Trinity College Dublin

Monitoring & Predicting QoS in IoT Services

Author:
Gary White

Supervisor:
Prof. Siobhán Clarke

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy

in the

School of Computer Science and Statistics
O'Reilly Institute, Trinity College Dublin, Ireland

2020

Declaration of Authorship

I, Gary White, declare that this thesis titled, ‘Monitoring & Predicting QoS in IoT Services’ and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- The library may lend or copy the thesis or any part thereof on request.

Signed: ____

Date: ____

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.

- T. S. Eliot, Four Quartets

TRINITY COLLEGE DUBLIN

Abstract

Faculty of Engineering, Mathematics and Science
School of Computer Science and Statistics
O'Reilly Institute, Trinity College Dublin, Ireland

Doctor of Philosophy

Monitoring & Predicting QoS in IoT Services

by Gary WHITE

Internet of Things (IoT) applications can be built from a number of heterogeneous services provided by a range of devices, which are potentially resource constrained and/or mobile. These device characteristics can lead to the services deployed on them becoming unreliable as the device may lose power or move out of range. As these services and applications continue to be more widespread, a key research challenge is how to make them more reliable. The reliability of an application is influenced by the time to detection (TTD) of a failure and the time to recovery (TTR) of services in the application after the failure. TTD and TTR are affected by the accuracy of the prediction and by the time it takes to receive the prediction. This thesis focuses on reducing TTD by improving the prediction accuracy and reducing TTR by reducing the time it takes to receive the prediction.

Accurate short-term forecasts allow dynamic systems to adapt their behaviour when degradation is detected e.g., transportation forecasting supports alternative routing of traffic before gridlock and wind power forecasting enables the use of dispatchable energy such as hydroelectric power to reduce the difference between power consumption and power generation in the network. This rationale can be applied to service-oriented computing when creating and managing service applications, where such applications are composed of available collaborating services. The faster a problem with a service can be detected, the faster a suitable replacement service can be chosen. Previous approaches that have focused on QoS forecasting have used traditional time series methods, but these are not suitable as QoS does not exhibit traditional time series patterns (i.e., sudden peaks caused by network congestion or a device switching to a power saving mode). More modern recurrent neural network-based approaches such as GRUs and LSTMs have been proposed but the long training time means they take longer to incorporate recent QoS values. This can lead to a reduction in forecasting accuracy in dynamic IoT environments. This thesis proposes a noisy-echo state network approach that has been designed to be deployed at the edge of the network. The reduced training time allows the model to incorporate recent QoS values on devices at the edge. The results show increased forecasting accuracy compared to current state of the art approaches when tested on a combined dataset of IoT and web services, reducing TTD.

Once a problem has been detected with one of the services in a composition, the application needs to be recovered by using a functionally equivalent service with high QoS. Given that candidate services may not be currently executing, predictions based on a time series of current QoS values are not appropriate. Recommending possible replacement services requires a technique that avails of similar users' recent experience of those services, which is the most up-to-date information available about the services' QoS. In service-oriented architectures, collaborative filtering is a key technique for service recommendation based on QoS prediction. Matrix factorisation has emerged as one of the main approaches for collaborative filtering as it can handle sparse matrices (as users invoke only a small subset of the large amount of possible services) and produces good prediction accuracy. However, this process is resource-intensive requiring computational resources not available at the edge of the network. User information must be submitted to a central server or cloud, which can lead to a number of issues related to user privacy and extra network delay when updating the model with new QoS information. As QoS varies over time, it is essential to update the QoS prediction model to ensure that it is using the most recent values to maintain prediction accuracy. The request time, from submitting a user's information to receiving the QoS metrics for candidate services, is also important, as the time to choose replacement services is limited during a dynamic service adaptation. This thesis proposes a stacked autoencoder with dropout on a deep edge architecture, which reduces the training and request time, compared to traditional matrix factorisation algorithms, while maintaining predictive accuracy. To evaluate the accuracy of the algorithms, the actual and predicted QoS values are compared using standard error metrics such as MRE and RMSE. In addition, the predictions are used by a service composition engine to evaluate the impact the predictions have on the response time and throughput of service compositions. The results show a reduction in the time it takes to receive the prediction, reducing TTR.

The combination of improvements to both TTD and TTR allow for increased reliability and more strict levels of QoS. The thesis also highlights how this increased level of quality can lead to new IoT applications such as augmented reality in IoT and quantified-self frameworks for improving personal productivity.

Acknowledgements

This thesis was made possible by the collecitve effort of a group of great people. First and foremost, I would like to thank my supervisor Prof. Siobhán Clarke for giving me the opportunity to do a PhD, and for her guidance, expertise, and continuous support during this process. I would like to thank to my colleagues in the SURF project - Christian Cabrera, Fan Li and Andrei Palade. It has been great to work with you and share your ideas. Many thanks to all the Computer Science Department members who have provided useful advice and feedback during the course of my PhD. I would like to thank my parents and family for their constant love and support. I am grateful for the financial support from Science Foundation Ireland, which has offered me the opportunity to study for a PhD. I am also grateful to Nvidia and Intel for their academic grants to support this work.

Gary White
University of Dublin, Trinity College

Contents

List of Figures

List of Tables

List of Publications

- Gary White, Siobhán Clarke. *"Short-term QoS Forecasting at the Edge for Reliable Service Applications"* in IEEE Transactions on Services Computing, 2020.

- Gary White, Siobhán Clarke. *"Urban Intelligence With Deep Edges"* in IEEE Access, vol. 8, pp. 7518-7530, 2020.

- Gary White, Andrei Palade, Christian Cabrera, Siobhán Clarke. *"Autoencoders for QoS Prediction at the Edge"* in the 2019 IEEE International Conference on Pervasive Computing and Communication (PerCom) Kyoto, Japan, 2019.

- Gary White, Zilu Liang, Siobhán Clarke. *"A Quantified-Self Framework for Exploring and Enhancing Personal Productivity"* in the 2019 Conference on Content-based Multimedia Indexing and Applications (CBMI) Dublin, Ireland, 2019.

- Gary White, Andrei Palade, Christian Cabrera, Siobhán Clarke. *"IoTPredict: Collaborative QoS Prediction in IoT"* in the 2018 IEEE International Conference on Pervasive Computing and Communication (PerCom) Athens, Greece, 2018.

- Gary White, Andrei Palade, Siobhán Clarke. *"Forecasting QoS Attributes Using LSTM Networks"* in the 2018 IEEE International Joint Conference on Neural Networks (IJCNN) Rio de Janeiro, Brazil, 2018.

- Gary White, Christian Cabrera, Andrei Palade, Siobhán Clarke. *"Augmented Reality in IoT"* in the 8th International Workshop on Context-Aware and IoT Services, (CIoTS@ICSOC) Hangzhou, China, 2018.

- Gary White, Siobhán Clarke. *"Smart Cities with Deep Edges"* in the 1st International Workshop on Urban Reasoning from Complex Challenges in Cities (UrbReas@EMCL) Dublin, Ireland, 2018.

- Gary White, Vivek Nallur, Siobhán Clarke. *"Quality of Service Approaches in IoT: A Systematic Mapping"* in the Journal of Systems and Software, 132:186 - 203, 2017. http://www.sciencedirect.com/science/article/pii/S016412121730105X

- Gary White, Andrei Palade, Siobhán Clarke. *"QoS Prediction for Reliable Service Composition in IoT"* in the 2nd International Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities (ISYCC@ICSOC) Malága, Spain, 2017

- Gary White, Andrei Palade, Christian Cabrera, Siobhán Clarke. *"Quantitative Evaluation of QoS Prediction in IoT"* in the 3rd International Workshop on Recent Advances in the Dependability Assessment of Complex Systems (RADIANCE@DSN) Colorado, USA, 2017.

Chapter 1

Introduction

The emergence of a new generation of cheaper and smaller wireless devices with a number of communication protocols has led to the formation of the IoT [1]. The number of these devices is predicted to grow at an exponential rate, with the latest forecasts predicting that there will be around 28 billion connected devices by 2021 [2]. The huge number of devices in the IoT will lead to a wide variety of services that can provide information from a number of sources that traditionally were not connected, such as, surveillance cameras, environment monitoring sensors, smart bins and actuators [3]. This leads to applications in many different domains, such as home automation, industrial automation, medical aids, traffic management and many others [4].

1.1 Service-oriented Architecture

Service-oriented architecture (SOA) is a style of software design in which applications make use of services available over a network [5]. SOA is a loosely coupled arrangement of service providers and service consumers, which means at design time consumers can use any services that fulfil their needs [6]. Although the concepts of SOA were established before web services they made services much more accessible [7]. This is because web services are built on top of well-known and platform independent protocols. These protocols include HTTP, XML, UDDI, WSDL, and SOAP [8]. These protocols fulfil a key requirements of a SOA, that services be dynamically discoverable and invokable [8]. SOA requires that a service have a platform-independent interface contract, this requirement is fulfilled by XML. SOA stresses interoperability, this requirement is fulfilled by HTTP. To manage the complexity of service discovery and composition several middlewares were proposed [9–11].

With an increase in the use of IoT devices, services were not just available from traditional web services, but also from a number of previously unconnected devices [12]. Figure 1.1 shows a small scale scenario, with services provided from different service types including web services (WS), residing on resource rich devices, wireless sensor networks (WSN), which may be resource-constrained and controlled by a software defined network, and autonomous service providers (ASP), who are independent mobile users in the environment with intermittent availability. The services provided by devices in the environment are registered in the gateways and can be used in a number of applications in a variety of different domains [13]. Applications can be created by combining multiple services from different service types. For example, an application might perform some analysis and flood prediction using a water level service available from an autonomous service provider and a machine learning and storage service available as a web service.

To compose and manage the services provided by these devices an IoT middleware is commonly used [14]. To create and manage IoT applications, this thesis uses components from the SURF middleware: a Request Handler (RH), Service Registration Engine (SRE), Service Discovery Engine (SDE), quality of service (QoS) Monitor and Service Composition & Execution Engine (SCEE) [15]. The RH establishes a request/response communication channel with the user and forwards the request to the other middleware components. The SRE registers the available services in the environment. The SDE identifies the functional requirements of the services, which can be used to satisfy the request and sends this list of candidate services to the QoS Monitor. The QoS monitor makes predictions for the non-functional QoS factors such as response time, throughput and availability of the candidate services that can be used to satisfy the request from the SDE. The SCEE then chooses the services with the best QoS that can satisfy the request and executes them. The QoS monitor collects time series data from currently executing services to train a QoS model that can forecast if one of the services is about to degrade in quality. If a service starts to degrade in quality an alert is sent to the SCEE to choose a suitable replacement service.

Service applications can be used in a range of domains that have different QoS requirements based on the sensitivity and criticality of the application. Application QoS can typically be categorised as best effort (no QoS), differentiated services (soft QoS) and guaranteed services (hard QoS) [16]. In the hard QoS case, there are strict hard real-time QoS guarantees. This is appropriate for safety critical applications such as

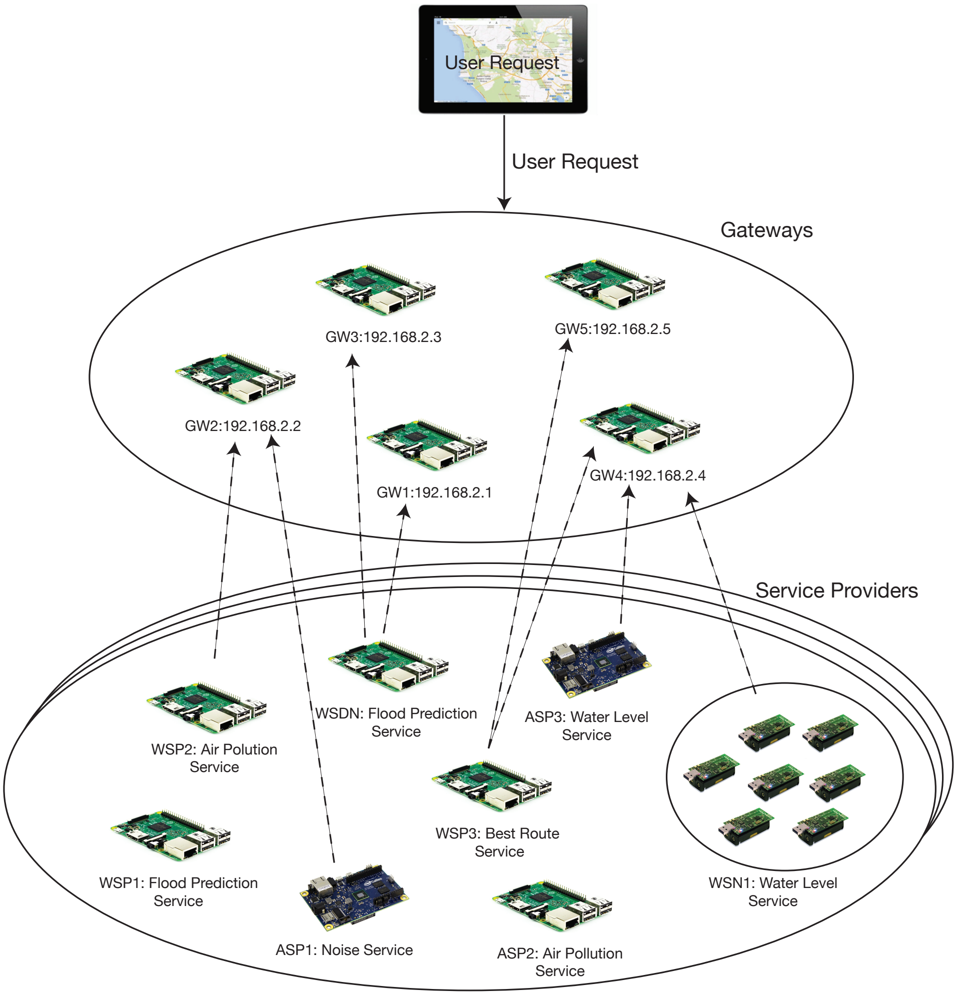


Figure 1.1: Small Scale Scenario

remote surgery in a hospital or collision avoidance in a self-driving car system. Soft QoS does not require hard real-time guarantees but needs to be able to reconfigure and replace services that fail. This could be a routing application that uses air quality, flooding and pedestrian traffic predictions, to provide the best route through the city or an augmented reality application using local weather to provide a more immersive experience. If one of the services is about to fail, the application should be recomposed using suitable replacement services. The final case is best effort, where there are no guarantees when a service fails. This thesis is focused on improving the reliability of soft QoS applications. The notion of reliability in this thesis is the uptime of the IoT application, which we aim to improve by reducing the time it takes to detect an error in one of the services and the time that it takes to choose a suitable replacement service. The QoS parameters that are focused on in the thesis are response time and throughput, but the QoS prediction algorithms can be applied to any quantitative QoS

Urban Application	Data Type	Traffic Rate	Tolerable Delay	Number of IoT Devices	Criticalicty
Waste Management [19]	Historical Data	>= 100 MB per day	> 30 mins	>= 1000-1million per city	Low
Structural Health [19]	Historical Data	>= 10 MB per day	> 30 mins	>= 10-1000 per building	Medium
Air Quality Monitoring [19]	Historical Data	>= 10 MB per day	> 5 mins	>= 1000-1million per city	Medium
Noise Monitoring [19]	Historical Data	>= 100 MB per day	> 5 mins	>= 1000-1million per city	Medium
Wearable IoT	Stream Data	<= 1 GB per device	> 5 mins	>= 1-10 per person	Medium
Traffic Congestion [19]	Historical Data	>= 100 MB per day	> 5 mins	>= 1000-1million per city	Low
Smart Parking [19]	Event Data	>= 10 MB per day	> 1 min	>= 1000-1million per city	Low
Smart Home [19]	Stream/ Massive Data	>= 10 MB per house per day	1 s - 10 mins	>= 10 - 100 per house	Medium
Smart Energy [20]	Stream/ Massive Data	>= 100 GB per day	10 ms - 10mins	>= 1 million per grid	Medium
Remote Surgery [21]	Stream/ Massive Data	>= 50 MB per second	<= 100 ms	>= 1-10 per surgery	High
Augmented Reality [22]	Stream/ Massive Data	>= 100 MB per second	<= 10 ms	> 200,000 globally	Low
Autonomous Vehicles [22]	Stream/ Massive Data	>= 100 GB per vehicle per day	<= 10 ms	>= 50-200 per vehicle	High

Table 1.1: Urban Intelligence Applications

factor.

1.2 Challenges

There are a number of challenges associated with being able to provide soft QoS in an IoT environment. These challenges come from extensive reading of the state of the art, experience with proof of concept implementations and discussions with experts at conferences such as PerCom. This is not a complete list of the challenges in an IoT environment, but the challenges that we felt were most important and the ones that we address in this thesis. There are other challenges in IoT, such as privacy and security that are important but are not the focus of this thesis so are not included in the related challenges:

C.1 Dynamic Environment. IoT devices that act as service providers and gateways can be mobile, especially for urban intelligence applications in smart cities, such as traffic congestion, augmented reality and autonomous vehicles [17]. This dynamic environment adds challenges not found in traditional web services such as service provider devices moving out of range of a gateway, services changing QoS level more frequently and devices changing power modes [18]. The added dynamicity requires the QoS model to incorporate changes in the network much more frequently.

C.2 Reduced Tolerable Delay. Tolerable delay is the maximum acceptable latency for an application between a user requesting a service and getting a response back. Table 1.1 shows a number of urban intelligence applications and their tolerable delay requirements. These requirements have increased over the years with recent applications such as augmented reality and autonomous vehicles requiring tolerable delays of less than or equal to ten milliseconds [22]. This is much more strict than previous generation urban applications, such as waste management or structural health

Characteristics	IoT	Edge	Cloud
Deployment	Distributed	Distributed	Centralised
Components	Physical Devices	Edge nodes	Virtual
Big Data	Source	Process	Process
Computational	Very Limited	Limited	Unlimited
Storage	Very Limited	Limited	Unlimited
Response Time	NA	Low	High
QoS	NA	High	Medium
Energy	Low	Low	High

Table 1.2: Computing Layer Characteristics

that have tolerable delays of over thirty minutes [19]. These traditional applications can be updated and reconfigured in the cloud. However, more recent applications have reduced tolerable delay that it not manageable in the cloud due to the time taken to reach the cloud from IoT devices and should be managed at the edge one hop away from data generation.

C.3 Increased Traffic Rate. Table 1.1 shows the traffic rate for urban intelligence applications [19–22]. There is a dramatic increase in the traffic rate of modern applications such as autonomous vehicles (>= 100 GB per vehicle per day) and augmented reality (>= 100 MB per second) [22] compared to traditional applications such as structural health (>= 10 MB per day) and air quality monitoring (>= 10 MB per day) [19]. The cumulative data rate for even a small fraction of users in a modest-size city would saturate its metropolitan area network: 12,000 users transmitting 1080p video would require a link of 100 gigabits per second; a million users would require a link of 8.5 terabits per second, which is infeasible with current infrastructure [23]. This means we cannot rely on all data and QoS metrics being reported to the cloud and must do some processing on the data at the edge to reduce traffic rate.

C.4 Critical Applications. Table 1.1 also shows the level of criticality of the applications, which indicates the threat to users if an application fails. Applications such as autonomous vehicles and remote surgery have high criticality, which indicates a serious threat to human life if the application fails. Applications such as smart homes and smart energy have medium criticality, which indicates a threat of injury to users if the application fails. The QoS predictions for currently executing services and candidate services must therefore be accurate to avoid not forecasting that a service is about to fail or choosing a replacement service with low QoS that could cause the application to fail.

C.5 Limited Resources. Computing at the edge introduces challenges on the computational power available [24]. Table 1.2 highlights the main differences between IoT, Edge and Cloud computing. In the cloud, there are unlimited computational resources that can easily be scaled up [25]. At the edge of the network, which we define as being one hop away from data generation, these resources are limited and on the actual IoT devices these resources are very limited [26]. This means that QoS algorithms designed to be deployed and updated at the edge of the network need to be less resource intensive to not overload the limited resources.

1.3 Existing Solutions

Approaches for QoS prediction can be broken into two main categories: those that focus on forecasting future QoS values for currently executing services and those that make QoS predictions for candidate services at design time and during a runtime service re-composition. This section presents an overview of the existing solutions for both approaches and highlights the current research gaps. The research questions to be answered in the thesis are then proposed based on this analysis.

1.3.1 Currently Executing Services

A recent study conducted a detailed empirical analysis of time series forecasting for dynamic QoS factors of web services [27], though they do not take into account a number of more recent neural network-based approaches. Time series forecasting techniques can be categorised into two groups: classical methods based on statistical/mathematical concepts and modern heuristic methods based on artificial intelligence algorithms [28]. The former includes exponential smoothing models, regression models, ARIMA models, threshold models and GARCH models [29]. The latter includes artificial neural networks, which is extended to include recurrent neural network (RNN) and long short-term memory (LSTM)-based approaches.

Classical QoS forecasting methods have included traditional AR-based methods/variations, such as SETAR, ARIMA and GARCH models as well as baseline approaches such as linear moving average [30] and persistence models. Two time series methods ARIMA and GARCH have been combined to produce more accurate QoS forecasting results than traditional ARIMA but require extra processing time [31]. Two common time series methods ARIMA and SETARMA were also considered, but the method used depends on the linearity of the predicted QoS time series training data [32].

The incoming time series is tested to detect whether it is linear, in which case the ARIMA method is used otherwise the SETARMA method is adopted. The traditional ARMA model has also been used without the integrated (I) component used in ARIMA [33, 34]. Traditional time series approaches such as ARIMA are not resource intensive and can be deployed and updated at the edge of the network, which makes them suitable for C.1, C.2, C.3 and C.5. However, these models are not designed for sudden peaks caused by network congestion or devices changing power mode. This can lead to increased forecasting errors, which makes them unsuitable to be used for medium critical applications in C.4.

Approaches based on traditional AI algorithms have focused on artificial neural networks (ANNs) but they have not been compared against any established method [35–37]. RNNs have developed in recent years as the most used neural network for time series forecasting as they can utilise past data through feedback paths. Recent experiments using the LSTM architecture developed from RNNs have shown promising results in other domains such as event forecasting [38], acoustic modelling [39] and sequence tagging [40]. Initial attempts to use deep learning for long-term service composition have used generated synthetic data and have been deployed at the cloud layer [41]. These approaches can create more complex models than traditional time series methods and have shown improved forecasting accuracy on synthetic data [41]. However, they have a long training time and are resource intensive, which is why they are typically deployed in the cloud, making them unsuitable for the challenge of reduced tolerable delay, C.2 and the traffic sent to the cloud, C.3. The large training time is also unsuitable for dynamic environments such as IoT as it takes longer to incorporate recent changes in the environment, which can lead to increased forecasting error making them unsuitable for C.4.

1.3.2 Candidate Services

When an anomaly or a degradation in QoS for a particular service is detected by the QoS monitor, a service composition and execution engine changes this service for a functionally equivalent candidate service that can provide an acceptable level of QoS. To switch to an appropriate service, the middleware should have some knowledge of the QoS values of the service. However, if the service has not been invoked before then the middleware will have no QoS values for that service, so will not know if it is a suitable replacement. The service provider can list QoS values in the service description, but these values are not suitable as the user side QoS depends on the time and location of invocation, which cannot be known at the provider side [42]. The

QoS values in service descriptions are often not maintained as it would mean that the service description would need to be constantly updated [27]. One solution is to use a collaborative filtering approach, where QoS values reported from similar users and services are used to make predictions of the missing QoS value for the service [42].

There are two main collaborative filtering methods, which can typically be classified as either memory or model-based. Memory-based approaches store the training data in memory and in the prediction phase, similar users are identified based on the current user. There are a number of approaches that use this technique, including user-based approaches [43], item-based approaches [44] and their combination [45]. Memory-based approaches are resource intensive as they calculate the similarity between all the individual users and services, which makes them unsuitable for C.5. This requires the algorithms to be deployed in the cloud, which increases the delay during a dynamic service composition making them unsuitable for C.2 and the traffic sent to the cloud, making them unsuitable for C.3.

Model-based approaches to collaborative filtering, which employ a machine learning technique to train a predefined model from a training dataset, have become increasingly popular. Latent factor models create a low-dimensional factor model, on the premise that there are only a small number of factors influencing the QoS [46]. Matrix factorisation has emerged as one of the most-used approaches for latent factor models [47], [48]. There has been work on extending matrix factorisation-based collaborative filtering by taking into account alternative sources of information such as time [49] or combining content-based features [50]. There are also approaches that take into account additional factors such as the location of the users in the environment [51], [48]. Other approaches have focused on handling users who are contributing false values using reputation-based matrix factorisation [52]. Initial experiments using deep learning model-based approaches such as Restricted Boltzmann Machines have also been conducted [53]. One limitation of model based approaches is if a new user or service is added to the set, the model has to be updated and trained again [54]. This can be very computationally expensive in a dynamic environment where users and services are mobile, C.5. This requires the algorithms to be deployed in the cloud, which increases the delay during a dynamic service composition and the traffic sent to the cloud, making them unsuitable for C.2 and C.3.

1.3.3 Research Gaps and Observations

Existing solutions for QoS prediction have explored both currently executing and candidate services. For currently executing services traditional time series approaches have been used. These models have short training times so can be deployed and updated at the edge of the network. However, they are not designed to capture the sudden changes and peaks caused by network congestion or devices changing power mode. This can lead to increased forecasting errors. More modern LSTM-based approaches can create more complex models that can capture these sudden changes in the environment. However, these approaches have a long training time and are resource intensive, which is why they are deployed in the cloud, making them unsuitable for the challenge of reduced tolerable delay. The large training time is also unsuitable for dynamic environments such as IoT as it takes longer to incorporate recent changes in the environment, which can lead to increased forecasting error.

For candidate services, memory-based approaches are resource intensive as they require the calculation of similarity between all individual users and services. This requires the algorithms to be deployed in the cloud, which increases the delay during a dynamic service composition and the traffic sent to the cloud. Model-based approaches such as matrix factorisation are also resource intensive as they require the generation of latent features to construct the model. These approaches are not suited for dynamic environments as they have to be updated if a new user or service comes in to the environment.

The existing solutions for currently executing and candidate approaches have focused on deploying and training algorithms in the cloud due to the resource intensiveness of the algorithms. There is a research gap to to provide the tolerable delay required by modern urban intelligence applications in Table 1.1 at the edge of the network. Deploying these algorithms at the edge rather than in a cloud environment also reduces the time taken to receive QoS predictions as well as the traffic sent to the cloud [55]. This can lead to less failures especially during dynamic service re-composition, where suitable replacement services must be chosen quickly before the application fails [56].

To provide the tolerable delay demands in modern urban applications there is a requirement for the development of new, accurate QoS prediction algorithms that provide:

1) Accurate QoS forecasting of currently executing services at the edge. This allows a service composition and execution engine to know when a service is about to fail with reduced delay, increasing the time to find a suitable replacement service.

2) Accurate QoS prediction of replacement services at the edge. This allows any services that do fail to be quickly replaced at the edge of the network during a dynamic service adaptation.

1.3.4 Research Questions

This thesis explores the question of how to enable reliable IoT applications in a modern dynamic pervasive computing environment. This question can be decomposed into the time to detect (TTD) an error in one of the currently executing services and the time to recovery (TTR) of the application after one of the services has failed by choosing a suitable replacement service:

RQ.1 To what extent can the accuracy of forecasting to support TTD be improved, by using a lightweight model at the edge to incorporate recent changes in QoS?

RQ.2 To what extent can the time to receive predictions of TTR be reduced, by updating a model at the edge of the network, while maintaining QoS prediction accuracy?

1.4 Thesis Approach

The reliability of service applications can be affected by TTD, TTR and the time to failure (TTF) of the service providers. At the middleware level there is control over TTD and TTR. However, TTF is influenced by the service providers and the amount of resources that they make available for reliable execution of their services [57]. This thesis focuses on reducing TTD and TTR to enable the use of IoT services in modern applications with low tolerable delay.

Assumptions The following assumptions are made about the IoT environment and the QoS prediction algorithms:

A.1 The operating environment for QoS prediction is dynamic with QoS factors changing due to congestion in the network or the movement of mobile gateways and services. This means that the QoS models must update regularly to incorporate new changes in the network.

A.2 Algorithms must be deployed and updated on edge devices. This is required to manage the strict tolerable delay demands of <= 10 ms of modern applications such as augmented reality [22].

A.3 The functional requirements of the services are handled by a service composition and execution engine, which uses a goal-driven approach to create a service dependency graph with multiple possible candidate services [58]. The TTR algorithm is used to make QoS predictions for the initial composition and the TTD algorithm is used to monitor and forecast any errors in the currently executing services. If one of the services begins to degrade in quality, it is detected by the TTD algorithm, which sends an alert to the service composition and execution engine to select a replacement service using the QoS factors provided by the TTR algorithm.

Hypothesis This thesis investigates how to increase the reliability of IoT service applications by first decomposing reliability into its constituent parts of TTD, TTR and TTF. The main body of the thesis then focuses on improving the reliability by reducing the TTD and TTR components of reliability. The hypothesis is that the development of algorithms at the edge of the network with reduced training time will be able to update more frequently to capture the dynamic QoS in an IoT environment. A 'deep edge' architecture is also introduced to allow for faster training of neural networks at the edge of the network. The deep edge architecture is a combination of deep neural networks and edge networks deployed one hop away from data generation. The ability to make accurate QoS predictions at the edge will reduce both TTD and TTR, leading to an improvement in the reliability of service applications.

Objectives Existing proposals for QoS prediction have focused on deploying algorithms in the cloud. This work's high level research objective is to investigate QoS prediction mechanisms that can address the limitation of these systems as follows:

O.1 The forecasting accuracy in TTD is important as making an incorrect prediction that a service is about to fail when it is not causes a lot of additional processing to find a replacement service. Predicting that a service is not about to fail when it will is also damaging, causing the application to fail without a suitable replacement service. This work aims to improve the forecasting accuracy of TTD to ensure that the service is only replaced when it is about to fail or degrade in quality.

O.2 Quick recovery and replacement of services during a dynamic service adaptation increases the chance of a service being replaced before a user notices a drop in quality. This work aims to reduce the request time of TTR by deploying an accurate QoS prediction algorithm for candidate services at the edge of the network, while maintaining prediction accuracy.

O.3 With increased demands in tolerable delay there is a need for smart algorithms and analysis at the edge of the network one hop away from data generation. Deep learning algorithms have won numerous competitions in pattern recognition and machine learning [59], but are resource intensive making them difficult to deploy at the edge. This work aims to develop a suitable 'deep edge' architecture for deploying and updating deep learning algorithms at the edge that can be used to reduce TTD and TTR.

1.5 Thesis Contribution

This thesis presents a method to improve the reliability of IoT applications at the edge by focusing on reducing the **T**ime **T**o **D**etect an error and the Time to **R**ecover from that error (TTDR). The first part of TTDR forecasts when currently executing services may be about to degrade in quality or fail using time series information about currently executing services available at the gateway. The second part of this approach focuses on making accurate predictions for replacement services at the edge using previous execution information available at the gateway. A deep edge architecture is implemented to allow for the deployment and training of deep learning models one hop away from data generation. This architecture uses embedded GPUs (Jetson Tx2) as gateways to register the services.

This thesis describes an in-depth study into the area of reliability in IoT applications and makes the following contributions to knowledge:

Noisy-echo state networks (RQ.1) Existing TTD approaches such as RNNs have long training times caused by having to train all connected internal weights including input-to-RNN, RNN-internal and RNN-to-output weights. This thesis proposes a noisy-echo state network architecture approach that has been designed to reduce training time, while allowing the model to incorporate recent QoS values on devices at the edge. This is a specific reservoir architecture where only the RNN-to-output weights of the network are trained, which greatly reduces the training time. This allows the model to quickly retrain on an edge device, enabling it to incorporate any recent changes in the service quality. Our results show increased forecasting accuracy compared to state of the art approaches when tested on IoT and web services datasets. This reduces TTD, increasing the overall reliability of the application.

IoTPredict (RQ.2) Existing TTR matrix factorisation-based approaches have used Pearson's Correlation Coefficient to calculate the most similar users and services. The IoTPredict approach proposes a novel neighbourhood-based prediction approach for the IoT, which uses an alternative similarity computation mechanism. The accuracy of the algorithms is evaluated by comparing the actual and predicted QoS values using standard error metrics such as MRE and RMSE, which IoTPredict shows improved accuracy. In addition, an alternative evaluation technique using the predictions as part of a service composition and measuring the impact that the predictions have on the response time and throughput of the final composition is also used.

Stacked autoencoder (RQ.2) Existing TTR approaches have focused on improving the QoS prediction accuracy by deploying resource intensive algorithms in the cloud. This thesis focuses on reducing the training time for candidate service QoS predictions, which allows us to analyse the QoS for users in a highly dynamic environment such as IoT. A stacked autoencoder approach is designed to be executed on edge devices to reduce the time to receive predictions, while maintaining high predictive accuracy. The stacked autoencoder approach is evaluated in the same manner as IoTPredict. The results show a reduction in the request time of TTR, increasing the overall reliability of the application.

1.6 Thesis Scope

QoS factors such as response time and throughput are end to end, which means that every layer of the IoT architecture has an impact on the final values. Delays at the device level, network level and middleware level are cumulative and increase the final delay for the user. A large amount of work has been conducted at the network level in IoT [60]. This thesis focuses on providing QoS support at the middleware level of the IoT architecture. This involves making forecasts for when a service may be about to fail and making QoS predictions for services at design time and during a runtime dynamic adaptation.

This thesis focuses on provided soft QoS, which does not require hard real-time guarantees but needs to reconfigure and replace services that fail. There are approaches that focus on providing hard QoS with real-time guarantees for domains such as nuclear power [61] and avionics [62]. These approaches introduce a large amount of overhead with additional redundancy needed to provide this level of quality [63]. These approaches can also only work in limited environments where devices are connected by wired networks [64]. This is not suitable for a typical IoT environment where devices and gateways may be mobile.

One way to increase the reliability of IoT service applications would be to stop the services that they use failing as much. This is difficult in a dynamic and mobile environment, but increasing the time to failure of services would reduce the number of times that the application would need to forecast a failure and therefore the amount of times the middleware would need predictions for replacement services during a dynamic service adaptation. Approaches have tried to increase this value by resource provisioning [57] and optimising service placement [65]. This work focuses on forecasting and replacing these services rather than improving the time to failure of the individual services.

There are many different architectures for managing IoT services with a growing number of approaches using the cloud [66]. With the decreasing tolerable delay as shown in Table 1.1 for modern applications such as augmented reality and autonomous vehicles it is necessary to have the algorithms deployed close to the service at the edge of the network. This work focuses on algorithms that can be trained at the edge of the network to allow for accurate QoS predictions and the required tolerable delay.

1.7 Thesis Structure

The rest of the chapters are structured as follows:

2. State of the Art analyses current state of the art approaches for QoS prediction. It presents a systematic mapping across the layers of the network to show the end to end nature of QoS metrics. A detailed analysis of current QoS prediction approaches at the middleware layer is then presented where research gaps are identified, in particular focusing on predicting QoS in currently executing and candidate services.

3. Design describes the design objectives and required features of this thesis according to the challenges identified in this chapter. The overall structure of the approach to increase the reliability of IoT applications by reducing TTD and TTR through the use of noisy-echo state networks and stacked autoencoders is then described in detail.

4. Implementation details the implementation of the algorithms to reduce TTD and TTR and the middleware components they are designed to be used with. The deep edge architecture used to deploy the algorithms one hop away from data generation to reduce the prediction delay is also described.

5. Evaluation compares how accurate the prediction algorithms are to current state of the art approaches. The chapter first describes the experimental setup and metrics used to evaluate the prediction accuracy and then presents the results of the experimental evaluation.

6. Conclusion summarises the thesis and demonstrates some of the applications that are now capable of executing correctly with accurate QoS predictions and low tolerable delay. Future work that can build on top of the thesis is also highlighted.

1.8 Chapter Summary

This chapter introduces the context of this research, together with the limitation of existing works, the thesis approach and the thesis contributions. Creating reliable applications with low tolerable delay is a necessary problem with the development of recent applications such as augmented reality. Existing proposals for reliable service applications have focused on web services and algorithms deployed in the cloud, which are unable to manage the strict tolerable delay demands needed for these modern applications. This thesis proposes TTDR, a combination of improvements in the time to detect an error and the time to recovery from that error on a deep edge architecture to improve the overall reliability of IoT applications.

Chapter 2

State of the Art

QoS is a complex term that contains a number of individual factors such as response time, throughput and availability [67]. QoS factors such as response time are end-to-end and a delay in any layer from the device to the user receiving the response is cumulative. Section 2.1 presents the results of a systematic mapping giving a high-level overview of state of the art approaches that have been conducted across different layers of the IoT architecture. From this initial mapping, the following sections focus on the middleware layer where research gaps were identified. Section 2.2 gives an overview of reliability describing the trade-offs between approaches such as state-based and user-centric reliability. The remaining sections focus on the specific methods to improve reliability. Section 2.3 focuses on approaches to forecast currently executing services and Section 2.4 focuses on approaches for QoS prediction of candidate services.

2.1 Mapping QoS in IoT

To make QoS predictions of services for users, QoS factors need to be monitored and reported [68]. There are a number of quality factors that can be monitored in IoT services. Figure 2.1 shows the ISO/IEC software quality model for software products. This gives a detailed description of the quality categories and factors that can be monitored in a typical software product. These quality factors are useful as they allow users to specify desired behaviour such as increased availability or better response time during a service composition. However, a number of quality factors such as response time are cross layer as a delay in any layer will affect the final response time. In this section, we present a structured literature review using the systematic mapping process [69, 70]. This structured literature review allows us to map QoS approaches across the

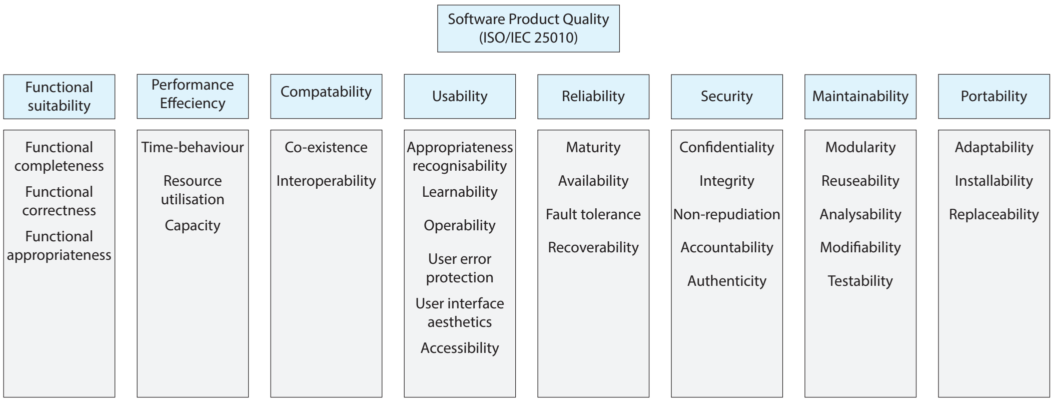


Figure 2.1: ISO/IEC 25010 Proposal of Quality Model for Software Products

layers of the IoT architecture to identify areas that previous approaches have focused on and point out areas that need more attention [71].

2.1.1 Search Design

A combination of search strategies were used including web searches in academic databases and a snowballing process [72], where references from the papers selected by full text reading are included in the search and suitable papers are added to the final selection. The papers are first selected from the largest academic databases in Computer Science [73, 74], using the search syntax in Table 2.1, which shows the results for each of the individual searches. The search syntax is constructed using a combination of PICO (Population, Intervention, Comparison and Outcome) and keywords from high quality papers. As identified in other systematic mappings the PICO method is not always fully applicable [70]. In our case, we retrieve keywords from the population and intervention research questions. Population refers to the specific application area that we are interested in, which is this case is an IoT environment. The intervention of this search refers to a procedure, software methodology or tool that has been used in the context of this study, which in this case is Quality of Service or QoS or Monitoring.

After retrieving papers from the initial search procedure the standard systematic mapping process is applied to filter out irrelevant candidates, which can be seen in Figure 2.2. The first stage was to remove duplicate papers in the search results from the academic databases, this was automatically handled by the reference manager, Mendeley. The next stage was the selection of papers by title, this was used to quickly remove articles whose scope was unrelated to QoS in IoT. Papers were then selected by abstract, removing papers that did not present a quality approach as a contribution.

Database	Search Syntax	Results
IEEE	(QoS OR .QT.Quality of Service.QT. OR .QT.Monitoring.QT.) AND (.QT.Internet of Things.QT. OR IoT)	1,611
Science Direct	((((qos or ”quality of service” or ”monitoring”)) WN KY) AND (((IoT or ”Internet of Things”)) WN KY))	71
SCOPUS	TITLE-ABS-KEY((qos or ”quality of service” or ”monitoring”) and (IoT or ”Internet of Things”))	1,067
WOS	(TS=((qos OR ”quality of service” OR ”monitoring”) AND (iot OR ”Internet of Things”)))	383
Engineering Village	((((qos or ”quality of service” or ”monitoring”)) WN KY) AND (((IoT or ”Internet of Things”)) WN KY))	2,395

Table 2.1: Searches in Databases

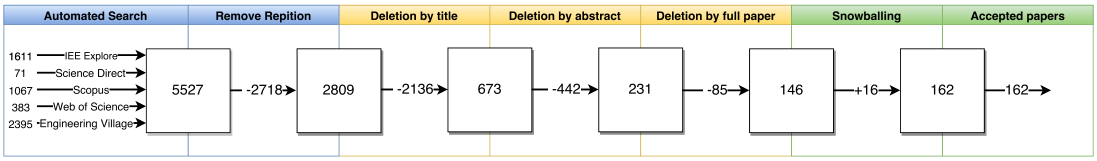


Figure 2.2: Selection of the Mapping Articles

The full papers of the remaining results are then read and the papers that present a quality approach and the definition of the quality factors are selected to be used in the mapping. The final stage is to add additional references from the selected papers into a final list of accepted papers through the snowballing process [72]. Figure 2.2 shows the results at each stage of this process, with 162 papers being selected from an initial selection of 5527.

2.1.2 Summary of Findings in Systematic Mapping

Figure 2.3 shows the distribution in approaches through the layers of the IoT architecture. The focus area is used to structure the topic in the layers of the IoT architecture. The research approaches are classified using an established classification approach to extract the contribution and research facets of the approaches that were used in the mapping [75]. The left side of the figure shows the contribution of each article (Tool, Method, Process, Model, Metric). The right side of the figure shows the research facet of each article and identifies the research method used (Evaluation, Validation, Solution Proposal, Philosophical, Experience Report, Opinion) [75]. The results are interesting for both facets and it can be identified that there has been a focus on the middle communication layers of the architecture from the physical layer to the network layer.

Looking specifically at the contribution facets in Figure 2.3 it can be seen that process contributions, which present architectural contributions have been the most popular. However, at specific layers such as the network layer approaches have been much more diverse, with contributions of models, tools and metrics. The contribution of models and metrics are useful as they allow comparison between different approaches, such as different MAC approaches [76, 77]. There have been fewer tools built as many of the proposals are initial solution proposals, however there are some exceptions with an approach providing a toolset for managing QoS in IoT cloud systems [78].

The research facets in Figure 2.3 show that many of the approaches have been solution proposals, which have used initial validations rather than evaluations in a realistic environment [75]. The most popular method of validation is by simulation either through the use of an established simulation environment such as NS2 [79, 80] or by creating a simulation using a program such as Matlab with given parameters [81, 82]. Other research facets such as philosophical, opinion or experience papers can provide an alternative perspective on current research approaches, how the field is structured or practical experience from a real implementation. This can be just as valuable as a solution proposal, but there has been a lack of these research facets through the layers of the IoT architecture. The systematic mapping gives a high level overview of a number of approaches used to provide QoS throughout the layers of the IoT architecture and the contribution and research facets used. Having identified a research gap at the middleware level, a more detailed analysis of state of the art QoS approaches focusing on a specific QoS factor is conducted. Reliability was the most mentioned quality factor in the systematic mapping [71] and is identified as one of the most important characteristics of an IoT system [83], so a more detailed analysis of approaches providing this factor is conducted at the middleware level.

