impact the results in ways that are hard to quantify.

Wu *et al.* (2017) introduced a deviation-based neighborhood model specifically tailored for context-aware QoS prediction i.e., response time in the realm of IoT. The method they propose consists of two key stages: Firstly, carrying out a primary QoS prediction estimation, which is grounded on deviations related to both the service and the user. Secondly, they conducted refinements of these predictions by utilizing item-based CF. However, this approach struggles with the data sparsity as it is a memory-based CF, particularly in its second stage where it refines primary QoS predictions using item-based CF. Data sparsity refers to the problem of having a large number of items (or services in the case of IoT) and users, but each user has only interacted with or rated a small fraction of the items. As a result, the user-item interaction matrix (or user-service interaction matrix in the IoT context) is mostly filled with unknown values which makes it difficult for similarity estimation and feature capture.

Furthermore, the authors in (Chen, Yu, Zheng, Shen & Guo, 2022) introduced a context-aware feature interaction strategy intended for the QoS prediction of IoT services i.e., response time and throughput. The proposed methodology is designed to recognize both low-level and high-level feature interactions using context-based data and user invocation records. It operates in three stages: First, it comprehends low-order feature interactions by breaking down the sparse QoS matrix between the user and service with the aid of a factorization machine. Second, it learns high-order feature interactions both explicitly and implicitly via a MLP and a deep cross network. Finally, it combines the results of both low and high-order feature interactions using a fusion based on a parametric matrix. However, this approach might not inherently distinguish the importance of different feature interactions. This means that all feature interactions could be treated equally, which might not reflect their actual importance in predicting the QoS. By assigning equal weight to all feature interactions, the model could over-emphasize unimportant interactions and under-emphasize important ones. This could lead to less accurate predictions.

The authors in (Liang, Chen, Yin, Zhou & Ying, 2022) introduced a CF based response time prediction, termed as Recurrent Neural Network-based Collaborative Filtering (RNCF). This method specifically integrates a multi layer Gated Recurrent Unit (GRU) structure within the neural CF framework to model the dynamic status of physical environments or network conditions, and to distribute the invocation records over various time intervals. GRUs are designed to handle the vanishing gradient problem better than simple RNNs, however, they can still struggle with very long sequences of data. If the invocation records span long time intervals, this could reduce the effectiveness of the method. It should be noted that most of these above described CF based approaches performed the QoS prediction for either cloud services or various web services and considered only response time, throughput and reliability as their QoS metrics for prediction.

Next, there are various studies that applied machine learning techniques to predict the QoS metrics for IoT. For this, the authors in (Abdellah, Abdulkareem Mahmood & Koucheryavy, 2020a) performed the delay forecasting in IoT communication by utilizing both multistep ahead prediction (MSP) and single-step ahead prediction (SSP) methods with Time Series Nonlinear Autoregressive with Exogenous Input (NARX) Recurrent Neural Networks. The accuracy of the predictions were assessed using three distinct neural network training algorithms: Levenberg-Marquardt (Trainlm), Conjugate Gradient with Fletcher-Reeves updates (Traincgf), and Resilient Backpropagation (Trainrp) in term of Mean Square Error (MSE), Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE). However, they used a simulated dataset of an IoT environment and simulated data might not capture all the complexities, noise, and variability of real-world data. Next the authors in (Ateeq, Ishmanov, Afzal & Naeem, 2019a) suggested the prediction of the delay metric using straight forward Deep Neural Network (DNN) which encompasses forward and backward passes, as well as an examination of hyperparameters. The analysis yielded promising outcomes, particularly with regards to the dimensions of the training data, quantity of layers, number of neurons in each respective layer, and epochs. The characteristics employed in this research were derived from various layers of the network, including the application layer, the MAC layer, and the physical layer. The same authors in (Ateeq, Ishmanov, Afzal & Naeem, 2019b) predicted the energy consumption and Packet Delivery Ratio (PDR) using different regression models such as linear regression, gradient boosting, random forest, and DNNs by drawing upon association among communication parameters. However, the performance of DNNs could be influenced by the way the data is selected and processed in the time domain. If not handled properly, this could lead to issues such as leakage of future information into past data or overlooking temporal dependencies.

Additionally, Hou *et al.* (2021) proposed a throughput prediction method by deploying a CNN combined with a target vectorization technique. This was chosen due to their throughput distribution being centralized and focused around certain values. However, it should be noted that the data for this study was produced from a simulated factory scenario and the model was trained on a specific factory use case. Subsequently, the authors of (Abdellah, Artem, Muthanna, Gallyamov & Koucheryavy, 2020b) conducted a study where they used a LSTM network to predict the throughput of IoT traffic within a 5G communication network. The data for this research was produced via an IoT traffic generator and included features such as timestamps, byte counts, and packet counts. However, LSTM cannot handle the long term dependencies adequately, as capturing dependencies over extremely long sequences for LSTM is challenging due to the limitations in memory and computation.

1.3 IoT QoS Prediction based on Mobility using Federated Learning

Similar to the above subsection, which performs the literature review of our second objective i.e., to the QoS prediction of various IoT applications, this part of the section emphasizes more on distributed QoS prediction by considering the mobility of IoT devices, which is the topic of our third contribution. As the literature is similar and is already presented in the above section, we just present the state-of-the-art literature specifically regarding the distributed QoS mechanisms at the Edge and IoT.

The authors in (Zhang, Pan, Qi & He, 2021b) introduced a distributed edge Quality of Service (QoS) prediction (DEQP2) model designed specifically for Edge Computing Networks (ECNs) that emphasizes privacy preservation and utilized the Laplace vector mechanism for distributed privacy protection processing at the edge. In this framework, a user engages edge services by connecting to the nearest edge server, where corresponding QoS records are safeguarded. Following this, the cloud center carries out QoS prediction based on CF. The DEQP2 model harnesses the distributed characteristics of the EC paradigm, deploying the privacy-preserving process to the edge servers. Subsequently, the authors proposed the Distributed Edge Differential Privacy (DEDP) QoS prediction algorithm, which takes into account both user preferences and the edge environment to create a more comprehensive predictive model.

The authors in (Zhang, Jin, Dong, Song & Bouguettaya, 2022) proposed Edge-Laplace QoS prediction method, to address the user mobility and information leakage challenges often faced in QoS prediction within mobile edge environments. Edge-Laplace QoS is designed to precisely and efficiently predict the QoS of various web services and for this an edge region is partitioned into various geographical locations to acquire accurate edge QoS data information and accommodate the dynamic nature of the edge environment. Furthermore, to safeguard the user privacy in these mobile edge environments, the authors applied a differential privacy technique to add dynamic disguises to the original QoS data within the edge environment. Moreover, a CF method is utilized to identify and utilize similar users' access records based on the geographic locations of their accessed servers for QoS prediction. However, both of these above methods applied privacy protection mechanisms, which can potentially introduce noise into the data, that may impact the accuracy of the QoS predictions. Therefore, there is a trade of between privacy and accuracy.

Next, Qi *et al.* (2018) took into account a distributed scenario, where historical QoS data is dispersed across multiple platforms. This refers to QoS values for the same services that are collected from a multitude of users across various platforms. For this, the authors integrate Locality-Sensitive Hashing (LSH) and one of its variants, MinHash, into mobile service recommendations and propose a novel service recommendation methodology based on a two-stage LSH. This approach aims to provide privacy-preserving and scalable mobile service recommendations. However, Locality-Sensitive Hashing, despite its effectiveness, is not immune to hash collisions, where different inputs produce the same output hash and this could lead to inaccuracies. Further, this study is not taking into consideration the dynamic fluctuation of QoS data over time which may occur in a varied environment.

The authors in (Zhang, Zhang, Luo & Luo, 2020b) present a privacy-preserved QoS prediction of services methodology. The proposed prediction model is co-trained on both the user and server end by employing FL techniques within a decentralized framework. To ensure user privacy, only the parameters of the local model need to be shared with the central server, negating the need to transmit substantial quantities of local data. Moreover, authors introduced a federated matrix factorization algorithm with the goal of decreasing the necessary communication rounds by augmenting the number of local training epochs of the model. However, matrix factorization can struggle with the sparsity of data that impact the quality of the local models trained on each device, which in turn affects the aggregated global model.

Li *et al.* (2022c) proposed a personalized federated tensor factorization framework for QoS prediction of an IoT service in a distributed setting. Initially, they utilized tensors to depict multidimensional QoS data and established a unique personalized model for each edge server. Subsequently, each edge server conducts local tensor factorization and interacts with the parameter master, learning information from other edge servers during the training process. By ensuring coherence between the global public component and local personalized public components, the global model is capable of learning information from edge servers throughout the training phase. However, in this framework, the participant role is played by an edge server, which is equipped to gather user data, as opposed to being a mobile user device.

Further, authors in (Jin, Zhang, Dong, Zhu & Bouguettaya, 2023) proposed a privacy-aware QoS forecasting model called Edge-PMAM (Edge QoS forecasting with Public Model and Attention Mechanism) that consists of four steps: Firstly, the authors gathered the edge location information and QoS datasets to create a spatio-temporal edge user QoS dataset. Longitude and latitude values in the edge location information actually helped to determine the geographical distribution of edge servers. Secondly, they transformed the longitude and latitude values of the edge servers into plane coordinate values using the Miller projection and used the K-means clustering to decide the number of clusters of the plane coordinate data, based on the elbow method or silhouette coefficient, hence dividing the whole area into different regions. Thirdly, public model training is performed by using the public dataset of each region to train an LSTM model with an attention mechanism to acquire the public model. The weight parameters of these public models will be transferred to users of corresponding regions for personalized forecasting. Fourthly, a personalized forecasting is performed. For this process, a user incorporates the weight parameters from the public model associated with their specific region as the initial parameters for their individual private model. The private model is then trained on the user's private data, which consists of the temporal QoS data generated by user and service interactions and a continuous update of the weight parameters of the private model is performed. Each user's private model is optimized in a continuous fashion with the training iterations. However, the effectiveness of the public model is based on the quality of the clusters determined by the K-means algorithm, which is highly dependent on initial cluster centers and is susceptible to converge to local optima, resulting in sub-optimal clustering and therefore affecting the model's performance.

Moreover, Zhang et al. established a decentralized QoS prediction framework using FL along with the matrix factorization technique by encompassing a local model trained on the user side and a global model jointly trained on the server side with a differential privacy in (Zhang, Zhang, Luo & Ji, 2020a). The authors also incorporate a user reputation mechanism into the model to assess the trustworthiness of different users, while aiming to forecast the response time and throughput of various services. However, if the reputation scoring is not robust, it could unfairly impact the contribution of certain users to the global model.

1.3.1 Conclusion and originality of the research

In conclusion, regarding the classification of the IoT traffic and devices, most existing methods are based on two core strategies: device fingerprinting and machine learning techniques. While these strategies have been effective to some extent in differentiating between IoT and non-IoT devices and their traffic, there are different shortcomings associated with them. Some of the existing methods rely heavily on a single feature, such as packet length, to distinguish between different types of IoT traffic. This could limit the model's ability to handle the complexity and variability of IoT traffic, which can change depending on the type of device, its function, the network conditions, etc. For instance, two different devices might send data packets of similar length, but their network behavior can still be very different based on factors like transmission frequency, protocol used, or interaction with other devices.

Furthermore, deciding on which features to include in the model is also a major challenge. The wrong choice can lead to sub-optimal model performance or make the model overly complex, leading to overfitting (where the model performs well on the training data but poorly on new, unseen data). On the other hand, if the classification models involve high-dimensional feature vectors then it can also be computationally intensive and making them unsuitable for scenarios where computational resources or power are limited. Therefore, to address these issues, in this thesis, we incorporate a fine-grained feature set at different network levels such as flow, device and packet level. A fine-grained feature set can capture the detailed and diverse information about the network traffic. However, to make the proposed model computationally efficient, a systematic and effective feature selection mechanism i.e., ANOVA is employed, to ensure that the most relevant and informative features are used for model training thus, enhancing the model performance and reducing the risk of overfitting. In the existing literature, there are also some studies that exhibits scalability problems i.e., with the increase in number of IoT devices. Hence, the ability of these methods to process and classify data from a larger number of devices in real-time becomes a concern. Accordingly, we propose a two stage learning framework that uses the logistic regression at stage 0 and MLP at stage 1 to produced the classification results of IoT traffic related to 28 different devices, which splits the complexity between the two stages.

Regarding the QoS prediction of IoT applications, there exist two approaches as either CF based methods or machine learning based solutions. For the memory based CF methods, the data sparsity problem is the most challenging one. For the matrix factorization based CF methods feature association patterns can be captured in the data, however, only the interaction of linear features is taken into account in such methods and the complicated interaction of nonlinear features is ignored, which causes less efficient representation learning. Furthermore, most of these CF-based techniques have been traditionally employed to predict QoS for web services. The QoS metrics taken into consideration typically include response time, throughput, and service reliability. However, their effectiveness when applied to IoT environments, which might have different or additional considerations, is not explicitly mentioned and could be a matter of further exploration.

On the other hand, for the neural network based solutions some of the works have not thoroughly examined the actual prediction task with respect to time, especially in emerging IoT application scenarios. This can become critical in the context of IoT, where time-series data play a significant role. Moreover, some Machine Learning (ML) studies are primarily designed for short-term sequence predictions. These models often encounter difficulties when dealing with long sequence dependencies, meaning they have trouble learning from patterns that occur over large periods of time or many data points. This limitation restricts their ability to train on long sequences of data, which could be a crucial factor in many IoT applications where historical trends significantly influence future outcomes. Therefore, in this thesis, we provide a 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. Following, we provide the multistep prediction of each QoS in both univariate and multivariate settings and in order 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 was the first work which provided a transformer based QoS prediction for IoT applications.

Regarding the QoS prediction with respect to mobility, some studies have implemented specific privacy-preserving methods in their models. While these solutions are instrumental in maintaining user confidentiality, they can have an adverse effect on the accuracy of the model. Further, the studies that applied matrix factorization techniques in a distributed environment led to the issue of data sparsity. Moreover, some of the related research has utilized FL, however, the performance of their global model can be heavily influenced by the parameters of the specific machine learning model used. Therefore, the task of correctly setting the parameters for the machine learning model becomes vital. In this thesis, we solve these problems in such a way that we propose a FL approach that trains the client models on diverse IoT devices and sensors data, resulting in a more comprehensive QoS forecasting. For this, we introduce a temporal transformer with sparse attention that can efficiently model long sequences by capturing dependencies between different positions without suffering from the vanishing gradient problem.

CHAPTER 2

A DEEP LEARNING APPROACH FOR IOT TRAFFIC MULTI-CLASSIFICATION AT THE EDGE

Aroosa Hameed[1] , John Violos[1] , Aris Leivadeas[1]

[1]Département de génie logiciel et des TI, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Paper published in *IEEE Access*, February 2022

2.1 Abstract

As the number of Internet of Things (IoT) devices and applications increases, the capacity of the IoT access networks is considerably stressed. This can create significant performance bottlenecks in various layers of an end-to-end communication path, including the scheduling of the spectrum, the resource requirements for processing the IoT data at the Edge and/or Cloud, and the attainable delay for critical emergency scenarios. Thus, a proper classification or prediction of the time varying traffic characteristics of the IoT devices is required. However, this classification remains at large an open challenge. Most of the existing solutions are based on machine learning techniques, which nonetheless present high computational cost, whereas they are not considering the fine-grained flow characteristics of the traffic. To this end, this paper introduces the following four contributions. Firstly, we provide an extended feature set including, flow, packet and device level features to characterize the IoT devices in the context of a smart environment. Secondly, we propose a custom weighting based preprocessing algorithm to determine the importance of the data values. Thirdly, we present insights into traffic characteristics using feature selection and correlation mechanisms. Finally, we develop a two-stage learning algorithm and we demonstrate its ability to accurately categorize the IoT devices in two different datasets. The evaluation results show that the proposed learning framework achieves 99.9% accuracy for the first dataset and 99.8% accuracy for the second. Additionally, for the first dataset we achieve a precision and recall performance of 99.6% and 99.5%, while for the second dataset the precision and recall attained is of 99.6% and 99.7% respectively. These results show that our approach clearly outperforms

other well-known machine learning methods. Hence, this work provides a useful model deployed in a realistic IoT scenario, where IoT traffic and devices' profiles are predicted and classified, while facilitating the data processing in the upper layers of an end-to-end communication model.

2.2 Introduction

Internet of Things (IoT) allows tens of billion devices to be connected over the Internet. Nonetheless, the rapid increase of IoT devices has also resulted in a colossal increase of the data generated by IoT devices. Specifically, the total data has quadrupled in just five years from 145 ZB in 2015 to 600 ZB in 2020 (Ivanov, 2019). Furthermore, IoT not only enables new applications, but introduces new types of devices as well. For example, in the context of a smart environment, thousands of non-traditional Internet devices are used including smart sensors, alarms, traffic lights, cameras, weather stations, etc. generating an unprecedented volume of data for a variety of smart applications such as healthcare, industrial control, transportation and so on. However, these IoT devices are usually of limited computational abilities (Mukhopadhyay & Suryadevara, 2014) and cannot manipulate locally the data generated.

This often urges the offloading of computational hefty IoT tasks to a remote infrastructure, a process called task offloading (Saeik *et al.*, 2021). Edge Computing (Mao, You, Zhang, Huang & Letaief, 2017) is a viable solution for the task offloading as it allows to offer the necessary networking and computational resources at the edge of the network enabling at the same time the real time processing of the IoT data. However, as explained in (Dechouniotis *et al.*, 2020), it is extremely difficult to estimate the edge resources needed due to the fact that (i) the IoT data are randomly generated, as a consequence of the different types of devices and their dynamic cycle activity; and (ii) when there is a large number of IoT devices, the total communication delay may be affected on account of the constrained nature of the IoT access networks.

Hence, the importance to predict the time varying characteristics of the IoT devices (such as activity patterns, signaling patterns etc.) becomes evident. Furthermore, the classification of similar devices facilitates the estimation of the generated workload and can better guarantee a specific level of Quality of Service (QoS). Therefore, by classifying the IoT devices into different categories, the prediction of traffic characteristics can be more efficiently done. Additionally, a more accurate prediction of the resource requirements at the IoT access network (i.e. spectrum) and Edge infrastructures (i.e. computational and communication resources), can be achieved.

However, such an IoT device classification, often called device fingerprinting (Xu, Zheng, Saad & Han, 2016), presents several challenges. In particular, the existing IoT classification techniques do not consider the fine-grained characterization of IoT traffic, while they suffer from high computational cost for the data extraction and processing, and are often affected by high dimensional data and complexity. Accordingly, in this paper, we propose a two-stage based deep learning architecture in order to classify the IoT devices by considering a fine-grained set of network characteristics (features). To do so, firstly, we propose a two-step preprocessing algorithm while employing a feature selection and prioritization technique for the feature set under consideration. Our approach, facilitates the distribution of the features in the two stages avoiding the high dimensionality and overfitting problems of the training data.

The novelty of this paper lies in proposing a very accurate but considerably more light-weighted approach than the existing ones. Furthermore, the feature selection and prioritization along with the combination of a deep learning model creates a unique and innovative approach for the problem of the IoT device classification. The novelty of our approach is strengthened by the fact that it can be generalized and applied in different datasets without losing any accuracy. Thus, the reproducability of the results and the stability of our approach in different IoT contexts fortify the originality introduced.

In particular, the major contributions and novelty of this paper can be summarized as follows:

1. In order to perform a classification of the IoT devices, we have suggested an extended feature set comprising of flow, device, and packet level features. This approach provides a fine grained characterization of the traffic flow with less computational complexity for the classification.

2. A two step preprocessing algorithm is proposed that assigns relevance weights to the nominal (representing the qualitative data with numeric codes) features and provides scaling of the dataset using a MinMaxScaler method.
3. A statistical feature selection technique is employed to select the features with regard to their contribution to the classification of IoT devices. Furthermore, an investigation of correlated features at each level is provided using the Pearson correlation coefficient.
4. A two stage learning framework is presented with 99.9% accuracy for the first dataset under consideration and 99.8% for the second one, which proves the generalization of our approach. To determine the IoT device classification, we compute the classes for certain nominal and multi valued attributes at learning stage 0 using logistic regression. Following, we perform the final classification for numerical and single-valued features at stage 1 using a multi layer perceptron (MLP) neural network. The MLP network takes as an input a feature subset at each time and classifies IoT devices in a context of a smart environment. Furthermore, to achieve the optimal or near optimal MLP architecture, a random search based keras tuner is employed.

The rest of the paper is structured as follows: Section 2.3 highlights the related work in traffic classification, covering the most important methods and technologies applied in the IoT traffic classification domain. Section 2.4 provides the system model and necessary preliminaries for comprehending the classification problem in the context of the IoT domain. Additionally, this Section covers the description of the feature sets, their statistical characteristics and feature correlation, information that is necessary for the domain of data analysis that our paper touches upon. Section 2.5 presents the two-stage proposed learning framework for the IoT device classification problem. Section 2.6 explains the algorithmic form of proposed preprocessing and learning model along with their asymptotic analysis. Sections 2.5 and 2.6 fall under the domains of deep learning, machine learning and problem complexity, presenting all the necessary technical details. Section 2.7 provides the performance evaluation results for both datasets under consideration. The conclusions and the future directions of this work are presented in Section 2.8. Finally, Table 2.1 presents the set of abbreviations used in this paper.

Table 2.1 List of abbreviations

Abbreviations	**Meaning**
AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
ANOVA	Analysis of Variance
BoW	Bag-of-Word
CMMPP	Coupled Markov Modulated Poisson Processes
DF	Do not Fragment Flag
DT	Decision Tree
FC	Fully Connected
FNR	False Negative Rate
FPR	False Positive Rate
GB	Gradient Boosting
IaT	Interarrival Time
IoT	Internet of Things
IP	Internet Protocol
KNN	K Nearest Neighbor
LR	Logistic Regression
LSTM	Long short-term memory
ML	Machine Learning
MLP-ANN	Multi-Layer Perceptron Artificial Neural Network
MSS	Maximum Segment Size
M2M	Machine-to-Machine
NB	Naive Bayes
NOP	No Option
OP	Output Layer
PDR	Packet Drop Rate
QoS	Quality of Service
RF	Random Forest
ReLU	Rectified Linear Units
RFE	Recursive Feature Elimination
SVM	Support Vector Machine
TTL	Time-to-Live
TCP	Transport Control Protocol
WS	Window Size
WSO	Window Size Option

Table 2.2 Comparison of related works

Category	**Ref**	**Technology**	**Traffic source**	**Features**
Aggregated Traffic Models	(Laner & et al, 2013)	Coupled Markov Modulated Poisson Processes (CMMPP)	Simulated data for 30000 devices	No. of devices, distribution, time period
	(Laner *et al.*, 2015)	Aggregated+SMM+CMMPP	Fleet management	time, packet size, direction, IP address, port no., APN
Fingerprinting	Miettinen *et al.* (2017)	ML based model + SDN based traffic monitoring	Collected 27 IoT devices data	link protocol, network protocol, transport protocol, IP options, IP address, port numbers
	Bezawada *et al.* (2018)	Device Fingerprinting+DT, GBM and Majority voting	Collected data of 14 IoT devices	Packet header features+ payload length, entropy, win.size
	Meidan *et al.* (2017b)	Device white listing + RF	17 devices data	time to live statistical information
	Aneja *et al.* (2018)	4 DFP models+CNN	IPhone/ipad data	636 and 608 IAT graphs of Apple devices used for CNN
	Apthorpe & et al (2017)	Separate and label streams, examine traffic rate	Collected data from four smart devices	IP addresses, TCP ports, DNS queries
Machine Learning	Lippmann & et al. (2003)	KNN, DT, MLP, SVM	OS identification data	WS, TTL, DF, MSS, WSO, port no., NOP
	Kotak & Elovici (2021)	Single layer neural n/w	Traffic data from Sivanathan *et al.* (2019)	TCP payloads converted to grayscale images
	Hameed & et al. (2020)	Regression approaches	Real time IoT application data	Node id, total messages transmitted and received, timestamps, success rate, latency, PDR, throughput
	Santos & et al (2018)	Random Forest	Traffic data from Sivanathan *et al.* (2019)	Packet size, packet volume, IaT, duration, URG & PSH
	Abdellah *et al.* (2020a)	LSTM	Simulation data	Timestamp, bytes count, and the packets count
	Shahid & et al (2018)	RF, DT, SVM, KNN, NN, NB	Generate data from four IoT devices	packets sent, packets received, IaT between packets sent, IaT between packets received
	Lopez-Martin & et al (2017)	CNN+LSTM	RedIRIS dataset	source & dest. port, payload, WS, timestamp, direction
	Meidan *et al.* (2017a)	GBM, RF, XGboost	Network traffic data	source IP, destination IP, port numbers
	Sivanathan *et al.* (2019)	NB+RF Classifier	Collected data from smart lab	port no., domain name, cipher suite, flow volume, duration, rate, DNS interval, NTP interval
	Hameed & et al. (2020)	LR+GBM	Traffic data from Sivanathan *et al.* (2019)	IP and MAC addresses, port no., TTL, protocol, IaT, packet size, traffic rate, burstiness)

2.3 Related work

For the IoT device classification, significant emphasis has been given into aggregated traffic models, fingerprinting, and machine learning based solutions. The aggregated traffic models resort to mathematical and statistical distribution-based methods, which involve several probability distributions and mathematical techniques like stochastic processes to model the traffic. Following, the fingerprinting methods are used to identify the IoT devices leveraging information from network traces in order to correlate datasets. In particular, this category of classification identifies a device using information from the network packets during the communication over the network.

Regarding the ML based schemes, they utilize existing algorithms to automatically learn complex patterns from the IoT traffic data. The algorithms used in these schemes are classified according to how the learning process is conducted. Four main classes are used to group ML algorithms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. However, in the current literature, mostly supervised learning, unsupervised learning or a combination of these two are utilized in order to analyze, predict and model the IoT traffic and device characteristics.

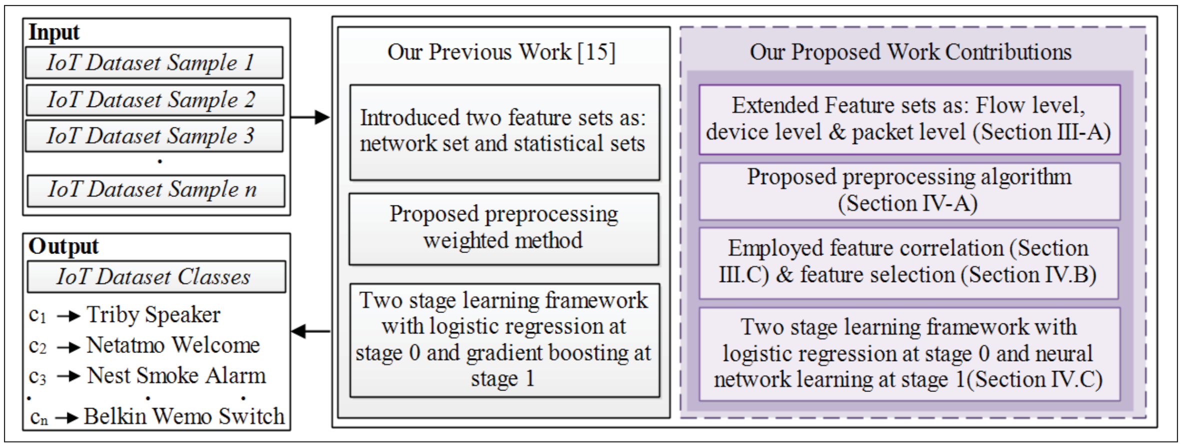

Input
IoT Dataset Sample 1
IoT Dataset Sample 2
IoT Dataset Sample 3
IoT Dataset Sample n
Output
IoT Dataset Classes
c1 → Triby Speaker
c2 → Netatmo Welcome
c3 → Nest Smoke Alarm
cn → Belkin Wemo Switch
Our Previous Work [15]
Introduced two feature sets as: network set and statistical sets
Proposed preprocessing weighted method
Two stage learning framework with logistic regression at stage 0 and gradient boosting at stage 1
Our Proposed Work Contributions
Extended Feature sets as: Flow level, device level & packet level (Section III-A)
Proposed preprocessing algorithm (Section IV-A)
Employed feature correlation (Section III.C) & feature selection (Section IV.B)
Two stage learning framework with logistic regression at stage 0 and neural network learning at stage 1(Section IV.C)

Figure 2.1 Overview of our previous work vs. proposed work contributions (shown in the purple boxes)

With respect to aggregated traffic models, Laner & et al (2013) proposed a Coupled Markov Modulated Poisson Processes (CMMPP) framework to capture the traffic behavior of a single machine-type communication along with the collective behavior of tens of thousands of devices. In (Laner *et al.*, 2015) a classification strategy is designed for a fleet management use case incorporating three classes of M2M traffic states, namely periodic update, event-driven, and payload exchange. The authors in (Orrevad, 2009) proposed a model that estimates the M2M traffic volume generated in a wireless network-enabled connected home. However, the above works do not consider the fine-grained characterization of the IoT traffic, whereas the complexity of such methods grows linearly with the number of the devices. Furthermore, common communication patterns identified can be attributed to any sensing device under a specific use case (limitation 1).

There is also a significant effort to identify the type of the IoT devices using the fingerprinting method. For example, “IoT Sentinel” (Miettinen *et al.*, 2017) is a classification system that can recognize and identify the IoT devices immediately after they are connected to a network using a single attribute vector with 276 network features. The “IoT Sentinel” framework can be further improved by extracting additional network features such as payload entropy, TCP payload length, and TCP window size (Bezawada *et al.*, 2018). Similarly, in (Meidan *et al.*, 2017b) almost

300 network attributes are used from each TCP traffic session to classify the devices, using a majority voting for every 20 consecutive sessions.

The work in (Aneja *et al.*, 2018) utilized a deep learning approach in order to perform the device fingerprinting using the packet interarrival time. However, this approach is computationally intensive as all packet level information is utilized without any selection strategy. In (Apthorpe & et al, 2017), the traffic patterns of encrypted network flows are used to reveal the existence of a specific device inside a home network. However, obtaining such a great number of features require specialized hardware accelerators, thus resulting in high computational cost, longer classification duration and limited scalability due to the need of a deep packet inspection functionality (limitation 2).

Some related works also employed machine learning in order to perform traffic and device classification. Lippmann & et al. (2003) compared the K-nearest neighbor (KNN), Support Vector Machine (SVM), Decision Tree (DT) and Multilayer Perceptron (MLP), using the packet header information and concluded that KNN and DT provide better results. Kotak & Elovici (2021) classified nine different device flows based on the device type using artificial neuron network. Regarding traffic classification, the authors in (Hameed & et al., 2020) predicted the QoS behavior of five different IoT applications in a smart building context, using several regression based ML approaches.

The work in (Santos & et al, 2018) shows how to classify traffic and perform device identification using random forest. The list of key features used in the classification included the packet size, volume of packets, inter-arrival time, duration, urgent and push flags. Additionally, the authors in (Abdellah, Artem, Muthanna, Gallyamov & Koucheryavy, 2020c) performed a prediction of the IoT network traffic using Long Term Short Memory (LSTM). The features of dataset consisted of the timestamp, bytes count, and the packet count. A more comparative approach, was introduced in (Shahid & et al, 2018), where the authors presented a method to recognize the IoT devices using random forest, decision tree, SVM, k-nearest neighbors, simple neural network and naive bayes approaches.

Lopez-Martin & et al (2017) classified the traffic applications using a multi-class neural network, which is proven to be effective in complex data structures. The authors in (Meidan *et al.*, 2017a) proposed an individual binary classification model for each class in order to eliminate the complexity issue of multi-class classification. Sivanathan *et al.* (2019) utilized the statistical attributes, signaling patterns and cipher suites along with machine learning for IoT device classification.

Nonetheless, these ML approaches are affected by the high data dimensionality, they are sensitive to the hyper-parameter tuning and they require a large number of training data. Moreover, the main constraint of the multi-class classification is scalability, as the high number of classes makes the classifier more complex and updating requires full retraining (limitation 3). A summary of the papers reviewed in this section is given in Table 2.2.

In our preliminary work (Hameed & et al., 2020), we tried to address some of these limitations by relying on typical machine learning techniques, such as logistic regression and gradient boosting. In this paper, we extend our preliminary framework to provide a more complete and detailed IoT multi-classification approach based on a deep learning solution. As this research is an extension of our previous study, we used the same IoT dataset (Sivanathan *et al.*, 2019). However, in order to prove the generalization of our proposed methodology we also performed our experiments with a second IoT dataset (Ren *et al.*, 2019). Additionally, herein, we include a more extended feature set at three different levels such as: device, flow and packet.

This work also introduces a feature correlation mechanism, whereas specific features are selected for training models which is not included in our previous work. Furthermore, for the new two stage learning framework, we apply an optimal searched neural network architecture at the second stage. Finally, a completely new performance evaluation section is presented. The particular section includes a new set of results for both datasets, new experiments, and additional comparisons with machine learning and deep learning approaches. The differences between our previous and proposed work are given in Figure 2.1.

The extensions made in this paper are aligned in such a way to address the above cited limitations:

- To overcome limitation 1, we incorporate a fine-grained feature set at different network levels i.e., flow, device and packet level.
- To address limitation 2 and the high computational costs of complex features, we employ a statistical feature selection (i.e., ANOVA score) to select a subset of the available features at a time instance t.
- To address limitation 3, we propose a two-stage learning framework. Firstly, a relevance weighting-based preprocessing is performed for the available features, whereas different subsets of the selected features are utilized across these two stages to avoid the high dimensionality issue. Finally, the tuned hyperparameters are utilized in a neural network that achieves 99.9% accuracy for the first dataset and 99.8% for the second.

2.4 Problem Setup

In this section, we describe and formulate the IoT traffic classification problem, where different IoT devices are combined to their respective classes according to their distinctive characteristics. To help the reader follow the modeling of our work, Table 2.3 summarizes the key notation used throughout this paper.

In particular, a smart environment (e.g. smart city, home, grid, etc.) can be modeled as a network of S smart devices, generating M traffic flows. The devices are represented by the set $D = \{d_1, d_2.., d_s\}$, where d_s indicates the s^{th} smart device, where $1 \leq s \leq S$. Similarly, the set $T = \{t^1_{d_1}, t^2_{d_2}, ..., t^m_{d_s}\}$ represents the generated traffic flows, where $t^m_{d_s}$ denotes the m^{th} traffic flow in T generated by the s^{th} device, with $1 \leq m \leq M$ such that $M \subseteq S$. Furthermore, each traffic flow is constituted by a number of packets denoted by $P = \{p_{1m}, p_{2m}, ..., p_{km}\}$ where p_{km} represents the k^{th} packet of the m^{th} flow.

Regarding the features, the set F denotes the distinctive properties of the traffic flow $t^m_{d_s}$ which we want to classify. Each packet in P is a D-dimensional set of the network elements

Table 2.3 Summary of the key notation

Symbols	Meaning
D	Set of devices
S	Smart devices
M	Traffic flows
m^{th}	Last traffic flows
d_s	Last smart device in set S
T	Set of generated traffic flow
$t^m_{d_s}$	Last traffic flow generated by last device in set S
P	Set of packets generated by traffic flow
k^{th}	Last packet generated in the traffic flow
p_{km}	Last packet generated in the last flow
F	Set of features
f_i	Last feature in the feature set F
G	Set of training instances
X	Set of total sample instances
x_r	Last instance of features in set X
C	Set of classes or labels
c_q	Last class of features in set C
q^{th}	Last class
r^{th}	Any input sample of set X
D	Input vector of specific dimension
f_i	Feature in feature space
cov	Covariance between two features
σ	Standard deviation
ρ	Pearson's correlation coefficient
v	Feature vector
p	Probability for a combination of independent variables
β_0	Intercept
β_n	Regression coefficients
y_i	Dependent variable
x_n	Independent variable
l^{th}	Layer of neural network
$O_i^{(l)}$	Output of the i^{th} neuron at l^{th} layer
V	Nonlinear activation function
w	Weight of a neural network connection
B	Bias value applied at a layer
$O(n)$	Linear complexity
t	Number of training examples
e	Number of epochs
d	Number of neurons in each layer
$\cap$	Intersection between two sets
$\notin$	Not element of
$\Leftrightarrow$	Equivalent of

under consideration. These elements are represented as a feature space F, such that $F = \{f_1, f_2, f_3, .., f_i\}$, where f_i represents the i^{th} feature in the feature space F with $1 < i \leq 11$ (in this work we assume 11 distinctive features).

The set F consists of device, flow and packet level features, where f_1 represents the interarrival time, f_2 denotes the source IP address, f_3 is the destination IP address, f_4 shows the transport protocol used by each flow, f_5 is the source port number, f_6 denotes the destination port number, f_7 is the Time-to-Live (TTL) information, f_8 denotes the window size used by the transport layer, f_9 indicates the length of a packet, and f_{10} denotes the source Ethernet address, and f_{11} is the destination Ethernet address.

Furthermore, we assume that we have a given training set G, including pairs of input samples along with their class labels as $G = \{(x_1, c_1), (x_2, c_2), \ldots, (x_r, c_q)\}$. Accordingly, the set $C = \{c_1, c_2, ..., c_q\}$ denotes the available classes, where $c_q \in C$ represents the q^{th} class in C, while $C \subset D$ and $q \leq n$. Furthermore, $x_r \in X$ is the r^{th} input sample of the total set of samples $X = \{x_1, x_2, ..., x_r\}$, such that $X \subset P$ and $r \leq k$. Hence, the IoT Traffic Classification problem is defined as the task of estimating the class label c_q to the input vector x_r, where x_r belongs to a subset of a feature space F, $x_r \in X \subset F$. This task is accomplished using a classification rule or function $f(x) : X^D \rightarrow C$ that can predict the label C of unseen D dimensional input vector x_r.

