IoT and Cloud Service Centric Framework for Enablement of Smart Cities

A Thesis

submitted in partial fulfilment of the requirements for the award of the degree of

Doctor of Philosophy

in

Computer Science and Engineering

Submitted by:

Maggi Bansal
(Registration No: 951503013)

Under the guidance of

Dr. Inderveer Chana
Professor, CSED
Dean (Student Affairs), T.I.E.T, Patiala

Dr. Siobhán Clarke
Professor
School of Computer Science and Statistics
Trinity College Dublin, Ireland

Thapar Institute of Engineering and Technology
Patiala-147004, Punjab, India

July 2024

Certificate

I hereby certify that the work, which is being presented in the thesis titled ***"IoT and Cloud Service Centric Framework for Enablement of Smart Cities"***, in partial fulfillment of the requirements for the award of the degree of ***Doctor of Philosophy*** submitted to Computer Science and Engineering Department of ***Thapar Institute of Engineering and Technology, Patiala*** is an authentic record of my own work carried out under the supervision of Dr. Inderveer Chana and Dr. Siobhán Clarke. I have cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

The matter presented in this thesis has not been submitted either in-part or full to any other University/Institute for the award of any other degree.

(Maggi Bansal)
Registration No. 951503013

This is to certify that the above statements made by the candidate are correct and true to the best of my knowledge.

Verified by:

(Dr. Inderveer Chana)
Supervisor
Professor, Computer Science and Engineering Department
Thapar Institute of Engineering and Technology, Patiala, Punjab, India

(Dr. Siobhán Clarke)
Supervisor
Professor, School of Computer Science and Statistics
Trinity College Dublin, Dublin2, Ireland

Dedicated to Shri Radha Krishna

Acknowledgement

This PhD thesis is the outcome of a challenging journey, upon which God has bestowed upon me wisdom and perseverance to accomplish this work successfully. First and foremost, I am eternally grateful to Shri Krishna for his everlasting blessings, endless grace and support at each step in my life.

I would like to express my sincere gratitude to my esteemed supervisors, Dr. Inderveer Chana, Professor, Thapar Institute of Engineering and Technology, Patiala and Dr. Siobhán Clarke, Professor, School of Computer Science and Statistics, Trinity College Dublin, Ireland for their expert guidance, enthusiastic encouragement, and useful critiques of this research work. Their guidance helped me in all the time of research and writing of this thesis.

I want to express my sincere gratitude to Dr. Inderveer Chana for her constant advice and encouragement at every step of my PhD studies. Her constant support, encouragement and immense knowledge lightened up the way in my darkest times. The time and energy spent on careful and patient proof readings of this manuscript helped me to present this thesis with the desired quality. Her constant pursuit for perfection, guidance and valuable suggestions have guided me at every step of my career. Particularly, I value her respect towards professionalism and the desire to excel higher and higher for accepting me as her student.

Sincere thanks must also goes to Dr. Siobhán Clarke for giving me an opportunity to work under her supervision. I am grateful for her continuous support and immense knowledge. It never felt that we have worked oversees, as she was always willing to answer my questions. Despite the time zone difference, we had several interactions for carrying out this research. Her observations and comments helped me to establish the overall direction of the research and to move forward with investigation in depth. I am indebted for her insightful discussions and enlightening suggestions that assisted me to shape the direction of this research.

I am highly thankful to Dr. Shalini Batra, Head, Computer Science and Engineering Department for her constant motivation and support. I wish to express my gratitude to Dr. Seema Bawa, Dr. Anju Bala, Dr. Amit Kumar for being my Doctoral committee members and advising me at the early stage of my research studies. I am gratefully acknowledge their valuable feedback while ensuring the progress of my research work. I am also thankful to PhD. Coordinator Dr. Sushma Jain and all the faculty and staff

members of Computer Science and Engineering Department for always helping me and and being a source of inspiration.

I would also like to thank Dr. N. Tejo Prakash (Dean of Research and Sponsored Projects) for academic support. My sincere thanks to Dr. Padmakumar Nair, Director, Thapar Institute of Engineering and Technology, Patiala for all the facilities that have been instrumental in creation of a healthy research environment in university.

I wish to further extend my thanks my friends for their constant support. My friends Dr. Ravneet Kaur and Dr. Tsering Yang Zoom deserve a special mention. A big thanks to my dear friends for their endless support and motivation that inspired me to carry out this journey.

This thesis would never have been completed without the unconditional support of my family. My hardworking parents deserve a special mention here. My father S. Bal Krishan Bansal and my mother Smt. Suresh Bansal, who inculcated moral, ethical and religious values in me. I owe a lot to them, who encouraged and helped me at every stage of my personal and academic life, and longed to see this achievement come true. I would always be grateful to them for their patience and compassion in every sphere of life. My mother is my role model and has always been an eternal source inspiration for me. I gratefully acknowledge the patience and love of my brothers, sister-in-laws and nieces that helped me to overcome the difficulties encountered during my life.

Last but not the least, my warmest thanks to my husband Hakikat Rai Goyal, whose unconditional support and confidence in me during all these years is so appreciated. I would extend my deep sense of gratitude to all and one in my in-laws who extended their cooperation to me.

Maggi Bansal

Abstract

Smart City is a new governance model that aims to efficiently manage the civic infrastructure and public services to enhance the city administration for overall benefit of citizens. Smart cities use inter-disciplinary and complementary technologies, such as Internet of Things (IoT), Big Data, Cloud Computing, and Edge Computing to address urbanization challenges like accessible transportation systems, efficient energy supply, insights-based urban planning, and so on. The synergistic integration of these technologies enable IoT-based data-driven intelligent smart city applications like smart traffic systems, smart street lighting, smart governance, and many more to achieve sustainable urban growth.

To enable IoT-based data-driven smart city applications, a comprehensive study of big data management across several IoT applications is undertaken. The analysis of existing literature led to two outcomes – a taxonomy of big data management in IoT and identification and definition of 13 V's challenges for big data in IoT. Further, current status of big data management in smart city frameworks is analyzed. The review of current works reveal that the state-of-the-art research works analyze and manage only the IoT big data, while geospatial data of a city is largely ignored. Since, multi-resolution geospatial datasets of a city represent spatial regions at different managerial levels, hence, using the geospatial data can lead to significant insights at spatial hierarchies for smart governance of the city. Therefore, this research undertakes to propose a cloud-based smart city framework named as, Cloud4IoTCity that considers the integration of city's geospatial datasets with IoT big data for smart governance. The proposed Cloud4IoTCity framework successfully analyzes the IoT big data, which is available as batch data to enable smart governance applications in a request-response model. This proposed framework is enhanced to accommodate the real-time smart city IoT applications that operate in continuous sense-process-actuate control loop by utilizing edge computing layer. In this regard, the optimal placement of application services among the three-tier IoT-edge-cloud architecture to minimize latency and satisfy application's diverse resource requirements is a major challenge, which is addressed in this work. In this regard, a Deep Reinforcement Learning (DRL) model named as, UrbanEnQoSPlace is proposed that solves the optimal service placement problem to minimize the overall latency and energy consumption for a set of smart city applications, while satisfying application's diverse resource requirements in a smart city scenario.

Cloud4IoTCity, a cloud-based framework that performs holistic big data management

(i.e., store, process, analyze, visualize) and spatio-temporal analysis of IoT big data is proposed. The Cloud4IoTCity is designed to consider the fusion of IoT and geospatial data of a smart city and enable spatio-temporal analysis (both, descriptive and predictive) of fused urban data. Cloud4IoTCity comprises of four layers: storage layer, processing layer, service layer, and application layer. At first, the fusion of IoT big data with city's geospatial data is done to build the storage layer of the framework on the cloud object store in the form of Delta Lake tables that are based on the recent Lakehouse storage architecture. Then, processing layer comprises of a customized compute cluster for distributed big spatial data processing of underlying fused data to support upper layer services. The service layer offers various data engineering and data processing services to the upper layer application. The data engineering service include, data fusion to create storage layer, add new data, z-order data layout optimization, time travel to a dataset version. Data processing services include spatio-temporal descriptive and predictive analysis, and map-based visualization services. Finally, the application layer features a cloud-based Software-as-a-Service (SaaS) application that enables end-users i.e., smart city managers to run service layer services on the underlying data. Further, to enable descriptive analysis in the form of spatio-temporal queries, a taxonomy of fifteen spatio-temporal queries is proposed, which defines five spatial, four temporal, and six spatio-temporal queries to understand urban dynamics from space-time dimensions.

Secondly, to validate the proposed framework, a case study on urban traffic analysis for Dublin city is implemented that uses real IoT traffic dataset from Dublin. This IoT dataset comprises of three years (2020-2023) hourly traffic readings from inductive-loop sensors deployed at 500+ sites across the city. This IoT dataset is integrated with Dublin's three geospatial datasets – Dublin postal districts (small areas), Dublin administrative areas (large admin zones) and Dublin city (entire city). Integration of IoT dataset with geospatial datasets of the city enables the spatio-temporal analysis at multiple scales of space-time resolutions. All the service layer services are implemented as PySpark scripts on a customized compute cluster having Delta Lake, Apache Spark and Apache Sedona software. The service layer services are accessible to end-users through the developed cloud-based web application named as Urban Analytics. The framework is validated for multi-resolution spatio-temporal descriptive analysis (query-based retrieval for proposed fifteen spatio-temporal query types) and predictive analysis (site-level hourly or daily traffic prediction). Experimental results show that distributed processing of queries reduces the average query runtime by 15.25% on a cluster with 4 worker nodes and by 28.12% on a cluster with 8 worker nodes as compared to a cluster with 1 worker node. Experimental results show that employing the z-order data layout optimization on data reduces the average query runtime by 34.12% as compared to the non z-ordered layout

of data.

Finally, the second aspect for smart city enablement i.e., edge-based efficient service delivery for real-time IoT applications in a smart city is addressed. Among the various edge computing paradigms, the Multi-access Edge Computing (MEC) has been considered in this research as the telecommunication networks have matured over the years and are seen widely deployed across every city. Further, this research considers federation of multiple MEC providers in a city to enable service placement seamlessly across multiple MEC vendors. To solve the service placement problem for a set of real-time smart city IoT applications in 'Urban IoT - Federated MEC - Cloud' architecture, UrbanEnQoSPlace DRL model is proposed. The proposed model is derived from the Dueling Deep-Q-Network to solve the placement problem by minimizing the overall latency and energy consumption for all applications, while satisfying their IoT requirements (sensors/actuators type and invoke-rate), resource requirement (cpu, memory) and per-flow latency, bandwidth requirements. Further, to enable the satisfaction of IoT locality constraints (i.e., match sensors, actuators capabilities), a novel model policy named as, ϵ-greedy with mask is proposed. The proposed policy adds a boolean mask on the standard ϵ-greedy policy, such that the valid placement nodes (model actions) for a sensing/actuating service are set to true, while others are set to false. This policy satisfies the IoT constraints by masking invalid placement nodes, which enables UrbanEnQoSPlace to converge faster. Experiments are conducted to validate the proposed UrbanEnQoSPlace DRL model against state-of-the-art DRL algorithms. Experimental results for convergence analysis, reward analysis show the superior performance of UrbanEnQoSPlace. Further, UrbanEnQoSPlace is also validated for scalability by varying i) no. of applications ii) no. of application services and iii) no. of placement nodes. Finally, experiments show that the proposed ϵ-greedy with mask policy results in 96.9% reduction in number of constraints violation as compared to the standard ϵ-greedy policy for solving the placement problem using UrbanEnQoSPlace model.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Urbanization is a global trend, as cities have become major hubs for economic growth, generating more than 80% of the global Gross Domestic Product (GDP). In addition, advanced education, modern healthcare, and better quality of life have always attracted people worldwide to move from rural areas into urban cities. According to United Nations Statistics, more than 50% of the world's population (4.4 billion people) live in cities today. With this urbanization trend, by 2050, more than 2/3rd of the world's population (6 billion people) is expected to live in cities. However, this rapid urbanization places significant pressure on city governments to efficiently manage their civic infrastructure and public services for a large population, such as accessible transportation systems, efficient energy supply, maintaining air and water quality, and inclusive governance for sustainable urban growth.