2.2 Reliability

Reliability refers to the proper working of the system based on its specification [84]. Reliability must be implemented in software and hardware throughout all the IoT layers as a delay or failure in any layer can cause a delay or loss of data for the application [85]. Section 2.1 shows how many QoS approaches have focused on the communication layers of the IoT network with a lack of focus on some layers such as deployment and middleware. This section takes a closer look at reliability in the middleware level. Reliability is the likelihood that a system will remain operational for the duration of the task and is a means of representing fault tolerance [86]. Research in enterprise applications refer to fault tolerant approaches, which are included in the analysis in

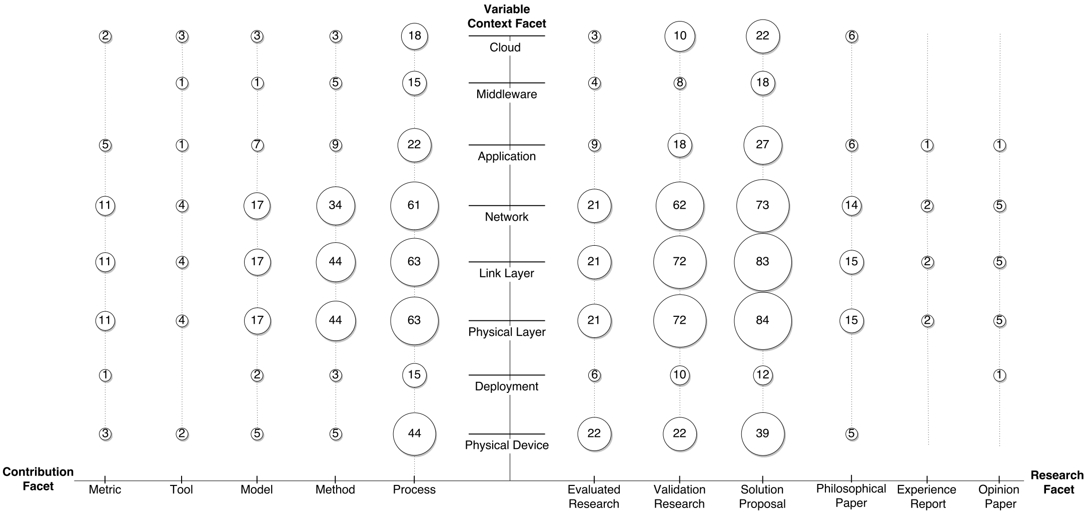


Figure 2.3: QoS Mapping of Layers of the IoT Architecture

Section 2.2.1 [87–89]. State-based approaches described in Section 2.2.2 build up a markov chain of the possible failure patterns, while user-centric approaches described in Section 2.2.3 focus on monitoring the actual QoS from services and reacting to changes.

2.2.1 Fault Tolerant Approaches

Fault tolerance mechanisms are an important component of fault life-cycle techniques, which have an important role in the reliability of enterprise SOA [90]. These mechanisms deal with situations where failures or faults occur during live execution in an enterprise application and provides the ability to keep the system performing its function correctly. Enterprise SOA can have strict requirements about the level of reliability needed, for example the Amazon Compute service level agreement has a monthly uptime percentage of greater than 99.0% [91]. To handle these faults in a timely manner, a fault tolerant system can use either spatial or temporal redundancy, including replication of hardware (with additional components), software (with special programs), and time (with the diversification of operations) [87]. With significant advancement in the field of distributed computing, the amount of dynamic enterprise distributed applications have been substantially increased because of flexible architectures [92]. Replication in terms of data can be an effective technique especially when the data consists of multimedia objects [93]. Enterprise SOA can provide high levels of reliability by adding additional resources and replication, which may not be available for users in an IoT environment.

Software fault tolerance is widely employed for building both reliable stand alone systems as well as distributed systems [88]. The major software fault tolerance techniques include recovery block [94], N-Version Programming (NVP) [95] N-Self checking programming [89] and distributed recovery block [96]. In the area of service-oriented computing, the cost of developing redundant services is greatly reduced, since functionally equivalent services can be employed for building diversity-based fault-tolerant service-oriented systems [57, 97, 98]. These services can be easily deployed on multiple devices. These redundant services can be switched to in a dynamic fashion at runtime [99]. A rigorous development process can be used to build a reliable connector, which is a critical component used to insert detection actions (e.g., runtime assertions) and recovery mechanisms (based on various replications strategies) [100]. Within the connector, lots of fault tolerance strategies can be implemented (e.g., active or passive replication strategies). Passive strategies have been discussed in FT-SOAP [100], FT-CORBA [101], and in Region-KNN [102]. Active strategies have been investigated in FT-Web [103], Thema [11], WSReplication [104], and Perpetual [105]. These connector-based strategies have been deployed using web services with built-in replication mechanisms to perform error recovery, these replication mechanisms are not available in typical IoT services.

Another subset of fault tolerant approaches are those that aim to provide hard QoS with real-time guarantees for high critical domains such as nuclear power [61] and avionics [62]. These approaches introduce a large amount of overhead with additional redundancy to provide this level of quality [63, 106]. Fault diagnosis systems based on control theory are widely applied to guarantee the safety of nuclear powerplants, but these methods require highly detailed analysis of the reliability of each component that will be used in the system [107]. These approaches are also designed to work in limited static environments where devices are connected by wired networks [64]. This is not suitable for a typical IoT environment where devices and gateways may be mobile and connected using wireless networks.

2.2.2 State-based Approaches

A Markov chain process is a discrete random process that undergoes transition from one state to another in a chainlike manner, where every state depends on the current state and not on the entire system [108]. In state-based reliability approaches, the probabilistic control flow and reliability states are mapped to a state space model that can undergo transition from one state space to another [109]. Various methods have used a Markov chain process to analyse or predict the reliability of service-based applications by computing the time frame for each job in execution [110]. It is used to determine whether the selected task has successfully completed its job in the specified time frame or not. State-based approaches have also been used for estimating the reliability of heterogeneous architectures consisting of batch-sequential/pipeline, call-and-return and fault tolerance styles [111]. Reliability of the overall application is dependent on the reliability of the individual services as a delay or failure in any of services affects the final result of the application.

To manage errors such as time-out failures, blocking failures, network failures, etc., which can occur during service invocations, a hierarchical model can be used [112]. This hierarchical model suggests tackling various errors in different layers using the Markov state principle to map layers into different physical states. Markov models, queuing theory, graph theory, and Bayesian analysis have been used to predict reliability performance by dealing with blocking, time-out, matchmaking, network, program and resource failure in hierarchical manner [113–115]. Errors arising in different layers can be predicted by specifying certain predefined criteria such as, in case of the request layer, if the number of requests are larger than the length of request queue, this will lead to overflow [116]. Systems with time constraints are critical because they have to complete their assigned task or job within the defined time frame. Calculating the time frame for each job in execution, network time and processing time of the job are key indicators, which can be used.

These models have been used to develop fault tolerant computing systems by detecting faults and failures in resources before their implementation and recovery to allow execution to continue without being crashed [117]. There are limitations to using this approach in a dynamic IoT environment as they have problems dealing with large scale, heterogeneous and dynamic environments as this leads to a huge increase in the possible number of states. As large scale, heterogeneous and dynamic environments are key characteristics of IoT they would be unsuitable for these applications [118].

2.2.3 User-Centric Approaches

Systems behaviour can change under different circumstances and various factors that are outside the control of service providers such as unpredictable network speed and variable communication links can affect the reliability performance of even the same set of services [119]. There are a lot of variations produced in predicting performance reliability under state-based and architecture-level prediction approaches. There are some other measurements that can be made in this regard, which can be termed as user-centric [42]. User-centric approaches can be characterized as multi-stage problem-solving processes where the system is conceived in terms of user behaviour. As the reliability of any system has a direct impact on the system usage, these models predict reliability considering all measures for both types of stake holders i.e., service users and vendors [120]. User-centric reliability is a data-driven approach that uses the QoS metrics that a user receives by interacting with system [121].

The WS-DREAM model is a user-centric approach based on collaborative assessment mechanism to assess reliability of SOA-based applications by providing a real time test environment for testing the service reliability of various users in different geographical dispersed locations [42]. Here, all users can share their results making it easier to assess recent QoS values in a distributed environment. This type of approach is normally termed as Black Box Testing, where users are concerned with the end result only [122]. Users are incentivised to report these values as they will be able to receive QoS predictions for services that they have not invoked using the similarity between themselves and other users. The more they use the system the more accurate the predictions that will be generated as there will be more data points to calculate similarity with other users and services.

2.2.4 Summary

We summarize the advantages and limitations of the reliability approaches in Table 2.2. To cope with the dynamic changes in an IoT environment we propose using a user-centric approach as it allows sharing of QoS information from other similar users in the environment, rather than trying to construct a state-based approach to model failure in resources. IoT is a dynamic environment, with mobile devices and gateways. It is difficult to model how users and gateways will move in the environment and the impact this will have on the quality of service. It is also not possible to use the fault tolerant approaches in enterprise SOA as IoT relies on the services provided through service providers and users cannot demand them to add additional redundancy or replication. Having identified a research gap at the middleware level and completed a more detailed analysis of reliability, we propose a user-centric approach at the middleware layer to improve reliability. Section 2.3 provides a detailed analysis of user-centric approaches for currently executing services and Section 2.4 analyses the user-centric approaches for candidate services QoS prediction.

Approach	Advantage	Limitation
Fault tolerant approaches	Very high reliability	Increased overhead Additional redundancy Strict requirements
State-based approaches	High reliability with given states	Number of states can grow exponentially Difficult to predict state transitions
User-centric approaches	Provide high reliability Robust to sudden changes	Require a large amount of user data

Table 2.2: Reliability Summary

2.3 Executing Services

For currently executing services, the task is to identify when a service is about to fail or degrade in quality. The basic step of predicting future QoS values can be thought of as a time series problem, where the next value in the series must be predicted [123]. The goal is to forecast the next value in the time series to identify that a service may be about to fail or degrade and send an alert there is a need to switch to an alternative candidate service [124].

2.3.1 Benchmark Methods

The persistence (naive) method is the simplest forecasting method in time series research and is used as a baseline model [125]. The model uses the last observed time series value y_i as its forecast y_{i+1} of the next value in the predicted time series.

$$y_{i+1} = y_i \tag{2.1}$$

The average model is a simple extension of the persistence model that takes into account the most recent n values.

$$y_{i+1} = \frac{1}{n} \sum_{x=i-n}^{i} y_x \tag{2.2}$$

Both these models are very simple requiring no training time, which makes them suitable to be deployed on limited resources (C.5) and to quickly incorporate recent changes in the network (C.1). However, the models are not capable of capturing complexity in the time series and produce poor forecasting accuracy, not suitable for critical applications (C.4) [126].

2.3.2 ARIMA Methods

AutoRegressive Integrated Moving Average (ARIMA) models were originally proposed by Box and Jenkins [127] to model time series data and forecast their future values. They have been used for a wide range of time series forecasting applications, such as socio-economic forecasting [128], car flow forecasting [129] and water quality prediction [130]. ARIMA models for forecasting consist of three components: Auto-Regressive (AR), Integrated (I) and Moving Average (MA). These parts can be used together or independently to perform time series forecasting. For each component, there is a number (p for AR, d for I and q for MA) that must be determined for an ARIMA model to make forecasts. Suitable values are determined by analysing the past training data to determine if integration (I) is needed to make the time series stationary (mean and variance do not change over time). Once the time series has been made stationary the AR and MA lags can be calculated by constructing an Autocorrelation and Partial Autocorrelation plot and choosing a suitable number of lags. This process can be automated by performing a grid search over component values on a sample of the training dataset, which is how the p, d and q values are chosen in Chapter 5.

If no integration is required the time series y_t is said to be modelled by an ARMA model of orders p and q, denoted by ARMA(p,q), if it satisfies:

$$\phi_p(B)y_t = \theta_q(B)\varepsilon_t \tag{2.3}$$

where, ε_t is a sequence of independent normal errors with zero mean and variance σ^2. The backshift operator B is defined as $B_{xt} = x_{t-1}$. The autoregressive polynomial is $\phi_p(B) = (1-\phi_1 B-\phi_1 B^2-\cdots-\phi_p B^p)$ with order p and $\theta_q(B) = (1+\theta_1 B+\theta_1 B^2+\cdots+\theta_q B^q)$ is the moving average polynomial with order q. The autoregressive and moving average coefficients are $\phi = (\phi_1, \phi_2, \ldots, \phi_p)^T$ and $\theta = (\theta_1, \theta_2, \ldots, \theta_p)^T$ respectively. The equation 2.3 can be simplified and written:

$$y_{i+1} = \sum_{i=0}^{p} \theta_i y_{t-1} + \sum_{i=0}^{p} \theta_i \varepsilon_{t-1} + \varepsilon_t \tag{2.4}$$

where, y_{i+1} is the next step in the time series, p are the past stationary observations, ε_t is the current error and ε_{t-i} for $i = 1, \ldots, q$ are the past errors. If d differences have been used to transform the original non-stationary time series z_t into the stationary one y_t, then z_t is said to be generated by an Autoregressive Integrated Moving Average (ARIMA) model of orders p, d, and q and denoted by ARIMA(p,d,q). The main assumptions of ARIMA models are serial dependency, normality, stationary and invertibility. Investigating whether the given time series data satisfies these assumptions is a crucial task, as unsatisfied assumptions lead to incorrect ARIMA models that in turn will provide incorrect forecasts. These assumptions are validated using the training data when creating the model.

ARIMA methods are lightweight, which makes them suitable to be deployed on limited resources (C.5) and to quickly incorporate recent changes in the network (C.1). However, these models have been used in traditional time series forecasting tasks and may not capture the sudden dynamics QoS in IoT cause by mobile devices, which can lead to decreased forecasting accuracy (C.4).

2.3.3 Holt-Winters

Holt-Winters is an exponential smoothing technique that uses data in an exponential window function [125]. It has been used in a wide variety of forecasting applications, such as air passenger forecasting [131], electrical demand forecasting [132] and sales forecasting [133]. Similar to the average method, the forecasting result is a weighted summation of past time series observations. However, in contrast to the average method, where the weights are identical for all observations or the persistence approach that only uses the last observation, Holt-Winters weights exponentially decay as the observations get older. Thus, more recent observations have larger weights and contribute more to the forecast result. The Holt-Winter method has two variations: the additive model and the multiplicative model, each of which forecasts time series with different properties. The additive method is preferred when the seasonal variations are roughly constant through the series as is the case for QoS data. The Holt-Winters seasonal method comprises the forecast equation and three smoothing equations - one for the level l_t, one for the trend b_t, and one for the seasonal component s_t, with corresponding smoothing parameters α, β^* and γ. m is used to denote the frequency of the seasonality, i.e., the number of seasons in a year.

$$y_{i+1} = l_t + b_t + s_{t-1} \tag{2.5}$$

$$l_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \tag{2.6}$$

$$b_t = \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \tag{2.7}$$

$$s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \tag{2.8}$$

The level equation shows a weighted average between the seasonally adjusted observation $(y_t - s_{t-m})$ and the non-seasonal forecast $l_{t-1} + b_{t-1}$ for time t. The trend equation is identical to Holt's linear method. The seasonal equation shows a weighted average between the current seasonal index, $(y_t - l_{t-1} - b_{t-1})$, and the seasonal index of the same season (i.e., m time periods ago).

Holt-Winters models are lightweight, which makes them suitable to be deployed on limited resources (C.5) and to quickly incorporate recent changes in the network (C.1). However, they have large testing time, which can increase the tolerable delay (C.2). They also have quite simply model complexity, which may not be able to capture the dynamics of the environment leading to decreased forecasting accuracy (C.4).

2.3.4 Gated Recurrent Units (GRU)

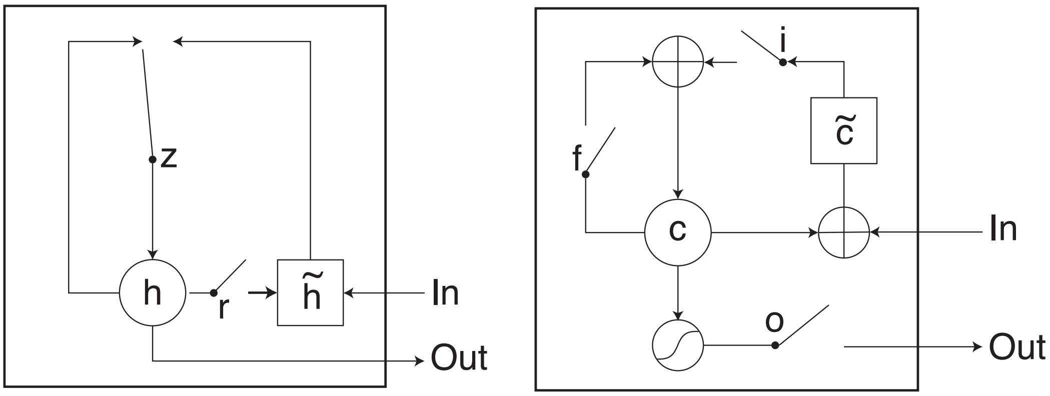
Artificial Neural Networks, especially recent developments in deep learning [134], have had a large impact on a number of different research areas from diagnosing cancer [135] to speech recognition [136]. They have also been shown to be useful for time series forecasting from a wide variety of domains [137], [138], [139], [140]. These approaches can approximate any function regardless of its linearity but can require a large amount of training data.

Gated recurrent units (GRU) have developed from recurrent neural networks (RNNs), which have been previously applied to various sequence learning tasks such as natural language processing, gas market models and speech signal modelling [141–143]. Despite their early success, the difficulty of training RNNs led to proposals to improve their basic architecture [144]. GRU and LSTM networks are the most successful variants of the traditional RNN architecture. GRUs have developed from the basic architecture of RNNs by modifying the functional form to make each recurrent unit adaptively capture dependencies on different time scales [145]. The GRU has gated units that modulate the flow of information inside the unit. This process is illustrated graphically in Figure 2.4a.

The activation h_t^j of the GRU at time t is a linear interpolation between the previous activation hj_{t-1} and the candidate activation $h_t^{\tilde{j}}$:

$$h_t^j = (1 - z_t^j)hj_{t-1} + z^j h_t^{\tilde{j}} : \tag{2.9}$$

where an update gate z_t^j decides how much the unit updates its activation, or content. The update gate is computed by:



(A) Gated Recurrent Unit (B) Long Short-Term Memory

FIGURE 2.4: Illustration of (a) GRU (b) LSTM (a) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation. (b) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content.

$$z_t^j = \sigma(W_z x_t + U_z h_{t-1})^j \tag{2.10}$$

The GRU, does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time. The candidate activation $\tilde{h_t^j}$ is computed similarly to that of the traditional recurrent unit [146]:

$$\tilde{h_t^j} = tanh(Wx_t + U(r_t \odot ht - 1))^j \tag{2.11}$$

where r_t is a set of reset gates and $\odot$ is an element-wise multiplication. When off (r_t^j close to 0), the reset gate effectively makes the unit act as if it is reading the first symbol of an input sequence, allowing it to forget the previously computed state. The reset gate r_t^j is computed similarly to the update gate:

$$r_t^j = \sigma(W_r x_t + U_r h_{t-1})^j \tag{2.12}$$

This process is illustrated graphically in Figure 2.4a.

GRUs are able to create complex models that can capture the dynamics of QoS in IoT and produce accurate forecasts (C.4), however they can also be resource intensive to train, which can be a challenge on devices at the edge (C.5). This can also lead to delays in incorporating recent changes in dynamic environments (C.1), which can decrease forecasting accuracy (C.4).

2.3.5 Long Short-term Memory (LSTM)

Long short-term memory networks (LSTMs) have also developed from recurrent neural networks (RNNs) with improvements on the initial architecture. They have been used in a wide variety of applications such as precipitation nowcasting [147], solar power forecasting [148, 149] and wind forecasting [150]. The initial RNN architecture was improved on by storing and retrieving information over a long period of time with explicit gating mechanisms and a built-in error carousel [143, 151]. This addressed the difficulties of training RNNs, in which the backpropagation dynamics caused the gradients in an RNN to either vanish or explode. Unlike the traditional RNN, which simply computes a weighted sum of the input and applies a nonlinear function, each j^{th} LSTM unit maintains a memory c_t^j at time t. The output h_t^j, which is the activation of the LSTM unit is then:

$$h_t^j = o_t^j tanh(c_t^j) \tag{2.13}$$

where o_t^j is an output gate that modulates the amount of memory content exposure. The output gate is computed by:

$$o_t^j = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t)^j \tag{2.14}$$

where σ is a logistic sigmoid function, W is the current weight matrix, U is the update matrix and V_o is a diagonal matrix. This procedure of taking a linear sum between the existing state and the newly computed state is similar to the GRU unit, though LSTMs can control the degree to which its state is exposed. The memory cell c_t^j is updated by partially forgetting the existing memory and adding a new memory content $\tilde{c_t^j}$:

$$c_t^j = f_t^j c_{t-1}^j + i_t^j \tilde{c}_{t-1}^j \tag{2.15}$$

The introduction of gates allow the LSTM unit to decide whether to keep the existing memory or to overwrite it. This improves on the traditional recurrent unit, which overwrites its content at each time-step. Intuitively, if the LSTM unit detects an important feature from an input sequence at an early stage, it easily carries this information (the existence of the feature) over a long distance, hence, capturing potential long-distance dependencies. This process is illustrated graphically in Figure 2.4b.

Extensions to the basic LSTM model have been proposed such as adding an encoder/decoder network before the LSTM layer [152]. The LSTM encoder learns a fixed length vector representation of the input time series and the LSTM decoder uses this representation to reconstruct the time-series using the current hidden state and the value predicted at the previous time-step. Other approaches have added an attention layer to the LSTM [153]. This attention layer is used to select relevant hidden states across all time steps. Stacked LSTM approaches using multiple LSTM layers have also been proposed to allow the hidden state of each network to operate at a different time-scale [154].

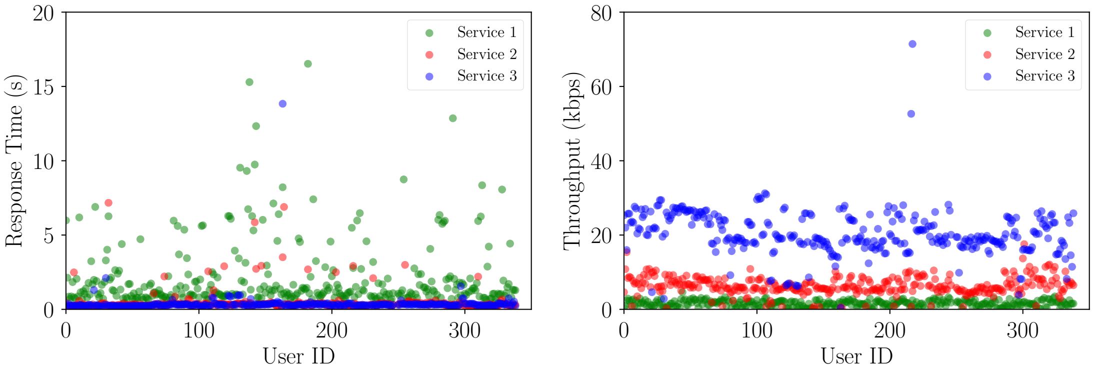
LSTMs are able to create complex models that can capture the dynamics of QoS in IoT and produce accurate forecasts (C.4), however they can also be resource intensive to train, which can be a challenge on devices at the edge (C.5). This can also lead to delays in incorporating recent changes in dynamic environments (C.1), which can decrease forecasting accuracy (C.4).

2.3.6 Summary

We summarize the advantages and limitations of the reliability approaches in Table 2.3. With current state of the art approaches there is a trade-off between the model complexity and the time to train the model. There were a number of challenges discussed in Chapter 1: C.1 Dynamic Environment, C.2 Reduced Tolerable Delay, C.3 Increased Traffic Delay, C.4 Critical Applications and C.5 Limited Resources. There are very lightweight models such as persistence and average that have no training time allowing them to handle C.1, C.2, C.3 and C.5 but the complexity of both models are very simple and it is not possible to capture the dynamics in the environment, which leads to a reduction in forecasting accuracy for C.4. Other approaches such as GRU and LSTM can generate much more complex models, but take longer to train and are deployed in the cloud. To manage the challenge of reduced tolerable delay for modern applications, C.2, there is a need to update our models on an edge device. There is also a need to react to the dynamic changes in QoS values by updating a model complex enough to capture the changes in the dynamic environment. This would ensure that the model is able to make accurate QoS predictions to avoid a failure to forecast a service that is about to degrade or fail, C.4, as when this happens the application will degrade in quality or fail.

Approach	Advantage	Limitation
Benchmark Methods	No training time Simple to implement and understand	Poor forecasting accuracy
ARIMA	Small training time, used in a large number of time series datasets	May not capture the sudden dynamics of QoS in IoT
Holt-Winters	Small training time Easily explainable	Simple model that may not capture the sudden dynamics of QoS in IoT
GRU	Can create complex models with large number of parameters	Large training time leads to delays in incorporating recent changes at the edge
LSTM	Can create complex models with large number of parameters	Large training time leads to delays in incorporating recent changes at the edge

TABLE 2.3: Executing Services Summary

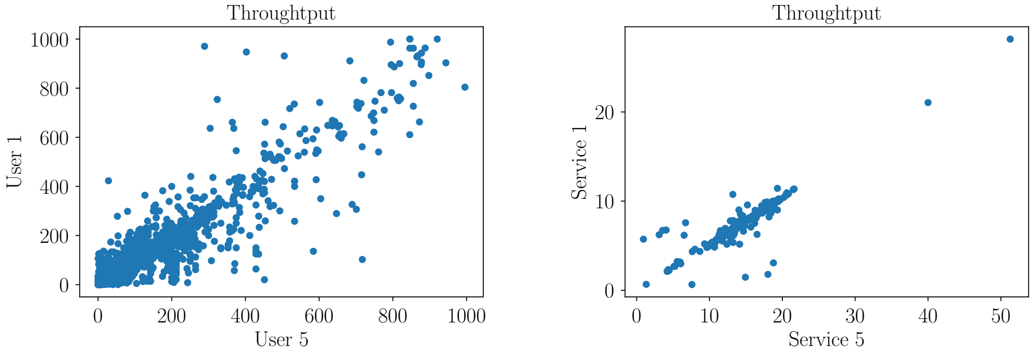


(A) Response Time for Users (B) Throughput for Users

FIGURE 2.5: Sample QoS Data

2.4 Candidate Services

When an anomaly or degradation in QoS for a particular service is detected, the service composition and execution engine should change to a functionally equivalent candidate service, which can provide an acceptable level of QoS. To switch to an appropriate service the system must have some knowledge of the QoS value of the service. However, if the user has not invoked the service before, then they will have no QoS values for that service, so will not know if it is a suitable replacement. The service provider can list QoS values in the service description, but these values are not suitable as the user-side QoS depends on the time and location of invocation, which cannot be known at the provider side. This means that users invoking the same services can have different values for QoS factors such response time and throughput. Figure 2.5 shows the distribution in response time and throughput for three services across three hundred and thirty nine users and is taken from the WS-DREAM dataset used in our experiments [42]. The users are in different locations and the QoS factors are measured at the user-side to show the impact of invoking a service at a different time and location. The QoS values are often not maintained in service registries as it would mean that



(A) Correlation Between Users

(B) Correlation between services

FIGURE 2.6: Correlation Between Users and Services

the service descriptions would need to be constantly updated. A collaborative filtering approach can be used, where QoS values reported from similar users and services are used to make predictions of the missing QoS value for the service. Figure 2.6 shows the correlation between two users and services on invoked throughput values from the WS-DREAM dataset [42]. This correlation can be exploited to identify similar users and services in the environment and predict missing QoS values for users, who have not invoked the service.

There are two main collaborative filtering methods, which can typically be classified as either memory or model-based. Memory-based approaches store the training data in memory and in the prediction phase similar users are sorted based on the current user. Model-based approaches create a model of the user invocation matrix to avoid having to store all the training data in memory. Section 2.4.1 presents an analysis of memory-based approaches and Section 2.4.2 presents an analysis of current model-based approaches.

2.4.1 Memory-based Approaches

Approaches such as collaborative filtering have been used in other domains such as commercial recommender systems [46], [45], [155]. Memory-based approaches store the training data in memory and in the prediction phase similar users are sorted based on the current user. There are a number of approaches that use neighbourhood-based collaborative filtering including user-based approaches [43], item-based approaches [44] and their combination [45]. Sorting based on the current user or service can be resource intensive (C.5) when there are a large number of users and service in the system. Vector similarity (VS) [43] and Pearson correlation coefficient (PCC) [46] are often used as similarity computation methods.

2.4.1.1 User-based algorithms

User-based algorithms find similar users to the current user requesting the service composition to predict the QoS of candidate services. Initial user-based approaches used PCC to calculate the similarity between users [156]. The most similar users based on QoS values were then weighted using a similarity metric to make predictions for the missing QoS values. User-based algorithms can have problems when there are many services and few ratings. User profiles can also change quickly if the user is mobile and in a dynamic environment (C.1) causing the entire system model to be recomputed. This is resource intensive (C.5) and can lead to less accurate predictions (C.4) if the latest dynamic changes are not updated in the model before predictions are needed.

2.4.1.2 Item-based algorithms

Item-based algorithms can resolve some of the user-based algorithm problems in systems that have more users than items. With more users than items, each item tends to have more ratings than each user, so an item's average rating usually does not change quickly. In item-based algorithms, the similarities between different items are calculated and then the user rating to an item will be predicted by rating the values of similar items. In commercial systems, item-based collaborative filtering has also been used [44]. A method for predicting QoS values using the similarity between services and their combination with the geographical location of the service provider has also been developed [157]. Item-based algorithms resolve some of the problems of user-based approach when there are more users than items in the system, however it is still a memory-based approach that requires all service invocations to be stored in memory. This is resource intensive, especially when there is a large number of services as would be expected in an IoT environment (C.5). This environment is also dynamic (C.1), which can lead to sudden changes in service characteristics by devices moving to a different location that can result in decreased prediction accuracy (C.4).

2.4.1.3 Hybrid-based algorithms

In this method, both the similarities between the service and the similarities between users are computed. Then, a hybrid algorithm uses both these similarities to predict QoS values. These algorithms have achieved better results than the individual user or item-based approaches [45]. Other approaches have extended these initial approaches to include user reputation. Trust-aware qos prediction (TAP) presents a personalised

QoS prediction approach that first clusters the users and calculates their reputation, based on the clustering information of a beta reputation system [158]. Then a set of trustworthy similar users is identified according to the calculated user reputation and similarity. Finally, the set of similar services is identified by clustering the services and making predictions for active users by combining the QoS data of the trustworthy similar users and services.

Overall, memory-based algorithms have a high perceptual capability i.e., the user can understand how the QoS value was predicted and their implementation is relatively easy. However these approaches can have some problems. One of the problems is data sparsity. Due to the large number of services, users will have invoked only a very small proportion of the overall services. This causes users to have few shared services and thus reduces the accuracy of a neighbour's similarity, reducing the overall prediction accuracy (C.4). Another problem with memory-based approaches is the cold-start problem, when a new user or service is introduced to the system. There are no neighbours as the user has not invoked any services yet or the service has not been invoked by any user. Therefore, it is not possible to predict QoS values (C.4). Finally, scalability is one of the major limiting factors for memory-based approaches. If the number of services and users is high, as would be expected for urban applications with millions of services and users then the cost of calculating the similarity between individual users and services will be very high (C.5). This can cause problems especially in a dynamic environment (C.1) such as IoT, where users and services can be mobile leading to sudden changes in QoS.

2.4.2 Model-based Approaches

Model-based approaches were developed to avoid some of the limitations of memory-based methods such as scalability and data sparsity. In these approaches, the user-service invocation dataset is used to train a model to predict unknown QoS values. Latent factor models create a low-dimensional factor model, on the premise that there are only a small number of factors influencing the QoS [46]. Matrix factorisation (MF) is a model-based approach used for QoS prediction that has been shown to produce accurate results (C.4) in QoS prediction and recommender systems [47], [48]. The first step in the algorithm is to factorise the sparse user-service matrix and then use $V^T H$ to approximate the original matrix, where the low-dimensional matrix V denotes the user latent feature space and the low-dimensional matrix H represents the service latent feature space, using the latent factor model [159]. The rows in the two matrices represent different features with each column in V representing a user and each column in H denoting a service. The model assumes that the original matrix can be represented by a small number of latent features (around 20-30) such as the network load, location of invocation and provider resources. In a dynamic environment such as IoT (C.1), the model may need to be retrained frequently to update changes in users and services QoS, which can be resource intensive (C.5).

2.4.2.1 Latent Features-based Approaches

The main advantage of latent features-based approaches is that they can somewhat solve the problem of data sparsity and scalability [159]. Various methods have been proposed in this research area, in which the MF method is used to predict QoS values. Nonnegative matrix factorization (NMF) differs from other matrix factorization methods in that it enforces the constraint that the latent factors must be nonnegative [160]. These constraints lead to a parts-based representation because they allow only additive, not subtractive, combinations. This non-negativity makes the resulting matrices easier to inspect. Extended matrix factorisation (EMF) is a MF-based approach that takes into account two additional regularisation terms in the MF equation to involve closer neighbours who are more similar in the prediction. One of the regularisation terms is the neighbours' information of the user's side and the other regularisation term is a modified Pearson Correlation Coefficient algorithm to find the similarity between services, which can improve accuracy [161]. Other approaches such as Probabilistic matrix factorization (PMF) have used a probabilistic-based matrix factorisation approach, which scales linearly with the number of observations and can handle sparse and imbalanced data. Neighbourhood Integrated Matrix Factorization (NIMF) is a method to provide improvement in predicting QoS values by combining MF for global information and neighbourhoods for local information [162]. In NIMF, the similarity between users is calculated using PCC and the top-k most similar users are identified, then MF is used for prediction.

Other approaches have developed methods based on Singular Value Decomposition (SVD) to overcome the cold-start problem by adding service provider information and the country where the service was provided [163]. In hierarchical matrix factorisation (HMF), the geographic information of users and services is combined with the MF method and provides a hierarchical approach to predict QoS values [164]. The local neighbourhood matrix factorization (LoNMF) method combines domain knowledge with the MF method and has thus been able to improve the MF method with a two-step neighbourhood selection mechanism [165]. LoNMF uses the geographical information of users and services to create a local user-service matrix. Integrating MF with the network map (NAMF), provides a network-aware method [166]. NAMF first processes the network map to measure the distance of the users on the network. Then the neighbours calculate the similarity between each user in terms of their distance in the network and add the similarity weight between them as a regularisation term to the base MF model.

Reputation-based matrix factorisation (RBF) is a personalised prediction method for QoS values that integrates matrix factorisation with user reputation-aware methods [52]. There has also been approaches based on the combination of model and memory-based methods [167]. There are context-aware methods based on MF, where content information from users, services or their interactions is combined with the MF method [168]. Initial experiments using deep learning model-based approaches such as Restricted Boltzmann Machines have also been conducted [53].

Model-based approaches have advantages in that they can handle the data sparsity and cold start problems better than memory-based methods. However, the most important advantage is in scalability, as these approaches can be used for the large number of users and services that are expected in an urban city environment. However, in spite of these advantages, these methods have some problems. If a new user or service comes into the environment and is added to the set, the model has to be created and trained again [54]. The cost of training these models is high as it is a resource-intensive process to generate the latent features they use (C.5). This makes them unsuitable for dynamic environments where users and services are mobile and move out of range of gateways or switch to a power saving mode (C.1). These methods also have multiple parameters that must be properly set as their values have a significant effect on the prediction accuracy (C.4).

2.4.3 Summary

We summarize the advantages and limitations of the reliability approaches in Table 2.4. This section has explored the current state of the art methods to predict QoS for candidate services. Memory-based algorithms have a high perceptual capability, but they have a number of problems with data sparsity, cold-starts and scalability. Scalability is one of the major limiting factors of memory-based algorithms as there is expected to be millions of services and users in an urban environment, which would make the cost of calculating the neighbours of services very high (C.5). This means the data and analysis needed for this approach would have to be conducted in the cloud, which would increase the traffic rate (C.3.) Model-based approaches have been designed to solve some of the challenges of memory-based approaches and have shown

Approach	Advantage	Limitation
Memory-based	High perceptual capability	Problems with data sparsity, cold-starts and scalability
Model-based	Can better handle data sparsity and cold start problem	Large training time required to generate latent features

Table 2.4: Candidate Services Summary

improvements in being able to handle the data sparsity and cold-start problems. These methods can also handle a large number of users and services as the latent features means there is only a small number of values to find the similarity between. However, these models have a large training time to generate these latent features and must be retrained if a new user or service is introduced to the system. This makes them unsuitable for a dynamic IoT environment where users and services are mobile and move out of range or change to a power saving battery mode (C.1). Therefore, there is a need for a lightweight model-based approach that can reduce the training time to allow a model to be updated quickly to include dynamic changes, while also maintaining the prediction accuracy.

2.5 Chapter Summary

This chapter has looked at current state of the art approaches to provide QoS through the layers of the IoT architecture. The analysis was then focused at the middleware level where a research gap was identified to provide user-centric reliability. Current user-centric approaches that focused on increasing the reliability of IoT applications by making predictions for currently executing and candidate services were identified. Figure 2.7 shows the kiviat diagram for forecasting QoS in currently executing services and Figure 2.8 shows the kiviat diagram for predicting QoS for candidate services.

The kiviat digram in Figure 2.7 and Figure 2.8 map to the challenges in Chapter 1. To handle a dynamic environment (C.1) and limited resources (C.5), the approaches should have a small training time to incorporate recent changes in the network and not overload the devices. To manage the reduced tolerable delay (C.2) and increased traffic rate (C.3) the model should also be able to be deployed at the edge or on the IoT devices. The physical proximity of an edge device one hop away from data generation reduces the latency to a level acceptable for modern application such as augmented reality. The edge device also reduces the amount of traffic that needs to be reported to the cloud over the core network. The model should also have a low testing time

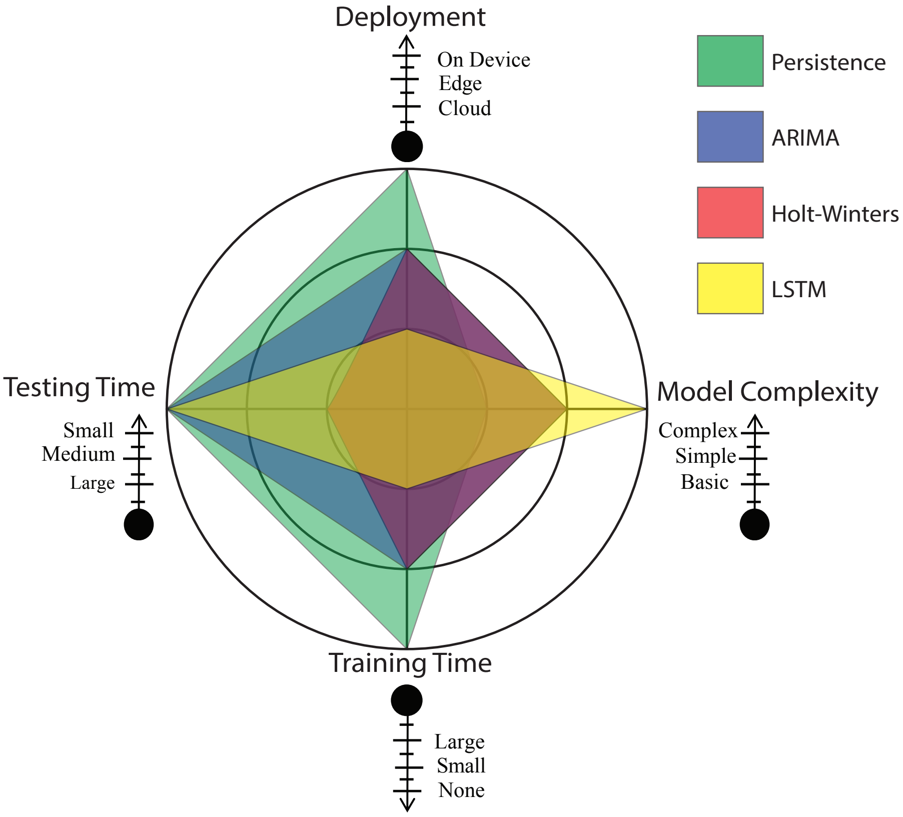


FIGURE 2.7: Kiviat Diagram of QoS Prediction for Currently Executing Approaches

to allow for reduced tolerable delay (C.2) and not offload any of the training process until the prediction is needed. To manage the accuracy requirements of medium and high critical applications (C.4) the model should be complex enough to capture the dynamics of the environment and make predictions that a very close to the actual values with low standard error metrics (MRE, RMSE).

Figure 2.7 shows that approaches for predicting QoS of currently executing services have covered different portions of the requirements. Benchmark models such as persistence and average can be deployed on an IoT device as they have no training time, but the model is very limited and does not capture the dynamics in the environment. Approaches such as ARIMA and Holt-Winters add more complexity to the model, which adds to the training and testing time, which can cause some problems for C.1. However, these models can still be deployed at the edge making them suitable for C.2, C.3, and C.5. These approaches allow for additional model complexity that can increase the prediction accuracy compared to the baseline models, which is needed for C.4. The LSTM approach has large training time that allows for additional model complexity, but the long training time can decrease prediction accuracy in dynamic environments, which causes some problems for C.4.

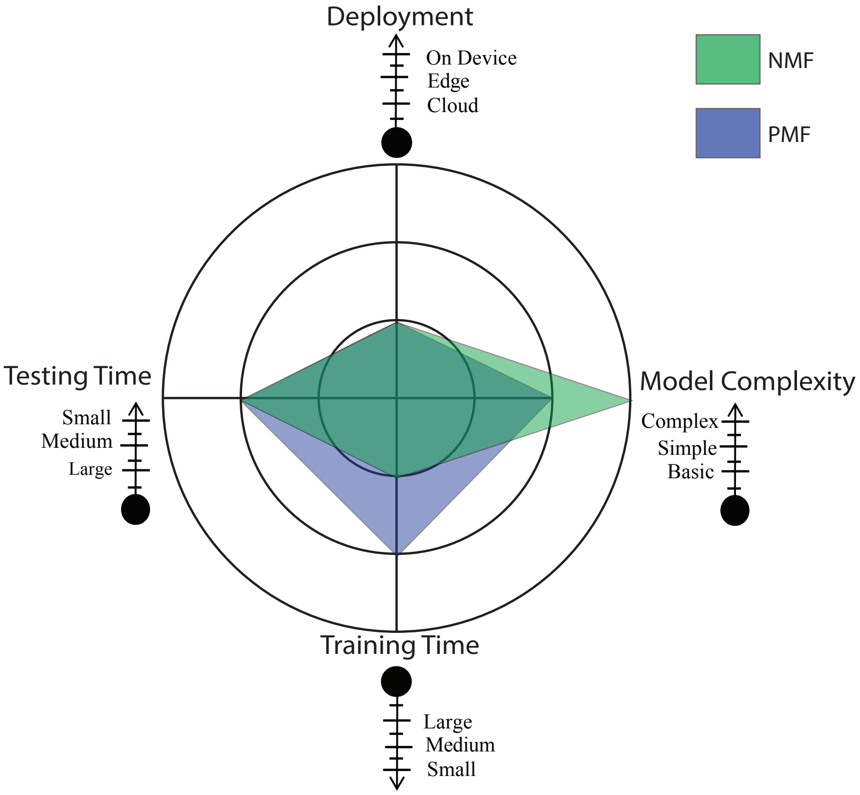


FIGURE 2.8: Kiviat Diagram of QoS Prediction for Candidate Service Approaches

Figure 2.8 shows that current candidate QoS prediction approaches are quite similar and have been deployed in the cloud with large training times to find similar replacement services. Current approaches including PMF and NMF are designed to be deployed in the cloud, which leads to increased tolerable delay, C.2 and increased traffic rate to the cloud, C.3. Some approaches such as PMF have reduced training time slightly by using probabilistic methods, but these methods still need to generate the latent features, which is resource intensive, C.5. The hypothesis is that a lightweight algorithm can be deployed at the edge of the network to reduce the time it takes to receive the predictions, while maintaining the prediction accuracy for dynamic service providers. The combination of improvements to predicting both currently executing and candidate services would improve the overall reliability of IoT applications.

Chapter 3

Design

The review of the state of the art, presented in the previous chapter, has identified a number of challenges in current QoS prediction approaches. Open issues include that they focus on being deployed in the cloud with unlimited resources, which makes them unsuitable to be used at the edge of the network. This leads to increased delay when making predictions, which is likely to be greater than the tolerable delay required by modern applications, such as augmented reality. This chapter introduces TTDR, the combination of algorithms proposed to reduce both TTD and TTR. Section 3.1 gives an overview of reliable service applications in IoT, how they can be modelled and the individual TTD and TTR parameters that were focused on in this thesis. Section 3.2 describes the design objectives and the features required in the approach to manage the challenges described in Section 1.2. Section 3.3 describes the system environment used to specify the scope of the contributions. Section 3.4 highlights the design decisions that were made to ensure the contributions have the required features. Section 3.5 describes the design of the noisy echo-state network to reduce TTD and Section 3.6 shows the design of the initial IoTPredict algorithm and the follow-on stacked autoencoder approach to reduce TTR.

3.1 Reliable Service Applications

Providing reliable service applications in IoT is difficult due to a number of challenges such as the devices being resource constrained and possibly mobile [169]. Service applications can be used in a range of domains that have different QoS requirements based on the sensitivity and criticality of the application [170]. Application QoS can typically be categorised as best effort (no QoS), differentiated services (soft QoS) and guaranteed services (hard QoS). In the hard QoS case, there are strict hard real-time QoS guarantees. This is appropriate for safety critical applications such as remote surgery or collision avoidance in a self-driving car system. Soft QoS does not require hard real-time guarantees but needs to be able to reconfigure and replace services that fail. An example is a routing application that uses air quality, flooding and pedestrian traffic predictions, to provide the best route through the city. If one of the services is about to fail, the application should be recomposed using suitable replacement services. The final case is best effort, where there are no guarantees when a service fails.