2.4.1 Feature description

As mentioned earlier, the available features can be categorized as follows:

2.4.1.1 Device level features

In this category we consider the source and destination MAC addresses of the devices. Such features are extracted directly from the traffic traces. These features offer a characterization of the IoT traffic independent of the other two levels of features.

2.4.1.2 Flow level features

This includes features such as source and destination IP addresses, protocol type of a flow, source and destination port numbers, the TTL information of a flow, and the window size used by the flow. This set can be used to extract the packet level features of a flow described below.

2.4.1.3 Packet level features

This category includes the timestamp, the interarrival time (IaT), and the length of the packets. The interarrival time is the amount of time that elapses between a packet reception and the arrival of the one following it. As timestamp follows the normal (guassian) distribution, to calculate the interarrival time feature, we analyzed and extracted the time between the successive incoming traffic packets following a Gaussian's distribution with an average rate of 1 (since at each time unit one packet arrives). All of the above features along with their description are illustrated in Table 2.4. To prove the generality of our approach, we used the same feature sets for both datasets under consideration.

Table 2.4 Description of features in both datasets

Level	Variable	Features	Description
Packet Level	f_1	IaT	Avg. time b/w two consecutive packet receptions
	f_9	Length	The length of network packet
Flow Level	f_2	IP.src	Source IP address
	f_3	IP.dst	Destination IP address
	f_4	Protocol	Protocol used by the flow
	f_5	Port.src	Port number of the client
	f_6	Port.dst	Port number of the server
	f_7	TTL	Maximum number of hops left for each packet to reach the destination
	f_8	WS	The amount of bytes the receiving device is capable to receive
Device destination Level	f_{10} f_{11}	MAC.src MAC.dst	Source MAC address Destination MAC address

2.4.2 Statistical characteristics of the features

Each feature f_i in the feature space F has its own distribution, which is represented by the number of different statistical characteristics over different smart devices. The analysis of such distributions can be useful in order to identify which features are most important for the classification. In this work, we considered three statistical characteristics of the distribution of each feature, such as: mean, median and standard deviation. Table 2.5 summarizes the statistical characteristics of each feature for both datasets. However, for illustration purposes we plot the probability distribution of the features under consideration for the first dataset only, as shown in Figure 2.2. As can be seen, the interarrival time shows a Gaussian distribution (as explained in the previous subsection), while all other features illustrate an exponential distribution.

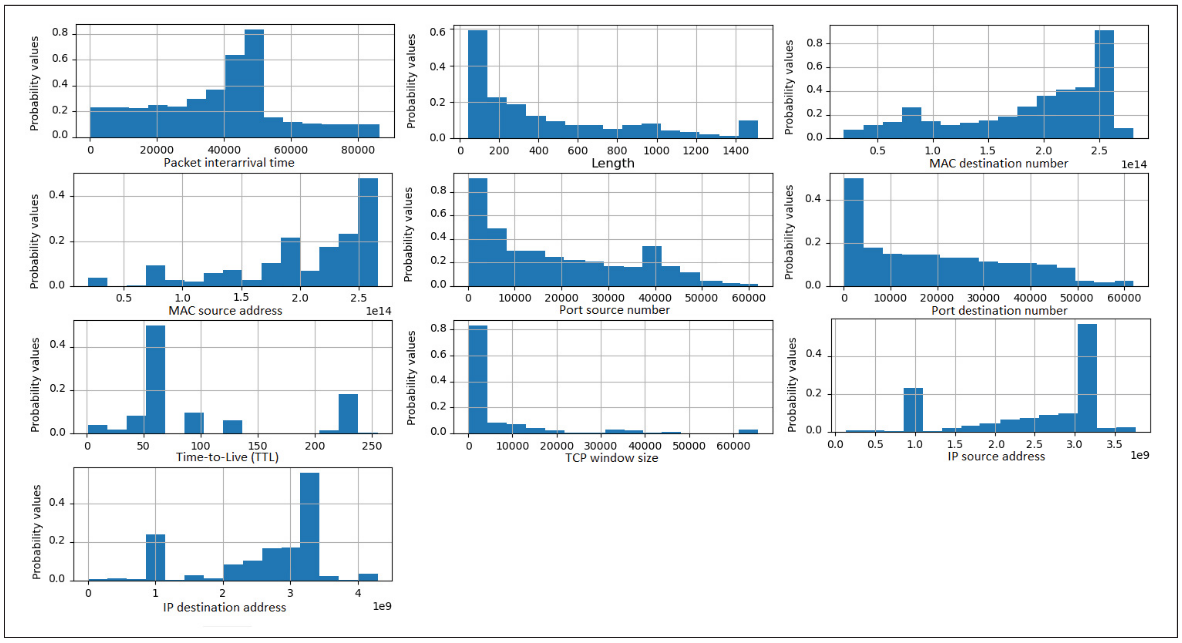


Figure 2.2 Probability distributions of IoT traffic flow features of Dataset 1

2.4.3 Feature correlation

One very important aspect of the performance of the classification is the correlation between the features. Hence, in this work we consider the feature correlation from two perspectives. Firstly,

Table 2.5 Statistical characteristics of IoT traffic features

Datasets	Measures	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}
Dataset 1	Mean	0.45	0.67	0.59	0.50	0.35	0.38	0.37	0.07	0.16	0.74	0.69
	Median	0.50	0.85	0.75	0.65	0.06	0.13	0.24	0.02	0.03	0.86	0.81
	S.D	0.22	0.28	0.25	0.45	0.37	0.37	0.26	0.18	0.30	0.28	0.28
Dataset 2	Mean	0.39	0.001	5.21	0.29	0.38	0.44	0.33	0.09	0.01	0.61	0.46
	Median	0.38	1.45	1.43	0.33	0.11	0.67	0.25	0.01	0.01	0.69	0.39
	S.D	0.39	0.03	0.002	0.12	0.36	0.34	0.20	0.22	0.01	0.19	0.18

we examine which features are correlated within the feature space. The correlation between two features say, f_i and f_j, is calculated using the Pearson's correlation coefficient which is given as:

$$\rho_{(f_i,f_j)} = \frac{cov(f_i, f_j)}{\sigma_{f_i}\sigma_{f_j}} \tag{2.1}$$

where $cov(f_i, f_j)$ is the covariance between features f_i and f_j, whereas $\sigma_{(f_i)}$ and $\sigma_{(f_j)}$ represent the standard deviation of the i^{th} and j^{th} feature respectively. The value of correlation coefficient lies between -1 and 1. If there is no correlation between the features f_i and f_j then $\rho_{(f_i,f_j)} = 0$. A perfect negative correlation is found if $\rho_{(f_i,f_j)} = -1$ and a perfect positive correlation is found if $\rho_{(f_i,f_j)} = 1$. We plot the correlation between features for the first dataset as a heat map, which is shown in Figure 2.3.

As it can be seen, the source IP address is more correlated to TTL, destination port number, source MAC addresses and destination IP addresses. Furthermore, the destination IP address and source port number, the destination IP address and destination MAC address, the packet length and destination MAC address, the source MAC address and source port number, the source port number and destination port number are also highly correlated features.

Secondly, we find the correlation between the input vector features and the target class labels. Then based on the relationship between independent variables (i.e., feature space) and dependent variable (i.e., class label) we select the features for our learning (classification) framework. This is further discussed in Section 2.5.3.

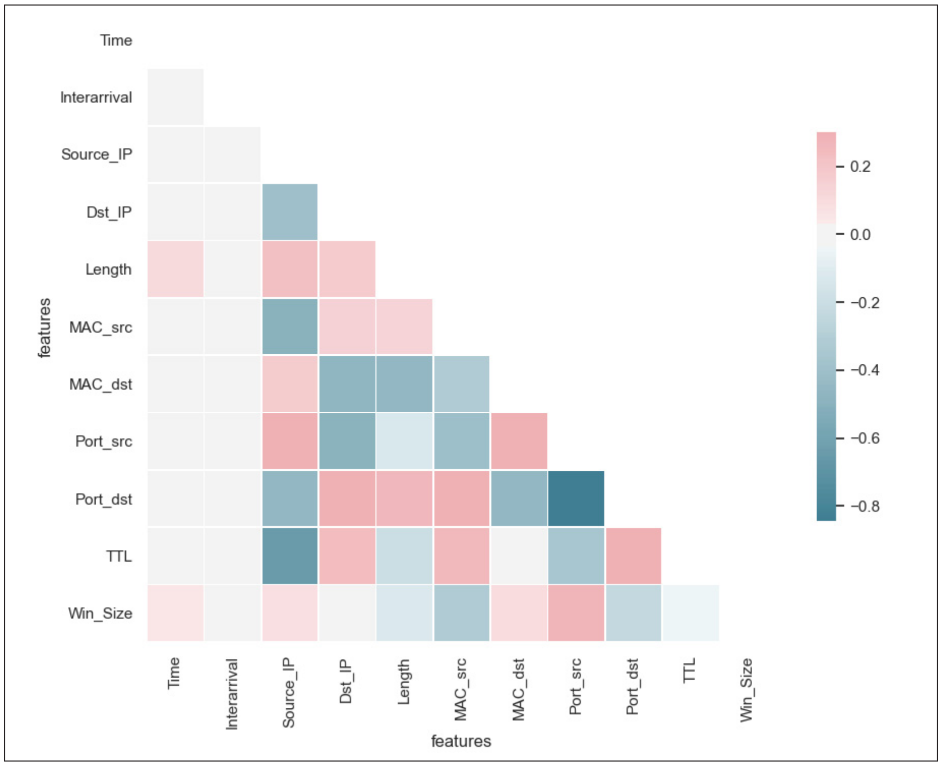


Figure 2.3 Correlation between IoT traffic features of Dataset 1

2.5 Proposed Classification Framework

2.5.1 Overview

The proposed classification framework consists of three key steps as shown in Figure 2.4 and discussed in the following sections.

1. **Preprocessing the IoT Traffic (Section IV-B):** It is the first step executed and it aims at providing the weighted preprocessing of dataset along with the rescaling, imputation and transformation of traffic traces.
2. **Selecting the most relevant features (Section IV-C):** It consists of the selection of the most important features, which are highly correlated to the class labels, using the ANOVA filter based selection method.
3. **Two-stage learning model (Section IV-D):** Here the classification of the IoT traffic traces is done using the proposed two stage learning model. At stage 0, the classification is performed

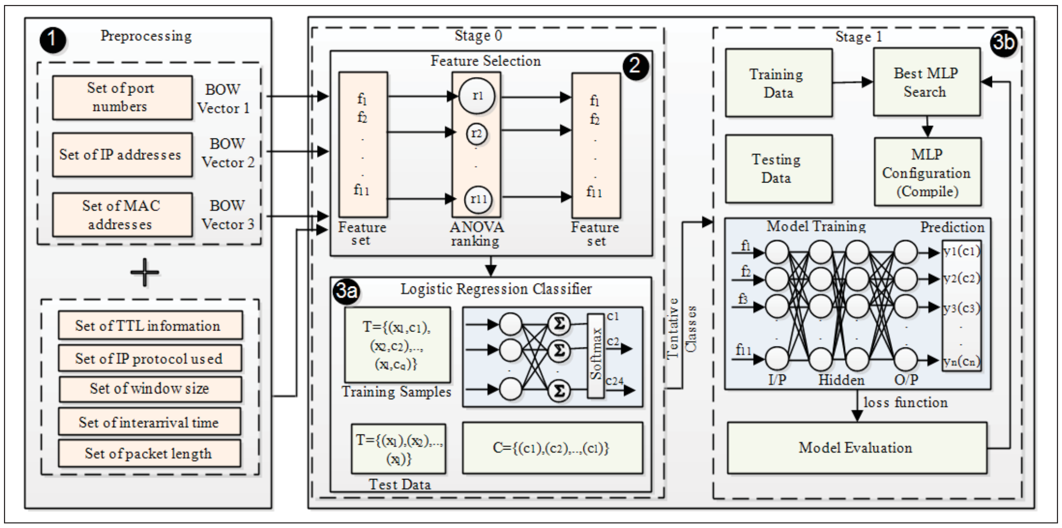


Figure 2.4 Overview of proposed two-stage classification framework

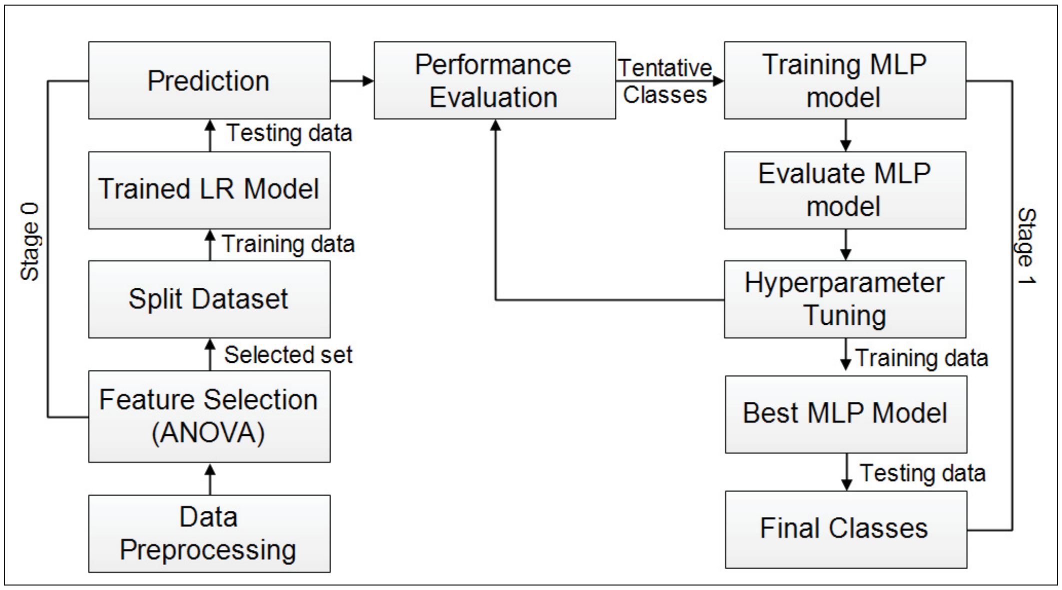


Figure 2.5 Operational flow of the proposed work

by applying a logistic regression technique, while the tentative classes are provided. At stage 1, a neural network is applied to provide the final classes.

The operational flow of the proposed work is provided in Figure 2.5.

2.5.2 Data Preprocessing

During the data preprocessing, a basic filtering of the dataset is performed in order to remove some of the non-meaningful packets such as ping, DNS requests, etc. The features such as TTL, window size, packet length are already numerical, whereas the interarrival time feature is converted to seconds. Following, we observed that some of the features such as "set of port numbers (f_5 and f_6)", "set of IP addresses (f_2 and f_3)" and "set of MAC addresses (f_{10} and f_{11})" are nominal and multi-valued (having more than one value with a single data instance). As machine learning classifiers cannot deal with such data, we converted these features into a numerical form using a two-step procedure.

Firstly, we perform the data cleaning by passing the nominal vectors to the Bag-of-Word (BoW) model. Secondly, as the BoW assigns the same importance to each vector word, we have proposed a relevance weighting to assign a prioritized importance to each word within each vector. These relevance weights, attributed to each feature vector, are passed to the stage 0 classifier and is given by Equation (2.2):

$$Relevance\,Weight = wf_{w,v} \times vf_{w,v} \tag{2.2}$$

where $wf_{w,v}$ denotes the word frequency of a word w within a vector v and $vf_{w,v}$ represents the total vector frequency. Herein, the vectors consist of the "port numbers vector", "IP addresses vector", and "MAC addresses vector". The word frequency $wf_{w,v}$ is defined as the number of times that w occurs in v and is given using Equation (2.3):

$$wf_{w,v} = \frac{number\,of\,occurrence\,of\,a\,word\,in\,a\,vector}{number\,of\,words\,in\,that\,vector} \tag{2.3}$$

Because frequent words are less informative than rare words, the vector frequency, $vf_{w,v}$ is given as Equation (2.4).

$$vf_{w,v} = \log \frac{number\,of\,vectors}{number\,of\,vectors\,containing\,word\,w} \tag{2.4}$$

After this step, we impute the missing values of features using their mean value and re-scale the dataset between 0 and 1 using the MinMaxScaler technique.

2.5.3 Feature selection

The supervised feature selection is a way to choose the input features that are believed to be the most useful to a model in order to predict the target variable. For our supervised feature selection method, we resort to either wrapper methods or filter based methods. A wrapper based method, such as Recursive Feature Elimination (RFE), selects the features that are performing well.

However, for the selection of features from our feature space F, we employed the filter-based feature selection technique (Brownlee, 2020) which uses the statistical methods to score the relationship between the features and the target labels i.e., class labels. Specifically, we have selected the ANOVA (Analysis of Variance) F-value feature selection technique because our input features are quantitative or become quantitative after preprocessing and the target class labels are of categorical nature (i.e. c_1 indicates a belkin wemo switch, c_2 represents smart cam and so on).

2.5.4 Proposed two-stage learning model

2.5.4.1 Stage 0 classifier

The Logistic Regression method is employed at stage 0, which takes the selected set of features for the training, as given by the ANOVA F-value. The reason that we have selected this classifier is that it has been proven to perform well for very large data sets (Backhaus, Erichson, Gensler, Weiber & Weiber, 2023), as in the case of a smart environment. The logistic regression technique investigates the association among the independent variables and the dependent variables of the problem. In our scenario, the selected features are the independent variables and the device categories (e.g. hubs, cameras, etc.) are the dependent variables. The goal is to estimate the probability p for a combination of independent variables using the following logit function:

$$logit(p) = ln\frac{p}{1-p} \tag{2.5}$$

where ln is the natural logarithm and p denotes the probability of an independent variable. The anti log of (2.5) allows us to find the estimated regression equation given by Equation (2.6):

$$logit(p) = ln\frac{p}{1-p} = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + ... + \beta_n * x_n \Rightarrow$$
$$p = \frac{e^{\beta_0+\beta_1*x_1+\beta_2*x_2+...+\beta_n*x_n}}{1 + e^{\beta_0+\beta_1*x_1+\beta_2*x_2+...+\beta_n*x_n}} \tag{2.6}$$

where β_0 is an intercept, β_1, β_2, and β_n are the regression coefficients, x_1 is the first independent variable, x_2 is the second independent variable, and x_n is the n^{th} selected feature. In order to calculate β coefficients, we employed the Gradient Descent method (Henry, 2021). The general form of Equation (2.6) is given as:

$$p(y_i|x_1, x_2, ..., x_n) = \frac{1}{1 + e^{-(\beta_0+\beta_1*x_1+\beta_2*x_2+...+\beta_n*x_n)}} \tag{2.7}$$

where y_i represents the dependent variable i.e., the i^{th} IoT device class, which we predict based on x_1, x_2, and x_n. After calculating the regression coefficients the testing component comes into effect, where the classifier uses the regression coefficients and computes the estimated regression for each testing instance using Equation (2.7). Finally, stage 0 classifier performs a first tentative prediction.

2.5.4.2 Stage 1 classifier

In order to optimally classify the IoT devices, we architect the Multi-Layer Perceptron Artificial Neural Network (MLP-ANN) (Okwu & Tartibu, 2021) based classification as our stage 1 classifier. MLP-ANNs are composed of multiple neurons that are arranged in the form of an input, output, and hidden layers. In this work, the architecture of MLP-ANN consists of one input layer with 11 neurons, because we have 11 different features to be passed as an input to the neural network. Following, we optimize the number of hidden layers, while the output layer consists of *n* number of neurons depending on the number of labelled classes *n* found in each of the dataset.

MLP-ANN provides two major processes for the classification task. Firstly, it performs the forward propagation process, which feeds the features to the input layer neurons. In our case, all quantitative features along with the output from stage 0 classifier (i.e., tentative classes) are fed to an input layer. Following, the input layer propagates these data to the hidden layers and then to the output layer. The neurons in each of the neural network layer calculates the weighted sum as output which is then passed to the activation function and is given by Equation (2.8).

$$O_i^{(l)} = V^{(l)}(\sum_j w_{(i,j)}^{(l)} \times O_j^{(l-1)} + B_i^{(l)}) \tag{2.8}$$

where the superscripts on variables represent the layer number and the subscripts represent the neuron numbers in the respective layer. The $w_{(i,j)}^{(l)}$ denotes the weight of a connection between the i^{th} neuron of layer l and the j^{th} neuron of layer $l-1$; $B_i^{(l)}$ represents the bias value applied at the l^{th} layer for the i^{th} neuron; $O_i^{(l)}$ denotes the output of the the i^{th} neuron at the l^{th} layer and V^l represents the nonlinear activation function applied at layer l. This work applied the Rectified Linear Units (ReLU) activation function at the input layer and the softmax activation function at the output layer.

The above process continues till the output layer predicts a label, i.e., class of an IoT device, which is then compared with the actual label and a loss value is calculated using a loss function based on the categorical cross entropy. Secondly, a back propagation is done in which weights are updated using the predicted output, desired output and their difference. The goal is to minimize the loss by finding the optimal weights value. The optimization function that we applied is based on the Adaptive Moment Estimation (Adam) because it is proved to be very robust for large datasets.

To model an optimal MLP-ANN, we used the Keras tuner (O' Malley *et al.*, 2019) along with the Random Search technique. For the hyper parameter optimization, we determine the optimal number of hidden layers, the optimal number of neurons in each layer (i.e., a search between 22 and 512 neurons), and the learning rate (i.e., a search between 1e-2 and 1e-4) using a random search tuner. Following, these parameters are passed to the Adam optimizer, since we want to achieve the best performance along with the least computational complexity.

2.6 Classification Algorithm

2.6.1 Algorithm Description

The preprocessing algorithm (Algorithm 2.1) consists of the $PREP$ procedure, which firstly generates the BoW representations using the function $generate\_BOW()$. Then, the relevant weights are calculated by employing the $word\_Freq()$ and $vector\_Freq()$ functions, which takes BoW as an input. Following, the features are scaled using the function $MinMaxScaler()$. Algorithm 2.2 depicts the learning model consisting of two procedures, namely, $LOGREG$ and MLP. In the $LOGREG$ procedure, the input labels x and output labels y are split into training and testing data using the function, $split()$. Next, the filter-based feature selection is done using the statistical method called ANOVA score and this is achieved by employing the $SelectKBest()$ function. Then the $LogisticRegression()$ generates and fit the model using the $fit()$ function. The prediction is done using the $predict()$ which contains the x_tst as testing dataset.

The MLP procedure generates the classification results based on the MLP-ANN which takes stage's 0 results along-with the other features. At this stage, firstly the data are split using $split()$ and then a sequential model is created using the function, $build\_model()$. Following, the keras tuner is applied to search the number of models using $RandomSearch()$, which takes the sequential model, the number of trials per search, the max trials allowed and the search objective as an input. Then, the $getBestModel()$ returns the model with the highest validation accuracy across all models given by the $RandomSearch()$. Finally, we fit the model with $fit()$ for 70 epochs and then call the $predict()$ function.

Algorithm 2.1 Preprocessing Algorithm

```
**Input:** $f_2$,$f_3$,$f_5$,$f_6$,$f_{10}$,$f_{11}$, $devices$
// $f_2$ and $f_3$ are source and destination IP addresses; $f_5$ and $f_6$ are source and destination
port numbers; $f_{10}$ and $f_{11}$ are source and destination MAC addresses; and $devices$
labels.
**Output:** $\mathbf{x}_{norm}$,$y$

1  PREP($f_2$,$f_3$,$f_5$,$f_6$,$f_{10}$,$f_{11}$, $devices$)
2  $BOW_1 \leftarrow generate\_BOW(f_2, f_3)$
3  $BOW_2 \leftarrow generate\_BOW(f_5, f_6)$
4  $BOW_3 \leftarrow generate\_BOW(f_{10}, f_{11})$
5  $wf \leftarrow word\_Freq(BOW_1, BOW_2, BOW_3)$
6  $vf \leftarrow vector\_Freq(BOW_1, BOW_2, BOW_3)$
7  $rel_{weight} \leftarrow wf \times vf$
8  **set** $x \leftarrow dataset(BOW_1, BOW_2, BOW_3, rel_{weight})$
9  **set** $y \leftarrow dataset(devices)$
10 **set** $x_{norm} \leftarrow MinMaxScaler(x)$
```

2.6.2 Asymptotic Analysis

Proposition 1. The computational complexity of $PREP$ procedure is $O(n)$

Proof: The PREP procedure running time depends on the number of feature vectors, represented as n. Lines 2-4 take a constant time as they split the vectors into words, thus $O(1)$. Lines 5-6 and 8-9 are assignment statements and each requires $O(1)$ operations. For the rel_{weight} statement (line 7) the complexity is $O(1) * O(n) = O(n)$. However, line 10 depends on the number of feature vectors n and thus, in the worst-case scenario needs $O(n)$. Accordingly, the overall time complexity of PREP procedure is linear i.e., $O(1) + O(1) + O(n) + O(n) = O(n)$.

Proposition 2. The computational complexity of $LOGREG$ procedure is $O(n)$.

Proof: Line 2 is a simple assignment statement (i.e., $O(1)$) and lines 3-4 require $O(n)$ computation time in the worst scenario. Regarding the training time (lines 5-6) of LOGREG the complexity is $O(t * n)$ where t is the number of training examples and n is the number of selected data features used for the classifier training. Additionally, the testing time taken by line

Algorithm 2.2 Learning Algorithm

```
Input: $x_{norm}$, $y$, $y_{pred}$, $f_1$, $f_4$, $f_7$, $f_8$, $f_9$, $devices$
// $x_{norm}$ is the dataset instances, $y$ is the class labels, $y_{pred}$ is the output of Stage 0
classifier passed to the Stage 1 classifier, $f_1$ is the interarrival time, $f_4$ is the IP protocol
used, $f_7$ is the TTL, $f_8$ and $f_9$ are the window size and packet length, $devices$ are the
class labels.
Output: $y_{pred}$
// $y_{pred}$ is the final output of Stage 1 classifier.
1  LOGREG($x_{norm}$,$y$)
2  set $x_{tr}, x_{tst}, y_{tr}, y_{tst} \leftarrow split(x, y, test_{size} \leftarrow 0.2)$
3  set $x_{tr} \leftarrow selectKBest(Anova_{score}, x_{tr})$
4  set $x_{tst} \leftarrow selectKBest(Anova_{score}, x_{tst})$
5  set $model \leftarrow LogisticRegression(max_{iter} \leftarrow 3000)$
6  set $fit \leftarrow model.fit(x_{tr}, y_{tr})$
7  set $y_{pred} \leftarrow model.predict(x_{tst})$
8  Output: $y_{pred}$                                        ▷ Stage 0
9  MLP($y_{pred}$,$f_1$,$f_4$,$f_7$,$f_8$,$f_9$, $devices$)
10 set $x \leftarrow dataset(y_{pred}, f_1, f_4, f_7, f_8, f_9)$
11 set $y \leftarrow dataset(devices)$
12 set $x_{tr}, x_{tst}, y_{tr}, y_{tst} \leftarrow split(x, y, test_{size} \leftarrow 0.2)$
13 set $m \leftarrow build\_model()$
14 set $tuner \leftarrow RandomSearch(m, tuner.obj(val_{acc}), max_{tr} \leftarrow 3, search_{tr} \leftarrow 1)$
15 set $model \leftarrow tuner.getBestModel(num_{models} \leftarrow 1)$
16 set $history \leftarrow model.fit(x_{tr}, y_{tr}, epochs \leftarrow 70)$
17 set $y_{pred} \leftarrow model.predict(x_{tst})$
18 Output: $y_{pred}$ : $FS \leftarrow devices$                 ▷ Stage 1
```

7 is $O(n)$. Thus, the LOGREG takes $O(1) + O(n) + O(t * n) + O(n) = O(n)$, which can be beneficial for low latency applications that require a fast classification method.

Proposition 3. The computational complexity of MLP procedure is $O(nd)$

Proof: In the MLP procedure, lines 10-12 consist of simple assignments i.e., $O(1)$. Line 13 indicates the $build\_model()$ function of the neural network and its complexity is $O(n * d * t * e)$, where for proposition 3, n represents the number of layers, d denotes the number of neurons in each layer, t is the number of training examples and e is the number of epochs. Because we are using 80% training examples i.e., 664796 for 70 epochs, the complexity for this part is $O(n*d*664796*70) = O(nd)$. Following, $RandomSearch()$ (line 14) takes $O(n)$ for the worst scenario and line 15 takes a constant amount of time i.e., $O(1)$. Line 16 takes $O(t)$ and testing time taken by the line 17 is $O(n)$. Thus, the MLP takes $O(1)+O(nd)+O(n)+O(1)+O(t)+O(n) = O(nd)$ time.

The overall complexity, T of the proposed learning framework is represented in term of n as: $T(n) = O(n) + O(n) + O(nd) = O(n)$. Thus, it is a linear time learning work.

2.7 Performance Evaluation

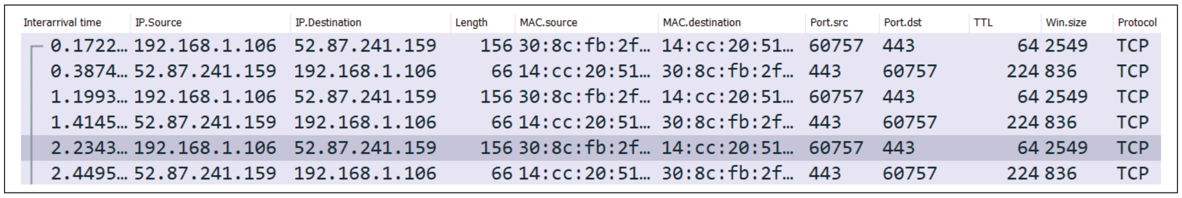
2.7.1 Model Implementation and Frameworks

2.7.1.1 Dataset Description

In this work, we have used two different datasets provided by (Sivanathan *et al.*, 2018) and (Ren *et al.*, 2019) consisting of IoT traffic traces in a smart environment. The description of both datasets is provided as follows:

Dataset 1 (Sivanathan *et al.*, 2018) consists of network traffic traces from 28 smart devices. As we have considered a subset of the network traffic, which is a total of 12000317 labeled instances of 22 IoT devices, for this dataset we have 22 distinctive classes. The devices are namely, smart phone, belkin wemo switch, belkin wemo motion sensor, dropcam, HP printer, iphone, laptop, nest protect smoke alarm, netatmo welcome, netatmo weather station, PIX star photo frame, samsung tab, samsung smartcam, smart things, TP link camera, TP link plug, TP link router, triby speaker, withings smart baby monitor, withings smart scale, ipv4mcast and amazon echo.

Dataset 2 (Ren *et al.*, 2019) consists of traffic traces of from 81 IoT devices which are located at various US and UK locations. These devices belongs to cameras, smart hubs, home automation, TVs, audio devices and home appliances categories. For the second dataset, a total of 40588450 labeled instances of 68 IoT devices were used in this work.



Interarrival time	IP.Source	IP.Destination	Length	MAC.source	MAC.destination	Port.src	Port.dst	TTL	Win.size	Protocol
0.1722…	192.168.1.106	52.87.241.159	156	30:8c:fb:2f…	14:cc:20:51…	60757	443	64	2549	TCP
0.3874…	52.87.241.159	192.168.1.106	66	14:cc:20:51…	30:8c:fb:2f…	443	60757	224	836	TCP
1.1993…	192.168.1.106	52.87.241.159	156	30:8c:fb:2f…	14:cc:20:51…	60757	443	64	2549	TCP
1.4145…	52.87.241.159	192.168.1.106	66	14:cc:20:51…	30:8c:fb:2f…	443	60757	224	836	TCP
2.2343…	192.168.1.106	52.87.241.159	156	30:8c:fb:2f…	14:cc:20:51…	60757	443	64	2549	TCP
2.4495…	52.87.241.159	192.168.1.106	66	14:cc:20:51…	30:8c:fb:2f…	443	60757	224	836	TCP

Figure 2.6 Samples of IoT traffic traces from dataset 1

A sample of the network trace used from the first dataset is provided in Figure 2.6. Nonetheless, since we have used the same feature space for both datasets, Figure 2.6 reflects the traces from the second dataset as well. The feature called "MAC address" of each device is used to provide the label to each network trace in both of the datasets.

2.7.1.2 Experiment setup

The configuration settings used for our experiments and for both datasets are listed in Table 2.6. The proposed model was implemented in Python (version 3.8.2). In Table 2.6, the No. of architectures represents the number of different classification solutions used during our experimentation. These architectures/solutions are further explained in Section 2.7.1.3. Following, the total number of instances provides the number of labelled instances used from each dataset and the total number of classes represents the total number of distinct device types. The reason that we have selected a subset of the labelled instances for each dataset, is because these datasets span over a period of about two months and the training of such a large amount of data can create several big data challenges. Furthermore, as shown later, we also managed to achieve a very good performance by using only the specific subset of these datasets. Accordingly, the selected subset of data under evaluation resulted in a slightly reduced number of classes for each dataset.

Regarding the number of tuner trials, this value represents the keras tuner trials that we executed for our proposed model. In more details, for the first dataset, we noticed that after 5 trials we have achieved the best hyper-parameter configuration and for the second dataset after 3 trials. The reason for executing several trials, is that the keras tuner uses a different set of parameters (i.e. learning rate, number of layers and number of neurons in each layer) at each trial and then it selects the best performing configuration. Nonetheless, we have not seen a significant variation between the accuracy of the different trials. Lastly, we split both of the dataset instances into three groups as: 60% training instances, 20% validation and 20% testing instances, which is a common split ratio in the machine learning domain.

For the evaluation of the classification performance , we have considered the following well known classification metrics:

Table 2.6 Configurations used in the experiments

N/W Settings	No. of Architectures	Total Instances	Total Classes	No. of Tuner Trials	Data Split	Metrics
Dataset 1	5	12000317	24	5	60:20:20	Precision/Recall/F1/Accuracy
Dataset 2	5	40588450	68	3	60:20:20	Precision/Recall/F1/Accuracy

Table 2.7 Description of model architectures applied to the multi-classification problem

Architecture	Details
I: LR+GB	Stage 0: LR - Stage 1: GB
II: NB+RF	Stage 0: NB - Stage 1: RF
III: MLP	Single Stage : MLP with IP(11)-FC (100)-FC(100)-OP(22)
IV: LR(RFE)+MLP	Stage 0: LR along with RFE - Stage 1: MLP with IP(11)-FC(100)-FC(100)-OP(22)
V: LR(Anova)+MLP(Keras Tuner)	Stage 0: LR along with Anova-score feature selection - Stage 1: MLP with IP(11)-FC (n)-FC (n)-OP(22)

1. Precision: It is the ability of a classifier to not label an instance that is actually negative as positive and is given as:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{2.9}$$

2. Recall: Recall calculates the rate of all the positive instances, which is also called true positive rate and is given as:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative} \tag{2.10}$$

3. F1-score: It is the harmonic mean of the precision and recall metrics and is given as:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.11}$$

4. Accuracy: It is the proportion of correctly classified instances and is given as:

$$Accuracy = \frac{CorrectPredictions}{TotalPredictions} \tag{2.12}$$

5. Confusion matrix: It is a table that is used to describe the classifier performance on a set of test data for which the true values are known.

The values of recall, precision, F1-score, confusion matrix and accuracy are calculated between [0,1] with 1 indicating the best and 0 the worst performance. However, a decrease from 1 towards 0 is good for the loss function of the network.

2.7.1.3 Architecture models

We have applied different composite models consisting of neural networks along with traditional machine learning algorithms to see their suitability for the IoT traffic multi classification problem. Table 2.7 provides the description of the different network architectures. The LR represents the logistic regression algorithm and GB denotes the gradient boosting algorithm (architecture I) (Hameed, Violos, Santi, Leivadeas & Mitton, 2021). The NB is Naive Bayes algorithm at stage 0 and RF denotes applying random forest at stage 1 (architecture II) (Sivanathan *et al.*, 2019). IP(x) stands for the input layer of neural network with x number of neurons. FC(x) denotes the fully connected layer of neural network with x number of nodes (or neurons). OP(x) represents the output layer of neural network with x number of classes i.e., neurons.

MLP represents the multi layer Perceptron neural network with an input layer consisting of 11 neurons, two fully connected layer and one output layer with 22 classes (architecture III). LR(RFE)+MLP denotes the logistic regression at stage 0 with recursive feature elimination method and MLP at stage 1 with one input layer, two fully connected layers and one output layer (architecture IV). LR(Anova)+MLP (keras tuner) denotes the logistic regression at stage 0 with the Anova based feature selection and MLP at stage 1 (architecture V), which is the two-stage learning model proposed in this paper.

For comparison purposes, it is important to mention that the accuracy of existing works are less than the proposed framework, as shown in the following subsection. For example, the proposed framework in (Kotak & Elovici, 2021) achieves an accuracy of 99.0%, the authors in (Lopez-Martin & et al, 2017) achieve 96% accuracy, while in (Meidan *et al.*, 2017a) the accuracy is 99.2%. However, for our evaluation, we compared the proposed framework with the architecture I (Hameed *et al.*, 2021) and architecture II (Sivanathan *et al.*, 2019), which both use the first dataset.

Additionally, to better illustrate the efficiency of our work, we also compare our proposed architecture V with the architectures III and IV which are based on the MLP neural network. For all the neural network-based architectures (i.e. III to V), the training was done with a number of epochs between 50 and 100. The training was stopped earlier if an increase in the number of epochs did not lead into an improvement of the loss function.

Furthermore, for the activation functions we used the ReLU along with the softmax activation which was applied at the last output layer. The loss functions used was the categorical cross entropy. Finally, the optimization was done with the Adaptive Gradient (AdaGrad) for the architectures III and IV and with Adam for architecture V. The particular configurations gave the best results for each of the examined architectures.