To enable futuristic sustainable cities, contemporary technologies, such as the Internet of Things (IoT) and Cloud Computing, are essential to provide data-driven intelligent public services and insights-based urban planning, leading to smart cities. The success of smart city applications hinges on the efficient management and analysis of urban big data. Therefore, a framework based on IoT and cloud computing technologies is required to manage (i.e., store, process, analyze, and visualize) smart city big data for efficient service delivery of smart city applications.

This chapter provides an overview on smart cities, its enabling technologies, classification of applications and deployment architectures for smart city applications. It outlines the problem statement and research motivation to undertake research for this thesis. This chapter concludes with thesis objectives, contributions, and its organization.

1.1 Smart Cities: an Overview

Smart cities use Information and Communication Technologies (ICT) to efficiently manage the public services and city administration. Smart cities involve synergistic integration of technology, people, and government, such that the technology enables digitized city infrastructure operated by advanced technologies; the people are social and innovative to create a sustainable learning society; and the government policies encourage technology adoption, use of digitized urban data, and citizen engagement to enable an intelligent and inclusive city [2].

Owing to the promising benefits of the smart city vision, governments across the globe embraced this technical, social, and political innovation. The first smart city in literature is Barcelona [3], and other notable examples include Europen Union (E.U.) cities [4], Santander city of Spain [5], Padova city of Italy [6], 51 Chinese cities [7], 143 cities among Europe and USA [8], 100 cities in India [9] and many others. In addition, I.T. companies also start providing technical smart city solutions, such as IBM's Smarter Planet [10], Cisco's Smart Connected Communities [11], and Microsoft's CityNext [12].

Smart city use cases span all urban life dimensions, including transportation, healthcare, energy, waste management, livable buildings, and environment for overall urban governance. Figure 1.1 shows major smart city application domains defined by [13]. Giffinger et al. [13] evaluate smart city applications from 200 cities and enlists six characteristics to define a smart city namely, smart economy, smart people, smart mobility, smart governance, smart living, and smart environment.

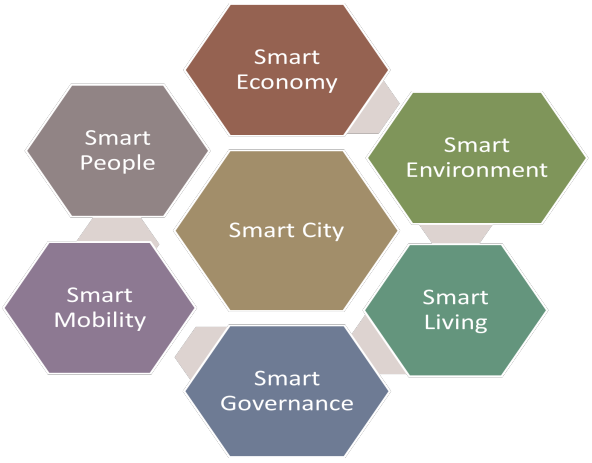


Figure 1.1: Smart City Application Domains

1.1.1 Enabling Technologies for a Smart City

The practical realization of a smart city relies on a number of diverse, yet complementary technologies:

i) **Internet of Things**: The Internet of Things (IoT) [14, 15] is a revolutionary technology that equips everyday things and items with sensing, identification, and wire-

less connectivity features such that the IoT-enabled 'smart things' can be sensed, monitored, and controlled remotely. For enabling a smart city, the use of IoT implies that city infrastructure, such as public street lights, garbage bins, traffic lights, water pipes, and many others, should be equipped with appropriate sensors that generate sensor-based urban data sent to the computing platforms across the internet, where the acquired urban big data is analyzed for provisioning various smart city applications [6].

ii) **Cloud Computing**: According to NIST [16] cloud computing is defined as

> "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, storage, servers, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction".

This cloud model comprises of: i) five essential characteristics – on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service; ii) three service models – Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS); and iii) four deployment models – private cloud, community cloud, public cloud, and hybrid cloud.

Although cloud computing is an independent and mature technology, it is a key enabler to IoT technology. The resource-limited IoT devices depend on cloud computing for offloading the IoT big data and its computation on the resource-rich cloud platforms for provisioning cloud-based IoT services and applications. Jin et al. [1] presents the cloud-centric vision of IoT, wherein cloud performs the IoT data storage, analysis, and management to deliver IoT-based smart applications.

In addition to the cloud computing, novel computing paradigms, such as edge and fog computing [17, 18] have emerged to enable low-latency IoT applications. These computing paradigms work in collaboration with the cloud, such that the network devices in the device-to-cloud continuum like routers, edge servers, and so on are used as intermediate compute nodes, while the cloud is used as the ultimate compute node. Such in-network data pre-processing reduces the overall latency and uplink bandwidth for real-time IoT applications.

iii) **Big Data and Analytics**: Big Data is popularly characterized by its 3 V's - Volume, Velocity, and Variety [19]. Big data comprises large collection of diverse structured, semi-structured, and unstructured data that grows exponentially over time. Smart cities generate heterogeneous big data, which includes IoT data from

numerous sensors deployed across the city, open data, historical data, and geospatial data of the city. The real success of smart city applications hinges on the intelligent analysis and efficient management of urban big data. Therefore, the heterogeneous urban big data should be subject to efficient data storage, analysis and management [20, 21, 22].

In addition to big data technologies, an ubiquitous resource-rich and powerful infrastructure is required to store, process and analyze the big data in a distributed and scalable manner. Hence, cloud computing offers its hardware, software and compute resources for efficient management and analysis of big data on the cloud [23]. The cloud-based big data management offers data-to-knowledge extraction services, which comprise the business logic for data-driven intelligent smart city applications.

1.1.2 Smart City Architecture

Jin et al. [1] propose an IoT and cloud-centric architecture for enabling a smart city as shown in Figure 1.2. In the proposed smart city architecture, the bottom layer represents

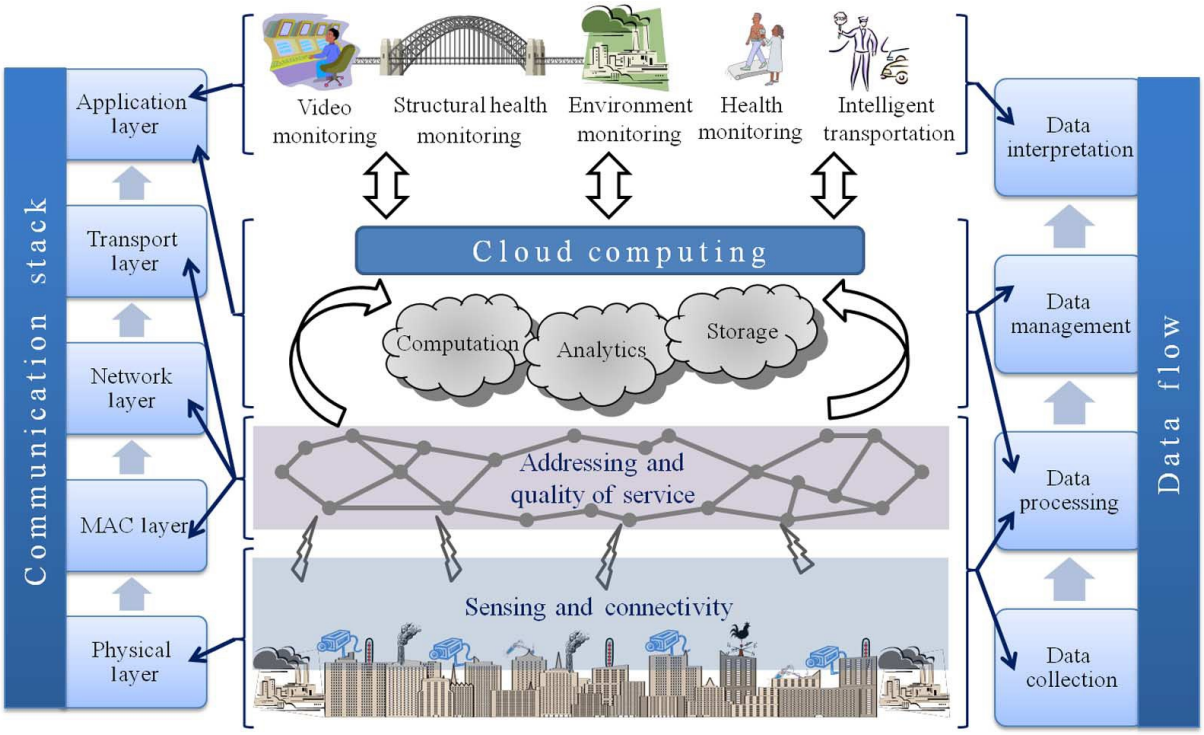


Figure 1.2: IoT and Cloud-Centric Architecture for Smart City [1]

a smart city, which comprises of numerous IoT sensors and devices deployed across the city that generate and transmit IoT big data to the internet, in turn, to an ubiquitous

cloud computing platform. At the cloud computing layer, the received smart city data is subject to big data management and analysis technologies for urban data storage, processing, and analysis. In other words, the cloud platforms enable data-to-knowledge extraction for smart city data in order to deliver knowledge-based intelligent Software-as-a-Service (SaaS) cloud applications. Cloud-based smart city applications span across various application domains, such as smart governance, smart environment, smart transportation, and many more.

1.1.3 Smart City Applications

Based on the application design, the smart city applications can be classified as things-centric or user-centric applications. Among these, the things-centric applications can be further classified as real-time IoT applications or the visualization-based city dashboards. While the user-centric applications can be classified as government-centric or citizen-centric applications, shown in Figure 1.3.