As this thesis focuses on attaining soft QoS guarantees, it is assumed that there will be some downtime and the goal is to try to reduce the amount of downtime. For example, if there is a 28 day budget and a user wants 99.9% availability then there can be 40 min downtime, 99.99% = 4 min, 99.999% = 24 s. The level of availability needed will depend on the criticality of the application, but the goal is the same: to reduce the amount of downtime. An error budget can be used to identify the individual factors that influence how much error the system will have [171, 172]. Error budgets have traditionally been used in mechanical engineering tasks, including all the factors that can contribute to the final error in a workpiece, i.e., the machine, process, auxiliary equipment and the interactions between them [173]. They have also been used in other design tasks such as LiDAR accuracy [174] and soil mapping [175]. In this thesis, reliability is quantified by a cost to time budget, which is the impact that a failure will have on the total time budget, taking into account the time to detect the error and the time to recover from the error as well as other parameters. The goal is to reduce the cost for individual users, as this increases the reliability of the overall system. This enables subsequent efforts to be directed at specific reliability factors such as the time to detect an error, rather than a general approach focused on improving reliability in IoT systems. The cost to time budget is based on descriptions from a production service system environment [176] and is calculated as follows:

$$\varepsilon = \frac{(TTD + TTR) \times impact\%}{TTF} \tag{3.1}$$

where ε is the cost to the time budget, TTD is the time to detection, TTR is the time to recovery, impact% is the percentage of users that are effected by the failure and TTF is the time to failure of the provider services. From a middleware perspective, TTF cannot be controlled as it is up to the providers who enable the services to improve it. Increasing TTF would reduce the cost to the time budget as the application would have to recover from service degradation less often, increasing the overall reliability. There have been suggestions for approaches to increase TTF through fault tolerance

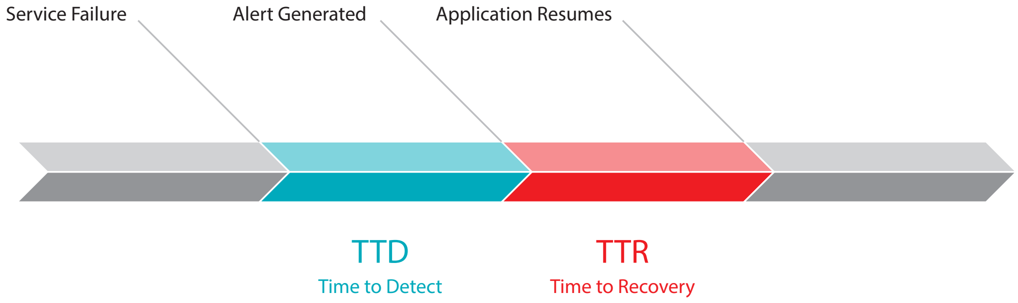


FIGURE 3.1: Impact of Time to Detection and Time to Recovery

approaches such as resource provisioning to provide extra resources to services when needed [57], increased maintenance scheduling [177] and the use of docker containers on devices to enable reliable distribution of services on devices [178]. The impact percentage is beyond the control of the prediction algorithms as they do not influence the number of users invoking the specific services, so a constant value is assumed. However, this parameter could be explored in future work with the composition and execution engine by distributing users across services that meet their QoS requirements rather than selecting the service with the best QoS value, as this could overload the best service leading to a drop in QoS. The service composition and execution engine could randomly choose from the top-k ranked services that fulfil the users QoS requirements to avoid this problem.

This thesis focuses on TTD and TTR. The impact of both of these parameters can be seen in Figure 3.1. TTD is the time from a service failure until an alert is generated so that the middleware knows that there is a failure. Once the alert has been generated, TTR is the time to find a replacement service and recompose the application to a working state. TTD focuses on detecting errors as soon as possible, to reduce the cost to the time budget. Short-term forecasting of QoS values for the next time step in the series allows for fast adaptation and the reduction of TTD. After analysing state of the art approaches in the previous chapter, challenges such as long training times for some models and lack of model complexity for others made it difficult for them to be deployed at the edge of the network and maintain a high prediction accuracy. In this thesis, a model based on echo state networks is proposed that is designed to be deployed at the edge of the network. Instead of training all the hidden nodes of the network, only the hidden to output nodes are trained, which greatly reduces the training time needed, allowing this model to be trained at the edge of the network. This has a number of benefits in response time and availability as well as reducing TTD, which increases the overall reliability of the application. The noisy-echo state network design is discussed in detail in Section 3.5.2

TTR focuses on recovering from service degradation by making accurate QoS predictions for suitable replacement services. To decrease the value of this parameter, an initial matrix factorisation-based approach is proposed, called IoTPredict, that reduces prediction error for replacement IoT services using an alternative similarity computation method designed for IoT services [179]. One of the limitations of this approach is that it uses matrix factorisation latent features, which are resource intensive to train. This means that the model has to be trained in the cloud, so user data must be reported to a centralised location. In this thesis, a stacked autoencoder model is also proposed that can be deployed on a deep edge architecture. The deep edge architecture uses embedded GPUs at the edge of the network to reduce the training time for more complex models. The stacked autoencoder approach is designed to be deployed on this architecture at the edge of the network. The stacked autoencoder model reduces the time to receive predictions, while maintaining prediction accuracy, which reduces TTR, increasing the overall reliability of the application. Both the IoTPredict and Autoencoder design are discussed in detail in Section 3.6.

3.2 Design Objectives and Required Features

This section outlines the design objectives and required features to manage the challenges described in Section 1.2. These design objectives and required features influence the decisions made in Section 3.4 when designing the TTD and TTR algorithms.

F.1 Fast Training Time. IoT device and gateways can be mobile especially in urban intelligence applications in smart cities (C.1). A QoS prediction approach for currently executing or candidate services should incorporate the most recent changes in the dynamic environment and any failures due to the wireless communication protocol. To incorporate recent QoS changes in the model before the next prediction is needed, the algorithm must have a fast training time. Based on the analysis of the IoT dataset collected we found 30s was a suitable monitoring frequency to capture the dynamics in an IoT environment, this is not a precise number, but provides an indication of the time frame that we consider to be fast [180]. This greatly reduced the standard deviation compared to the established web service dataset [181] making it more suitable modern IoT applications, such as augmented reality [182].

F.2 Low Tolerable Delay. Table 1.1 shows a number of urban intelligence applications and the increased demands on tolerable delay for recent applications (C.2). To be able to achieve the tolerable delay required by applications such as augmented reality that can use IoT services, it is necessary to deploy these QoS prediction algorithms close to the data generation and users to reduce latency. The algorithm that is used to generate the QoS predictions should also have a small test time to generate the predictions to ensure that this does not become a bottleneck.

F.3 Reduced Cloud Traffic Rate. Table 1.1 also shows the traffic rate for urban intelligence applications (C.3). The large amount of data generated by these applications can cause congestion on the network and increase energy usage. Sending large amounts of user data to the cloud is also a privacy risk as this can be valuable information for attackers. Therefore, a required feature is to manage these QoS values and predictions closer to the data generation, which reduces the amount of user information transported over the core network.

F.4 Prediction Accuracy. As the prediction algorithms are design to be used with applications that have medium criticality there is a need to maintain good prediction accuracy (C.4). This is important as if the forecasting algorithm fails to forecast that the service is about to degrade in quality or uses a replacement service that has low QoS, then this will cause the application to fail. Therefore, maintaining accurate prediction accuracy is an important feature to ensure that the reliability of the application is achieved.

F.5 Deep Edges. Computing at the edge introduces challenges on the computational power available for deploying and training models (C.5). However, more recent embedded GPUs haven been created that can be deployed at the edge of the network with low power requirements. An IoT architecture should use these more modern devices available at the edge to increase the complexity of models that can be used one hop away from data generation.

3.3 System Environment

This thesis focuses on supporting reliable service-based IoT applications. We define an application as being reliable by reducing the cost to time budget defined in Equation 3.1. This thesis focuses on decreasing the TTD and TTR parameters. The services that are provided to users in the environment can come from a range of heterogeneous devices, with different memory, CPU and battery life. Services can be provided from different service types including web services (WS), residing on resource rich devices in data centres that have access to large amounts of memory, CPU and battery life. Wireless sensor network (WSN) are typically deployed at the edge of the network using less powerful devices with limited battery life in a distributed architecture [183]. The final service type considered in our system environment is autonomous service providers (ASP), who are independent mobile users in the environment with intermittent availability [184]. They can provide services using their mobile phones but may suddenly fail due to a user's phone reaching low battery. Applications can be created by composing multiple services from different service types using a service composition engine [185]. For example, an application might perform some analysis and flood prediction using a water level service available from an autonomous service provider and a machine learning and storage service available as a web service.

Recent urban intelligence applications such as augmented reality have increased the demands on QoS factors such as tolerable delay. This means that applications composed for augmented reality should have high levels of QoS with low response time ($\leq$ 10ms). The application should also be able to adapt to changes in QoS, as service providers can be mobile so some of the services may degrade in quality. As the applications have low tolerable delay this process should happen quickly to avoid the user noticing a problem with the application. To achieve the required response time for adaptive services the middleware and prediction engine are deployed at the edge of the network on the gateways. The prediction algorithms should also have high accuracy to avoid choosing a replacement service with low QoS or failing to forecast that one of the services in the application is about to degrade, which could cause the application to fail.

Traditional IoT gateways such as Raspberry Pis and Intel Galileos do not have GPUs, making it difficult to train more complex models at the edge of the network. Embedded GPU devices such as the Jetson Tx2, have increased the training resources available for gateways at the edge of the network and make it possible to deploy and train more complex models at the edge. These devices have less resources than traditional desktop GPUs with a small form factor and reduced power consumption making them easy to deploy at the edge of the network. The design proposed in this thesis focuses on deploying smart prediction models on these embedded GPUs at the edge to provide an increased level of QoS.

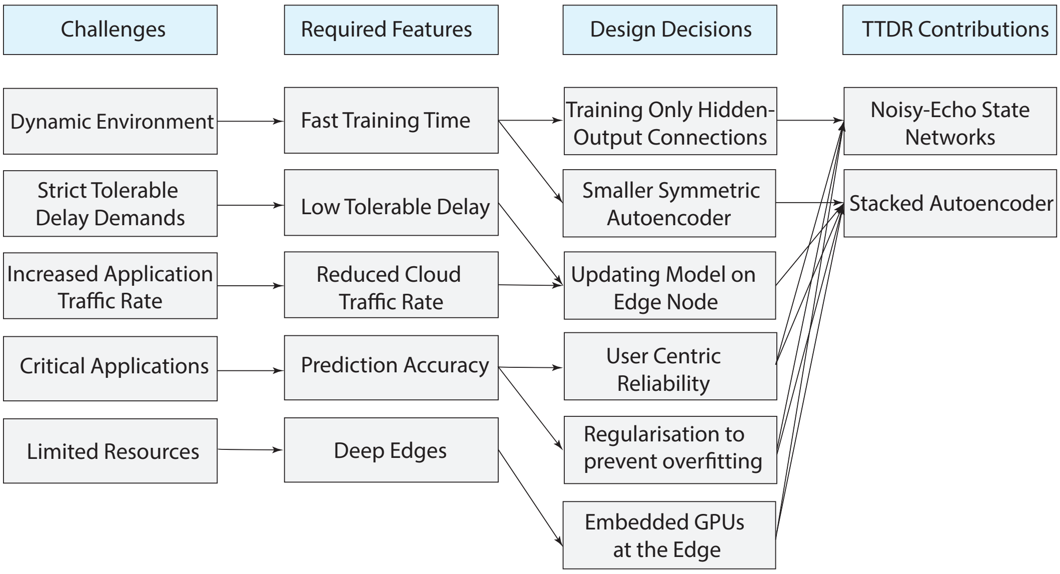

Challenges
Required Features
Design Decisions
TTDR Contributions
Dynamic Environment
Strict Tolerable Delay Demands
Increased Application Traffic Rate
Critical Applications
Limited Resources
Fast Training Time
Low Tolerable Delay
Reduced Cloud Traffic Rate
Prediction Accuracy
Deep Edges
Training Only Hidden-Output Connections
Smaller Symmetric Autoencoder
Updating Model on Edge Node
User Centric Reliability
Regularisation to prevent overfitting
Embedded GPUs at the Edge
Noisy-Echo State Networks
Stacked Autoencoder

FIGURE 3.2: Mapping of the Challenges (Section 1.2), Required Features (Section 3.2), Design Decisions (Section 3.4) and TTDR Contributions (Section 3.5 and Section 3.6).

3.4 Design Decisions

Reliability can be improved in a number of different ways as discussed in Section 2.2. This thesis focuses on reducing the TTD and TTR parameters in the cost to time budget defined in Equation 3.1. The following section outlines the decisions made to address the design objectives and required features in Section 3.2.

3.4.1 Fast Training Time

Fast training times are needed to incorporate the dynamic changes in the IoT environment, as some service providers and users can be mobile. A fast training time allows a model to update any new QoS data about the services before making a prediction for a new user. This can also increase the prediction accuracy as the model is updated with the latest changes in the network.

Design Decision 1: Training Only Hidden-Output Connections Training only the hidden-output connections in the noisy-echo state approach is used to achieve fast training times for the TTD approach. The hypothesis is that if the model is able to train faster it can capture the dynamic changes in the environment faster than other forecasting approaches such as LSTM and GRU. These models have large training time making it difficult for them to incorporate recent changes in the environment. The LSTM models can produce more accurate models in environments in less dynamic environments, when there is not as much of a change between the training and testing datasets.

Design Decision 2: Smaller Symmetric Autoencoder A four layer symmetric architecture with a reduced number of nodes in each layer is used to lower the training time for TTR. Gradients of complex functions such as autoencoders have a tendency to vanish or explode as an error is propagated through the layers. RMSprop optimisation was used to deal with this problem by using a moving average of the squared gradients to normalise the gradient itself. This has the effect of balancing the step size, which decreases the step for large gradients to avoid exploding and increases the step for a small gradient to avoid vanishing. This allows the model to achieve improved accuracy faster than a simple gradient descent approach.

3.4.2 Low Tolerable Delay & Reduced Cloud Traffic

Low tolerable delay is necessary to manage modern applications such as augmented reality. One of the simplest ways to reduce this compared to traditional cloud-based applications is to move the analysis and predictions closer to the edge of the network one hop away from data generation. This reduces the communication time and cloud traffic rate as user QoS does not have to be sent to the cloud but only to the local gateway.

Design Decision 3: Updating Models on Edge Nodes To manage the tolerable delay required by modern applications such as augmented reality, the models are designed to be updated on edge nodes. This adds additional constraints to the model that are not taken into account when training traditional deep learning models in the cloud. There is a limitation on the computational resources available at the edge and the time available to train the model to cope with dynamic changes in the network. The models are designed to try and maximise the prediction accuracy taking into account these factors.

3.4.3 Prediction Accuracy

The prediction accuracy of the models is very important as failing to predict that a service is about to fail or replacing a service with a low QoS value can cause the application to fail. Also, falsely predicting a service is about to fail when it actually will not causes a lot of additional processing in the middleware. Therefore, it is important to maintain the prediction accuracy of the model, while deploying and updating it at the edge.

Design Decision 4: User-Centric Reliability To support the prediction accuracy in a dynamic IoT environment a user-centric reliability approach is taken. Other approaches such as state-based reliability were also experimented with, but there was difficulty in modelling the dynamic environment. User-centric reliability focuses on the users in the environment sharing their QoS values to receive better QoS predictions for the candidate services that they can invoke. As the model training takes place at the edge, there is increased privacy for users compared to having to share their data in the cloud.

Design Decision 5: Regularisation to Prevent Overfitting One of the problems with training on local data at the edge is that it can lead to overfitting. For the TTD approach, a small amount of noise is added to increase the robustness of the model. This also helps the stability of the echo state model, to avoid getting stuck in a local minimum or maximum, which improves the reliability of training. For the TTR approach, dropout is added to the model so that some of the neurons are randomly ignored or dropped out. This has the effect of considering each layer to have a different number of nodes and connectivity to the prior layer. This means that each update to a layer during training is performed with a different configuration of the layer, which increases the robustness for testing.

3.4.4 Deep Edges

This thesis focuses on deploying and updating models at the edge of the network to manage the reduced tolerable delay demands. To manage the accuracy of the models as well as deploying them at the edge of the network, this thesis coins the term 'deep edges' [55]. This is a combination of the benefits of deep neural networks and edge networks. To achieve these deep edges, new devices capable of training these models at the edge of the network were needed [182].

Design Decision 6: Embedded GPUs at the Edge Traditional IoT gateways such as Raspberry Pis and Intel Galileos have low power CPUs such as Quad Core

1.2GHz Broadcom BCM2837 64bit. These devices do not have access to GPUs, which are used to speed up the training of more complex models, such as deep neural networks. Training using a CPU imposes strict limitations on the complexity and accuracy that can be generated. This thesis makes use of new embedded GPUs that can be deployed at the edge of the network, such as the Nvidia Jetson Tx2. This device has the same form factor as traditional IoT gateways, but can be used to speed up the training of deep neural networks at the edge. This allows for a greater range of models that can be deployed at the edge of the network.

3.5 TTD

This section describes the design for reducing TTD to improve the overall reliability of service applications. An explicit problem definition is given in Section 3.5.1 to describe the exact problem that is being solved. The design of the noisy-echo state network approach to reduce TTD is described in Section 3.5.2.

3.5.1 Problem Definition

A time series is a series of data points such as QoS values indexed in time order. Time series forecasting uses a model trained on previous values to predict future values. In this thesis, a QoS time series is defined as:

$$Y_{0 \dots t} = y_0, y_1, \dots, y_{t-1}, y_t \tag{3.2}$$

where $Y_{0 \dots t}$ denotes a set of values of a QoS time series during the period $0 \dots t$ and y_i represents the value of the time series at time i. For example, the response time series during the period $0 \dots t$, $RT_{0 \dots t}$ is shown:

$$RT_{0 \dots t} = rt_0, rt_1, \dots, rt_{t-1}, rt_t \tag{3.3}$$

where rt_i represents the response time of the service at time i. In this problem, $curr$ denotes the present and $0 < curr < t$ splits the time series $Y_{0 \dots t}$ into two smaller time series: the training data time series $Y_{0 \dots curr}$ and the testing data time series $Y_{curr+1 \dots t}$ as follows:

$$\underbrace{rt_0, rt_1, \ldots, rt_{curr-1}, rt_{curr},}_{\text{Training data time series}} \quad \underbrace{rt_{curr+1}, rt_{curr+2}, \ldots, rt_{t-1}, rt_t}_{\text{Testing data time series}} \tag{3.4}$$

The assumption is that, at time *curr*, the observations of the training data time series are known and available, while the observations of the testing data time series are unknown and need to be predicted. In this defined problem, the training data time series is used in predictor generation as defined below and the testing data time series is used in performance measurement accuracy.

Each evaluated time-series forecasting approach goes through two phases: (1) predictor generation and (2) forecast production. In the predictor generation phase, which uses the training data time series, each time series approach must fit a predictor. It must learn from the training data time series. Then, in the forecast production phase, the predictor incrementally produces QoS forecasts.

Predictor generation can be simply expressed as:

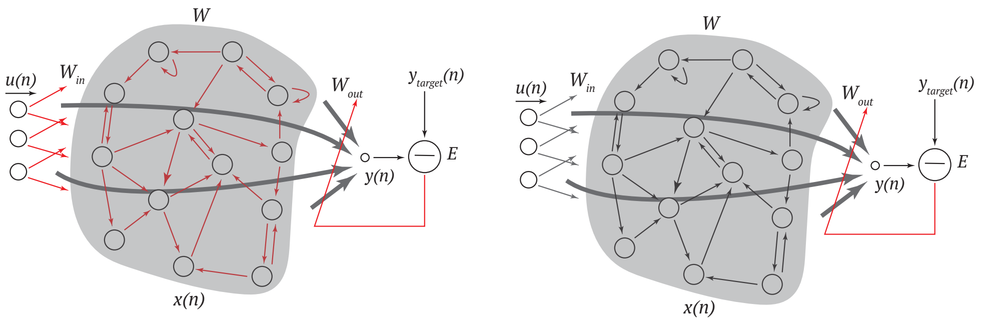
$$Predictor \leftarrow TA(Y_{0 \ldots curr}) \tag{3.5}$$

where Predictor denotes a time-series predictor generated by a time-series approach TA and the training data time series $Y_{0 \ldots curr}$ is the approaches input. TA represents a time-series approach that can generate a time-series predictor.

Forecast production is performed after a predictor is obtained. The forecasts will be incrementally produced by the predictor as the time series extends as follows:

$$\begin{gathered} f_{curr+1} \leftarrow Predictor(Y_{0 \ldots curr}) \\ f_{curr+2} \leftarrow Predictor(Y_{0 \ldots curr+1}) \\ \vdots \\ f_t \leftarrow Predictor(Y_{0 \ldots t+1}) \end{gathered} \tag{3.6}$$

where f_{curr+1} denotes a forecast of the time series at time $(curr + 1)$ and Predictor $(Y_{0 \ldots t-1})$ means a previously fitted time series predictor with the time series $Y_{0 \ldots t-1}$ as input. The forecasting is always one step ahead, at time *curr* the predictor can forecast the value one time step ahead $curr + 1$. As the value of *curr* increases with



(A) RNN Architecture

(B) ESN Architecture

FIGURE 3.3: (a) Traditional gradient descent-based RNN training adapts all connection weights (red) including input-to-RNN, RNN-internal and RNN-to-output weights. (b) In reservoir computing, only the RNN-to-output weights are adapted.

time moving forward the entire forecast can be gradually obtained. The model can then be updated by using the latest time series value as part of the training data.

3.5.2 Echo State Networks

Echo state networks (ESNs) belong to a special type of recurrent neural network that is characterised by having a very fast learning procedure, compared to traditional RNN-based approaches. This is one of the required features for our TTD approach. The basic idea of ESNs is similar to Liquid State Machines [186]. They both belong to the class of dynamic systems implemented according to the reservoir computing framework [187]. The main idea of reservoir computing is to (i) drive a randomly initialised, large, fixed recurrent neural network with the input signal, which induces in each neuron within this reservoir a nonlinear response signal, and (ii) use a linear combination of all of these response signals to generate the desired output. To maintain the high modelling capacity of ordinary recurrent neural networks, a large (e.g., 100 hidden neurons) RNN is used as a dynamic reservoir in the hidden layer of the ESN, which can be activated by suitably presented input and/or feedback of output. The complex structure of the dynamic reservoir allows for ESNs to have a high capability for modelling complex dynamic systems. The difference between a traditional RNN architecture and the ESN architecture is shown in Figure 3.3. The reason for the large difference in training times can be seen in Figure 3.3a as the RNN approach has to train the input-to-RNN, RNN-internal and RNN-to-output weights, while in Figure 3.3b the ESN approach trains only the RNN-to-output weights. The large difference in training time allows the model to incorporate recent QoS values in a dynamic edge environment much faster than traditional RNN approaches.

ESNs have been used in a variety of different applications such as classifying time-independent tasks [188], intrusion detection [189], adaptive control [190] and harmonic distortion measurements [191]. ESNs have also been used in a range of recognition tasks including speech and handwriting recognition. Initial approaches focused specifically on the recognition of Japanese vowels [192] and digits [193, 194]. Further attempts focused on continuous speech recognition with a large committee of predictive classifiers using ESNs [195]. Handwriting recognition is similar with many techniques being applied to both [196]. ESNs have also found bio-medical applications such as real-time detection of epileptic seizures with low latency and high accuracy [197]. This enables treatments for epilepsy that are based on closing-the-loop: rapidly detecting the seizure and actively counteracting it using medication or brain stimulation.

3.5.2.1 Basic Model

ESNs can be applied to supervised machine learning tasks: for a given training input signal $u(n) \in \mathbb{R}^{N_u}$, a desired output signal $y^{target}(n) \in \mathbb{R}^{N_y}$ is known. In this case $n = 1, \ldots, T$ is the discrete time and T is the number of data points in the training dataset, which can consist of multiple sequences of varying length. The general task is to learn a model with output $y(n) \in \mathbb{R}^{N_y}$, where $y(n)$ matches $y^{target}(n)$ reducing the error measure $E(y, y^{target})$ and generalising well to unseen data. The root mean squared error (RMSE) is used as the error measure:

$$E(y, y^{target}) = \sqrt{\frac{1}{T}\sum_{n=1}^{T}(y_i(n) - y_i^{target}(n))^2} \tag{3.7}$$

The time series forecasting problem can be re-framed as a supervised learning problem, to make it suitable to be used by an ESN. This is done by using previous time steps as input variables and the next time step as the output variable. The following simple example shows exactly how the original time series data can be restructured to be a supervised learning problem, by using a sliding window.

```
time, measure                     X, y
   1, 100                         ?, 100
   2, 110                       100, 110
   3, 108            ->         110, 108
   4, 115                       108, 115
 115, ?
```

The ESN can then be used to make predictions for this supervised problem. ESNs use an RNN with leaky-integrated discrete-time continuous value units. The update equations are [198]:

$$\tilde{x}(n) = tanh(W^{in}[1; u(n)] + Wx(n-1)) \tag{3.8}$$

$$x(n) = (1-\alpha)x(n-1) + \alpha\tilde{x}(n) \tag{3.9}$$

where $x(n) \in \mathrm{I\!R}^{N_x}$ is a vector of reservoir neuron activations and $\tilde{x}(n) \in \mathrm{I\!R}^{N_x}$ is its update, all at time step n, tanh($\cdot$) is applied element-wise, $[\cdot\,,\cdot\,]$ stands for a vertical vector (or matrix) concatenation, $W^{in} \in \mathrm{I\!R}^{N_x \times (1+N_u)}$ and $W \in \mathrm{I\!R}^{N_x \times N_x}$ are the input and recurrent weight matrices respectively, and $\alpha \in (0, 1]$ is the leaking rate. Multiple wrappers can be used besides tanh such as sigmoid and relu, which are evaluated in Section 3.5.2.3. The linear readout layer is defined as [187]:

$$y(n) = W^{out}[1; u(n); x(n)] \tag{3.10}$$

where $y(n) \in \mathrm{I\!R}^{N_y}$ is network output, $W^{out} \in \mathrm{I\!R}^{N_y \times (1+N_u+Nx)}$ the output weight matrix and $[\cdot\,,\cdot\,]$ again stands for a vertical vector (or matrix) concatenation. An additional non-linearity can be applied to $y(n)$ in Equation 3.10 as well as feedback connections W^{fb} from $y(n-1)$ to $\tilde{x}(n)$ in Equation 3.8. A visual representation of the ESN is given in the schematic in Figure 3.4.

3.5.2.2 Producing a Reservoir

The reservoir is the central component of the model and acts (i) as a nonlinear expansion and (ii) as a memory of input $u(n)$. The nonlinear expansion is similar to kernal methods in machine learning. The reservoir is (i) a nonlinear high-dimensional expansion $x(n)$ of the input signal $u(n)$. At the same time, (ii) the reservoir serves as a memory, providing temporal context. This is a crucial reason for using RNNs in the first place. Both aspects combined should provide a rich and relevant enough signal space in $x(n)$, such that the desired $y^{target}(n)$ can be obtained by a linear combination from it. However, there are some tradeoffs between (i) and (ii) when setting the parameters of the reservoir [199].

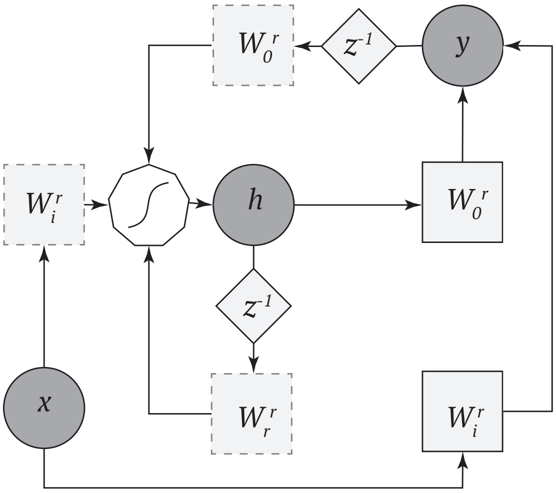


FIGURE 3.4: Illustration of ESN architecture. The polygon represents the non-linear transformation performed by neurons and z^{-1} is the unit delay operator. The circles represent input x, state h, and output y. Solid line squares W_r^0 and W_i^0 are the trainable matrices of the readout, while dashed line squares, W_r^r, W_o^r and W_i^r are randomly initialised matrices.

The reservoir is defined by the tuple (W^{in}, W, α). The input and recurrent connection matrices W^{in} and W are sparse with nonzero elements that follow a specific distribution such as symmetric uniform, discrete bi-valued or normal distribution centered around zero. The effect of more modern initialising techniques such as sparse and orthogonal reservoir matrices (SORM) [200] and Yilda [201] compared to a traditional naive approach where weights are assigned a uniform randomised variable between -0.5 and 0.5 as well as other hyperparameters are evaluated in Section 3.5.2.3.

The leaking rate α of the nodes in Equation 3.9 is a highly important parameter for time series forecasting as it controls the speed of the reservoir update dynamics discretised in time. The reservoir update dynamics can be described in continuous time as an ordinary differential equation (ODE)[187]:

$$\dot{x} = -x + tanh(W^{in}[1; u] + Wx) \tag{3.11}$$

Making an Euler's discretisation of this ODE in time, taking

$$\frac{\triangle x}{\triangle t} = \frac{x(n+1) - x(n)}{\triangle t} \approx \dot{x} \tag{3.12}$$

gives the discrete time equations 3.8 and 3.9 with α taking the place of the sampling interval $\triangle t$. Therefore, α can be regarded as the time interval in the continuous world between two consecutive time steps in the discrete realisation. The effect of setting α is comparable to that of re-sampling $u(n)$ and $y^{target}(n)$ when the signals are slow [202, 203]. Setting a small α, that induces show dynamics of $x(n)$, can dramatically increase the duration of short-term memory in ESN [204].

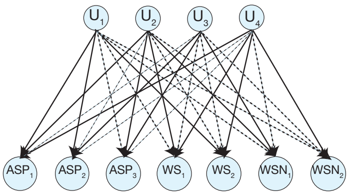
3.5.2.3 Noisy-Echo State Networks

One of the problems encountered with the basic ESN model is the lack of stability in the reservoir. This is where the echo-state property is lost and the system enters into an oscillatory behaviour [205]. This can lead to a number of problems during training as the network may fail to converge, leaving no way to make forecasts. It can also cause problems during the exploitation phase as perturbed data can influence the output forecasts, increasing the error. One of the contributions that we make to the basic echo state network approach in this thesis is to add some small amount of noise during sampling to recover from these perturbed signals [206]. Equation 3.8 can be updated to include this additional parameter:

$$\tilde{x}(n) = tanh(W^{in}[1; u(n)] + Wx(n-1) + v(n)) \tag{3.13}$$

where $v(n)$ is a small uniform white noise term (typically 0.0001 to 0.01). This allows the use of the same kind of input during training as will be encountered during exploitation, but makes the training a bit more varied than is expected in the exploitation phase, which increases robustness. The noise makes the network states wobble around different weights, which can lead to reduction in errors. Previous approaches for generating extremely high precision mathematical functions have found a stability precision tradeoff. Larger noise resulted in more stable models that converged to the desired attractor from a larger subspace of starting states, while having a less precise prediction [207]. QoS data is less deterministic, with the additional noise both making the model more stable. The basic rule that was followed was to use the same kind of input during training as would be encountered during exploitation, but to make the training input a bit more varied than expected in the exploitation phase.

To make accurate predictions for the IoT and web services there are a number of parameters that need to be set. Here, the experiments run to set those values are illustrated. The experimental setup for the dataset and metrics used to evaluate these hyperparameters is described in Section 5.2.1.7.



(A) Invocation Graph

	ASP_1	ASP_2	ASP_3	WS_1	WS_2	WSN_1	WSN_2
U_1	0.9	0.23			0.12		
U_2	0.43		1.22	0.23		0.33	
U_3		0.30	1.39			0.37	
U_4	0.23			0.34	0.34		0.24

(B) User-service Matrix

	ASP_1	ASP_2	ASP_3	WS_1	WS_2	WSN_1	WSN_2
U_1	0.9	0.23	0.91	0.25	0.12	0.22	0.30
U_2	0.43	0.23	1.22	0.23	0.25	0.33	0.36
U_3	0.80	0.30	1.39	0.38	0.36	0.37	0.15
U_4	0.23	0.23	1.20	0.34	0.34	0.22	0.24

(C) Predicted Matrix

FIGURE 3.5: Demonstration of QoS Prediction in IoT

3.6 TTR

TTR should be reduced to improve the overall reliability of service applications. Section 3.6.1 defines exact problem that is being solved. The design of the initial IoT-Predict approach is then described in Section 3.6.2 and the final stacked autoencoder approach designed to reduce TTR further is described in Section 3.6.3.

3.6.1 Problem Definition

To provide QoS values on m IoT services for n users, a middleware needs to invoke at least $n \times m$ services. This is very difficult in an IoT environment where there are a large number of services and users. Without this QoS information, a service composition and execution engine cannot select the optimal components based on their QoS and must make a choice based on whatever QoS information is available. This leads to choosing potentially non-optimal services, which can cause service degradation at runtime and execution errors.

The QoS value of IoT service s observed by user u can be predicted by exploring the QoS experiences from a user similar to u. A user is similar to u if they share similar characteristics, which can be extracted from their QoS experiences with different services. By sharing local QoS experience among users, the QoS value of a range of IoT services including ASPs, web services and WSNs can be predicted, even if the user u has never invoked the service s before. This can be modelled as a bipartite graph as illustrated in Figure 3.5a, $G = (U \cup S, E)$, such that each edge in E connects a vertex in U to S. Let $U = \{u_1, u_2, ..., u_4\}$ be the set of component users, $S = \{ASP_1, ASP_2, ..., WSN_2\}$ denote the set of IoT services and E (solid lines) represent the set of invocations between U and S. Given a pair $(i, j), u_i \in U$ and $c_j \in S$, edge e_{ij} corresponds to the QoS value of that invocation. Given the set E, the task is to predict the weight of potential invocations (broken lines).

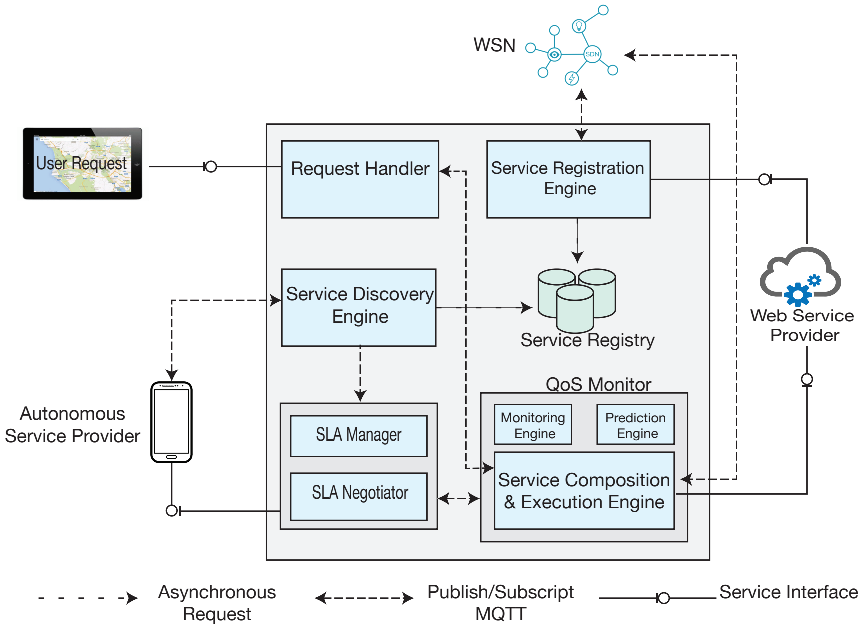
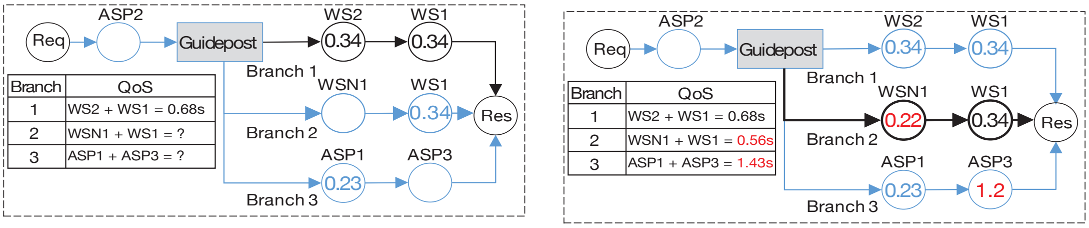


FIGURE 3.6: Middleware Architecture

Many different components are needed to create service based applications as shown in the architecture in Figure 3.6. The service composition and execution engine (SCEE) is responsible for the composition and execution of services discovered by the service discovery engine (SDE) as discussed in Section 1.1. The QoS monitor is responsible for monitoring the QoS values from the different services, which are stored in the service registry. Figure 3.7 illustrates a list of available services in the environment identified to satisfy a user request, which was received from the request handler. The SCEE creates a list of service flows based on the concrete service providers received from the SDE. The flows are then merged based on the service description. If two or more services in the flow have the same input, the SCEE creates a guidepost to enable the invocation of one of these services based on QoS requirements [208]. As some of the QoS values can be missing from the registries, the goal of the collaborative framework is to make predictions for the missing values.

An execution guidepost $G = \{R_{id},\ D\}$ is a split-choice control element of the composition process and maintains a set of execution directions D for a composition request R_{id} [208]. These execution directions will be referred to as branches. Each element in the set D is defined $d_j = \{id,\ w,\ q\}$ where j $\leq |D|$. The set w represents the services in the branch and id represents the identifier of the branch. The value q reflects the branch's aggregated QoS values [209], which can be calculated according to predefined formulas [210]. The branch that maximises/minimises an objective function will be selected by the guidepost during execution. This objective function can contain a



Branch	QoS
1	WS2 + WS1 = 0.68s
2	WSN1 + WS1 = ?
3	ASP1 + ASP3 = ?

Branch	QoS
1	WS2 + WS1 = 0.68s
2	WSN1 + WS1 = 0.56s
3	ASP1 + ASP3 = 1.43s

(A) Service Flow without QoS Prediction (B) Service Flow with QoS Prediction

FIGURE 3.7: Service Composition Flows

number of different non-functional QoS factors that can be specified by the user in the service request.

Consider the response time for each branch. The formula in Equation 3.14 calculates the response time by aggregating the response time value of each component service in a sequential flow [210]. In this formula, rt_i is the response time of service i. However, it is possible that this value could not be calculated because of a missing QoS value from an individual service candidate in the flow, or the value being out-of-date.

$$Response\ Time\ (RT) = \sum_{i=1}^{n} rt_i \tag{3.14}$$

To address this problem, the QoS monitor uses QoS predictions for missing QoS values across each branch stored in the guidepost. Figure 3.7 shows the flows created by the SCEE for User 4 (U_4) in Figure 3.5. The response time values recorded during the service discovery phase were 0.34s for service provider WS_2, 0.34s for WS_1 and 0.23s for ASP_1. The response time values for WSN_1 and ASP_3 were not recorded. In Figure 3.7a, when the execution reaches Guidepost G, only Branch 1 can aggregate the response time, which is not optimal. If the composition selects the branch with the lowest reported response time, it will select Branch 3, which is also not optimal. Only when the predicted values for the missing service QoS are used does the composition choose the optimal Branch 2, which can be seen in Figure 3.7b.

The result of matrix factorisation is visualised for the flows generated in Figure 3.7, in Figure 3.5b, where each table entry shows an observed weight in Figure 3.5a. The task can be framed as a matrix completion problem, where the goal is to fill in the remaining values to have a fully completed matrix as shown in Figure 3.5c. In the remainder of this section, the overall framework used to generate the predicted values at a higher level based on a specific example is presented.

One of the problems that emerges given the large number of services in the IoT is that there are a number of functionally similar services for users to choose, which reduces the possibility of having commonly invoked services. In the small-scale example in Figure 3.5b, with half the values reported there are few services common to users. For example, U_1 has only one service in common with U_2 and U_3 and two services in common with U_4. The values come from the public dataset released by Zheng et al. [42], which consists of a matrix of the response time and throughput of 339 users for 5,825 web services. As this dataset is for web services, the HetHetNets traffic model is used to add heterogeneous IoT traffic data to the existing dataset [211], which is described in more detail in Section 5.3.1.1. The original dataset only has the user service matrix as features, which makes it difficult to calculate the similarity between users with few commonly invoked services.

The first step in the IoTPredict algorithm, discussed in more detail in Section 3.6.2.1, is to factorise the sparse user-service matrix and then use V^TH to approximate the original matrix, where the low-dimensional matrix V denotes the user latent feature space and the low-dimensional matrix H represents the service latent feature space, using the latent factor model [159]. The additional features in the latent feature space represents the underlying structure in the data, computed from the original dataset using matrix factorisation. As these matrices are dense, they allow for similarity computation between all the users and services in the matrix, which solves the original problem of having sparse matrices. The stacked autoencoder algorithm uses a different approach based on artificial neural networks and RMSprop optimisation. The first part of the neural network called the encoder compresses the original data into a set of latent features. The second part of the network called the decoder uses this latent representation to reconstruct the original data, tuning the weights to reduce the error in the network.

Once there is access to the low dimensional dense matrices, the similarity between different users and services can be computed using their latent features (Section 3.6.2.2). Traditional approaches for QoS prediction have used Pearsons Correlation Coefficient (PCC), to calculate the similarity between users and services. However after conducting some experiments on the IoT data, it was found that there were a number of assumptions PCC makes that are not satisfied, such as having no outliers and the variables being approximately normally distributed. This work proposes an alternative non-parametric similarity computation mechanism that does not make these assumptions called Kendall's Tau [212].

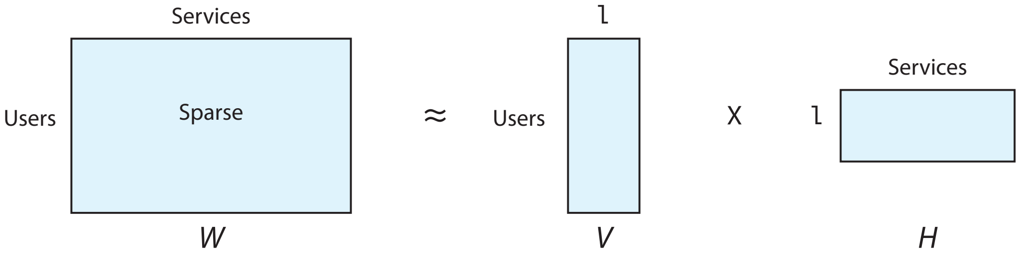


Figure 3.8: Illustration of Factorising Original Matrix Into Two Low-Rank Matrices

Once the similarity between users and services has been computed, predictions can then be made for the missing values by using the top-k largest Tau values for the users and services (Section 3.6.2.3). The predictions for the users and services are weighted based on how similar they are and using an equal weighting of user and service-based prediction.

3.6.2 IoTPredict

IoTPredict is the first iteration of this work to improve the accuracy of TTR for IoT services [179]. It uses a latent features-based approach to compare the similarity between users to make predictions for the missing values.

3.6.2.1 Latent Features Learning

Matrix factorisation is employed to learn the latent features of the users and services, by fitting a model to the user-service matrix from the original dataset. The matrix with the raw QoS values is factorised into two low-rank matrices V and H, as seen in Figure 3.8. The QoS usage experience of a user is typically determined by a small number of factors, such as the network load, location of invocation and provider resources. The latent feature model ensures an accurate and low-dimensional representation of the original matrix. Latent feature are hidden features that cannot be directly observed, an example would be text document analysis where words extracted from the document are features. When the raw data of words is factorised you can find topics with similar semantic meaning such as sail-boat, schooner and steamer factorising into a topic such as ship or boat.

Let Ω be the set of all pairs $\{i, j\}$ and Λ be the set of all known pairs (i, j) in Ω. Consider the matrix $W \in \mathbb{R}^{m \times n}$ consisting of m users and n services. Let $V \in \mathbb{R}^{l \times m}$ and $H \in \mathbb{R}^{l \times n}$ be the latent user and service feature matrices. Each column in V

represents the l-dimensional user-specified latent feature vector of a user and each column in H represents the l-dimensional service-specific latent feature of a service. An approximating matrix $\tilde{W} = V^T H$ is employed to learn the user-service relationship W [160]:

$$w_{ij} \approx \tilde{w}_{ij} = \sum_{k=1}^{l} v_{ki} h_{ki} \tag{3.15}$$

To learn matrices V and H from the obtained QoS values in the original matrix W, a cost function is constructed to evaluate the accuracy of the approximation. The standard Euclidean distance between the two matrices is used as the cost function.

$$F(W, \tilde{W}) = \parallel W - \tilde{W} \parallel_F^2 = \sum_{ij} (w_{ij} - \tilde{w}_{ij})^2 \tag{3.16}$$

where $\parallel \cdot \parallel_F^2$ denotes the Frobenius norm.