We have also experimented with different LSTM configurations. In particular, we executed five tuner trials to find the best hyperparameters such as number of layers, LSTM units, learning rate, etc. However, these models gave less accurate results, (i.e., 70% of accuracy). Moreover, we also considered the AdaGrad optimizer for the architecture V but it produced an accuracy of 85% and we decided to show only the results of the best configuration, which uses the Adam optimizer.

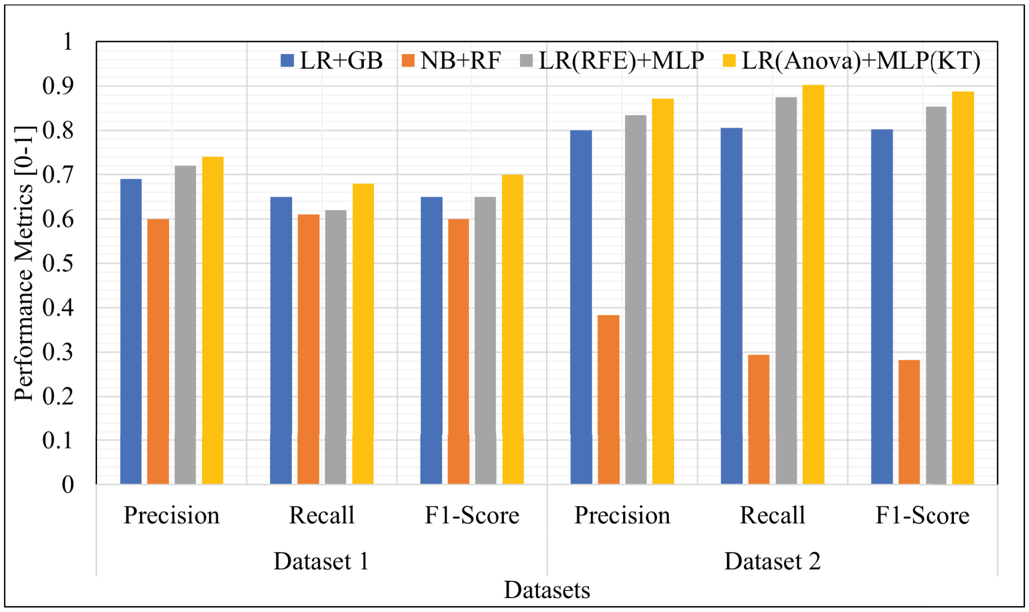


Figure 2.7 Performance comparison at stage 0

2.7.2 Results

2.7.2.1 Impact of architectures

2.7.2.1.1 Stage 0

Figure 2.7 illustrates the performance of the different network architectures at stage 0, in terms of precision, recall and F1 score for both datasets. We have only considered the architectures I, II, IV, and V for this part, because architecture III i.e., MLP does not consist of two stages. In terms of the precision, our proposed architecture V provides the highest value i.e., 0.74 followed by LR(RFE) + MLP with 0.72 and LR+GB with 0.69 value for the first dataset. Regarding the second dataset, the same trend is noticed, as architecture V provides the highest value i.e., 0.87 followed by LR(RFE) + MLP with 0.83 and LR+GB with a value of 0.79.

In contrast, NB + RF performed poorly for both datasets, i.e., 0.6 for the first dataset and 0.4 for the second. This means that 40% of the labelled instances were wrongly classified as positive for the first dataset and 60% were wrongly classified as positive for the second. This can be attributed to the fact that the precision values of some devices were zero and less than 0.17 for many other. As an example, in the first dataset the most misclassified devices for the NB+RF were the Belkin Switch, HP printer, Netatmo Welcome, PIX-STAR, Samsung tab and TP link camera.

When looking into the recall metric, we see that the proposed architecture V also outperformed the rest of the models, followed by the LR+GB and LR(RFE)+MLP for the first dataset. However, for the second dataset, LR(RFE)+MLP(KT) is followed by LR(RFE)+MLP and LR+GB, while architecture V remains the most efficient solution. Once again NB+RF gives the least average recall for both datasets, with 0.61 and 0.29 for dataset 1 and 2. The reason for this behavior is that the majority of instances were 100% misclassified. For instance, for the first dataset, out of 22 classes, instances of 8 classes were 100% incorrectly classified.

Lastly, we observe that the architecture V gives the highest value of F1 score among all architectures at stage 0, with a value of 0.7 for the first dataset, followed by LR+GB and LR(RFE)+MLP which both give an F1-score of around 0.65, whereas NB+RF achieves only 0.6. For the second dataset, our proposed architecture presents a F1-score of 0.89 followed by LR(RFE)+MLP, LR+GB, and NB+RF which give a F1-score of 0.85, 0.80, and 0.28 respectively.

2.7.2.1.2 Stage 1

At this stage all five network architectures are considered as shown in Figure 2.8 for both datasets. Moreover, we also included the accuracy in our evaluation metrics, since the output of Stage 1 is our final classification. As it can be seen, our proposed architecture (LR(Anova)+MLP(KT)) attained an accuracy of 0.999, a precision of 0.996, a recall of 0.995 and a F1-score of 0.996 for the first dataset. Regarding, the second dataset, it achieved an accuracy of 0.998, a precision of 0.996, a recall of 0.997 and a F1-score of 0.997. Furthermore, LR(RFE)+MLP(KT) provided reasonable results followed by the other architectures for both of the datasets.

Once again, NB+RF continued to under-perform for both datasets at stage 1. Specifically, for the dataset 1, the NB+RF achieved a performance of only 0.78 for recall, 0.8 for precision and 0.77

for F1-score because 3335 training instances of Belkin switch class, 374 instances of HP printer class, 262 instances of the TP link camera class and 31 iPhone class instances were incorrectly classified. Similarly, for the dataset 2, the particular model achieved a performance of only 0.33 for recall, 0.29 for precision and 0.31 for F1-score because many instances of devices such as Tphilips Hub US, TP link bulb US, Sousvide US, TP link plug UK, T wemo plug UK, T wemo plug US, Wans view cam wired US, wans view cam wired UK, smart thing hub UK,sousvide UK,T philips hub UK,TP link bulb UK,TP link plug US were incorrectly classified.

Additionally, the NB+RF provided an accuracy of 0.77 for dataset 1 and 0.92 for dataset 2. Further analysis showed that for the first dataset, there were 5 classes incorrectly classified out of 22 and for the second dataset, there were 13 misclassified classes out of 68. As accuracy is the ratio of these numbers, we corroborate the poor performance of architecture II as shown in Figure 2.8.

After analyzing the results of stage 1, we conclude that our architecture V and its variation (architecture VI) provide the best classification results in terms of all performance metrics for both of the datasets. This is a significant observation that proves the robustness of our framework that works equally well for different datasets with different number of classes. That is not the case for architectures I-III, which presented a great deviation in the attained results between the two datasets.

2.7.2.2 Impact of features

Figure 2.3 illustrated the correlation of the full set of features for the first dataset. However, it is critical to understand which features have a higher importance (rank value) provided by the feature selection method in the classification process. For this purpose, we provide the full set of features along with their ranks, as calculated by Anova score and RFE for dataset 1, in Figure 2.9. The most important features selected for both datasets are provided in Table 2.8.

For the architectures I, II, III, we have used all features during the training and testing phases, thus, we only compare the architectures IV and V to see the feature importance. Specifically, we

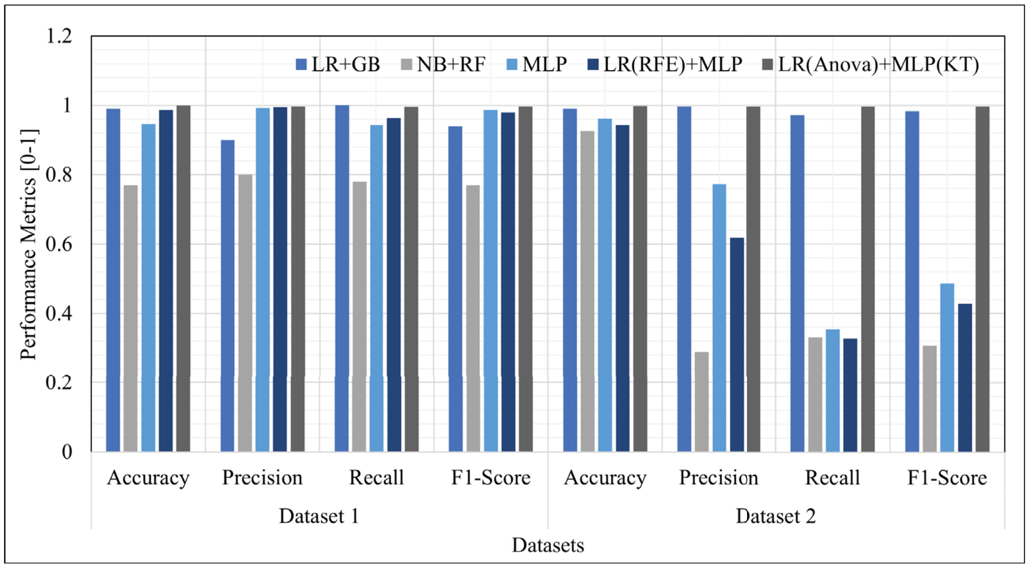


Figure 2.8 Performance comparison at stage 1

illustrate the ranks provided by the RFE for architecture IV and the ranks provided by the Anova score for architecture V. The rank values are between 0 and 1. It can be seen that the highest rank provided by Anova was 0.8 given to the feature 2 i.e., source IP address and the least rank given by Anova score was 0.14 for feature 4 i.e., IP protocol used by device. For the RFE method, the highest rank was provided to feature 2 i.e., 0.7 and the least to the feature 7 i.e., TTL information. The features were selected in decreasing order of their ranks by the architectures.

In more details, Table 2.7 provides the information about the features utilized by each architecture along with the performance metrics of each architecture for both datasets. The first three architectures used all 11 features. However, as mentioned earlier, architecture IV selected the features by RFE and architecture V selected the features by ANOVA method. For the first dataset, the selected features by RFE for architecture IV consists of the source IP address (f_2), interarrival time (f_1), source port number (f_5), destination Ethernet address (f_{11}), window size (f_8), destination port number (f_6), source Ethernet address (f_{10}) and IP protocol used (f_4). In contrast, the selected features by Anova for architecture V consists of source IP address (f_2), packet length (f_9), window size (f_8), source Ethernet address (f_{10}), destination port number (f_6), TTL (f_7), destination IP address (f_3), and source port number (f_5).

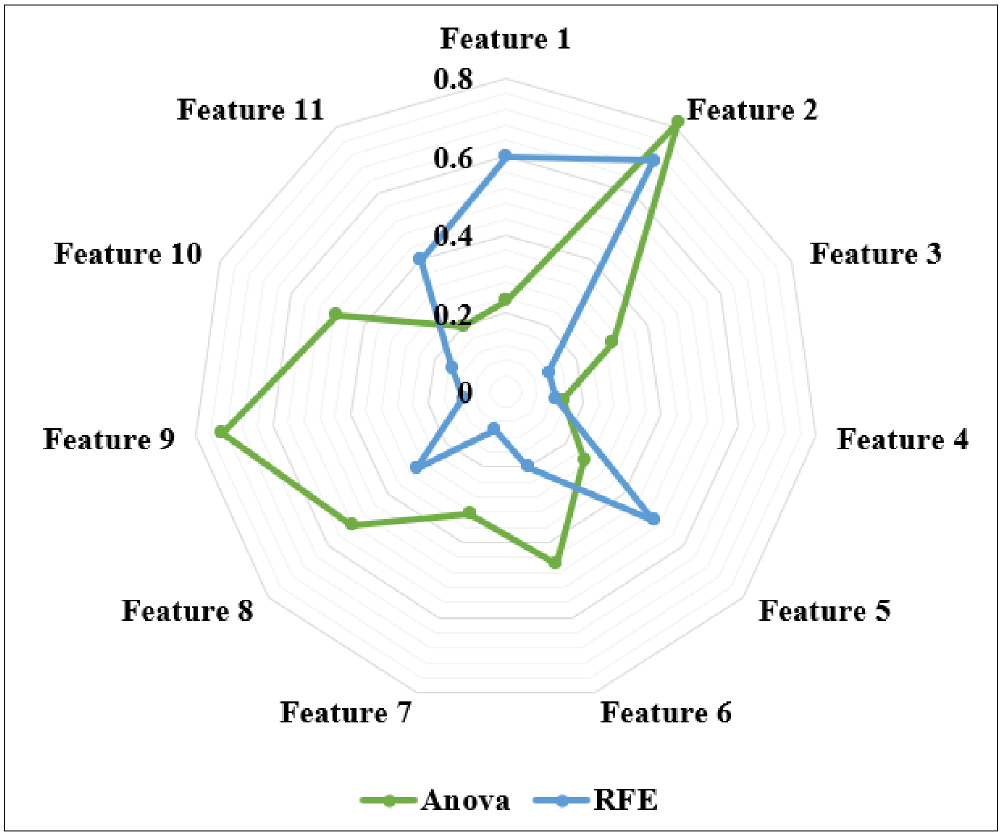


Figure 2.9 Feature ranks provided by the feature selection methods

Table 2.8 Classification performance metrics vs features employed

Datasets	Architecture	Features	Precision	Recall	F1	Accuracy
Dataset 1	LR+GB	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.900	1.000	0.940	0.990
	NB+RF	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.800	0.780	0.770	0.770
	MLP	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.992	0.943	0.986	0.946
	LR(RFE)+MLP	$f_2, f_1, f_5, f_{11}, f_8, f_6, f_{10}, f_4$	0.994	0.964	0.979	0.986
	LR(Anova)+MLP(Tuner)	$f_2, f_9, f_8, f_{10}, f_6, f_7, f_3, f_5$	0.996	0.995	0.996	0.999
Dataset 2	LR+GB	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.997	0.972	0.984	0.990
	NB+RF	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.289	0.331	0.307	0.926
	MLP	$f_1, f_2, f_3, f_4, f_5..., f_{11}$	0.774	0.354	0.486	0.961
	LR(RFE)+MLP	$f_5, f_6, f_8, f_{10}, f_{11}$	0.619	0.328	0.429	0.943
	LR(Anova)+MLP(Tuner)	f_4, f_5, f_6, f_7, f_8	0.997	0.997	0.997	0.999

For the second dataset, the selected features by RFE in architecture IV are the source port number (f_5), destination port number (f_6), window size (f_8), MAC address of source (f_{10}) and

MAC address of destination (f_{11}). For the architecture V, the selected features are the type of protocol (f_4), port number of source (f_5), port number of destination (f_6), TTL (f_7) and window size (f_8). Therefore, source IP address, packet length, window size, source Ethernet address, destination port number, TTL, destination IP address, and source port number are more relevant to classify labels for dataset 1 and the features as protocol, port number of source, port number of destination,TTL and window size are more important for the classification in the second dataset.

To better illustrate the impact of feature selection in the resulted accuracy, we provide the following formal logic representation for the first dataset. Nonetheless, the same logic can be easily applied for the second dataset as well.

In more detail, we are representing the actual and selected feature sets of dataset 1 as: $R = \{f_2, f_1, f_5, f_{11}, f_8, f_6, f_{10}, f_4\}$ and $A = \{f_2, f_9, f_8, f_{10}, f_6, f_7, f_3, f_5\}$ respectively. According to these sets, we model $R \cap A$ as follows:

$$R \cap A = \{x | x \in R : x \in A\} \Leftrightarrow \{f_2, f_5, f_8, f_6, f_{10}\} \tag{2.13}$$

The intersection $R \cap A$ gives the features that were used by both architectures. However, in order to evaluate the impact of the feature selection in the overall performance, we need to identify the features that were not included in both architectures, which is captured as follows:

$$R - A = \{x | x \in R \wedge x \notin A\} \Leftrightarrow \{f_1, f_{11}, f_4\} \tag{2.14}$$

Equation (2.14) provides the features that are only included by RFE and these are the interarrival time, the destination MAC address and the IP protocol used. Since, architecture IV presented an inferior performance than architecture V, we can safely say that these three features did not provide a well aligned information with the features given by $R \cap A$. Following, we extract the features included by the Anova score method but not from the RFE:

$$A - R = \{x | x \in A \land x \notin R\} \Leftrightarrow \{f_9, f_7, f_3\} \tag{2.15}$$

As (2.15) suggests, the packet length, TTL and destination IP address are the features that they are only considered by Anova and thus, by architecture V. Interestingly, we see that when these features are included in $R \cap A$ such that $(R \cap A) \cup (A - R) = A$, the performance increased significantly. Thus, the features $\{f_9, f_7, f_3\}$ have a positive impact in the performance of architecture V as they increased the accuracy to 99.9%, precision to 99.6%, recall to 99.5% and f1- score to 99.6% for dataset 1.

2.7.2.3 Performance of Architecture V

In this part of the evaluation, we present the detailed results of the proposed architecture V for the first dataset, however, the accuracy, precision, recall and F1 score for both datasets can be found in Table 2.8, as shown earlier.

2.7.2.3.1 Performance of stage 0

As we have proved the superior performance of our proposed two-stage classifier (architecture V), in this part of the section we delve into the details of the performance of the particular framework.

Accordingly, for the first dataset, Figure 2.10 illustrates the performance metrics per device for stage 0. Some devices such as Belkin sensor, Dropcam and TP link router presents the highest performance, i.e., recall=1; precision=1 and F1-score=1, all aggregated to 3. The lowest precision is noticed for the belkin wemo switch i.e., 0.61, while the lowest recall and F1-score are observed for the Samsung smartcam i.e., 0.53 and 0.65 respectively. Furthermore, for the SmartCam the aggregated value is 2.04 since the F1 score is 0.65, the recall is 0.53, whereas the precision is significantly high, i.e., 0.86. For the Netatmo weather station device, the aggregated value is 2.09 as the precision is reasonably good, i.e., 0.88 but the recall and F1 score are

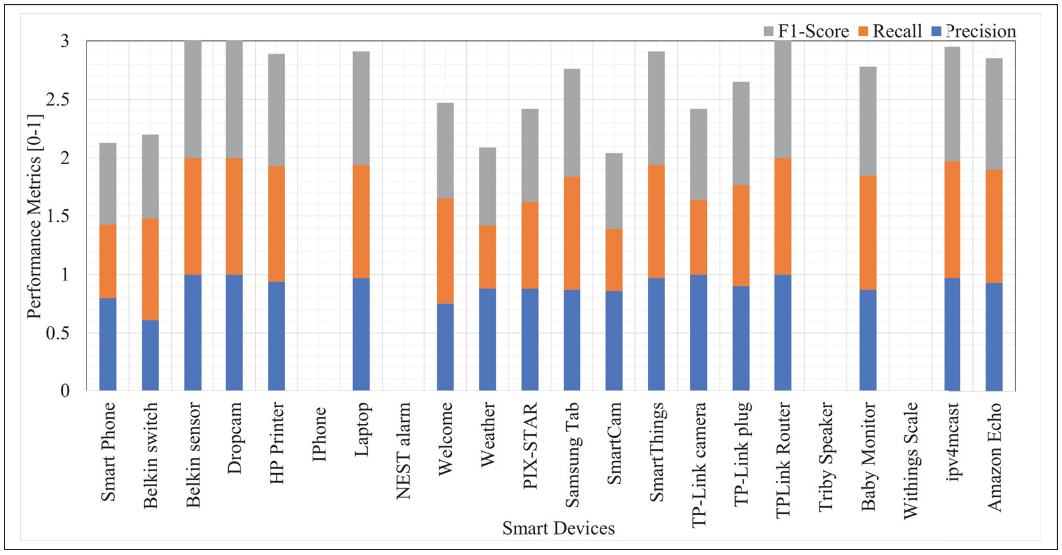


Figure 2.10 Performance comparison per device for architecture V

relatively low i.e., 0.54 and 0.67. However, there were some devices such as withings scale, triby speaker, nest alarm, and iPhone for which precision, recall and F1-score were zero. The reason is that the instances of such devices were misclassified in other categories.

Following, we plot the confusion matrix of dataset 1 to give the overall performance of stage 0 as shown in Figure 2.11. The row entries of a confusion matrix depict the actual values and the column entries depicts the predicted values for the 22 classes. All the diagonal entries correspond to correct classification whereas entries above diagonal are all Type I error (also called False Positive Rate (FPR)) and entries below are Type II error (also called False Negative Rate (FNR)). The goal is to minimize the Type I and Type II errors close or equal to zero.

At the main diagonal there are four exception cases: (i) the worst classification is noticed for the iPhone device, since 58% instances of the particular device were classified as Samsung galaxy tab, 22% instances were misclassified as TP link router, and 20% were misclassified as amazon echo thus depicting 100% FPR; (ii) for the nest protect smoke alarm the classification value is 0% with 100% FPR because it was misclassified as Samsung tab; (iii) for the triby speaker, we notice a 28% of misclassification as laptop (Type II error), and 72% of misclassification as netatmo

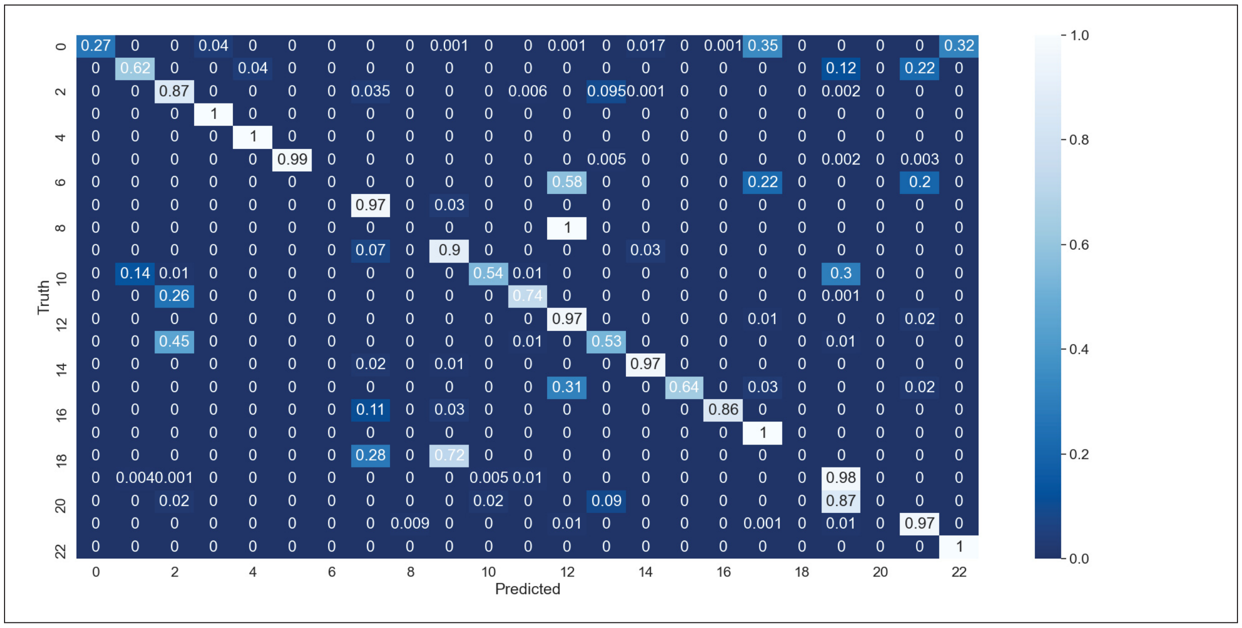


Figure 2.11 Confusion matrix for stage 0 of architecture V of dataset 1

welcome (Type II error); (iv) for the withings smart scale, we noticed 87% of misclassification as baby monitor (Type II error), 9.6% of misclassification as Samsung smartcam (Type II error), 1.9% of misclassification as Netatmo welcome, and 1.9% instances were incorrectly classified as belkin wemo switch.

This behavior is attributed to the following reasons: (a) there were 50 instances of iPhone compared to 3242, 87580 and 6231 of galaxy tab, TP link router and amazon echo instances; (b) 41 nest protect smoke alarm instances compared to 3242 instances of Samsung galaxy tab; (c) 771 triby speaker instances compared to 21815 laptop instances and 3995 instances of netatmo welcome; and (d) 52 withings smart scale instances compared to 5912, 4895, 3995 and 4407 instances of baby monitor, Samsung smartcam, Netatmo welcome and belkin wemo switch respectively. Thus, the prediction value for these devices is much higher as compared to iPhone, nest protect smoke alarm, triby speaker and withings scale.

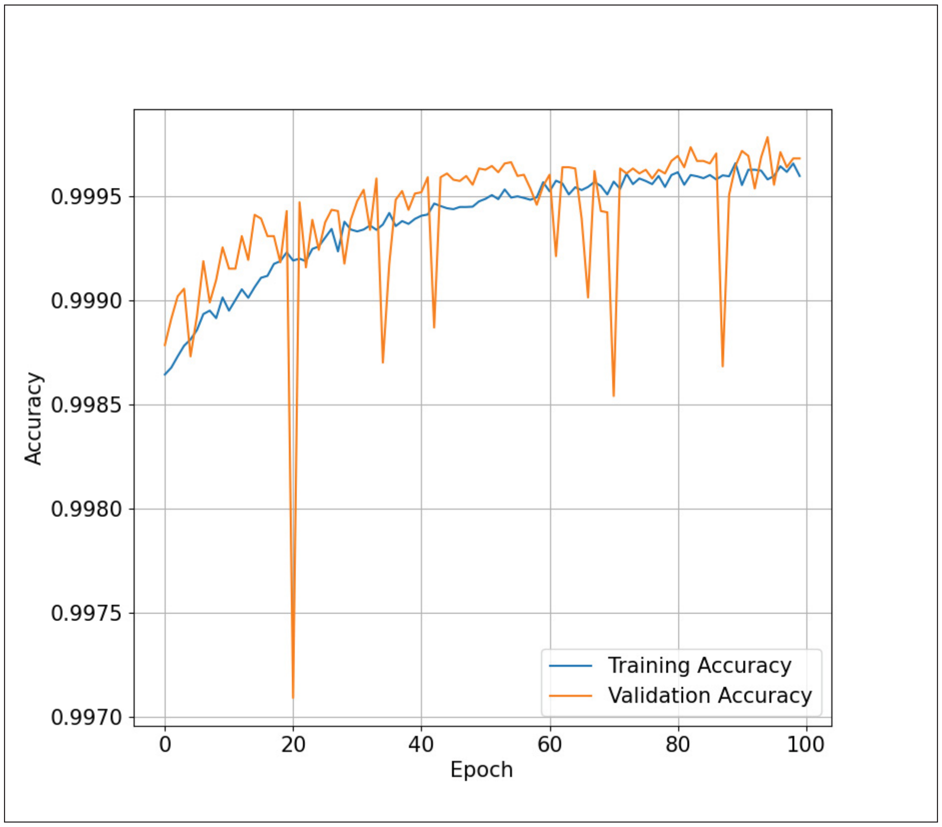


Figure 2.12 Training vs. validation accuracy of architecture V for 100 epochs

2.7.2.3.2 Performance of stage 1

Figure 2.12 depicts the training and testing accuracy, over the 100 epochs for the first dataset. The network model, i.e., optimized MLP at stage 1 of LR(Anova)+MLP (keras tuner), achieves better training accuracy i.e., 0.9997292 and validation accuracy i.e., 0.99962693 as the number of epochs increases. The initial accuracy values start from 0.998 at epoch 1 and the accuracy value does not change significantly after epoch 60. Regarding the spikes noticed, Keras Tuner estimates a close to optimal neural network topology using an exploitation versus exploration approach.

In the exploitation stage, it tries to improve the neural network topology, which output the most accurate results. In the exploration stage, it tries to randomly examine new neural network

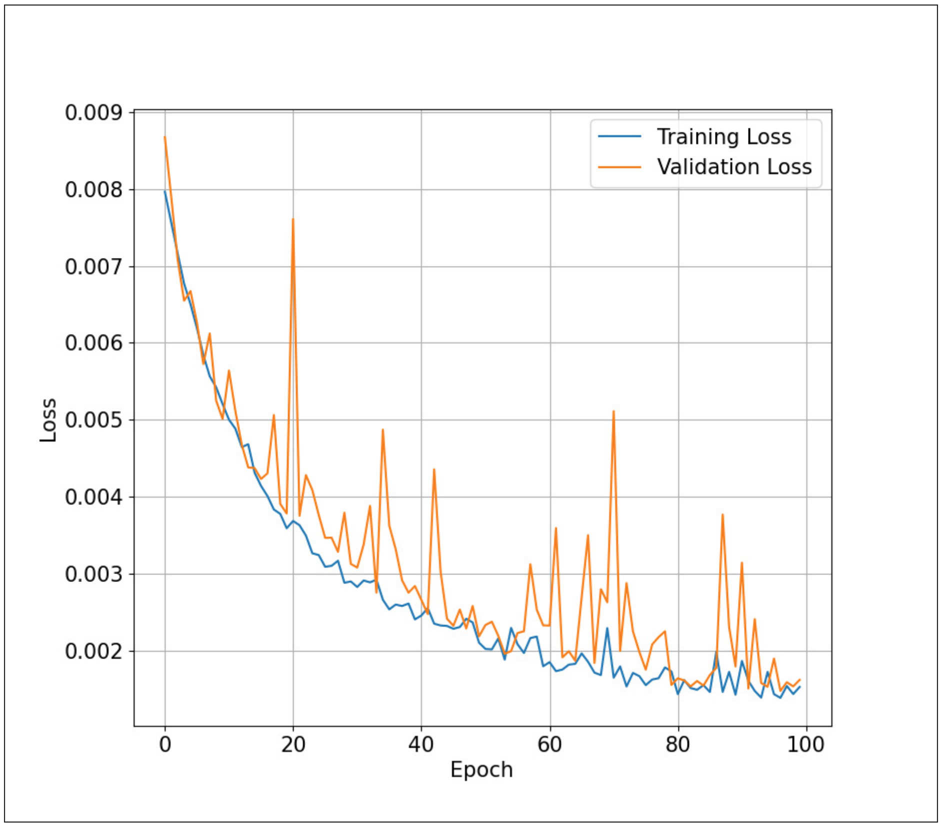


Figure 2.13 Training vs. Testing loss functions for stage 1 of architecture V

topologies that have not been explored yet. The exploration may help the optimisation process to escape from a local optimal, resulting however to the spikes noticed in Figure 2.12. Yet, the optimisation process manage to converge due to this exploitation stage.

Following, we have plotted the loss function for the training and testing datasets across the 100 epochs as shown in Figure 2.13. The learning curve shows the decay of the categorical cross entropy loss function with respect to the number of epochs. This curve is helpful in predicting whether our model is overfitted, underfitted or is fit to testing and training datasets. We see that the loss function for both training and testing decays to low values i.e., 0.001193 for training and 0.001516 for the testing datasets at epoch 100. The spikes are due to the use of a random search hyper tuner and the reasons discussed above. Furthermore, training and testing losses decrease

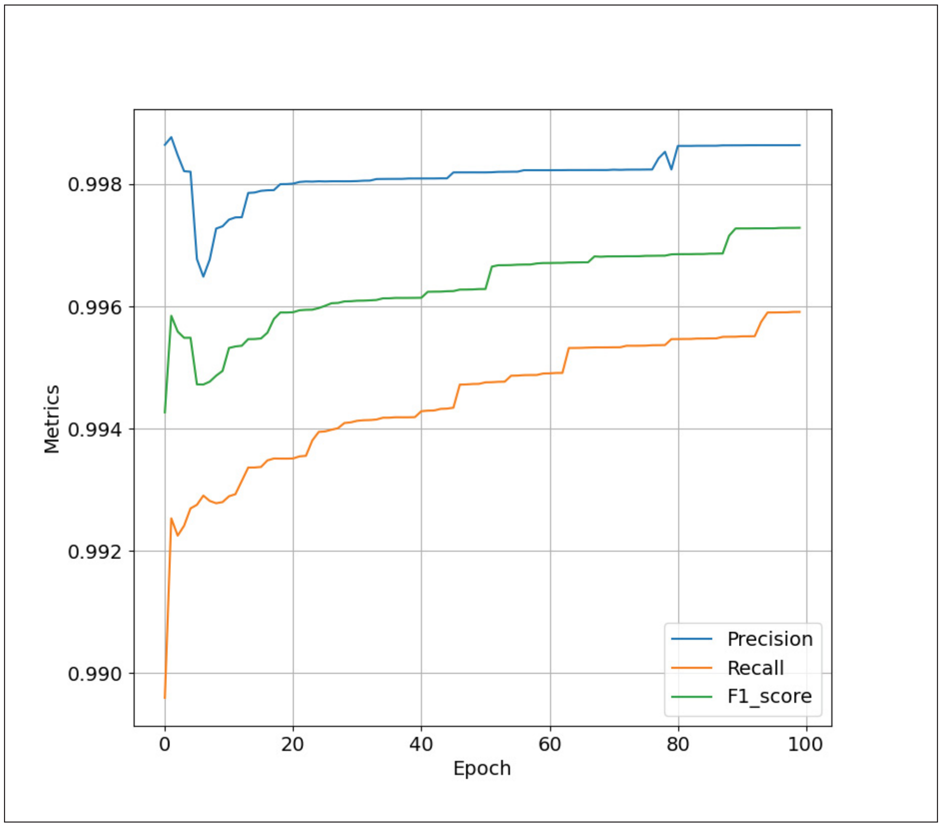


Figure 2.14 Comparison of performance metrics for stage 1 of architecture V over 100 epochs

and are stabilized around the same point i.e., after epoch 80 for training data. The model thus successfully captures the classification patterns.

Next, Figure 2.14 depicts the performance metrics for 100 epochs at stage 1. The precision is high as compared to the other two performance metrics i.e., 0.996923 at the epoch 100. It can also be observed that the precision metric for the neural network does not exhibit significant changes after the epoch 80. Regarding the recall, it is lower compared to the precision and F1-score i.e., 0.9957 at epoch 100 and it shows a constant behavior after the epoch 95. For the F1-score, the value is 0.9964 at the epoch 90 and it does not present any significant changes after this point.

2.7.3 Limitations

Even though our framework provides very encouraging results, it still presents some limitations that stem from the intrinsic data nature of the IoT traffic multi-classification problem. This includes the extra overhead of monitoring the infrastructure to collect the traces, the construction of a training dataset, and the computational overhead for the model training. In addition to that a classification task is a supervised learning approach. This means that if new types of IoT devices are connected in the local network a new cycle of data collection, annotation and training should begin in order to update the model.

2.8 Conclusion

In this work, we studied the problem of IoT traffic classification. To solve this problem we have proposed a composite learning framework that consists of two stages. After an initial data preprocessing, the network traces are passed to stage 0, where a feature selection mechanism and a Logistic Regression classifier are applied. In particular, an ANOVA filter based selection technique decides on the most important features to be used by the stage 0 classifier. The tentative classification of the stage 0 classifier along with the remaining features were then passed to the stage 1 classifier, which used an optimal multi-layer perceptron neural network architecture that provides the final classification.

Following, a detailed experimentation and comparison with various composite architectures on two different IoT datasets have been performed. We concluded that the proposed framework can considerably increase the performance of the classification in terms of recall, precision, F1-score, accuracy and confusion matrix metrics. Regarding the accuracy, our proposed model achieved a 99.9% accuracy for the first dataset and a 99.8% accuracy for the second dataset, proving the generalization aspects of our approach.

The particular model is of utmost importance in an IoT to Cloud continuum communication model, where different IoT devices need to be classified and their traffic profiles be accurately predicted. This precise classification can positively contribute to the proper estimation of the required resources from the subsequent Edge and Cloud layers where the IoT traffic will be processed and analyzed.

The future direction of this work lies in the combination of our proposed model with a resource allocation mechanism that will be able to leverage this workload estimation and dynamically change the allocation strategy at the access and Edge networks. Finally, we aim to include other machine learning techniques such as K-means clustering along with unsupervised methods to address the limitations of classifying new and unknown types of IoT devices.

CHAPTER 3

TOWARDS QOS PREDICTION BASED ON TEMPORAL TRANSFORMERS FOR IOT APPLICATIONS

Aroosa Hameed[1] , John Violos[1] , Aris Leivadeas[1] , Nina Santi[2] , Rémy Grünblatt[3] , Nathalie Mitton[2]

[1]Département de génie logiciel et des TI, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] INRIA Lille-Nord, France
[3] SAMOVAR, Télécom SudParis, Institut Polytechnique de Paris, Évry, France

Paper published in *IEEE Transactions on Network and Service Management*, October 2022

3.1 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, down sampling 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.

3.2 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 (Cisco, 2022). 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 (Pratap, Kumar & Kumar, 2021). 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 (Saeik *et al.*, 2021). 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 (Dechouniotis *et al.*, 2020). 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 (Marjani *et al.*, 2017). 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 (S. & Ram, 2022). Therefore, there is huge interest in analyzing the IoT traffic profiles by applying various machine learning techniques (Hameed *et al.*, 2021).

For example, several studies applied traditional time series algorithms or deep learning models to predict the IoT traffic behavior (Abdellah *et al.*, 2020a)-(Lai, Chang, Yang & Liu, 2018). In the studies (Abdellah *et al.*, 2020a)-(Lopez-Martin, Carro & Sanchez-Esguevillas, 2019), 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 (Abdellah *et al.*, 2020a)-(Hameed *et al.*, 2021) several QoS prediction mechanisms are presented, however, without considering any time dependencies. Secondly, for the works (Fan *et al.*, 2019)-(Lopez-Martin *et al.*, 2019), 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 (Hou *et al.*, 2021), (Fan *et al.*, 2019) and (Wu *et al.*, 2021), which is an important element to capture the dependencies among multiple QoS metrics. Additionally, some works, such as (Hochreiter & Schmidhuber, 1997)-(Lai *et al.*, 2018), applied deep learning specifically for the time series forecasting 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 (Adjih *et al.*, 2015) 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 3.3 presents the related work and current limitations. Section 3.4 gives a detailed information on the challenges of IoT time series data. Section 3.5 summarizes the real time dataset generation of the considered IoT applications. Section 3.6 presents the proposed model along with its algorithmic form and asymptotic analysis. Section 3.7 provides the experimentation setup and illustrates the results and the efficiency of the proposed solution. Finally, Section 3.8 concludes the paper.