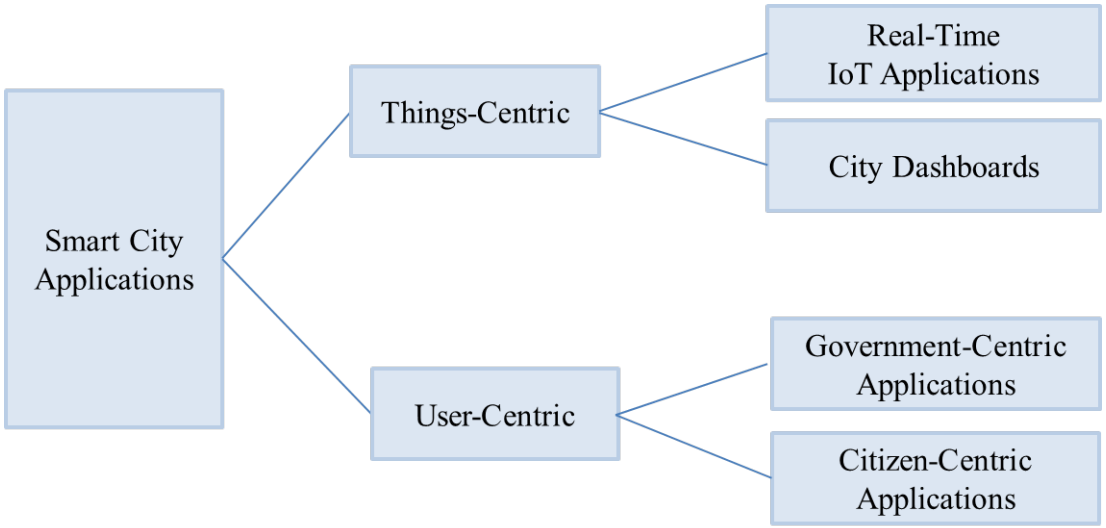


Figure 1.3: Classification of Smart City Applications

i) **Things-Centric Applications:** The smart city applications, which involve real-time data from IoT sensors deployed across the city fall into this category. These applications involve either the real-time smart city IoT applications that use smart actuators or city dashboards that use data visualization to show the city at a glance, defined as follows:

   a) **Real-time IoT Applications:** The smart city applications based on the *sense-process-actuate* model are classified as real-time IoT applications. For example, smart waste management [18], smart street lighting and smart traffic

management [24], and many others.

b) **City Dashboards:** The real-time IoT data, when subject to rich visualizations without any extensive storage, enable the development of city dashboards. These dashboards pull the real-time IoT data from their sources and create interactive plots and visualizations to show the current status of monitored IoT data and surveillance cameras. Examples include, the Dublin dashboard [25], London dashboard [26], Hawaii and New York dashboards [27].

ii) **User-Centric Applications:** The smart city applications that respond to end-user requests in a request-response manner through web or mobile applications fall under this category. These applications involve urban big data analysis of IoT and other historical urban data. These are further classified based on the type of user as government (govt.)-centric or citizen-centric applications, defined as follows:

a) **Government-Centric Applications:** The smart city applications designed for government users like smart city managers, administrative officers, policy makers are categorized as government-centric applications. For example, urban planning [28, 29, 30, 31] applications for smart governance.

b) **Citizen-Centric Applications:** The smart city applications designed for citizens fall under citizen-centric smart city applications category. For example, smart parking [32], smart tourism [33], citizen-centric air quality monitoring [34], and many more.

1.1.4 Application Deployment Architectures

The smart city applications are deployed using the cloud-based architecture or the edge-based architecture [35]. In cloud-based architecture, the resource-rich cloud platform enables the data lifecycle management i.e., storage, processing, analysis, visualization, and results retrieval to deliver smart city applications as cloud-based software-as-a-service (SaaS) applications. For example, Jin et al. [1] analysed the urban IoT data from noise sensors to deliver the urban noise mapping as a cloud-based SaaS application. In addition, the cloud-based architecture offers scalability, elasticity, pay as-per use, and security features favourable for smart city applications. In cloud-based architecture, the smart city application deployment is managed by the Cloud Service Provider (CSP) on its cloud infrastructure.

However, in the cloud-based architecture, the large number of network nodes between sensing devices and the cloud increases the application latency, which degrades the Quality of Service (QoS) for critical real-time smart city applications (like transportation,

healthcare). Therefore, the edge-based architecture [17, 18], is recommended for such applications. In edge-based architecture, the sensor data filtering, pre-processing, aggregation is done by the network edge nodes while forwarding the resource-intensive computation to the distant cloud. This results in reduced latency and reduced uplink bandwidth for the edge-to-cloud backbone network, favourable for real-time applications. In edge-based architecture, the smart city applications are deployed dynamically on the three-tier IoT-edge-cloud architecture to optimize the application QoS (latency, energy, cost, and so on) [36]. Thus, latency-critical real-time smart city IoT applications must be deployed in an edge-based architecture.

1.2 Challenges of Smart City Applications

Smart city applications face numerous challenges [24, 35, 37] discussed as follows:

Challenge 1: Diverse Application Requirements
Smart city applications have diverse application requirements. The data-driven smart city applications rely on big data analysis of IoT big data. Therefore, addressing the data management challenges [24, 35, 37, 38, 39] are the main requirements for smart city applications.

In this context, leveraging the geospatial data along with IoT data is a new promising research direction for smart city applications [40, 41, 42, 43]. Furthermore, integration of IoT and geospatial data makes it possible to analyse urban data from spatial dimension, which can ultimately enable analysis from spatio-temporal dimensions [44].

Challenge 2: Heterogeneity
A smart city comprises heterogeneous data from multi-source, multi-format urban data, such as IoT data from urban sensors and geospatial data. Thus, the smart city applications must consider the integration of heterogeneous urban data viz. IoT big data and the geospatial data [24, 37, 45, 41, 42].

Challenge 3: Dynamic Environment
A city evolves over time and generates new IoT data. Thus, the smart city applications must allow the addition and analysis of recent data.

Challenge 4: Scalability
A smart city comprises of large number of IoT devices that generate urban big data. Thus, the smart city applications must perform scalable storage and processing for urban big data [37, 44].

1.3 Problem Statement

Smart cities are seen as a promising vision by governments across the globe to address urbanization challenges. However, enabling smart city applications involve interdisciplinary technologies and solutions from diverse areas including IoT, cloud providers, network edge operators, data analysts, developers, researchers and many more. The problem statement for enabling smart cities is defined as follows:

- Smart city applications involve use of diverse technologies, such as IoT, big data, and cloud computing integrated in a systematic manner. Therefore, a smart city framework based on these technologies is needed to efficiently manage i.e., store, process, analyze, and visualize the smart city data.
- The historical data from IoT-based smart city applications is available as open data. Moreover, a city is managed at several spatial levels like area, zone, and city, for which geospatial datasets of corresponding spatial regions are also available. However, these multi-source urban big data viz. IoT and geospatial datasets are available in disparate formats. Therefore, systematic integration of IoT big data with city's geospatial datasets is needed in the smart city framework to analyze IoT data with respect to multi-level spatial regions of the city to assist the smart city managers in regional-level smart governance and urban planning.
- The real-time IoT applications in a smart city have strict Quality of Service (QoS) requirements in terms of, latency and bandwidth [6]. To satisfy the QoS requirements for such applications, edge computing paradigms [17, 18] must be used, such that selected application services run on the edge network nodes, while remaining services run on the cloud computing nodes. The success of real-time IoT applications depends on the placement of services among edge computing and cloud computing nodes. Further, a smart city can have multiple real-time IoT applications and multiple edge network providers. Thus, substantial research is needed in the direction of optimal service placement for a set of real-time IoT applications in the multi-tier smart city architecture.

1.4 Research Motivation

Research motivation of this work is to assist smart city governments, cloud vendors, edge operators and research community for enabling a smart city. The research motivation for this work is as follows:

- Success of IoT applications is limited by data related issues [24, 46, 47]. Thus, it motivates the current research in the direction of IoT and big data integration.
- A city is managed and governed at multiple spatial hierarchies, such as area, zone, and city. Therefore, geospatial datasets at multiple spatial resolutions can be integrated with IoT big data for value-added analysis towards better governance and growth of the city [41, 42], which is not considered in current smart city frameworks. Further, this promising data integration will enable analysing city data from spatio-temporal perspective, which is also not addressed yet [44]. These gaps motivate the current research for integrating IoT big data with city's geospatial dataset(s) and to enable spatio-temporal urban data analysis.
- The real-time IoT applications in a smart city use edge computing to satisfy application QoS (latency, bandwidth, cost, and so on) by deploying application services in a multi-tier IoT-edge-cloud architecture [36]. This motivates the current research in the direction of edge computing based service placement in the multi-tier architecture for real-time smart city applications.

This thesis explores the question of how to design an IoT and cloud-based framework for enablement of a smart city. Since, in a smart city both types of applications – cloud-based analytical applications and edge-based real-time IoT applications co-exist. Thus, enabling a smart city has two aspects: efficient big data management on the cloud and efficient service delivery of real-time IoT applications in the edge-cloud collaboration. The specific research questions are as:

RQ.1 How does the fusion of geospatial and IoT big data help the smart city managers to perform spatio-temporal analysis of urban data?

RQ.2 How does edge computing supports efficient service delivery for real-time IoT applications in a smart city?

1.5 Thesis Objectives

The objectives of this thesis are as follows:

i) To explore and analyze existing Cloud computing and IoT techniques for enabling smart city applications.

ii) To propose and design service-oriented cloud framework for urban data infrastructure to perform storage, indexing and retrieval of smart data.

iii) To design on-demand composition of software services for predictive analysis and knowledge extraction of city-wide real time data.

iv) To verify and validate Cloud infrastructure management for efficient service delivery and enablement of sustainable smart city applications like energy, healthcare, transportation etc.

1.6 Thesis Contributions

This thesis contributes to the body of existing knowledge as:

i) **13 V's Challenges of Big Data in IoT:** An extensive literature survey on smart city enabling technologies is carried out. Existing literature on big data in IoT is analyzed to understand the prevalent big data management approaches used in IoT applications. Then, impact of IoT on big data is assessed to identify big data challenges in IoT, which are expressed in the form of V's terminology. The identified 13 V's challenges for big data in IoT are Volume, Velocity, Variety, Veracity, Value, Variability, Visualization, Validity, Vulnerability, Volatility, Venue, Vocabulary, and Vagueness (section 2.1.4).

ii) **Proposed Cloud4IoTCity Framework:** Cloud4IoTCity, a smart city framework for efficient big data management and spatio-temporal analysis is proposed. The proposed framework performs the fusion of IoT big data with the city's geospatial data to enable spatio-temporal analysis. The proposed framework supports both, descriptive analysis (query-based retrieval of past data) and predictive analysis (ML-based forecasting for future time) at multiple space-time scales. The fusion of multi-resolution geospatial datasets enable the spatio-temporal analysis at multiple spatial scales of the city, which is helpful for area/zone/city based analysis of smart city data.

iii) **Proposed Taxonomy of Spatio-Temporal Queries:** A taxonomy of fifteen spatio-temporal queries for smart city applications is proposed. The proposed taxonomy categorizes fifteen queries into three types: five spatial, four temporal, and six spatio-temporal queries. The proposed taxonomy is used in the current research as a baseline list of spatio-temporal queries to perform descriptive analysis of fused urban data in the proposed Cloud4IoTCity framework.

iv) **Proposed UrbanEnQoSPlace Deep Reinforcement Learning (DRL) Model:** UrbanEnQoSPlace, a DRL model for optimally placing the application services of real-time smart city IoT applications in the three-tier 'Urban IoT - Federated Edge

- Cloud' architecture is proposed. The proposed DRL model solves the optimal placement problem for real-time smart city IoT application services with an aim to minimize both, latency and energy consumption for a set of IoT applications, while satisfying their IoT, compute, and communicate resource requirements.

v) **Proposed ϵ-greedy with Mask Policy:** A novel policy for UrbanEnQoSPlace DRL model is proposed to satisfy the IoT constraints while solving the service placement problem. The IoT locality constraints are defined as the type and invoke-rate of sensor/actuator required by each sensing actuation service of an IoT application. The proposed policy produces a boolean-mask that satisfies the IoT-based locality constraints by matching the sensor/actuator requirements of a service with potential placement nodes. The policy produces a boolean mask, such that the valid placement nodes for sensing actuating services are set to true, while the remaining invalid placement nodes are set to false. This boolean mask for an application is used by the standard ϵ-greedy policy to take placement actions as per the mask in the UrbanEnQoSPlace DRL model.