The optimisation problem can then be solved by using the optimisation objective function in [160]:

$$\begin{aligned} \min_{V,H} \quad & f(V,H) = \sum_{(i,j)\in\Lambda} [\tilde{w}_{ij} - w_{ij} log \tilde{w}_{ij}], \\ s.t. \quad & \tilde{w}_{ij} = \sum_{k=1}^{l} v_{ki} h_{ki}, \\ & V \geq 0, \\ & H \geq 0. \end{aligned} \tag{3.17}$$

The objective function in Equation 3.17 can then be minimised using incremental gradient descent to find a local minimum, where one gradient descent step intends to decrease the square of the prediction error of only one rating, that is $\tilde{w}_{ij} - w_{ij} log \tilde{w}_{ij}$. V and H are updated in the opposite direction of the gradient descent in each iteration [213].

3.6.2.2 Similarity Computation

The similarities of the latent features for different users and services in matrix V and H can be calculated using a correlation coefficient to measure the similarity. Pearson

Correlation Coefficient (PCC) is widely used for correlation computation in collaborative filtering [213], [46], [214]. The correlation between users u_i and u_j is defined by performing PCC computation on their l-dimensional latent feature vectors V_i and V_j with the following equation [46]:

$$S(u_i, u_j) = \frac{\sum_{k=1}^{l}(v_{ik} - \bar{v}_i)(v_{jk} - \bar{v}_j)}{\sqrt{\sum_{k=1}^{l}(v_{ik} - \bar{v}_i)^2}\sqrt{\sum_{k=1}^{l}(v_{jk} - \bar{v}_j)^2}} \tag{3.18}$$

where $v_i = (v_{i1}, v_{i2}, \ldots, v_{il})$ is the latent feature vector of user u_i and v_{ik} is the weight on the k^{th} feature. $\bar{v}_i$ is the average weight on the l-dimensional latent features for user u_i. The similarity between two users $S(i, j)$ falls into the interval [-1, 1], where a larger value indicates higher similarity.

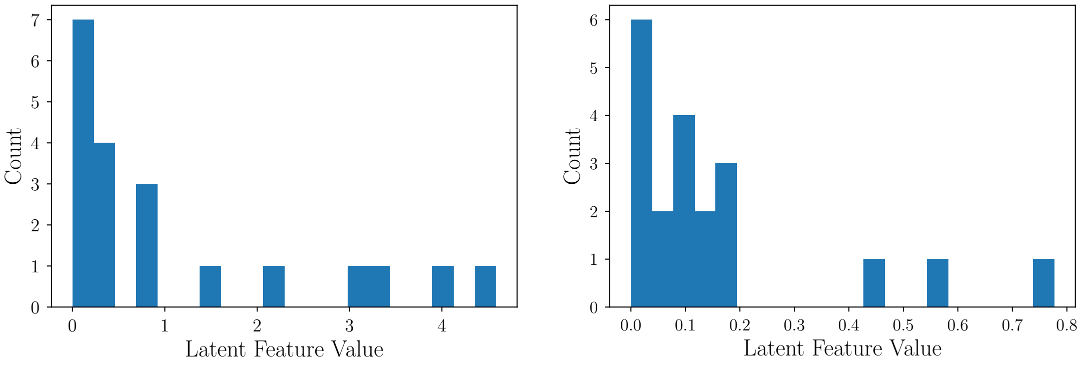
PCC is also employed to compute the similarity between service s_i and service s_j as follows:

$$S(s_i, s_j) = \frac{\sum_{k=1}^{l}(h_{ik} - \bar{h}_i)(h_{jk} - \bar{h}_j)}{\sqrt{\sum_{k=1}^{l}(h_{ik} - \bar{h}_i)^2}\sqrt{\sum_{k=1}^{l}(h_{jk} - \bar{h}_j)^2}} \tag{3.19}$$

where $h_i = (h_{i1}, h_{i2}, \ldots, h_{il})$ is the latent feature vector of service s_i and h_{ik} is the weight on the k^{th} feature. $\bar{h}_i$ is the average weight on l-dimensional latent features for service s_i.

PCC makes a number of assumptions that must be taken into consideration [215]. The variables being correlated must be approximately normally distributed. However, as can be seen in Figure 3.9, which shows the histogram of latent feature values for the users and services after conducting matrix factorisation, it is clear that the distribution is not normal. PCC also assumes that outliers are kept to a minimum or removed entirely. Figure 3.10 is a scatter plot that shows the correlation between two users and services. The figure shows that in the latent features there can be a number of outliers, which effects PCC as it assumes a linear relationship.

As it is difficult to justify removing the latent outliers, an alternative approach is used, in particular, a non-parametric correlation coefficient, Kendall's Tau, which is much less sensitive to outliers [212]. Kendall's Tau measures the degree of monotonic relationship between variables, and calculates the dependence between ranked variables, which makes it feasible for non-normally distributed data. The similarity between two users u_i and u_j is defined by performing Kendall's Tau on their l-dimensional latent feature



(A) User Latent Feature Histogram

(B) Service Latent Feature Histogram

FIGURE 3.9: Latent Feature Histogram

vectors V_i and V_j. Let $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ be a set of observations from V_i and V_j. Any pair of observations from V_i and V_j are said to be concordant if the ranks for both elements agree i.e., if both $x_i < x_j$ and $y_i < y_j$, or if both $x_i > x_j$ and $y_i > y_j$. They are said to be discordant, if $x_i > x_j$ and $y_i < y_j$, or $x_i > x_j$ and $y_i < y_j$. If $x_i = x_j$, or $y_i = y_j$, the pair is a tie [212].

$$\tau = \frac{c-d}{c+d} \tag{3.20}$$

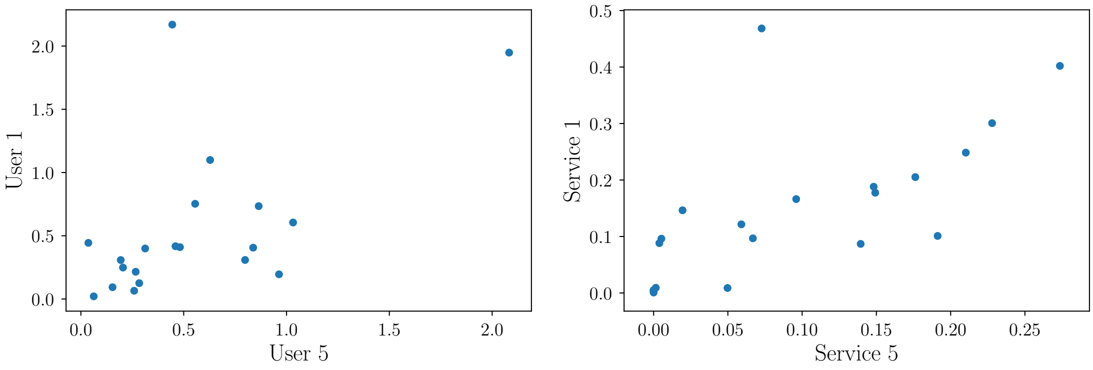
where,
c = the number of concordant pairs,
d = the number of discordant pairs.

If ties are present among the two ranked variables, the following equation shall be used instead:

$$\tau = \frac{c-d}{\sqrt{n(n-1)/2 - T}\sqrt{n(n-1)/2 - U}} \tag{3.21}$$

$$T = \sum_t t(t-1)/2 \tag{3.22}$$

$$U = \sum_u u(u-1)/2 \tag{3.23}$$



(A) User Latent Feature

(B) Service Latent Feature

FIGURE 3.10: Latent Feature Correlation

where,

t = number of observations of variable x that are tied,

u = number of observations of variable y that are tied.

The same formula can also be used to compute the similarity between the services through the use of concordant and discordant pairs.

3.6.2.3 Missing QoS Value Prediction

The similarity values can then be used to identify similar neighbours to the current user by ordering the values. Kendall's Tau falls into the interval [-1, 1], where a positive value denotes similarity and a negative value denotes dissimilarity. QoS usage experience from dissimilar users negatively effects the prediction accuracy, so users with a negative Tau value are excluded from the similarity set. Only the QoS usage experience of users with the top-k largest Tau values are used to predict the QoS of the user, as these are the most similar. The top-k similar users for user u_i are defined as Ψ_i:

$$\Psi_i = \{u_k | S(u_i, u_k) > 0, rank_i(k) \leq K, k \neq i\} \tag{3.24}$$

where $rank_i(k)$ is the ranking position of user u_k based on user u_i and K denotes the size of set Ψ_i. The top-k IoT services for service s_j can be denoted Φ_j by:

$$\Phi_j = \{s_k | S(s_j, s_k) > 0, rank_i(k) \leq K, k \neq j\} \tag{3.25}$$

where $rank_j(k)$ is the ranking position of service s_k based on service s_i and K denotes the size of set Ψ_j. To predict missing values for w_{ij} in the user service matrix, user-based approaches use the values from the top-k similar users as follows:

$$w_{ij} = \bar{w}_i + \sum_{k \in \Psi_i} \frac{S(u_i, u_k)}{\sum_{a \in \Psi_i} S(u_i, u_a)} (w_{kj} - \bar{w}_k) \tag{3.26}$$

where $\bar{w}_i$ and $\bar{w}_k$ are the average observed QoS values of different services by users u_i and u_k respectively. For service-based approaches, entry values of top-k similar services are employed for predicting the missing entry w_{ij} in a similar way:

$$w_{ij} = \bar{w}_j + \sum_{k \in \Phi_j} \frac{S(i_j, i_k)}{\sum_{a \in \Phi_j} S(i_j, i_a)} (w_{ik} - \bar{w}_k) \tag{3.27}$$

where $\bar{w}_j$ and $\bar{w}_k$ are the average observed QoS values of different services s_i and s_k by different users respectively. The predicted values using Eq. 3.26 and Eq. 3.27 are combined for a more precise prediction in the following equation:

$$w^*_{ij} = \lambda \times w^u_{ij} + (1 - \lambda) \times w^s_{ij} \tag{3.28}$$

where w^u_{ij} denotes the predicted value by user-based approach and w^s_{ij} denotes the predicted value by the service-based approach. The parameter λ controls how much the hybrid prediction relies on user-based or service-based approach. The proposed algorithm is summarised in Algorithm 1. The input W is the user-service matrix, l is the number of latent features and λ controls how much the hybrid prediction relies on user-based or service-based approach. The output of the algorithm is W^*, the completed matrix.

3.6.3 Autoencoder

An autoencoder is a type of artificial neural network, which can be used to learn efficient data encodings and latent features [216]. An autoencoder learns to compress data from the input layer to the hidden layer and decode the hidden layer into something that closely matches the original data, as can be seen in Figure 3.11. The encoder receives input data and transforms it into a new representation called a latent variable. The decoder receives this latent variable and tries to reconstruct the original input minimising the reconstruction error between the input and the output [217]. As the hidden layers contain fewer neurons than the input, the autoencoder engages in dimensionality reduction through Non Linear Principle Component Analysis (NLPCA) [218].

Algorithm 1 IoTPredict Algorithm

```
Input: $W, l, \lambda$
Output: $W^*$
Learn V and H by applying latent feature learning on W
for $all(u_i, u_j) \in U \times U$ do
    calculate the similarity $S(u_i, u_j)$ by Eq. 3.20
end for
for $all(s_i, s_j) \in S \times S$ do
    calculate the similarity $S(s_i, s_j)$ by Eq. 3.20
end for
for $all(i, j) \in \Lambda$ do
    construct similar set $\Psi_i$ by Eq. 3.24
    construct similar set $\phi_i$ by Eq. 3.25
end for
for $all(i, j) \in \Omega - \Lambda$ do
    calculate $w_{ij}^u$ by Eq. 3.26
    calculate $w_{ij}^s$ by Eq. 3.27
    $w_{ij}^* = \lambda \times w_{ij}^u + (1 - \lambda) \times w_{ij}^s$ by Eq. 3.28
end for
```

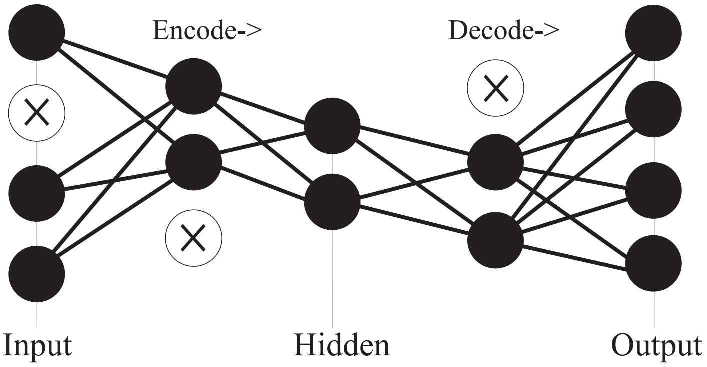


FIGURE 3.11: Autoencoder Architecture with Dropout

The main idea behind the latent variables extraction is that some knowledge relative to user preferences is hidden in raw data and dimensionality reduction techniques can exploit this [219]. Hence, extracting the data needed to explain the outcomes of the ratings matrix into users and items representations. The latent features also help to deal with the sparsity problem in IoT environments.

A classical autoencoder is typically implemented as a one hidden-layer neural network that takes a vector $x \in \mathbb{R}^D$ as input and maps it to a hidden representation $z \in \mathbb{R}^K$ through a mapping function [216]:

$$z = h(x) = \sigma(W^T x + b), \tag{3.29}$$

where W is a $D \times K$ weight matrix and $b \in \mathbb{K}$ is an offset vector. The resulting latent representation is then mapped back to a reconstructed vector $\hat{x} \in \mathbb{R}^D$ through

$$\hat{x} = \sigma(W^{'} z + b^{'}) \tag{3.30}$$

The reverse mapping may optionally be constrained by tied weights, where $W^{'} = W$. The parameters of this model are trained to minimize the average reconstruction error

$$\underset{W,W^{'},b,b^{'}}{argmin} \frac{1}{n} \sum_{i=1}^{n} \ell(x_i, \hat{x}_i) \tag{3.31}$$

where ℓ is a loss function such as the square loss or the cross entropy loss.

Autoencoders are learned automatically from data examples. This means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that it does not require any new engineering, only the appropriate training data. This allows them to be used in a wide variety of domains such as anomaly detection [220], sentiment analysis [221] and activity recognition [222]. This is the case for user centric QoS approaches where the factors that effect the QoS such as network congestion or device changing to a power saving mode are contained in the latest QoS values. The model should be lightweight with a small training time to incorporate those changes in the model as soon as possible.

3.6.3.1 Stacked Autoencoder

Traditional autoencoders can often just become identity networks and fail to learn the relationships between data. Our contribution in this thesis is to tackle this problem using a stack of autoencoder layers combined with corrupting inputs pushing the network to denoise the final inputs [223], [224]. The input can be corrupted using a number of different approaches such as Guassian noise, Masking noise or Salt and Pepper noise [225]. This modifies a traditional autoencoder loss function to emphasize the denoising aspect of the network. It is based on two main hyperparameters α, β, which decide on whether the network would focus on denoising the input α or reconstructing the input β.

$$L_{2,\alpha,\beta}(x, \tilde{x}) = \alpha\Big(\sum_{j \in J(\tilde{x})} [nn(\tilde{x}_j) - x_j]^2\Big) + \beta\Big(\sum_{j \notin J(\tilde{x})} [nn(\tilde{x}_j) - x_j]^2\Big) \tag{3.32}$$

Where $nn(x)_k$ is the k^{th} output of the network, $\tilde{x}$ is the corrupted input x, J are the indices of the corrupted element of x. Another regularisation technique that is often used in deep neural networks is dropout [226]. It randomly drops out units from both the hidden and visible layers. Figure 3.11 shows an example of a stacked autoencoder after applying dropout. The neurons marked with x have been dropped and contain no input or output connections. Each unit in the network is retrained with fixed probability p, which is evaluated for the optimal values in Section 5.3.1.6. Hidden layer activation before dropout is according to the following equation:

$$h^k(x) = g(a^k(x)) \tag{3.33}$$

where layer k ranges from 1 to L, which is the label of the hidden layer. g is the activation function sigmoid that is used in our experiments. The equation after dropping out units in hidden layers with probability p is:

$$h^k(x) = g(a^k(x)) \odot m^k \tag{3.34}$$

With $k = L + 1$, output layer is:

$$h^{L+1}(x) = o(a^{L+1}(x)) \tag{3.35}$$

where o is also an activation function.

A stacked autoencoder with additional layers for greater expressive power is used in the experiments. The use of stacked autoencoders captures a useful hierarchical grouping of the input. An example from the use of autoencoders in vision problems is that the first layer of a stacked autoencoder tends to learn first-order features in the raw input such as the edges of an image. The second layer can then use these features to learn second-order features such as what edges tend to occur together to form contours or corner detectors, with additional layers using these second-order features [227]. One of the best ways to obtain good parameters for a stacked autoencoder is to use greedy layer-wise training [228]. The raw inputs are trained on the first layer to obtain parameters and transform the raw input into a vector consisting of the activation of the hidden units. The second layer is then trained on this vector and this process is repeated for subsequent layers, using the output of each layer as input for the subsequent layer. The weights learned during the training process are then used to make predictions for users with missing QoS values.

Designing an autoencoder architecture for the edge is an interesting problem as there is a balance between the limitations of the devices that you are training on and the ability to deliver accurate QoS predictions. The autoencoder approach is designed to have a small number of neurons to reduce training and invocation time [229]. Multiple different numbers of layers were tested to identify the impact that this had on the composition accuracy and testing time of the stacked autoencoder. The use of dropout on each layer while training was found to avoid overfitting while maintaining better prediction accuracy compared to adding noise to the inputs.

The autoencoder is implemented with 4 fully connected layers. The encoding part of the network has two layers of 20 and 10 neurons and the decoding part of the network has two layers of 10 and 20 neurons. The sigmoid activation was used between the layers of the network. The dropout value for each of the layers is evaluated in Section 5.3.1.6. The root mean squared error is used as the loss function and RMSprop as the optimiser with learning rate = 0.01 and weight decay = 0.5. Experiments were conducted using Adam optimisation [230], but better results were found using RMSprop. The network is trained for 20 epochs (passes through the entire training dataset). These parameters were found after extensive experimentation with alternative hyperparameters.

3.7 Chapter Summary

Figure 3.12 shows the updated kiviat diagram from Figure 2.7 for the TTD approaches. The TTD approaches are quite distinct. The Persistence approach can be deployed on the IoT device, while having basic model complexity. On the other end of the scale, approaches such as LSTM have complex models but have to be deployed in the cloud. The noisy-echo state network approach that we develop increases the area covered in the diagram by deploying a more complex model at the edge of the network. Whether this approach can improve prediction accuracy compared to existing approaches while maintaining a reduced training time is evaluated in Chapter 5.

Figure 3.13 shows the updated kiviat diagram from Figure 2.8 for the TTR approaches. The existing TTR approaches have been quite similar in having a matrix factorisation-based approach deployed in the cloud. This chapter proposed the IoTPredict approach designed to improve prediction accuracy using an alternative similarity metric. This was then expanded upon using a stacked autoencoder approach, designed to maintain prediction accuracy, while being deployed at the edge of the network reducing the response time. The stacked autoencoder approach allows for additional model

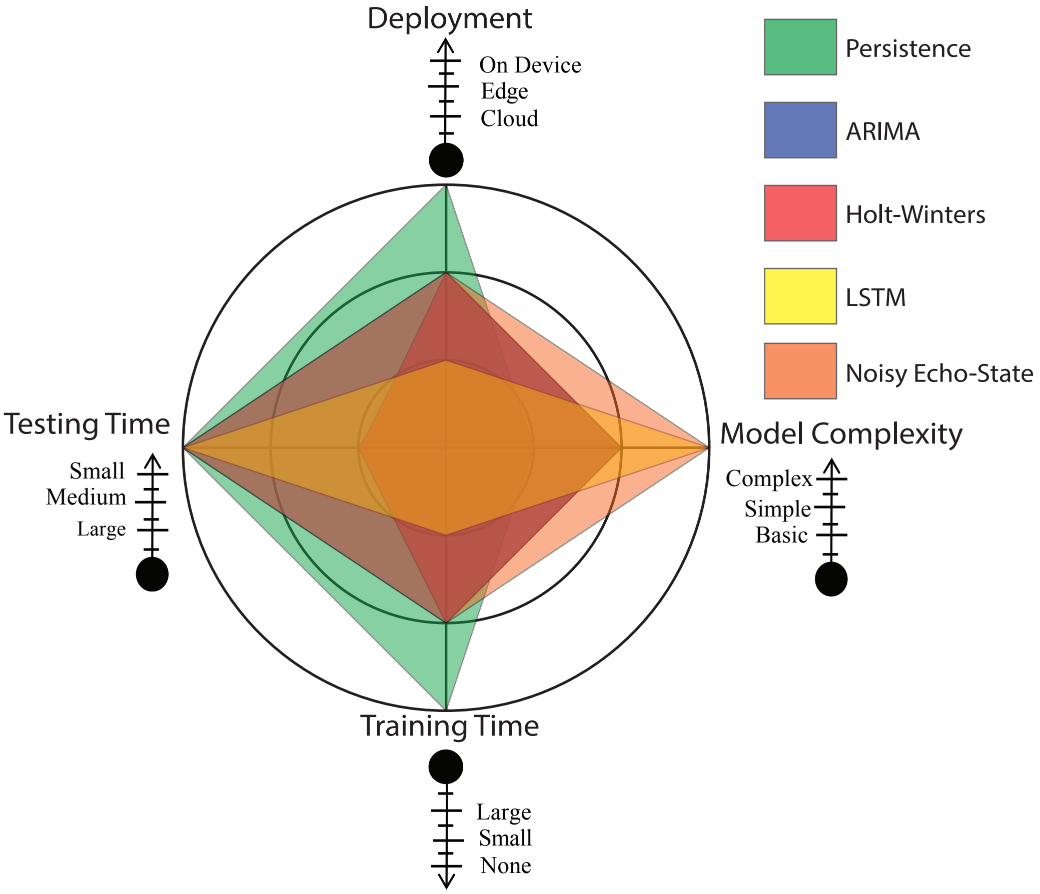


Figure 3.12: Kiviat Diagram of TTD Approaches Including the Noisy-Echo State Design

complexity at the edge of the network, which was not possible with existing matrix factorisation-based approaches.

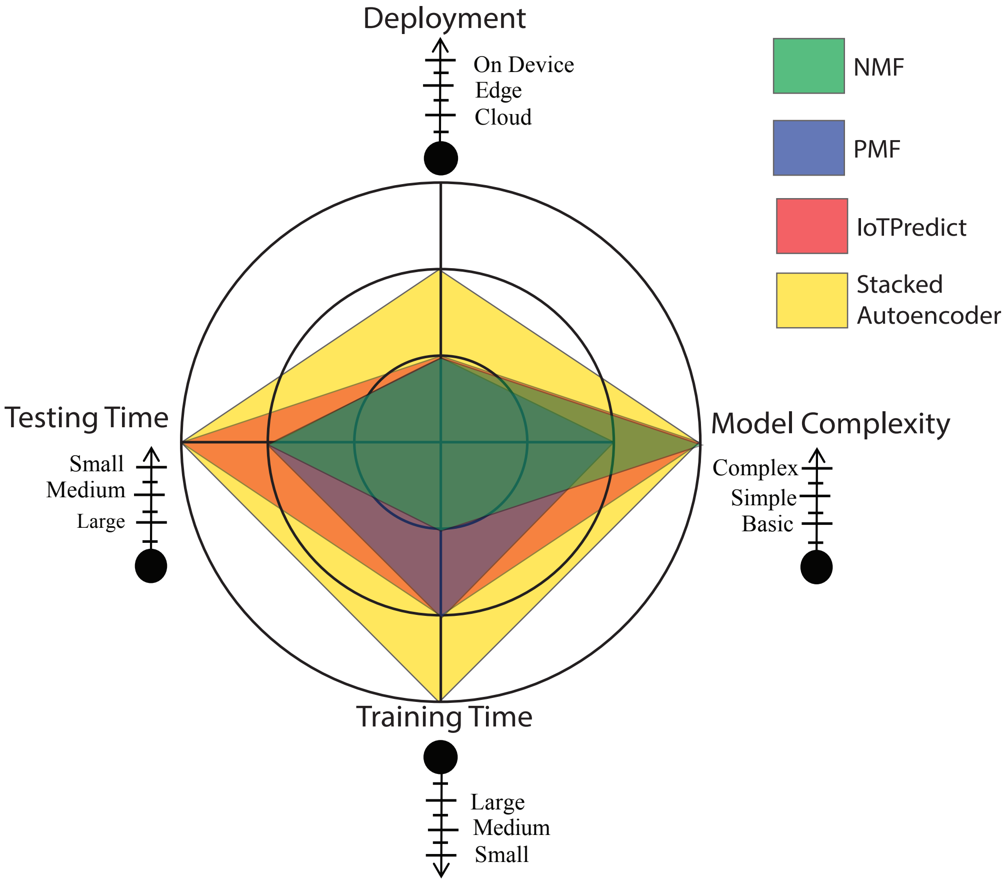


FIGURE 3.13: Kiviat Diagram of TTR Approaches Including the Stacked Autoencoder Design

Chapter 4

Implementation

Chapter 3 describes the design of the components of TTDR, the design decisions and required features that were taken into account when making those decisions. This chapter details the implementation of TTDR. TTDR is designed to be implemented in a QoS monitoring engine in the Service-centric network for URban-scale Feedback systems (SURF) middleware [13, 58, 179, 231]. The SURF middleware offers the necessary components for service discovery, QoS-aware service composition and optimisation, QoS monitoring and prediction and runtime service adaptation when one of the services fails during execution.

Section 4.1 introduces the SURF middleware and Section 4.2 describes the main components and the messages and information sent between the components. Section 4.3 introduces the data model that is used to manage the QoS parameters in the service description. Section 4.4 shows the sequence of execution to compose a service-based application and when the TTD and TTR algorithms are used by other components of the middleware. Section 4.5 describes the deep edge architecture on which the TTDR algorithms are deployed. Section 4.6 details the implementation and class diagrams of the TTD algorithm and Section 4.7 describes the implementation and class diagrams of the TTR algorithm. Section 4.8 provides a summary of the chapter.

4.1 SURF Middleware

The SURF middleware is a distributed service oriented middleware that can be deployed in a network of IoT Gateways [58]. As illustrated in Figure 3.6 the main components are the Service Discovery Engine (SDE), the QoS Monitor, which contains the Monitoring and Prediction Engine, the SLA Manager and Negotiator, and the Service Composition and Execution Engine (SCEE). The QoS monitor is the focus of this thesis and the component that we make a contribution to with the TTD and TTR algorithms. There are two additional utility components to facilitate the main middleware components: a Request Handler and a Service Registry. The QoS prediction algorithms are designed to be used independent of the specific service composition algorithm as long as it provides a list of candidate services. The other middleware components outside of the QoS monitor are described to show how the QoS prediction algorithms would interact with other components in a service oriented middleware. They were not implement as part of this thesis, but were used to help evaluate the accuracy of the QoS predictions.

The middleware is designed to be deployed on IoT gateways, specifically the deep edge architecture described in Section 4.5, to allow the prediction algorithms to be trained at the gateway on local data. The middleware can manage services from a number of different providers (i.e., web services, WSN services, and mobile service providers) as shown in Figure 3.6 and is deployed on the gateways in the deep edge architecture in Figure 4.4. The remainder of this section briefly describes each component.

4.1.1 Request Handler

Service consumers such as citizens in a smart city or software developers creating applications, access the available services through requests. In this thesis, a request is defined as $r = < I, O, QoS >$, consisting of the request inputs, outputs and QoS requirements [208]. The Request Handler (RH) is a utility module that obtains the users request and formalises it in a formation that can be used by the SDE to find the service composition plans that can functionally satisfy the component request. The non-functional QoS parameters are handled by the QoS Monitoring and Prediction Engine to ensure that the user gets the level of quality specified [56]. The request handler exposes an interface that receives requests, establishes a communication channel between the application and user and sends requests to the other components in the architecture.

4.1.2 Service Registry Engine

The Service Registry Engine (SRE) is a distributed component that can store the description information of services. The component facilitates the service discovery process [13]. Each service has a description $S_{desc} = < id, I, O, D >$ consisting of a service identifier (id), inputs (I), outputs (O) and domains (D). The QoS levels cannot be stored in the service description as they are dynamic and depend on user side factors such as network quality, which is why the QoS Monitoring and Prediction Engine are needed to ensure the required level of QoS is achieved. When a new service provider becomes available it can register its services in the gateway.

4.1.3 Service Discovery Engine

The Service Discovery Engine (SDE) is a distributed component used to perform the discovery of service composition configurations that can satisfy users' requests. This component is notified of users' request by the Request Handler component. When such a notification is received, the SDE searches for services in the distributed registry. The output of this search is a set of service composition configurations (or service plans) that satisfy the request from the functional perspective (i.e., input and output matching). Each service plan can have one or more services. In addition to the service components, each service plan includes the data flow and control flow relations between these service components [232]. The set of service plans are passed to the SCEE to choose the best plan according to the non-functional requirements (i.e., QoS parameters) provided by the QoS Monitor.

4.1.4 QoS Monitor

The QoS Monitor contains the Monitoring and Prediction engines that are used to provide the QoS parameters required by users in their service requests. The QoS Monitoring Engine captures the user-side QoS and stores the values in a distributed registry. The communication links for invoking the services are diverse, which may affect the personal QoS experience of users, so they are monitored using the monitoring engine. The QoS prediction engine contains the TTD and TTR algorithms. The TTD algorithm is used to forecast degradation of currently executing services. If one of the services is forecast to fail or degrade in quality a service degradation alert is sent to the SCEE and there is a need for a dynamic service reconfiguration. The service reconfiguration then uses the TTR algorithm to provide QoS predictions for the candidate replacement services. The services that satisfy the users QoS request are then chosen and the application is recomposed. The TTR predictions are also used in the initial service configuration to compose the application that will satisfy the users QoS requirements.

4.1.5 SLA Negotiator and SLA Manager

The SURF middleware contains two SLA components that deal with the SLA negotiation and compliance process: SLA Negotiator and SLA Manager. The SLA Negotiator analyses the agreement templates submitted by some service providers, selects the candidate service providers that have the potential to satisfy the requirements based on QoS predictions from the prediction engine. It then negotiates on behalf of service consumers to solve possible conflicts between service providers and consumers [233]. The SLA manager ensures compliance with the agreed terms at runtime, during the service execution. This component subscribes to performance degradation notifications from the QoS Monitoring Engine through a publish/subscribe broker. This triggers a SLA negotiation in case the performance does not comply with guarentee terms and service negotiation is available [231].

4.1.6 Service Composition and Execution Engine

The Service Composition and Execution Engine (SCEE) searches for the optimal service composition configurations using the QoS information that the QoS Monitor and QoS Prediction Engine provide. The SCEE is initialised using the set of service plans provided by the SDE. This component merges all these plans into a service dependency graph using an AND/OR graph structure [58]. In a dynamic environment, service components may have intermittent availability, or, if the devices that provide these service components are mobile, the service components may become permanently unavailable. Because of the frequent changes, finding a QoS-optimal service composition configuration (service plan) may be difficult. Also, a user may want to explore various trade-offs between the QoS attributes. The SCEE addresses these requirements by distributing the optimisation process over the available gateways [58]. Service component failures caused by the intermittent availability of the nodes are managed by requesting candidate service QoS values from the QoS prediction engine to choose suitable replacement services.

4.2 Component Diagram

Figure 4.1 shows the component diagram of the middleware and how the components communicate with each other. Users can request services through a Service Request and Service Providers can register services in the middleware through a Service Registration Request. The RH facilitates the interaction between the middleware and the

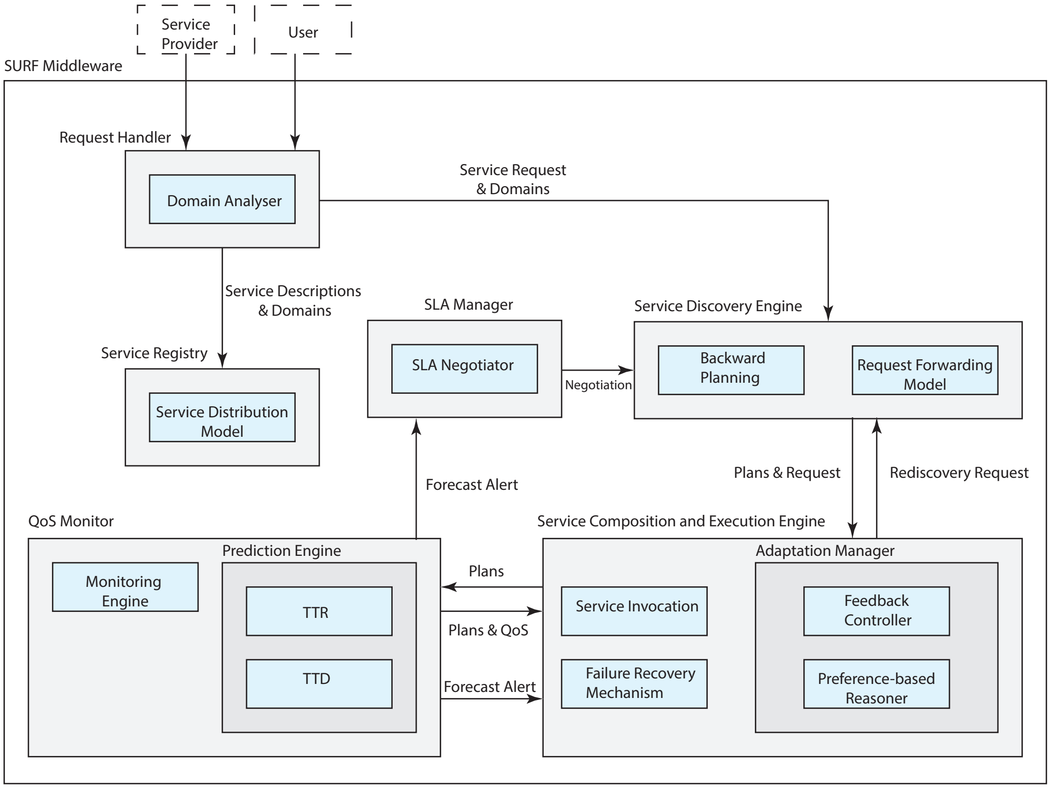


Figure 4.1: Component Diagram

service providers and users. The RH includes a domain analyser that analyses a service registration request or service request to identify its domain. The SRE receives the Service Descriptions & Domains and follows a distribution model based on urban context as well as their identified domains [13]. SRE relies on a P2P overlay network to distribute registered services and form a federated view of distributed services in each IoT gateway. The SDE component discovers services registered in the middleware. It receives a service request and domains by the request handler and performs a distributed search based on the urban context. This search aims to reduce the search space using domain and city information to provide efficient discovery. It performs matchmaking by applying backward planning to discover dependencies between services satisfying functional requirements defined by the user as input/output parameters in the request [208]. The discovered services are passed to the composition engine to be executed.

The SCEE is responsible for the execution of the composition plans generated by the SDE. This component ensures a scalable and resilient service composition execution by using a decentralised composition mechanism to perform the invocation of composition plans and a backtracking approach when the invocation fails [208]. It uses a loop controller to maximise the global utility of the selected composition plan by adjusting the user preferences on the non-functional requirements. The QoS monitor contains the Monitoring and Prediction Engine that is responsible for making predictions for the QoS values. The predictions for the candidate services are generated using a collaborative filtering TTR approach with data from other users in the the environment collected by the monitoring engine. The monitoring engine is also used to monitor currently executing services. This time series dataset is then used by the TTD algorithm to forecast the QoS values for currently executing services. The TTD algorithm sends an alert to the SCEE if a degradation in service quality is forecast. This alert allows the SCEE to switch to an alternative candidate service using the predicted values from the TTR algorithm before the user notices a change in the QoS. If there are no available services that are suitable to switch to then the SLA negotiator tries to negotiate with service providers and sends the negotiated values to the service discovery engine.

4.3 Data Model

Figure 4.2 presents the data entities that are used to exchange messages between the components and to store information about the QoS in the service descriptions. Each entity has simple attributes with their respective types, they can be optional or mandatory. Mandatory attributes are marked with a line (-) and optional are marked with a star (*) in the diagram. Each entity can also have attributes, which type is another entity, represented by relations between the entities. Whether the attributes are compulsory or not is expressed through the relation cardinality. The entities are:

- ServiceDescription represents the description of a service.
- ServiceParameter represents the service functionality as input/output parameters. One service description must have at least 1 input and 1 output.
- QoSParameter represents quality of service information. One service description can have 0 or multiple QoS Parameters.
- Domain represents the service domain information. One service description can have 0 or multiple domains.
- Location represents the location of a service. One service description can have 0 or 1 location.
- Type represents the service type. One service description must have 1 service type.

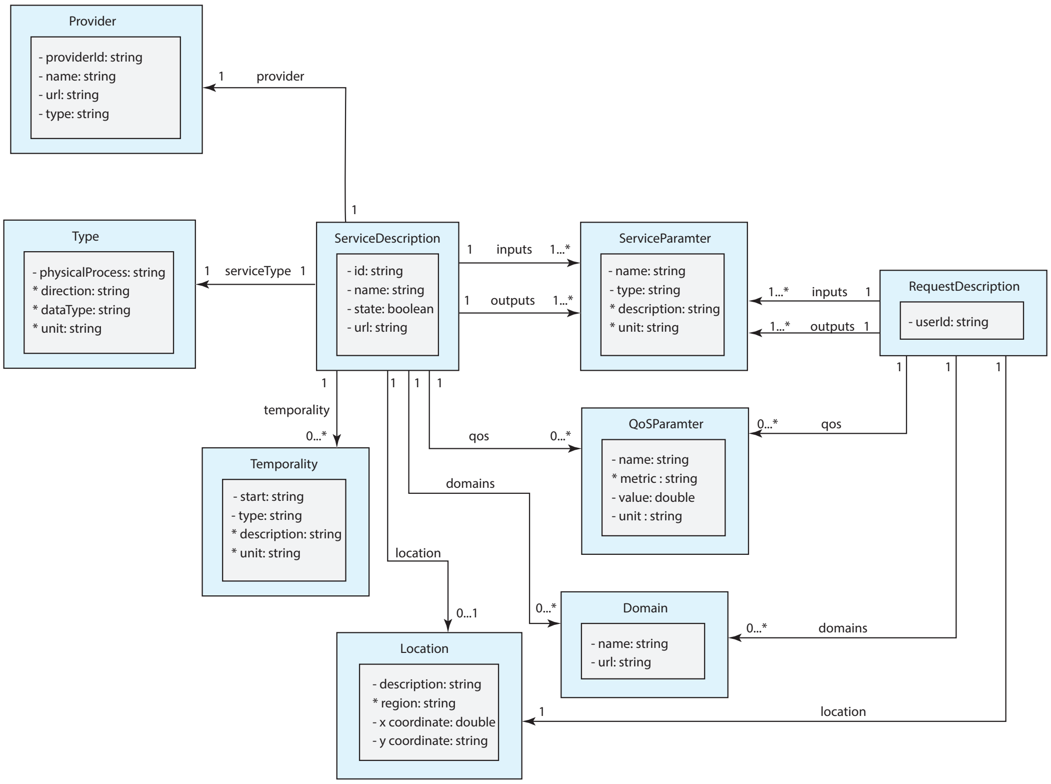


Figure 4.2: Data Model

- Provider represents the service provider information. One service description must have 1 service provider.
- RequestDescription represents a user request in the middleware. One request description must specify at least one input and one output as functional requirements. Additionally, it can have optional non-functional requirements such as QoS parameters, domains, and location.
- Temporality represents the operation period of the service.

4.4 Sequence Diagram

Figure 4.3 shows the sequence diagram when a user requests a service from the middleware. This diagram gives a detailed flow of when each processes happens and how the individual QoS prediction algorithms feature in the overall service composition flow. The steps to handle a user request are:

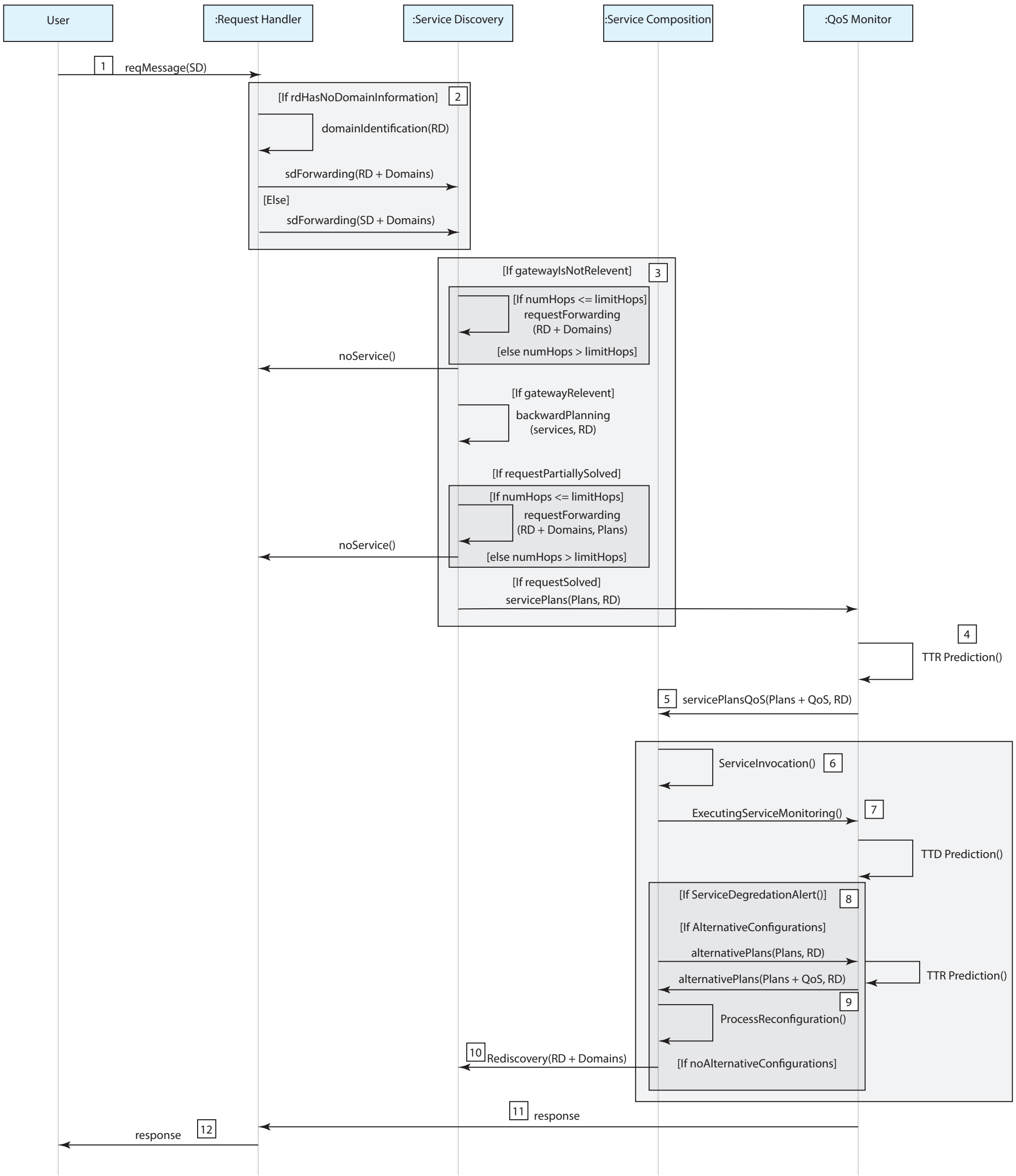

User
:Request Handler
:Service Discovery
:Service Composition
:QoS Monitor
1
reqMessage(SD)
[If rdHasNoDomainInformation]
2
domainIdentification(RD)
sdForwarding(RD + Domains)
[Else]
sdForwarding(SD + Domains)
[If gatewayIsNotRelevent]
3
[If numHops <= limitHops]
requestForwarding
(RD + Domains)
[else numHops > limitHops]
noService()
[If gatewayRelevent]
backwardPlanning
(services, RD)
[If requestPartiallySolved]
[If numHops <= limitHops]
requestForwarding
(RD + Domains, Plans)
noService()
[else numHops > limitHops]
[If requestSolved]
servicePlans(Plans, RD)
4
TTR Prediction()
5
servicePlansQoS(Plans + QoS, RD)
ServiceInvocation()
6
ExecutingServiceMonitoring()
7
TTD Prediction()
[If ServiceDegredationAlert()]
8
[If AlternativeConfigurations]
alternativePlans(Plans, RD)
TTR Prediction()
alternativePlans(Plans + QoS, RD)
9
ProcessReconfiguration()
10
Rediscovery(RD + Domains)
[If noAlternativeConfigurations]
11
response
response
12

Figure 4.3: Sequence Diagram

1. User sends a request message to the RH. This message includes a request description with the functional and non-functional requirements that the middleware needs to satisfy.