3.3 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.

3.3.1 Deep Learning for QoS Prediction

The authors in (Abdellah *et al.*, 2020a), 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 (Ateeq *et al.*, 2019a) 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 (Said & Tolba, 2021) 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 (Hou *et al.*, 2021) 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.* (2019) 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 (Wu *et al.*, 2021), 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.* (2020b) 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 (Lopez-Martin *et al.*, 2019) 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 (Ateeq *et al.*, 2019b), (Hameed *et al.*, 2021), 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.

3.3.2 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 (Hochreiter & Schmidhuber, 1997) 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 (Sutskever, Vinyals & Le, 2014) 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 (Cheng, Xie, Wu, Yu & Li, 2019) 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 (Bai, Kolter & Koltun, 2018b). The authors showed that TCN performed better than RNN models for time series forecasting tasks. Salinas & et al (2020) 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 (Lai *et al.*, 2018) 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.

3.3.3 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 (Fan *et al.*, 2019)-(Lopez-Martin *et al.*, 2019). There are only few studies that provide the QoS prediction (Abdellah *et al.*, 2020a)-(Hameed *et al.*, 2021). 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 (Hou *et al.*, 2021), (Fan *et al.*, 2019) and (Wu *et al.*, 2021). 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 (Hochreiter & Schmidhuber, 1997)-(Salinas & et al, 2020). 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.

3.4 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 $Q = \{q^1_1, q^2_2, ..., q^t_i\}$

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 $M : X_{t,\tau(i)} \longrightarrow \tilde{X}_{t,h(i+\tau)}$ by minimizing some loss function.

Table 3.1 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

3.5 Edge Computing Infrastructure and Dataset Construction

3.5.1 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 Networks (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 3.1, 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 3.1. The only common 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 (Santi *et al.*, 2021).

Table 3.2 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

3.5.2 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 3.2.

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:

$$PDR = \frac{total_{rec\_pack}}{total_{trans\_pack}} * 100 \tag{3.1}$$

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:

$$PLR = \frac{total_{loss\_pack}}{total_{rec\_pack}} * 100 \tag{3.2}$$

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

$$Throughput = \frac{total_{rec\_pack}}{time_{last\_pack} - time_{first\_pack}} \tag{3.3}$$

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:

$$Latency = \frac{\sum_{t\_pack}(time_{rec\_pack} - time_{trans\_pack})}{total_{trans\_pack}} \tag{3.4}$$

3.5.3 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 3.1. 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:

$$X\_normalized = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.5}$$

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.

3.6 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.

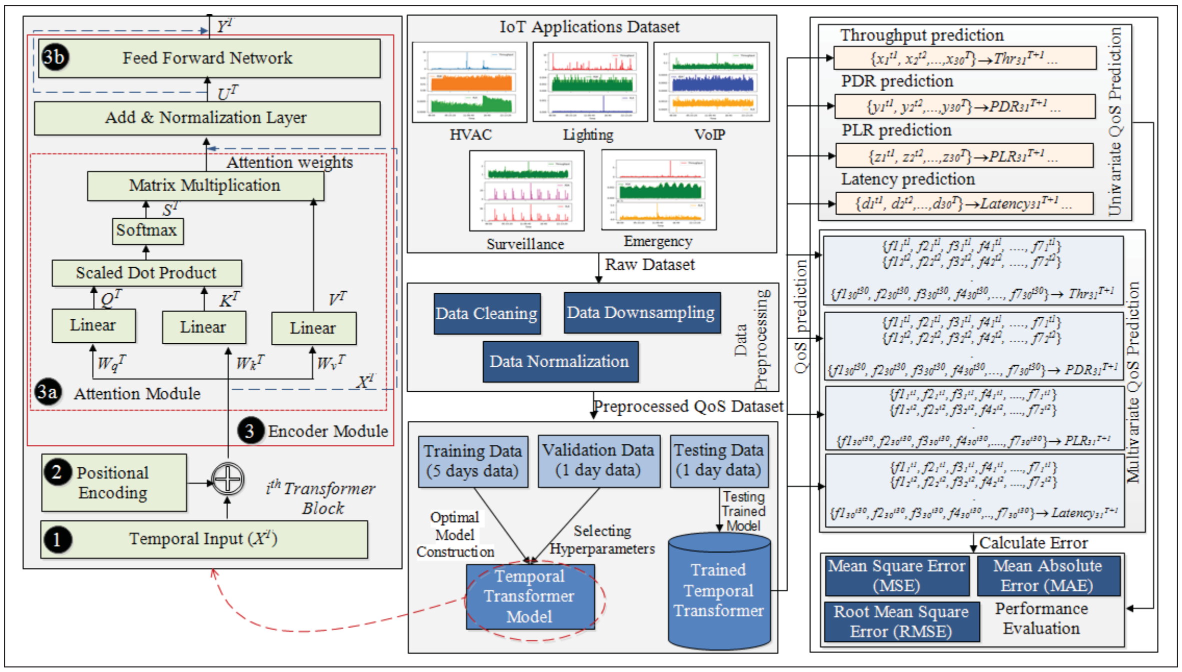


Figure 3.1 Overview of Proposed System for QoS Prediction

3.6.1 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 Figure 3.1, we first generate the real IoT data for five different applications as discussed already in the Section 3.5.1. Then, our second step is to take all of these raw datasets and engineer the new useful features as discussed in Section 3.5.2. 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.

3.6.2 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 (Vaswani *et al.*, 2017) (Zerveas & et al., 2021). 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.

3.6.2.1 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 3.4. 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.

Figure 3.2 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

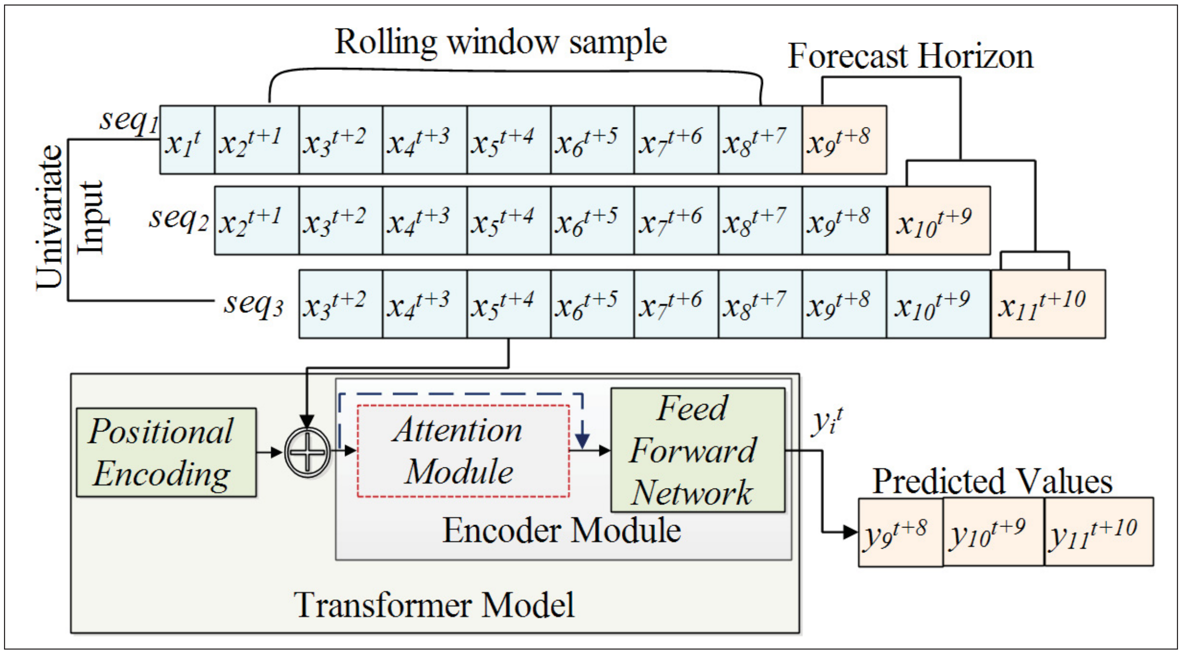

Rolling window sample
Forecast Horizon
Univariate Input
seq_1
x_1^t x_2^{t+1} x_3^{t+2} x_4^{t+3} x_5^{t+4} x_6^{t+5} x_7^{t+6} x_8^{t+7} x_9^{t+8}
seq_2
x_2^{t+1} x_3^{t+2} x_4^{t+3} x_5^{t+4} x_6^{t+5} x_7^{t+6} x_8^{t+7} x_9^{t+8} x_{10}^{t+9}
seq_3
x_3^{t+2} x_4^{t+3} x_5^{t+4} x_6^{t+5} x_7^{t+6} x_8^{t+7} x_9^{t+8} x_{10}^{t+9} x_{11}^{t+10}
Positional Encoding
Attention Module
Feed Forward Network
Encoder Module
Transformer Model
y_i^t
Predicted Values
y_9^{t+8} y_{10}^{t+9} y_{11}^{t+10}

Figure 3.2 Overview of univariate input and output of prediction

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 Figure 3.2 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.

3.6.2.2 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:

$$PE_{pos,2i} = sin(\frac{pos}{10000^(2i/d_{mod})}), 0 < pos < N-1 \tag{3.6}$$

$$PE_{pos,2i+1} = cos(\frac{pos}{10000^(2i+1/d_{mod})}), 0 < pos < N-1 \tag{3.7}$$

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 Figure 3.2. This is then passed to the encoder module of the temporal transformer.

3.6.2.3 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 Figure 3.2. 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 Figure 3.1. 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.

3.6.2.3.1 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 Figure 3.1 and it is computed as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.8}$$

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 Equation (3.9).

$$h_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{3.9}$$

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 Equation (3.10) in order to represent the importance between two QoS values in terms of their correlations.

$$MultiHead(Q, K, V) = concat(h_1, h_2, ..., h_n)W^0 \tag{3.10}$$

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.

3.6.2.3.2 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:

$$FFN(x) = Relu(0, xW_1 + b_1)W_2 + b_2 \tag{3.11}$$

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 3.1 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 $D^{test}$
Output: Future values of QoS as $\tilde{Y} = \{y_1^t, y_2^{t+1}, ..., y_{h-1}^{T-1}, y_h^T\}$
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 $\tilde{Y} \leftarrow model.predict(D^{test})$
```

3.6.3 Algorithm Description

Our proposed QoS prediction algorithm (Algorithm 3.1) 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 3.2 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

Algorithm 3.2 Temporal Transformer Algorithm