1.7 Thesis Organization

The current Chapter 1 provides an introductory overview of this thesis, and the rest of the thesis is organized as:

Chapter 2 Literature Survey: This chapter presents a survey of smart city enabling technologies i.e., IoT, big data, and cloud computing. The outcomes of this extensive survey are a taxonomy of big data management in IoT, identification of cloud services of major public cloud vendors to support IoT and big data on their platforms, and 13 V's challenges of big data in IoT. Then, this chapter reviews big data management approaches for smart cities and identifies research gaps in big data management for smart city. Further, this chapter examines the role of edge computing to support real-time smart city applications. In this regard, various service placement research works from literature are reviewed. Chapter 2 is partially derived from:

- **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "A Survey on IoT Big Data: Current Status, 13 V's Challenges, and Future Directions", *ACM Computing Surveys (CSUR)*, [**ACM**], vol. 53, no. 6, pp. 1-59, article no. 131, 2020. [SCI Indexed, Impact Factor = 10.282] `https://doi.org/10.1145/3419634` [**Citations = 104**]

- **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "Enablement of IoT based

Context-Aware Smart Home with Fog Computing", *Journal of Cases on Information Technology (JCIT)*, [**IGI Global**], vol. 19, no. 4, pp. 1-12, 2017. [Web of Science ESCI and Scopus Indexed] `https://doi.org/10.4018/JCIT.2017100101` [**Citations = 21**]

- **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "Autonomic Occupancy Detection of an IoT-Based Smart Building Using Deep Neural Network", *2nd International Conference on Machine Intelligence for Research and Innovation (MAITRI 2024)*, Jammu & Kashmir, India, Proc. in [**Springer**] (Springer Nature), 2024. [Scopus Indexed]

Chapter 3 Cloud4IoTCity: Proposed Smart City Framework: This chapter proposes Cloud4IoTCity smart city framework based on the fusion of IoT big data and city's geospatial data to enable spatio-temporal analysis of smart city IoT data. This chapter describes the layered design of the proposed framework, which comprises of four layers namely, storage layer, processing layer, analysis layer, and application layer. In addition, this chapter also proposes a taxonomy of fifteen spatio-temporal queries for smart city applications, which is used for descriptive analysis in the proposed framework. Chapter 3 is partially derived from:

- **Maggi Bansal**, Inderveer Chana, and Siobhán Clarke, "Cloud4IoTCity: A Cloud Framework based on Fusion of Geospatial and IoT Big Data for Efficient Management and Spatio-Temporal Analysis of Smart City Data", *Information Fusion*, [**Elsevier**], [SCI Indexed, Impact Factor = 14.7, Under Review]

Chapter 4 Validation of the Proposed Framework: This chapter performs validation of the proposed framework by implementing it for a case study on Urban Traffic Data Analysis for Dublin city. This chapter describes the layer-wise implementation of the proposed framework for the undertaken case study and showcases map-based visualization results for descriptive and predictive analysis services for the case study. This chapter shows experiment results to validate the framework using query runtime results for the fifteen spatio-temporal queries across different cluster sizes. Chapter 4 is partially derived from:

- **Maggi Bansal**, Inderveer Chana, and Siobhán Clarke, "Cloud4IoTCity: A Cloud Framework based on Fusion of Geospatial and IoT Big Data for Efficient Management and Spatio-Temporal Analysis of Smart City Data", *Information Fusion*, [**Elsevier**], [SCI Indexed, Impact Factor = 14.7, Under Review]

Chapter 5 Efficient Service Delivery for Real-Time Smart City IoT Applications: This chapter proposes UrbanEnQoSPlace, a DRL-based service placement

model for real-time smart city IoT applications. The proposed service placement model is developed from the Dueling Deep Q-Network model. This chapter describes the system model for service placement, followed by proposed model and experimental results. Chapter 5 is partially derived from:

- **Maggi Bansal**, Inderveer Chana, and Siobhán Clarke, "UrbanEnQosPlace: A Deep Reinforcement Learning Model for Service Placement of Real-Time Smart City IoT Applications, *IEEE Transactions on Services Computing*, [**IEEE**], vol. 16, no. 4, pp. 3043-3060, 2023. [SCI Indexed, Impact factor = 5.5] `https://doi.org/10.1109/TSC.2022.3218044` [**Citations = 11**]

Chapter 6 Conclusions and Future Scope: This chapter concludes the research work done and highlights its contributions to the research. This chapter also provides valuable suggestions for future research in this direction.

Chapter 2

Literature Survey

Previous chapter provides an overview of thesis. It discusses fundamental concepts related to Smart Cities. It discusses enabling technologies and challenges for smart city applications. Henceforth, it describes research motivation, lays out research objectives and concludes the chapter with thesis contributions and organization.

This chapter presents finding of an in-depth survey conducted on enabling technologies for a smart city i.e., IoT, big data, and cloud computing. This chapter presents key findings of the undertaken study, which includes, a taxonomy for big data in IoT, cloud services to support IoT and big data, 13 V's challenges of big data in IoT. After surveying the enabling technologies, this chapter reviews the current literature on big data management for smart cities and identifies research gaps in this direction. Finally, the role and research works on edge-computing to support real-time smart city IoT applications are presented.

Chapter organization is as follows: Section 2.1 surveys enabling technologies for a smart city. Then, Section 2.2 reviews state-of-the-art research works on big data management for a smart city. Section 2.3 discusses the support of edge-computing to enable real-time smart city IoT applications.

2.1 Big Data Management in IoT

This section reviews the existing literature on big data management in IoT. The analysis covers research works from various IoT domains, such as smart transport, smart city, smart buildings, and many more.

- **Smart Transport**

  The transport domain offers numerous IoT applications, such as real-time monitoring of a smart car [48], smart traffic light management [49], parking and traffic prediction [50, 51, 52].

  Husni et al. [48] performs real-time monitoring of a smart car using On-Board Diagnostics (OBD) II system. The real-time vehicular data such as speed, engine RPM, fuel level, and so on collected by the OBD system is sent to the driver's smartphone via Bluetooth, from where the data is forwarded to the IBM Bluemix cloud platform for subsequent storage and visualization.

  Tärneberg et al. [49] regulates the red-green signals of the traffic light based on real-time volume of traffic without violating the punctuality schedule of public buses. The IoT data from in-road induction-loop sensors, vehicle GPS, public buses schedule is sent to AWS IoT cloud platform for storage and processing. For data processing both, stream processing using Kinesis and event-processing are applied. Event processing is done using Lambda functions to determine traffic throughput for regulating the smart traffic lights.

  Zheng et al. [50] performs predictive analysis to predict availability of free parking slots based on analysis of parking data available from two cities, Melbourne and San Francisco. The authors find that regression tree gave better prediction accuracy than neural networks and support vector regression

  Lv et al. [51] performed short-term traffic prediction using Stacked Auto-Encoder (SAE) deep learning model. The SAE model gave better prediction accuracy than conventional Machine Learning (ML) models.

  Wan et al. [52] uses Mobile CrowdSensing (MCS) data gathered by drivers, which include real-time traffic data (such as mobileID, speed, location etc.). This acquired data is forwarded to the cloud platform for analysis. The analysis is performed by creating a weighted traffic network and suggesting a shortest re-route to drivers, in case a traffic congestion event is detected.

- **Smart City**

IoT enabled smart cities aim to improve the urban services provided to its citizens. Nevertheless, the scope of data in a smart city is quite broad that comprises of sensor data [29, 53], open data [33], multimedia [54], social media [55], and mobile Crowdsensing (MCS) [56, 57].

Rathore et al. [29] proposes a framework for smart city that includes both, real-time and batch processing for city data. Real-time processing is done using Spark, Storm S4, and VoltDB over Hadoop and batch processing is done using MapReduce. Rathore et al. [53] proposes a smart city framework based on Hadoop ecosystem for capturing and analyzing smart city data. The smart city system uses Spark over Hadoop for real-time analysis and also, includes big graph processing using Giraph over Hadoop for smart vehicular services.

Sun et al. [33] proposes TreSight, a context-aware system that uses data from heterogeneous sources- city sensors, hotspots, wearable bands, city's open data etc. for offering context-aware recommendations for smart tourism in a smart city. Usman et al. [54] focuses on big multimedia data generated in smart city and discusses various machine learning techniques and platforms for managing multimedia data for smart cities.

Farajidavar et al. [55] applies multi-view deep learning for extracting city events from twitter streams. Zarko et al. [56] extends openIoT platform with publish-subscribe module named, CUPUS that enables crowdsensed data acquisition via mobile users in the city. The work was validated with mobile crowdsensing data gathered for air quality monitoring in a smart city. Hromic et al. [57] used openIoT platform for real-time processing of sensor data streams at the edge nodes. The mobile crowdsensed air quality data is analyzed at edge nodes for spatiotemporal correlation analysis using data partitioning and event processing.

Different from aforementioned IP-based smart city frameworks, Piro et al. [58] proposes Information Centric Network (ICN) based framework for smart city services. The ICN network architecture uses content-based addressing, rather than IP-based addressing, which enables receiver-driven data exchange.

- **Smart Buildings**

  IoT applications for smart buildings include, building automation [59, 60, 61, 62], occupants behavioral analysis [63] and energy management in smart buildings [64]. The building automation use cases [59, 60, 61, 62] implemented gateway level architecture to connect the in-house locally connected sensors to the cloud. In particular, Kasmi et al. [60] uses Raspberry Pi integrated with surveillance camera as the gate-

way, while Mainetti et al. [61] uses GSN middleware for protocol translation from 6LoWPAN to IP and as gateway to cloud, and Plageras et al. [62] uses an edge router as gateway to the cloud.

Fog-based smart home platforms include [59, 63]. Bansal et al. [59] proposes ICON, a context-aware smart home framework, in which the smart home data such is processed on the fog and cloud platforms based on user-defined rules and policies. Yassine et al. [63] performs occupant's behavioral analysis using frequent pattern mining on the fog nodes, so as to discover the usage patterns of occupants' for different appliances with respect to time.

Derguech et al. [64] proposes an autonomic approach for predictive analysis that involves dynamic source selection for selecting the source of open data based on prediction error in previous analysis round. The predictive analysis was done using various ML algorithms to predict the power usage for a smart building using the smart meter data obtained from building and weather data from open sources.

- **Smart Living**

  IoT enables sensors to monitor wide range of phenomena such as environmental monitoring [34, 65, 66], daily life activities [67, 68], and healthcare [69, 70] with an aim to improve the well-being and overall quality of life of people. Garcia-de-Prado et al. [34] proposed air4People, an IoT based context-aware system for real-time air quality monitoring. Air4People uses user's current location, activity, nearby places, personal illness, and weather as context parameters to provide personalized context-aware notifications to citizens regarding the deteriorated air quality in their vicinity. Jung et al. [65] proposed a system for air quality monitoring that uses both context-aware and semantic processing techniques. The semantically represented sensor data is subject to spatial-temporal aggregation followed by rule-based processing to offer context-aware alerts for mitigating the air pollution. Trilles et al. [66] proposed a broker-based framework for distributed processing of real-time sensor streams from multiple sources and applied the cumulative sum (CUSUM) algorithm for anomaly detection in environmental monitoring data streams.