2. The domain analyser checks the user's service request description (RD) for domain information, if available it forwards the RD to the SDE. Otherwise, the analyser applies topic modelling on the RD and identifies the topic or topics of the service description. Finally, the domain analyser compares the extracted topic or topics with the domain knowledge base and identifies the domain information of the RD, and forwards it to the SDE.

3. SDE receives discovery request from the RH. It extracts the domain information and compares it with the gateway's domains. If the gateway is relevant (i.e., the surrounding places near the gateway offer city services that include the request domains), SDE performs backward planning to discover services according to input/output parameters. If the request is solved, after the backward planning, the SDE sends the list of services (i.e., plans) to the Monitoring component. If the request is partially or not solved, the SDE continues the distributed search, forwarding the request to the top-k relevant gateways. If the gateway is not relevant for the request, the SDE forwards the request to non-similar gateways (i.e., gateways that have different domains than the not relevant gateway). The request is forwarded until the limit of hops is reached.

4. The TTR algorithm in the Prediction Engine in the QoS monitor makes the prediction for user side QoS parameters such as response time and throughput of the candidate services. The predictions are then sent with the plans to the SCEE.

5. The SCEE receives a list of composition plans from the Prediction Engine. Each plan contains a set of services. Each service in a composition plan needs to be invoked to solve the composition request. The servicePlansQoS (Plans + QoS, RD) message is sent by the Prediction Engine to SCEE and represents the set of service composition plans generated by the service discovery messages. This message also includes the RD.

6. The SCEE enters the service composition execution loop, which performs the execution of the service composition. Based on user preferences, and available resources in the environment, the SCEE component selects a composition plan, which maximises the satisfaction of user non-functional requirements. The execution is decentralised. This means that the execution (invocation) of a service will be done in the gateway which registers this service.

7. The Monitoring Engine is used to monitor the current executing services. The data from this monitoring is then used to train the TTD forecasting algorithm to make predictions whether the services may be about to fail or degrade in quality.

8. If the TTD algorithm in the Prediction Engine sends a ServiceDegradationAlert() then the SCEE looks for alternative configurations. The alternative plans are then sent to the Prediction Engine and updated with the latest QoS values using the TTR algorithm.

9. The processReconfiguration() takes the list of alternative configurations and updates the currently executing application to use the services with the new QoS.

10. The rediscovery request is performed when the invocation of a service failed and there are no alternative composition plans. The SCEE sends a rediscovery message, which has the RD and domains that need be covered by the request. The SDE receives a rediscovery request from the SCEE. It performs the same process as in step 3.

11. Response message represents the output generated by the SCEE. This message is sent to the RH component.

12. Response message represents the result produced by the middleware components to the user request.

4.5 Deep Edges

Figure 4.4 shows the deep edge architecture. This architecture uses a number of embedded GPUs for gateways. The ability to train and update models one hop away from data generation has a number of benefits and provides some solutions to the challenges outlined in Section 1.2. The devices that are used are Jetson Tx2's that have access to a 256-core NVIDIA Pascal^{TM} GPU, Dual-Core NVIDIA Denver 2 64-Bit CPU and 8GB 128-bit LPDDR4 Memory, while only consuming 7.5W of power. These embedded GPUs allow for much more additional processing and reduced training time at the edge of the network compared to traditional IoT gateway devices such as Raspberry Pi's and Intel Galileo's shown in the small scale scenario in Figure 1.1, which fulfills Objective 3. These devices have limited memory and no GPU so do not allow for the training of more complex prediction models that can be used to create more accurate service compositions. This improves the limited resources that were available for training deep learning models in traditional IoT architectures at the edge of the network (C.5).

The Deep Edge architecture has been used in this thesis to improve the training resources available for prediction algorithms used in the QoS Monitor. As the algorithm is deployed at the edge, predictions must be made for both IoT services in the surrounding environment and web services available online, to compose a wide variety of service-based applications. For example, we can create an application that does some processing and storage of data from IoT sensors. The data is available on the IoT devices and the processing and storage of the data is available from a cloud service

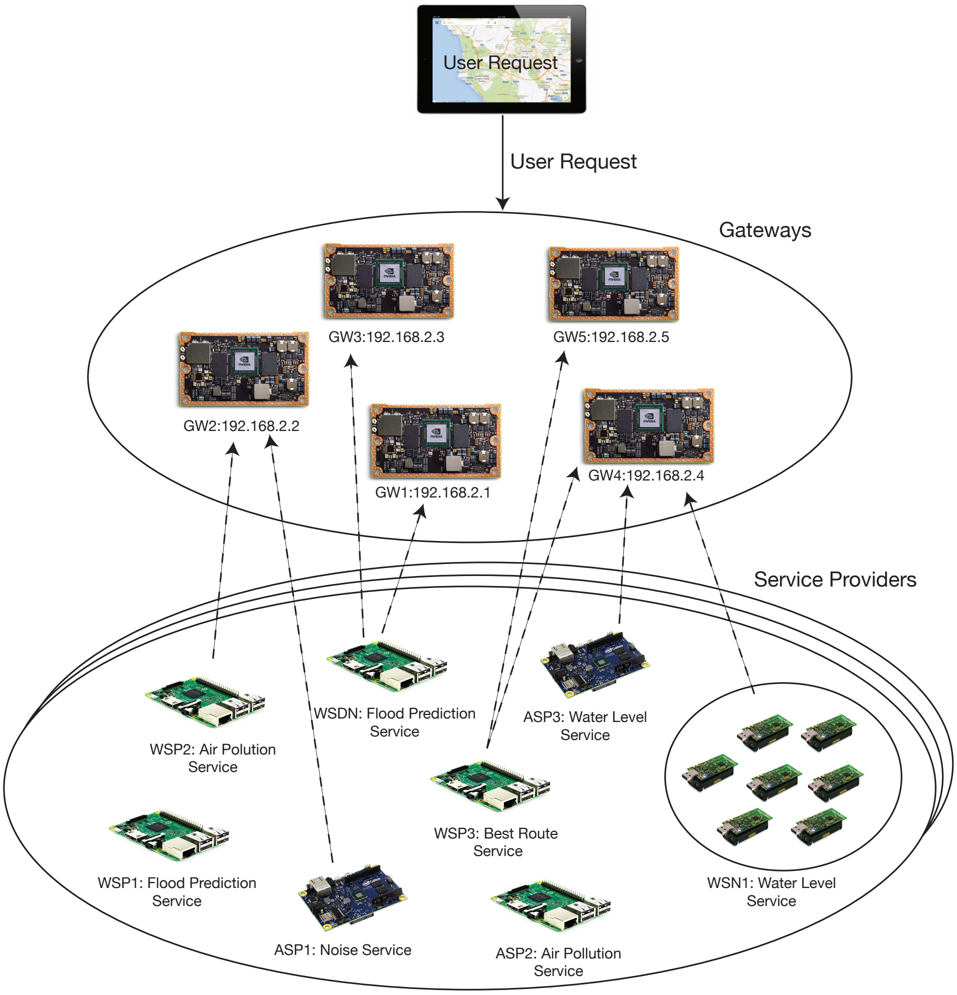


Figure 4.4: Deep Edge Architecture

provider. The TTDR algorithms on the edge act as a link between these two planes and can make predictions for the QoS of both service types. This allows a middleware to compose a reliable application using the benefits of cloud, edge and IoT devices. The architecture can also be used to perform additional processing for more complex service composition and service registry models. For example, a deep reinforcement learning algorithm can be used to optimize the service execution compared to the lower-powered ant colony-based approach [234]. The SRE can also use more recent deep learning-based approaches for domain classification [235].

One of the simplest ways to reduce the tolerable delay and jitter (C.2) to the level needed in modern Urban Intelligence applications, such as augmented reality is to reduce the distance of the communication. When the predictions are located one hop away at the edge, the response time and failure rate of the communication process are reduced, making it easier to provide the tolerable delay required by this challenge. The increased accuracy from these deep learning models one hop away from data generation will have a large impact for future applications. For example, autoencoders and deep convolutional networks can be used to provide face recognition in CCTV cameras for emergency response [236]. This reduced latency and jitter with the combination of increased object detection accuracy from deep learning algorithms has the potential to make it possible for applications such as augmented reality to work effectively as part of a modern smart city.

Another benefit to training and updating these models at the edge of the network is the reduced amount of traffic sent to the cloud (C.3). This is helpful to avoid congestion on the network as all users are not reporting their data to the cloud. This also helps to improve the privacy of users, which is important with the introduction of the General Data Protection Regulation (GDPR) on the 25^{th} May 2018 as part of the EU Data Protection Directive [237]. Users have become much more interested in what data is being collected about them, how that data is stored and who will have access to their data. There is increasing reluctance to release raw sensor data to an IoT cloud hub, and users and organisations want finer-grain control over the release of that data [238]. Users should be able to delete any data, which they deem to be sensitive and providers should use denatured data with faces in images being blurred and sensor readings being coarsely aggregated or omitted at certain times of day or night [239]. Current IoT architectures for urban reasoning, in which data is transmitted directly from sensors to a cloud hub makes such fine grained control impossible. On the other hand, an edge device can run trusted software modules called privacy mediators that execute on the device and perform denaturing and privacy-policy enforcement on the sensor streams [240]. Edge computing can provide a foundation for scalable and secure privacy that aligns with natural boundaries of trust, while still allowing for urban reasoning on the denatured data.

As applications become more dependent on the cloud, they increase their vulnerability to cloud outages. The assumption that there is always good end-to-end network quality and few network or cloud failures is not always applicable. This can happen in countries with a weak network infrastructure or a cyber-attack being carried out on the cloud provider such as denial of service [241]. It can also happen through human error from the cloud provider such as the outage of the Amazon S3 web service due to a typo [242], which can have catastrophic effects. Edge computing has the potential to alleviate cloud outages and provide a fallback service that can temporarily mask cloud inaccessibility. During a failure, the edge device could serve as a proxy for the cloud and perform critical services [239]. This allows urban reasoning applications to function in a smart city and to provide services even when there is a cloud outage.

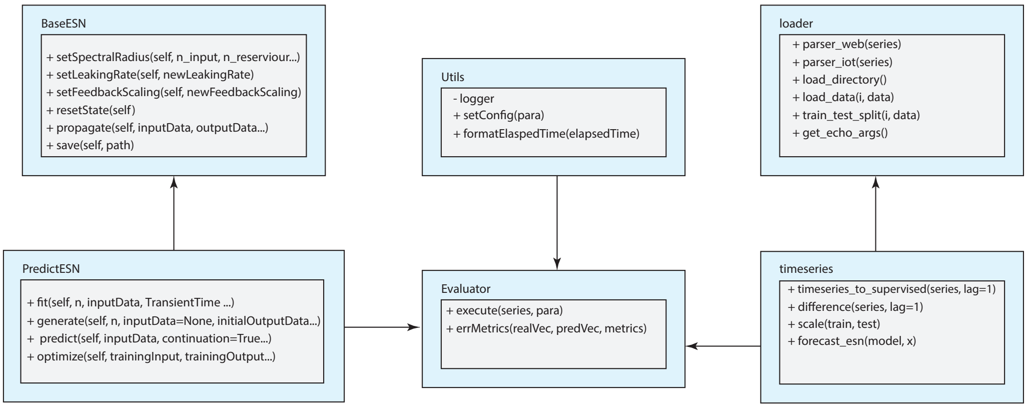

BaseESN
+ setSpectralRadius(self, n_input, n_reserviour...)
+ setLeakingRate(self, newLeakingRate)
+ setFeedbackScaling(self, newFeedbackScaling)
+ resetState(self)
+ propagate(self, inputData, outputData...)
+ save(self, path)
Utils
- logger
+ setConfig(para)
+ formatElaspedTime(elapsedTime)
loader
+ parser_web(series)
+ parser_iot(series)
+ load_directory()
+ load_data(i, data)
+ train_test_split(i, data)
+ get_echo_args()
PredictESN
+ fit(self, n, inputData, TransientTime ...)
+ generate(self, n, inputData=None, initialOutputData...)
+ predict(self, inputData, continuation=True...)
+ optimize(self, trainingInput, trainingOutput...)
Evaluator
+ execute(series, para)
+ errMetrics(realVec, predVec, metrics)
timeseries
+ timeseries_to_supervised(series, lag=1)
+ difference(series, lag=1)
+ scale(train, test)
+ forecast_esn(model, x)

FIGURE 4.5: TTD Class Diagram

When the failure is repaired, actions committed to the edge device can be propagated to the cloud for reconciliation.

4.6 TTD Implementation

The TTD algorithms and evaluation code is implemented in Python 3.6. This language is chosen as it has excellent libraries and frameworks for machine learning and data analysis. Python is also available on multiple platforms, which makes it easy to write code on a desktop computer and deploy the same code to an edge device using Pipenv to manage the dependencies from all the modules.

A BasicESN class is used to implement the basic functions of creating an echo state network such as setting the spectral radius, leaking rate, feedback scaling and propagation. From this base class we derive the PredictESN class that implements the noisy-echo state network approach. This class includes methods to generate the dynamic reservoir. The class also includes a fit function that is used to fit the training data to the ESN. The predict function is used to predict the output signal using the signals generated in the dynamic reservoir.

A number of utility classes are implemented to mange the experimental evaluation and the loading and preprocessing of data. The loader class contains a number of functions to load and parse data from the IoT and web service datasets. As the datasets from IoT and web services are in different formats, a separate parser is required to load the data correctly. The timeseries class contains a number of functions that are used to preprocess the time series into the format needed for the experiments, such as differencing the time series and transforming the timeseries to a supervised learning

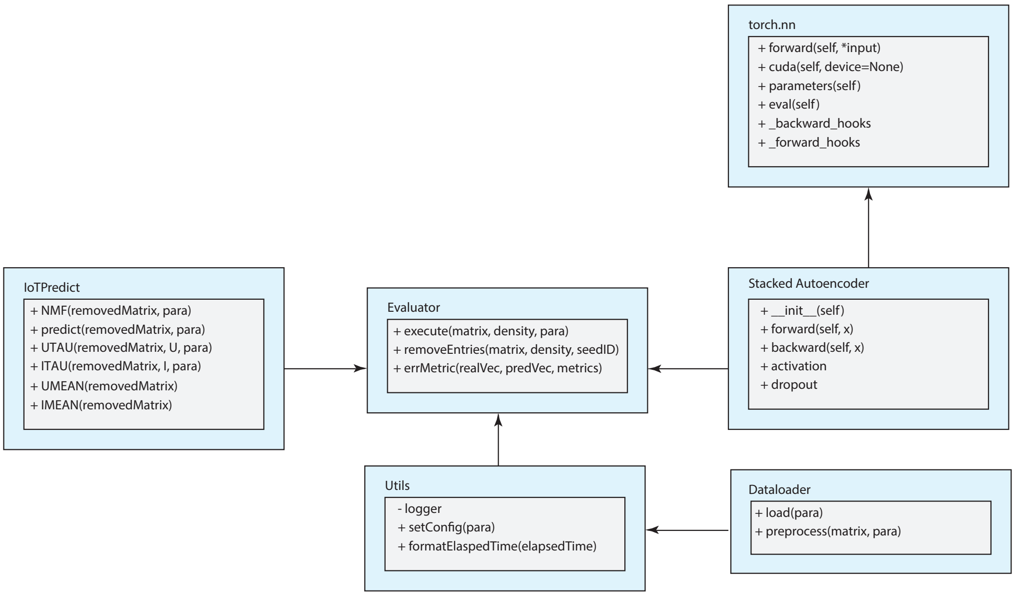

torch.nn
+ forward(self, *input)
+ cuda(self, device=None)
+ parameters(self)
+ eval(self)
+ _backward_hooks
+ _forward_hooks
IoTPredict
+ NMF(removedMatrix, para)
+ predict(removedMatrix, para)
+ UTAU(removedMatrix, U, para)
+ ITAU(removedMatrix, I, para)
+ UMEAN(removedMatrix)
+ IMEAN(removedMatrix)
Evaluator
+ execute(matrix, density, para)
+ removeEntries(matrix, density, seedID)
+ errMetric(realVec, predVec, metrics)
Stacked Autoencoder
+ __init__(self)
+ forward(self, x)
+ backward(self, x)
+ activation
+ dropout
Utils
- logger
+ setConfig(para)
+ formatElaspedTime(elapsedTime)
Dataloader
+ load(para)
+ preprocess(matrix, para)

Figure 4.6: TTR Class Diagram

problem. The utils class contains functions for managing the timing and logging information. The evaluator class is used to evaluate the accuracy of the algorithms on the time series and calculate the specific error metrics.

4.7 TTR Implementation

The TTR algorithms and evaluation code is also implemented in Python 3.6. This language is chosen as it has access to deep learning libraries such as torch [243] that can be used to implement neural architectures such as the autoencoder approach. It also allows for the reuse of some the previous utility modules from the TTD implementation to manage the logging, elapsed time and loading configurations.

The stacked autoencoder approach is implemented using the pytorch library. This module provides a number of utility functions to allow for training on GPU cuda cores. Forward and backward hooks are executed during the forward and backward pass to identify the value of the gradients when debugging. The stacked autoencoder class inherits from this base class and creates the architecture of the network in the forward and backward pass. The activation and dropout are also specified and passed to the forward and backward pass. The IoTPredict algorithm has a base NMF function that is used to generate the latent features that are used in the model. There are also some additional functions to calculate the mean of the service and users matrix.

A number of additional utility classes are implemented to allow for the evaluation of the algorithms. The dataloader class provides functions to load the matrix of values and also preprocess the values based on a list of parameters. The utils class is used to log information and set the configuration and elapsed time. The evaluator class is then used to calculate the accuracy of the algorithms using the error metrics. It also provides the function to remove entries from the initial full matrix to a specific density, which allows for the evaluation of the algorithms at different matrix densities.

4.8 Chapter Summary

This chapter outlined the implementation details of the TTDR algorithms and also how these algorithms can be used in combination with other service components as part of a distributed middleware. The chapter begins by describing the components of the SURF IoT middleware and the messages and commands that are passed between the components. It also presents the key requirements that are needed from other components in the middleware and the functionality required in the monitoring and prediction engine in the component diagram. The sequence diagram gives the flow of a service execution, illustrating when the TTD and TTR algorithms are executed. The data model that is used to specify the QoS parameters in the service description with other key parameters such as the provider, location and domain is also described.

The specific implementation details about the deployment of the algorithms on a deep edge architecture are then outlined. This gives additional processing power at the edge of the network that can be used to train the algorithms. The implementation details and class diagram of the TTD and TTR algorithms and the utility classes and functions used to evaluate them are also then described. This shows the overall structure of the implementation for both approaches.

Chapter 5

Evaluation

Previous chapters have introduced the challenges in current approaches as well as the features and design decisions that have been taken to address these challenges. In this chapter, the impact that the design decisions have on addressing those challenges is evaluated. The evaluation approach is designed to answer the research questions posed in Chapter 1, which focus on the prediction accuracy and the training and testing time of the individual algorithms for TTD and TTR.

The chapter is structured as follows: Section 5.1 introduces the evaluation approach of the TTD and TTR models for dynamic IoT environments. Section 5.2 describes the experimental setup, datasets, metrics, statistical tests and results for the TTD approaches and Section 5.3 describes the experimental setup and results for the TTR approaches. Section 5.4 presents a summary of the chapter.

5.1 Evaluation Approach

The evaluation approach is broken into two main parts. Section 5.2 evaluates the TTD approaches and Section 5.3 evaluates the TTR approaches. This section describes the evaluation approach taken for both reliability factors.

5.1.1 TTD Evaluation Approach

A forecasting accuracy study is proposed on edge devices to evaluate RQ.1, to what extent can the accuracy of forecasting to support TTD be improved, by using a lightweight model at the edge to incorporate recent changes in QoS. This study uses the datasets collected from dynamic IoT and web services to evaluate the forecasting accuracy of the prediction models for the next time step in the series. Additional users in the system are generated using Apache JMeter. This simulates the sudden changes in the demand for services that would be experienced with mobile users in the environment and is captured in the collected dataset. The algorithms are trained on an edge device, a Jetson Tx2 to test the training time of the algorithms at the edge of the network and evaluate how this would impact the prediction accuracy.

5.1.2 TTR Evaluation Approach

A collaborative filtering accuracy study is proposed using devices designed to be deployed at the edge of the network to evaluate RQ.2 to what extent can the time to receive predictions of TTR be reduced, by updating a model at the edge of the network, while maintaining QoS prediction accuracy. A QoS dataset that contains multiple user invocations of the same services from different locations is used to evaluate the accuracy of the prediction approaches at different matrix densities using standard error metrics. The predictions are also used as part of a service composition to evaluate the impact they would have in a real scenario. When the predictions are used as part of a service composition the ranking of the services is more important than the prediction accuracy as the service composition engine will only choose the top service from the list of candidate services that can fulfil the task. For example, consider a set of services 1, 2 and 3 with a response time of 5ms, 6ms and 7ms respectively. It is more important to rank the services in the correct order i.e., service 1 has the best QoS, then service 2 and 3, than having lower standard prediction error for each of the services but mistakenly predicting that service 3 has better QoS than service 1. RQ.2 is focused on the algorithm being deployed at the edge of the network so there is a focus on training time. To evaluate the training time for each of the approaches they are deployed on the Jetson Tx2. This allows us to evaluate how long the models will take to incorporate new information in the model.

5.2 TTD

This section presents the experimental setup and results of the TTD approaches.

Device	**Configuration**
Linux Workstation	Model: Dell Optiplex 755 OS: Ubuntu Linux 14.04 LTS CPU: 2 GHz RAM: 4 GB
Raspberry Pi	Model: Raspberry Pi 3 B OS: Raspbian Jessie Release 8.0 CPU: 1.2 GHz RAM: 1 GB
Arduino	Model: Arduino WeMos D1R2 Microcontroller: ESP8266EX

Table 5.1: Hardware Configurations of the Devices Used to Collect IoT Data

5.2.1 Experimental Setup

5.2.1.1 IoT Services Description

Table 5.1 shows the hardware configuration of the devices that are used to collect IoT data from services using real sensors. The elements in the Device column represent the type of the devices. These names will be used in the description of the rest of the experimental setup without referring to the specific model. The elements in the Configuration column represent a detailed hardware description of each device. The setup included a Raspberry Pi 3 Model B [244], two Arduino WeMos D1R2 [245] devices, a Linux Workstation and a Cisco router model Linksys EA6400 [246]. The Linux Workstation is used to simulate a number of virtual users using Apache JMeter version 3.3 [247]. For the services, 10 virtual users are allocated to invoke the service. This invocation was initiated according to a Poisson Timer with $\lambda = 100$ ms and a delay offset of 300 ms.

In addition to the devices presented in Table 5.1, the following sensors are used to collect real physical data: a HC-SR501 Infrared Motion Sensor, a MQ-2 Smoke/LPG/CO Gas Sensor, a BMP180 Barometer Pressure/Temperature/Altitude Sensor, a DHT11 FR4 Temperature/Humidity Sensor Module and a Keyes KY-018 Photo resistor Module. These sensors are connected to the Arduino devices using the I/O pins, and their data offered as RESTful services. Figure 5.1 shows the layout of the devices used to collect the data. The details about the connection of the sensors to the Arduino boards are omitted to avoid overloading the figure and are represented by S1, S2, S3 and S4. Instructions of how these sensors can be connected to an Arduino board can be provided through a number of documented resources (e.g., Arduino [248]).

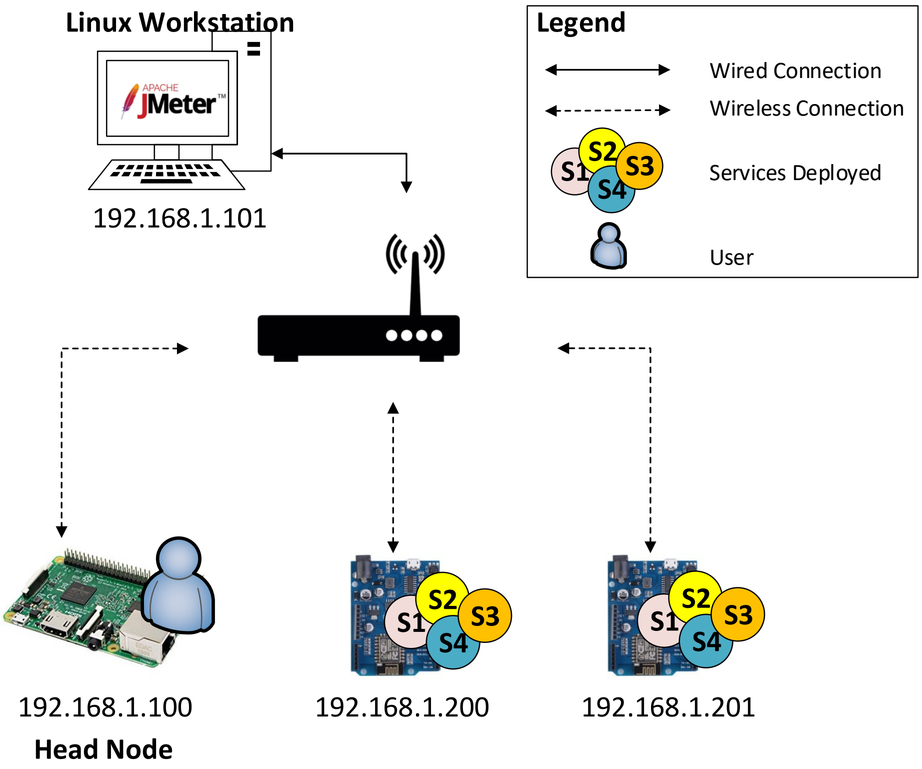


FIGURE 5.1: IoT Services Dataset Collection Setup

All devices used in the experiments are connected through a router. Two different types of connections are used: the Raspberry Pi and Arduinos are connected wirelessly to the router, while the Linux Workstation uses a wired connection. The ESP8266 Wi-Fi library [249] is used to establish a wireless connection between the Arduinos and the router, and the standard Wi-Fi driver is used for the Raspberry Pi device. For the services, we allocate 10 virtual users to invoke the service using JMeter to increase the workload. The device, which acts as the user, is four hops away from the services and is invoked every 30 seconds for 1 month. The web service dataset described in Section 5.2.1.2 is invoked every hour, allowing for a comparison on the impact that monitoring frequency can have on prediction accuracy. In a dynamic environment, such as IoT, services may need to be monitored more frequently to capture changes in the network. For our experimentation the time series is not pruned for outliers, i.e., values above Q3+1.5*IQR, where Q3 is the third quartile and IQR is the interquartile range. This is to evaluate the robustness of the forecasting approaches to outliers. The dataset is publicly available[1].

[1] https://www.scss.tcd.ie/~whiteg5/data/sensor_data.zip

5.2.1.2 Web Services Description

QoS time series data from web services is included to allow for the comparison of forecasting accuracy between heterogeneous services. An established web services dataset is used [181], which consists of monitored QoS data collected by invoking 10 real services every hour for four months [181]. The dataset contains (i) whether the service was available or not, (ii) the SOAP output message, (iii) the observed response time, (iv) the throughput, as the size of the SOAP message divided by the response time and (v) whether or not the service generated an exception. The experiments focus on the forecasting of the response time, measured in milliseconds. The dataset also provides a comparison of the impact of monitoring frequency as the IoT services are reported every 30 seconds, while the web services are reported every hour.

5.2.1.3 Metrics

Standard error metrics are used to evaluate the forecasting accuracy of the proposed algorithms. Root Mean Square Error (RMSE) is used, which is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i}(y_i - y_i^*)^2} \tag{5.1}$$

where y_i is the QoS value of the service observed by the user at time i and y_i^* denotes the forecast QoS value of the service. $RMSE$ gives large weight to extreme errors due to the squaring term.

In traditional machine learning applications such as image classification, a model is trained, tested and retrained if necessary until satisfactory results have been achieved on the training data set and then the model is evaluated on a holdout test data set. Once the model's performance is satisfactory, it can be deployed to production. When the model has been deployed to production it can classify data as it comes in. Eventually, after some period e.g., a few months, the model may be updated if a significant amount of new training data comes in. In this case, model training is a one time activity, or done at most in periodic intervals to maintain the model's performance or to take into account new information.

For time series forecasting, this is not the case. Instead, the models have to be retrained every time that a new generated forecast is needed. To understand why this is necessary, consider the difference between a traditional machine learning task, such as classifying images and a forecasting task. The visual characteristics of dogs are stable over time (unless looking at evolutionary time scales), so when a neural network is trained to recognize pictures of dogs, the features that define dogs are going to remain the same for the foreseeable future. Given enough data, the model trained this week should be good enough to classify dogs five years from now. The distribution of dog pictures is a stationary distribution, with constant mean and standard deviation over time. For time series models, it is almost always the case that the distribution of the development dataset and the production dataset do not follow the same distribution. This is especially relevant in QoS data, which can change due to devices changing to a power saving mode, additional users entering the environment and congestion on the network. To deal with the changing statistical properties of the data, the models require frequent retraining. From a practical point of view, this means that deploying forecasting algorithms to production is very different to other machine learning models. A static model cannot just be deployed and scored. Instead, it needs to be ensured that training and model selection can be done on the fly, in production.

For some time series evaluations the model is fit using the training data and then evaluated on the test data set. The way that this is done means the comparisons on the test dataset use different horizons. If the model is tested on the last 30% of the data set then the forecast errors will be for 1-step, 2-step, ..., 100-steps ahead. The forecast variance usually increases with the forecast horizon, so if we are simply averaging the absolute or squared errors from the test set, we are combining results with different variances. The solution used in the experiments is to obtain 1-step ahead errors on the test data. That is, we still use the training data to estimate any parameters, but when we compute forecasts on the test data, we use all of the data preceding each observation (both training and test data). Models with large training times such as LSTMs, however, take longer to train these new values and so have to make larger step ahead predictions than lightweight models such as persistence. This is realistic, as it is how these models are used in production.

The production environment in this experimental setup is at the edge of the network, so the training time is evaluated on a Nvidia Jetson Tx2, which is designed to allow for machine learning models at the edge of the network [250]. For each of the models, the training and prediction times are recorded on each of the datasets. The training time is the average time taken to train the model on the training dataset and the prediction time is the average time to generate a forecast given the test data. The model is trained every time a new observation is recorded. The training and testing are repeated twenty times to show the distribution of the time in seconds and the average prediction accuracy is reported using the average RMSE.

5.2.1.4 Statistical Tests

A number of statistical tests are used to provide a method for making quantitative decisions about a sample of test results. They test the hypothesis that is made about the significance of an observed sample. The Diebold-Mariano test is used to compare the forecast accuracy of two forecast methods [251]. The forecasting errors between the actual and the predicted values for each of the models are used. The two forecasts have equal accuracy if and only if the loss differential between the actual and predicted values for both forecasts have zero expectation for all t. The null hypothesis is that the two methods have the same forecast accuracy. The alternative two sided hypothesis is that method 1 and method 2 have different levels of accuracy.

So, the null hypothesis is tested:

$$H0 : E(dt) = 0 \forall t \tag{5.2}$$

versus the alternative hypothesis:

$$H1 : E(dt) \neq 0 \tag{5.3}$$

This allows us to establish the statistical significance of the forecasting accuracy, by either accepting the null hypothesis that the two forecasts have the same accuracy or the alternative hypothesis that the two forecasts have different levels of accuracy. If the forecasts have different levels of accuracy, RMSE is used to evaluate which of the approaches has increased forecasting accuracy.

As described in Chapter 2, ARMA time series forecasts requires the time series to be stationary. If this is not the case, an ARIMA time series needs to be used instead. To this aim, the Augmented Dickey-Fuller test (ADF) is used to check whether the time series contains a trend. The ADF test is a type of statistical test called a unit root test. The intuition behind a unit root test is that it determines how strongly a time series is defined by a trend. The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (i.e., has some time-dependent structure). The alternate hypothesis (rejecting the null hypothesis) is that the time series is stationary.

Null Hypothesis (H0): If failed to be rejected, it suggests the time series has a unit root, meaning it is non-stationary. It has some time dependent structure. Alternate

Statistic	Observations	Mean	Variance	Skewness	Kurtosis	ADF Statistic	p-value
pressureAtSeaLevel	6427	2.806157784	69.77728	5.686691091	41.01123855	-6.125976	0
tempBMP	6988	2.59879016	68.88195224	5.677025464	37.76670894	-6.018004	0
photosensor	6431	2.730159852	65.94653497	5.373051804	34.6657354	-5.770347	0.000001
pressure	5861	2.481476517	64.99827012	5.840726778	39.68851762	-7.003324	0
altitude	5898	2.492071935	61.99539434	5.773681557	39.32910766	-5.929004	0
realAltitude	5843	2.536463565	60.58226421	5.5962328	37.86496146	-5.598761	0.000001
motion	7468	2.258958091	50.50359924	5.627948712	40.84757715	-7.18275	0
tempDHT	6875	2.171278322	49.31313415	5.550758081	39.04585037	-8.909403	0
gas	6824	1.960211294	46.18680955	5.991423795	44.91672038	-7.849511	0
humDHT	6810	1.984306654	43.58372856	5.937605453	45.38663963	-9.487795	0

Table 5.2: IoT Dataset Statistics

Hypothesis (H1): The null hypothesis is rejected; it suggests the time series does not have a unit root, meaning it is stationary. It does not have time-dependent structure. The result is interpreted using the p-value from the test. A p-value below a threshold (such as 5% or 1%) suggests that the null hypothesis (stationary) can be rejected, otherwise a p-value above the threshold suggests the null hypothesis (non-stationary) has failed to be rejected. p-value > 0.05: Fail to reject the null hypothesis (H0), the data has a unit root and is non-stationary. p-value ≤ 0.05: Reject the null hypothesis (H0), the data does not have a unit root and is stationary. The ADF statistic, used in the test, is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at some level of confidence. Table 5.2 and 5.3 show the summary statistics and the results of the ADF test on each dataset. The null hypothesis that the time series are non-stationary can be rejected for each of the datasets, as the p-value ≤ 0.05, therefore the datasets are stationary and can be modelled using ARMA. If the time series were not stationary we could apply a data transformation, such as differencing or detrending were we make an estimate for the deterministic trend component and then remove the trend from the original data [252]. This would be a preprocess before training the algorithms on the data as approaches such as ARMA require the time series to be stationary.

The summary statistics in Table 5.2 and 5.3 show the difference between the two datasets. The IoT dataset in Table 5.2 has a more consistent number of observations with a lower variance and skewness compared to the web service dataset in Table 5.3. This could be due to the frequency with which the services are invoked with the IoT services invoked every 30 seconds and the web services invoked every hour. A large number of changes can happen in the network, at the device side and the user side, which can effect the QoS of the individual services over 1 hour compared to 30 seconds. This can lead to a large variance between the approaches, which is seen in Table 5.3.

Statistic	Observations	Mean	Variance	Skewness	Kurtosis	ADF Statistic	p-value
XMLDailyFacts1	2537	2117.56484	17830188.81	35.9073088	1554.139156	-33.313858	0
GetJoke1	2536	2768.735016	18618655.96	34.53991367	1475.876763	-46.093102	0
HyperlinkExtractor1	3082	5158.83939	176290910	32.60196063	1461.331138	-9.57377	0
FastWeather1	1544	5777.801813	963181837.3	18.6023973	355.22638	-6.986576	0
BLuiquidity1	6174	3348.459346	74765802.12	13.91627971	267.4534655	-7.911812	0
StockQuotes1	2943	3569.214747	69542150.93	10.04588825	141.2629578	-6.519845	0
Google1	2943	4179.506966	56681611.82	9.659332752	141.1674146	-5.79018	0
Amazon1	2943	7358.187563	59714572.65	8.881788334	152.3468532	-4.927402	0.000031
CurrencyConverter1	2535	12266.50256	1487559456	8.662690552	103.6593286	-8.18388	0
QuoteOfTheDay1	2942	4487.914004	29774588.88	5.644518854	34.49207629	-5.314479	0.000005

Table 5.3: Web Dataset Statistics

5.2.1.5 Baseline Approaches

All of the TTD algorithms discussed in the state of the art chapter are implemented and evaluated against the proposed approach. This gives a comprehensive overview of the current state of the art approaches using a number of different datasets. In total, we evaluate 14 approaches on 20 datasets from IoT and web services.

The approaches implemented can be divided into a number of categories. The first is the benchmark models, which are persistence and average methods that use the previous value or the average of a number of previous values to make predictions for the next value in the time series. These are the most simple methods with no training time. The next models are traditional time series models such as ARIMA-based models including AR, MA, ARMA and SARIMA. Exponential smoothing models such as Holt-Winters are also included. These models have been used in traditional time series forecasting tasks and have shown good accuracy performance.

The final category of models are deep learning models such as GRU and LSTM. A number of approaches that apply some variant or additional layers to the LSTM model are also included such as the LSTM with an Encoder Decoder layer, LSTM with an Attention layer and a stacked LSTM approach that uses multiple LSTM layers stacked on top of each other to allow for analysis at different time scales. These are more modern methods that traditionally have not been used for time series forecasting. Finally the noisy-echo state network approach outlined in this thesis is evaluated to compare against the existing state of the art approaches.

5.2.1.6 Threats to Validity

Construct validity threats can be introduced by the stochastic nature of the algorithms under evaluation, which may introduce bias in the used metrics. To mitigate these threats, each of the experiments were repeated 20 times. Statistical tests were used to further validate the results and ensure that there was a statistically significant difference in the accuracy of the approaches. There are also internal validity tests that can arise from the hyperparameter setting used to initialise the noisy-echo state approach as well as the comparison algorithms such as ARIMA, which require p (period to lag), d (number of differencing operations to get time series stationary) and q (number of lags of the error component, where the error component is not explained by the trend or seasonality) parameters to be set. To set the parameters for the baseline models a grid search method is used on a subset of the data to find the best parameters. The best performing parameters were then used in the prediction accuracy experiments. For the noisy-echo state network, the experiments performed for individual hyperparameters were outlined in Section 3.5.2.3. These experiments help to inform the design decision of how to set the individual hyperparameters, in particular setting the leaking rate and the weight generation method in the noisy-echo state network.

Threats to external validity are linked to the selected benchmark models, the QoS dataset and the experimental setup environment. To mitigate these threats, a large number of baseline approaches, which cover the main time series forecasting approaches have been used in the experiments as described in Section 5.2.1.5. The datasets used are a combination of an IoT dataset that we have collected ourselves using IoT devices and sensors in our lab to create a realistic QoS dataset for IoT services, as described in Section 5.2.1.1. An established web service dataset is also used to evaluate how the approaches would forecast a different type of service, which are invoked at a different interval as described in Section 5.2.1.2.

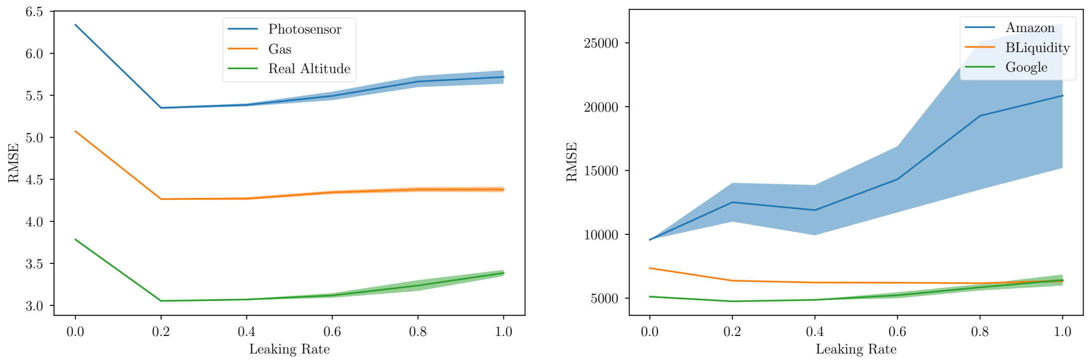
Interpretive validity is the extent to which the conclusions from the experiments are reasonable given the data, which can also be influenced by researcher bias. The full results of the experiments are reported for each of the prediction approaches on each dataset, which is 280 results just for the prediction accuracy. Appropriate statistical tests have also been used to evaluate the statistical significance of the alternative forecasting approaches. This allows the reader to evaluate whether the conclusions drawn from the experiments are reasonable, given these results.

5.2.1.7 Hyperparameters

In Section 3.5, we discussed multiple hyperparameters set for the noisy echo state network model investigates in this paper. The hyperparameters are selected using a subset of the data. The datasets are clustered into 3 groups using the variance, skewness and kurtosis. One dataset from each of the clusters is used to select the hyperparameters, which provides a robust configuration. Here, we illustrate the experiments run to set those values.

Leaking Rate One of the most important parameters when creating the echo state reservoir, especially for data that changes over time, is the leaking rate. The leaking rate can be regarded as the speed of the reservoir update dynamics, discretised in time. The leaking rate α is effectively the resampling of $u(n)$ and $y^{target}(n)$. The leaky rate should be set to match the speed of the dynamic of $u(n)$ and $y^{target}(n)$. Figure 5.2a shows the effect of the leaking rate on three of the IoT datasets. The results follow a similar pattern with a reduction in error as the leaking rate is increased to 0.2. After the initial decrease the error begins to raise again as the leaking rate approaches 1. Setting α to a smaller value such as 0.2 induces slow dynamics of x(n), this can dramatically increase the duration of short-term memory in the ESN.

We also evaluate the leaking rate for a subset of the web services dataset. Figure 5.2b shows the effect of the leaking rate on the web services. The BLuiquidity and Google services follow a similar pattern to the IoT dataset, with an initial decrease in error up until 0.2 when it starts to increase again. The Amazon dataset, in which the noisy echo model did not perform well, shows increased error with a larger leaking rate. The value is minimised when the leaking rate is low and increases as the leaking rate grows, with a slight decrease at 0.4. It still follows the general principle of the other results of having a smaller value of α to increase the duration of short-term memory in ESN, so we set the leaking rate to be 0.2 for the final model.

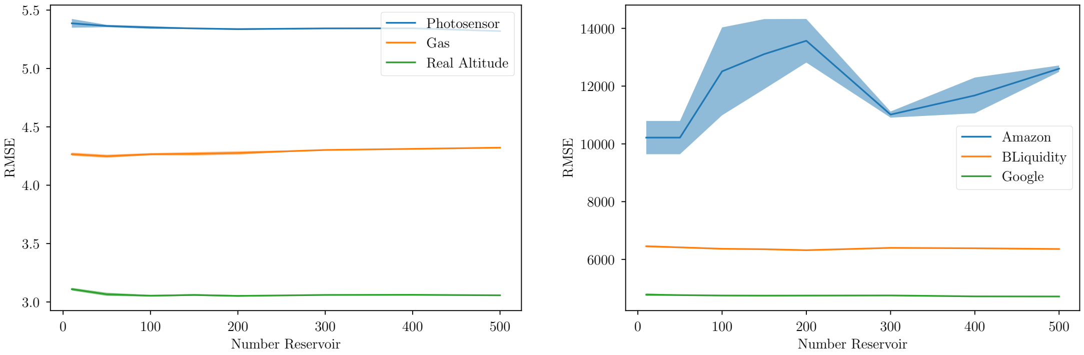


(A) Leaking Rate IoT

(B) Leaking Rate Web

FIGURE 5.2: Impact of Leaking Rate

Reservoir Size The reservoir size is the number of units in the network. Generally, with a larger reservoir, one can learn more complex dynamics, or learn a given dynamics with greater accuracy. However, if the reservoir becomes too large, it can lead to overfitting, with irrelevant statistical fluctuations in the training data learned by the model. This can most clearly be seen in the Amazon dataset, in Figure 5.3b, as the reservoir size becomes greater than 50. The other datasets in this figure show a less dramatic reduction in error up to a reservoir size of 100, at which point the error begins to increase or stay the same. The results for the IoT dataset in Figure 5.3a show a less dramatic change. There is a slight reduction in error up to a reservoir size of 100 after which, the error begins to increase gradually. Using these results we choose a reservoir size of 100 to allow our model to learn complex dynamics without overfitting to include irrelevant fluctuations in the training data as QoS data can be noisy.