```
Input: {$D^{train}, pos, dim, emb_{res}, h_s, num_h, d_{rate}, fil, k_s, act, x, res$}
// $D^{train}$ is the training dataset instances, $pos$ is the input sequence length, $dim$ is the
dimension representation, $emb_{res}$ 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, $act$ is the activation function, $x$ is the output of Module 2 and $res$ is
the results of MHA module within the Module 2.
Output: {$x$}
// $x$ is the predicted QoS value
1  INPUT_EMBEDDING($D^{train}, pos, dim$)
2  set $input \leftarrow Input\_Layer(D^{train})$
3  set $pos \leftarrow Positional\_encoding(pos, dim)$
4  set $emb_{res} \leftarrow Add(input, pos)$                    ▹ Module 1
5  ENCODER_MODULE($emb_{res}$, $h_s$, $num_h$,$d_{rate}$, $fil$, $k_s$, $act$)
6  set $x \leftarrow layer\_norm(emb_{res})$
7  set $x \leftarrow MHA(h_s, num_h, d_{rate}, x)$
8  set $x \leftarrow dropout(x)$
9  set $res \leftarrow x + input$                               ▹ MHA
10 set $x \leftarrow layer\_Norm(res)$
11 set $x \leftarrow layer\_Conv1D(fil, k\_s, act, x)$
12 set $x \leftarrow dropout(d\_rate, x)$                       ▹ Module 2
13 OUTPUT_MODULE($x$, $res$)
14 set $x \leftarrow layer\_GlobalAvgPooling1D(x)$
15 set $x \leftarrow layer\_Dense(x)$
16 set $x \leftarrow Add(x, res)$
17 set $x \leftarrow layer\_Norm(x)$                            ▹ Module 3
```

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 6 to line 9 and the FFN procedure is from line 10 to line 12. 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 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.

3.6.4 Complexity Analysis

Proposition 1: The computational complexity of Algorithm 3.1 is $O(n^2d)$.

Proof: Line 1 of Algorithm 3.1 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 3.1 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 3.2, line 2 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 3 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 4 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 2. 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 2 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 6 is the normalization of the previous layer and takes $O(n)$. Line 7 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 8 and 9 takes $O(n)$ time each because line 8 is applying a dropout operation to n number of neurons and line 9 is performing an addition operation which is performed in $O(n)$ time. Next, lines 10-12 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 14-17 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 3.2 is represented in terms of n as: $O(nd) + O(n^2d) + O(n) = O(n^2d)$.

3.6.5 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

[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

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]

3.7 Performance Evaluation

3.7.1 Model Implementation and Frameworks

3.7.1.1 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.

[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

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

Table 3.3 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

3.7.1.2 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:

$$MAE = \frac{1}{n}\sum_{n=1}^{n} |y_j - \overline{y}_j| \tag{3.12}$$

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:

$$MSE = \frac{1}{n}\sum_{n=1}^{n}(y_j - \overline{y}_j)^2 \tag{3.13}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{n=1}^{n}(y_j - \overline{y}_j)^2} \tag{3.14}$$

3.7.1.3 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 3.3.2. 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 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 (Lai *et al.*, 2018), 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.

Table 3.4 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

3.7.1.4 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 (Lai *et al.*, 2018).

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 transformer 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 3.3 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 3.3 provides the hyper-parameter tuning of the univariate prediction.

3.7.2 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 3.4. In Figure 3.3, 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

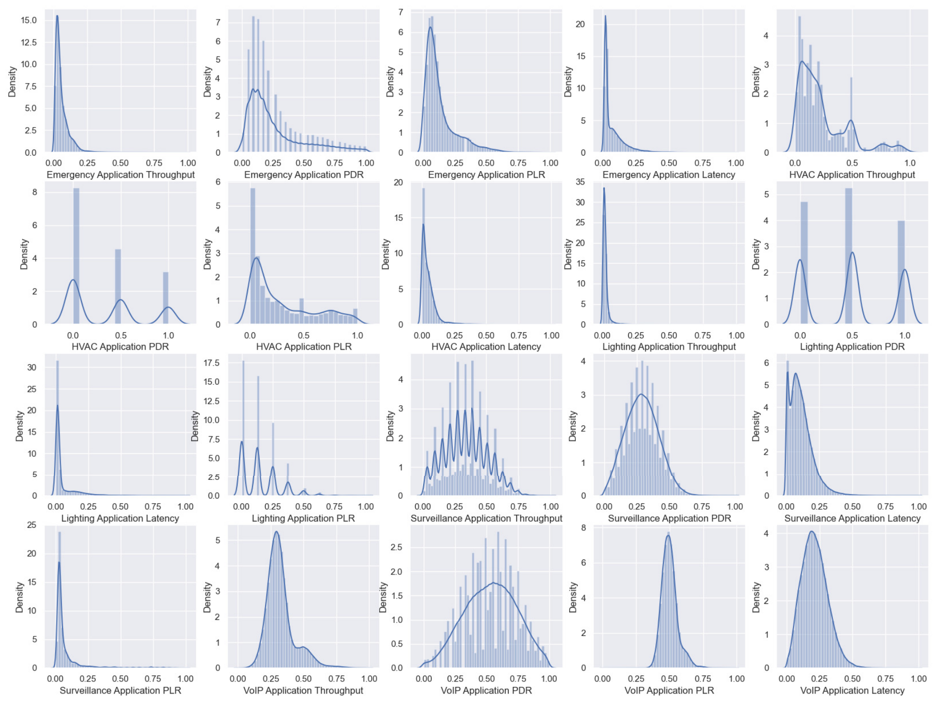


Figure 3.3 Probability distribution plots of QoS data for all IoT applications

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.

Table 3.5 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 3.6 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**

3.7.3 Results

3.7.3.1 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 3.5 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 Figures 3.4 and 3.5 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.

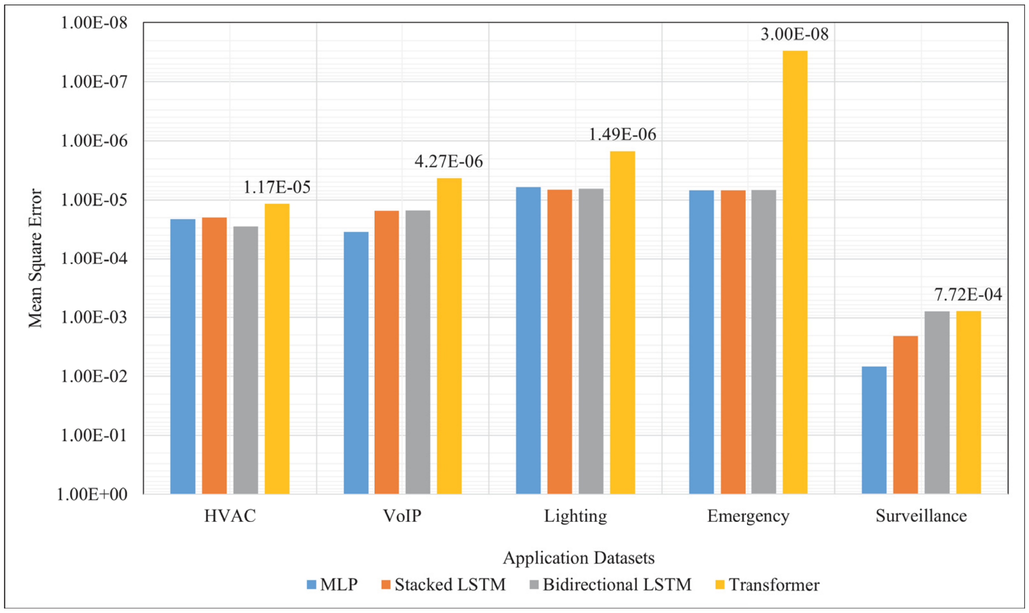


Figure 3.4 MSE of univariate throughput prediction across all datasets

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 3.4. 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.

Table 3.7 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**

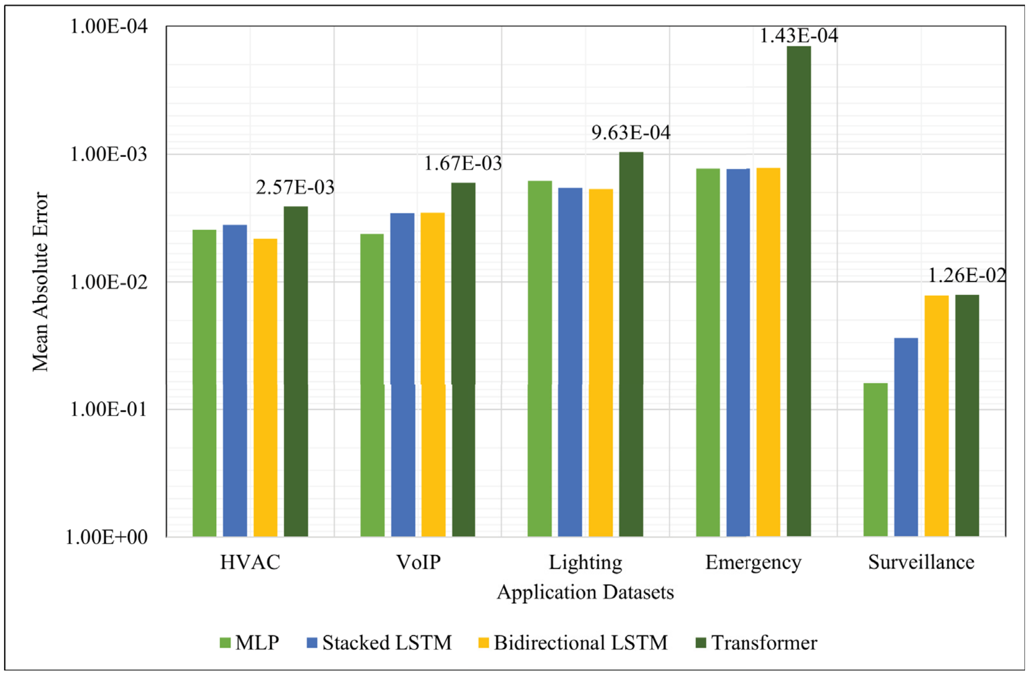


Figure 3.5 MAE of univariate throughput prediction across all datasets

Table 3.8 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**

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 Figures 3.4 and 3.5. 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 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 3.6. 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 Figure 3.6 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

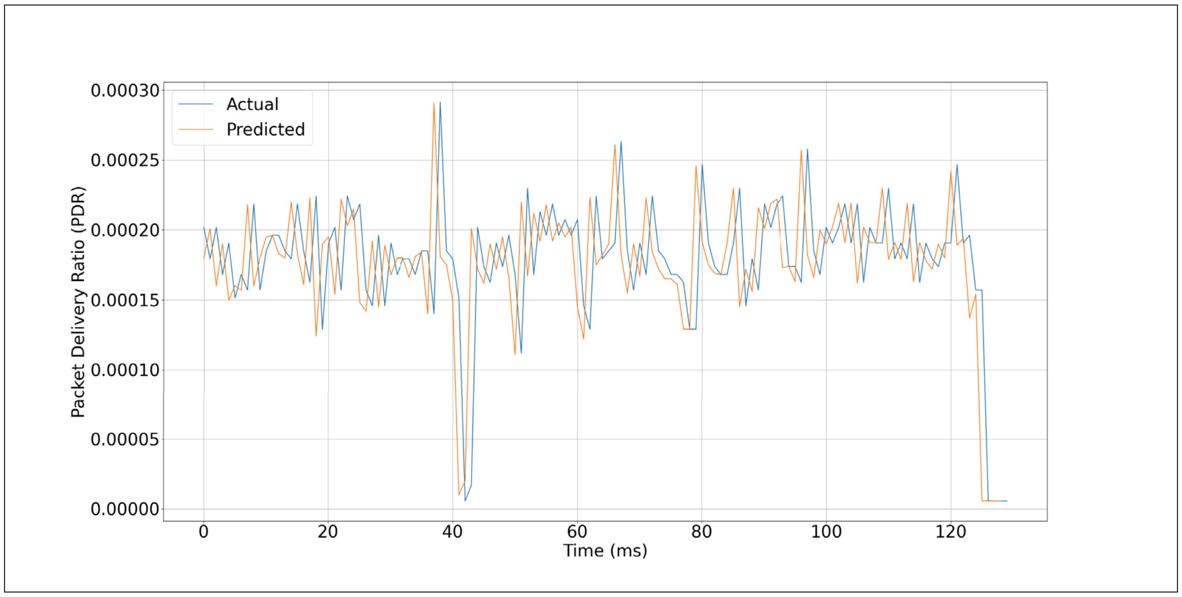


Figure 3.6 PDR prediction for surveillance application

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 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.

Table 3.9 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**

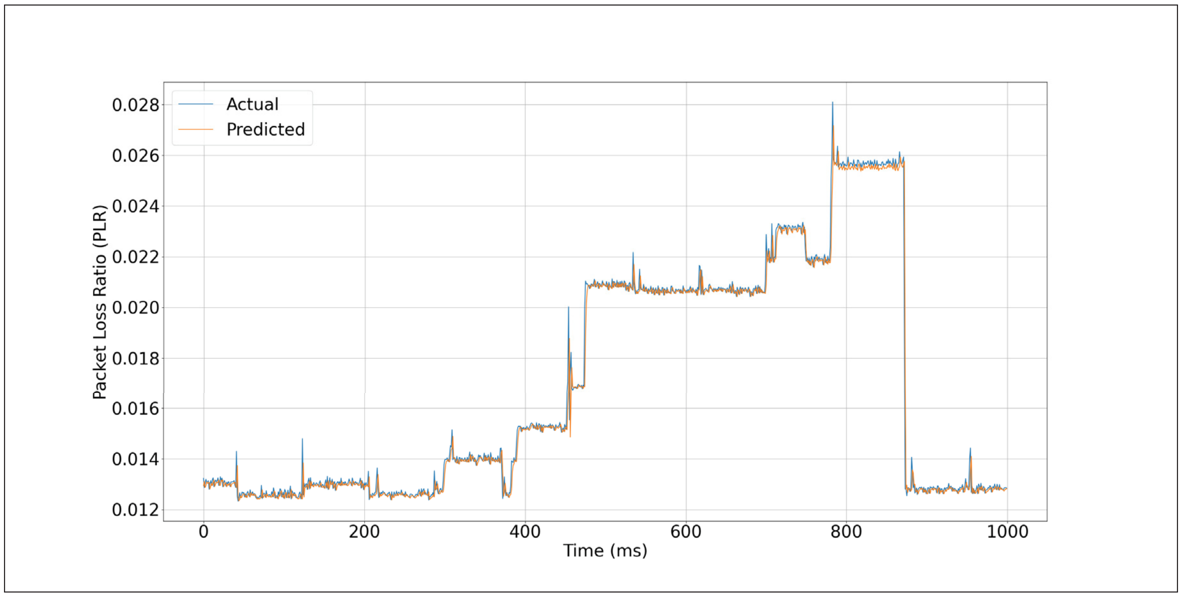


Figure 3.7 PLR prediction for surveillance application

Table 3.10 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**

Following, we provide the univariate PLR results for the five IoT applications in Table 3.7. 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 Figure 3.7, 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 3.8. 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 3.8, 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

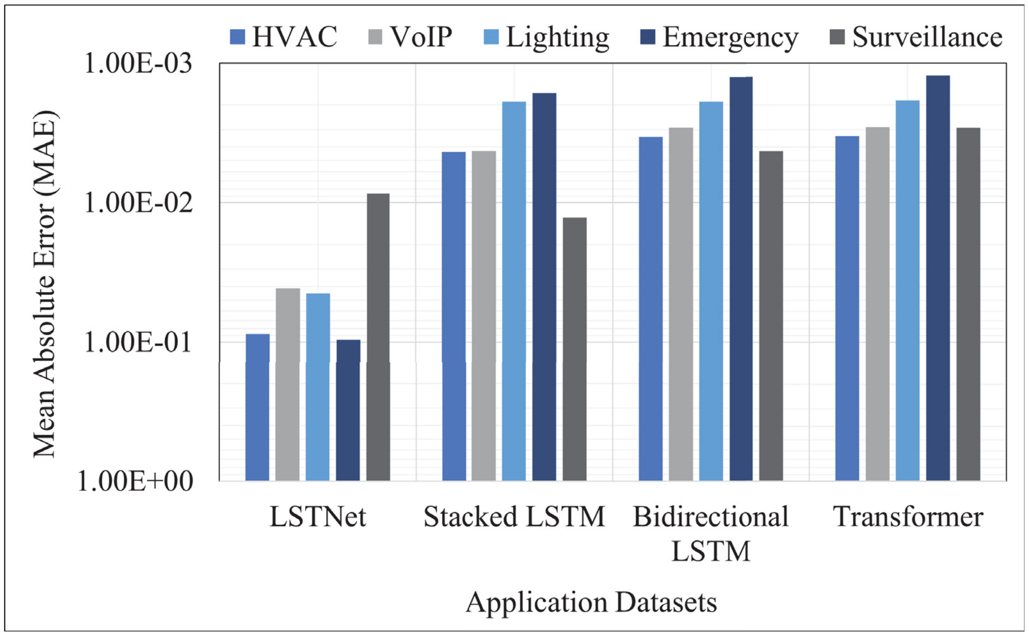


Figure 3.8 MAE of multivariate throughput prediction across all datasets

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

3.7.3.2 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 3.9. To better illustrate these results w.r.t. MAE, we plot them as well in Figure 3.8. 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 3.10 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 Figure 3.9, 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.

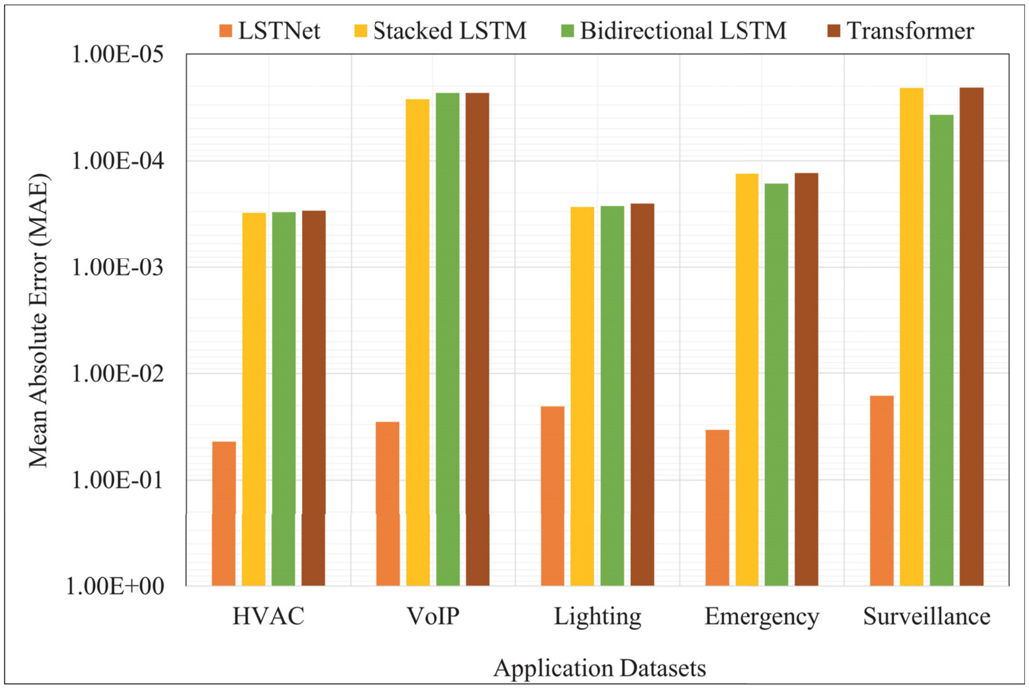


Figure 3.9 MAE of multivariate PDR prediction across all datasets

Table 3.11 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 3.12 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**

Moreover, we provide the results of the PLR prediction in Table 3.11. 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 in the IoT networks.

Lastly, Table 3.12 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.

3.8 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.

CHAPTER 4

FED-TST: FEDERATED TEMPORAL SPARSE TRANSFORMERS FOR QOS PREDICTION IN DYNAMIC IOT NETWORKS

Aroosa Hameed[1] , John Violos[1] , Aris Leivadeas[1] , Nina Santi[2] , Nathalie Mitton[2]

[1]Département de génie logiciel et des TI, École de Technologie Supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] INRIA Lille-Nord, France

Paper Under Review in *IEEE Transactions on Network and Service Management*, May 2023

4.1 Abstract

Internet of Things (IoT) applications generate tremendous amounts of data streams which are characterized by varying Quality of Service (QoS) indicators such as throughput, delay, etc. These indicators need to be accurately estimated in order to appropriately schedule the computational and communication resources of the access and Edge networks. Nonetheless, such types of IoT data may be produced at irregular time instances, while suffering from varying network conditions and from the mobility patterns of the edge devices. At the same time, the multipurpose nature of IoT networks may facilitate the co-existence of diverse applications, which however may need to be analyzed separately for confidentiality reasons. Hence, in this paper, we aim to forecast time series data of key QoS indicators, such as throughput, delay, packet delivery and loss ratio, under different network configuration settings. Additionally, to secure data ownership while performing the QoS forecasting, we propose the FeDerated Temporal Sparse Transformer (FeD-TST) framework, which allows local clients to train their local models with their own QoS dataset for each network configuration; subsequently, an associated global model can be updated through the aggregation of the local models. In particular, three IoT applications are deployed in a real testbed under eight different network configurations with varying parameters including the mobility of the gateways, the transmission power and the channel frequency. The results obtained indicate that the forecasting performance of our proposed approach is more accurate than the identified state-of-the-art centralized and distributed solutions.

4.2 Introduction

The proliferation of Internet of Things (IoT) applications generated a continuous stream of time-stamped data of various granularity. These data need to be analyzed in order to take actions and add the necessary intelligence to the IoT applications. Edge Computing can invoke this intelligence much faster by placing communication and computational resources closer to the source of data (Saeik *et al.*, 2021). However, besides the analysis of the content of the data, there is a second type of analysis, the network analysis, which is equally important (Hanes, Salguiero, Grossetete, Barton & Henry, 2017). Through this analysis, the traffic characteristics of the IoT applications can be learned and the network conditions can be estimated, in order to better schedule the resources of the access and edge networks and to take preventive actions in case a network performance drift is expected to happen.

However, this type of network analysis is a hectic and challenging process for various reasons. First of all, IoT devices belonging to different applications follow disparate data generation modes (i.e., poll-based, periodic, event-driven, etc.). Secondly, IoT access networks are usually wireless and lossy, and they operate in unlicensed bands. This makes them prone to interference and unstable connection. Lastly, the IoT gateways or the devices themselves can be mobile (i.e., robots, drones, etc.), which can also affect the performance of the communication. These complexity levels could lead to a varying Quality of Service (QoS) behavior for each application. Hence, it is important to propose an efficient QoS forecasting model, which will use the time series data of how packets are generated by the IoT devices (i.e., when packets are transmitted/received) along with the network characteristics of the access network (i.e., frequency channel, etc.).

Time series forecasting is the task of analyzing the time-stamped data, both past and present, to make accurate future predictions that can be used in strategic decision making. However, time series forecasting becomes challenging when working with data that contains variables that change frequently (as in the case of IoT access networks) and events that cannot be controlled (Hahn, Langer, Meyes & Meisen, 2023) (as in the case of IoT data generation). In such scenarios, it is important to utilize appropriate techniques to reduce uncertainty and enhance the accuracy of the predictions.

However, due to the complexity and continuously shifting patterns of IoT data under different controllable and uncontrollable factors, it is difficult to apply traditional ML approaches while dealing with the non-stationary QoS forecasting problems. Additionally, the most prominent Machine Learning (ML) techniques used to predict QoS metrics, such as (Ateeq *et al.*, 2019a; Hameed *et al.*, 2021; Bardalai, Neog, Dutta, Medhi & Deka, 2022; Hou *et al.*, 2021; Abdellah *et al.*, 2020a; White & Clarke, 2022; Liu, Sheng, Zhang, Chu & Xu, 2019; Liu, Sheng, Xu, Chu & Zhang, 2022; Li, Wen & Wang, 2019; Jin *et al.*, 2023; Zhang *et al.*, 2020a), lack the ability to handle both long and short term dependencies at the same time, as training over longer sequences of past data degrades the accuracy of the prediction (S. & Ram, 2022). Recently, some works also applied transformer models for time series forecasting (Hameed *et al.*, 2022), because of their ability to capture long-term dependencies in the data using the combination of positional encoding and multi-head self attention mechanisms. Nonetheless, the transformer model requires time and memory that grows quadratically with the sequence length, which excludes their use on long sequences. Consequently, this motivated us to introduce a sparse transformer model, that entails less complexity for long sequences without sacrificing performance.

Nonetheless, a typical problem of both transformer models (sparse and traditional) is that they require large amount of data to be trained. In the IoT context, this could lead to stressing the computing entity that has to gather the time series data and analyze them. Additionally, to increase local efficiency and to respect confidentiality requirements, the data streams of different IoT applications may have the exigency of being treated separately. Accordingly, these two reasons inspired us to resort to a more distributed machine learning approach leveraging the Federated Learning (FL) technique. FL allows multiple clients to train a shared global model, by aggregating the local updates from decentralized and distributed clients, in order to improve the global model's accuracy, while tackling data scarcity and preserving the privacy of clients data (Mangla, 2022). To the best of our knowledge, this is the first work that introduces an adaptation of federated sparse transformers to forecast the QoS metrics of an IoT communication network.

More specifically, in this work, we deploy three different IoT applications in a real testbed to predict typical QoS metrics. For this prediction, a sparse transformer-based architecture is introduced that efficiently leverages the time dependency by investigating both the short and long input sequence dependencies without any performance degradation. Finally, we explore the effectiveness of our QoS forecasting sparse transformer in a FL setting in order to enhance the accuracy while coping with the data heterogeneity, scarcity and privacy issues. Accordingly, the main contributions of this work can be summarized as follows:

- We consider three different real time IoT applications that present different requirements in terms of number of devices and data generation distribution.
- We deploy the applications in a real testbed comprised of 100 IoT devices and 3 mobile robots generating data over an IEEE 802.15.4 access network, creating a lossy communication under a random access network for different mobility, transmission power and frequency channel configurations.
- We design and implement a QoS predictor based on the Sparse Temporal Transformer approach that models the temporal dependencies within long input sequences of data and computes the attention scores using only a subset of the input positions.
- We provide both univariate and multivariate predictions of four major QoS metrics such as Throughput, Packet Delivery Ratio (PDR), Packet Loss Ratio (PLR) and Latency.
- We implement a FL scheme with multiple clients to enable the collaborative learning of our sparse transformer models by leveraging the distributed historical data of different IoT applications.

The remainder of the paper is structured as follows. Section 4.3 highlights relevant works on QoS forecasting in an IoT/Edge environment. Section 4.4 introduces the system model and problem statement. Section 4.5 describes the testbed implementation and the dataset generation. In Section 4.6, the detailed design of our proposed solution, named FeD-TST, is provided. Section 4.7 discusses the attained results, while Section 4.8 summarizes our conclusions and offers some future directions.

4.3 Related Work

QoS forecasting can be a key concern in an IoT context. Accordingly, traditional ML approaches used to play a major role in QoS prediction for their simplicity and interpretability. For instance, the authors in (Ateeq *et al.*, 2019a) predicted the delay of IoT applications in an IEEE 802.15.4 access network using a Deep Neural Network (DNN) with forward and backward passes. Similarly, the authors in (Hameed *et al.*, 2021) proposed the use of several regression based approaches to predict the throughput of six different IoT applications in an IEEE 802.15.4 access network. Additionally, the authors in (Bardalai *et al.*, 2022) predicted the throughput of audio, video and sensor data of an IoT healthcare application using an ARIMA/GARCH model. For the throughput prediction, a Convolutional Neural Network (CNN) with a target vectorization technique was also used in (Hou *et al.*, 2021).

In terms of both single and multi-step ahead prediction, the authors in (Abdellah *et al.*, 2020a) predicted the delay of a simulated dataset of an IoT environment using a nonlinear autoregressive exogenous (NARX) Recurent Neural Network (RNN). A different type of RNN called Echo State Network (ESN) was also employed in (White & Clarke, 2022), for QoS forecasting at the edge of an IoT network. The authors also introduced random noise into the internal states of the network in order to provide more robust forecasts, while addressing the stability problem of ESNs.

Regarding QoS prediction at the Multi-Access Edge Computing (MEC), Liu *et al.* (2019) provided a multi QoS prediction framework using contextual factors. Firstly, they introduced a workload prediction mechanism for future scheduling services using Support Vector Machine (SVM) and optimized it with an Artificial Bee Colony (ABC) algorithm. Secondly, they performed the multi QoS prediction using Case Based Reasoning (CBR), which is a problem-solving methodology that involves using past experiences (cases) to solve new problems. This work was further extended in (Liu *et al.*, 2022) providing the QoS prediction for both real-time and future MEC services, using CBR and an optimized SVM. In contrast, the authors in (Li *et al.*, 2019) introduced a matrix factorization method that predicts unknown QoS values using a location based user cluster's information and user's reputation information.

So far, all of the above proposed methods are centralized, while the data privacy is not considered. Accordingly, Jin *et al.* (2023) proposed a privacy-aware QoS forecasting model based on Long Short-Term Memory (LSTM) and attention called Edge-PMAM (Edge QoS forecasting with Public Model and Attention Mechanism). This work consisted of public and private models for privacy aware and personalized QoS forecasting. The Edge regions were divided using the Miller projection with k means clustering and for each region the model consisted of an attention layer on top of a LSTM to improve the performance of the public and private models. Furthermore, Zhang *et al.* (2020a) designed a QoS prediction model based on FL. The authors firstly identified the untrustworthy users and then processed the unreliable data to predict the QoS in a MEC setting.

However, the above mentioned studies have different limitations that can be summarized as follows:

- The existing studies based on traditional ML approaches in (Ateeq *et al.*, 2019a)-(Hou *et al.*, 2021) do not consider the temporal aspect of the prediction. In contrast, these studies employ static models that treat each input sample independently, while not considering the sequential or time-dependent nature of the data. Furthermore, they only forecast one of the two most typical QoS metrics i.e., throughput or delay.
- The RNN models used in (Abdellah *et al.*, 2020a), (White & Clarke, 2022) and (Jin *et al.*, 2023) can only handle the short term sequence prediction efficiently. Specifically, during back-propagation, where gradients are propagated from the output to the input, the gradients can diminish or vanish as they are repeatedly multiplied by weight matrices and cause the vanishing gradient problem. As a result, the network struggles to update the weights effectively, particularly for earlier time steps, limiting its ability to capture long-term dependencies.
- The CBR approach used in (Liu *et al.*, 2019) and (Liu *et al.*, 2022) is difficult to be applied in the context of IoT, where a vast amount of data is generated from various heterogeneous

devices. This could make building a comprehensive case base challenging, while the selection of relevant cases from the case base is difficult in large and heterogeneous datasets.

- The work in (Li *et al.*, 2019) and (Zhang *et al.*, 2020a) used matrix factorization techniques, which heavily depend on the data sparsity at the current time slots. Furthermore, the proposed approach in (Li *et al.*, 2019) assumed that the user clusters and user reputations remain fixed over time. However, network conditions and user behavior can be dynamic, which may require the model to be updated or retrained. Moreover, the work in (Zhang *et al.*, 2020a) used reputation mechanism which relies on assigning reputation scores to clients based on their trustworthiness. However, determining the trustworthiness of users can be subjective and prone to biases.

Herein, we solve the above mentioned challenges as follows: (i) Firstly, we provide the detailed forecasting of four QoS metrics such as throughput, PDR, PLR, and latency by considering the temporal aspects in both univariate and multivariate settings; (ii) Secondly, to overcome the vanishing gradient problem in the training of long QoS data sequences, we are introducing a temporal transformer with sparse attention. The particular architecture can efficiently model long sequences by capturing dependencies between different positions without suffering from the vanishing gradient problem. Additionally, the sparse attention mechanism further optimizes the computational efficiency of our approach, particularly for resource-constrained IoT environments. (iii) Thirdly, we employ a FL approach that trains the client models on diverse IoT devices and sensors data, resulting in a more comprehensive QoS forecasting. Furthermore, our FL model continuously updates the model using distributed data from IoT devices ensuring that the model adapts to changing network conditions and application dynamics. Finally, the FL aggregates model updates from multiple clients providing a more objective and collective perspective on client behavior and trustworthiness.

4.4 System Model and Problem Statement

4.4.1 System Model

We consider a set A of different IoT applications such that, $A = \{a_1, a_2, ..., a_k\}$. For each application $a_k \in A$, there is a number of n static sensor nodes denoted by the set $S^k = \{s_1^k, s_2^k, ..., s_n^k\}$ that generate data following a data distribution D^k, with $S^1 \cup S^2 \cup ... \cup S^k = S$. Furthermore, for each application, the dataset is generated for a specific duration T. Each dataset can be represented by a sequence of data points $x_{n,t}^k$ that describe the data generated by sensor n, belonging to application k at time step $t \in T$. To lighten the notation, the k superscript that denotes the application which a sensor node n is associated with will be dropped in the rest of the paper.

Similarly, we can represent the set of access points (APs), that form the edge computing environment, as a set of mobile robots $R = \{r_1, r_2, ..., r_m\}$. Each mobile robot $r_m \in R$ has a set of coordinates $G_m = \{(x_{m,1}, y_{m,1}), (x_{m,2}, y_{m,2}), ..., (x_{m,T}, y_{m,T})\}$ that represent its movement trajectory over a period of time. The static sensor nodes continuously transmit their generated data to the mobile robots for each application.

The sensors belonging to an application $a_k \in A$, send their data to a unique robot r_m for all time steps T, which is represented as a binary function $z : A \times R \times T \rightarrow \{0, 1\}$. The function $z(a_k, r_m, t) = 1$ if application a_k sends data to robot r_m at timestamp t, and $z(a_k, r_m, t) = 0$ otherwise. In other words, there is one source and one destination of data for each sensor node within each application at a specific timestamp.

4.4.2 Modeling of Network Uncertainties

One of the contributions of this work, is to find an appropriate prediction model that will be able to accurately estimate major QoS metrics, under dynamic network conditions. Thus, four different network dynamics/uncertainties are considered:

1. **Interference:** At each time slot that a pair of sensor nodes $s_n, s_{n'} \in S$ utilizes the same frequency channel, an interference level $I_{n,n'} = f(P_n, d_{n,n'}, \epsilon)$ is created, where P_n represents the transmission power of sensor node s_n, $d_{n,n'}$ is the distance between sensors s_n and $s_{n'}$ and ϵ represents the characteristics of the wireless channel. In this work, we have tested two different channel allocation techniques: i) all applications can use the same frequency channel (inter and intra-application interference) or ii) each application is associated with a different frequency channel (intra-application interference).
2. **Transmission power:** Each sensor $s_n \in S$ can be set with a different transmission power P_n to find a balance between transmission range and interference. Herein, the transmission power can be either 0 or $12dBm$.
3. **Mobility:** The robots acting as the mobile access points can also create another level of communication uncertainty. To evaluate the impact of mobility in the QoS prediction, two mobility settings are available: i) static robots (i.e. $G_m = \emptyset$) and ii) mobile robots ($G_m \neq \emptyset$).
4. **Heterogeneity of data:** Finally, the QoS estimation can be affected by the way the data are generated (i.e., event-based, periodic, or hybrid). Accordingly, during the generation of the datasets all the above three data generation distribution have been used.

Based on the above described complexities, there are two interference models I i.e., same channel or different channel allocation, two power models P i.e., either 0 or $12dBm$ and, two mobility patters G i.e., static and mobile. Hence, κ network configurations are examined, with $|\kappa| = |I| \times |P| \times |G| = 2 \times 2 \times 2 = 8$. For all these network configurations, the applications follow a heterogeneous data generation with the three distributions mentioned above (e.g., event-based, periodic, hybrid).

4.4.3 Problem Formulation of QoS Forecasting

In our proposed cross-device FL setting, the set $C = \{c_1, c_2, ..., c_l\}$ represents the number of clients, which are connected to a global server $\mathcal{GS}$. Furthermore, the set $Q^\kappa = \{q_1^t, q_2^t, q_3^t, q_4^t\}^T$ represents the different QoS metrics attained for all network configurations κ, where q_1^t is the throughput, q_2^t is the PDR, q_3^t is the PLR and q_4^t is the latency metric at timestamp t for the different IoT applications. Then each client $c_l \in C$ has its own local QoS dataset, which can be represented as $D_l \subseteq Q^{\kappa}$.

Specifically, D_l is a multivariate QoS dataset, such as $D_l = \{(q_1^1, q_2^1, q_3^1, q_4^1), (q_1^2, q_2^2, q_3^2, q_4^2), ..., (q_i^t, q_i^t, q_i^t, q_i^t)\}$ where each tuple $(q_1^t, q_2^t, q_3^t, q_4^t)$ represents a single observation in the dataset at time t. For a univariate setting (i.e., only throughput), the QoS dataset is represented as $D_l = \{q_1^1, q_1^2, ..., q_1^t\}$ and similarly for the other metrics.

For each QoS dataset (either univariate or multivariate time series), a fixed length input sequence of window size τ such that $\tau \in \mathbb{N}$, is used to predict the next values. To generalize, for any time series X the input sequence with a window size τ can be represented as $\{(x_{t+i_1}, x_{t+i_2}, ..., x_{t+i_\tau}) \mid 1 \leq i_1 \leq T - \tau + 1, 1 \leq i_2 \leq T - \tau + 2, ..., 1 \leq i_\tau \leq T - \tau + \tau\}$. Given this input sequence, we aim at performing a multi-step forecasting of Δ QoS values in the future, i.e., $\tilde{X} = \{\tilde{x}_1^t, \tilde{x}_2^{t+1}, ..., \tilde{x}_{\Delta-1}^{T-1}, \tilde{x}_\Delta^T\}$. Similarly, if $\Delta = 1$ it becomes a one-step ahead problem. Hence, the forecasting is performed using a learning model $\mathcal{M}$, such as $\mathcal{M} : X \to \tilde{X}$ which is parameterized by some weights $\mathcal{W}$. Thus, the goal is to minimize a loss function, between the real X and predicted $\tilde{X}$ values.

Hence, in the FL context, each client c_l will iteratively use its local training dataset D_l and training model $\mathcal{M}_l$ in order to minimize the loss function, such as:

$$\arg\min_{\mathcal{W}_l^k} \mathcal{L}_l^k(\mathcal{M}_l^k, D_l^k, \mathcal{W}_l^k) \tag{4.1}$$

where $\mathcal{L}_l^k(.)$ represents the loss function at round/iteration k, with $K = \{1, 2, 3, ..., k | k \in \mathbb{N}\}$, with the $\mathcal{M}_l^k$, D_l^k, and $\mathcal{W}_l^k$ as its parameters. Each local weight $\mathcal{W}_l^k$ is then transmitted to the server for updating the server's global weight parameter $\mathcal{W}_k^*$ at iteration k, which is called the global update. Then, the learning problem tries to find an optimal model parameter vector $\mathcal{W}_k^*$ and the goal is to minimize the global loss function $\mathcal{L}^*$ over the number of iterations k using a given dataset.

Table 4.1 IoT applications parameters for data generation

Application	#Nodes	Generation type	Lambda	Period(s)
VoIP	50	Periodic	0	0.0635
Surveillance	25	Exponential	196.74	1
Emergency Response	25	Hybrid	0.0333	30

4.5 Use Case and Dataset Generation

In order to create the QoS datasets, we implemented an IoT/Edge Computing use case composed of 100 static transmitting sensor nodes that want to offload their data. Each sensor node belongs to one of the three available IoT applications: i) VoIP, that emulates an automatic help desk virtual assistant, ii) Surveillance, which includes a set of cameras for security, and iii) Emergency Response, which monitors critical areas of a building (leakage on gas pipes, fire alarms, etc.). The three applications produce data according to different distribution modes. Specifically, the periodic mode generates data every i^{th} time instance, the event-based mode produces data following an exponential law with an occurrence rate of λ, and the hybrid mode which is a combination of both periodic and event-based modes. Table 4.1 provides an overview of the parameters used for the applications under consideration. Furthermore, each application is associated with a unique mobile robot that receives this data while moving according to a configured path.

The particular use case was implemented in the FIT IoT-LAB (Adjih *et al.*, 2015), an open testbed that allows large-scale experiments and provides openly programmable IoT nodes located on several sites. We used the IoT-LAB M3, Zolertia Firefly and Decawave DWM1001 boards as IoT nodes and the turtlebots as the mobile access points. All nodes communicate with each other through the 802.15.4 wireless protocol, thereby rendering them suitable for our IoT and Edge Computing use case. The locations of each sensor node in the experimental setup are shown in Figure 4.1.

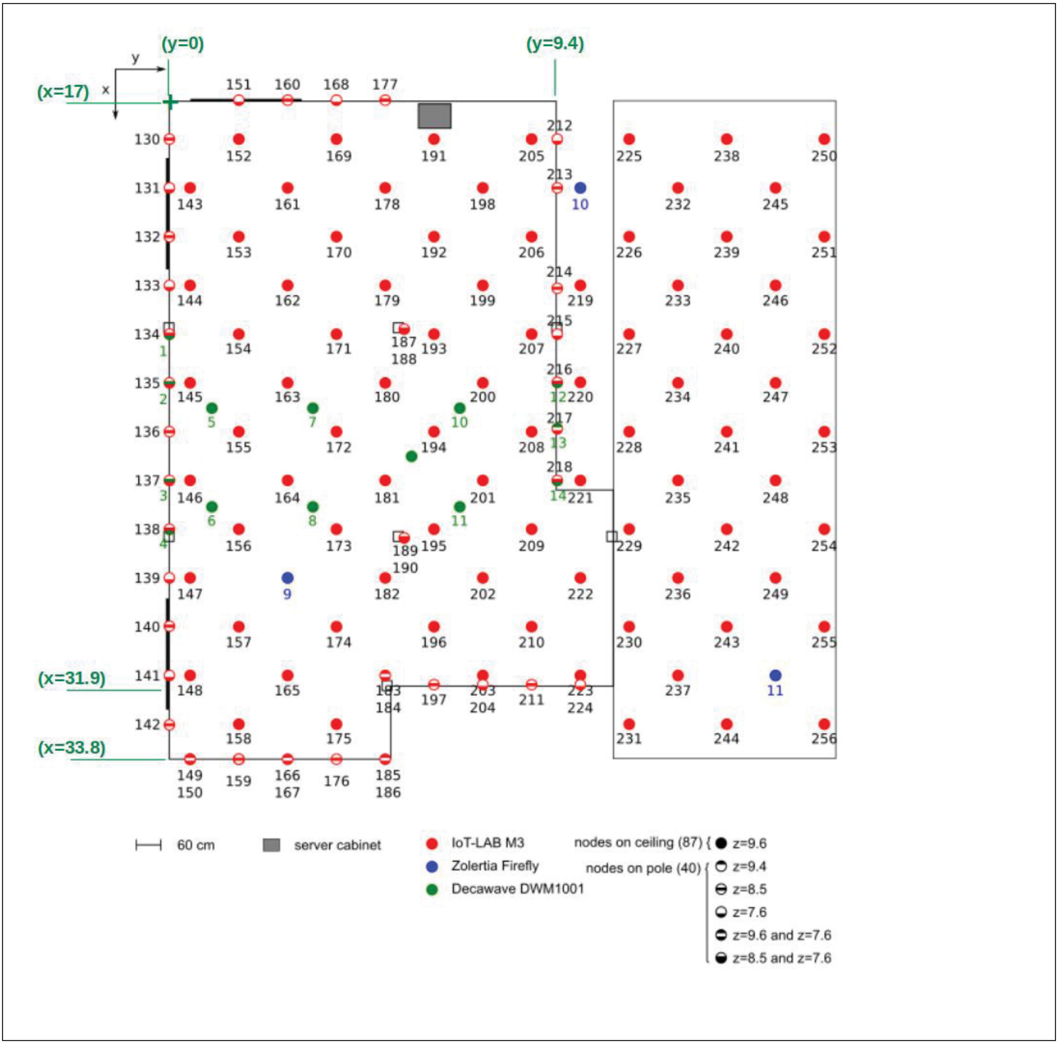


Figure 4.1 Sensor nodes location deployed in FIT IoT-LAB

As stipulated by the particular access protocol, the payload size was set to 127*B* for all applications in all configurations. For the latter, and as explained in Section 4.4.2 there are eight network configurations, which are presented in Table 4.2.

We collected all the data transmitted by the sensor nodes and the data received by the mobile robots. These raw data include the features at sending side and features at receiving side. Specifically, at the sending side we collected the timestamp at which a message is transmitted; the name of the sensor node that transmits the message; the id of the transmitted message; the transmission power; the frequency channel; the receiving robot; whether the message was

Table 4.2 Network configurations with different complexities

Configurations	Mobility	tx_power	Channel	Channel value
1	Mobile	0dBm	Different	{11,16,21}
2	Mobile	12dBm	Different	{11,16,21}
3	Mobile	0dBm	Same	{11}
4	Mobile	12dBm	Same	{11}
5	Static	0dBm	Different	{11,16,21}
6	Static	12dBm	Different	{11,16,21}
7	Static	0dBm	Same	{11}
8	Static	12dBm	Same	{11}

transmitted successfully or not; and the x and y coordinates of the robot. At the receiving side, the following features were collected: timestamp; the name of the receiving robot; the message id; the name of the transmitting sensor; the delay; the channel; and the coordinates of the robot.

Following, a feature engineer process was followed that resulted into the final QoS time series datasets used for our experimentation. Specifically, the features engineered using the initial raw data are: timestamp; robot name; total transmitted messages at specific timestamp; total received messages at the specific timestamp; the time when the first message is transmitted; the time when the last message is transmitted; PLR; Delay; PDR; and Throughput.

4.6 Proposed Model

4.6.1 Overview of FeD-TST

To forecast the QoS metrics of varying network uncertainties, we present FeD-TST, which combines the Federated Learning with Temporal Sparse Transformer (TST). FeD-TST is a secure distributed system for both univariate and multivariate Time Series Forecasting (TSF) as shown in Figure 4.2.

In this framework, there is a central server and multiple client nodes. The clients will first train their models locally based on the TST and then share their model weights with the other clients

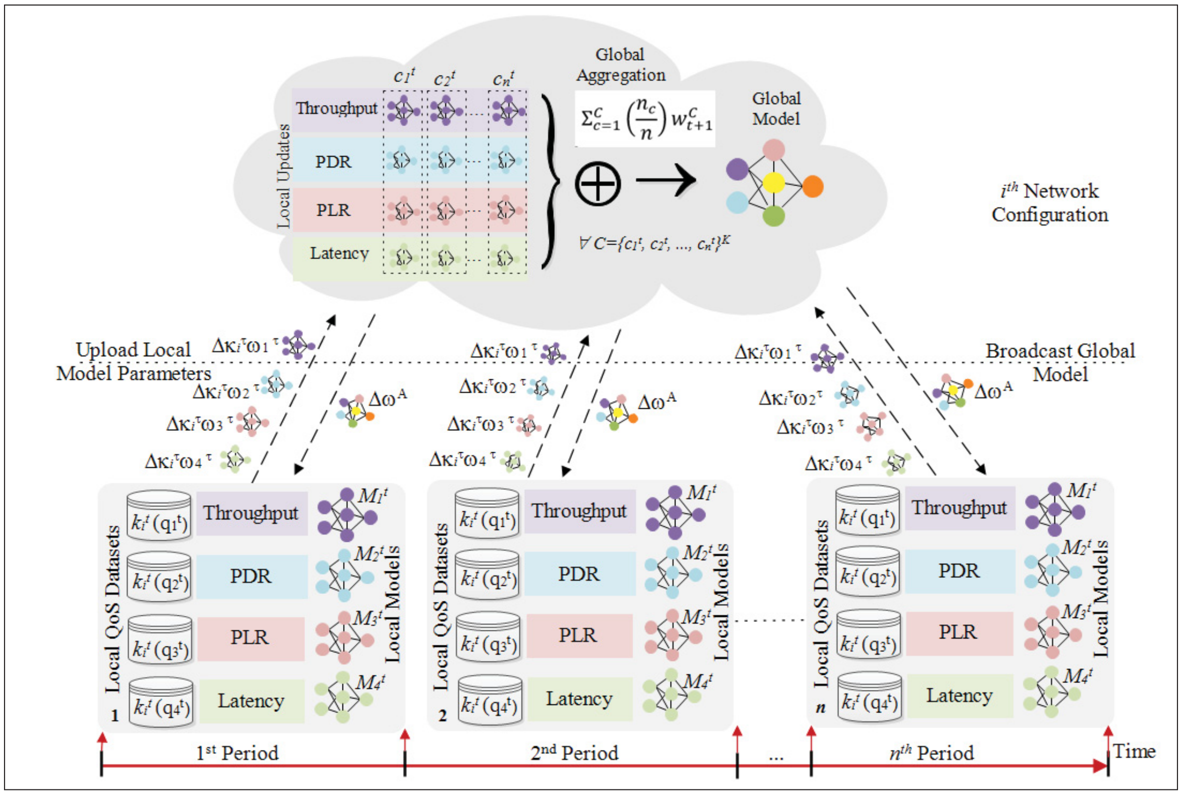


Figure 4.2 Federated learning driven IoT QoS Forecasting System

via the central server. For each connected client and for each network configuration, the hidden layer weights and parameters of the client model (i.e., TST) are represented as ω. After the uploading of this information, the global aggregation is performed at the FeD-TST server and the global model parameters are sent back to each client. Following, each client loads these global weights to its TST model's hidden layers.

The TST model of each local client is illustrated in Figure 4.3. In the proposed TST, we first generate the real time IoT data of the three different applications and eight different network configurations as discussed in Section 4.5. Following, these datasets are pre-processed using normalization, down sampling and cleaning procedures and are prepared as univariate or multivariate inputs. The third step is to divide the dataset into training, validation and testing. The training involves several components, such as the encoder module as shown in Figure 4.3a, the decoder module as shown in Figure 4.3b, the multi-head sparse attention module as shown

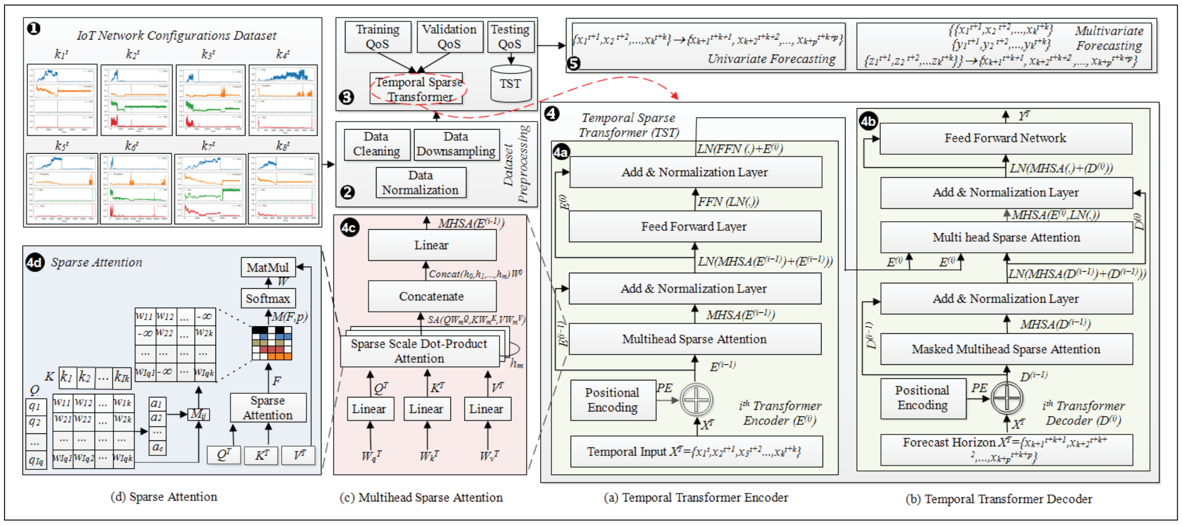

IoT Network Configurations Dataset
Training QoS
Validation QoS
Testing QoS
Temporal Sparse Transformer
TST
Data Cleaning
Data Downsampling
Data Normalization
Dataset Preprocessing
Univariate Forecasting
Multivariate Forecasting
Temporal Sparse Transformer (TST)
Add & Normalization Layer
Feed Forward Layer
Multihead Sparse Attention
Positional Encoding
Temporal Input
Feed Forward Network
Multi head Sparse Attention
Masked Multihead Sparse Attention
Forecast Horizon
Sparse Attention
MatMul
Softmax
Linear
Concatenate
Sparse Scale Dot-Product Attention
(d) Sparse Attention
(c) Multihead Sparse Attention
(a) Temporal Transformer Encoder
(b) Temporal Transformer Decoder

Figure 4.3 Proposed Temporal Sparse Transformer (TST) Model for IoT QoS Forecasting

in Figure 4.3c and a sparse attention module as shown in Figure 4.3d. Both the encoder and decoder modules consist of multiple identical sub-layers that are stacked to each other. Finally, uni/multivariate forecasts are produced along with the evaluation based on the test dataset. In the rest of this section, we provide the details of each of the component of our proposed model.

4.6.2 Sparse Scaled Dot-Product Attention

The traditional transformer model utilizes the self-attention which is a form of a global attention to model the long-term dependencies (Vaswani *et al.*, 2017). In this work, the sparse scaled dot-product attention is employed to overcome the shortcoming of the global attention, which is to attend all the positions in the input QoS sequence leading to attend irrelevant or noisy information that might not be beneficial for the QoS forecasting task. As shown in Figure 4.3d, a sparse scaled dot-product attention based on top-p selection is used to reserve the most important QoS segments. The attention mechanism consists of three main parameters named query, key and value, which consist the input of the sparse attention. Specifically, the query, $Q \in \mathbb{R}^{m \times d_q}$ is a query vector of a given input sequence of a certain QoS metric at a specific time step to retrieve information from the key-value pairs. Each query vector is a transformed version of the corresponding time step in the QoS input sequence and is computed using a linear transformation. The key, $\mathcal{K} \in \mathbb{R}^{n \times d_k}$ is a vector for each time step in the QoS sequence to determine the relevance of each element in the sequence with respect to the query. Finally, the value, $\mathcal{V} \in \mathbb{R}^{n \times d_v}$ is a vector containing information associated with each time step in the QoS sequence to compute the output of attention mechanism. The n denotes the length of the key-value pairs, m is the length of the query vector, d is the dimension of the corresponding QoS vector, and $d_q = d_k$. To be more specific, $\mathcal{Q}$, $\mathcal{K}$ and $\mathcal{V}$ are the linear mappings of the input QoS sequence, such as $\{x_1^1, x_2^2, x_3^3, ..., x_i^t\}$, so that, $\mathcal{Q} = W_{\mathcal{Q}} x_i^t$, $\mathcal{K} = W_{\mathcal{K}} x_i^t$, $\mathcal{V} = W_{\mathcal{V}} x_i^t$, where $W_{\mathcal{Q}}$, $W_{\mathcal{K}}$ and $W_{\mathcal{V}}$ denote the learned weight matrices. To generate the attention scores, $\mathcal{F}$, the dot product of $\mathcal{Q}$ and $\mathcal{K}$ is performed which is then divided by $\sqrt{d_k}$ as follows:

$$\mathcal{F} = \mathcal{Q}\mathcal{K}^T / \sqrt{d_k} \tag{4.2}$$

The attention scores computed using the above equation represent the relation between different QoS values e.g., x_1^1 and x_2^2 in the input sequence. For example, the larger the attention score values are, the higher the relevance of the QoS values will be at a specific time step. After computing the attention scores, the sparse attention performs a masking operation $M(.)$ on the scores to select the top-p contributing values as follows:

$$M(\mathcal{F}, p)_{ij} = \begin{cases} \mathcal{F}_{ij} & if \mathcal{F}_{ij} \geq \theta_i \\ -\infty & otherwise \end{cases} \tag{4.3}$$

The θ_i represents the p^{th} largest element of each row i of the score matrix $\mathcal{F}$. After obtaining the highest attention values through the top-p selection, the softmax operation is applied to normalize the scores as:

$$SA\_scores = SoftMax(M(\mathcal{F}, p)) \tag{4.4}$$

where SA_scores denotes the normalized sparse attention values and $M(.)$ is assigned a negative infinity value if the attention scores are less than the threshold value.

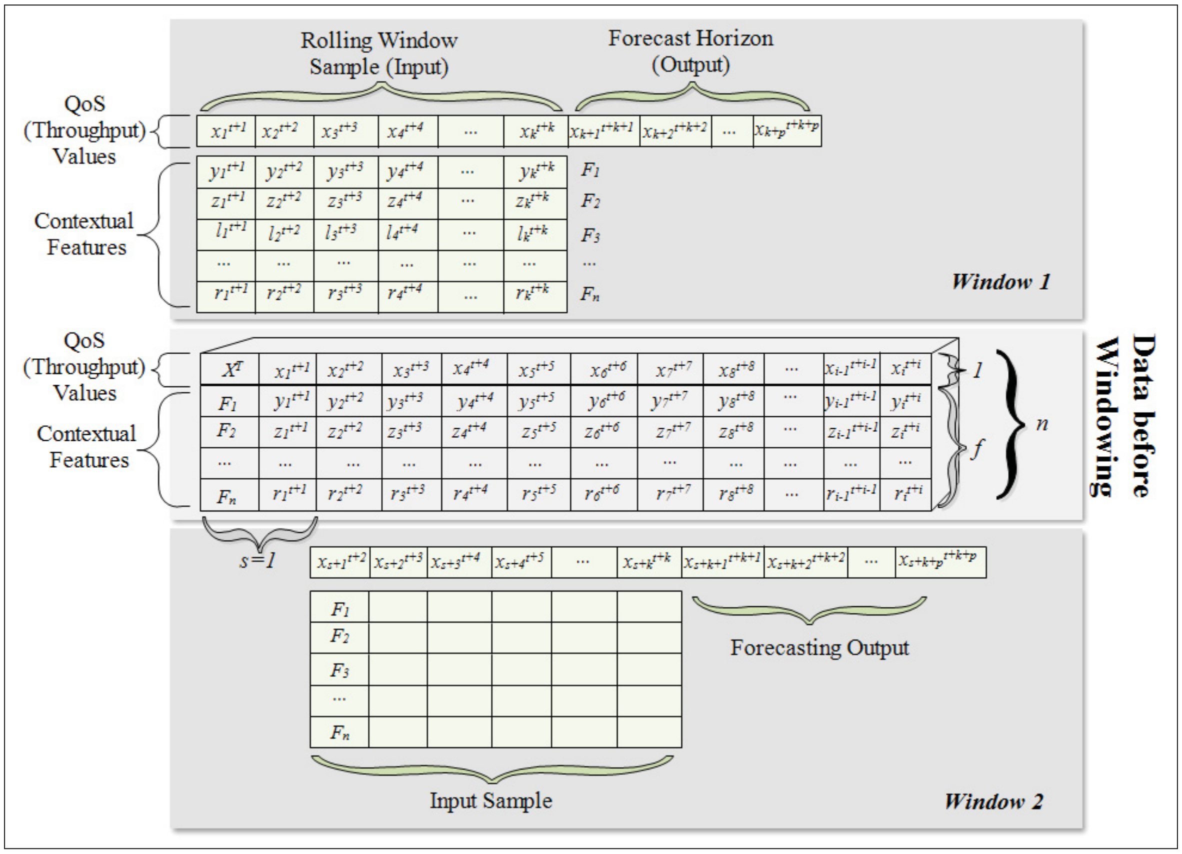


Figure 4.4 Overview of Sliding window for Input and Output of FeD-TST Model

4.6.3 Temporal Sparse Transformer

4.6.3.1 Input and output of the TST Model

The QoS values are both the input and output of the TST, since past QoS values (input) will be used to predict the future ones (output). Additionally, some contextual features are added only to the input to help with the prediction of the future QoS values. Specifically, during the training of the TST model, the input to the encoder module consists of n contextual features, such as $\{F_1, F_2, ..F_n\}$, along with the QoS values, both forming a set of X^T input samples. Similarly, the input samples for the decoder module consists of the same contextual features with however, the actual (target) QoS values for a defined forecast horizon. In contrast, the output of the decoder consists of the predicted QoS values. As mentioned earlier, we are solving both the univariate and multivariate QoS forecasting. Therefore, if it is a univariate forecasting setting, then the input sample consists of only one contextual feature i.e., timestamp and a forecasting feature (i.e., throughput), thus, $n = 2$. In case of multivariate forecasting, the QoS metric (i.e., throughput) will be forecasted based on the previous time steps of all contextual features, namely the timestamp (F_1), total transmitted messages (F_2), total received messages (F_3), time first message transmitted (F_4), time last message transmitted (F_5), PDR (F_6), PLR (F_7), latency (F_8) and throughput (F_9) itself. This means that the input samples are created using multiple contextual features and $n > 2$. However, the final forecasting output generated by the TST model will be the throughput values for the specific forecasting horizon. The same approach will be applied for the other three QoS metrics, such as, PDR, PLR and latency.

The contextual features along with the QoS values form the time series vectors, and they need to be converted into a suitable format to be used by the TST model. For this, the sliding window technique is used and is depicted in detail in Figure 4.4. In the sliding window approach, the time series vectors are divided into smaller windows of fixed length, also called input samples, and each window is passed as an input to the TST model. Related to these input samples/windows, there are two important parameters i) the rolling window size, which is the length of the sliding window that determines the number of time series data values in each window and ii) the forecast horizon, which is the number of future steps to be predicted.

In Figure 4.4, the first input sample called “Window 1" consists of the first k values of a certain QoS metric, thus one input sample has a dimension of $k \times n$ where n is the number of features. Given the Window 1 as an input to the TST model, the model can predict the QoS values for p number of steps ahead, i.e., from the time step x_{k+1}^{t+k+1} to x_{k+p}^{t+k+p}, based on the window of the previous samples. The TST output contains the QoS values without the contextual features. Here, the rolling window size is represented as k and the forecast horizon as $k + p$. For the next input sample, the rolling window size is shifted iteratively with a step size of 1, represented as s, with $s < k$. Thus, the next window consists of input samples from x_{s+1}^{t+2} to x_{s+k}^{t+k}, and the expected output contains QoS values for time steps x_{s+k+1}^{t+k+1} to x_{s+k+p}^{t+k+p}.

4.6.3.2 QoS Temporal Positional Embedding

The TST model encodes the temporal position information to extract the long-term temporal dependencies from the time series QoS input sequences. The traditional attention mechanism can also capture the relations among the QoS input sequence (Reza, Ferreira, Machado & Tavares, 2022), however, it neglects the ordering information in such time-series data. Therefore, in order to make use of the order of the input sequence, there is a positional encoding added in the TST model to add the location information to each input sequence value. The positional encoding can be described as follows:

$$PE_{pos,2i} = \sin\left(\frac{pos}{\alpha^{2i/d}}\right) \tag{4.5}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{\alpha^{2i/d}}\right) \tag{4.6}$$

where pos is the positional index of the QoS value in the input sequence, i is the length of the QoS input sequence, α is a user defined scalar and d is the dimensionality of the encoded position in the input sequence.

4.6.3.3 Long-Term Temporal QoS Extraction

To solve the long-term QoS forecasting problem, we propose a TST network to extract the long-term temporal relations from the entire QoS input sequence. The TST model consists of ρ number of QoS TST encoders and the corresponding ρ number of QoS TST decoders, and its architecture is shown in Figure 4.3a and Figure 4.3b. Each QoS forecasting encoder consists of the multi-head attention, fully-connected feed forward network, dropout (Pan, Coen-Cagli & Schwartz, 2021) and layer normalisation (Liu, Ren, Zhang, Sun & Zou, 2021) components. To express the i^{th} TST encoder, the abstract form is given as follows:

$$E^{(i)} = TST(E^{(i-1)}) = \mathcal{LN}(\mathcal{FFN}(\tilde{E}^{(i)}) + \tilde{E}^{(i)}) \tag{4.7}$$

where $E^{(i)}$ is the output of the i^{th} TST encoder, $\mathcal{LN}(.)$ is the operation of normalization layer, $\mathcal{FFN}(.)$ represents the fully connected feed-forward network and it can be expanded into:

$$\mathcal{FFN}(\tilde{E}^{(i)}) = max\left(0, \tilde{E}^{(i)}W^1 + b^1\right)W^2 + b^2 \tag{4.8}$$

where $\mathcal{FFN}(.)$ contains two fully-connected layers with dropout and a ReLU activation with W^1, b^1, W^2 and b^2 as its corresponding learnable parameters. The $\mathcal{FFN}(.)$ thus gives the nonlinear transformation of its input. Additionally, $\tilde{E}^{(i)}$ denotes the intermediate feature of the QoS encoder and can be represented as follows:

$$\tilde{E}^{(i)} = \mathcal{LN}(MHSA(E^{(i-1)}) + E^{(i-1)}) \tag{4.9}$$

where $MHSA(.)$ is the multi-head sparse attention, which uses m different linear transformations and analyzes the previous QoS encoder layer features i.e., $E^{(i-1)}$. The multi-head sparse attention process can be written as:

$$MHSA(E^{(i-1)}) = Concat\,(h_0, h_1, h_2, ..., h_m)\,W^0 \tag{4.10}$$

where $Concat$ is the concatenation operation performed on all attention heads, W^0 is the linear transformation of the concatenated output, and h_m represents the m^{th} attention head and is given as:

$$h_m = \mathcal{SA}\left(QW_m^Q, KW_m^K, VW_m^V\right) \tag{4.11}$$

where $\mathcal{SA}$ is the sparse attention as discussed in section 4.6.2 and is computed using Equations 4.2-4.4.

The QoS decoder of the TST has the same structure as the QoS encoder, however, two additional operations are added. Firstly, a subsequent mask to the first attention block is added, which ensures that the forecasting of the position i in the input sequence can only rely on the known outputs of positions which are less than i, thus avoiding the auto-regressive behavior. This addition is given as:

$$\tilde{D}^{(i)} = \mathcal{LN}(M\tilde{H}SA(D^{(i-1)}) + D^{(i-1)}) \tag{4.12}$$

where $M\tilde{H}SA$ is the masked multi-head attention for the decoder and $D^{(i-1)}$ represents the output from the previous decoder.

Secondly, an attention block is added to the decoder that performs the attention operation on the output of the encoder block. Based on these discussed additions, the QoS decoder is given as:

$$\begin{aligned} D^{(i)} &= \mathcal{LN}(MHSA(\tilde{D}^{(i)}, E^{(i)}) + \tilde{D}^{(i)}) \\ \hat{D}^{(i)} &= \mathcal{LN}(FFN(D^{(i)}) + D^{(i)}) \end{aligned} \tag{4.13}$$

where $E^{(i)}$ is the output from the i^{th} encoder and $\hat{D}^{(i)}$ is the output from the i^{th} decoder.

4.6.4 Proposed FeD-TST

The key idea of FeD-TST scheme is that the clients co-train the TST model presented above, while keeping the QoS data locally for each network configuration. To be more specific, FeD-TST, as shown in Figure 4.2 consists of the following steps:

4.6.4.1 Initialization

The secure client and server connection is first established, along with the server's selection of a subset of clients from all connected clients to participate in the training for the following round/iteration. After all clients are enumerated, the model parameters are initialized and broadcast to each client, such as the number of encoder and decoder blocks ρ, the number of communication rounds $\mathcal{K}$, the number of local epochs $\mathcal{E}$, the learning rate η, the loss function $\mathcal{L}$, the initial weight w^0 which is the starting point of the global model's parameters before the training process begins, and the time step size s.

4.6.4.2 Training of the local TST models on Clients

After the initialization step, each local client trains a TST model using their respective local QoS data for the k^{th} round, and the training results will be used for the next $k + 1$ round. The same process repeats at each round.

4.6.4.3 Uploading of the local TST model weights

When the training of each client's local TST model is completed, each client extracts the respective TST weights and then uploads them to the server for the next round $k + 1$.

4.6.4.4 Aggregation of the model weights

According to the received TST model weights from each client, the server then performs the aggregation mechanism, called Federated Averaging (FeDAvg), which aggregates the local model updates from the participating clients and computes a global model update. The aggregated results are then broadcast to each client. This iterative process is repeated until the loss function converges or a maximum number of rounds is reached. The algorithmic implementation of all the components presented in this Section along with their complexity analysis is provided in the Appendix 1.

4.7 Experimental Evaluation

In this section, the evaluation of our proposed FeD-TST framework is provided. First, a brief overview of the network configurations is provided and their respective datasets. Following, the baseline comparative methods and the performance metrics are outlined . Lastly, the results of all experiments are provided along with their the comparative analysis.

4.7.1 Model Implementation and Frameworks

4.7.1.1 Network Configurations and Datasets

Under the time series forecasting settings, we forecast the four following QoS metrics as: (i) Throughput; (ii) PDR; (iii) PLR and (iv) Latency in two different time series settings such as univariate and multivariate. The window size for both settings is set to be 30. For the network configurations with mobility as a parameter, the duration of the corresponding QoS datasets is 6 hours stipulated by the battery duration of the mobile robots. In contrast, for the network configurations with static access points, the duration of the respective QoS datasets is 10 hours. We applied the FeD-TST framework to each of the network configuration individually under both univariate and multivariate settings. Finally, it is noted that we use 60% of the data for training, 20% of the data for validation and the remaining 20% for testing purposes.

4.7.1.2 Baseline Methods

For comparison purposes, we evaluate our proposed FeD-TST model against the most popular centralized and decentralized learning models that are appropriate for time series forecasting. Regarding the centralized baselines we have used the i) LSTM, ii) Bidirectional LSTM (Bi-LSTM), iii) Attention + BiLSTM, iv) Sequence to Sequence (Seq2Seq), v) Seq2Seq + Attention (Li *et al.*, 2022a), vi) Temporal Convolutional Networks (TCN) + BiLSTM (Chen, Kang, Chen & Wang, 2020), and vii) Time2Vec + Transformer (Kazemi *et al.*, 2019). Finally, for the decentralized comparison we have employed the i) FL + LSTM, and the ii) FL + Multi-head Attention (MHA), which is a temporal transformer model (Hameed *et al.*, 2022) adapted to the FL setting. The description of all these approaches along with the hyperparameters used are provided in the Appendix 2.

4.7.1.3 Performance Metrics

In order to evaluate the performance of each method, three widely used metrics have been selected: Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE), given as:

$$\begin{aligned} MAE &= \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \\ MSE &= \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \\ RMSE &= \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \end{aligned} \tag{4.14}$$

where $\hat{y}_i$ and y_i are the forecasted QoS value and its corresponding ground truth. To further evaluate the effectiveness of the proposed approach, we use the communication cost model as proposed in (Xia, Ye, Tao, Wu & Li, 2021). The particular cost is modeled as $2 \cdot \mathcal{K} \cdot \mathcal{N} \cdot \Omega \cdot \mathcal{M}_{size}$ where $\mathcal{K}$ is the total number of communication rounds, $|\mathcal{LC}|$ is the total number of client nodes, Ω is the fraction of client nodes to be selected, $\mathcal{M}_{size}$ is the size of the ML model. The latter is equal to the total number of trainable parameters $P_{\mathcal{M}}$ multiplied by the size of the parameters in bits Υ i.e., (4bits, 8bits, 16bits, 32bits).

4.7.1.4 Implementation Details

We implemented the proposed FeD-TST model and all baseline federated approaches using the Flower Federated Framework (Beutel *et al.*, 2022) with 512 local clients for four QoS datasets under eight network configurations, respectively. For each of the QoS forecasting, we trained every algorithm for 30 communication rounds. All the centralized and decentralized models were executed in a Python environment with open-source TensorFlow libraries. All models were trained on high-performance Linux 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.

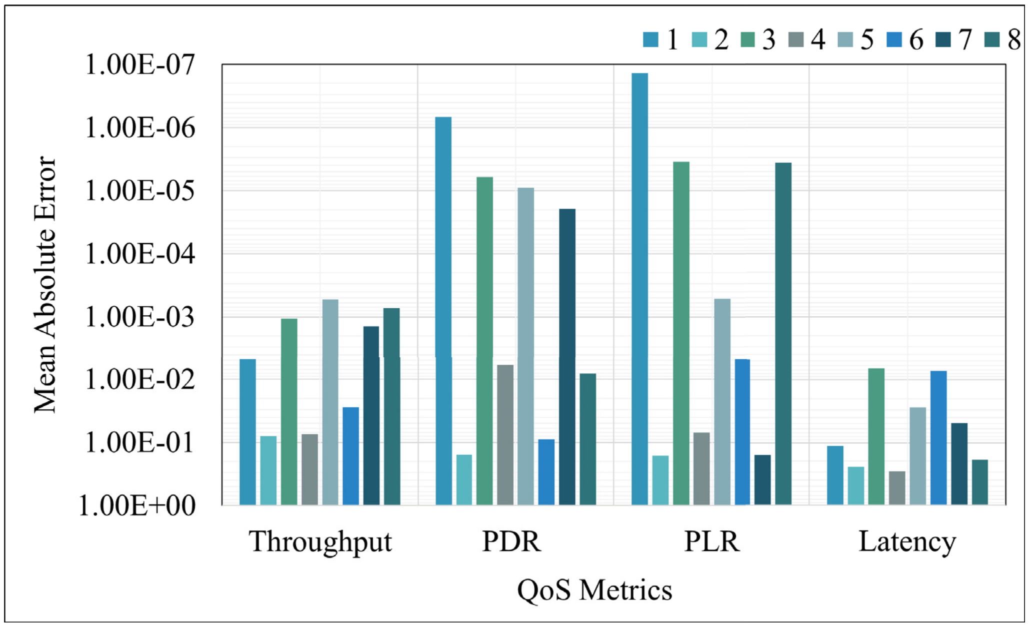


Figure 4.5 Network configuration impact on QoS forecasting

4.7.1.5 Results

For both of the time series forecasting settings, we present the accuracy efficiency of all four different QoS metrics, while considering the eight different network configurations. In particular, the results will be assessed with respect to the impact of network configurations, the communication cost, and the forecasting efficiency of the FL model.

4.7.2 Impact of Network Configurations

To evaluate the impact of the network conditions on the prediction accuracy, we first plot the MAE when using our proposed FeD-TST solution under all eight network configurations. As shown in Figure 4.5, the forecasting error for the throughput is the least for the fifth network configuration (i.e., *Static_diff_channel_0dbm*) followed by network configuration 8 (i.e., *Static_-same_channel_0dbm*). This is because the robot access points remained stationary, which reduced the variability in the network conditions leading to more stable transmissions. Thus, the deep learning model can learn more easily such scenarios.

Table 4.3 Univariate forecasting results for throughput, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.002180008	0.058064374	0.017221582	0.019661525	0.007008886	0.000043146	0.028044095	0.007652375
	MAE	0.031486913	0.199266738	0.082996892	0.073941417	0.075111779	0.001767353	0.130296992	0.053803794
	RMSE	0.046690557	0.240965503	0.131231025	0.140219554	0.08371909	0.006568581	0.167463712	0.087477852
BiLSTM	MSE	0.002149963	0.058196797	0.017138673	0.008286225	0.006855734	0.000059553	0.027234415	0.007399871
	MAE	0.03109576	0.201700344	0.08100321	0.043326636	0.073171018	0.001759178	0.12784306	0.051513432
	RMSE	0.046367696	0.241240122	0.130914755	0.091028704	0.08279936	0.007717055	0.165028528	0.086022505
ATT+BiLSTM	MSE	0.002113405	0.06045324	0.016387171	0.008071625	0.007019641	0.000026737	0.028635743	0.007840934
	MAE	0.03286028	0.203475403	0.078651295	0.043766762	0.074738003	0.001766987	0.132077915	0.055272665
	RMSE	0.045971786	0.245872406	0.128012387	0.089842222	0.083783299	0.005170763	0.169220988	0.088549048
Seq2Seq	MSE	0.002172816	0.058766967	0.017565562	0.007910387	0.006828733	0.000036975	0.028332603	0.007925499
	MAE	0.034427306	0.202370203	0.082538805	0.045117619	0.074294335	0.001800946	0.131756	0.053090038
	RMSE	0.046613471	0.24241899	0.132535134	0.088940356	0.082636146	0.006080671	0.168322913	0.089025274
Seq2Seq+ATT	MSE	0.002108766	0.057875243	0.017857116	0.007336685	0.007259331	**0.000011513**	0.027428818	0.007885314
	MAE	0.031982373	0.201074426	0.094457217	0.042848431	0.072002338	**0.00172142**	0.125179741	0.054124594
	RMSE	0.045921299	0.24057274	0.13363052	0.085654454	0.085201709	**0.003393085**	0.165616478	0.08879929
TCN+BiLSTM	MSE	0.00217524	0.05769644	0.016973583	0.007007257	0.007166868	0.000020477	0.02818864	0.008333666
	MAE	0.035093832	0.198256349	0.081892429	0.045072616	0.076047148	0.001763019	0.133661025	0.062342796
	RMSE	0.046639469	0.240200832	0.130282705	0.083709361	0.084657358	0.004525111	0.16789473	0.091288916
T2V+Transformer	MSE	0.002094785	0.059257085	0.016875335	0.006954281	0.006820699	0.000048149	0.03231086	0.008300878
	MAE	0.032487425	0.204504672	0.080115092	0.0402296	0.074963033	0.001738727	0.139963191	0.053888586
	RMSE	0.045768823	0.243427782	0.129905099	0.083392333	0.082587524	0.006938978	0.179752218	0.091109152
FL+LSTM	MSE	0.026109913	0.198509365	0.019059159	0.009396685	0.007740845	0.000105776	0.036035966	0.008500859
	MAE	0.121700913	0.245523065	0.083601423	0.048500054	0.073999718	0.004875254	0.153384194	0.05503843
	RMSE	0.161543608	0.444212198	0.13799575	0.096909925	0.087978683	0.010284722	0.189783856	0.092178144
FL+MHA	MSE	0.002752693	0.093995392	0.029476583	0.013653285	0.015197324	0.000150662	0.063143753	0.035793204
	MAE	0.038101345	0.248628452	0.130680636	0.061146908	0.101738915	0.007402018	0.198087156	0.147313178
	RMSE	0.052466109	0.306586683	0.171687454	0.116847269	0.123277426	0.01227443	0.251284212	0.189190924
FeD-TST	MSE	**3.67E-05**	**0.007006747**	**0.000226781**	**0.006597863**	**3.48511E-05**	0.001039662	**0.003347858**	**0.001060865**
	MAE	**0.004760905**	**0.079123348**	**0.001064445**	**0.040155964**	**0.000532004**	0.027471544	**0.001423224**	**0.000721199**
	RMSE	**0.006055055**	**0.083706312**	**0.001505924**	**0.081227229**	**0.005090348**	0.032243785	**0.018297152**	**0.001002998**

Next for the PDR and PLR, the FeD-TST provides better forecasting for the first network configuration (i.e., *Mobile_diff_channel_0dbm*). The reason is that PDR and PLR can be severely affected by the level of the interference. Thus, when using different frequency channels, the likelihood of interference between the applications is reduced, leading to more reliable transmissions and improved forecast accuracy. In contrast, when the interference becomes more important the PDR and PLR change more significantly, affecting the overall accuracy of the forecast. The same observation is drawn when the transmission power increases, which also leads to higher interference and the forecasting accuracy decreases for such configurations.

Lastly for the latency, FeD-TST achieves the best forecasting for the third network configuration (i.e., *Mobile_same_channel_0dbm*). In the particular configuration, we expect to see an increased level of interference, which leads to more re transmissions and thus longer delays. However, the proposed model managed to learn such complex behaviors providing good forecasting accuracy.

4.7.3 Communication Cost

In terms of communications cost, Figure 4.6 provides the comparison of the proposed FeD-TST with the rest of the benchmarks, when varying the fraction of the dataset used by the training process, from 10 to 100%. It should be noted, that the particular Figure illustrates the communication of only one network configuration (i.e., *Mobile_same_channel_0dbm*), due to space limitations. Nonetheless, similar results were obtained for the rest of the configurations as well.

The key observation of the communication cost, is that for all federated learning based methods, such as *FL+LSTM*, *FL+MHA* and our proposed *FeD-TST*, the cost remains constant for all different fractions of the dataset used for the training. This is attributed to the fact that there is no actual datasets transferred between the clients and server. Instead, only the updates of the global and local agents are exchanged, resulting in a consistent and constant communication cost. However, the communication costs of the centralized models increase with the fraction of the dataset used in the training process. Among the centralized models, TCN+BiLSTM shows the highest communication cost, as it has the highest number of trainable parameters compared to the other models. Finally, the proposed FeD-TST gives the least communication cost of $8.2056E + 08$, compared to the other FL approaches, as FL+MHA yields a communication cost of $9.43992E + 08$ and FL+LSTM of $9.81624E + 08$.

4.7.4 Univariate Results

In Table 4.3, the three error metrics i.e., MSE, MAE and RMSE, when forecasting the throughput for all methods and network configurations, are presented. From these results, the following observations can be drawn. Firstly, the proposed FeD-TST model outperforms the other two distributed solutions (i.e., FL+LSTM and FL+MSA) in almost all network configurations. There are two main reasons for this efficiency: i) FL+LSTM can suffer from the vanishing gradients when processing long sequences, which can make it difficult to learn long-term dependencies. However, FeD-TST can avoid this problem by using the attention module, which is used to

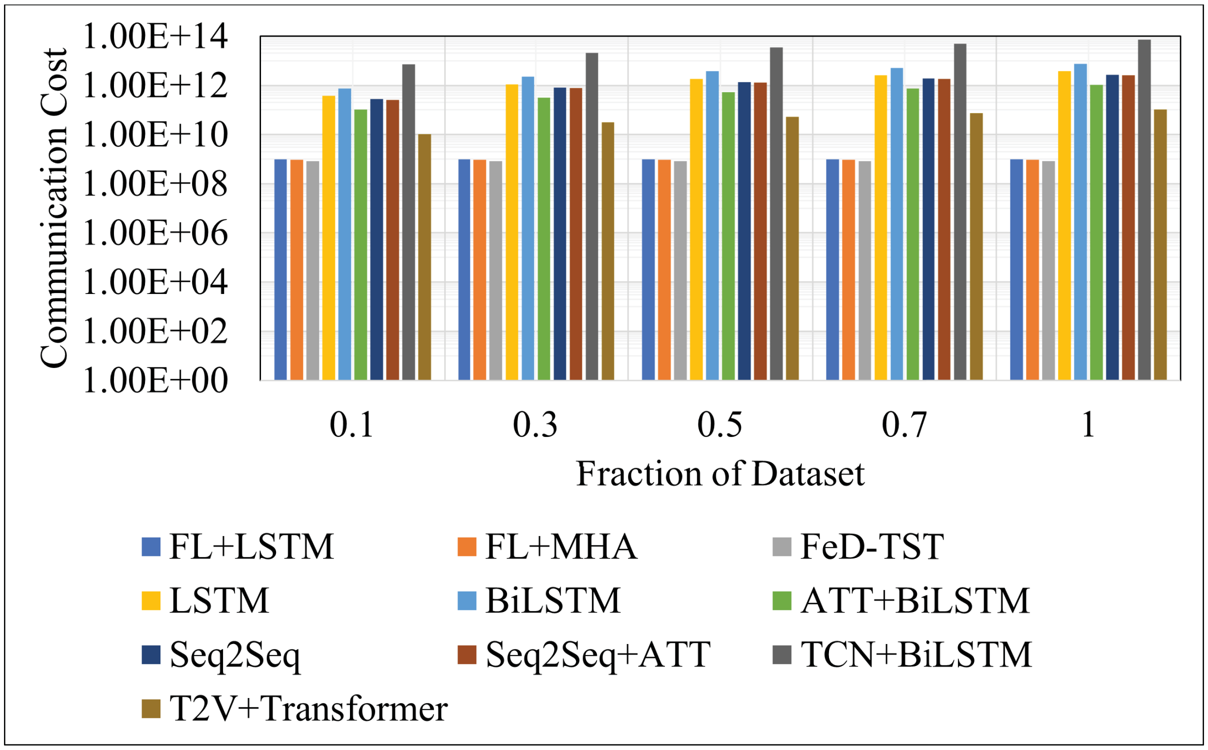


Figure 4.6 Communication Cost

capture the dependencies between different throughput values in the input sequence. This allows the FeD-TST model to learn which parts of the input sequence are more relevant for the throughput forecasting at each time step. By propagating this information across the entire sequence using the attention module, FeD-TST avoids the problem of vanishing gradients. Furthermore, the attention module in FeD-TST is more effective than the gating mechanism in FL+LSTM in terms of capturing long-term dependencies and patterns in the data, especially for longer throughput input sequences. Moreover, FeD-TST uses the positional encoding mechanism which encodes the position of each throughput value in the input sequence. This empowers the FeD-TST to differentiate between various throughput observations in the sequence, even if they have the same value and better handle the longer throughput sequences; ii) FL+MSA uses a self-attention module, which computes the importance of each throughput value in the input sequence based on its relationship with all the other throughput values in the sequence. This means that the self-attention considers all throughput values in the sequence when computing the attention weights. Thus, this can lead the self-attention to be affected by noisy or irrelevant

Table 4.4 Univariate forecasting results for PDR, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.018257154	0.037851512	0.000608792	0.001952939	0.001112278	0.01857656	0.076177719	0.001061075
	MAE	0.080767698	0.156077519	0.001516378	0.003028177	0.001499788	0.080090836	0.245758619	0.003359817
	RMSE	0.135119038	0.194554657	0.024673704	0.044192069	0.033350831	0.136295855	0.276003114	0.032574153
BiLSTM	MSE	0.018449913	0.025352443	0.001411486	0.001915232	0.000599673	0.019455916	0.074454278	0.002206909
	MAE	0.080295092	0.12848674	0.001740071	0.003789981	0.001141312	0.07880042	0.241632533	0.008116467
	RMSE	0.135830455	0.159224505	0.037569752	0.04376336	0.024488231	0.139484464	0.272863113	0.046977751
ATT+BiLSTM	MSE	0.018410313	0.024726183	0.00088561	0.001182298	0.000512866	0.018651737	0.076006	0.000577173
	MAE	0.080084426	0.125214189	0.001705488	0.003541321	0.001272777	0.079920708	0.245711249	0.002313198
	RMSE	0.135684608	0.157245613	0.029759198	0.03438456	0.02264655	0.136571364	0.275691856	0.024024419
Seq2Seq	MSE	0.018686683	0.02546426	0.000627586	0.002250746	0.000353211	0.019352935	0.074024439	0.000891181
	MAE	0.084082524	0.128587491	0.001882692	0.004034477	0.002321665	0.080620867	0.247234381	0.002561207
	RMSE	0.136699243	0.159575247	0.025051661	0.047442024	0.018793907	0.139114827	0.272074326	0.029852654
Seq2Seq+ATT	MSE	0.017895584	0.02488008	0.000994254	0.002043545	0.000256814	**0.018469424**	0.072522882	0.000677193
	MAE	0.079086621	0.125329378	0.001816939	0.004525415	0.001136588	**0.078296446**	0.235733602	0.002153011
	RMSE	0.133774376	0.157734206	0.031531789	0.045205586	0.016025408	**0.13590226**	0.269300728	0.026022931
TCN+BiLSTM	MSE	0.018551141	0.024655612	0.000600939	0.001797786	0.000350983	0.019236453	0.074489966	0.001268126
	MAE	0.082181067	0.125796429	0.00392316	0.003801887	0.001281348	0.081557265	0.2375388	0.007112032
	RMSE	0.136202572	0.157021055	0.024514057	0.04240031	0.01873454	0.13869554	0.272928499	0.035610752
T2V+Transformer	MSE	0.018813977	0.02562528	0.000637369	0.001975316	0.016922135	0.018765976	0.074159335	0.001057309
	MAE	0.086577267	0.130660705	0.001173224	0.005953995	0.055162054	0.079134315	0.243209761	0.006711487
	RMSE	0.137164053	0.16007898	0.025246166	0.044444524	0.130085105	0.136988963	0.272322117	0.032516286
FL+LSTM	MSE	0.019515449	0.098753192	0.000603801	0.135606289	0.001285934	0.025663415	0.107867897	0.000866407
	MAE	0.086151034	0.269703567	0.001251696	0.366929799	0.001705134	0.085382722	0.23624748	0.008104443
	RMSE	0.139641225	0.314214915	0.024572313	0.368222237	0.035859846	0.160190925	0.32837072	0.029434759
FL+MSA	MSE	0.02645552	0.048118532	0.006903093	0.022831427	0.003011749	0.020869514	0.095742323	0.001933012
	MAE	0.122321516	0.17536521	0.071824536	0.118473649	0.008171082	0.091781527	0.254539758	0.009358659
	RMSE	0.162651524	0.219359368	0.083084859	0.151100725	0.054879408	0.144462854	0.309422553	0.043966036
FeD-TST	MSE	**07.52E-13**	**0.024042883**	**4.86E-11**	**0.000736637**	**8.88E-11**	0.023981718	**5.03E-10**	**0.000481912**
	MAE	**6.82E-07**	**0.123524003**	**6.07E-06**	**0.002928413**	**9.01E-06**	0.089999422	**1.95E-05**	**0.002050021**
	RMSE	**8.67E-07**	**0.155057675**	**6.97E-06**	**0.027141048**	**9.42E-06**	0.154860318	**2.24E-05**	**0.021952493**

throughput values in the sequence, even if they are not relevant for the forecasting. In contrast, the sparse attention in FeD-TST only considers a subset of the throughput values in the input sequence when computing the attention weights. This subset of values is typically chosen based on their relevance for the forecasting, which means that noisy or irrelevant QoS values in the sequence may be ignored by the model making FeD-TST more robust than FL-MSA.

The second observation extracted from Table 4.3 is that the FeD-TST model also outperforms all centralized solutions and for almost all network configurations. The reason is that the FeD-TST enables the model to learn from multiple clients with potentially different data distributions. This can help our model to better adapt to new data and be more robust to changes in the data distribution over time. Furthermore, FeD-TST can leverage the collective knowledge of multiple clients at the same time, in order to improve its performance on throughput forecasting.

The final observation, is that for the network configuration 6, the Seq2Seq+ATT model performed slightly better than FeD-TST. This can attributed to the fact that the dataset generated for configuration 6 may not have been diverse enough in terms of the throughput observations. This can adversely impact the distributed learning, since FL models are trained on data that are distributed across multiple clients, which can help improve generalization. However, if the data distributions across the clients are too similar, the FL model may not be able to effectively capture the underlying patterns in the data. Finally, the Seq2Seq+ATT model is a data-efficient model and it can achieve good performance even with fewer homogeneous training samples.

Table 4.5 Univariate forecasting results for PLR, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.021792671	0.033600801	0.000170506	0.003800143	0.000898948	0.006792275	0.028396553	0.004032325
	MAE	0.082623192	0.150432311	0.007338254	0.023558733	0.013402833	0.030336409	0.13075646	0.032339007
	RMSE	0.14762341	0.183305213	0.013057785	0.061645296	0.029982454	0.082415258	0.168512768	0.063500594
BiLSTM	MSE	0.020697191	0.034122424	0.000401793	0.004283569	0.000542852	0.006855659	0.041219532	0.003004725
	MAE	0.080600034	0.151636813	0.008385715	0.024967082	0.012687792	0.027902375	0.158798501	0.031972674
	RMSE	0.143865183	0.184722559	0.020044774	0.06544898	0.023299188	0.08279891	0.203025937	0.054815371
ATT+BiLSTM	MSE	0.020245053	0.033294998	0.00014359	0.00421118	0.000547313	0.006935762	0.028205546	0.003443814
	MAE	0.080818832	0.151242344	0.007237762	0.023958626	0.013155007	0.028010578	0.132771995	0.032751093
	RMSE	0.142285113	0.182469171	0.011982922	0.064893606	0.023394728	0.083281221	0.167945069	0.058684019
Seq2Seq	MSE	0.021252736	0.033916195	0.000523421	0.003230895	0.000885887	0.00682344	0.027612433	0.002580039
	MAE	0.086687473	0.153022372	0.007574055	0.022760354	0.013292484	0.028000094	0.131795246	0.032047241
	RMSE	0.145783182	0.184163502	0.022878392	0.056840967	0.029763853	0.082604119	0.166169893	0.050794084
Seq2Seq+ATT	MSE	0.020900074	0.032672588	0.000460761	0.003551169	0.000549879	0.006830279	0.028939158	0.00355437
	MAE	0.085837735	0.148762972	0.009517111	0.022397076	0.012634716	0.032682136	0.131375186	0.032834303
	RMSE	0.14456858	0.180755604	0.021465339	0.059591688	0.023449497	0.0826455	0.170115131	0.059618536
TCN+BiLSTM	MSE	0.020198194	**0.031638767**	0.000394537	**0.002814738**	0.000232801	0.006485215	0.028564665	0.003535622
	MAE	0.078534621	**0.148136144**	0.008401207	**0.022391848**	0.012239431	0.02950272	0.132377563	0.033134242
	RMSE	0.142120352	**0.177872896**	0.019862957	**0.053054102**	0.015257805	0.080530831	0.169010844	0.0594611
T2V+Transformer	MSE	0.020911002	0.034417311	0.000920884	0.004394198	0.000889586	0.006719895	0.030964824	0.002978452
	MAE	0.085307551	0.153888459	0.009341288	0.02465749	0.013932025	0.031952691	0.137978923	0.030472741
	RMSE	0.14460637	0.18551903	0.030346072	0.066288749	0.029825929	0.081974968	0.175968248	0.054575195
FL+LSTM	MSE	0.022094503	0.041293263	0.046052642	0.004675237	0.810819387	0.006800786	69.65638733	0.003036108
	MAE	0.075020067	0.159862995	0.137425244	0.02439647	0.367824584	0.021475384	1.586332083	0.028901355
	RMSE	0.148626059	0.203149363	0.214597806	0.068371393	0.867065966	0.082466155	8.330111504	0.055099569
FL+MSA	MSE	0.026332522	0.053814277	0.002095943	0.009201947	0.144768745	0.008289964	0.042631045	0.006102875
	MAE	0.09593612	0.185807839	0.013791043	0.049586143	0.285889983	0.040396597	0.162858874	0.05424583
	RMSE	0.16227299	0.231979042	0.045781475	0.095926777	0.380484879	0.091049239	0.206472874	0.078120902
FeD-TST	MSE	**2.66E-14**	0.038911376	**1.91E-11**	0.008929118	**3.05E-05**	**3.67E-05**	**0.026158862**	**2.60E-11**
	MAE	**1.37E-07**	0.16107823	**3.48E-06**	0.070162557	**0.00051405**	**0.004760905**	**0.127036972**	**3.57E-06**
	RMSE	**1.63E-07**	0.197259665	**4.37E-06**	0.094494008	**0.005524271**	**0.006055055**	**0.161737015**	**5.10E-06**

Similarly, Table 4.4 presents the error metric results for the PDR under all network configurations. As it can be seen, the proposed FeD-TST model performs better than all other models for all configurations except for the network configuration 6 (i.e., static robots, different channel allocation, and 12*dBm* transmission power). Once more and for the reasons described above the Seq2Seq+ATT model exhibited the best performance. This means that the Seq2Seq+ATT model is able to well capture the temporal patterns that affect the PDR dataset, such as changes in network congestion or interference, and make more accurate future forecasts.

Regarding PLR, and as seen in Table 4.5, FeD-TST provides the best results for 6 configurations. Specifically, for the second (i.e., mobile robots, different channel, $12dbm$) and fourth network configuration (i.e., mobile robots, same channel, $12dbm$), TCN+BiLSTM showcased the best performance. In both configurations the access points are mobile and the transmission power is high. Nonetheless, from the same Table and for the first (mobile robots, different channel, $0dbm$) and third configuration (mobile robots, same channel, $0dbm$), it can be seen that FeD-TST performed better. By looking close into these configurations, we can see that the transmission power may have an impact on how the learning models perform.

Specifically, when a network configuration contains mobile access points the network topology and data distribution can change over time. This means that the data may not be as sparse, as all access points may have relevant information at different times. Also the sparsity of PLR datasets can be influenced by the transmission power. When the transmission power is set to 0dBm, the signals do not travel far, resulting in a shorter propagation range, and higher PLR. Since the FeD-TST model can handle large amounts of data more efficiently, the model may have more information to work with, when there is a noticeable amount of PLR, allowing it to attend the subset of important PLR values using the sparse attention. Therefore, it can perform well on this type of PLR datasets. However, when the transmission power is set to 12dBm, the signals can travel further, resulting in a more sparse PLR dataset. Hence, TCN+BiLSTM, which uses convolutional layers to extract the features from the PLR input data, can capture well local patterns, while the BiLSTM layer can effectively capture the longer-term temporal relationships between the PLR values. Overall, TCN+BiLSTM performs better than the FeD-TST model in scenarios where the dataset is more sparse.

For the latency dataset and as seen from Table 4.6, the FeD-TST is well suited for all network configurations except configuration 8 (static robots, same channel, $12dBm$), where the ATT+BiLSTM model gives the best performance. The reason is that when multiple robots are

Table 4.6 Univariate forecasting results for Latency, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.024408949	0.06017804	0.088636216	0.056182637	0.017420559	0.009039901	0.100882114	0.040418583
	MAE	0.080140304	0.214223589	0.248948097	0.201964768	0.049288355	0.033133292	0.291552399	0.167005577
	RMSE	0.156233636	0.245312127	0.297718349	0.237028769	0.131986964	0.095078395	0.317619448	0.201043734
BiLSTM	MSE	0.024206015	0.0603612	0.088730471	0.084598273	0.01688086	0.009230823	0.101561174	0.041677659
	MAE	0.079947131	0.214269641	0.249370197	0.198644096	0.047570767	0.033800875	0.291890608	0.165094546
	RMSE	0.155582824	0.245685164	0.297876604	0.246165022	0.129926363	0.096077172	0.31868664	0.204151069
ATT+BiLSTM	MSE	0.024179881	0.084598273	0.089001176	0.057675521	0.01729584	0.009121923	0.102110922	**0.040280716**
	MAE	0.080108745	0.246165022	0.240791322	0.203930682	0.046483801	0.03072686	0.293887956	**0.14324601**
	RMSE	0.155498813	0.290857822	0.298330648	0.240157283	0.131513648	0.095508757	0.319547997	**0.200700563**
Seq2Seq	MSE	0.024997304	0.05980379	0.089389425	0.055409534	0.016375065	0.008770341	0.099581648	0.040978067
	MAE	0.078904381	0.214706046	0.256589036	0.201288012	0.050110525	0.033967249	0.281536809	0.168885758
	RMSE	0.158105358	0.244548135	0.298980642	0.235392298	0.127965093	0.093650097	0.315565599	0.202430399
Seq2Seq+ATT	MSE	0.024013291	0.60308408	0.086830182	0.057301211	0.017024882	0.008396982	0.103578993	0.041083115
	MAE	0.081703762	0.21346644	0.254700822	0.202772912	0.049581186	0.035931874	0.295579004	0.170103562
	RMSE	0.154962224	0.245577704	0.294669615	0.239376713	0.130479431	0.091635047	0.321836904	0.2026897
TCN+BiLSTM	MSE	0.032147098	0.062922512	0.09267886	0.054981224	0.016755645	0.009074548	0.103784816	0.04146944
	MAE	0.11613974	0.213559675	0.255984481	0.199181873	0.046739142	0.034320761	0.29798527	0.16241555
	RMSE	0.179296121	0.250843602	0.304432029	0.234480753	0.129443598	0.095260424	0.322156508	0.203640468
T2V+Transformer	MSE	0.024686465	0.064907747	0.09201476	0.055858478	0.016280688	0.008851462	0.103110644	0.040447623
	MAE	0.082032614	0.222181234	0.266272543	0.199272798	0.052664447	0.031812765	0.298888429	0.166716834
	RMSE	0.15711927	0.254769988	0.303339347	0.236343982	0.1275958	0.094082209	0.321108461	0.201115943
FL+LSTM	MSE	0.027547928	80687.98438	0.103246123	4.166488647	0.019056881	0.009888737	0.121394642	0.047525864
	MAE	0.077818892	18.25476456	0.230045661	0.455275744	0.053534083	0.036810573	0.274667084	0.170613229
	RMSE	0.165957242	283.6138611	0.32129097	2.016258478	0.138044104	0.099438779	0.348386973	0.217998505
FL+MSA	MSE	0.024906835	0.115047574	0.245384723	0.098320305	0.023086162	0.01547756	0.136887431	0.07021708
	MAE	0.077157654	0.27891928	0.433125585	0.257558107	0.077514783	0.060696498	0.318601936	0.196365908
	RMSE	0.157818988	0.339186639	0.495363235	0.313560694	0.151941314	0.124408841	0.369983017	0.264985055
FeD-TST	MSE	**0.023777915**	**0.059448318**	**4.40E-05**	**0.054488377**	**0.001039662**	**7.52E-05**	**0.002381638**	0.066853426
	MAE	**0.076379195**	**0.212089206**	**0.006626479**	**0.198644096**	**0.027471544**	**0.007280115**	**0.04880077**	0.190088391
	RMSE	**0.15420089**	**0.243820257**	**0.006636661**	**0.233427456**	**0.032243785**	**0.00867211**	**0.048802029**	0.2585603

operating on the same channel, they can interfere with each other's signals. This becomes more evident when the robots are static and they use high transmission power, increasing this way the common serving area, which however will suffer from more dense interference levels. Inevitably, this high interference will lead to a latency increase. Therefore, forecasting the latency in such a network configuration is more challenging for federated learning models.

4.7.5 Multivariate Results

Table 4.7 presents the multivariate results for the throughput prediction for different network configurations. As in the case of univariate prediction, our proposed model, Fed-TST outperformed all of the rest methods and for almost all network configurations. Additionally, the prediction errors are reduced compared to the univariate setting, since in this second set of results, the FeD-TST model does not only capture the temporal dependencies of the forecasting variable e.g., throughput values but the temporal dependencies of all the other features at the same time (i.e.,

Table 4.7 Multivariate forecasting results for Throughput, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.028607518	0.098101639	0.013346768	0.06513989	0.027364537	0.009801793	0.019369793	0.009325607
	MAE	0.132310821	0.306853361	0.09260656	0.195485626	0.094224957	0.0679559	0.135783555	0.077761354
	RMSE	0.169137572	0.313211812	0.115528215	0.255225175	0.1654223	0.099004003	0.139175403	0.096569184
BiLSTM	MSE	0.02742663	0.101624222	0.013520551	0.070678015	0.027028323	0.00956266	0.011623326	0.009524192
	MAE	0.129706767	0.312485707	0.093181167	0.211890758	0.085585064	0.065992332	0.103169602	0.0786563
	RMSE	0.165609873	0.318785543	0.116277903	0.26585337	0.16440293	0.097788856	0.107811532	0.097591965
ATT+BiLSTM	MSE	0.029332515	0.037467551	0.012400092	0.0409477	0.027625441	0.00918847	0.02887653	0.008108388
	MAE	0.134291403	0.192225229	0.089390103	0.133535994	0.054213672	0.063080543	0.165644779	0.07195941
	RMSE	0.171267378	0.193565365	0.111355702	0.202355382	0.166209028	0.095856508	0.169930955	0.090046588
Seq2Seq	MSE	0.029094397	0.222091576	0.013109092	0.078138339	0.087997502	0.075132291	0.002492687	0.007884014
	MAE	0.133091282	0.469024161	0.091876588	0.265520788	0.2902785	0.216263738	0.043581509	0.070501437
	RMSE	0.170570797	0.471265929	0.114494943	0.279532357	0.296643729	0.274102701	0.049926819	0.08879197
Seq2Seq+ATT	MSE	0.029802806	0.197385415	0.013370054	0.076762567	0.09746884	0.072463298	**0.001641729**	0.008784528
	MAE	0.134996629	0.440116966	0.092956957	0.250004146	0.307147377	0.210497291	**0.033518686**	0.075263339
	RMSE	0.172634893	0.444280784	0.115628948	0.277060583	0.3122	0.269190078	**0.040518256**	0.093725811
TCN+BiLSTM	MSE	0.028206558	0.049226831	0.013811762	0.05078849	0.032273292	0.011554496	0.004679834	0.008682814
	MAE	0.131378731	0.219623558	0.093851078	0.145438506	0.061250018	0.078781507	0.061909672	0.074983242
	RMSE	0.167948081	0.221871203	0.117523452	0.225363019	0.179647688	0.10749184	0.068409315	0.09318162
FL+LSTM	MSE	0.022230878	0.026630133	0.02434326	0.030587779	0.047000714	0.03989619	0.058205191	0.020379072
	MAE	0.116344981	0.12812157	0.117841065	0.138585195	0.203050449	0.158704802	0.18743296	0.105792277
	RMSE	0.148784027	0.162836522	0.155774087	0.17474249	0.216674656	0.199541911	0.241099924	0.142648384
FL+MSA	MSE	0.055654656	0.219694048	0.123248108	0.834618747	0.290231615	0.079888575	0.062982544	0.022085747
	MAE	0.202963769	0.439056277	0.317067206	0.88365382	0.483849257	0.249765113	0.208570436	0.107883148
	RMSE	0.235912398	0.46871531	0.351067096	0.913574696	0.538731515	0.282645643	0.250963241	0.148612738
FeD-TST	MSE	**2.97E-05**	**0.000250346**	**0.002767468**	**1.40E-05**	**0.002059271**	**0.000360866**	0.003959762	**0.00255931**
	MAE	**0.0043**	**0.012030067**	**0.039089505**	**0.003005067**	**0.041864872**	**0.015221067**	0.050758883	**0.034195092**
	RMSE	**0.0054**	**0.015822316**	**0.052606728**	**0.003742877**	**0.045379192**	**0.018996468**	0.062926643	**0.050589621**

throughput, PDR, PLR, latency, time first packet transmitted, time last packet transmitted, total transmitted packets, total received packets). By modeling the temporal dependencies between these multiple features, the FeD-TST is able to better forecast the future forecasting variable e.g., throughput values. Furthermore, the sparse attention mechanism is more useful in multivariate settings, as these settings consist of multiple features which however, may or may not be relevant to the target forecasting variable. Thus, the sparse attention efficiently attends only relevant features and the FeD-TST model learns a more accurate representation of the input data, while providing more accurate forecasts than the other models.

Only for the seventh network configuration (static robots, same channel, $0dbm$), the Seq2Seq+ATT centralized model performed slightly better than the FeD-TST. By further analyzing this behavior, we found that the size of the input sequence had an impact on the final performance of the FeD-TST. In other words, the input sequence length may not provide enough information for the model to effectively capture the temporal relationships among the multiple features, however an increase in the input sequence length may reduce the forecasting error metrics.

Similar observations were drawn for PDR, PLR, and Latency, where the FeD-TST consistently gave the best performance for the large majority of the network configurations. Nonetheless, for page limitations purposes, we have decided to include the Tables of these multivariate QoS forecasting in the Appendix 3 of this document. Overall, FeD-TST managed to find the best accuracy performance with the minimum prediction error in 53 out of 64 univariate and multivariate experiments performed, which proves that it can uniformly outperform the rest of the centralized and distributed approaches in a dynamic and uncertain network environment.

4.8 Conclusion

In this work, we investigated the QoS forecasting problem by formulating it as a univariate and multivariate time series forecasting problem in a federated learning setting. In particular, a new framework, FeD-TST was introduced that promotes an efficient QoS forecasting for a number of heterogeneous network configurations with various IoT applications that stress the IoT access network creating several levels of QoS uncertainty. To evaluate our framework, we firstly generated real-time datasets in a real testbed for eight different network configurations that considered the mobility of robots acting as access points, the frequency channel, and the transmission power allocation as configuration knobs. Following, we presented a novel Federated learning based sparse temporal transformer based architecture (FeD-TST), which learns temporal representations and their long term complex dependencies in a federated fashion, for the forecasting of four important QoS metrics, namely, throughput, PDR, PLR and latency. The FeD-TST allowed each local client to train their own local models on their respective datasets and share only the model weights with a global server. By doing so, each client can benefit from the knowledge gained by their peers and improve their model training, while still preserving the privacy of their data. The TST architecture leverages the sparse attention mechanism, which restricts the attention computation to a subset of input elements that are deemed important for the output, resulting in a reduced time complexity. Finally, we performed an extensive experimental evaluation in which we proved that our proposed FeD-TST achieves superior performance for most of the network configurations for both univariate and multivariate settings, as compared with several competitive benchmark methods.

As a future work, we aim to explore alternative attention techniques, such as compressed attention and investigate their impact on the accuracy achieved. Furthermore, we would like to investigate the aggregation strategies other than the FeDAvg at the server side to improve future forecasts of several key QoS metrics.

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusions

The IoT is a network of interconnected physical devices, vehicles, appliances, and other items that collect and exchange data over the internet, boasting the ability to interact with their environment, communicate with each other, and often operate autonomously. This network plays an essential role in the modern digitization of life, dramatically enhancing the public's daily experiences by providing a system of interrelated computing, sensing, and communication. However, the exponential growth and diversity of IoT devices also bring several challenges, including resource limitations, the demand for real-time data processing, network connectivity, and data transmission efficiency. EC emerges as a potential solution to these challenges by bringing computational and communication resources closer to the network's edge, thereby reducing latency and enabling efficient data processing. Despite these benefits, the unpredictable data generation from IoT devices and potential interference in wireless communications pose new challenges. Furthermore, the varying QoS requirements due to IoT device mobility and the necessity of predicting the time-varying characteristics of IoT devices further reinforce an uncertain communication. Therefore, this thesis revolved around solving these challenges and made concrete contributions in three different aspects.

As a first step the challenge of classifying IoT devices based on their traffic characteristics was presented in Chapter 2. For this, we proposed a composite learning framework that consisted of two stages. Initially, the network traces were passed to an initial data preprocessing module. Following preprocessing, the traces were given as input to Stage 0, where a feature selection mechanism and a Logistic Regression classifier were employed. The feature selection process utilized an ANOVA filter-based technique to identify the most relevant features for the Stage 0 classifier. The tentative classification results from Stage 0, along with the remaining features, were then forwarded to the Stage 1 classifier. The Stage 1 classifier utilized an optimal multi-layer perceptron neural network architecture to perform the final classification of the IoT traffic. This architecture was carefully designed to provide accurate and efficient classification results. To evaluate the performance of the proposed framework, we conducted detailed experiments and comparisons. We used two different IoT datasets specifically created for a smart city scenario. These datasets are utilized to assess the effectiveness of the composite architectures in comparison to other existing approaches. The experimental results demonstrated that the proposed framework significantly improved the classification performance across various evaluation metrics. These metrics included recall, precision, F1-score, accuracy, and confusion matrix analysis. Notably, the proposed model achieved a 99.9% accuracy for the first dataset and a 99.8% accuracy for the second dataset, highlighting the robustness and generalization capabilities of the proposed approach. Overall, the chapter 2 presented a comprehensive analysis of the problem of IoT traffic classification. By introducing a composite learning framework and conducting thorough experiments, we were able to demonstrate the effectiveness and superiority of the proposed approach in accurately classifying the IoT devices based on their network characteristics, thus contributing to advancements in the field of IoT data analysis and classification.

In Chapter 3, we focused on addressing the QoS prediction problem in the context of coexisting and heterogeneous IoT applications that introduced varying levels of QoS uncertainty to the IoT access network. To achieve efficient QoS prediction, we formulated the problem as a univariate and multivariate time series forecasting task. To support our investigation, we generated five distinct real-time datasets that represented different IoT applications such as HVAC, lighting, VoIP, surveillance, and emergency response. These datasets served as the basis for evaluating the performance of our proposed framework. We introduced a novel transformer-based architecture specifically designed for QoS prediction. This architecture leveraged the power of attention mechanisms to capture temporal representations and intricate dependencies over long-term sequences. Four important QoS metrics, namely throughput, PDR, PLR, and latency were predicted, while through an extensive experimental evaluation, it was demonstrated that our proposed temporal transformer outperformed several competitive time series baseline methods. The superiority of our approach was evident across all five IoT applications and for both univariate and multivariate settings. In conclusion, the proposed method contributed a new framework for efficient QoS prediction in the presence of diverse IoT applications. The transformer-based architecture effectively modeled the temporal aspects and complex dependencies of the time series data, resulting in superior performance compared to existing methods. This work opens up possibilities for improved QoS management and resource allocation in IoT networks, ultimately enhancing the reliability and performance of IoT applications.

Finally, in Chapter 4, we extended the QoS prediction problem solved in Chapter 3 by taking into consideration the mobility of devices as well in a distributed setting. For this, we formulated the problem as a time series forecasting (TSF) task, considering the complexities introduced by heterogeneous network configurations and various IoT applications that impose different levels of QoS uncertainty on the IoT access network. Firstly, real-time datasets in a real testbed were generated, encompassing eight different network configurations. These configurations incorporated factors such as the mobility of access points, frequency channel allocation, and transmission power allocation as parameters and these datasets served as the basis for evaluating the performance of our proposed framework. We introduced a novel architecture called FeD-TST specifically designed for QoS forecasting. FeD-TST allowed each local client to train their own models on their respective datasets and share only the model weights with a global server. The TST architecture, within FeD-TST, leveraged a sparse attention mechanism that reduced time complexity by focusing attention on a subset of input elements deemed important for the output. Through an extensive experimental evaluation, we demonstrated the superior performance of our proposed model across various network configurations, both in univariate and multivariate settings. In conclusion, Chapter 4 contributed a new framework, FeD-TST, for efficient QoS forecasting in the context of federated learning.

5.2 Future Work

The future work lies in two main research direction as follows:

5.2.1 Extending IoT device classification

In our current work, a large-scale IoT traffic analysis and identification was presented that significantly enhanced the practicality of real-world IoT device identification and classification. One possible direction can be considered as the integration of our IoT device classification framework with a dynamic resource allocation mechanism. In this way, our framework can accurately classify IoT traffic and predict traffic profiles. Thus, it can provide valuable insights into the anticipated resource needs of different devices and services on the network. For example, devices or services classified as high-priority or high-bandwidth may require more resources compared to others. Furthermore, the dynamic resource allocation mechanism could leverage this classification data to make informed decisions about resource distribution. For instance, it could allocate more bandwidth to devices or services predicted to experience high traffic in the near future, or it could conserve energy on devices that are predicted to be idle. These allocations could also be continuously adjusted as new classification and prediction data comes in. Ultimately, this integration could lead to a more intelligent, adaptive, and efficient IoT network. It could help ensure that resources are not only used optimally but are also preemptively allocated in anticipation of future demand, thereby improving the overall performance of the IoT network.

Another possible future direction of this work is the exploration of unsupervised learning methods, particularly used for dealing with new and unidentified types of IoT devices. When the new IoT devices connect to the network, they may exhibit unique traffic patterns however, our model, trained only on known devices, might not be able to classify them accurately. For example, if a new type of IoT device is introduced to the network, and this device generates traffic patterns that are substantially different from the ones the model was initially trained on. The clustering algorithm e.g., K-means could recognize this as a new, distinct group of data points and create a new cluster for these devices. This new cluster, in turn, would represent a new class in our classification model, thereby enhancing the model's ability to recognize and classify this new type of device in the future.

5.2.2 Extending QoS Prediction using Federated Learning

While our FeD-TST approach has shown promising results, there is room to further refine and optimize the federated learning process. This might include developing techniques to handle non-IID data distributions among clients or exploring the alternative aggregation techniques other than the FeDAvg to capture the knowledge gained across different local models. Another possible direction is to explore methods that can further enhance privacy and security in FeD-TST. The promising research methods in this context are advanced cryptography methods such as Homomorphic Encryption (HE) or Secure Multi-Party Computation (SMPC) and differential privacy techniques, which mathematically quantify the privacy leakage of a system. Another possible future direction is an integration of other advanced machine learning methods, such as Reinforcement Learning (RL), where a RL agent could be trained to optimize the prediction of QoS metrics over time and to dynamically adjust its strategies based on the rewards (accuracy of prediction) it receives. This approach could allow the model to make more sophisticated predictions, while taking into account the changing network conditions.

APPENDIX I

APPENDIX OF CHAPTER 4

1. Algorithms and Computational Analysis

In this appendix we decompose the FeD-TST on its algorithmic implementation and provide its complexity analysis. Four algorithms in total have been designed each representing a specific module of the FeD-TST model. Algorithm I-1 executes the sparse attention module. The algorithm begins by splitting the query, key, and value vectors into h_n separate heads (lines 2-4). Next, it computes the dot product of q and k and applies an attention mask (line 5-7) to obtain a masked similarity matrix M. The attention mask can be different depending on the attention mode chosen. For instance, if the attention mode is casual, the mask will prevent the attention to future positions. The masked similarity matrix M is then normalized using a softmax function to obtain the attention weights (line 8). These attention weights are then used to compute a weighted sum of the value vectors (line 9). Finally, the resulting score vectors from all attention heads are merged to obtain the output SA_scores (line 10).

Algorithm-A I-1 Sparse Attention Algorithm