  Mulero et al. [67] proposed an elderly behavior monitoring system that uses IoT and linked open data technologies. The proposed system captures personal data such as daily activities, health, nutrition, mobility, memory etc. from elderly people in cities and uses semantic abstractions and linked open data management system to store and query the captured data via semantic query endpoints.

  Yacchirema et al. [68] proposed a fog-cloud based system to detect sleep apnea

and offer recommendations for least polluted areas to perform physical activity. The proposed system collects sleep, activity, and physiological data and performs real-time processing on the fog layer to send notifications to the caregivers upon detection of sleep abnormality, followed by batch processing on the cloud to predict the least polluted areas. Sandha et al. [69] implemented real-time processing of health parameters (heart rate, BP etc.) using Kafka and Spark for predicting the risk of heart attack and calculating the stress index. [70] propose smart health-care monitoring system for elderly using wearable sensors. The proposed system gathers real-time physiological data from wearable sensors and send it to cloud for processing and any disorder in patient's data is send to his doctor.

- **Miscellaneous**

  This section discusses those works from literature that are not domain-specific, but report significant work regarding IoT big data management. These research works are found across various data lifecycle stages: data acquisition [71, 72, 73, 74], data storage [75, 76, 77, 78, 79], processing [80, 81, 82, 83, 84, 85, 86, 87, 88].

  - **Efficient Data Acquisition**

    The research works [71, 72, 73] related to data acquisition focused on achieving energy efficiency while acquiring sensor data from battery-limited IoT devices. For example, Alduais et al. [71] proposed selective forwarding of sensor data from WSN sink node to cloud, only if the difference between current and previous sensor readings exceeds a pre-defined threshold. In this way, the number of transmissions and energy of WSN network is conserved. Dinh et al. [72] conserved the energy of sensor nodes by regulating the sleep cycle of physical sensors and predicting the sensor value from their virtual representation on the cloud. The sensor value prediction was done using internal and external information correlation. Capponi et al. [73] proposed to achieve energy efficiency for opportunistic mobile Crowdsensing scenario. The proposed data collection framework selects users for sending crowdsensed data, in such a way to minimize the energy consumed in data collection and transmission while maximizing the utility of data collection. Simmhan et al. [74] propose SAT-VAM, a low-cost air pollution monitoring system for the cities of Delhi and Mumbai that involves sensor data calibration to predict the variables from raw signals. The data-driven calibration helps to use low-cost sensors in the proposed system.

  - **IoT Data Storage**

    The research works related to IoT data storage improve storage efficiency ei-

ther by addressing the heterogeneity of IoT data [75] or by proposing indexes to ease the search and retrieval of massive IoT data [76, 89, 77] or by hybrid storage and management of real-time and batch IoT data [78]. Jiang et al. [75] proposed a solution combining multiple databases with Hadoop to efficiently store and process structured and unstructured IoT data and validated the system with smart logistics use case. The proposed system features RESTful interfaces and multi-tenant support. Li et al. [76] proposed IoTMDB, a NoSQL-based storage solution for IoT. The IoTMDB system implements two-tier indexing for efficient space-time queries, proposes new query syntaxes for IoT specific data retrieval and uses ontology for data sharing. Gauch et al. [89] proposes a query expansion mechanism to enhance the retrieval efficiency. This work proposes to automatically discover similar words from data in databases and use the extracte tags to increase the search efficiency for multiple databases. Perez et al. [77] proposed servIoTicy, a cloud-based storage platform for IoT data. The servIoTicy uses CouchBase as the baseline data store, and ElasticSearch as the serach and indexing engine to provide scalable data retrieval via REST API. Wu et al. [78] proposed HSFRH-IoT, a framework to store both real-time and batch data. HSFRH-IoT plugged the real-time Redis database into HBase that manages the historical storage. Singh et al. [79] propose a novel system based on electrocardiogram (ECG) to distinguish among the healthy and heart-patients. The proposed system is validated using a dataset and real human subjects.

– **IoT Data Processing**

The research works dealing with IoT data processing are as: Xu et al. [80] proposed an analytics approach for querying time-series databases via a pattern matching query. The proposed work discretized time signals, compresses the discrete segments and builds a two-layer index for fast querying of time-series data. Yang et al. [81] proposed a cloud-based framework for IoT data curation that applies on-cloud data cleaning and compression for IoT data. Peng et al. [82] proposed iCloudFog, a system that integrates cloud and fog computing for IoT data processing. Sharma et al. [83] proposed a system for collaborative IoT data processing among edge and cloud nodes. In the proposed framework, the cloud computing layer guides the edge layer for real-time processing. Tang et al. [84] enabled deep learning CNN model to process data on IoT client by using vector quantization and building a deep learning model using the low-level compute library of ARM microprocessor (ACL). In addition, Lambda Architecture (LA) [85, 86, 87] is a new trend for IoT data process-

ing on the cloud, which uses pre-computed views to perform both batch and stream (speed) analysis on the incoming data and sends the analysis results to a serving layer for quick retrieval using queries. Yamato et al. [85] proposed edge-cloud integrated lambda architecture for predictive maintenance of industrial machines. The speed layer of lambda architecture executes on the edge nodes for anomaly detection and forwards the anomaly data to the cloud for batch processing. The batch layer performs failure detection using rule processing and periodically updates the anomaly detection ML model of the edge layer. Dissanayake et al. [86] proposed lambda-architecture enabled by a multi-agent layer for IoT data processing. The agent layer contains different kinds of agents such as speed agent, batch agent, security agent etc. for forwarding data from IoT devices to the lambda-architecture on cloud. The proposed architecture employs two-layer batch processing along with indexing and replication in the serving layer for low-latency and reliable retrieval of processed data. Diaz et al. [87] proposed the lambda-CoAP architecture for IoT data processing on the cloud. The proposed work executes Lambda Architecture on the cloud for combined stream and batch processing based on pre-computed views. The architecture uses lightweight CoAP middleware that enables it to be deployed in resource constrained devices and the architecture provides interfaces to query the processed data. Iqbal et al. [88] propose a novel data model based on the computational intelligence technique that involves biologically inspired evolutionary spatio-temporal state machine for many use cases of smart city.

2.1.1 Taxonomy of Big Data in IoT

Following the analysis of state-of-the-art literature in the previous section, this section proposes a classification of approaches applied for big data management in various IoT applications with data-to-knowledge value chain. A taxonomy for big data management in IoT is proposed by categorizing existing literature on IoT big data management at four data lifecycle stages: i) Data Acquisition and Communication ii) Data Storage iii) Data Processing and iv) Data Retrieval. The proposed taxonomy of big data management in IoT is shown in Figure 2.1 and is explained below.

i) **Data Acquisition and Communication**

The categorization of IoT data acquisition is done on the basis of two questions:

- How the IoT data is acquired? This includes both, the source of IoT data and the communication protocol used to collect data for IoT applications.

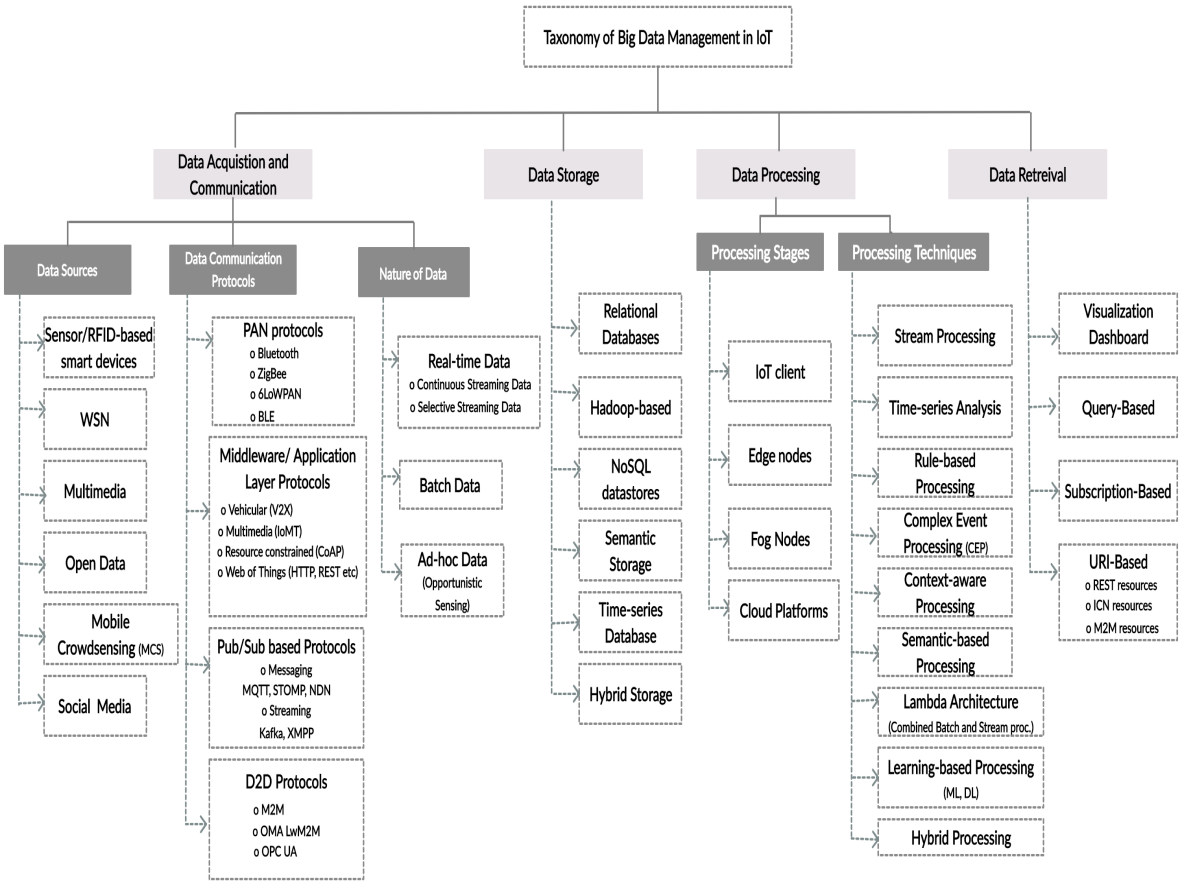


Figure 2.1: Taxonomy for Big Data Management in IoT

- What is the data sensing frequency or pattern?