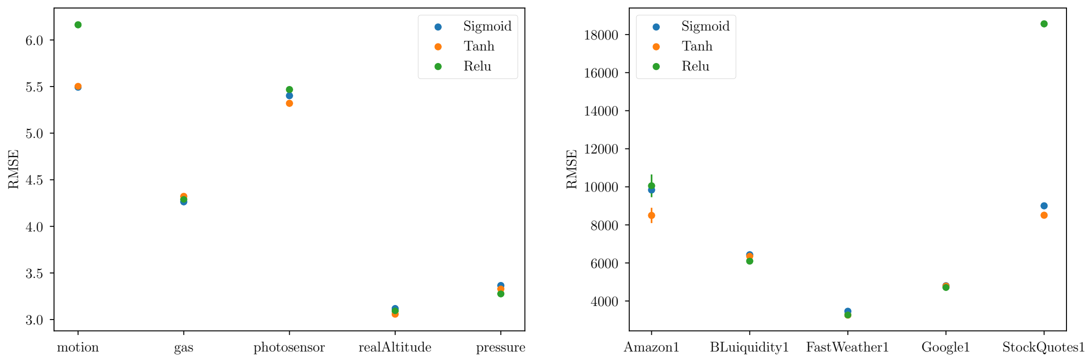
(A) Number Reservoir IoT

(B) Number Reservoir Web

FIGURE 5.3: Impact of Reservoir Size

Activation Function A number of activation functions are available for echo state networks. Previous approaches have traditionally used Tanh activation. We performed some experimentation on the effect of alternative activation functions such as Sigmoid and Relu to evaluate their impact on the final accuracy of the model. Figure 5.4a shows the impact of the activation function on a subset of the IoT dataset. We show the average and standard deviation of 10 runs of the experiment on each dataset. The results for the IoT dataset show that the Sigmoid and Tanh perform quite similarly, with Tanh performing better in more of the datasets overall. Relu shows good performance for the pressure dataset but can increase the error in other datasets that are more skewed.

Figure 5.4b shows the results for the web services data. It follows a similar pattern to the IoT data with Tanh showing improvement compared to the other two approaches. Relu can show improved results in datasets such as BLuiquidity1 but is also prone to outliers as in the StockQuotes1 dataset, which has a larger variance. Based on these results, we choose Tanh to be the activation function that we use in our final model.



(A) Activation Function IoT

(B) Activation Function Web

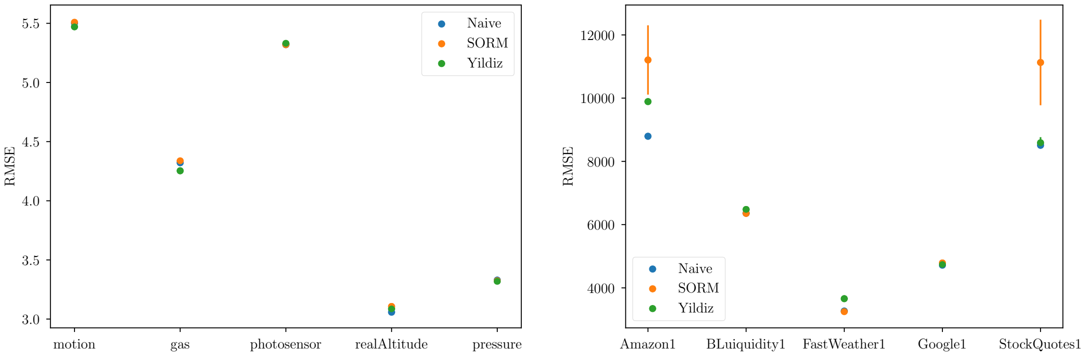
FIGURE 5.4: Impact of Activation Functions

Weight Generation When the reservoir is created, the weights of the neurons are set to an initial weight. There have been a number of alternative approaches to initialise these weights, from a basic naive approach where the weights are assigned a randomised value between -0.5 and 0.5. The SORM technique has been designed to reduce the amount of randomness [200]. This method creates a weight matrix with predefined singular spectrum that can guarantee stability (echo state property) and minimise the impact of noise on the training process. We also evaluate a sparse random sign approach [201]. We plot the mean and standard deviation of the results in Figure 5.5a and 5.5b.

Figure 5.5a shows the impact of the weight generation method on the IoT dataset. We can see that for the IoT dataset the weight generation method had a small impact on the final results with the Yildiz method showing improvement in the gas dataset and the naive method showing improvement in the realAltitude dataset. Figure 5.5b shows the impact of the weight generation method on the web services dataset. In this case, there is more variation in the final results especially with the SORM method that has a lot of variation and returns poor results for the Amazon1 and StockQuotes1 dataset. From the two other approaches, the naive method returns the most reliable results, showing increased performance compared to Yildiz in the Amazon1 and FastWeather1 dataset. Based on the results from these experiments, we use the naive method in our final experiments as it provided the most consistently good performance.

5.2.2 Results

This section presents the results of the TTD evaluation. Section 5.2.2.1 evaluates the prediction accuracy of the approaches, Section 5.2.2.2 evaluates the training time of the approaches and Section 5.2.2.3 evaluates the prediction time of the approaches.



(A) Weight Generation IoT (B) Weight Generation Web

Figure 5.5: Impact of Weight Generation

5.2.2.1 Prediction Accuracy

Table 5.4 shows the RMSE error between the actual and predicted values of the IoT dataset. The 14 algorithms tested on 10 different IoT datasets provide a comprehensive overview of the current state of the art. The results are also evaluated using the Diebold-Mariano (DM) Test. The null hypothesis is that each time series has equal predictive accuracy and the alternative hypothesis is that the models have different levels of accuracy. The significance level for statistical significance is 0.05 and the significance level for high statistical significance is 0.01. In Table 5.4 and Table 5.5, $P < 0.05 = *$ and $P < 0.01 = **$ compared to the proposed noisy-echo state network approach. The most accurate forecasting model for each dataset is in bold font. The results in Table 5.4 show that the noisy-echo model produces the most accurate QoS forecasts for 90% of the IoT data and the forecasts have statistically significant different levels of accuracy. The Holt Winters algorithm produces better results for the TempDHT dataset. Further analysis of this dataset shows that it has a smallest combined variance, skewness and kurtosis. This makes the dataset highly suitable for an exponential smoothing method as new values will not vary much from the previous.

The prediction accuracy can be affected by increased training times as the model may not have incorporated the most recent values before making a prediction. The increased training time of the LSTM approaches shown in Table 5.6 causes the model to have to make predictions for new values based on the old model, while the old model is being updated with new data points. This can cause an increase in errors if there is sudden congestion or changes in the network. Forecasting approaches with short training times such as Holt-Winters and the echo-state based approaches can quickly incorporate these values into the model before the next time series data is reported

Method	motion	gas	tempDHT	humDHT	tempBMP	photosensor	pressureAtSeaLevel	realAltitude	pressure	altitude
Persistence	7.810238**	5.890278**	7.411874**	6.103446**	8.108597**	7.488159**	7.019569**	4.30112**	4.683283**	3.149722**
Average	6.354379**	4.826091**	6.288365**	4.867702**	6.745602**	5.839891**	5.563181**	3.290822**	3.717615**	3.244886**
AR	5.924289**	4.486149**	5.887825**	4.774765**	6.316112**	5.836597**	5.394121**	3.273735**	3.44607**	3.18818**
ARMA	5.823097**	4.561525**	5.6930385**	4.884718**	6.383319**	5.992053**	5.612935**	3.733769**	3.827525**	3.695597**
Holt Winters	5.479644**	4.315179**	5.498252**	4.549918**	5.858203**	5.367755**	4.952462**	3.066059**	3.326135**	3.090484**
ARIMA	5.957925**	4.69692**	5.988508**	4.962601**	6.468264**	6.152145**	5.764984**	4.094006**	4.233293**	4.076958**
SARIMA	5.633425**	4.45195**	5.736247**	4.724357**	6.064921**	5.595941**	5.209394**	3.156171**	3.417940**	3.183883**
GRU	7.577633**	5.370888**	7.411886**	6.103448**	7.512873**	7.488179**	9.939786**	4.301400**	4.683184**	3.157885**
LSTM	7.207294**	5.565407**	7.322619**	5.487462**	8.013495**	6.759292**	8.413381**	4.125590**	4.050942**	3.125581**
LSTM Encoder Decoder	7.457435**	5.459793**	7.393157**	5.894806**	7.537269**	7.086604**	8.464745**	4.301197**	4.683329**	3.107436**
LSTM Attention	7.689534**	5.706162**	7.411874**	6.103446**	12.398120**	7.488155**	7.685253**	4.290354**	4.683286**	3.274812**
LSTM Stacked	7.577633**	5.370888**	7.411886**	6.103448**	7.512873**	7.488179**	9.939786**	4.301400**	4.683184**	3.157885**
ECHO	5.469087**	4.265062**	5.539449*	4.469455**	5.846969**	5.344445**	4.949313*	3.055276*	3.317153**	3.087040**
Noisy ECHO	**5.451934**	**4.25136**1	5.5386996	**4.459078**	**5.844422**	**5.319939**	**4.949122**	**3.052062**	**3.16274**	**3.079277**

Table 5.4: IoT Forecasting RMSE

Method	Amazon1	BLuiquidity1	CurrencyConverter1	FastWeather1	GetJoke1	Google1	HyperlinkExtractor1	QuoteOfTheDay1	StockQuotes1	XMLDailyFacts1
Persistence	10975.65**	7132.73**	47234.75**	3755.30**	3193.56**	5226.96**	4677.66**	3188.34**	10512.41**	10207.65**
Average	8840.14**	7552.72**	39614.98**	3072.11**	3369.60**	5205.82**	4033.95**	3170.86**	9784.41**	8552.70**
AR	8714.60**	6839.55**	34644.79**	3474.57**	3119.32**	5259.74**	4067.34**	3719.43**	9178.95**	7507.64**
ARMA	8104.12**	6862.14**	34659.43**	4046.78**	None	5201.29**	3876.76**	3324.05**	8992.52**	7483.07**
Holt Winters	8620.56**	6965.58**	34988.72**	3755.30**	3122.08**	4948.82**	3884.82**	2992.80**	9745.93**	7555.26**
ARIMA	8103.98**	6862.14**	34659.43**	4046.78**	3121.97**	5201.29**	3876.76**	3325.52**	8992.52**	7483.26**
SARIMA	8891.03**	6948.66**	34051.60**	5670.82**	None	5287.16**	3850.15**	3833.11**	9153.89**	7498.65**
GRU	10983.43**	7137.13**	46273.83**	3925.01**	3472.33**	5237.84**	4944.61**	3216.84**	10512.79**	10217.17**
LSTM	10969.00**	7143.52**	47134.05**	3906.66**	3156.37**	5229.79**	4837.26**	3215.65**	10512.54**	10009.93**
LSTM Encoder Decoder	10982.53**	7118.12**	41431.63**	4049.74**	3194.55**	5233.42**	4855.98**	3216.36**	10493.08**	10141.25**
LSTM Attention	10983.78**	7136.57**	57918.30**	3807.50**	3227.77**	5231.94**	4770.09**	3188.35**	10512.54**	10233.90**
LSTM Stacked	10132.56**	7132.75**	39211.29**	3755.35**	5447.82**	5470.33**	4677.74**	3027.74**	10512.41**	10009.94**
ECHO	9574.92*	6346.36**	33721.18**	3187.58**	3038.93**	4741.28**	3848.73*	2800.81**	8667.31**	9297.17*
Noisy ECHO	9576.48	**6340.22**	**32631.25**	**3163.25**	**3023.48**	**4720.73**	**3847.45**	**2791.08**	**8504.15**	9291.16

Table 5.5: Web Forecasting RMSE

as their training time is less than 30 seconds. This leads to these methods having increased prediction accuracy.

Table 5.5 shows the RMSE error between the actual and predicted values for the web services dataset. Again, the most accurate forecasting method for each dataset is bolded and the statistical significance is shown using stars. The results show the improved forecasting accuracy using the noisy-echo forecasting approach for 80% of the test datasets. The two datasets that get better predictions from traditional time series-based approaches ARMA and ARIMA are Amazon1 and XMLFacts1. Further analysis of these datasets show that they have a small combined variance and skewness. This makes them suitable for traditional autoregressive approaches such as ARMA and ARIMA.

One of the problems with the autoregressive approaches (AR, ARMA, ARIMA, SARIMA) in this experiment was that they sometime fail to converge as in the case of GetJoke1. GetJoke1 has a large skewness and kurtosis values that causes the ARMA and ARIMA model not to converge. One of the limitations of these approaches is that, if the dataset is skewed then the model may not converge, meaning that no forecasts can be produced. The large training time of the LSTM-based approaches shown in Table 5.7 leads to an increase in prediction errors compared to other models that can be retrained more frequently at the edge.

Method	motion	gas	tempDHT	humDHT	tempBMP	photosensor	seaLevel	realAltitude	pressure	altitude
Persistence	-	-	-	-	-	-	-	-	-	-
Average	-	-	-	-	-	-	-	-	-	-
AR	0.19661	0.173055	0.17612	0.179871	0.180327	0.15533	0.156224	0.13247	0.131492	0.131594
ARMA	4.738197	3.997468	3.930942	4.146136	3.906832	3.626856	3.531937	3.07438	3.187256	3.156328
Holt Winters	4.831115	4.541888	4.28464	4.479004	4.884842	4.786444	3.951013	3.64523	4.90282	4.438811
ARIMA	4.195067	3.869153	3.833079	3.694446	3.727166	3.406544	3.215134	2.840352	2.928405	2.85305
SARIMA	149.0606	143.3560	134.8984	134.3577	154.9273	159.4404	169.4713	131.4500	136.7973	115.5670
GRU	8957.631	8355.466	8024.337	8024.337	8896.441	8297.486	8357.352	7473.455	7645.846	7199.695
LSTM	311835.7	288716.2	317709.1	276973.1	303094.9	283282.0	290341.6	260300.4	266914.9	254378.9
LSTM Encoder Decoder	376839.2	349387.7	339228.0	337518.8	367999.6	342267.5	351155.1	315377.7	323044.1	305964.1
LSTM Attention	371951.3	339359.2	371538.0	328683.8	367999.6	338429.2	350553.3	309731.4	319886.0	304479.4
LSTM Stacked	773839.2	716150.9	769460.6	695167.2	747827.1	693634.5	707128.2	636118.9	647677.8	625030.7
ECHO	12.9389	11.9702	12.4487	12.7227	12.7227	11.3671	11.3033	13.1534	10.6236	13.2142
Noisy ECHO	12.8825	11.9980	12.5956	12.6658	12.7275	11.2330	11.4149	13.1377	10.7337	13.3578

Table 5.6: IoT Training Time (seconds)

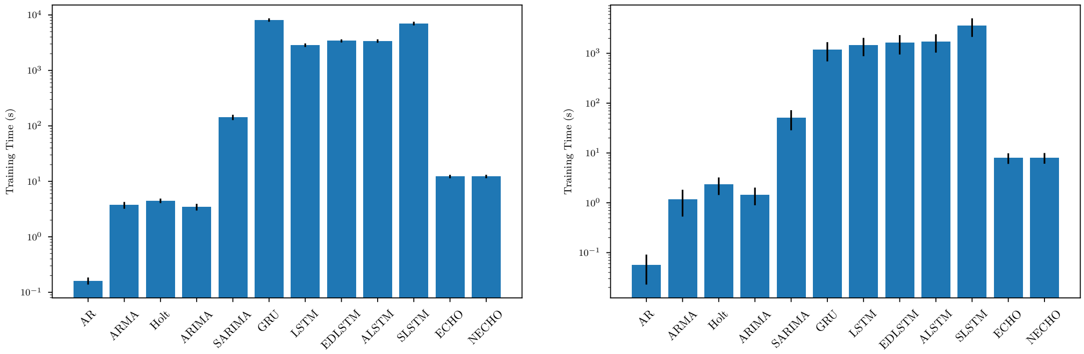
5.2.2.2 Training Time

Table 5.6 shows the training times for each of the models used in the IoT service experiments. There is a large variation in training time from models that have no training time to models that require hours to train. The average and persistence models require no training as they make forecasts using either the last value or the average of the three previous values. Therefore, only the storage of these values and some analysis during testing is required to make the predictions. The next range of training times are those that can update their models in less than thirty seconds to accommodate recent QoS values from dynamic services. This quick update time ensures that the model can respond to sudden changes in the service such as network overload or a device switching to a power saving mode. These are the traditional autoregressive, exponential smoothing forecasts such as Holt Winters and ECHO-based approaches. One surprise with the experimental results is the time required for the SARIMA model, compared to the other traditional autoregressive models such as ARMA and ARIMA, but this can be explained by the additional seasonal modelling used to try to increase the prediction accuracy. As it can take over a minute to train, this can lead to missing some data points when making forecasts. The final range of models are those that require large training times greater than a couple of hours. These are the traditional RNN-based approaches such as GRU and LSTM, which makes them unsuitable for short term forecasts at the edge as they take a long time to incorporate recent changes in the network or device failures.

Table 5.7 shows the training times for each of the models used in the web service experiments. The training times follow a similar pattern to the IoT dataset with the same models grouped into a range of no training times, short training times of a less than thirty seconds, larger training times of a couple of minutes and the longest taking over a couple of hours. The training times are slightly shorter than the IoT dataset

Method	Amazon1	BLuiquidity1	CurrencyConv1	FastWeather1	GetJoke1	Google1	LinkExtractor1	Quote1	StockQuotes1	XMLFacts1
Persistence	-	-	-	-	-	-	-	-	-	-
Average	-	-	-	-	-	-	-	-	-	-
AR	0.052694	0.150727	0.043863	0.022758	0.04403	0.053106	0.055179	0.052349	0.052991	0.043407
ARMA	0.882086	2.873928	0.85237	0.52214	0.800686	1.058162	1.248833	0.994781	1.079717	1.482804
Holt Winters	2.21504	3.466027	2.509222	1.120705	3.988224	1.853427	2.954879	1.765269	1.72112	1.711918
ARIMA	0.989633	2.851033	1.437518	0.913856	1.414293	1.650149	1.564868	1.010735	1.628183	1.104375
SARIMA	50.29268	53.13638	19.02887	25.06535	75.93919	73.95002	25.60217	58.61540	77.70087	44.50840
GRU	1168.388	2514.126	949.316	649.932	991.655	1046.114	1187.169	1184.678	1101.000	1031.767
LSTM	1435.565	3022.565	1212.179	796.496	1221.661	1373.269	1467.770	1425.833	1385.752	1248.279
LSTM Encoder Decoder	1611.996	3461.631	1331.263	822.663	1381.123	1483.571	1645.584	1640.092	1517.033	1434.624
LSTM Attention	1701.353	3570.491	1412.371	918.521	1459.617	1627.364	1727.061	1687.721	1653.125	1484.987
LSTM Stacked	3520.223	7491.751	2950.277	1963.969	2998.524	3355.695	3588.372	3532.353	3384.077	3082.312
ECHO	7.41785	13.14107	7.64465	5.93231	7.87751	7.45632	7.93177	7.48932	7.47702	7.09368
Noisy ECHO	7.625165	13.305066	7.66095	5.883748	8.056742	7.449957	8.033465	7.644891	7.612619	7.10852

TABLE 5.7: Web Training Time (seconds)



(A) IoT Dataset Training Time

(B) IoT Dataset Training Time

FIGURE 5.6: Average Training Time

because it is a slightly smaller dataset, but with the RNN-based approaches still taking a couple of hours.

Figure 5.6 presents an overview figure of the training time for all the datasets. It shows the mean and standard deviation of the training times for all the IoT datasets in Figure 5.6a and Web datasets in Figure 5.6b. The figure is plotted using a logarithmic y-axis to allow for the methods to be more easily compared against each other. The persistence and average methods are not included in this figure as they do not require any training time.

In Figure 5.6a, one of the first observations that can be made is that the standard deviation in the training time is small. As the IoT datasets are of a similar size, there is not much variation between the datasets, which allows for comparison between the individual approaches. The range of training times can be seen clearly with each step on the y axis indicating a 10 times greater training time. The next level of training times are for the traditional time series forecasting techniques, such as AR, ARIMA, Holt and the echo state-based approaches. The SARIMA approach then has an increase in training time, but the RNN-based approaches such as GRU and LSTM have a very large training time orders of magnitude greater than the traditional time series-based

Method	motion	gas	tempDHT	humDHT	tempBMP	photosensor	seaLevel	realAltitude	pressure	altitude
Persistence	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
Average	0.000044	0.000044	0.000044	0.000044	0.000045	0.000044	0.000044	0.000044	0.000044	0.000044
AR	0.000791	0.000731	0.00074	0.000733	0.000742	0.00067	0.000663	0.000595	0.000598	0.000596
ARMA	0.006415	0.005869	0.003015	0.00596	0.005989	0.005604	0.00559	0.005158	0.005181	0.005207
Holt Winters	0.967671	0.864094	0.928017	0.853228	0.94472	0.86443	0.865268	0.782894	0.788073	0.777138
ARIMA	0.006702	0.006234	0.006262	0.006242	0.006363	0.005933	0.00592	0.005519	0.00551	0.005559
SARIMA	0.559103	0.508898	0.513655	0.536163	0.523918	0.480207	0.480434	0.44074	0.437828	0.436803
GRU	0.005012	0.005157	0.004865	0.004865	0.005417	0.005817	0.006038	0.005789	0.00612	0.005286
LSTM	0.007849	0.008263	0.009753	0.00726	0.008833	0.00922	0.009403	0.009478	0.009769	0.008616
LSTM Encoder Decoder	0.011234	0.011509	0.010367	0.010517	0.012382	0.012655	0.013247	0.013353	0.013663	0.012055
LSTM Attention	0.005699	0.00625	0.007968	0.005355	0.007187	0.007261	0.007619	0.007686	0.007648	0.006757
LSTM Stacked	0.018237	0.019023	0.022927	0.017121	0.020248	0.020735	0.021877	0.021502	0.022599	0.019761
ECHO	0.001041	0.001051	0.001034	0.001021	0.00101	0.001019	0.00105	0.001033	0.001036	0.001001
Noisy ECHO	0.000999	0.000996	0.000991	0.001	0.000989	0.000984	0.000987	0.000993	0.001002	0.000987

Table 5.8: IoT Prediction Time (seconds)

approaches. This means it takes longer for them to incorporate new QoS values into the model.

Figure 5.6b shows the training time for the QoS forecasting approaches on the web services dataset, the standard deviation is slightly increased due to the variety in sizes of the web services dataset. The same level of difference in the training times can be seen in 5.6b as in Figure 5.6a, with the approaches falling into the same groups of traditional time series and echo state-based approaches and the RNN-based approaches taking longer.

5.2.2.3 Prediction Time

The prediction time is the time taken for a model to make a forecast, given test data. Compared to the training time, the prediction time to generate the forecasts is quite small, usually under a second. Table 5.8 and 5.9 show the prediction times for both the IoT and web services dataset. For most of the approaches, the time to generate forecasts is very small < 0.005s. However, SARIMA, with values around 0.5s and Holt Winters, with values around 0.8s, are orders of magnitude larger. The baseline approaches with no training time - persistence and average, also have very low prediction times as they both require only a simple lookup of previous data and some basic analysis. For making short-term predictions, the prediction time is important as if it takes too long, it can increase the tolerable delay to greater than what is required by the application. Modern applications such as augmented reality have a low tolerable delay of 10ms or 0.01s. These experiments were designed to identify any trade-offs in models that may leave additional processing until after the model has been trained. The SARIMA and Holt Winters methods have much larger testing times compared to the existing approaches that could cause some problems for generating predictions for these applications.

Method	Amazon1	BLuiquidity1	CurrencyConv1	FastWeather1	GetJoke1	Google1	LinkExtractor1	Quote1	StockQuotes1	XMLFacts1
Persistence	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001
Average	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005
AR	0.000351	0.000641	0.000323	0.000256	0.000323	0.000352	0.000362	0.00035	0.000351	0.000321
ARMA	0.001384	0.005402	0.001332	0.001186	0.001332	0.00141	0.003185	0.001381	0.003074	0.002787
Holt Winters	0.39066	0.825778	0.343205	0.201683	0.34329	0.397382	0.416639	0.396691	0.396949	0.346397
ARIMA	0.001701	0.002381	0.003102	0.002407	0.003129	0.003405	0.003515	0.001688	0.003404	0.003123
SARIMA	0.222839	0.461416	0.192956	0.120416	0.193494	0.223178	0.234576	0.221257	0.223097	0.192215
GRU	0.006131	0.006629	0.00551	0.006559	0.006034	0.004448	0.005752	0.006544	0.005051	0.006618
LSTM	0.009456	0.008883	0.009244	0.009059	0.009305	0.009868	0.00941	0.008715	0.009427	0.008954
LSTM Encoder Decoder	0.0131	0.013872	0.012224	0.00607	0.012881	0.010941	0.012516	0.013331	0.011361	0.01404
LSTM Attention	0.008775	0.009184	0.008407	0.009185	0.008684	0.007967	0.008768	0.008975	0.00841	0.008972
LSTM Stacked	0.024779	0.02517	0.024003	0.02541	0.024269	0.023171	0.023736	0.024771	0.023565	0.025053
ECHO	0.001036	0.000999	0.001039	0.001004	0.001026	0.001027	0.001041	0.001037	0.001028	0.000993
Noisy ECHO	0.000997	0.000996	0.000992	0.000984	0.001002	0.000989	0.001004	0.000997	0.00099	0.00101

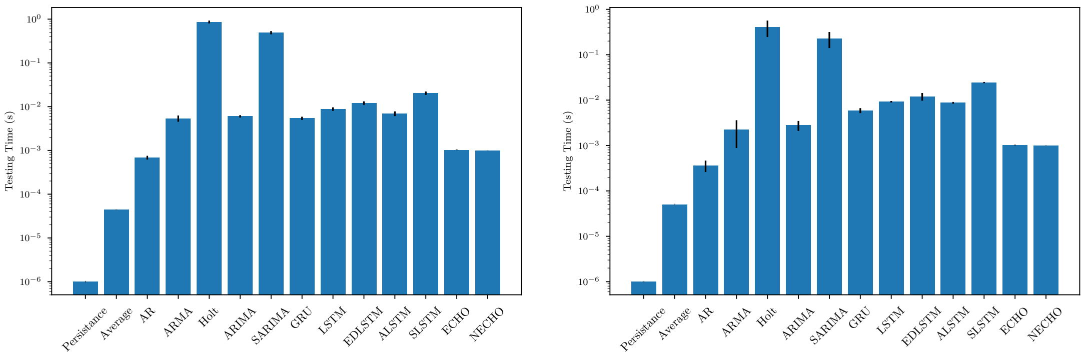
Table 5.9: Web Prediction Time (seconds)

Figure 5.7 presents an overview figure of the testing time for all the datasets. It shows the mean and standard deviation of the testing times for all the IoT datasets in Figure 5.7a and Web datasets in Figure 5.7b. The figure is plotted using a logarithmic y-axis to allow for the methods to be more easily compared against each other. The persistence and average methods are included in this figure, although as can be seen in Figure 5.7 they both have small testing times.

From Figure 5.7, the majority of the testing times for the traditional time series approaches and even the more modern RNN-based approaches are quite similar and small enough to be used in a QoS forecasting algorithm at the edge of the network. There are two exceptions that can be seen in Figure 5.7a, which are the Holt-Winters and the SARIMA approaches. These approaches are orders of magnitude greater than the others but the testing time is still under one second. This may cause problems for more high frequency applications but would be acceptable for most IoT applications with soft QoS forecasting guarantees. The Holt Winters approach had the best forecasting accuracy on the tempDHT dataset in Table 5.4 and a small training time but there is a tradoff in the prediction generation time.

Figure 5.7b shows the testing time for the web service approaches. The results follow the same pattern as Figure 5.7a with the Holt-Winters and SARIMA approaches being the two outliers with large testing times. The other traditional time series and echo state-based approaches are suitable to be used for applications such as augmented reality as they can be deployed at the edge of the network and have a low testing time. However, of the approaches with reduced training and testing time the approach that was able to generate the most accurate forecasts was the noisy-echo state network model.

RQ.1 proposed in Chapter 1 asked the question, to what extent can the accuracy of forecasting to support TTD be improved, by using a lightweight model at the edge to incorporate recent changes in QoS? The results of the experimental evaluation have shown that the noisy-echo state network has increased the forecasting accuracy for



(A) IoT Dataset Testing Time

(B) IoT Dataset Testing Time

FIGURE 5.7: Average Testing Time

the 9/10 (90%) of the IoT datasets and 8/10 (80%) of the web service datasets for an overall improvement in 17/20 (85%) of the combined datasets. The training time of the model has been evaluated on an edge device and is small enough to allow for the incorporation of recent changes in QoS. The testing time of the noisy-echo state network is also evaluated to ensure some of the training procedure is not left until predictions are generated and it has been shown to be small enough to be used in a dynamic environment. Therefore, the answer to RQ.1 is that a lightweight noisy-echo state network deployed at the edge to incorporate recent changes in QoS can be used to improve the accuracy of forecasting to support TTD in 85% of cases, on a combination of IoT and web service datasets. The 15% of cases where the noisy-echo state network approach was found to not be as effective were in the datasets with low variance, skewness and kurtosis. In these datasets traditional time series approaches such as Holt-Winters and ARMA were found to be more accurate.

5.3 TTR

This section presents the experimental setup and results of the TTR approaches.

5.3.1 Experimental Setup

5.3.1.1 Dataset

To test both the IoTPredict and the Stacked Autoencoder approach, an established dataset is used to control for any differences in QoS that can be caused by invoking services at different times. The dataset released by Zheng et al. [42] is used, which

Statistics	Response Time	Throughput
Number of Observations	1,974,675	1,974,675
Number of Invocations	1,873,838	1,831,253
Number of Users	339	339
Number of Services	5,825	5,825
Min	0.002s	0.004kbps
Max	27.285s	1000kbps
Mean	2.9s	47.56kbps
Variance	5.886s	12276kbps
Skewness	2.61s	4.96kbps
Kurtosis	11.03s	28.58kbps

TABLE 5.10: Descriptive Statistics of TTR Dataset

consists of a matrix of the response time and throughput of 339 users from 30 countries for 5,825 real-world web services from 73 counties. The users are a number of distributed computers from PlanetLab and are not co-located with the services. Response time is the time duration between a user sending a request and receiving a response, while throughput denotes the data transmission rate (e.g. kbps) of a user invoking a service. The reason for the use of a dataset instead of a testbed is to allow more users and services to be evaluated, as there are no reported testbeds that have access to 339 users and 5825 services. The time varying nature of QoS can also lead to the algorithms being evaluated using different values in testbeds making it harder to evaluate the impact of the algorithm.

As this dataset is for web services, which are usually deployed in the cloud, they have better response time than might be expected from low power devices. To make the data applicable for the experimentation, the HetHetNets traffic model is used to add heterogeneous traffic data to the existing dataset [211]. This provides a realistic and manageable traffic model that can be applied in many contexts such as Wi-Fi, ad-hoc and sensor networks. The parameter is set to $\lambda = 2$, for modelling network traffic in sensor networks and the IoT [211]. Table 5.10 shows the descriptive statistics of the dataset, which shows the difference in scale of the datasets with the response time ranging from 0.002s to 27.285s and the throughput ranging from 0.004kbps to 1000kbps. The two datasets are skewed, which can be seen from the skewness and kurtosis results.

As there is full access to the dataset, a percentage of the values can be removed to be used as the training set. The algorithms are then evaluated by making predictions for the missing values in the dataset. This allows us to evaluate how the algorithms perform for different matrix densities from 5-35% in our experiments using standard error metrics, which are introduced in the following section.

5.3.1.2 Metrics

A combination of metrics are used to evaluate the prediction accuracy of the proposed algorithms, in particular, standard error metrics such as the Mean Relative Error (MRE) and Root Mean Square Error (RMSE). MRE is defined as:

$$MRE = \frac{1}{N}\sum_{i,j}\frac{|w_{i,j}(t) - w^*_{i,j}(t)|}{w_{ij}(t)} \tag{5.4}$$

RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j}(w_{i,j} - w^*_{i,j})^2} \tag{5.5}$$

where $w_{i,j}$ is the QoS value of service c_j observed by user, u_i, $w^*_{i,j}$ denotes the predicted QoS value of the service c_j by user u_i. N is the number of predicted QoS values, which normalises the prediction accuracy across the matrix densities. $RMSE$ gives large weight to extreme errors due to the squaring term.

One of the problems with standard error metrics is that it can be difficult to know how the difference in accuracy of the predictions will perform as part of a realistic IoT application [56, 253]. For example, if there are 100 candidate services and the algorithm makes an accurate prediction for the top 90 but makes bad predictions for the last 10 the overall prediction accuracy will be influenced by the last 10 services even though is it unlikely that they would be used in the composition. The ranking of the top services is more important as they are the most likely to be invoked. To evaluate this, the predictions are used as part of a service composition process to compose an application. A number of composition paths are generated with services available in the environment identified to satisfy a user request by the request handler. The service composition engine creates a list of service flows based on the concrete services received from the service discovery engine. The flows are then merged based on the service description. If two or more services in the flow have the same input, the composition engine creates a guidepost to enable the invocation of the services based on QoS.

In this thesis, the QoS metrics of response time and throughput are considered for each branch. The response time is calculated by the mean response time values of each service in the branch and the throughput is the minimum throughput of the set of selected candidate services [210], as shown in Equation 5.6. In this formula, rt_i is the response time of service *i*. The throughput value is calculated using the formula in Equation 5.7, which selects the lowest throughput $min(th_i)$ offered by the services in a sequential flow [210]. These formulae require the response time and throughput values of each service component in the flow to calculate the aggregate values. It is possible that these values can not be calculated if the required QoS data is missing or is out-of-date. The predictive composition process uses the predicted values generated through collaborative filtering to choose the optimal flow. 10 composition branches with 10 services in each are created. Once the branch has been chosen using the predicted values the actual values based on the original data are reported. This allows for comparison between the two prediction approaches and the optimal composition that could have been chosen.

$$Response\ Time\ (RT) = \sum_{i=1}^{n} rt_i \tag{5.6}$$

$$Throughput\ (T) = min(th_i) \tag{5.7}$$

To evaluate the suitability of deploying the algorithm at the edge of the network, the training time on an edge device is evaluated, in this case a Jetson Tx2, which has increased performance compared to traditional IoT devices, such as raspberry pis fulfilling Objective 3. This time constrains how often the algorithm can be updated with new values, which has an impact on the accuracy when used on time varying values such as QoS. The time taken to train the algorithms is measured at different matrix densities. The experiment is repeated 20 times showing the average and standard deviation of the predictions in the results section.

The request time of the algorithms is also evaluated showing how quickly a distributed middleware could receive QoS predictions for users in the environment. In this measurement, each algorithm is deployed in the environment needed to train it and the network delay is included in the response. For the request time, 5000 requests are made to the services running on each of the devices and the distribution of data is shown using a box plot. This gives a realistic request time and shows the impact of being able to deploy an algorithm on an edge node vs. deploying the algorithm in the cloud. The network test is conducted in Trinity College Dublin and each of the devices are connected using a Wi-Fi-based network.

5.3.1.3 Statistical Tests

To compare the statistical significance of the difference in results such as the training time, an independent samples t-test is used [254]. This test compares the means of two independent samples on a given variable. To conduct an independent samples t-test, one categorical or nominal independent variable and one continuous or interval scaled dependent variable is needed. A dependent variable is a variable on which the scores may differ, or depend on the value of the independent variable. An independent variable is the variable that may cause, or simply be used to predict, the value of the dependent variable. The independent variable in a t-test is simply a variable with two categories (e.g., men and women, university students and university professors, etc.). In this type of t-test, the goal is to know whether the average scores on the dependent variable differ according to which group one belongs (i.e., the level of the independent variable). For example, a researcher may want to know if the average height of people (height is the dependent, continuous variable) depends on whether the person is a man or a woman (gender of the person is the independent, categorical variable).

Regarding the specific case of independent samples t-test, the question to be answered is whether the difference between the two sample means is large compared to the difference that would be expected by just selecting two different samples. Phrased another way, is to know whether the observed difference between the two sample means is large relative to the standard error of the difference between the means. This can be used to evaluate if the training times of different algorithms are statistically significant. The general formula for this question is as follows:

$$t = \frac{\text{observed difference between sample means}}{\text{standard error of the difference between the means}} \tag{5.8}$$

or

$$t = \frac{\bar{X}_1 - \bar{X}_2}{s_{\bar{x}_1 - \bar{x}_2}} \tag{5.9}$$

where:

$\bar{X}_1$ is the mean for sample 1
$\bar{X}_2$ is the mean for sample 2
$s_{\bar{x}_1 - \bar{x}_2}$ is the standard error of the difference between the means

Once the test statistic has been calculated it can be used in the formula for calculating the t value. This can be used with the degrees of freedom by adding the two sample sizes together and subtracting 2. So the formula is $df = n_1 + n_2 - 2$. The test statistic can then be used to see if the results are significant. The test has a number of assumptions about the data such as independent observations and the assumption of normality, which are verified before the test is used. The assumption of normality is tested using the Shapiro–Wilk test.

Other than testing the presence of a significant difference among the different methods, it is of practical interest to estimate the magnitude of such a difference. Cohen's *d* effect size, is used to indicate the magnitude of a main factor treatment on the dependent variables [255] (the effect size is considered small for $0.2 \leq d < 0.5$, medium for $0.5 \leq d < 0.8$ and large for $d \geq 0.8$). For independent samples, it is defined as the difference between the means, divided by the pooled standard deviation of both groups:

$$Cohen's\ d = (M_2 - M_1)/SD_{pooled} \tag{5.10}$$

where:

$$SD_{pooled} = \sqrt{(SD_1^2 + SD_2^2)/2} \tag{5.11}$$

5.3.1.4 Baseline Approaches

The experiments for TTR consider different kinds of approaches. In particular, matrix factorisation-based approaches: NMF, PMF, IoTPredict and a deep learning approach called stacked autoencoder. NMF is a standard non-negative matrix factorisation approach that decomposes the original dataset into non-negative latent features that are used to find similar users and services. PMF provides a probabilistic approach using Gaussian assumptions on the known data and the factor matrices. IoTPredict uses latent features with an alternative nonparametric similarity comparison method called Kendall's tau to incorporate the similarities. This method uses a hybrid model-based approach calculating the similarity for both the users and services. Finally, the stacked autoencoder approach is proposed to evaluate how deep learning methods can be used for a collaborative filtering task.

5.3.1.5 Threats to Validity

Construct validity threats can be introduced by the stochastic nature of the algorithms under evaluation, which may introduce bias in the metrics used. To mitigate these threats, each of the experiments were repeated 20 times. Statistical tests were used to further validate the results and ensure that there was a statistically significant difference in the accuracy of the approaches. There are also internal validity tests that can arise from the hyperparameter setting used to initialise the IoTPredict and Stacked Autoencoder approaches. The design decisions in choosing the parameters for the model are described in Section 3.6. The best-performing parameters were used in the prediction accuracy experiments.

Threats to external validity are linked to the selected benchmark models, the QoS dataset and the experimental setup environment. To mitigate these threats, a number of baseline approaches covering different matrix factorisation methods are used. The dataset used is large, with 339 users and 5825 services. This gives a comprehensive evaluation of the prediction accuracy of the approaches.

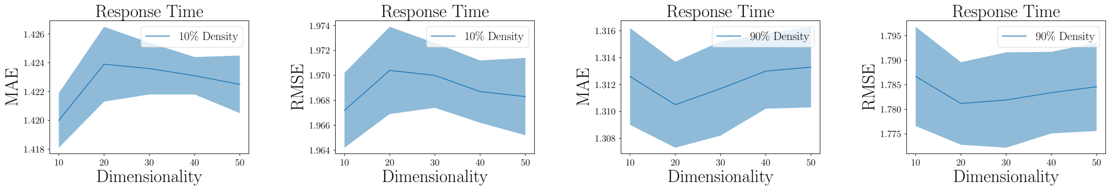
Interpretive validity is the extent to which the conclusions from the experiments are reasonable given the data, which can also be influenced by researcher bias. The full results of the experiments are reported for each of the prediction approaches on the dataset and appropriate statistical tests are used. This allows the reader to evaluate whether the conclusions drawn from the experiments are reasonable given the results.

5.3.1.6 Hyperparameters

Both TTR algorithms have a number of hyperparameters that need to be set to ensure an accurate model is created. In this section we conduct experiments to evaluate the effect of different hyperparameters on model accuracy for both approaches. We first evaluate the impact of hyperparameters on the IoTPredict algorithm:

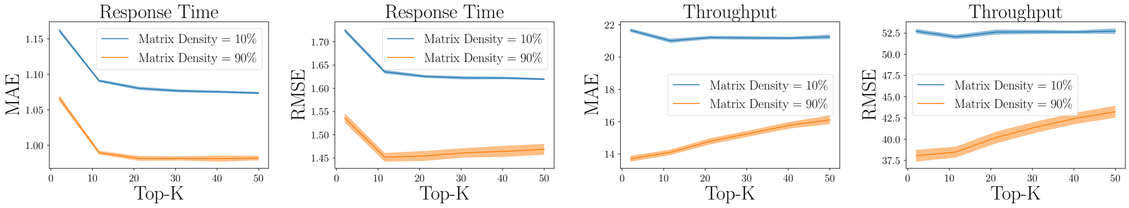
Dimensionality is the number of latent features used by the algorithm. In this section we investigate the impact this parameter has on the prediction error. Figure 5.8a and 5.8b show the impact of the latent features for 10% density on the response time dataset. The error is minimised using 10 latent features and as more latent features are added the error increases, indicating an overfitting problem.

Figure 5.8c and 5.8d show the impact of the latent features for 90% density. In this case, the MAE and RMSE are minimised with around 20-30 latent features. The



(A) (B) (C) (D)

FIGURE 5.8: Impact of Dimensionality on Response Time at 10% and 90% density



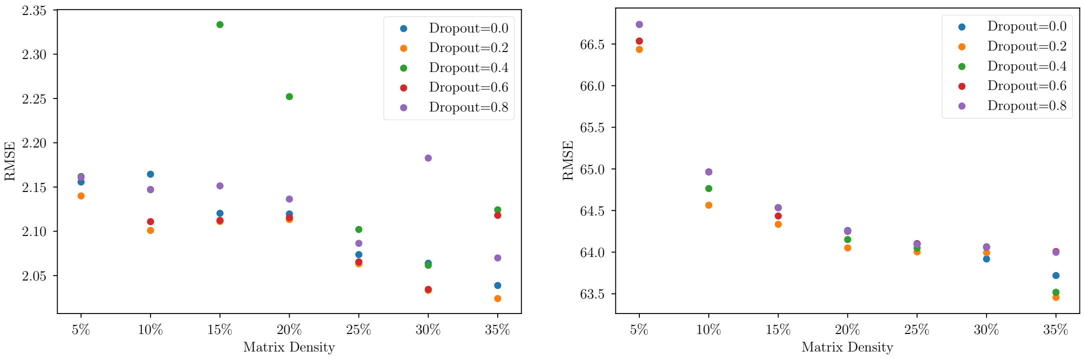
(A) (B) (C) (D)

FIGURE 5.9: Impact of Top-K on Response Time and Throughput

results in Chapter 5 are conducted with 20 latent features, which reduces error for more dense matrices as shown by these results. The results also show that the number of latent features can be tuned to improve performance if the matrix density is already known, to avoid the under/overfitting problem. In this section, we focus on providing a detailed description of our own model parameters rather than a comparison against existing approaches, which is conducted in Chapter 5.

Top-K is the number of similar users selected to make the predictions for the individual user, which can impact the prediction accuracy of the approaches. Figure 5.9 shows how the error changes for the 10% and 90% matrix densities as the top-k users are increased from 2-50, with the dimensionality set to 20 latent features. Figure 5.9a and 5.9b show the impact of the number of top-k users chosen on the response time dataset. In this case, we see that the the error drops considerably as K increases from 2 to 10 in both the MAE and RMSE. As K continues to increase, the error either remains constant or increases slightly. This can be seen when the matrix density is both at 10% and 90%.

In the throughput dataset, the response is slightly different. The 10% matrix density remains almost constant even as the number of top-k users changes. The results for the 90% matrix density are interesting and show that the error increases with the number of top-k users. The increased error is caused by using values from an increased number of users that are not similar, which decreases the prediction accuracy. Figure 5.9 also shows the problem when trying to optimise the algorithm for one QoS factor,



(A) Impact of Dropout on Response Time

(B) Impact of Dropout on Throughput

FIGURE 5.10: Impact of Dropout

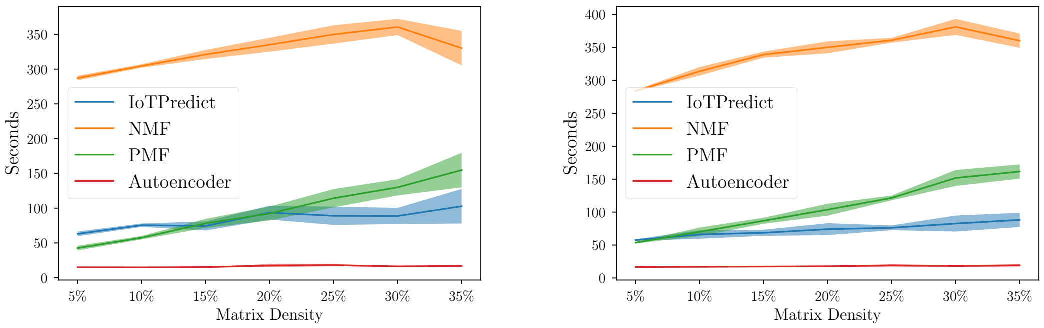
throughput, as it can make the algorithm less accurate for other QoS factors such as response time.