```
   Input: $\mathcal{Q}, \mathcal{K}, \mathcal{V}, h_n, att$ // $\mathcal{Q}, \mathcal{K}, \mathcal{V}$, is the query, key and value vectors for the $l^{th}$ client
          at time $T$, $h_n$ denotes the number of heads and $att = strided$ is the attention
          mode for the training.
   Output: $SA\_scores$
1  SPARSE_MODULE($\mathcal{Q}, \mathcal{K}, \mathcal{V}, h_n, att$)
2  set $q \leftarrow Split\_heads(\mathcal{Q}, h_n)$
3  set $k \leftarrow Split\_heads(\mathcal{K}, h_n)$
4  set $v \leftarrow Split\_heads(\mathcal{V}, h_n)$
5  set $w \leftarrow MUL(q, k)$
6  set $mask \leftarrow get\_attn\_mask(timesteps, att = strided)$
7  set $M \leftarrow masking(mask, w)$
8  set $Weight \leftarrow Softmax(M)$
9  set $SA \leftarrow MUL(Weight, v)$
10 set $SA\_scores \leftarrow Merge\_heads(SA)$
```

The second algorithm (Algorithm I-2) implements four modules of the TST, namely the INPUT MODULE, ENCODER MODULE, DECODER MODULE, and OUTPUT MODULE. The INPUT MODULE takes as input the training dataset instances D_l^T for the l^{th} client at time T, the input sequence length (len), and the dimension representation for the training (dim). It then creates the input layer by applying an embedding layer to D_l^T. Then the positional encoding (PE) is generated for the input sequence. Finally, to obtain the final input representation, it adds the input layer with the positional encoding (line 2-4). The output of this module is passed to the ENCODER MODULE.

Specifically, the ENCODER MODULE takes as an input the output from the INPUT MODULE i.e., $input\_res$, an epsilon value ϵ, the number of heads h_n, the attention mode as local or strided att, and the dropout rate dp. Firstly, the input representation is normalized through the $\mathcal{LN}$ function. Following, the space attention, as discussed in Algorithm I-1, is applied to the input representation to obtain the encoder output. Then, the dropout operation is performed using the $\mathcal{DP}$ layer to the encoder output (line 6-8). Finally, the encoder output and input representation are added to obtain the residual connection, which is normalized using the $\mathcal{LN}$ function (line 9-10).

The DECODER MODULE takes as input the output from the INPUT MODULE i.e., $input\_res$, the output from the ENCODER MODULE i.e., enc_res, an epsilon value ϵ, the number of heads h_n, the attention mode as local or strided att, the dropout rate dp, the number of filters (fil), the kernel size δ, and the activation function σ. First, the input representation is normalized and a masked SPARSE attention is performed to it in order to obtain the decoder's output, which is then applied to the dropout function (line 12-14). The residual connection output is obtained by adding and normalizing the decoder's output and the input representation (line 15-16). After this, the SPARSE attention is applied again to the encoder output and the residual connection output in order to obtain the encoder-decoder representation, which is later applied to a dropout function (line 17-18). The next step involves the obtaining of the residual connection, which is then normalized (line 19-20). Lastly, the 1D convolutional layer to the residual connection

Algorithm-A I-2 Temporal Sparse Transformer Algorithm