Thus, data acquisition and communication stage is taxonomized on the basis of three aspects: data sources, data communication protocol, and the nature of data described as follows:

a) **Data Sources:** The most common source for IoT data acquisition includes IoT-enabled smart devices that have embedded sensors or RFID technology for autonomous data sensing. For example, embedded sensors in smart thermostats, lights etc. in smart home [59], RFID-based cargo in smart logistics [75], and smart traffic lights [49] etc. Another sensor-based source of IoT data is WSN that involves the deployment of an array of sensors spread over a sensing region. WSN is used to sense a region using large number of sensors distributed spatially for example smart university [61], environment monitoring [65, 66] etc. Besides, multimedia is an important source of IoT data [33] especially for, surveillance, smart industry, and smart transport use cases. Open data that may include weather data, maps, city information etc. is used as

3rd party external resource to augment data for IoT applications [33, 64, 67]. Mobile Crowdsensing (MCS) is a novel source of IoT data [52, 56, 73] that leverages built-in sensors in the user's smartphone to participate in data acquisition for IoT applications. Yet another important source of IoT data, is social media. Some research works from literature integrate social media data into IoT applications [55, 34].

b) **Data Communication Protocols:** The data generated by IoT devices needs to be communicated to server side (very often, the cloud platform) for further storage and processing of IoT data. In the analyzed IoT applications researchers have used variety of protocols, which can be broadly classified into 4 categories as: i) PAN (Personal Area Network) protocols, which include protocols like Bluetooth, ZigBee, 6LoWPAN (Low power Wireless PAN over IPv6), BLE (Bluetooth Low Energy) etc. for short range low power communication and are mostly used for indoor applications such as smart buildings, examples from literature include [68, 82, 83, 86], ii) Middleware/Application Layer protocols that define application level communication for IoT applications such as Vehicular communication V2X (Vehicle-to-Anything) [52], resource-constrained communication using CoAP (Constrained Application Protocol) [33, 68, 72, 87], multimedia communication using IoMT (Internet of Multimedia Things) [54] or Web of Things (application level communication using existing web standards such as HTTP, REST APIs, social networks, web sockets etc.) [59, 29, 53, 55, 57] and many more, iii) Pub/Sub protocols that allow broker-based loosely coupled asynchronous communication between publisher and subscriber in two modes- a) pub/sub messages by using protocols like MQTT (Message Queuing Telemetry Transport) [49, 63, 66, 77, 85, 90], STOMP (Simple Text Oriented Messaging Protocol) [66, 77], NDN (Named Data Networking) [58], b) as data stream based on pub/sub model by using Kafka [66, 69, 90], or XMPP (Extensible Messaging and Presence Protocol for streaming XML data) [57], iv) Device-to-Device communication, which include protocols like M2M (Machine-to-Machine) [63, 72, 91], OMA LwM2M (Open Mobile Alliance Lightweight M2M) [92], and OPC UA (Open Platform Communications Unified Architecture) [92] that communicate small payload of data transferred directly by machines over any communication channel such as serial communication, Power Line Connection (PLC), or wireless, these protocols are often used in industrial automation to enable IIoT (Industrial IoT).

c) **Nature of Data:** The data sensing pattern is an important parameter related to IoT data acquisition, as it guides the further storage, recovery, and processing of IoT data. The data sensing pattern determines the nature of data and three common types for 'nature of IoT data' identified from literature are as: i) real-time stream data, which is acquired from sensor-based IoT devices, WSN etc. in real-time, either as continuous stream of data or as selective stream of data, ii) batch data, which is usually acquired from open data, social media, 3rd party data sources, iii) ad-hoc data, wherein neither the sender of data nor the sensing frequency is fixed, it involves opportunistic sensing in MCS scenario where the IoT application dynamically recruits the users to participate in data sensing on the basis of their battery and other application requirements. Examples from literature for each type of sensing pattern are as: real-time continuous streaming data [59, 48, 49, 33, 54] and many more, real-time selective streaming data [71, 72], batch data [50, 51, 55] and ad-hoc data [52, 56, 73]. The selective sensing of real-time stream is done for conserving energy during data acquisition, though it might require cloud-based data completion techniques for further storage and processing as per application requirements.

ii) **Data Storage** Common data storage techniques found from IoT literature include, relational database [48, 60, 67], Hadoop-based storage [29, 53, 33, 68], NoSQL (Not Only SQL) datastores [49, 76, 77], semantic storage [56, 57, 64, 65, 90], time-series database [80], and hybrid storage [75, 78, 93].

iii) **Data Processing**

The IoT data processing is categorized on the basis of two questions:

- What is the stage of data processing? It is characterized by the computing infrastructure such as IoT client, edge/fog/cloud computing used to process IoT data
- What technique or approach is used to process IoT data?

Thus, IoT data processing is taxonomized on the basis of two aspects: processing stages and processing techniques as follows:

a) **Processing stages:** The most common computing infrastructure to process IoT data includes the resource-rich remote cloud platform [48, 49, 50, 51, 52] and many more. However, the recent trend is to move processing of IoT data closer to the end-user such as, fog nodes that process data on any of the network nodes in the user-to-cloud continuum [59, 63, 68, 82]; edge nodes that

process data on the network node lying at exactly 1-hop from IoT client such as the edge router or gateway [57, 83]; and recently, even on the IoT client [84].

b) **Processing Techniques:** The type of processing technique applied to extract data-driven knowledge is dictated by the end-user objectives for using an application. The common IoT data processing techniques found from literature include: Stream processing [57, 60, 77, 94], Time-series processing [48, 72, 80], Rule-based processing [67, 95], Complex Event Processing (CEP) [52, 57, 66, 68, 69, 75], Context-aware processing [59, 65, 96], Semantic-based processing [56], Lambda Architecture [85, 86, 87], Learning-based processing using ML (Machine Learning) or DL (Deep Learning) [50, 51, 54, 64, 84, 97, 98, 99], and Hybrid processing such as, CEP + stream proc. [49], stream proc. + MapReduce [29], and others include [53, 33, 55, 63, 34, 81, 90, 93].

iv) **Data Retrieval**

After processing, the data-driven knowledge is retrieved by IoT applications using these data retrieval modes: i) visualization dashboards [48, 29, 55, 57, 60, 63, 64, 65], ii) query-based retrieval [67, 76, 78, 80, 95], iii) subscription-based retrieval [49, 56, 66, 68, 69, 77] using publish/subscribe model where the centralized data publisher a.k.a broker provides data to the subscriber nodes, iv) URI (Uniform Resource Identifier)-based retrieval such that every data resource is addressable with unique URI, which is enabled either at application layer via RESTful web services [48, 33, 34, 75, 93] or at network layer via Information-Centric Networking (ICN) [58] or at device level via M2M resource tree [91, 92].

2.1.2 Cloud Services for IoT and Big Data

Since IoT devices are resource-constrained, hence, resource-rich cloud computing platforms are used to augment the IoT technology. Thus, cloud providers start offering various services to support IoT and IoT big data on their cloud platforms.

IoT Portfolio of Public Cloud Vendors

This section reviews the IoT portfolio of four major public cloud vendors namely, Google, Amazon, Microsoft, and IBM. Cloud providers support IoT technology by offering cloud services for all four data lifecycle stages – IoT data acquisition, data storage, data processing, and knowledge delivery, summarized in Tables 2.1, 2.2, 2.3, and 2.4 respectively.

- **Google Cloud Platform (GCP)**

  GCP offers these products for IoT: i) Google Cloud IoT – the IoT PaaS to develop

Table 2.1: Cloud Services for IoT Data Acquisition

Type of IoT Integration Service	**Google Cloud Platform (GCP)**	**Amazon Web Services (AWS)**	**Microsoft Azure**	**IBM Bluemix**
IoT Device Management	Cloud IoT Core	Things Registry	Azure IoT Hub	Device Schema
Data Integration Service at Gateway	Cloud IoT Core	Device Gateway	IoT Hub	REST APIs
Custom Hardware	Edge TPU	AWS Button	-	IBM Power System AC 922
Service connecting Gateway to Cloud Native Services	Cloud Pub/Sub	Rules Engine	Azure IoT Hub	internally managed by WIoTP

Table 2.2: Cloud Services for Big Data Storage

Type of Storage Service	**Google Cloud Platform (GCP)**	**Amazon Web Services (AWS)**	**Microsoft Azure**	**IBM Bluemix**
Relational Database (for Structured data)	Cloud SQL, Cloud Spanner	Amazon RDS, Aurora	Azure SQL Database	IBM Db2 on cloud
Object Storage (for Unstructured data)	Cloud Storage	Amazon S3	Azure Blob Storage	IBM Cloud Object Storage
NoSQL Database (for Semi-structured data)	Cloud Bigtable	Amazon DynamoDB	Azure Cosmos DB	IBM Cloudant
Data Warehouse	BigQuery	Amazon Redshift	Azure Synapse	IBM DB2 Warehouse on cloud
Data Lake	Cloud Storage	AWS Data Lake on S3	Azure Data Lake Storage	IBM Data Lake

Table 2.3: Cloud Services for Big Data Processing

Type of Processing Service	**Google Cloud Platform (GCP)**	**Amazon Web Services (AWS)**	**Microsoft Azure**	**IBM Bluemix**
Stream Processing (proprietary)	Cloud Dataflow	Amazon Kinesis, AWS IoT Analytics	Azure Stream Analytics	IBM Streaming Analytics
Stream Processing (hadoop-based)	Cloud Dataproc	Amazon Elastic MapReduce (EMR)	Azure HD Insight	IBM Analytics Engine
Batch Processing	Cloud Dataflow	Amazon EMR	Azure Batch	IBM SPM Batch Processing
Function-as-a-Service (FaaS)	Cloud Functions	AWS Lambda Functions	Azure Functions	IBM Cloud Functions
ML/AI Analysis	Google Cloud AI Platform	Amazon SageMaker	Azure Machine Learning	IBM Watson Machine Learning, IBM Watson Studio

Table 2.4: Cloud Services for IoT Knowledge Delivery

Type of Knowledge Extraction Service	**Google Cloud Platform (GCP)**	**Amazon Web Services (AWS)**	**Microsoft Azure**	**IBM Bluemix**
Visualization Service	Cloud Data Studio	Amazon QuickSight	Microsoft Power BI, Azure Time Series Insights	IBM cognos Analytics
Actuation/ Push Notification Service	Cloud Pub/Sub	AWS IoT Events	Azure IoT Hub	IBM Cloud Push Notifications, IBM WIoTP Trigger Rules

and deploy IoT applications on GCP [100], ii) Android Things – the embedded OS for running android on IoT devices [101] iii) Cloud IoT Edge – the edge platform for IoT [102], iv) Edge TPU – the custom-built, application-specific hardware accelerated chips for running pre-trained ML models at the edge in an energy-efficient mode [103].

GCP offers the Cloud IoT Core service for registering and connecting IoT devices to the Google Cloud IoT. This service also performs device management. Once, the IoT data reaches GCP through this service, it can be further stored, processed and analyzed using any of the GCP services.

- **Amazon Web Services (AWS)**
  AWS offers three products specific to IoT: i) AWS IoT Core – the IoT PaaS to develop and deploy IoT applications on AWS [104], ii) AWS Button – a small customized hardware device, whose buttons are pre-configured with the user's IoT application running on the AWS [105], iii) AWS Greengrass – lite-version software that localizes AWS services to a local network of connected devices in order to deal with intermittent internet connectivity [106].

  The AWS IoT PaaS comprises of these four components – Device Gateway, Things Registry, Device Shadow, and Rules Engine. The Device gateway allows the IoT client devices to connect with AWS IoT. Once connected, the Things Registry service registers and maintains the IoT devices metadata. The Rules Engine service is used to route incoming IoT data to AWS native cloud endpoints and the Device Shadow service is used to deal with intermittent connectivity.

- **Microsoft Azure**
  Azure offers these IoT specific products: i) Azure IoT Solutions Accelerator- the IoT PaaS [107] that offers three pre-configured solutions namely, remote monitoring, predictive maintenance, and connected factory for quick application development

by using these templates, ii) Azure IoT Central: the IoT SaaS by Azure [108], iii) Azure IoT Edge: the edge SDK that extends Azure services to the edge gateway [109], and iv) Windows 10 IoT: the embedded OS to run Windows 10 on IoT devices[110].

Both, the IoT PaaS and SaaS i.e. the Azure IoT Suite and Microsoft IoT Central use the IoT Hub service for connecting and registering the IoT devices to the Azure cloud platform. Further the application development can be done using any of the Azure services.