The stacked autoencoder approach also has hyperparamters, such as the dropout value:

Dropout The dropout parameter is evaluated for the optimal values in Figure 5.10. Figure 5.10 shows the experimentation with different dropout values. Figure 5.10a achieves the most consistent results with dropout = 0.2. For dropout = 0.4 there are some outliers for matrix density of 15% and 20%. This shows the large impact that dropout can have on the final accuracy of the model. Figure 5.10b shows the impact of dropout for the throughput dataset. This figure also shows the most consistent results using dropout = 0.2, across the different matrix densities. The large dropout = 0.8 in this case leads to the loss of information in the training an has the lowest prediction accuracy. A dropout value = 0.2 is chosen for the final experiments.

5.3.2 Results

This section presents the results of the TTR evaluation. Section 5.3.2.1 evaluates the training time of the TTR approaches to evaluate whether they are suitable to be deployed on an edge device. Section 5.3.2.2 evaluates the request time of the approaches, which is the time is takes to receive the predictions of TTR including the network delay. The following sections focus on the prediction accuracy, with 5.3.2.3 describing the use of standard error metrics to evaluate the prediction accuracy and Section 5.3.2.4 describing the sending of predictions to a composition engine to evaluate the impact that the predictions have on a final service composition.



(A) Response Time Training Time

(B) Throughput Training Time

FIGURE 5.11: Training Time of Algorithms

5.3.2.1 Training Time

Looking at Figure 5.11a, there is a large difference between the training times of the four algorithms, with the three matrix factorisation approaches taking much longer and increasing in time as the matrix density increases. The average training time for the NMF algorithm is over 300s, PMF is over 100s, IoTPredict is over 60s and the autoencoder approach takes 15s. There is a trade-off between PMF and IoTPredict, with PMF introducing less overhead for matrices with densities between 5-25% and IoTPredict introducing less overhead for matrices between 25-35% density. There is a relative speed-up of 20 and 4 times using the autoencoder approach relative to the NMF and IoTPredict matrix factorisation approaches. The results are highly statistically significant with a p-value < 0.01 and an effect size of 11.5 between the autoencoder and PMF approach at 5%. The results are also statistically significant at 35% with a p-value < 0.01 between the mean training times of the autoencoder and the next best training time for IoTPredict with an effect size of 4.886, which is large.

The results are similar for the throughput training time as seen in Figure 5.11b, with a large difference between the four approaches. At 5% matrix density, the autoencoder takes 15s, PMF takes 55s, IoTPredict takes 57s, and NMF takes 283s. As the matrix density increases the matrix factorisation approaches increase the time to train the model, while the autoencoder approach stays almost the same. At 35% matrix density the autoencoder takes 18s, IoTPredict takes 88s, PMF takes 150s and NMF takes 381s. In this case, the speed-up using the autoencoder approach is 4.8 times and 21 times compared to the IoTPredict and NMF approaches. The results are statistically significant with a p-value < 0.01 and an effect size of 33.88 at 5% and 8.93 at 35%. The dramatic reduction in training time for the autoencoder approach allows it to be deployed and trained on devices at the edge of the network one hop away from users.

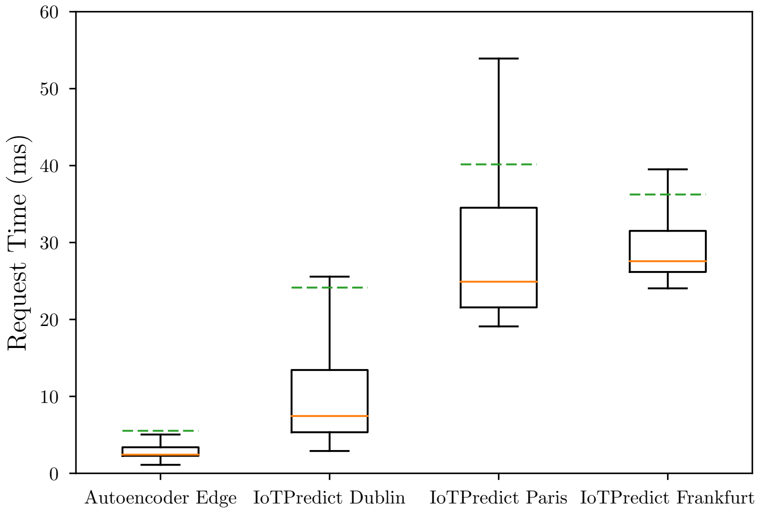


Figure 5.12: Request Time

This allows the model to constantly update with new values reported by users in the environment. This keeps the predictions accurate as QoS attributes can vary with time so it is important to update the model with recent QoS values.

5.3.2.2 Request Time

The request time is the total end-to-end time between requesting QoS predictions from the prediction algorithm to returning a response including network delay. This is an important factor when using a collaborative filtering approach for time varying values especially in dynamic service re-compositions as the composition engine needs quick access to QoS values for candidate services. The validation time of IoTPredict, NMF and PMF are the same as the matrix has been completed during training so only a constant time lookup is needed to find the QoS value of the service. The IoTPredict algorithm is deployed in the cloud as the larger training time is not suitable for edge devices. A number of cloud instances in different countries are used, to show how the cloud location can impact request time. The autoencoder approach is deployed on the Jetson Tx2 at the edge, where it is trained.

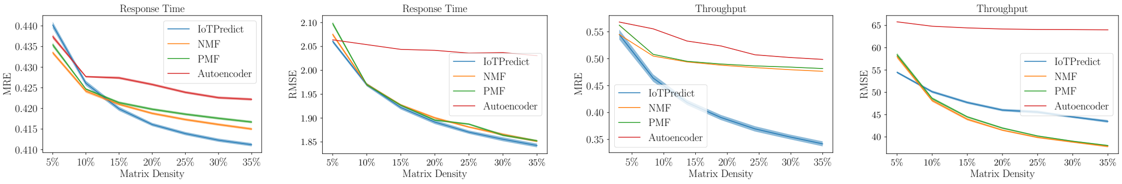
Figure 5.12 shows a boxplot with the median request time in orange and the average request time in green. The IoTPredict algorithm is implemented in three data centres and is invoked by a request from a node in Trinity College Dublin. This explains some of the variation between the data centre locations with the IoTPredict approach having reduced request time in Dublin compared to the other locations, with an average response time of 24.1ms. This is a good response time for a cloud-based approach, though it may be a special case as the Amazon Dublin data centre is located very close to Trinity College, which would not typically be the case for most cloud-based services. To evaluate this, two other geographically-close data centre locations in Paris and Frankfurt are tested. The average response time was 40.2ms in Paris and 36.2ms in Frankfurt.

The advantage of being able to deploy the stacked autoencoder at the edge can clearly be seen, with the average response time reduced to 5.40ms. This improvement in response time is especially important for dynamic service recomposition, where one of the services is forecast to fail and a new candidate service needs to be chosen. The reduction in this time will allow for a greater number of applications to recompose before the user notices a failure. Given the current distribution of data centres worldwide, the results would typically be worse for cities in South America and Asia where there may be greater distance to the nearest data centre and worse network links.

5.3.2.3 Standard Error Accuracy

Figure 5.13 shows the MRE and RMSE between the actual and predicted values for the evaluated algorithms. Figure 5.13a and 5.13b show the standard error metrics for the response time dataset. Figure 5.13a shows the MRE between the approaches, NMF reduces the prediction error for matrix density $< 15\%$. For matrix densities greater than this, the IoTPredict algorithm produces the best results. The RMSE results in Figure 5.13b show a slight improvement for the IoTPredict approach across all the matrix densities with a sizeable difference between the matrix factorisation approaches and the stacked autoencoder approach.

For the throughput dataset in Figure 5.13c, the IoTPredict algorithms produces the best MRE results across all the matrix densities. Figure 5.13d show the RMSE for the throughput dataset and the difference in accuracy of the matrix factorisation and stacked autoencoder approach, with the NMF and PMF algorithms producing the most accurate results. However, it is difficult to evaluate how the standard error accuracy will have an impact in an actual service composition. When the service composition and execution engine is selecting a suitable service composition, it will choose from the top few candidate services. As discussed in more detail in Section 5.3.1.2, it is more important that the best available service be chosen for the service composition rather than having low standard error metrics for all the available services. The prediction algorithm could make bad predictions for the services with very low QoS, which would influence the overall accuracy of the algorithm even though these services are unlikely



(A) Response time MRE (B) Response time RMSE (C) Throughput MRE (D) Throughput RMSE

FIGURE 5.13: Impact of Matrix Density on Response Time and Throughput

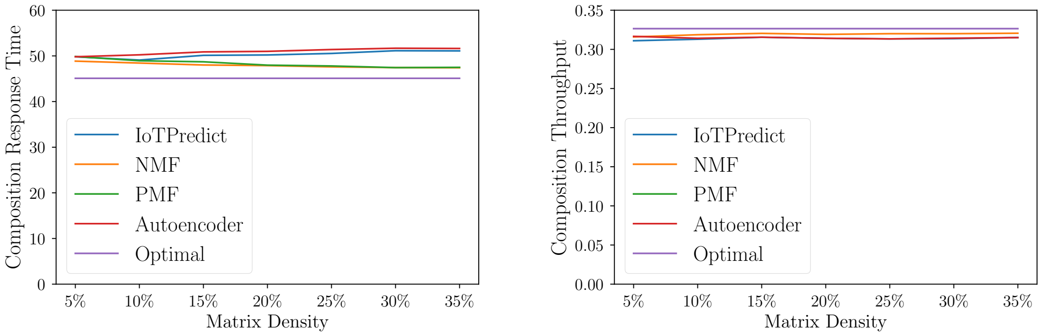
to be invoked. This is why the composition accuracy is also evaluated when using these individual prediction approaches.

5.3.2.4 Composition Accuracy

The predictions are passed to the service composition engine to evaluate the impact of prediction accuracy on the final service composition. Then, a greedy-based service composition algorithm is used to choose one of the 10 paths based on the predictions and show the average result for all the users in the dataset. The experiment is repeated 20 times and the average values are shown. As the QoS values are taken from a dataset, they do not vary with time and allow the algorithms to be evaluated with the same user values.

Figure 5.14a shows the impact of the predictions for the response time of the final composition. For low matrix densities, which would be expected in an IoT environment the results are similar with less than one second separating the final compositions at 5% matrix density. Comparing the average forecast prediction at 5% density the t statistic is 0.29 with a p-value of 0.77, which cannot reject the null hypothesis that there is a different between the means. At 35% density the t statistic is -1.29 and the p-value is 0.2, which again cannot reject the null hypothesis that there is a difference between the means. As the matrix density increases, a reduction in response time can be seen by using the PMF algorithm, however this is not statistically significant.

The results for the throughput of the composition path follow a similar pattern in Figure 5.14b with the composition engine able to generate very similar compositions based on each of the prediction algorithms for low matrix densities. For larger matrix densities greater than 15% a slight improvement in using the NMF approach can be seen. The autoencoder approach has slightly reduced throughput at 5% compared to the next best matrix factorisation-based approach, however with a p-value of 0.944 it is not statistically significant. At 35% the the autoencoder also slows slightly reduced throughput but with a p-value of 0.924 it is not statistically significant.



(A) Response Time of Service Composition (B) Throughput of Service Composition

Figure 5.14: QoS Attributes of Service Composition

The overall results for the service composition have shown that the prediction accuracy using standard error metrics such as MRE and RMSE may not have as direct an impact on the composition accuracy as thought in the state of the art. The reason for this is that standard error metrics can give large error value for services with poor QoS. For example, a service that has a response time of 15s but the model predicts a response time of 7s will incur a large error using standard metrics, but is still unlikely to be used in the final composition if an alternative candidate service has a response time of 10ms. This is seen in the difference between the prediction and composition accuracy experiments.

RQ.2 proposed in Chapter 1 asked the question, to what extent can the time to receive predictions of TTR be reduced, by updating a model at the edge of the network, while maintaining QoS prediction accuracy? The results of the experimental evaluation has shown that the stacked autoencoder approach has reduced the training time compared to the matrix factorisation-based approaches. The reduced training time allows the autoencoder to be trained and updated at the edge of the network, dramatically reducing the request time from 24.1ms for the approaches deployed in the cloud environment to 5.40ms. The reduction in training time allows the autoencoder approach to manage the 10ms tolerable delay required by modern IoT applications, such as augmented reality described in Table 1.1. The composition accuracy results have shown that there is no statistical difference in QoS between the compositions generated using predictions from the matrix factorisation-based approaches and the stacked autoencoder approach. Therefore, the answer to RQ.2 is that the time to receive predictions of TTR can be dramatically reduced by 4.46 times using a stacked autoencoder approach at the edge of the network, while maintaining QoS prediction accuracy for service compositions.

5.4 Chapter Summary

This chapter has presented the evaluation of the TTD and TTR approaches that have been proposed in this thesis. Section 5.2 was designed to evaluate RQ.1, which asks to what extent can the accuracy of forecasting to support TTD be improved, by using a lightweight model at the edge to incorporate recent changes in QoS? The results of the experimental evaluation have shown that the noisy-echo state network can be deployed and trained on an edge node, while increased the forecasting accuracy for the 9/10 (90%) of the IoT datasets and 8/10 (80%) of the web service datasets for an overall improvement in 17/20 (85%) of the combined datasets. This has addressed the limitations of existing approaches, such as LSTM and persistence models, that were either too heavyweight to be trained at the edge and took too long to incorporate recent changes in the model or were to simplistic to build a model that could capture the complexities of the environment. Therefore, the answer to RQ.1 is that a lightweight noisy-echo state network deployed at the edge to incorporate recent changes in QoS can be used to improve the accuracy of forecasting to support TTD in 85% of cases, on a combination of IoT and web service datasets. The 15% where the noisy-echo state network approach was found to not be as effective were in the datasets with low variance, skewness and kurtosis. In these datasets traditional time series approaches such as Holt-Winters and ARMA were found to be more accurate.

Section 5.3 was designed to evaluate RQ.2, which asks to what extent can the time to receive predictions of TTR be reduced, by updating a model at the edge of the network, while maintaining QoS prediction accuracy? The experiments show how the training time of the TTR algorithm can be reduced dramatically by using a stacked autoencoder on a deep edge architecture. The reduced training time allows the stacked autoencoder algorithm to be deployed on the edge reducing the request time to 5.40ms compared to 24.1ms for the cloud-based approaches. The reduction in request time allows the autoencoder approach to manage the 10ms tolerable delay required by modern IoT applications, such as augmented reality described in Table 1.1. The prediction accuracy of the IoTPredict approach was shown at low matrix densities using standard error metrics, however when the TTR predictions were used as part of a service composition no statistically significant difference in the QoS of the final compositions was observed. Therefore, the answer to RQ.2 is that the time to receive predictions of TTR can be dramatically reduced by 4.46 times using a stacked autoencoder approach, while maintaining QoS prediction accuracy for service compositions.

Urban Application	Data Type	Traffic Rate	Tolerable Delay	TTDR Delay	Criticalicty	Suitable
Waste Management [19]	Historical Data	>= 100 MB per day	>30 mins	6.8 ms	Low	Yes
Structural Health [19]	Historical Data	>= 10 MB per day	>30 mins	6.8 ms	Medium	Yes
Air Quality Monitoring [19]	Historical Data	>= 10 MB per day	>30 mins	6.8 ms	Medium	Yes
Noise Monitoring [19]	Historical Data	>= 100 MB per day	>30 mins	6.8 ms	Medium	Yes
Wearable IoT	Stream Data	<=1 GB per device	>30 mins	6.8 ms	Medium	Yes
Traffic Congestion [19]	Historical Data	>= 100 MB per day	>5 mins	6.8 ms	Low	Yes
Smart Parking [19]	Event Data	>= 10 MB per day	>1 min	6.8 ms	Low	Yes
Smart Home [19]	Stream/ Massive Data	>= 10 MB per house per day	1 s - 10 mins	6.8 ms	Medium	Yes
Smart Energy [20]	Stream/ Massive Data	>= 100 GB per day	10 ms - 10mins	6.8 ms	Medium	Yes
Remote Surgery [21]	Stream/ Massive Data	>= 50 MB per second	<= 100 ms	6.8 ms	High	No
Augmented Reality [22]	Stream/ Massive Data	>= 100 MB per second	<= 10 ms	6.8 ms	Low	Yes
Autonomous Vehicles [22]	Stream/ Massive Data	>=100 GB per vechicle per day	<= 10 ms	6.8 ms	High	No

Table 5.11: Urban Intelligence Applications TTDR Suitability

Table 5.11 shows the urban intelligence applications that were defined in Chapter 1. The table is updated with a TTDR Delay column and a column indicating the suitability of the TTDR approach. The deployment of both algorithms at the edge of the network reduces the TTDR delay, which is 6.8ms. This makes it suitable to manage the tolerable delay for each of the urban intelligence applications included modern approaches, such as augmented reality. The applications that the TTDR approach is not suitable for are the high criticality applications such as remote surgery and autonomous vehicles, as the TTDR approach is focused on providing soft QoS guarantees more suitable for medium and low criticality applications. These applications require a more fault tolerance-based approach, with a large number of redundant services available exclusively to the user on resource rich devices [256]. For applications such as autonomous vehicles a software stack is typically designed to use services available locally in the car for real time tasks such as collision detection, due to the high criticality and real time requirements of these tasks [257]. The community will be able to make use of the results of this evaluation by the QoS time series dataset that we publicallly release to evaluate the TTD approaches. We hope that the success of deploying the noisy-echo state network and the stacked autoencoder approach at the edge of the network will encourage future research in this area.

Chapter 6

Conclusion

This thesis has investigated improving the reliability of IoT applications. This chapter summaries the contributions, discusses the tradeoffs and outlines scope for future work. Section 6.1 provides a summary of the main contributions of each chapter in the thesis. Section 6.2 provides a discussion of the contributions and some of the limitations of the proposed approaches. The section also discusses some use cases to show how the prediction algorithms can be combined with other technologies such as lifelong learning to create applications such as augmented reality with IoT services. Section 6.3 outlines the scope for extending this work looking at how privacy preservation, malicious user detection and transfer learning could be included. Section 6.4 provides a final remark summarising the final chapter.

6.1 Thesis Summary

This thesis has been organised into six chapters. In this section we give a summary of the main contributions in each of the previous chapters.

Chapter 1 - Introduction introduced the problem of reliability in IoT with a description of the challenges of being able to provide reliability in a dynamic IoT environment. The limitations of existing approaches were identified and research gaps were explored, which lead to the two research questions. The overall approach of the thesis was then discussed with the assumptions that had been made and the hypothesis and objectives that were expected in the thesis. The thesis contributions were then given as well as the scope of the area that the thesis focused on.

Chapter 2 - State of the Art provided a comprehensive review of current state of the art approaches across the layers of the IoT, starting with a systematic mapping. This gave a high level overview of current state of the art approaches and identified research gaps at specific layers of the IoT, such as the middleware layer. The systematic mapping also showed the specific contributions that were needed in this layer - evaluated approaches using real data from devices. This influenced the experimental approach to collect a new dataset from deployed devices. The chapter then focused on specific reliability approaches that have been proposed in the state of the art and the limitations of using the state-based and fault tolerant techniques used in enterprise applications in IoT. The chapter then focuses on approaches that have been used to make QoS predictions for currently executing and candidate services. A number of approaches are described and organised into a taxonomy, such as model and memory-based collaborative filtering approaches. Finally, a Kiviat diagram illustrates the main differences between current state of the art approaches.

Chapter 3 - Design describes the main components of reliable service applications and how they can be modelled using a cost to time budget. This budget identifies the main components that affect the time budget, which are TTD, TTR and TTF. The design objectives and required features, system model and design decisions that were used in the design phase are introduced. The problem statement and design of the noisy-echo state network approach to reduce TTD are described. The noisy-echo state network approach focuses on increasing the forecasting accuracy of TTD approaches, while maintaining a reduced training time to allow it to be deployed on edge devices such as the Jetson Tx2. The following section then introduces the TTR problem statement as well as the initial IoTPredict approach, which used a latent features-based approach to try to improve the prediction accuracy for candidate services. A description of the stacked autoencoder approach follows, which focuses on reducing the training time of the TTR approach to allow it to be deployed on an edge device, while maintaining accurate QoS predictions. This allows for reduced request time during service adaptation, allowing the composition engine to change to the best ranked service based on QoS from the prediction engine before the user notices a failure.

Chapter 4 - Implementation provides the implementation details of the individual approaches. The chapter also introduces the other middleware components with which the QoS prediction algorithms are designed to be used, such as the service registration engine and the service composition engine. A component diagram illustrates the messages and data that are passed between the components to create reliable service applications. A sequence diagram shows the exact sequence of messages between the components showing when the QoS predictions are needed by the other components. The deep edge architecture that was used to deploy the QoS prediction algorithms at the edge of the network is then introduced, which allows for increased processing power at the edge of the network allowing more powerful models to be designed and used. Finally the implementation of the TTD and TTR approaches are given with class diagrams to show how the main classes interact with each other, how the prediction approaches were evaluated and the utility functions that were used to load and pre-process data.

Chapter 5 - Evaluation presents the evaluation of the proposed TTD and TTR approaches. The experimental setup to evaluate the TTD approaches is introduced, which includes the collection of a new IoT dataset as well as an established web services dataset. The results are then shown for the prediction accuracy, training time and request time to answer RQ.1: to what extent can the accuracy of forecasting to support TTD be improved, by using a lightweight model at the edge to incorporate recent changes in QoS? The results for the prediction accuracy have shown that for 9/10 (90%) of the IoT datasets and 8/10 (80%) of the web service datasets for an overall improvement in 17/20 (85%) of the combined datasets, by using the noisy-echo state based approach.

The evaluation of the TTR approaches first introduces the experimental setup that was used to evaluate the algorithms including the dataset that was used. The results are then presented to answer RQ.2: to what extent can the time to receive predictions of TTR be reduced, by updating a model at the edge of the network, while maintaining QoS prediction accuracy? The training time and request time of the approaches are then evaluated showing that the stacked autoencoder approach can reduce the training time needed. This allows the model to be deployed at the edge of the network, which greatly reduced the time to receive predictions. The prediction accuracy shows that matrix factorisation-based approaches such as IoTPredict were able to reduce the prediction error, however when these predictions are used as part of a service composition there is no statistically significant difference between the final service composition QoS metrics. This shows how it is possible to reduce the time to receive predictions of TTR by updating a model at the edge of the network, while maintaining QoS prediction accuracy for the service composition.

6.2 Discussion

TTDR, the combination of improvement to TTD and TTR has been evaluated using real IoT and web service QoS data, with the results compared to the existing baselines. The TTD approach has shown improved forecasting accuracy compared to other current state of the art approaches, while maintaining a small training time, allowing it to be deployed at the edge of the network. The TTR approach has shown a reduction in the training time allowing it to be deployed at the edge of the network, reducing the request time, while maintaining prediction accuracy for the final service composition. The reduced request time improves the reliability of IoT applications especially during a dynamic service adaptation when there is a small amount of time to find a suitable replacement candidate service. In this section, the contributions to the body of knowledge are outlined, followed by an introduction to potential use cases.

6.2.1 Thesis Contribution

This thesis has made three contributions to the body of knowledge. The first is the improved accuracy of the TTD forecasts using a noisy-echo state network approach. The increased forecasting accuracy allows the middleware to react more quickly to dynamic changes in the environment and perform a proactive service adaptation to select suitable replacement services before there has been an actual failure. The small training and testing time allow the algorithm to be deployed on an edge device. The limitations of the work are discussed in Section 6.2.1.1.

The second and third contribution focuses on the reduction of the TTR once an error has been detected by the TTD approach. The second contribution is IoTPredict, which was able to improve the prediction accuracy for QoS values. An important component of the TTD approach is the request time needed to receive the predictions, especially during a dynamic service composition where there is a limited amount of time to replace the failing service with a candidate service that can fulfil the users QoS requirements. The third contribution of this thesis was a stacked autoencoder approach that can be used to reduce the training time to allow the autoencoder approach to be deployed on an edge device. This reduces the request time, allowing the candidate service to be chosen more quickly. A comparison with current state of the art approaches for composition accuracy showed that there was no statistically significant difference between the composition accuracy of the proposed approach and current state of the art prediction algorithms. Therefore, we have shown how the proposed approach can be deployed at the edge of the network, while maintaining accurate QoS predictions.

This has a number of other benefits in terms of user privacy and the reduced traffic rate of data being sent to the cloud. The limitations of the work are discussed in Section 6.2.1.2.

6.2.1.1 Time to Detection Limitations

The evaluation shows that the noisy-echo state network approach increases the forecasting accuracy of the time to detection while maintaining a short training time, addressing RQ.1 in Section 1.3.4. However, these results rely on users accepting the monitoring component as part of the middleware to build up the collection of QoS time series values. Some users may have privacy and security concerns over releasing QoS information to a middlware but this information is needed to provide accurate QoS forecasts to users. Additional privacy preserving learning techniques are planned to allow the models to train on local QoS data while only updating the changes in gradient in the updated model, which can be encrypted and sent to the local gateway, discussed in more detail in Section 6.3.1. This federated learning approach may be able to provide QoS forecasts with improved privacy but may reduce the forecast accuracy so needs to be evaluated in future work.

The noisy-echo state approach has reduced the prediction error compared to other forecasting approaches, however there is still some prediction error, which can lead to a service not being forecast that it will fail. This is why our approach is not recommended for safety-critical hard real-time applications such as autonomous vehicle collision detection, as any failure could be have serious threat to human life. The TTD approach is designed to be used for soft QoS guarantees in applications such as augmented reality, which need reduced latency, but are not safety critical. The forecasting approach may also introduce some additional overhead if there is a forecast that a service will fail, when it actually will continue to operate within the agreed limits. This can cause the middleware to perform a dynamic service composition to recompose the application with a suitable replacement service, when it is not needed. This problem can be improved by further improving the forecasting accuracy of the noisy-echo state network approach.

6.2.1.2 Time to Recovery Limitations

The evaluation shows that the stacked autoencoder approach reduces the request time during adaptive service applications by allowing the model to be deployed at the edge of the network. This approach also relies on users reporting the QoS values to the middleware, which raises the same privacy and security concerns as users may not be comfortable releasing this information. The TTR approach will investigate distributed learning approaches such as federated learning to avoid this problem.

The TTR approach is designed to provide soft QoS guarantees where we accept that there will be some downtime. For safety critical applications such as collision detection for autonomous vehicles an alternative fault tolerant approach with large amounts of additional redundancy would be needed. The TTDR approach is designed for non-safety critical applications such as augmented reality that have a strict tolerable delay to create an immersive environment, but can handle some service downtime without a threat to human life.

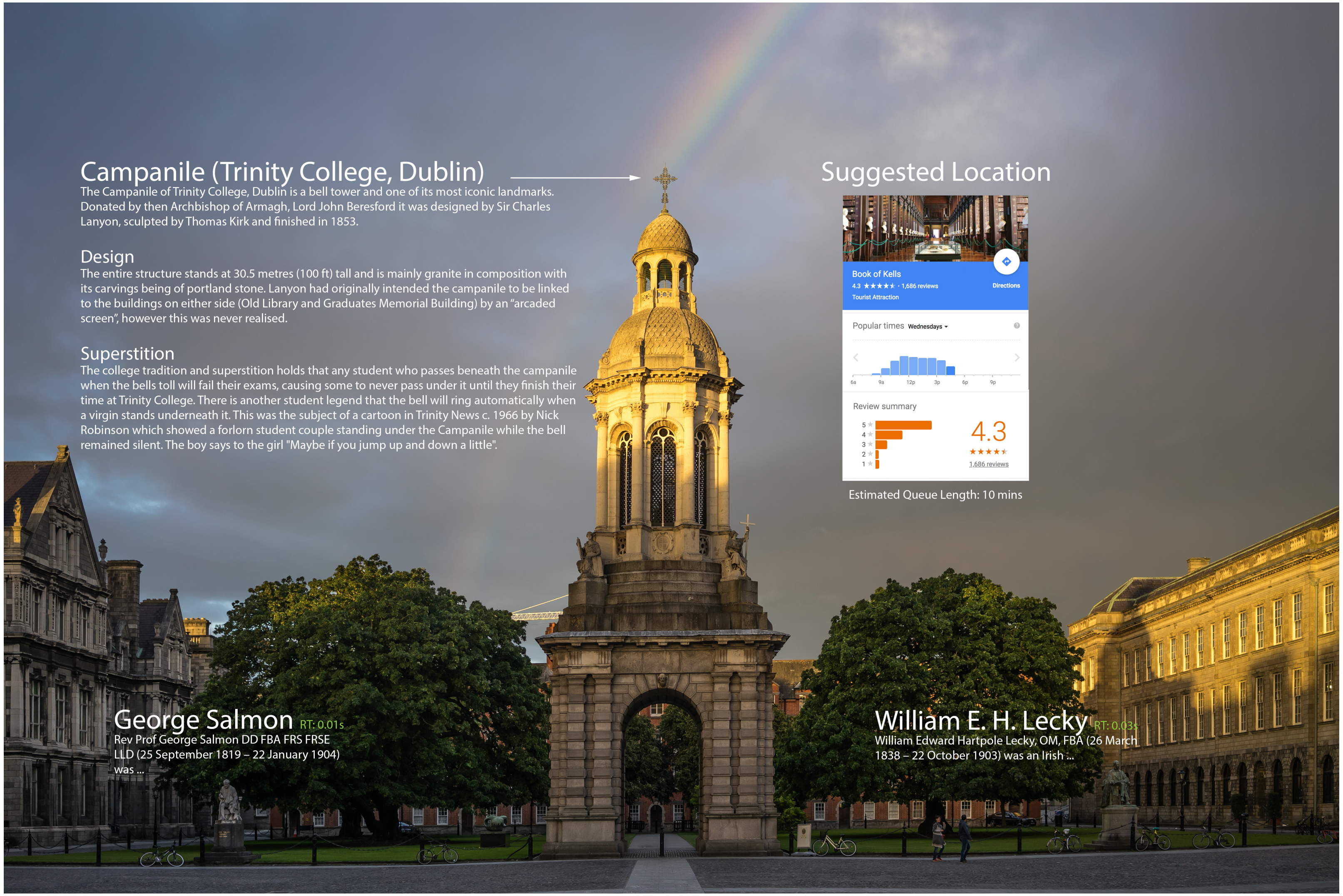
6.2.2 Use Cases

Increased reliability and QoS allow a number of modern applications to take advantage of services available in the environment that previously would have been too unreliable or not as responsive to adaptation to meet the QoS demands of the application. Section 6.2.2.1 discusses ideas on the use of TTDR in augmented reality applications and Section 6.2.2.2 discusses ideas in quantified-self applications.

6.2.2.1 Augmented Reality

Augmented reality applications have a reduced tolerable delay as seen in Table 1.1, requiring the forecasting accuracy of the TTD approach and the reduced request time of the TTR approach to find a suitable replacement service, when integrating IoT services. To work effectively in the real world, these applications require a number of innovations from different fields to tackle the real-time technical challenges such as lifelong learning for context and situation awareness, on-the-fly QoS-aware service orchestration for dynamic environments and an edge architecture to distribute these intelligent functions [258]. The prediction algorithms proposed in this thesis have been proposed as part of a Context-Aware Edge intelligence for Service orchestration in Augmented Reality applications, named as *CAESAR*. This is a collaborative research plan with the University of Helsinki and the University of St Andrews. This plan is for further research to be conducted on improved QoS at the edge and to pursue some of the future work that is highlighted in Section 6.3.

Figure 6.1 is a smart city application that is described in a ‘blue sky’ paper on the possible applications of combining AR and reliable IoT services [259]. The application



Campanile (Trinity College, Dublin)

The Campanile of Trinity College, Dublin is a bell tower and one of its most iconic landmarks. Donated by then Archbishop of Armagh, Lord John Beresford it was designed by Sir Charles Lanyon, sculpted by Thomas Kirk and finished in 1853.

Design

The entire structure stands at 30.5 metres (100 ft) tall and is mainly granite in composition with its carvings being of portland stone. Lanyon had originally intended the campanile to be linked to the buildings on either side (Old Library and Graduates Memorial Building) by an "arcaded screen", however this was never realised.

Superstition

The college tradition and superstition holds that any student who passes beneath the campanile when the bells toll will fail their exams, causing some to never pass under it until they finish their time at Trinity College. There is another student legend that the bell will ring automatically when a virgin stands underneath it. This was the subject of a cartoon in Trinity News c. 1966 by Nick Robinson which showed a forlorn student couple standing under the Campanile while the bell remained silent. The boy says to the girl "Maybe if you jump up and down a little".

Suggested Location

Book of Kells
4.3 ★★★★☆ · 1,686 reviews
Tourist Attraction

Popular times Wednesdays

Review summary

4.3 ★★★★☆ 1,686 reviews

Estimated Queue Length: 10 mins

George Salmon RT: 0.01s
Rev Prof George Salmon DD FBA FRS FRSE LLD (25 September 1819 – 22 January 1904) was ...

William E. H. Lecky RT: 0.03s
William Edward Hartpole Lecky, OM, FBA (26 March 1838 – 22 October 1903) was an Irish ...

Figure 6.1: Tourist Interacting with Services in AR

is for smart tourism, which would be applicable to a range of scenarios e.g., museums, historical sites, points of interest in a city and smart campuses. The AR application uses services available from smart beacons in the surrounding location to display additional information about the point of interest. For example, a statue could provide additional information about when the statue was created, who created the statue and the meaning behind it. For statues of people such as previous Provosts in a campus, AR can make the statue come to life and digitally move around, while presenting this additional information about their history and creation. The TTR QoS prediction algorithm is used to make accurate QoS predictions for these services, which can be seen in the green writing beside the service for additional information about the statues. This means that the services have suitable response time and can be used as part of the application. The AR application can also make use of web services provided by the middleware to make recommendations of places to visit next with ratings and queue length times from IoT services. Figure 6.1 shows an example of an application that could be developed with the combination of lifelong context and situational awareness, QoS aware service composition and an edge architecture to distribute these intelligent functions.

6.2.2.2 Quantified-Self

An increase in the availability and reduction in cost of wearable devices has allowed people to self-track multiple streams of data. Activity trackers such as the Fitbit Charge 3, Samsung Gear Fit2 Pro and Apple Watch Series 4 are now used by a large number of consumers. More recent devices have explored additional activities that users may want to track, such as connected inhalers, smart insulin pens and asthma monitors. These novel streams of data can be integrated and mined to gain insights that enable preventive actions in managing chronic conditions such as diabetis and asthma (e.g., ADAMM Asthma Monitor can detect the symptoms of an asthma attack before its onset) [260, 261].

The rise in popularity of these devices has led to the development of communities of quantified self and lifelogging [262]. This is a movement to incorporate technology into data collection on aspects of a person's daily life, with the goal of improved physical, cognitive and/or emotional health. A lot of work has been done on the physical output, especially with the emergence of P4 medicine: predictive, preventive, personalised and participatory [263]. The TTD and TTR algorithms can be used to improve existing applications reliability to ensure that the user has as much information available as possible from tracking sensors and services available in the surrounding area. Applications will also be able to integrate services in the environment such as weather and pollution information, which may have an impact on their physical activity. Initial experiments with quantified-self data to increase productivity has shown some missing data due to a service not being available or failing [264]. This initial work can be extended with the TTDR approach to increase reliability of services that track data about users. This will allow for more detailed quantitative information and rules to be generated for the user.

6.3 Future Work

There is considerable scope for extending this work. Future research directions are:

6.3.1 Privacy Preservation

With the introduction of GDPR, users have become much more careful about the data that is collected about them, how that data is stored and who will have access to that data. One way to train a model without the user ever releasing any personal information is through federated learning [265]. In this training method, a master model is deployed in the cloud and is updated from the embedded GPUs located throughout the city. The updates can be merged into the master model immediately in an encrypted fashion so that no individual update is stored online and no training data is exchanged. This would allow the training of accurate models, while respecting the privacy of users and conforming to GDPR. There are other alternative decentralised learning approaches that could be used such as decentralised deep learning [266], communication-efficient learning [267] and distributed optimisation [268].

6.3.2 Malicious User Detection

The current stacked autoencoder approach assumes that users are reporting accurate QoS values and there are no malicious users in the environment. However, some companies or individual users may report false QoS values about competing services so that users would be unlikely to use those services in the final composition. One approach to avoid this, is to perform some preprocessing on the dataset to cluster users based on the values they report. A user's reputation can then be updated if they are submitting QoS values that are significantly different to what other users in similar locations are experiencing. This reputation would then be taken into consideration

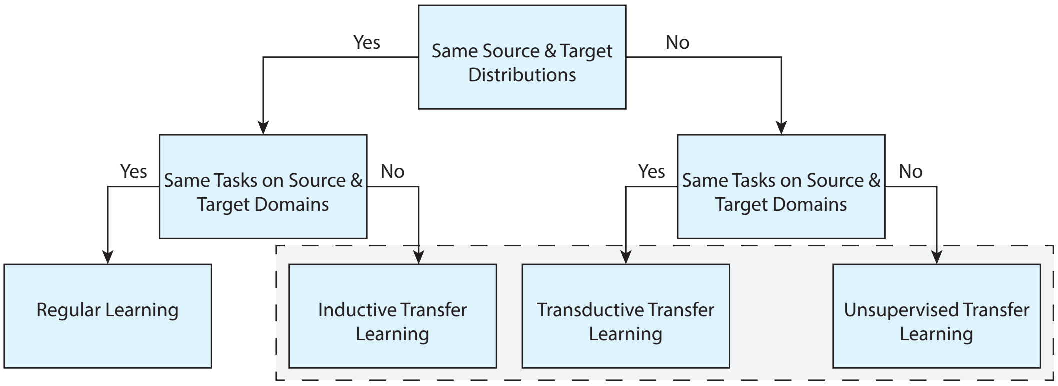

Same Source & Target Distributions
Yes
No
Same Tasks on Source & Target Domains
Yes
No
Same Tasks on Source & Target Domains
Yes
No
Regular Learning
Inductive Transfer Learning
Transductive Transfer Learning
Unsupervised Transfer Learning

Figure 6.2: Transfer Learning Categories

when making the QoS predictions and users with low reputation would not have much effect on the final composition.

6.3.3 Transfer Learning

An alternative way to speed up the training of models at the edge of the network is to use transfer learning. With the recent success of deep learning, a number of pre-trained models are now available to users who may not have the data or computational capability to train them. The computational capability at the edge of the network is limited compared to the capability in the cloud. Machine learning libraries such as Tensorflow have set up hubs to allow for the easy sharing of pre-trained models[1]. This allows easy access to large models that can take a very long time to train.

Figure 6.2 shows the different categories of transfer learning based on the relationship between the source and target distributions. The most simple case is regular learning where the source and target have the same distributions and are required to perform the same tasks. When the source and target have the same distribution or are in the same domain but the tasks that they are required to perform are different this is called inductive transfer learning. This category can further be broken down depending upon whether the source domains contain labelled data or not: if a lot of labelled data in the source domain are available then it is multi-task learning and if there is no labelled data from the source domain then it is self-taught learning [269]. When the source and target distribution are not the same but the tasks are similar it is called transductive transfer learning. In this situation, no labelled data in the target domain are available while a lot of data in the source domains are available. The final category

[1] https://www.tensorflow.org/hub

is unsupervised transfer learning where there is a difference in both the source and target distribution and tasks. This category focuses on solving unsupervised tasks in the target domain such as clustering [270] and dimensionality reduction [271], with no labeled data available in the source and target domains in training. The specific approaches that are planned to be focused on are feature extraction, fine-tuning and data augmentation.

Deep learning models are layered architectures that learn different features at different layers. These layers are finally connected to a last layer, which is usually fully connected, in the case of classification, to get the final output. This layered architecture allows us to utilise pre-trained networks without a final layer as a fixed feature-extractor for other tasks. A model can be used as a feature extractor where we freeze (fix weights and don't train) all the blocks of layers and flattening layer before the final fully connected layer. We update only the fully connected classifier block at the end of the model. This allows the new model to transform the data from a new domain task into a large dimension vector based on hidden states, thus enabling us to extract features from a new domain task using the source domain. In fine tuning, only the weights of the last few layers of the neural architecture are updated. This is slightly more resource intensive than the feature extraction approach, but can produce more accurate results. As deep neural networks are layered with the initial layers capturing the most basic features such as edges and the later layers capture more specific details about the task, we can freeze some of the first blocks and update the later ones. Data augmentation generates artificial data based on existing observation to improve model accuracy, generalisation and control overfitting [272]. These transfer learning techniques allow for a range of possibilities for training accurate models quickly at the edge of the network without user data being exposed to the cloud.

6.4 Final Remark

This thesis investigated how to improve the reliability of IoT applications in dynamic environments. A cost to time budget was created that models the parameters that can effect reliability in a dynamic IoT environment. Two of the parameters, TTD and TTR, were reduced to allow for the creation of more reliable applications. The discussion in Section 6.2 presented some of the limitations of the current TTD and TTR approaches. As they are both user-based reliability approaches they rely on users submitting their QoS values to the middleware. The approaches have improved both the forecasting and prediction accuracy, but they are still not suitable for hard real-time QoS applications such as autonomous vehicle collision detection. These applications are safety critical and would require a fault tolerant-based approach, with a number of redundant services available locally on the vehicle. There are a number of implications from this thesis, with a suitable dataset for evaluating TTD newly proposed algorithms will be able to compare themselves against state of the art approaches. The algorithms that we have proposed at the edge have shown reduced response time while being able to maintain prediction accuracy, which will lead to further algorithms being designed to be deployed at the edge of the network.

The original challenges defined at the start of the thesis were a dynamic environment, reduced tolerable delay, increased traffic rate, critical applications and limited resources. The contributions have been designed from the ground up to meet these challenges. Both the noisy-echo state network TTD approach and stacked autoencoder TTR approach are able to handle the dynamic environment by having a short training time. They are both able to be deployed at the edge of the network, which reduces the tolerable delay and traffic that has to be sent to the cloud. Both of the approaches can handle medium criticality applications, but are still not suitable for highly critical applications, such as autonomous vehicles and remote surgery, which require additional redundant services. The deep edge network has increased the limited resources available at the edge of the network by using embedded GPUs to train the algorithms.

The use cases presented in Section 6.2.2 show the applications that could potentially be created by combining the improvements in reliable IoT services with other technologies. The first use case on augmented reality shows how the improvements in reliability could be combined with lifelong learning for context and an edge architecture to distribute intelligent functions. This could transform the way that people interact with services in a smart city environment. The second use case shows how TTDR could be used in quantified-self applications to give users more detailed information about the activities that they do during the day e.g., physical activities such as running or what applications they are using on their laptop. The combination of this information with services available in the environment providing local information about air temperature, noise and pollution allows for detailed analysis of how these factors effect performance. Association rules mining can be used as in the productivity approach to define rules for the best sleep and activity level that a user should do to achieve the highest level of productivity [264]. Section 6.3 outlines the future work that is planned to extend the current approaches proposed in this thesis through the use of privacy protection, malicious user detection and transfer learning.

Bibliography

[1] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376, 2015.

[2] A. Nordrum. Popular internet of things forecast of 50 billion devices by 2020 is outdated, 2017. URL `https://spectrum.ieee.org/techtalk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated`.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015. ISSN 1553-877X. doi: 10.1109/COMST.2015.2444095.

[4] Paolo Bellavista, Giuseppe Cardone, Antonio Corradi, and Luca Foschini. Convergence of manet and wsn in iot urban scenarios. *Sensors Journal, IEEE*, 13 (10):3558–3567, 2013.

[5] Thomas Erl. *Service-oriented architecture*. Pearson Education Incorporated, 2005.

[6] Y Natis and Roy Schulte. Introduction to service-oriented architecture. *Gartner Group*, 14, 2003.

[7] Leonard Richardson and Sam Ruby. *RESTful web services*. " O'Reilly Media, Inc.", 2008.

[8] Sayed Hashimi. Service-oriented architecture explained. *ONDotnet.com (online)*, 2003. URL `https://www.cin.ufpe.br/~ajsc2/SOA%20explained.pdf`.

[9] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004.

[10] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, 28(1):1–18, 2005.

[11] Michael G Merideth, Arun Iyengar, Thomas Mikalsen, Stefan Tai, Isabelle Rouvellou, and Priya Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 131–140. IEEE, 2005.

[12] Seppo Leminen, Mika Westerlund, Mervi Rajahonka, and Riikka Siuruainen. Towards iot ecosystems and business models. In *Internet of things, smart spaces, and next generation networking*, pages 15–26. Springer, 2012.

[13] Christian Cabrera, Gary White, Andrei Palade, and Siobhán Clarke. The right service at the right place: a service model for smart cities. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10. IEEE, 2018.

[14] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science and Engineering Survey*, 2(3):94–105, 2011.

[15] Andrei Palade, Christian Cabrera, Gary White, Mohammad Abdur Razzaque, and Siobhán Clarke. Middleware for internet of things: A quantitative evaluation in small scale. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2017.