```
Input: $\{D_l^T, len, dim, input\_res, \epsilon, h_n, att, dp, enc\_res, fil, \delta, act, x, decoder\_res\}$
// $D_l^T$ is the training dataset instances for the $l^{th}$ client at time $T$, $len$ denotes the input sequence length, $dim$ is the dimension representation for the training, $input\_res$ is the output of INPUT MODULE, $\epsilon$ is the epsilon value, $h_n$ is the number of heads, $att$ is the attention mode as local or strided, $dp$ is the dropout rate, $enc\_res$ is the output of ENCODER MODULE, $fil$ is the number of filters, $\delta$ is the kernel size, $act$ is the activation function, $x$ is the output of Module 3 and $decoder\_res$ is the results of $\mathcal{SA}$ module within the Module 3.
Output: $x$ // $x$ is the final predicted value of QoS from the Module 4.
1  INPUT_MODULE($D_l^T, len, dim$)
2  set $TST_{input} \leftarrow Input\_Layer(D_l^T)$
3  set $PE \leftarrow Positional\_Encoding(len, dim)$
4  set $input\_res \leftarrow Add(TST_{input}, PE)$                    ▷ Module 1
5  ENCODER_MODULE ($input\_res, \epsilon, h_n, att, dp$)
6  set $x \leftarrow \mathcal{LN}(input\_res, \epsilon)$
7  set $x \leftarrow SPARSE(x, h_n, att)$
8  set $x \leftarrow \mathcal{DP}(dp, x)$
9  set $enc\_res \leftarrow x + input\_res$                ▷ $\mathcal{SA}$
10 set $enc\_res1 \leftarrow \mathcal{LN}(enc\_res)$                       ▷ Module 2
11 DECODER_MODULE ($input\_res, enc\_res, \epsilon, h_n, att, dp, fil, \delta, act$)
12 set $x \leftarrow \mathcal{LN}(input\_res, eps)$
13 set $x \leftarrow SPARSE(x, h_n, att)$
14 set $x \leftarrow \mathcal{DP}(dp, x)$
15 set $dec\_res \leftarrow x + input\_res$
16 set $dec\_res1 \leftarrow \mathcal{LN}(dec\_res)$
17 set $x \leftarrow SPARSE(enc\_res, dec\_res1, dec\_rec1, h_n, att)$
18 set $x \leftarrow \mathcal{DP}(dp, x)$
19 set $decoder\_res \leftarrow x + input\_res$            ▷ $\mathcal{SA}$
20 set $decoder\_res1 \leftarrow \mathcal{LN}(decoder\_res)$
21 set $x \leftarrow layer\_Conv1D(fil, \delta, \sigma, decoder\_res1)$
22 set $x \leftarrow \mathcal{DP}(dp, x)$                 ▷ Module 3
23 OUTPUT_MODULE($x, decoder\_res$)
24 set $x \leftarrow layer\_GlobalAvgPooling1D(x)$
25 set $x \leftarrow layer\_Dense(x)$
26 set $x \leftarrow Add(x, decoder\_res)$
27 set $x \leftarrow layer\_Norm(x)$                           ▷ Module 4
```

output is applied along with the dropout operation to obtain the final decoder output (lines 21-22).

Finally, the OUTPUT MODULE applies a global average pooling to the output of the convolutional layer, followed by a dense layer (lines 24-25). The output of the dense layer is added to the output of the DECODER MODULE, and the result is passed through a normalization layer (lines 26-27), which is the output of the TST algorithm (line 27).

The complete training processes of the FeD-TST framework are formally described in Algorithms I-3 and I-4 that provide the Global Server $\mathcal{GS}$ and Local Client ($\mathcal{LC}$) training algorithms respectively. The inputs for Algorithm I-3 are the number of communication rounds $\mathcal{K}$, the set of local clients $\mathcal{LC}$, where each local client is associated to a temporal instance $t \in T$ over a single network configuration κ, the initial global weight matrix w^0 with any arbitrary values, and the fraction fr of local clients to take part in each communication round. The algorithm starts by initializing the global model weight matrix w_g^k with w^0 and setting the first round to zero (line 1). Following, for each communication round the $\mathcal{GS}$ randomly selects a fraction of local clients belonging to a subset $\Omega \subseteq \mathcal{LC}$ (Line 2). At the same time, the $\mathcal{GS}$ shares the latest global weight matrix w_g^k with all the chosen local clients Ω. We denote $w_{l_\kappa^t}$ as the weights from local clients at time t associated with a specific network configuration κ. Next, each local client trains its own local model in parallel at each network configuration to find the local weight for that communication round (Line 3). The detailed steps of the Local client function are shown in Algorithm I-4.

Specifically, the input of Algorithm I-4 includes a dataset X_κ^t, the local epochs $\mathcal{E}$, the global model weight w_g^k, the learning rate η, the hyper-parameter μ and the global communication round $\mathcal{K}$. First, the $\mathcal{LC}$ updates its local model weights matrix $w_{l_\kappa^t}^k$ with the latest available global model weights matrix w_g^k (Line 1). Afterwards, the $\mathcal{LC}$ trains its $local\_model(w_{l_\kappa^t}^k, X_\kappa^t)$ according to the defined local epochs $\mathcal{E}$ and sequences $\tilde{X}_\kappa^t$ of window size τ (Line 2 - Line 10). We rely on the Stochastic Gradient Descent (SGD) approach to update the local model weight matrix, where $w_{l_\kappa^t}^*$ is the optimized local weight matrix , and ∇ is the gradient of the loss function

Algorithm-A I-3 Server (Global) Training Algorithm

```
Input: $\{\mathcal{GS}, \mathcal{K}, |\mathcal{LC}|, w^0, fr\}$
//$\mathcal{GS}$ is the global server, $\mathcal{K}$ is the total number of communication rounds, $|\mathcal{LC}|$ denotes
the total number of local clients, $w^0$ is the initial global weight matrix and $fr$ is the
fraction of local clients included in each round.
Output: $\{\mathcal{GS}\}$
//$\mathcal{GS}$ is the final global server
1 Initialize $w_g^k = w^0$ and $k = 0$
2 for each communication round $k = \{0, 1, ..., \mathcal{K} - 1\}$ do
3     $\mathcal{GS}$ randomly selects subset $\Omega$ of local clients $\mathcal{LC}$ such that $\Omega \subseteq \mathcal{LC}$ and $|\Omega| = fr$
4     /* $\mathcal{LC}$ performs training in parallel (Algorithm 4)*/
5     $\mathcal{GS}$ receives $w_{l_\kappa^t}^{k+1}$ from all clients $\mathcal{LC}$
6     $\mathcal{GS}$ updates $w_g^{k+1}$ to $\frac{1}{|\mathcal{LC}|} \sum_{n_\kappa^t \in \mathcal{LC}} w_{l_\kappa^t}^{k+1}$          ▷ $FeDAvg$
7     $\mathcal{GS}$ updates global_model $(w_g^{k+1})$
8 end for
```

Algorithm-A I-4 Client (Local) Training Algorithm