- **IBM Bluemix WIoTP**
  IBM Bluemix supports IoT by IBM Watson IoT Platform (WIoTP), the cloud service to develop and deploy IoT applications [111]. The WIoTP offers a GUI-based toolkit for step-by-step IoT application development. The WIoTP platform offers: real-time REST APIs for connecting IoT devices to WIoTP, device schema for defining device metadata, if (condition) → then (action) analysis rules for cloud analysis. The WIoTP GUI offers real-time visualization using cards and boards.

2.1.3 Big Data Challenges in IoT

The unique characteristics of IoT during data generation, transmission, and processing stages pose several challenges for big data in IoT. This section performs the cause-effect mapping of IoT Big Data (IoTBD) characteristics to the unique challenges they cause. The IoT big data challenges are named using V's terminology, analogous to V's terminology for 3 V's challenges of traditional big data. The projected 13 V's challenges of big data in IoT are shown in Figure 2.2. The big data challenges encountered during different IoT data lifecycle stages are as following:

IoT Data Generation

The unique characteristics related to IoT data generation leads to the following cause →effect challenges:

i) Large Number of Sensors →Billions of sensors create Voluminous sensor-based IoT data.

ii) Continuous 24x7 Data Generation →data streams arrive at a very high Velocity.

iii) Heterogeneous multi-source data →data exhibits Variety due to dissimilar format, structure, and data types.

iv) Lack of Trust in IoT Sources →creates uncertainty about the source of data, which in turn, creates uncertainty about the truthfulness of acquired data i.e., the chal-

lenge of Veracity arises.

v) Weakly secured data at IoT client →makes data Vulnerable to security and privacy attacks.

vi) Dynamism in IoT →IoT data shows dynamic behavior either due to client mobility across heterogeneous networks, or event-based dynamicity in sensor-data values. Either of the dynamicity leads to Variability in IoT data.

vii) Nature of Autonomous Sensor Data →the noisy and low-quality IoT data is often inaccurate, which raises the issue of data Validity i.e., correctness of IoT data.

viii) Multi-Source Data Generation →the issue of management of data from multiple touch points is termed as Venue.

IoT Data Transmission

Data transmission in IoT experiences the following cause →effect challenges:

i) Voluminous data →saturates network bandwidth and multiple IoT applications compete for bandwidth.

ii) High Velocity data →arriving from multiple sources experiences high delay jitter.

iii) Variety →variety of data makes application QoS hard to satisfy.

iv) Increased Attack Surface →long multi-hop client-to-cloud network journey across hierarchical, heterogeneous (non-IP and IP network) networks, exposes data at multiple spots that represent the potential attack surface for data, which makes the IoT data Vulnerable during the data transmission stage.

v) Dynamic IoT Devices →the migration of client's data and the corresponding virtual environment (containers, or VM) to manage mobility for dynamic IoT client results in unforeseen Variability in sensor readings.

vi) Prone to Re-Transmission Errors →in IoT ecosystem, the short-range non-IP networks suffers from intermittent network connectivity. Following this, the network's attempt to re-transmit the missing values often creates errors and makes the data inaccurate, which is termed as, Validity.

vii) In-network processing →intermediate processing of IoT data at edge routers, gateways, fog nodes encompasses multiple Venues within the network where the data is accessed and processed. The accounting and management of these heterogeneous venues is a critical challenge in IoT.

viii) High Network Latency →the multi-hop client-to-cloud journey results in huge network lag, which reduces the freshness and usefulness of received data for analysis in time-critical applications. In other words, IoT data for strict real-time applications is highly Volatile i.e., data has very short-term relevance for analysis. Volatility is the issue that relates to temporal irrelevance of recent past data for analysis.

IoT Data Processing

The data processing encounters following cause →effect challenges in IoT:

i) Multi-venue data processing →the distributed processing of IoT data across the client-to-cloud network nodes encompassing multiple Venues (such as edge/ fog nodes, and cloud platforms) requires cloud co-ordination for efficient management of partially-processed data, and management of multiple venues with regard to per-use metering, billing, and QoS satisfaction for IoT applications.

ii) Continuous Strict Real-time Analysis →strict real-time analysis demands extremely latest data to be continuously available for analysis. The issue of determining what quanta/time-frame of sensor readings is useful for analysis and making such data continuously available is termed as data Volatility.

iii) Meaningful Insights to be extracted →the efficacy of IoT applications lies in deriving Value by data-driven analysis of raw sensory data.

iv) Need for Semantic Processing →the challenge of creating, updating and managing sensor ontologies, and vocabularies for semantic processing is termed as Vocabulary.

v) Incompleteness in Multi-Source Data →when multi-source IoT data is combined together for meaningful analysis, there exists an incompleteness or inconsistency in information termed as Vagueness i.e., the data ambiguity in combining multi-source data.

vi) Summarized Knowledge Representation →the challenge of presenting the summarized knowledge to the end-user in a summarized and comprehensible manner is termed as Visualization. Visualization is challenging in IoT due to high-dimensional of data, continuous analysis loop, and varied types of data.

IoT Applications

The IoT applications pose following challenges:

i) Continual data-driven knowledge: continuous analysis of IoT data is aimed to extract meaningful Value from IoT data.

ii) Real-time Knowledge Representation: the extracted knowledge must be presented

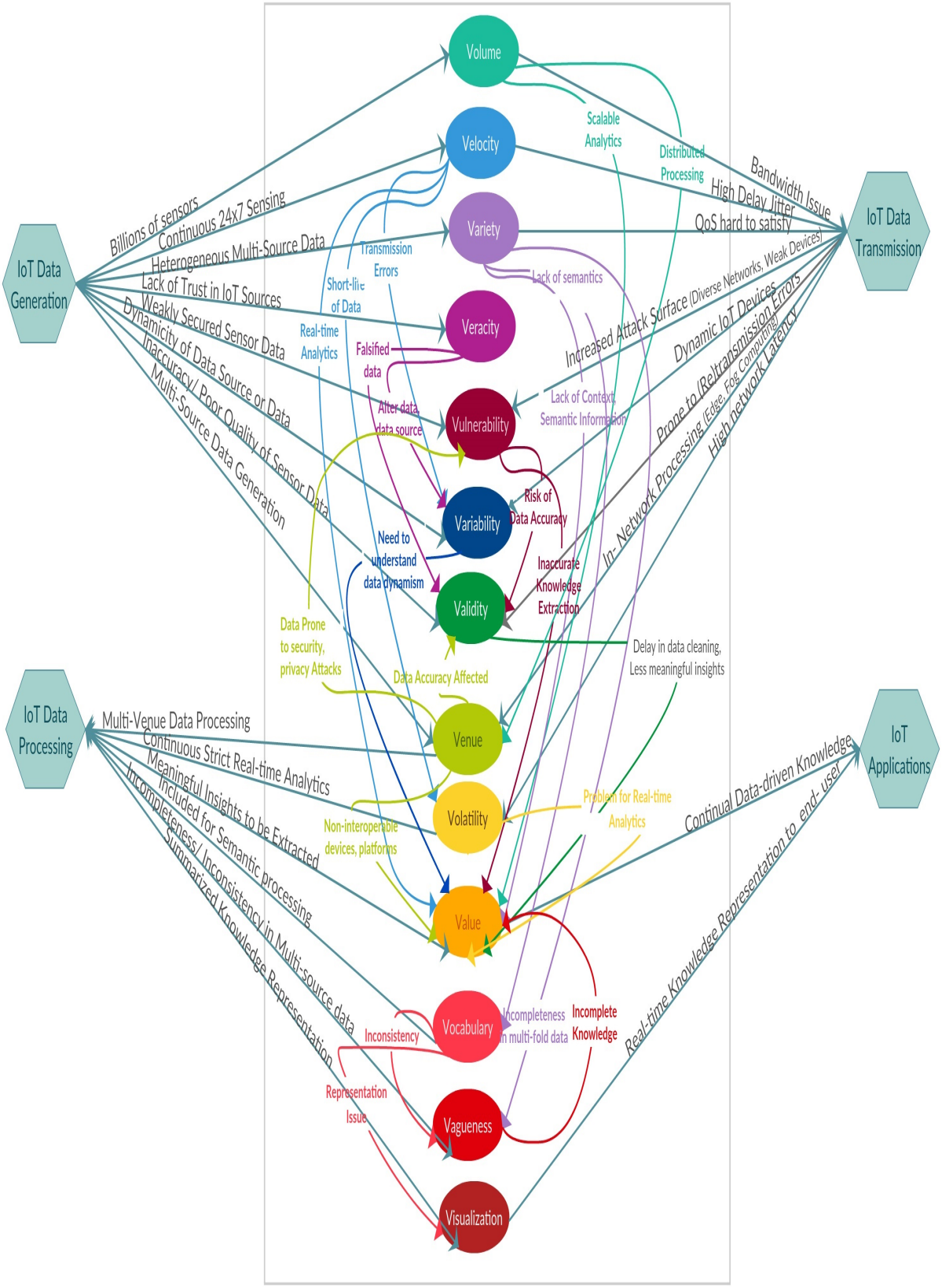


Figure 2.2: Challenges for Big Data in IoT

to end-user by appropriate interactive Visualizations that are easy to comprehend and demonstrate the real-time state of IoT devices.

2.1.4 Research Contribution: 13 V's Challenges of Big Data in IoT

Based on the identified 13 V's challenges for big data in IoT, this section presents the research contribution of this review study as 13V's challenges for big data in IoT defined as follows and shown in Figure 2.3.

Volume

This characteristic of Big Data denotes the colossal amount of data which makes the Big Data- "Big". The volume of IoT data is of the order of Zettabytes or even higher magnitudes. Undoubtedly, such enormous data requires distributed solutions for storage and processing. However, the existing distributed solutions suffer from scalability issue especially for large-scale IoT applications that involve both, real-time and historical data storage, and management. Further, high volume of IoT data strains the current network bandwidth, thereby challenging the optimized use of available network bandwidth among data-intensive applications of multiple stakeholders.

Variety

This characteristic refers to the heterogeneous formats of data including structured, semi-structured and unstructured data. The issue of variety relates to efficient data modeling, in order to store, query, process, and analyze the dissimilar formats data. Variety is a common characteristic in IoTBD, as IoT applications often involve multi-source data (examples, WSN, sensor-based IoT devices, multimedia, MCS, open data etc.), which is often generated in disparate formats.

Velocity

Velocity refers to the speed at which data is ingested. This issue deals with the ability of big data solutions to receive and process the rapidly arriving data. In IoT, the real-time streaming data arrives the computing infrastructures at very high velocity. Moreover, the data ingestion speed of multiple sources may vary, that demands flow control techniques to regulate the data ingestion rate from multiple sources.

Veracity

Veracity is the issue of uncertainty about truthfulness of captured data due to lack of trust in data source(s). In IoT, data inaccuracy is introduced at the data provenance level (i.e., data source level) due to several reasons such as, sensor malfunctioning owing to device wear-and-tear, lack of trust and reliability of data sources, and in some IoT applications

particularly relying on MCS people may intentionally provide false information for fun or misguidance. Such data inaccuracy, right at the data origin stage, hampers the quality of data, and hence, the overall value of analytics decisions based on such untruthful data.