[16] Andreas Vogel, Brigitte Kerherve, Gregor von Bochmann, and Jan Gecsei. Distributed multimedia and qos: A survey. *IEEE multimedia*, 2(2):10–19, 1995.

[17] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[18] Ling Li, Shancang Li, and Shanshan Zhao. Qos-aware scheduling of services-oriented internet of things. *IEEE Transactions on Industrial Informatics*, 10(2): 1497–1505, 2014.

[19] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi. Internet of things for smart cities. *IEEE Internet of Things Journal*, 1(1):22–32, Feb 2014. ISSN 2327-4662. doi: 10.1109/JIOT.2014.2306328.

[20] S Abdul Salam, Sahibzada Ali Mahmud, Gul Muhammad Khan, and Hamed S Al-Raweshidy. M2m communication in smart grids: Implementation scenarios and performance analysis. In *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 142–147. Ieee, 2012.

[21] Manuela Perez, Song Xu, Sanket Chauhan, Alyssa Tanaka, Khara Simpson, Haidar Abdul-Muhsin, and Roger Smith. Impact of delay on telesurgical performance: study on the robotic simulator dv-trainer. *International journal of computer assisted radiology and surgery*, 11(4):581–587, 2016.

[22] GSMA Netw. Unlocking commercial opportunities from 4g evolution to 5g, 2016. URL `https://www.gsma.com/futurenetworks/wp-content/uploads/2016/02/704_GSMA_unlocking_comm_opp_report_v5.pdf`.

[23] M. Satyanarayanan. The emergence of edge computing. *Computer*, 50(1):30–39, Jan 2017. ISSN 0018-9162. doi: 10.1109/MC.2017.9.

[24] Nitinder Mohan and Jussi Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6. IEEE, 2016.

[25] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. In *IEEE International Conference on Cloud Computing*, pages 626–631. Springer, 2009.

[26] Abderrahmen Mtibaa, Khaled A Harras, and Afnan Fahim. Towards computational offloading in mobile device clouds. In *2013 IEEE 5th international conference on cloud computing technology and science*, volume 1, pages 331–338. IEEE, 2013.

[27] Yang Syu, Jong-Yih Kuo, and Yong-Yi Fanjiang. Time series forecasting for dynamic quality of web services: An empirical study. *Journal of Systems and Software*, 2017.

[28] Neal Wagner, Zbigniew Michalewicz, Moutaz Khouja, and Rob Roy McGregor. Time series forecasting for dynamic environments: the dyfor genetic program model. *IEEE transactions on evolutionary computation*, 11(4):433–452, 2007.

[29] Peter J Brockwell, Richard A Davis, and Matthew V Calder. *Introduction to time series and forecasting*, volume 2. Springer, 2002.

[30] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang. Time-aware service recommendation for mashup creation. *IEEE Transactions on Services Computing*, 8(3):356–368, May 2015. ISSN 1939-1374. doi: 10.1109/TSC.2014.2381496.

[31] A. Amin, A. Colman, and L. Grunske. An approach to forecasting qos attributes of web services based on arima and garch models. In *2012 IEEE 19th International Conference on Web Services*, pages 74–81, June 2012. doi: 10.1109/ICWS.2012.37.

[32] Ayman Amin, Lars Grunske, and Alan Colman. An automated approach to forecasting qos attributes based on linear and non-linear time series modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 130–139. ACM, 2012.

[33] M. Godse, U. Bellur, and R. Sonar. Automating qos based service selection. In *2010 IEEE International Conference on Web Services*, pages 534–541, July 2010. doi: 10.1109/ICWS.2010.58.

[34] Y. Xia, J. Ding, X. Luo, and Q. Zhu. Dependability prediction of ws-bpel service compositions using petri net and time series models. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pages 192–202, March 2013. doi: 10.1109/SOSE.2013.8.

[35] Twittie Senivongse and Nitirojht Wongsawangpanich. Composing services of different granularity and varying qos using genetic algorithm. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, 2011.

[36] Yang Syu, Yong-Yi Fanjiang, Jong-Yih Kuo, and Shang-Pin Ma. Applying genetic programming for time-aware dynamic qos prediction. In *Mobile Services (MS), 2015 IEEE International Conference on*, pages 217–224. IEEE, 2015.

[37] Mahmod Hosein Zadeh and Mir Ali Seyyedi. Qos monitoring for web services by time series forecasting. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 5, pages 659–663. IEEE, 2010.

[38] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, 2017.

[39] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[40] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional -crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[41] Hamza Labbaci, B Medjahed, and Y Aklouf. A deep learning approach for quality-aware long-term service composition. In *Proc. of International Conference on Service-Oriented Computing*, pages 1–8, 2017.

[42] Z. Zheng, Y. Zhang, and M. R. Lyu. Investigating qos of real-world web services. *IEEE Transactions on Services Computing*, 7(1):32–39, Jan 2014. ISSN 1939-1374. doi: 10.1109/TSC.2012.34.

[43] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, 1998. ISBN 1-55860-555-X.

[44] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004. ISSN 1046-8188. doi: 10.1145/963770.963776.

[45] Hao Ma, Irwin King, and Michael R. Lyu. Effective missing data prediction for collaborative filtering. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–46. ACM, 2007. ISBN 978-1-59593-597-7.

[46] Paul Resnick et al. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, pages 175–186. ACM, 1994. ISBN 0-89791-689-1.

[47] Zibin Zheng, Yilei Zhang, and Michael R. lyu. Cloudrank: A qos-driven component ranking framework for cloud computing. In *Proceedings of the 2010 29th IEEE Symposium on Reliable Distributed Systems*, SRDS '10, pages 184–193, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4250-8.

[48] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu. Web service recommendation via exploiting location and qos information. *IEEE Transactions on Parallel and Distributed Systems*, 25(7):1913–1924, July 2014. ISSN 1045-9219. doi: 10.1109/TPDS.2013.308.

[49] Y. Zhang, Z. Zheng, and M. R. Lyu. Wspred: A time-aware personalized qos prediction framework for web services. In *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, pages 210–219, Nov 2011. doi: 10.1109/ISSRE.2011.17.

[50] Lina Yao, Quan Z Sheng, Aviv Segev, and Jian Yu. Recommending web services via combining collaborative filtering with content-based features. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 42–49. IEEE, 2013.

[51] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun. Personalized qos-aware web service recommendation and visualization. *IEEE Transactions on Services Computing*, 6(1):35–47, 2013.

[52] J. Xu, Z. Zheng, and M. R. Lyu. Web service personalized quality of service prediction via reputation-based matrix factorization. *IEEE Transactions on Reliability*, 65(1):28–37, March 2016. ISSN 0018-9529. doi: 10.1109/TR.2015.2464075.

[53] Le Van Thinh. Qos prediction for web services based on restricted boltzmann machines. *Journal of Service Science Research*, 9(2):197–217, Dec 2017. ISSN 2093-0739. doi: 10.1007/s12927-017-0010-6. URL `https://doi.org/10.1007/s12927-017-0010-6`.

[54] Xi Chen, Zibin Zheng, Qi Yu, and Michael R Lyu. Web service recommendation via exploiting location and qos information. *IEEE Transactions on Parallel and distributed systems*, 25(7):1913–1924, 2013.

[55] Gary White and Siobhán Clarke. Smart cities with deep edges. In *ECML PKDD 2018 Workshops*, pages 53–64, Cham, 2019. Springer International Publishing. ISBN 978-3-030-13453-2.

[56] Gary White, Andrei Palade, and Siobhán Clarke. Qos prediction for reliable service composition in iot. In *International Conference on Service-Oriented Computing*, pages 149–160. Springer, 2017.

[57] Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource provisioning for iot services in the fog. In *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*, pages 32–39. IEEE, 2016.

[58] Andrei Palade, Christian Cabrera, Gary White, and Siobhán Clarke. Stigmergic service composition and adaptation in mobile environments. In *International Conference on Service-Oriented Computing*, pages 618–633. Springer, 2018.

[59] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2014.09.003. URL `http://www.sciencedirect.com/science/article/pii/S0893608014002135`.

[60] Irfan Awan and Muhammad Younas. Towards qos in internet of things for delay sensitive information. In Maristella Matera and Gustavo Rossi, editors, *Trends in Mobile Web Information Systems*, pages 86–94, Cham, 2013. Springer International Publishing. ISBN 978-3-319-03737-0.

[61] FU Zhen-Xu and Zheng-Shi ZHANG. Research on rpr technology and its application in nuclear power plant. *DEStech Transactions on Computer Science and Engineering*, (ccme), 2018.

[62] Ahmed Amari, Ahlem Mifdaoui, Fabrice Frances, Jérôme Lacan, David Rambaud, and Loic Urbain. Aeroring: Avionics full duplex ethernet ring with high availability and qos management. In *European Congress on Embedded Real Time Software and systems 2016*, pages pp–150, 2016.

[63] Jonah Caplan, Zaid Al-Bayati, Haibo Zeng, and Brett H Meyer. Mapping and scheduling mixed-criticality systems with on-demand redundancy. *IEEE Transactions on Computers*, 67(4):582–588, 2017.

[64] Allen Starke, D Kumar, M Ford, Janise McNair, and A Bell. A test bed study of network determinism for heterogeneous traffic using time-triggered ethernet. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*, pages 611–616. IEEE, 2017.

[65] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, 2017.

[66] Hong-Linh Truong and Schahram Dustdar. Principles for engineering iot cloud systems. *IEEE Cloud Computing*, 2(2):68–76, 2015.

[67] Irfan Awan, Muhammad Younas, and Wajia Naveed. Modelling qos in iot applications. In *2014 17th International Conference on Network-Based Information Systems*, pages 99–105. IEEE, 2014.

[68] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *2010 IEEE International Conference on Web Services*, pages 369–376. IEEE, 2010.

[69] Marc Oriol, Jordi Marco, and Xavier Franch. Quality models for web services: A systematic mapping. *Information and Software Technology*, 56(10):1167 – 1182, 2014. ISSN 0950-5849. doi: 10.1016/j.infsof.2014.03.012. URL `http://www.sciencedirect.com/science/article/pii/S0950584914000822`.

[70] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th international conference on evaluation and assessment in software engineering*, volume 17, pages 1–10. sn, 2008.

[71] Gary White, Vivek Nallur, and Siobhán Clarke. Quality of service approaches in iot: A systematic mapping. *Journal of Systems and Software*, 132:186 – 203, 2017. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2017.05.125. URL `http://www.sciencedirect.com/science/article/pii/S016412121730105X`.

[72] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, page 38. Citeseer, 2014.

[73] Barbara Kitchenham, Rialette Pretorius, David Budgen, O. Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. Systematic literature reviews in software engineering – a tertiary study. *Information and Software Technology*, 52(8):792–805, 2010. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.03.006. URL `http://www.sciencedirect.com/science/article/pii/S0950584910000467`.

[74] Barbara Kitchenham, O. Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering – a systematic literature review. *Information and Software Technology*, 51(1):7–15, 2009. ISSN 0950-5849. doi: 10.1016/j.infsof.2008.09.009. URL `http://www.sciencedirect.com/science/article/pii/S0950584908001390`.

[75] Roel Wieringa, Neil Maiden, Nancy Mead, and Colette Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements engineering*, 11(1):102–107, 2006.

[76] Xiaoheng Deng, Lifang He, Jinsong Gui, Qionglin Peng, and Tingting He. Modeling and analysis mac layer performance for ieee 802.11 s wireless mesh network in smart grid. In *2015 International Conference on Identification, Information, and Knowledge in the Internet of Things (IIKI)*, pages 280–283. IEEE, 2015.

[77] Francois Despaux, Ye-Qiong Song, and Abdelkader Lahmadi. Modelling and performance analysis of wireless sensor networks using process mining techniques: Contikimac use case. In *2014 IEEE International Conference on Distributed Computing in Sensor Systems*, pages 225–232. IEEE, 2014.

[78] Hong-Linh Truong, Georgiana Copil, Schahram Dustdar, Duc-Hung Le, Daniel Moldovan, and Stefan Nastic. icomot–a toolset for managing iot cloud systems. In *2015 16th IEEE International Conference on Mobile Data Management*, volume 1, pages 299–302. IEEE, 2015.

[79] Pieter De Mil, Tim Allemeersch, Ingrid Moerman, Piet Demeester, and Wim De Kimpe. A scalable low-power wsan solution for large-scale building automation. In *2008 IEEE International Conference on Communications*, pages 3130–3135. IEEE, 2008.

[80] Degan Zhang, Guang Li, Ke Zheng, Xuechao Ming, and Zhao-Hua Pan. An energy-balanced routing method based on forward-aware factor for wireless sensor networks. *IEEE transactions on industrial informatics*, 10(1):766–773, 2013.

[81] Mudasir Ahmad. Reliability models for the internet of things: A paradigm shift. In *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pages 52–59. IEEE, 2014.

[82] Kallol Das and Paul Havinga. Evaluation of dect for low latency real-time industrial control networks. In *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, pages 10–17. IEEE, 2013.

[83] Bardia Safaei, Amir Mahdi Hosseini Monazzah, Milad Barzegar Bafroei, and Alireza Ejlali. Reliability side-effects in internet of things application layer protocols. In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 207–212. IEEE, 2017.

[84] Daniel Macedo, Luiz Affonso Guedes, and Ivanovitch Silva. A dependability evaluation for internet of things incorporating redundancy aspects. In *Proceedings of the 11th IEEE International Conference on Networking, Sensing and Control*, pages 417–422. IEEE, 2014.

[85] James Kempf, Jari Arkko, Neda Beheshti, and Kiran Yedavalli. Thoughts on reliability in the internet of things. In *Interconnecting smart objects with the Internet workshop*, volume 1, pages 1–4, 2011.

[86] Arun K Somani and Nitin H Vaidya. Understanding fault tolerance and reliability. *Computer*, (4):45–50, 1997.

[87] Pat Pik Wah Chan, Michael R Lyu, and Miroslaw Malek. Making services fault tolerant. In *International Service Availability Symposium*, pages 43–61. Springer, 2006.

[88] Sergio Gorender, Raimundo Jose de Araujo Macedo, and Michel Raynal. An adaptive programming model for fault-tolerant distributed computing. *IEEE Transactions on Dependable and Secure Computing*, 4(1):18–31, 2007.

[89] J-C Laprie, Jean Arlat, Christian Beounes, and Karama Kanoun. Definition and analysis of hardware-and software-fault-tolerant architectures. *Computer*, 23(7): 39–51, 1990.

[90] Michael R Lyu. Software reliability engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*, pages 153–170. IEEE, 2007.

[91] Amazon compute service level agreement, 2019. URL `https://aws.amazon.com/compute/sla/`.

[92] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005.

[93] Keqiu Li, Hong Shen, Keishi Tajima, and Liusheng Huang. An effective cache replacement algorithm in transcoding-enabled proxies. *The Journal of Supercomputing*, 35(2):165–184, 2006.

[94] Brian Randell. Recovery blocks. *Encyclopedia of Software Engineering*, 2002.

[95] Algirdas Avizienis. The methodology of n-version programming. *Software fault tolerance*, 3:23–46, 1995.

[96] KH Kim and Howard O. Welch. Distributed execution of recovery blocks: An approach for uniform treatment of hardware and software faults in real-time applications. *IEEE transactions on Computers*, 38(5):626–636, 1989.

[97] Chen-Liang Fang, Deron Liang, Fengyi Lin, and Chien-Cheng Lin. Fault tolerant web services. *Journal of Systems Architecture*, 53(1):21–38, 2007.

[98] Nik Looker, Malcolm Munro, and Jie Xu. Increasing web service dependability through consensus voting. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, volume 2, pages 66–69. IEEE, 2005.

[99] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Francesco Lo Presti, and Raffaela Mirandola. Qos-driven runtime adaptation of service oriented architectures. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 131–140. ACM, 2009.

[100] Nicolas Salatge and Jean-Charles Fabre. Fault tolerance connectors for unreliable web services. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 51–60. IEEE, 2007.

[101] Guang-Way Sheu, Yue-Shan Chang, Deron Liang, Shyan-Ming Yuan, and Winston Lo. A fault-tolerant object service on corba. In *Proceedings of 17th International Conference on Distributed Computing Systems*, pages 393–400. IEEE, 1997.

[102] Xi Chen, Xudong Liu, Zicheng Huang, and Hailong Sun. Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation. In *2010 IEEE international conference on web services*, pages 9–16. IEEE, 2010.

[103] Giuliana Teixeira Santos, Lau Cheuk Lung, and Carlos Montez. Ftweb: A fault tolerant infrastructure for web services. In *Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05)*, pages 95–105. IEEE, 2005.

[104] Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris. Ws-replication: a framework for highly available web services. In *Proceedings of the 15th international conference on World Wide Web*, pages 357–366. ACM, 2006.

[105] Sajeeva L Pallemulle, Haraldur D Thorvaldsson, and Kenneth J Goldman. Byzantine fault-tolerant web services for n-tier and service oriented architectures. In *2008 The 28th International Conference on Distributed Computing Systems*, pages 260–268. IEEE, 2008.

[106] N Ertugrul, W Soong, G Dostal, and D Saxon. Fault tolerant motor drive system with redundancy for critical applications. In *2002 IEEE 33rd Annual IEEE Power Electronics Specialists Conference. Proceedings (Cat. No. 02CH37289)*, volume 3, pages 1457–1462. IEEE, 2002.

[107] Yongjian Gong, Xiaoyan Su, Hong Qian, and Ning Yang. Research on fault diagnosis methods for the reactor coolant system of nuclear power plant based on ds evidence theory. *Annals of Nuclear Energy*, 112:395–399, 2018.

[108] U Narayan Bhat and Gregory K Miller. *Elements of applied stochastic processes*, volume 3. Wiley-Interscience Hobokenˆ eN. JNJ, 2002.

[109] Swapna S Gokhale. Software reliability analysis incorporating second-order architectural statistics. *International Journal of Reliability, Quality and Safety Engineering*, 12(03):267–290, 2005.

[110] James L Beck and Siu-Kui Au. Bayesian updating of structural models and reliability using markov chain monte carlo simulation. *Journal of engineering mechanics*, 128(4):380–391, 2002.

[111] Wen-Li Wang, Ye Wu, and Mei-Hwa Chen. An architecture-based software reliability model. In *Proceedings 1999 Pacific Rim International Symposium on Dependable Computing*, pages 143–150. IEEE, 1999.

[112] Yuan-Shun Dai, Yi Pan, and Xukai Zou. A hierarchical modeling and analysis for grid service reliability. *IEEE Transactions on Computers*, 56(5):681–691, 2007.

[113] Khosrow Moslehi, Ranjit Kumar, et al. A reliability perspective of the smart grid. *IEEE Trans. Smart Grid*, 1(1):57–64, 2010.

[114] Katerina Goševa-Popstojanova and Kishor S Trivedi. Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45 (2-3):179–204, 2001.

[115] Sherif Yacoub, Bojan Cukic, and Hany H Ammar. A scenario-based reliability analysis approach for component-based software. *IEEE transactions on reliability*, 53(4):465–480, 2004.

[116] Jun Zhao, Haitao Zheng, and Guang-Hua Yang. Distributed coordination in dynamic spectrum allocation networks. In *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005.*, pages 259–268. IEEE, 2005.

[117] Christopher Dabrowski. Reliability in grid computing systems. *Concurrency and Computation: Practice and Experience*, 21(8):927–959, 2009.

[118] Keyur K Patel, Sunil M Patel, et al. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *International journal of engineering science and computing*, 6(5), 2016.

[119] Zibin Zheng and Michael R Lyu. Collaborative reliability prediction of service-oriented systems. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 35–44. IEEE, 2010.

[120] Dong-Hee Shin. User centric cloud service model in public sectors: Policy implications of cloud services. *Government Information Quarterly*, 30(2):194–203, 2013.

[121] Delnavaz Mobedpour, Chen Ding, and Chi-Hung Chi. A qos query language for user-centric web service selection. In *2010 IEEE International Conference on Services Computing*, pages 273–280. IEEE, 2010.

[122] Mohd Ehmer Khan, Farmeena Khan, et al. A comparative study of white box, black box and grey box testing techniques. *Int. J. Adv. Comput. Sci. Appl*, 3(6), 2012.

[123] Ayman Amin, Alan Colman, and Lars Grunske. An approach to forecasting qos attributes of web services based on arima and garch models. In *2012 IEEE 19th International Conference on Web Services*, pages 74–81. IEEE, 2012.

[124] Zhi Zhong Liu, Zong Pu Jia, Xiao Xue, and Ji Yu An. Reliable web service composition based on qos dynamic prediction. *Soft Computing*, 19(5):1409–1425, 2015.

[125] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[126] Jef Hooyberghs, Clemens Mensink, Gerwin Dumont, Frans Fierens, and Olivier Brasseur. A neural network forecast for daily average pm10 concentrations in belgium. *Atmospheric Environment*, 39(18):3279–3289, 2005.

[127] Lon-Mu Liu, Gregory B Hudak, George EP Box, Mervin E Muller, and George C Tiao. *Forecasting and time series analysis using the SCA statistical system*, volume 1. Scientific Computing Associates DeKalb, IL, 1992.

[128] Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. Distance measures for effective clustering of arima time-series. In *Proceedings 2001 IEEE international conference on data mining*, pages 273–280. IEEE, 2001.

[129] Mascha Van Der Voort, Mark Dougherty, and Susan Watson. Combining kohonen maps with arima time series models to forecast traffic flow. *Transportation Research Part C: Emerging Technologies*, 4(5):307–318, 1996.

[130] Durdu Ömer Faruk. A hybrid neural network and arima model for water quality time series prediction. *Engineering Applications of Artificial Intelligence*, 23(4): 586–594, 2010.

[131] José D Bermúdez, Jose V Segura, and Enriqueta Vercher. Holt–winters forecasting: an alternative formulation applied to uk air passenger data. *Journal of Applied Statistics*, 34(9):1075–1090, 2007.

[132] James W Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, 54(8):799–805, 2003.

[133] Apostolos Kotsialos, Markos Papageorgiou, and Antonios Poulimenos. Long-term sales forecasting using holt–winters and neural network methods. *Journal of Forecasting*, 24(5):353–368, 2005.

[134] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[135] N Ganesan, K Venkatesh, MA Rama, and A Malathi Palani. Application of neural networks in diagnosing cancer disease using demographic data. 2010.

[136] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

[137] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2): 240–254, Mar 1994. ISSN 1045-9227. doi: 10.1109/72.279188.

[138] Sandhya Samarasinghe. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. CRC Press, 2016.

[139] Fi-John Chang, Pin-An Chen, Ying-Ray Lu, Eric Huang, and Kai-Yao Chang. Real-time multi-step-ahead water level forecasting by recurrent neural networks for urban flood control. *Journal of Hydrology*, 517:836 – 846, 2014. ISSN 0022-1694. doi: http://dx.doi.org/10.1016/j.jhydrol.2014.06.013. URL `http://www.sciencedirect.com/science/article/pii/S0022169414004739`.

[140] Bogdan Oancea and Stefan Cristian Ciucu. Time series forecasting using neural networks. *CoRR*, abs/1401.1333, 2014. URL `http://arxiv.org/abs/1401.1333`.

[141] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[142] Paul J Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[143] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *ArXiv e-prints*, December 2014.

[144] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[145] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL `http://arxiv.org/abs/1409.1259`.

[146] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

[147] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[148] André Gensler, Janosch Henze, Bernhard Sick, and Nils Raabe. Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks. In *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 002858–002865. IEEE, 2016.

[149] Shikhar Srivastava and Stefan Lessmann. A comparative study of lstm neural networks in forecasting day-ahead global horizontal irradiance with satellite data. *Solar Energy*, 162:232–247, 2018.

[150] Hui Liu, Xiwei Mi, and Yanfei Li. Smart multi-step deep learning model for wind speed forecasting based on variational mode decomposition, singular spectrum analysis, lstm network and elm. *Energy Conversion and Management*, 159:54–64, 2018.

[151] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

[152] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 130–139. IEEE, 2016.

[153] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[154] Mohamed Abdel-Nasser and Karar Mahmoud. Accurate photovoltaic power forecasting models using deep lstm-rnn. *Neural Computing and Applications*, pages 1–14, 2017.

[155] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002. ISSN 1573-1391.

[156] Lingshuang Shao, Jing Zhang, Yong Wei, Junfeng Zhao, Bing Xie, and Hong Mei. Personalized qos prediction forweb services via collaborative filtering. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 439–446. IEEE, 2007.

[157] Zhen Chen, Limin Shen, and Feng Li. Exploiting web service geographical neighborhood for collaborative qos prediction. *Future Generation Computer Systems*, 68:248–259, 2017.

[158] Kai Su, Bin Xiao, Baoping Liu, Huaiqiang Zhang, and Zongsheng Zhang. Tap: A personalized trust-aware qos prediction approach for web service recommendation. *Knowledge-Based Systems*, 115:55–65, 2017.

[159] Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Nips*, volume 1, pages 2–1, 2007.

[160] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[161] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu. An extended matrix factorization approach for qos prediction in service selection. In *2012 IEEE Ninth International Conference on Services Computing*, pages 162–169, June 2012. doi: 10.1109/SCC.2012.36.

[162] Z. Zheng et al. Collaborative web service qos prediction via neighborhood integrated matrix factorization. *IEEE Transactions on Services Computing*, 6(3):289–299, July 2013. ISSN 1939-1374.

[163] D. Yu, Y. Liu, Y. Xu, and Y. Yin. Personalized qos prediction for web services using latent factor models. In *2014 IEEE International Conference on Services Computing*, pages 107–114, June 2014. doi: 10.1109/SCC.2014.23.

[164] Pinjia He, Jieming Zhu, Jianlong Xu, and Michael R Lyu. A hierarchical matrix factorization approach for location-based web service qos prediction. In *2014 IEEE 8th International Symposium on Service Oriented System Engineering*, pages 290–295. IEEE, 2014.

[165] Wei Lo, Jianwei Yin, Ying Li, and Zhaohui Wu. Efficient web service qos prediction using local neighborhood matrix factorization. *Engineering Applications of Artificial Intelligence*, 38:14–23, 2015.

[166] Mingdong Tang, Zibin Zheng, Guosheng Kang, Jianxun Liu, Yatao Yang, and Tingting Zhang. Collaborative web service quality prediction via exploiting matrix factorization and network map. *IEEE Transactions on Network and Service Management*, 13(1):126–137, 2016.

[167] Kai Su, Liangli Ma, Bin Xiao, and Huaiqiang Zhang. Web service qos prediction by neighbor information combined non-negative matrix factorization. *Journal of Intelligent & Fuzzy Systems*, 30(6):3593–3604, 2016.

[168] Hao Wu, Kun Yue, Bo Li, Binbin Zhang, and Ching-Hsien Hsu. Collaborative qos prediction with context-sensitive matrix factorization. *Future Generation Computer Systems*, 82:669–678, 2018.

[169] Lixing Li, Zhi Jin, Ge Li, Liwei Zheng, and Qiang Wei. Modeling and analyzing the reliability and cost of service composition in the iot: A probabilistic approach. In *2012 IEEE 19th International Conference on Web Services*, pages 584–591. IEEE, 2012.

[170] Shigeru Yamada, Hiroshi Ohtera, and Mitsuru Ohba. Testing-domain dependent software reliability models. *Computers & Mathematics with Applications*, 24 (1-2):79–86, 1992.

[171] Donald L Phillips, Sandra L Brown, Paul E Schroeder, and Richard A Birdsey. Toward error analysis of large-scale forest carbon budgets. *Global Ecology and Biogeography*, 9(4):305–313, 2000.

[172] Ali Abdulshahed, Andrew P Longstaff, Simon Fletcher, and Alan Myers. Application of gnnmci (1, n) to environmental thermal error modelling of cnc machine tools. KTH Royal Institute of Technology, 2013.

[173] T Treib and E Matthias. Error budgeting—applied to the calculation and optimization of the volumetric error field of multiaxis systems. *CIRP annals*, 36(1): 365–368, 1987.

[174] Ayman Habib, Ki In Bang, Ana Paula Kersting, and Dong-Cheon Lee. Error budget of lidar systems and quality control of the derived data. *Photogrammetric Engineering & Remote Sensing*, 75(9):1093–1108, 2009.

[175] MA Nelson, TFA Bishop, J Triantafilis, and IOA Odeh. An error budget for different sources of error in digital soil mapping. *European Journal of Soil Science*, 62(3):417–430, 2011.

[176] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site Reliability Engineering: How Google Runs Production Systems.* " O'Reilly Media, Inc.", 2016.

[177] Palanivel A Kodeswaran, Ravi Kokku, Sayandeep Sen, and Mudhakar Srivatsa. Idea: A system for efficient failure management in smart iot environments. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 43–56. ACM, 2016.

[178] Antonio Celesti, Lorenzo Carnevale, Antonino Galletta, Maria Fazio, and Massimo Villari. A watchdog service making container-based micro-services reliable in iot clouds. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 372–378. IEEE, 2017.

[179] Gary White, Andrei Palade, Christian Cabrera, and Siobhán Clarke. IoTPredict: collaborative QoS prediction in IoT. In *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom) (PerCom 2018)*, Athens, Greece, March 2018.

[180] G. White, A. Palade, and S. Clarke. Forecasting qos attributes using lstm networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2018. doi: 10.1109/IJCNN.2018.8489052.

[181] Bice Cavallo, Massimiliano Di Penta, and Gerardo Canfora. An empirical comparison of methods to support qos-aware service selection. In *Proceedings of the 2Nd International Workshop on Principles of Engineering Service-Oriented Systems*, PESOS '10, pages 64–70, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-963-3. doi: 10.1145/1808885.1808899. URL `http://doi.acm.org/10.1145/1808885.1808899`.

[182] G. White and S. Clarke. Urban intelligence with deep edges. *IEEE Access*, 8: 7518–7530, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2963912.

[183] Mihaela Cardei and Ding-Zhu Du. Improving wireless sensor network lifetime through power aware organization. *Wireless networks*, 11(3):333–340, 2005.

[184] Sheng-Tzong Cheng, Jian-Pei Liu, Jian-Lun Kao, and Chia-Mei Chen. A new framework for mobile web services. In *Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops*, pages 218–222. IEEE, 2002.

[185] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, and Francesco Lo Presti. Flow-based service selection forweb service composition supporting multiple qos classes. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 743–750. IEEE, 2007.

[186] Wolfgang Maass. Liquid state machines: motivation, theory, and applications. In *Computability in context: computation and logic in the real world*, pages 275–296. World Scientific, 2011.

[187] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127 – 149, 2009. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2009.03.005. URL `http://www.sciencedirect.com/science/article/pii/S1574013709000173`.

[188] L. A. Alexandre, M. J. Embrechts, and J. Linton. Benchmarking reservoir computing on time-independent classification tasks. In *2009 International Joint Conference on Neural Networks*, pages 89–93, June 2009. doi: 10.1109/IJCNN.2009.5178920.

[189] Ding Hai-yan, Pei Wen-jiang, and He Zhen-ya. A multiple objective optimization based echo state network tree and application to intrusion detection. In *Proceedings of 2005 IEEE International Workshop on VLSI Design and Video Technology, 2005.*, pages 443–446. IEEE, 2005.

[190] S. I. Han and J. M. Lee. Fuzzy echo state neural networks and funnel dynamic surface control for prescribed performance of a nonlinear dynamic system. *IEEE Transactions on Industrial Electronics*, 61(2):1099–1112, Feb 2014. ISSN 0278-0046. doi: 10.1109/TIE.2013.2253072.

[191] J. Mazumdar and R. G. Harley. Utilization of echo state networks for differentiating source and nonlinear load harmonics in the utility network. *IEEE Transactions on Power Electronics*, 23(6):2738–2745, Nov 2008. ISSN 0885-8993. doi: 10.1109/TPEL.2008.2005097.

[192] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[193] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391 – 403, 2007. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2007.04.003. URL `http://www.sciencedirect.com/science/article/pii/S089360800700038X`. Echo State Networks and Liquid State Machines.

[194] D. Verstraeten, B. Schrauwen, D. Stroobandt, and J. Van Campenhout. Isolated word recognition with the liquid state machine: a case study. *Information Processing Letters*, 95(6):521 – 528, 2005. ISSN 0020-0190. doi: https://doi.org/10.1016/j.ipl.2005.05.019. URL http://www.sciencedirect.com/science/article/pii/S0020019005001523. Applications of Spiking Neural Networks.

[195] Mark D. Skowronski and John G. Harris. Automatic speech recognition using a predictive echo state network classifier. *Neural Networks*, 20(3):414 – 423, 2007. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2007.04.006. URL http://www.sciencedirect.com/science/article/pii/S0893608007000330. Echo State Networks and Liquid State Machines.

[196] Horst Bunke and Tamás Varga. *Off-Line Roman Cursive Handwriting Recognition*, pages 165–183. Springer London, London, 2007. ISBN 978-1-84628-726-8. doi: 10.1007/978-1-84628-726-8_8. URL https://doi.org/10.1007/978-1-84628-726-8_8.

[197] Pieter Buteneers, David Verstraeten, Pieter van Mierlo, Tine Wyckhuys, Dirk Stroobandt, Robrecht Raedt, Hans Hallez, and Benjamin Schrauwen. Automatic detection of epileptic seizures on the intra-cranial electroencephalogram of rats using reservoir computing. *Artificial Intelligence in Medicine*, 53(3):215 – 223, 2011. ISSN 0933-3657. doi: https://doi.org/10.1016/j.artmed.2011.08.006. URL http://www.sciencedirect.com/science/article/pii/S0933365711001175.

[198] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[199] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen. Memory versus non-linearity in reservoirs. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010. doi: 10.1109/IJCNN.2010.5596492.

[200] T. Strauss, W. Wustlich, and R. Labahn. Design strategies for weight matrices of echo state networks. *Neural Computation*, 24(12):3246–3276, Dec 2012. ISSN 0899-7667. doi: 10.1162/NECO_a_00374.

[201] Izzet B. Yildiz, Herbert Jaeger, and Stefan J. Kiebel. Re-visiting the echo state property. *Neural Networks*, 35:1 – 9, 2012. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2012.07.005. URL http://www.sciencedirect.com/science/article/pii/S0893608012001852.

[202] Mantas Lukoševicius, Dan Popovici, Herbert Jaeger, Udo Siewert, and Residence Park. Time warping invariant echo state networks. *International University Bremen, Tech. Rep*, 2006.

[203] Benjamin Schrauwen, Jeroen Defour, David Verstraeten, and Jan Van Campenhout. The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In Joaquim Marques de Sá, Luís A. Alexandre, Włodzisław Duch, and Danilo Mandic, editors, *Artificial Neural Networks – ICANN 2007*, pages 471–479, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74690-4.

[204] Herbert Jaeger. Long short-term memory in echo state networks: Details of a simulation study, 2012.

[205] Filippo Maria Bianchi, Lorenzo Livi, and Cesare Alippi. Investigating echo-state networks dynamics by means of recurrence analysis. *IEEE transactions on neural networks and learning systems*, 29(2):427–439, 2016.

[206] G. White and S. Clarke. Short-term qos forecasting at the edge for reliable service applications. *IEEE Transactions on Services Computing*, pages 1–1, 2020.

[207] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. 2010.

[208] Nanxi Chen, Nicolás Cardozo, and Siobhán Clarke. Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, 11(1):49–62, 2016.

[209] N. Chen, N. Cardozo, and S. Clarke. Goal-driven service composition in mobile and pervasive computing. *IEEE Transactions on Services Computing*, PP(99): 1–1, 2016. ISSN 1939-1374. doi: 10.1109/TSC.2016.2533348.

[210] Nebil Ben Mabrouk, Sandrine Beauche, Elena Kuznetsova, Nikolaos Georgantas, and Valérie Issarny. Qos-aware service composition in dynamic service oriented environments. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 123–142. Springer, 2009.

[211] M. Mirahsan, R. Schoenen, and H. Yanikomeroglu. Hethetnets: Heterogeneous traffic distribution in heterogeneous wireless cellular networks. *IEEE Journal on Selected Areas in Communications*, 33(10):2252–2265, 2015. ISSN 0733-8716.

[212] Maurice George Kendall. Rank correlation methods. 1948.

[213] Y. Zhang, Z. Zheng, and M. R. Lyu. Exploring latent features for memory-based qos prediction in cloud computing. In *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, pages 1–10, Oct 2011. doi: 10.1109/SRDS.2011.10.

[214] Zibin Zheng and Michael R. Lyu. Personalized reliability prediction of web services. *ACM Trans. Softw. Eng. Methodol.*, 22(2):12:1–12:25, March 2013. ISSN 1049-331X. doi: 10.1145/2430545.2430548. URL `http://doi.acm.org/10.1145/2430545.2430548`.

[215] Olgierd Hryniewicz and Janusz Karpiński. Prediction of reliability–the pitfalls of using pearson's correlation. *Eksploatacja i Niezawodność*, 16, 2014.

[216] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[217] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[218] Matthias Scholz, Martin Fraunholz, and Joachim Selbig. Nonlinear principal component analysis: neural network models and applications. In *Principal manifolds for data visualization and dimension reduction*, pages 44–67. Springer, 2008.

[219] Dangwei Li, Xiaotang Chen, Zhang Zhang, and Kaiqi Huang. Learning deep context-aware features over body and latent parts for person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 384–393, 2017.

[220] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, page 4. ACM, 2014.

[221] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of the conference on empirical methods in natural language processing*, pages 151–161. Association for Computational Linguistics, 2011.

[222] Lukun Wang. Recognition of human activities using continuous autoencoders with wearable sensors. *Sensors*, 16(2):189, 2016.

[223] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, 2008. URL `http://doi.acm.org/10.1145/1390156.1390294`.

[224] Pascal Vincent et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1756006.1953039`.

[225] Sanjeev Arora, Rong Ge, Ankur Moitra, and Sushant Sachdeva. Provable ica with unknown gaussian noise, with implications for gaussian mixtures and autoencoders. In *Advances in Neural Information Processing Systems*, pages 2375–2383, 2012.

[226] Nitish Srivastava et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

[227] J. Xu, L. Xiang, Q. Liu, H. Gilmore, J. Wu, J. Tang, and A. Madabhushi. Stacked sparse autoencoder (ssae) for nuclei detection on breast cancer histopathology images. *IEEE Transactions on Medical Imaging*, 35(1):119–130, Jan 2016. ISSN 0278-0062. doi: 10.1109/TMI.2015.2458702.

[228] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[229] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *CoRR*, abs/1605.07678, 2016. URL `http://arxiv.org/abs/1605.07678`.

[230] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[231] F. Li and S. Clarke. A context-based strategy for sla negotiation in the iot environment. In *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 208–213, March 2019. doi: 10.1109/PERCOMW.2019.8730752.

[232] Christian Cabrera, Andrei Palade, Gary White, and Siobhán Clarke. Services in iot: a service planning model based on consumer feedback. In *International Conference on Service-Oriented Computing*, pages 304–313. Springer, 2018.

[233] Fan Li. Service negotiation in a dynamic iot environment. In *Service-Oriented Computing – ICSOC 2018 Workshops*, pages 379–386, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17642-6.

[234] Hongbing Wang, Mingzhu Gu, Qi Yu, Huanhuan Fei, Jiajie Li, and Yong Tao. Large-scale and adaptive service composition using deep reinforcement learning. In *International Conference on Service-Oriented Computing*, pages 383–391. Springer, 2017.

[235] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *arXiv preprint arXiv:1711.11157*, 2017.

[236] Gaurav Goswami, Romil Bhardwaj, Richa Singh, and Mayank Vatsa. Mdlface: Memorability augmented deep learning for video face recognition. In *Biometrics (IJCB), 2014 IEEE International Joint Conference on*, pages 1–7. IEEE, 2014.

[237] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[238] Sofoklis Kyriazakos, Ramjee Prasad, Albena Mihovska, Aristodemos Pnevmatikakis, Harm op den Akker, Hermie Hermens, Paolo Barone, Alessandro Mamelli, Samuele De Domenico, Matthias Pocs, et al. ewall: An open-source cloud-based ehealth platform for creating home caring environments for older adults living with chronic diseases or frailty. *Wireless Personal Communications*, 97(2):1835–1875, 2017.

[239] Mahadev Satyanarayanan. The emergence of edge computing. *Computer*, 50(1): 30–39, 2017.

[240] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. Privacy mediators: Helping iot cross the chasm. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications*, HotMobile '16, pages 39–44, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4145-5. doi: 10.1145/2873587.2873600. URL `http://doi.acm.org/10.1145/2873587.2873600`.

[241] Qiao Yan, F Richard Yu, Qingxiang Gong, and Jianqiang Li. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE communications surveys & tutorials*, 18(1):602–622, 2015.

[242] Samuel Gibbs. Typo blamed for amazon's internet-crippling outage. `https://www.theguardian.com/technology/2017/mar/03/typo-blamed-amazon-web-services-internet-outage`. Accessed: May 2020.

[243] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[244] Raspberry pi 3 model b. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`. Accessed: May 2020.

[245] Arduino wemos d1. `https://wiki.wemos.cc/products:d1:d1`, . Accessed: May 2020.

[246] Cisco linksys ea6400. `https://www.linksys.com/us/p/P-EA6400/`. Accessed: May 2020.

[247] Apache jmeter. `http://jmeter.apache.org`. Accessed: May 2020.

[248] Arduino homepage. `https://www.arduino.cc/`, . Accessed: May 2020.

[249] Arduino libraries. `https://github.com/esp8266/Arduino/tree/master/libraries`, . Accessed: May 2020.

[250] Jetson developer kit user guide. `https://elinux.org/Jetson_TX2`. Accessed: May 2020.

[251] David Harvey, Stephen Leybourne, and Paul Newbold. Testing the equality of prediction mean squared errors. *International Journal of forecasting*, 13(2): 281–291, 1997.

[252] Zhaohua Wu, Norden E Huang, Steven R Long, and Chung-Kang Peng. On the trend, detrending, and variability of nonlinear and nonstationary time series. *Proceedings of the National Academy of Sciences*, 104(38):14889–14894, 2007.

[253] Joeran Beel and Stefan Langer. A comparison of offline evaluations, online evaluations, and user studies in the context of research-paper recommender systems. In *Research and Advanced Technology for Digital Libraries*, pages 153–168, Cham, 2015. ISBN 978-3-319-24592-8.

[254] Timothy C Urdan. *Statistics in plain English*. Routledge, 2011.

[255] Jacob Cohen. *Statistical power analysis for the behavioral sciences*. Routledge, 2013.

[256] Mehran Anvari, Craig McKinley, and Harvey Stein. Establishment of the world's first telerobotic remote surgical service: for provision of advanced laparoscopic surgery in a rural community. *Annals of surgery*, 241(3):460, 2005.

[257] Matthew McNaughton, Christopher R Baker, Tugrul Galatali, Bryan Salesky, Christopher Urmson, and Jason Ziglar. Software infrastructure for an autonomous ground vehicle. *Journal of Aerospace Computing, Information, and Communication*, 5(12):491–505, 2008.

[258] Jens Grubert, Tobias Langlotz, Stefanie Zollmann, and Holger Regenbrecht. Towards pervasive augmented reality: Context-awareness in augmented reality. *IEEE transactions on visualization and computer graphics*, 23(6):1706–1724, 2016.

[259] Gary White, Christian Cabrera, Andrei Palade, and Siobhán Clarke. Augmented reality in iot. In *Service-Oriented Computing – ICSOC 2018 Workshops*, pages 149–160, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17642-6.

[260] Emil Chiauzzi, Carlos Rodarte, and Pronabesh DasMahapatra. Patient-centered activity monitoring in the self-management of chronic health conditions. *BMC medicine*, 13(1):77, 2015.

[261] Meredith A Barrett, Olivier Humblet, Robert A Hiatt, and Nancy E Adler. Big data and disease prevention: from quantified self to quantified communities. *Big data*, 1(3):168–175, 2013.

[262] Cathal Gurrin, Alan F Smeaton, Aiden R Doherty, et al. Lifelogging: Personal big data. *Foundations and Trends in information retrieval*, 8(1):1–125, 2014.

[263] Leroy Hood and Mauricio Flores. A personal view on systems medicine and the emergence of proactive p4 medicine: predictive, preventive, personalized and participatory. *New biotechnology*, 2012.

[264] Gary White, Zilu Liang, and Siobhan Clarke. A quantified-self framework for exploring and enhancing personal productivity. In *Conference on Content-based Multimedia Indexing and Applications (CBMI) Dublin, Ireland*, 2019.

[265] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[266] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[267] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014.

[268] Ping Liu, Huaqing Li, Xiangguang Dai, and Qi Han. Distributed primal-dual optimisation method with uncoordinated time-varying step-sizes. *International Journal of Systems Science*, 49(6):1256–1272, 2018.

[269] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[270] Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Self-taught clustering. In *Proceedings of the 25th international conference on Machine learning*, pages 200–207. ACM, 2008.

[271] Zheng Wang, Yangqiu Song, and Changshui Zhang. Transferred dimensionality reduction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 550–565. Springer, 2008.

[272] Marcus D Bloice, Christof Stocker, and Andreas Holzinger. Augmentor: an image augmentation library for machine learning. *arXiv preprint arXiv:1708.04680*, 2017.