```
Input: $\{\mathcal{LC}, X_\kappa^t, \mathcal{E}, w_g^c, \eta, \mu, \mathcal{K}\}$
// $\mathcal{LC}$ is the local client model, $X_\kappa^t$ is the time series for the specific configuration $\kappa$, $\mathcal{E}$ is
the local epochs, $w_g^c$ is the global model weight matrix, $\eta$ is the learning rate, $\mu$ is the
model hyper parameters and $\mathcal{K}$ is the global communication round.
Output: $\{\mathcal{LC}\}$
// $\mathcal{LC}$ is the final local client model
1 Initialize $w_{l_\kappa^t}^k = w_g^k$
2 $\mathcal{LC}$ performs training local_model $(w_{l_\kappa^t}^k, X_\kappa^t)$
3 Timeseries_Sequence ← Split $X_\kappa^t$ into Sequence $\tilde{X}_\kappa^t$ of window size $\tau$
4 for each local epoch $i = \{0, 1, ..., \mathcal{E} - 1\}$ do
5     for each $\tilde{X}_\kappa^t \in X_\kappa^t$ do
6         $w_{l_\kappa^t}^* \leftarrow w_{l_\kappa^t} - \eta \nabla loss(w_{l_\kappa^t}, \tilde{X}_\kappa^t)$
7     end for
8 end for
9 Obtain local weights $w_{l_\kappa^t}^{k+1} \approx argmin_{w_{l_\kappa^t}} f(w_{l_\kappa^t}^*, w_g^k) = \mathcal{LL}(w_{l_\kappa^t}^*) + \lambda$
10 $\mathcal{LC}$ uploads $w_{l_\kappa^t}^{k+1}$ to $\mathcal{GS}$
```

$loss(w_{l_\kappa^t}, \tilde{X}_k^t)$ of window size τ with the learning rate η (Line 6). As a next step, instead of updating the local model weight matrix $w_{l_\kappa^t}^{k+1}$ with the weight matrix $w_{l_\kappa^t}^{*}$ of the minimized local loss function $\mathcal{LL}(w_{l_\kappa^t}^{*})$, the proximal term λ is integrated into $\mathcal{LL}(w_{l_\kappa^t}^{*})$ to limit the effect of local model updates on the overall global model. The local model weights matrix $w_{l_\kappa^t}^{k+1}$ is then updated with the minimum weights matrix: $w_{l_\kappa^t}^{k+1} \approx argmin_{w_{l_\kappa^t}} f(w_{l_\kappa^t}^{*}, w_g^k)$ (Line 9). Finally, the $\mathcal{LC}$ sends the local model weight matrix back to the Global server $(\mathcal{GS})$ (Line 10). Next, the $(\mathcal{GS})$ receives the weight matrices of the updated local models $w_{l_\kappa^t}^{k+1}$ and averages them to update the global model weights, w_g^{k+1} (Lines 4-5) in Algorithm I-3. Finally, the $(\mathcal{GS})$ updates the $global\_model(w_g^{k+1})$ with the local models weight matrices $w_{l_\kappa^t}^{k+1}$ (line 6 in Algorithm I-3) for $t \in T$. The Algorithm I-3 is terminated at the end of all communication rounds or once the loss function converges.

To assess the complexity of our proposed FeD-TST approach, we split the framework in two parts. The first part, which gives the computational complexity of the local TST models (Algorithms I-1 and I-2) and the second part, which analyzes the complexity of the Federated Learning communication (Algorithms I-3 and I-4). For the latter, the complexity depends on the number of clients $|\mathcal{LC}|$ participating in each round C of the federated training and can be calculated as $O(log(\mathcal{LC}))$ (Zhang, Wei & Berry, 2021a). The computation complexity of the first part, in order to update the weights matrix by the means of the TST model is given in the following section.

1.1 Complexity Analysis

Proposition 1: The computational complexity of the Algorithm I-1 is $O(n\sqrt{n})$

Proof: The computational complexity of lines 2-4 is $O(n)$ where n is the number of dimensions in the input tensors $\mathcal{Q}$, $\mathcal{K}$ and $\mathcal{V}$. Line 5 performs the multiplication operation on the q and k vectors and both are $1D$ vectors of length n, thus the complexity of this operation is $O(n)$. Line 6 takes $O(n\sqrt{n})$ as the operation *get_attn_mask* with a strided mode, attends only a subset of the time steps. Line 7 executes the assignment of masking values and takes constant time i.e.,

$O(1)$. Lastly, the complexity of lines 8-10 is $O(n)$ because *Softmax* depends on the size of the input vector M as it involves exponentiating each element of the M and performing a division. Similarly, the *MUL* and *Merge_heads* operations also depend on the number of dimensions of the input tensors such as $Weights$ and SA. Accordingly, the overall complexity of Algorithm 1 is represented in terms of n as: $O(n) + O(n) + O(n\sqrt{n}) + O(1) + O(n) = O(n\sqrt{n})$

Proposition 2: The computational complexity of the Algorithm I-2 i.e., TST is $O(nd + n\sqrt{n})$

Proof: For the *INPUT_MODULE*, the computational complexity of line 2 is $O(1)$ because the *Input_Layer* is used to instantiate a symbolic tensor as an input to the learning model and does not involve any significant computations; the complexity of line 3 is $O(nd)$ as the *Positional_Encoding* involves generating a matrix with sinusoidal values based on the length of the input sequence say n along with the dimension d of that sequence (i.e., the dimension for a univariate input sequence is 1 and for multivariate is 6) and it requires iterating over each position and each dimension to calculate the sinusoidal value. Next, assigning the positional encoding matrix to variable *PE* has a complexity of $O(1)$, as it is a constant-time operation. Further, the complexity of line 4 is $O(n)$, since the *Add* layer simply adds two input tensors element-wise and this operation requires a constant amount of time for each element. Thus, the overall complexity for the *INPUT_MODULE* is: $O(1) + O(nd) + O(n) = O(nd)$.

For the *ENCODER_MODULE*, the computational complexity of line 6 is $O(n)$, where n is the total number of elements in *input_res*; line 7 takes $O(n\sqrt{n})$ as the sparse attention with a strided mode, attends only a subset of positions say $(\sqrt{n})$; line 8 is applying the $\mathcal{DP}$ operation so its complexity is typically $O(n)$, as it operates element-wise on the input tensor x; lines 9 and 10 takes $O(n)$, since line 9 performs the addition operation and line 10 is normalizing the *enc_res*. Hence, the overall complexity for the *ENCODE_MODULE* is $O(n) + O(n\sqrt{n}) + O(n) + O(n) + O(n) = O(n\sqrt{n})$.

For the *DECODER_MODULE*, the computational complexity of line 12 is $O(n)$; the complexity of lines 13 and 17 is $O(n\sqrt{n})$ as they use the strided sparse attention; the complexity of lines 14-16 and lines 18-20 is $O(n)$ as these lines consist of the *dropout*, *addition* and *normalization* operations for n number of elements; the complexity of lines 18 and 19 depends on the number of filters, kernel size, previous layer outputs and activation function which is typically applied element-wise to the output of the convolution operation thus, in the worst case scenario these lines will exhibit a complexity of $O(n)$. The overall complexity of the *DECODER_MODULE* is $O(n) + O(n\sqrt{n}) + O(n) + O(n) = O(n\sqrt{n})$.

For the *OUTPUT_MODULE*, the computational complexity shows a linear behavior i.e., $O(n)$ as the lines 24-27 depend only on the length of the output from the previous layers and perform basic operations such as average, activation, addition and normalization.

Accordingly, the overall computational complexity of the proposed Algorithm I-2 is represented in terms of n as: $T(n) = O(nd) + O(n\sqrt{n}) + O(n\sqrt{n}) + O(n) = O(nd + n\sqrt{n})$ and can be simplified as $O(n\sqrt{n})$. This is because d is constant in the complexity $O(nd)$ for both settings, which means that the value of d does not depend on the input sequence length n. In this case, the complexity term $O(nd)$ can be simplified to $O(n)$.

2. Baseline Algorithms & Hyperparameters

In this appendix, we provide more details on the baseline algorithms used for comparison, while the hyperparameter configuration of each of these approaches is provided in Table I-1. It should be noted that these parameters are the same for all network configurations and that the optimizer used for all baselines and our proposed method is the Adam one, while the batch size was set to be 100. Finally, for the federated learning based methods, we set the number of communication rounds to be 30, the aggregation strategy for the server side was the FeDAvg algorithm and the number of local epochs was 100.

1. **LSTM:** The LSTM takes a sequence of QoS observations as input and typically consists of one or more LSTM layers, followed by one or more fully connected layers for the final forecasting of QoS values. The LSTM network architecture includes three main components (i) an input gate for determining how much new information should be allowed into the memory cell; (ii) a forget gate to determine how much information should be discarded

Table-A I-1 Hyperparameters used in all methods for both time series settings across all QoS datasets

Models	Hyperparameters	Value
LSTM	Number of LSTM units	100
	Activation function	relu
	Learning rate	1e-4
	Batch size	100
BiLSTM	Number of hidden units	100
	kernel_initializer	glorot_uniform
	Learning rate	1e-5
	Sequence length	30
ATT+BiLSTM	Attention mechanism	Self Attention
	Number of LSTM units	30
	BiLSTM activation	tanh
	Attention activation	relu
	Learning rate	0.0005
	Batch size	100
Seq2Seq	Number of encoder layers	3
	Number of decoder layers	1
	Number of hidden units	30
	Learning rate	0.0008
Seq2Seq+ATT	attention mechanism	Self Attention
	Number of hidden units	48
	activation	tanh
	learning rate	0.0008
TCN+BiLSTM	Number of TCN blocks	4
	Number of filters	32
	kernel size	16
	padding	'causal'
	activation	'tanh'
	BiLSTM hidden units	30
	learning rate	0.0001
T2V+Transformer	Number of heads	2
	Number of encoder layers	2
	number of feed forward	64
	Timesteps	30
	Learning rate	0.0008
FL+LSTM	number of clients	≥ 6
	Fraction of clients	100%
	Number of LSTM units	100
	learning_rate	$1e-4$
FL+MHA	Head size	64
	Number of heads	2
	dropout rate	0.1
	Number of transformer blocks	6
	Linear layer neurons	1024
	Filter dimensions	2
	number of clients	≥ 6
	Fraction of clients	100%
FeD-TST	Number of attention heads	256
	Head size	1024
	Filter size	64
	Number of transformer blocks	6
	Number of MLP units	1024
	Attention Mode	strided
	dropout rate	0.1
	learning rate	1e-6
	number of clients	≥ 6
	Fraction of clients	100%

Table-A I-2 Multivariate forecasting results for PDR, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.025376484	0.015863674	0.012133694	0.033557826	0.010215931	0.001553181	0.240865166	0.024999889
	MAE	0.126860133	0.098012105	0.096787978	0.15019938	0.086596898	0.029961921	0.488339741	0.12614061
	RMSE	0.159299982	0.125951077	0.110153049	0.183187951	0.10107389	0.039410419	0.490780161	0.158113532
BiLSTM	MSE	0.025236582	0.015690649	0.009789112	0.037502334	0.010475132	0.001242641	0.235032849	0.023964328
	MAE	0.126581692	0.096840975	0.085831458	0.159663698	0.090752476	0.026729854	0.482312624	0.123556575
	RMSE	0.158860261	0.125262319	0.09893994	0.193655192	0.102348093	0.03525112	0.484801866	0.154804159
ATT+BiLSTM	MSE	0.025570284	0.020930944	**0.001872818**	0.056448757	0.003671356	0.000815777	0.285043016	0.020710045
	MAE	0.127044161	0.120655717	**0.034403121**	0.195198673	0.041968134	0.022205845	0.531922871	0.114103137
	RMSE	0.159907111	0.144675306	**0.043276072**	0.237589471	0.060591713	0.028561803	0.533894199	0.143909852
Seq2Seq	MSE	0.025525104	0.0437667	0.027792311	0.056832227	0.2046942	0.077074158	0.328894673	0.020404245
	MAE	0.126883348	0.180243128	0.161392866	0.208786351	0.447453349	0.253005704	0.571669152	0.113296365
	RMSE	0.159765777	0.209204923	0.166710261	0.238395107	0.452431431	0.277622329	0.573493394	0.142843428
Seq2Seq+ATT	MSE	0.025569492	0.032245812	0.017828528	0.046243408	0.190555137	0.087843576	0.319531187	0.022504165
	MAE	0.126394561	0.146772905	0.12693602	0.188916556	0.431019692	0.274577396	0.563395886	0.11975392
	RMSE	0.159904633	0.17957119	0.133523511	0.215042805	0.436526216	0.296384169	0.565270897	0.150013884
TCN+BiLSTM	MSE	0.025064324	0.015246273	0.002248423	0.034247903	0.004607236	0.000689827	0.161620519	0.022040544
	MAE	0.125900357	0.096963628	0.036923015	0.151996426	0.028721076	0.018581239	0.399189365	0.118625733
	RMSE	0.158317162	0.1234758	0.047417542	0.185061889	0.067876624	0.026264548	0.402020546	0.148460581
FL+LSTM	MSE	0.030039437	0.041287486	0.070088893	0.074822351	0.088905558	0.039511204	0.083336249	0.022051753
	MAE	0.138459548	0.167551339	0.144682601	0.224284515	0.279592633	0.149327978	0.230864123	0.117232472
	RMSE	0.173077464	0.202952102	0.264695406	0.273371756	0.298069835	0.198613957	0.288496435	0.148181751
FL+MSA	MSE	0.151883692	0.045339461	0.05724353	0.454306275	0.195067286	0.144678041	0.081710711	0.483018905
	MAE	0.354675621	0.168622926	0.153972432	0.638280272	0.337508738	0.32461971	0.225243196	0.667602837
	RMSE	0.389722586	0.21293065	0.239256188	0.674022436	0.441664219	0.38036567	0.285850853	0.694995642
FeD-TST	MSE	**8.39E-08**	**0.000219183**	0.006694946	**2.54E-05**	**0.002069899**	**0.000359125**	**0.00343068**	**0.007323688**
	MAE	**0.000216889**	**0.011904503**	0.062316246	**0.003934593**	**0.011523922**	**0.015154229**	**0.046734169**	**0.06500271**
	RMSE	**0.000289732**	**0.014804841**	0.081822649	**0.005042353**	**0.045496136**	**0.018950595**	**0.058572009**	**0.085578553**

from the memory cell and (iii) an output gate to ensure how much of the information in the memory cell should be outputted. There is no global model in this case, as LSTM is trained for each QoS dataset in a centralized manner.

2. **Bidirectional LSTM (BiLSTM):** The Bidirectional LSTM is a type of LSTM network that can process a sequence of QoS inputs in both forward and backward directions, which allows the model to capture both past and future context for each time step in the QoS sequence. It consists of two LSTMs: one processing the input sequence in the forward direction, and the other processing the input sequence in the backward direction. The output of each LSTM is then concatenated to form the final output for each time step.
3. **Attention + BiLSTM:** This model combines the benefits of both BiLSTM and attention mechanism to improve the forecasting accuracy. The Attention mechanism is used to help the model focus on the most important parts of the input sequence during forecasts. It assigns weights to different parts of the input sequence based on their relevance to the current forecast, allowing the model to attend to the most informative parts of the input

sequence and ignore irrelevant information. Firstly, the model processes the input sequence using the BiLSTM layer, which generates hidden representations for each element of the sequence. Then, such hidden representations are used by the attention layer to get the attention weights which are then applied to the hidden representations to obtain a weighted sum. The weighted sum is then passed through a fully connected layer to produce the final output (i.e., QoS forecast).

4. **Sequence-to-Sequence:** A Seq2Seq model comprises of two main components, namely an encoder and decoder. The encoder takes the input sequence and converts it into a fixed-length representation, which is typically a vector. This representation, also called the context vector, captures the relevant information of the input sequence and is used by the decoder component which generates the output sequence as one element at a time. At each time step, the decoder takes the previously generated output element, along with the context vector, and uses them to generate the next output element. This process is repeated until the entire output sequence has been generated.
5. **Sequence-to-Sequence + Attention:** In this architecture (Li *et al.*, 2022a), the attention mechanism is added to the Seq2Seq architecture to improve the model's ability to focus on the most relevant parts of the input sequence during the decoding process. In the Seq2Seq model with Attention, the input sequence is first passed through an embedding layer to convert the discrete input tokens into continuous vector representations. The encoded sequence is then fed into the attention mechanism, which generates a weighted context vector based on the input sequence and the current decoder state. The context vector is then concatenated with the decoder's previous output and fed through the decoder RNN to generate the next output token. This process is repeated iteratively until the entire output sequence is generated.
6. **Temporal Convolutional Networks + BiLSTM:** In the Temporal Convolutional Networks (TCN) (Chen *et al.*, 2020) along with the BiLSTM layer, the TCN layers are used for feature extraction which use causal convolutions to process the input sequence, with dilation factors that increase exponentially across the layers. The output of the TCN layers is then fed into the BiLSTM layer, which process the sequence in both forward and backward direction.

This allows the model to capture both short-term and long-term dependencies in the input sequence. The final layer is a fully connected layer that maps the output of the BiLSTM layer to the forecasting (output) format.

7. **Time2Vec+Transformer:** In this architecture (Kazemi *et al.*, 2019), the Time2Vec model is used to embed the time-series data into a continuous vector space, which is then fed into the Transformer network for further processing. The Transformer network consists of multiple layers, each with its own self-attention and feedforward components. The self-attention component allows the model to focus on the most relevant parts of the input sequence, while the feedforward component applies non-linear transformations to the input features. The output of the Transformer network is then typically fed into one or more fully connected layers to produce the final output.
8. **FL+LSTM:** In this network architecture, the LSTM model is trained in a decentralized manner across multiple clients without sharing their local QoS data with a central server. The training process in FL involves multiple rounds of communication between the clients and the server, which aggregates the local updates received from the clients and sends the global model parameters back to the clients for further training. Each client has its own local LSTM model, and the server aggregates the local model updates to generate a global LSTM model. During each round of communication, the clients train their local LSTM models using their own local data and send the updated model parameters to the server. The server aggregates the model updates using a weighted average based on the number of samples and sends the updated global LSTM model back to the clients for further training. This process continues for several rounds until convergence.
9. **FL+MHA:** In this architecture, we applied our previous temporal transformer model (Hameed *et al.*, 2022) in the federated setting. The temporal transformer network is composed of the three following modules (i) an input embedding module, which takes the input into a specific tensor shape for the transformer along with providing the positional encoding, (ii) an encoder module consisting of a multi-head attention layer and feed-forward layers and (iii) an output module, which is used to provide the final prediction of the QoS dataset. So considering this transformer model, each local client has access to its own dataset

and trains a local temporal transformer network. These local models are then aggregated by the server using federated averaging to obtain a global model that captures the overall patterns and trends in the data. During training, the local models learn to capture the local patterns and trends in their own datasets, while the global model learns to generalize them across all the datasets. This approach allows for decentralized training while still capturing the global patterns in the data.

Table-A I-3 Multivariate forecasting results for PLR, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.025038941	0.085829899	0.002392895	0.027448104	0.011076534	0.067128993	0.119673633	0.001883547
	MAE	0.133605405	0.273123358	0.04587984	0.135984587	0.102786578	0.217031947	0.345302414	0.037316061
	RMSE	0.158236978	0.292967402	0.048917222	0.165674695	0.105245114	0.259092634	0.345938771	0.043399853
BiLSTM	MSE	0.158236978	0.292967402	0.048917222	0.165674695	0.105245114	0.259092634	0.345938771	0.043399853
	MAE	0.132812767	0.26347576	0.035682182	0.148675127	0.097600599	0.214792671	0.336972344	0.034692373
	RMSE	0.158153803	0.284322544	0.039330348	0.179501957	0.100934994	0.257189624	0.337591786	0.041591182
ATT+BiLSTM	MSE	0.025100036	0.104297191	**0.000298454**	0.033911135	0.000634337	0.064385353	0.007021591	0.003547903
	MAE	0.133864995	0.303442026	**0.01312486**	0.154987831	0.01701582	0.208894609	0.082569718	0.040551163
	RMSE	0.158429908	0.322950756	**0.017275817**	0.184149764	0.025186057	0.25374269	0.083794934	0.059564274
Seq2Seq	MSE	0.025047779	0.021822	0.008536431	0.0827658	0.10668844	0.073154119	0.084986213	0.001833366
	MAE	0.133894046	0.123885804	0.090917699	0.27145796	0.326401588	0.231702971	0.291134455	0.037281562
	RMSE	0.158264902	0.147722714	0.092392809	0.287690459	0.326631964	0.270470181	0.29152395	0.04281782
Seq2Seq+ATT	MSE	0.025070594	0.025402517	0.006077862	0.068647046	0.11512791	0.074626937	0.103903783	0.001755035
	MAE	0.134396355	0.134585947	0.076356004	0.241265618	0.339119511	0.234660944	0.321946763	0.036355348
	RMSE	0.158336962	0.159381672	0.077960647	0.262005812	0.33930504	0.273179313	0.322341097	0.041893137
TCN+BiLSTM	MSE	0.02506516	0.128794342	0.000631897	0.017174813	0.000720222	0.067145159	0.005607512	0.001262265
	MAE	0.134126641	0.342004925	0.020054774	0.11020899	0.020433414	0.217653834	0.070392203	0.026720259
	RMSE	0.158319801	0.358879287	0.025137552	0.13105271	0.026836948	0.25912383	0.074883327	0.035528375
FL+LSTM	MSE	0.087703511	0.049748622	0.036141779	0.070266478	0.066877857	0.14487806	0.06205624	0.011543612
	MAE	0.229409769	0.183092251	0.082843989	0.210182503	0.220609531	0.329201072	0.205156624	0.037259299
	RMSE	0.296055824	0.22295104	0.190040812	0.264971823	0.258534551	0.380628496	0.248941362	0.10736721
FL+MSA	MSE	0.050034031	0.053757463	3.119338512	5.649907589	8.785881996	0.14487806	0.060680598	0.099903829
	MAE	0.180061623	0.189969048	1.746221423	2.345210314	2.918813229	0.329201072	0.201134384	0.198856756
	RMSE	0.22368288	0.231856555	1.766164899	2.376953363	2.96409893	0.380628496	0.246334314	0.316075682
FeD-TST	MSE	**4.12E-07**	**3.27E-09**	0.008334739	**2.65E-05**	**0.00020217**	**1.68E-06**	**0.005168275**	**0.000906298**
	MAE	**0.00050366**	**1.49E-05**	0.065119371	**0.004061751**	**0.010798856**	**0.001012479**	**0.059207294**	**0.022389103**
	RMSE	**0.000641564**	**5.72E-05**	0.091294795	**0.005146937**	**0.014218667**	**0.00129578**	**0.071890719**	**0.030104777**

3. Multivariate prediction for PDR, PLR and Latency

In this part of the appendix we include the results of the multivariate forecasting of the PDR, PLR, and Latency QoS metrics. As shown in Tables I-2 and I-3, the FeD-TST consistently gives the best performance for the large majority of the network configurations. In only one network configuration (mobile robots, same channel, $0dBm$) FeD-TST did not manage to find the best prediction accuracy for both QoS metrics. For this configuration, a high level of interference and congestion is noticed since the robots are mobile and operate on the same channel. This

Table-A I-4 Multivariate forecasting results for Latency, best results are highlighted in bold

Configurations		1	2	3	4	5	6	7	8
Methods	Metrics	Mobile	Mobile	Mobile	Mobile	Static	Static	Static	Static
LSTM	MSE	0.005637791	0.043465063	0.008396188	0.094484142	0.002421324	0.00004434	0.110441343	0.001464042
	MAE	0.073327712	0.174966194	0.090718731	0.251945544	0.048501279	0.004837051	0.33225192	0.038033014
	RMSE	0.075085226	0.208482765	0.091630717	0.307382729	0.049206954	0.006658817	0.332327162	0.038262798
BiLSTM	MSE	0.001439423	0.033124764	0.005707397	0.075514612	0.002677222	0.000061788	7.016391754	0.001402291
	MAE	0.030769233	0.148010391	0.074070212	0.223332865	0.050801641	0.006053792	2.459266663	0.036949068
	RMSE	0.037939731	0.1820021	0.075547313	0.274799221	0.051741883	0.007860557	2.648847342	0.037447176
ATT+BiLSTM	MSE	0.000344244	0.051534736	0.000069114	0.169493278	2.534987688	0.000297986	0.254173652	0.000295972
	MAE	0.018082803	0.197647946	0.007791087	0.372243206	1.04771924	0.016313722	0.504086238	0.014796807
	RMSE	0.018553823	0.227012634	0.008313497	0.411695614	1.592164516	0.017262263	0.504156376	0.017203839
Seq2Seq	MSE	0.105809007	0.202318646	0.016844664	0.224655949	0.036616328	0.043741006	0.131628002	0.002322215
	MAE	0.325255403	0.42963242	0.129758603	0.404872489	0.191155483	0.208386975	0.362794546	0.048175558
	RMSE	0.325282964	0.449798451	0.129786996	0.473978849	0.191353933	0.209143505	0.362805736	0.048189364
Seq2Seq+ATT	MSE	0.097825267	0.202831502	0.017442157	0.213285898	0.040685284	0.042646595	0.145325763	0.001978905
	MAE	0.312745032	0.430422599	0.132059379	0.393857653	0.201548577	0.205651407	0.381142717	0.044481885
	RMSE	0.31277031	0.450368185	0.132068757	0.461828862	0.201705934	0.206510519	0.381216163	0.04448489
TCN+BiLSTM	MSE	0.000337262	0.017626127	0.780299902	0.053097963	0.00585584	0.000074349	0.087526544	**0.000098222**
	MAE	0.017728718	0.105348373	0.342651963	0.200600296	0.034037738	0.006348549	0.29580424	**0.00733645**
	RMSE	0.018364683	0.132763423	0.883345842	0.230429947	0.076523459	0.008622569	0.295848853	**0.009910695**
FL+LSTM	MSE	0.104215905	0.074008435	0.062736414	0.0833028852	0.0606529861	0.0579579472	0.116129011	0.047641791
	MAE	0.210479304	0.222253785	0.090594731	0.25273493	0.167973026	0.150502234	0.303388238	0.070485658
	RMSE	0.322496265	0.271815836	0.250428975	0.288488924	0.246236875	0.240110695	0.340605199	0.218234643
FL+MSA	MSE	0.09929058	0.31490236	0.03957324	0.38431263	0.95486623	0.09382478	0.11782001	0.03207614
	MAE	0.18505890	0.530799746	0.082799196	0.550994515	0.945636034	0.188609883	0.299379557	0.138352171
	RMSE	0.315104067	0.561161637	0.198930264	0.619929552	0.977172553	0.306308329	0.343249202	0.179098144
FeD-TST	MSE	**6.56E-06**	**1.56E-09**	**0.000033704**	**0.040740803**	**0.000081493**	**5.23E-07**	**0.070651543**	0.089611024
	MAE	**0.001430776**	**2.77E-05**	**0.004033627**	**0.159722609**	**0.007586633**	**0.000555087**	**0.26576637**	0.056043442
	RMSE	**0.002561764**	**3.95E-05**	**0.005805549**	**0.201843511**	**0.009027364**	**0.000722966**	**0.26580358**	0.299351007

can result in a considerable decrease of the PDR and an increase of the PLR. In a multivariate forecasting setting, where the PDR and PLR are being forecasted along with other features, the presence of noise and irrelevant data can make the QoS forecasting challenging. Even though Fed-TST model presents a sub-optimal performance the difference in the accuracy is marginal showing that it can adequately perform even in such challenging settings.

Lastly, for the latency results, Table I-4 shows that the forecast can be accurately made from the FeD-TST model for all network configuration but the last one (static robots, same channel, $12dBm$). TCN+BiLSTM is the model that achieves the highest performance for the particular configuration and the reason is that this model can handle the complexity and non-linearity of the data, as well as the interference and congestion caused by multiple access points operating on the same frequency channel with a high transmission power of 12 dBm. Moreover, the model effectively captured the interactions between different features and can make use of the relevant information to make accurate forecasts.

BIBLIOGRAPHY

(2023). Neural network models (supervised). Retrieved from: https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

Abdellah, A. R., Abdulkareem Mahmood, O. & Koucheryavy, A. (2020a). Delay prediction in IoT using Machine Learning Approach. *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pp. 275-279. doi: 10.1109/ICUMT51630.2020.9222245.

Abdellah, A. R., Artem, V., Muthanna, A., Gallyamov, D. & Koucheryavy, A. (2020b). Deep Learning for IoT Traffic Prediction Based on Edge Computing. *Distributed Computer and Communication Networks: Control, Computation, Communications*, pp. 18–29.

Abdellah, A. R., Artem, V., Muthanna, A., Gallyamov, D. & Koucheryavy, A. (2020c). Deep Learning for IoT Traffic Prediction Based on Edge Computing. *Distributed Computer and Communication Networks: Control, Computation, Communications*, pp. 18–29.

Adjih, C., Baccelli, E., Fleury, E., Harter, G., Mitton, N., Noel, T., Pissard-Gibollet, R., Saint-Marcel, F., Schreiner, G., Vandaele, J. & Watteyne, T. (2015). FIT IoT-LAB: A large scale open experimental IoT testbed. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 459-464. doi: 10.1109/WF-IoT.2015.7389098.

Aneja, S., Aneja, N. & Islam, M. S. (2018). IoT Device Fingerprint using Deep Learning. *2018 IEEE International Conference on Internet of Things and Intelligence System (IOTAIS)*, pp. 174-179. doi: 10.1109/IOTAIS.2018.8600824.

Aneja, S., Aneja, N., Bhargava, B. & Chowdhury, R. R. (2022). Device Fingerprinting Using Deep Convolutional Neural Networks. *Int. J. Commun. Netw. Distrib. Syst.*, 28(2), 171–198. doi: 10.1504/ijcnds.2022.121197.

Apthorpe, N. & et al. (2017). A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic.

Ateeq, M., Ishmanov, F., Afzal, M. K. & Naeem, M. (2019a). Predicting Delay in IoT Using Deep Learning: A Multiparametric Approach. *IEEE Access*, 7, 62022-62031. doi: 10.1109/ACCESS.2019.2915958.

Ateeq, M., Ishmanov, F., Afzal, M. K. & Naeem, M. (2019b). Multi-Parametric Analysis of Reliability and Energy Consumption in IoT: A Deep Learning Approach. *Sensors*, 19(2). doi: 10.3390/s19020309.

Backhaus, K., Erichson, B., Gensler, S., Weiber, R. & Weiber, T. (2023). Logistic Regression. In *Multivariate analysis an application-oriented introduction*. Springer Fachmedien Wiesbaden GmbH.

Bahad, P. & Saxena, P. (2020). Study of AdaBoost and Gradient Boosting Algorithms for Predictive Analytics. *International Conference on Intelligent Computing and Smart Communication 2019*, pp. 235–244.

Bai, L., Yao, L., Kanhere, S. S., Wang, X. & Yang, Z. (2018a). Automatic Device Classification from Network Traffic Streams of Internet of Things. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, pp. 1-9. doi: 10.1109/LCN.2018.8638232.

Bai, S., Kolter, J. Z. & Koltun, V. (2018b). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bardalai, P., Neog, H., Dutta, P. E., Medhi, N. & Deka, S. K. (2022). Throughput Prediction in Smart Healthcare Network using Machine Learning Approaches. *2022 IEEE 19th India Council International Conference (INDICON)*, pp. 1-6. doi: 10.1109/INDICON56171.2022.10040160.

Beutel, D. J. et al. (2022). Flower: A Friendly Federated Learning Research Framework.

Bezawada, B., Bachani, M., Peterson, J., Shirazi, H., Ray, I. & Ray, I. (2018). Behavioral Fingerprinting of IoT Devices. *Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security*, (ASHES ’18), 41–50. doi: 10.1145/3266444.3266452.

Brownlee, J. (2020, Aug). How to choose a feature selection method for machine learning. Retrieved from: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

Chen, Y., Kang, Y., Chen, Y. & Wang, Z. (2020). Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 399, 491-501. doi: https://doi.org/10.1016/j.neucom.2020.03.011.

Chen, Y., Yu, P., Zheng, Z., Shen, J. & Guo, M. (2022). Modeling feature interactions for context-aware QoS prediction of IoT services. *Future Generation Computer Systems*, 137, 173-185. doi: https://doi.org/10.1016/j.future.2022.07.017.

Cheng, H., Xie, Z., Wu, L., Yu, Z. & Li, R. (2019). Data prediction model in wireless sensor networks based on bidirectional LSTM. *EURASIP Journal on Wireless Communications and Networking*, 2019(1), 1–12.

Chowdhury, R. R., Aneja, S., Aneja, N. & Abas, E. (2020). Network Traffic Analysis Based IoT Device Identification. *Proceedings of the 2020 4th International Conference on Big Data and Internet of Things*, (BDIOT '20), 79–89. doi: 10.1145/3421537.3421545.

Cisco. (2022, Jan). Cisco. Retrieved from: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

Cisco Systems. [Available online: http://www.audentia-gestion.fr/cisco/pdf/at-a-glance-c45-731471.pdf]. (2017). Internet of Things at a Glance. Retrieved from: Whitepaper.

Dechouniotis, D., Athanasopoulos, N., Leivadeas, A., Mitton, N., Jungers, R. & Papavassiliou, S. (2020). Edge Computing Resource Allocation for Dynamic Networks: The DRUID-NET Vision and Perspective. *Sensors*, 20(8). doi: 10.3390/s20082191.

Ericsson. (2022). *Mobile data traffic outlook*.

Fan, L., Zhang, S., Wu, Y., Wang, Z., Duan, C., Li, J. & Yang, J. (2020). An IoT Device Identification Method based on Semi-supervised Learning. *2020 16th International Conference on Network and Service Management (CNSM)*, pp. 1-7. doi: 10.23919/C-NSM50824.2020.9269044.

Fan, X., Xiang, C., Gong, L., He, X., Chen, C. & Huang, X. (2019). UrbanEdge: Deep Learning Empowered Edge Computing for Urban IoT Time Series Prediction. *Proceedings of the ACM Turing Celebration Conference - China*, (ACM TURC '19). doi: 10.1145/3321408.3323089.

Hahn, Y., Langer, T., Meyes, R. & Meisen, T. (2023). Time Series Dataset Survey for Forecasting with Deep Learning. *Forecasting*, 5(1), 315–335. doi: 10.3390/forecast5010017.

Hameed, A. & et al. (2020). IoT traffic multi-classification using network and statistical features in a smart environment. *2020 IEEE 25th international workshop on computer aided modeling and design of communication links and networks (CAMAD)*, pp. 1–7.

Hameed, A., Violos, J., Santi, N., Leivadeas, A. & Mitton, N. (2021). 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)*, pp. 29-36. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics53846.2021.00020.

Hameed, A., Violos, J., Leivadeas, A., Santi, N., Grünblatt, R. & Mitton, N. (2022). Toward QoS Prediction Based on Temporal Transformers for IoT Applications. *IEEE Transactions on Network and Service Management*, 19(4), 4010-4027. doi: 10.1109/TNSM.2022.3217170.

Hanes, D., Salguiero, G., Grossetete, P., Barton, R. & Henry, J. (2017). *IoT Fundamentals: Networking Technologies, Protocol, and Use Cases for the Internet of Things*. Cisco Press.

Hauke, J. & Kossowski, T. (2011). Comparison of Values of Pearson's and Spearman's Correlation Coefficients on the Same Sets of Data. *Quaestiones Geographicae*, 30(2), 87-93. doi: doi:10.2478/v10117-011-0021-1.

Henry, M. (2021). Review on gradient descent algorithms in deep learning approaches. *SSRN Electronic Journal*. doi: 10.2139/ssrn.3817511.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735.

Hosmer, D. W., Lemeshow, S. & Cook, E. D. (2001). *Applied Logistic Regression*.

Hou, Y., Yano, K., Suga, N., Webber, J., Nii, E., Higashimori, T., Denno, S. & Suzuki, Y. (2021). A Study of Throughput Prediction using Convolutional Neural Network over Factory Environment. *2021 23rd International Conference on Advanced Communication Technology (ICACT)*, pp. 429-434. doi: 10.23919/ICACT51234.2021.9370905.

Hui, S., Wang, H., Xu, D., Wu, J., Li, Y. & Jin, D. (2022). Distinguishing Between Smartphones and IoT Devices via Network Traffic. *IEEE Internet of Things Journal*, 9(2), 1182-1196. doi: 10.1109/JIOT.2021.3078879.

IoT-LAB, F. (2022, Aug). Files · main · DruidNet / IOT mobility traces · GITLAB. Retrieved from: https://gitlab.inria.fr/druidnet/iot-mobility-traces/-/tree/main/.

Ivanov, N. (2019, Jan). Unleashing the internet of things with in-memory computing: IOT NOW news & amp; reports. Retrieved from: https://www.iot-now.com/2019/01/17/92200-unleashing-internet-things-memory-computing.

Jia, Z., Jin, L., Zhang, Y., Liu, C., Li, K. & Yang, Y. (2022). Location-Aware Web Service QoS Prediction via Deep Collaborative Filtering. *IEEE Transactions on Computational Social Systems*, 1-12. doi: 10.1109/TCSS.2022.3217277.

Jin, H., Zhang, P., Dong, H., Zhu, Y. & Bouguettaya, A. (2023). Privacy-Aware Forecasting of Quality of Service in Mobile Edge Computing. *IEEE Transactions on Services Computing*, 16(1), 478-492. doi: 10.1109/TSC.2021.3137452.

Kazemi, M. et al. (2019). Time2Vec: Learning a Vector Representation of Time. *arXiv e-prints*, arXiv–1907.

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I. & Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, 97, 219-235. doi: https://doi.org/10.1016/j.future.2019.02.050.

Kotak, J. & Elovici, Y. (2021). IoT Device Identification Using Deep Learning. *13th International Conference on Computational Intelligence in Security for Information Systems (CISIS 2020)*, pp. 76–86.

Lai, G., Chang, W.-C., Yang, Y. & Liu, H. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks.

Laner, M., Nikaein, N., Svoboda, P., Popovic, M., Drajic, D. & Krco, S. (2015). 8 - Traffic models for machine-to-machine (M2M) communications: types and applications. In Antón-Haro, C. & Dohler, M. (Eds.), *Machine-to-machine (M2M) Communications* (pp. 133-154). Oxford: Woodhead Publishing. doi: https://doi.org/10.1016/B978-1-78242-102-3.00008-3.

Laner, M. & et al. (2013). Traffic Models for Machine Type Communications. *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*, pp. 1-5.

Li, G., Li, F., Ahmad, T., Liu, J., Li, T., Fang, X. & Wu, Y. (2022a). Performance evaluation of sequence-to-sequence-Attention model for short-term multi-step ahead building energy predictions. *Energy*, 259, 124915. doi: https://doi.org/10.1016/j.energy.2022.124915.

Li, J., Wu, H., Chen, J., He, Q. & Hsu, C.-H. (2022b). Topology-Aware Neural Model for Highly Accurate QoS Prediction. *IEEE Transactions on Parallel and Distributed Systems*, 33(7), 1538-1552. doi: 10.1109/TPDS.2021.3116865.

Li, S., Wen, J. & Wang, X. (2019). From Reputation Perspective: A Hybrid Matrix Factorization for QoS Prediction in Location-Aware Mobile Service Recommendation System. *Mobile Information Systems*, (1574-017X), 8950508. doi: 10.1155/2019/8950508.

Li, X., Li, S., Li, Y., Zhou, Y., Chen, C. & Zheng, Z. (2022c). A Personalized Federated Tensor Factorization Framework for Distributed IoT Services QoS Prediction From Heterogeneous Data. *IEEE Internet of Things Journal*, 9(24), 25460-25473. doi: 10.1109/JIOT.2022.3197172.

Liang, T., Chen, M., Yin, Y., Zhou, L. & Ying, H. (2022). Recurrent Neural Network Based Collaborative Filtering for QoS Prediction in IoV. *IEEE Transactions on Intelligent Transportation Systems*, 23(3), 2400-2410. doi: 10.1109/TITS.2021.3099346.

Lippmann, R. & et al. (2003). Passive Operating System Identification From TCP / IP Packet Headers *.

Liu, F., Ren, X., Zhang, Z., Sun, X. & Zou, Y. (2021). Rethinking Skip Connection with Layer Normalization in Transformers and ResNets. *ArXiv*, abs/2105.07205.

Liu, Z., Sheng, Q. Z., Zhang, W. E., Chu, D. & Xu, X. (2019). Context-Aware Multi-QoS Prediction for Services in Mobile Edge Computing. *2019 IEEE International Conference on Services Computing (SCC)*, pp. 72-79. doi: 10.1109/SCC.2019.00024.

Liu, Z., Sheng, Q. Z., Xu, X., Chu, D. & Zhang, W. E. (2022). Context-Aware and Adaptive QoS Prediction for Mobile Edge Computing Services. *IEEE Transactions on Services Computing*, 15(1), 400-413. doi: 10.1109/TSC.2019.2944596.

Lopez-Martin, M. & et al. (2017). Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*, 5, 18042-18050. doi: 10.1109/ACCESS.2017.2747560.

Lopez-Martin, M., Carro, B. & Sanchez-Esguevillas, A. (2019). Neural network architecture based on gradient boosting for IoT traffic prediction. *Future Generation Computer Systems*, 100, 656-673. doi: https://doi.org/10.1016/j.future.2019.05.060.

Mangla, U. (2022). Application of Federated Learning in Telecommunications and Edge Computing. In Ludwig, H. & Baracaldo, N. (Eds.), *Federated Learning: A Comprehensive Overview of Methods and Applications* (pp. 523–534). Springer International Publishing. doi: 10.1007/978-3-030-96896-0_25.

Mao, Y., You, C., Zhang, J., Huang, K. & Letaief, K. B. (2017). A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, 19(4), 2322-2358. doi: 10.1109/COMST.2017.2745201.

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A. & Yaqoob, I. (2017). Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges. *IEEE Access*, 5, 5247-5261. doi: 10.1109/ACCESS.2017.2689040.

Meidan, Y., Bohadana, M., Shabtai, A., Guarnizo, J. D., Ochoa, M., Tippenhauer, N. O. & Elovici, Y. (2017a). ProfilIoT: a machine learning approach for IoT device identification based on network traffic analysis. *Proceedings of the Symposium on Applied Computing*.

Meidan, Y., Bohadana, M., Shabtai, A., Ochoa, M., Tippenhauer, N. O., Guarnizo, J. D. & Elovici, Y. (2017b). Detection of unauthorized IoT devices using machine learning techniques. *arXiv preprint arXiv:1709.04647*.

Miettinen, M., Marchal, S., Hafeez, I., Frassetto, T., Asokan, N., Sadeghi, A.-R. & Tarkoma, S. (2017). IoT Sentinel Demo: Automated Device-Type Identification for Security Enforcement in IoT. *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2511-2514. doi: 10.1109/ICDCS.2017.284.

Mukhopadhyay, S. C. & Suryadevara, N. K. (2014). Internet of Things: Challenges and Opportunities. In Mukhopadhyay, S. C. (Ed.), *Internet of Things: Challenges and Opportunities* (pp. 1–17). Cham: Springer International Publishing. doi: 10.1007/978-3-319-04223-7_1.

Nordrum, A. (2016, Aug). Popular internet of things forecast of 50 billion devices by 2020 is outdated. Retrieved from: https://spectrum.ieee.org/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated#toggle-gdpr.

O' Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L. et al. (2019). Keras-Team/Keras-Tuner: A hyperparameter tuning library for Keras. Retrieved from: https://github.com/keras-team/keras-tuner.

Okwu, M. O. & Tartibu, L. K. (2021). Artificial Neural Network. In *Metaheuristic Optimization: Nature-Inspired Algorithms Swarm and Computational Intelligence, Theory and Applications* (pp. 133–145). Cham: Springer International Publishing. doi: 10.1007/978-3-030-61111-8_14.

Orrevad, A. (2009). M2M Traffic Characteristics : When machines participate in communication.

Pan, X., Coen-Cagli, R. & Schwartz, O. (2021). Modeling Neural Variability in Deep Networks with Dropout. *bioRxiv*. doi: 10.1101/2021.08.19.457035.

Pinheiro, A. J., de M. Bezerra, J., Burgardt, C. A. & Campelo, D. R. (2019). Identifying IoT devices and events based on packet length from encrypted traffic. *Computer Communications*, 144, 8-17. doi: https://doi.org/10.1016/j.comcom.2019.05.012.

Pratap, A., Kumar, A. & Kumar, M. (2021). Analyzing the Need of Edge Computing for Internet of Things (IoT). *Proceedings of Second International Conference on Computing, Communications, and Cyber-Security*, pp. 203–212.

Qi, L., Zhang, X., Dou, W., Hu, C., Yang, C. & Chen, J. (2018). A two-stage locality-sensitive hashing based approach for privacy-preserving mobile service recommendation in cross-platform edge environment. *Future Generation Computer Systems*, 88, 636-643. doi: https://doi.org/10.1016/j.future.2018.02.050.

Ren, J., Dubois, D. J., Choffnes, D., Mandalari, A. M., Kolcun, R. & Haddadi, H. (2019). Information Exposure From Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach. *Proceedings of the Internet Measurement Conference*, (IMC '19), 267–279. doi: 10.1145/3355369.3355577.

Reza, S., Ferreira, M. C., Machado, J. & Tavares, J. M. R. (2022). A multi-head attention-based transformer model for traffic flow forecasting with a comparative analysis to recurrent neural networks. *Expert Systems with Applications*, 202, 117275. doi: https://doi.org/10.1016/j.eswa.2022.117275.

S., B. D. C. & Ram, M. (2022). *Recent advances in Time Series forecasting*. CRC Press, Taylor &amp; Francis Group.

Saeik, F., Avgeris, M., Spatharakis, D., Santi, N., Dechouniotis, D., Violos, J., Leivadeas, A., Athanasopoulos, N., Mitton, N. & Papavassiliou, S. (2021). Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195, 108177.

Said, O. & Tolba, A. (2021). Accurate performance prediction of IoT communication systems for smart cities: An efficient deep learning based solution. *Sustainable Cities and Society*, 69, 102830. doi: https://doi.org/10.1016/j.scs.2021.102830.

Salinas, D. & et al. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3), 1181-1191. doi: https://doi.org/10.1016/j.ijforecast.2019.07.001.

Santi, N., Grünblatt, R., Foubert, B., Hameed, A., Violos, J., Leivadeas, A. & Mitton, N. (2021). Automated and Reproducible Application Traces Generation for IoT Applications. *Proceedings of the 17th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, (Q2SWinet '21), 17–24. doi: 10.1145/3479242.3487321.

Santos, M. R. P. & et al. (2018). An efficient approach for device identification and traffic classification in IoT ecosystems. *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 00304-00309. doi: 10.1109/ISCC.2018.8538630.

Shahid, M. R. & et al. (2018). IoT Devices Recognition Through Network Traffic Analysis. *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5187-5192. doi: 10.1109/BigData.2018.8622243.

Singh, J. & Banerjee, R. (2019). A Study on Single and Multi-layer Perceptron Neural Network. *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 35-40. doi: 10.1109/ICCMC.2019.8819775.

Sivanathan, A., Sivaraman, V., Vishwanath, A., Wijenayake, C., Radford, A., Loi, F. & Gharakheili, H. H. (2018, Aug). Data collected for IEEE TMC. Retrieved from: https://iotanalytics.unsw.edu.au/iottraces.html.

Sivanathan, A., Gharakheili, H. H., Loi, F., Radford, A., Wijenayake, C., Vishwanath, A. & Sivaraman, V. (2019). Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics. *IEEE Transactions on Mobile Computing*, 18(8), 1745-1759. doi: 10.1109/TMC.2018.2866249.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. (NIPS'14), 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, Q., Xu, J., Chen, H. & He, B. (2017). Two improved continuous bag-of-word models. *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2851-2856. doi: 10.1109/IJCNN.2017.7966208.

Werneck Oliveira, G. W., Toscano Ney, R., Herrera, J. L., Macêdo Batista, D., Hirata, R., Galán-Jiménez, J., Berrocal, J., Murillo, J. M., Luiz Dos Santos, A. & Nogueira, M. (2021). Predicting Response Time in SDN-Fog Environments for IIoT Applications. *2021 IEEE Latin-American Conference on Communications (LATINCOM)*, pp. 1-6. doi: 10.1109/LATINCOM53176.2021.9647803.

White, G. & Clarke, S. (2022). Short-Term QoS Forecasting at the Edge for Reliable Service Applications. *IEEE Transactions on Services Computing*, 15(2), 1089-1102. doi: 10.1109/TSC.2020.2975799.

Wu, D., Xu, H., Jiang, Z., Yu, W., Wei, X. & Lu, J. (2021). EdgeLSTM: Towards Deep and Sequential Edge Computing for IoT Applications. *IEEE/ACM Trans. Netw.*, 29(4), 1895–1908. doi: 10.1109/TNET.2021.3075468.

Wu, H., Yue, K., Hsu, C.-H., Zhao, Y., Zhang, B. & Zhang, G. (2017). Deviation-based neighborhood model for context-aware QoS prediction of cloud and IoT services. *Future Generation Computer Systems*, 76, 550-560. doi: https://doi.org/10.1016/j.future.2016.10.015.

Xia, Q., Ye, W., Tao, Z., Wu, J. & Li, Q. (2021). A survey of federated learning for edge computing: Research problems and solutions. *High-Confidence Computing*, 1(1), 100008. doi: https://doi.org/10.1016/j.hcc.2021.100008.

Xu, Q., Zheng, R., Saad, W. & Han, Z. (2016). Device Fingerprinting in Wireless Networks: Challenges and Opportunities. *IEEE Communications Surveys & Tutorials*, 18(1), 94-104. doi: 10.1109/COMST.2015.2476338.

Zerveas, G. & et al. (2021). A transformer-based framework for multivariate time series representation learning. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2114–2124.

Zhang, M., Wei, E. & Berry, R. (2021a). Faithful Edge Federated Learning: Scalability and Privacy. *IEEE Journal on Selected Areas in Communications*, 39(12), 3790-3804. doi: 10.1109/JSAC.2021.3118423.

Zhang, P., Jin, H., Dong, H., Song, W. & Bouguettaya, A. (2022). Privacy-Preserving QoS Forecasting in Mobile Edge Environments. *IEEE Transactions on Services Computing*, 15(2), 1103-1117. doi: 10.1109/TSC.2020.2977018.

Zhang, Y., Zhang, P., Luo, Y. & Ji, L. (2020a). Towards Efficient, Credible and Privacy-Preserving Service QoS Prediction in Unreliable Mobile Edge Environments. *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pp. 309-318. doi: 10.1109/S-RDS51746.2020.00038.

Zhang, Y., Zhang, P., Luo, Y. & Luo, J. (2020b). Efficient and Privacy-Preserving Federated QoS Prediction for Cloud Services. *2020 IEEE International Conference on Web Services (ICWS)*, pp. 549-553. doi: 10.1109/ICWS49710.2020.00079.

Zhang, Y., Pan, J., Qi, L. & He, Q. (2021b). Privacy-preserving quality prediction for edge-based IoT services. *Future Generation Computer Systems*, 114, 336-348. doi: https://doi.org/10.1016/j.future.2020.08.014.