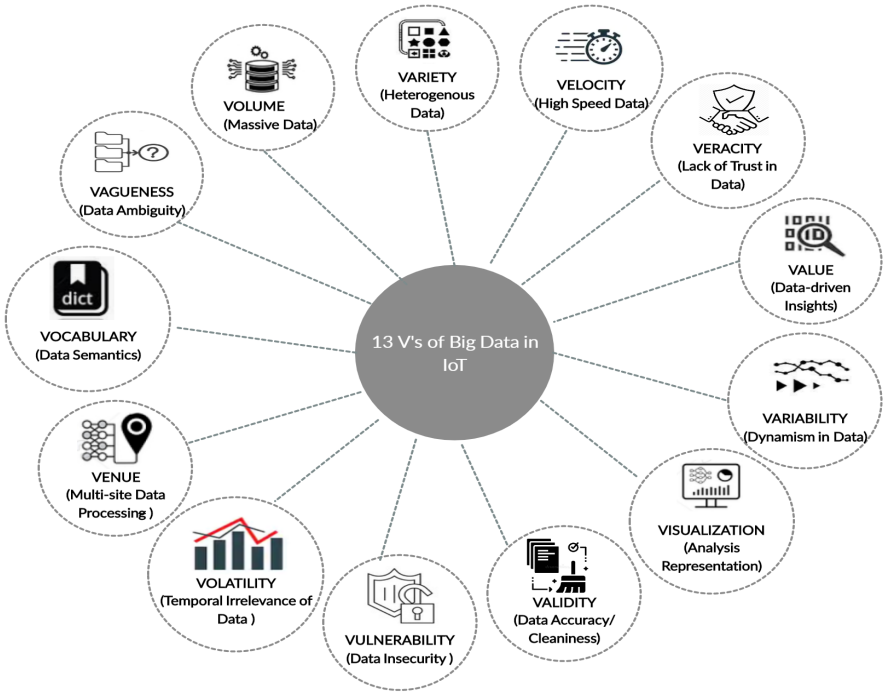


Figure 2.3: 13 V's Challenges of IoT Big Data

Value

Value implies the data-driven knowledge based on which, the intelligent actions/decisions are taken on the connected IoT devices. Knowledge extraction in IoT suffers from issues such as, nature of data: real-time data, fine-granule, weak in semantics, and space-time correlated; quality of data; and data processing required for both, real-time and historic data. However, the quality of analysis can be improved by the optimizing several factors such as choice of analysis algorithm, selection of data sample, by using clean and valid data for analysis.

Variability

Variability is the issue of dynamicity associated with the data or data sources. In IoT, dynamicity is introduced either due to mobility of IoT devices, or variability in sensor data due to occurrence of event-of- interest in real-time. The quickly moving IoT devices may suffer from intermittent data loss during frequent inter-network data migrations.

Visualization

Visualization is the issue of representing the summarized knowledge to the end-user in the

form of interactive graphics or visuals. Visualization in IoT is challenging due to high-dimensionality of data, visual inexpressiveness of some data formats (such as multimedia, ontologies, social data etc.), and continuously changing real-time scenario.

Validity
Validity is the issue of uncertainty about data accuracy or cleanliness to carry out data analysis. In IoT, the long multi-hop journey of data across heterogeneous networks affects data accuracy by introduction of noisy, missing or redundant values. Similar to veracity, validity also relates to the poor quality of IoT data and adversely impacts the value of analytics decisions. However, unlike veracity, validity is the degradation of data after it has been acquired i.e., mainly during data transmission, pre-processing and cloud storage stages before carrying out the actual analysis of data.

Vulnerability
Vulnerability is the issue of insecure data that is susceptible to security and privacy attacks. Ensuring data security and privacy in IoT is more complex due to several factors such as, weakly secured IoT clients, data passage through multiple touch points, increased attack surface owing to outsourced processing at edge/fog nodes, heterogeneous IP and non-IP protocols.

Volatility
Volatility denotes the short-term freshness of IoT data. In other words, volatility is the issue of the temporal irrelevance of past data for analysis, and deals with data concurrency, availability, and rapid retrieval. The real-time IoT data analysis involves only the current and recent data values, while discarding the distant past data values. In this regard, the issue of volatility relates to what quanta or time-frame of data to be used for real-time analysis? and to continuously make such data available for analysis.

Venue
Venue is the issue of varying access rights, ownership (public/private/hybrid) and non-interoperability of multiple venues used by data since its origin. In IoT, the data and computation offloading on the intermediate edge, fog nodes necessitates monitoring, and management of data across multiple heterogeneous venues.

Vocabulary is the issue of semantically representing data using ontologies, metadata, hashtags, data handles, dictionaries etc. In IoT, the vocabulary relates to creation, maintenance, and updating of standard cross-domain vocabulary for sensors, services, applications, platforms to enable semantic processing of IoT data.

Vagueness
Vagueness refers to the data ambiguity and incompleteness when combining data from

multiple sources or diverse paradigms. Diverse sources and paradigms vary in data modeling and metadata management techniques resulting in disjoint pieces of information while integrating multi-source data. The classic example of vagueness in IoT is seen while integrating IoT data with social network data, as the two paradigms model data in different ways viz. relationships between entities and device telemetry respectively.

2.2 Big Data Management in Smart City Frameworks

Previous section reviewed big data in IoT across several domains, however, this section confines to big data management in smart city domain. This section reviews state-of-the-art smart city frameworks and identifies research gaps in existing works.

2.2.1 State-of-the-Art Smart City Frameworks

This section reviews the current literature on big data management for smart cities.
Ramos et al. [112] propose a data lake based architecture for heterogeneous big data management for smart cities. This work propose to integrate heterogeneous data from different external databases, IoT sensors, stream data, data warehouses, and so on using Apache NiFi into a Hadoop data lake. The data lake supports distributed query processing and analysis ans uses a 3-rd party metadata management tool named as, HANDLE. The proposed data lake architecture is validated for a case study on zip-code wise geographic distribution of no. of citizens using three different smart city applications of Alagoas state, Brazil. The limitations of this work include, not a cloud-based architecture, did not use IoT data, and no use of spatial processing rather the zip code values column in application databases is used for querying the data.

Mete et al. [43] propose a framework for geospatial big data analysis for a smart city. This work performs spatial join on city's buildings energy-efficiency data with their geospatial data to perform spatial clustering for region-level assessment of energy-efficiency levels. This work compares the performance of Dask-GeoPandas vs. Apache Sedona to perform spatial operations like join and clustering, wherein Sedona performs better than Dask-GeoPandas. The limitations of this work include, non-IoT urban data, not a cloud-based architecture, and compute-intensive addition of new data into the warehouse storage.

Raif et al. [113] propose IISOBA, a big data architecture to integrate heterogeneous IoT, and drones data for a smart city. In this work, Apache NiFi is used to flow data from diverse sources to a Hadoop data lake, for which Apache Hive metastore is used for metadata management and Apache HBase is used to support queries over the urban data.

The limitations of this work include, lack of case study implementation and performance evaluation results, and it is not a cloud-based architecture.

Riberio et al. [114] propose a smart city big data framework based on microservices architecture. This work stores real-time data in Hadoop data lake with additional microservices for metadata manager, authentication, data querying, and so on. The limitations of this work include, not a cloud-based architecture, and lacks case study implementation and experimental results.

Ordonez-Ante el al. [115] propose Explora to perform spatio-temporal querying of smart city data. This work proposes two queries: snapshot temporal and historical spatial to analyze air quality monitoring stream data for the city of Antwerp in Belgium. The stream data summaries are continuously stored in a custom 3D cube data structure to represent continuous data views. In addition, for persistent storage, this work considers two implementations one using PostgreSQL as the spatial database approach and second using Apache Kafka and RocksDB as the distributed stream processing approach. Experimental results prove that distributed processing approach scales well as data grows rather than spatial database approach. The limitations of this work include, not a cloud-based architecture, limited analysis types, extensive storage space consumed for continuous and persistent storage.

Santos et al. [116] propose a Spatial Online Analytical Processing (SOLAP) framework for IoT data stream in a smart city. This work stores incoming data into HDFS and processed data into a spatial data warehouse. In addition, GeoSpark is used for spatial processing in the form of three types of spatial queries: spatial join, k-nn, and containment query followed by QGIS for visualization. The framework is validated for the traffic dataset for Aarhus city, Denmark. The limitations of this work include, no support for predictive analysis, limited types of spatial queries, ignored the temporal dimension.

Li et al. [117] propose JUST, a platform-level data engine for spatio-temporal analysis of data. This work takes spatio-temporal data from external data sources or databases and uses Apache Spark to perform three types of queries viz. spatial range, spatio-temporal range, and k-nn. The limitations of this work include, non-IoT data, not a cloud-based solution.

Alic et al. [118] propose BIGSEA, a platform for big data analysis of spatial trajectories from public transportation system. This work stores data in Ophidia that stores multi-dimensional data using a data cube abstraction and analyzes using Spark for entity matching of spatial coordinates and crowdedness prediction. The framework is validated using public transport trajectories from Brazil. The limitations of this work is that it is

designed specific to undertaken use case and is not adaptable for other smart city use cases, based on non-IoT data.

Anejionu et al. [119] propose SUDS, a cloud-based spatial big data processing system for open government data across major cities of U.K. This work considers social and economic data to propose urban metrics like transport availability, housing affordability, and so on. This work uses both spatial and non-spatial (social, economic) data, such that spatial data is stored using in data lake and spatial database PostGIS, while the non-spatial is stored using Snowflake. Descriptive queries on warehouse storage are run to measure the desined spatial urban metrics and geoserver is used for map-based visualizations. The limitations of this work include, data storage in silos i.e., separate spatial and non-spatial storage, and non-IoT data.

Osman et al. [120] propose Smart City Data Analytics Panel (SCDAP), a big data platform for analyzing smart city data from multiple sources like IoT, social media, Wireless Sensor Network (WSN). This platform stores data in multiple data-silos HDFS, NoSQL and SQL databases and then, process it using Apache Storm and Spark to analyze both real-time and batch smart city data. The limitations of the proposed platform are as non-cloud platform, siloed data storage, no visualization support, no proof-of-concept implementation.

Rathore et el. [53] propose a system for smart city real-time analytics. The proposed system uses Hadoop ecosystem, specially Spark over Hadoop to process smart city data from various IoT datasets. This work performs graph processing using Giraph on Hadoop for analyzing transport data. The limitations of this work include, non-cloud architecture, no consideration of spatial data.

Silva et al. [39] propose an architecture for smart city IoT big data management. The proposed architecture uses HDFS and Apache HBase for data storage, and uses MapReduce for offline data processing and Spark for online data processing. The limitations of this work include, non-cloud architecture, no fusion with non-IoT data.

Lv et al. [121] proposes integration of Geographic Information System (GIS) with cloud computing to perform 3D spatial processing of data using GIS-on-cloud to support various smart city uses cases. The limitations of this work include, lack of practical case study implementation, no big data handling, no IoT data used.

Bellini et al. [122] propose Sii-Mobility an architecture for smart city data management. The proposed architecture integrates multi-source city data, such as static POI data, real-time mobility data (bus tracking, mobility events, and so on) with Km4City smart city ontology. This work stores data using HDFS, Apache HBase, Apache Pheonix over

HBase, and Virtuoso (semantic data store) for semantic-based integration to produce noSQL records of data. The data is retrieved through SPARQL queries or the REST APIs to develop mobile, web applications for dashboards, city information systems. The limitations of this work include, non-cloud architecture, data inconsistency when musing multiple storage solutions.

Babar et al. [30] propose a three-layer big data management architecture for smart cities. The proposed data management architecture is based on Hadoop ecosystem and processes the data using MapReduce for analysing Aarhus city data. The limitations of this work include, non-cloud architecture, does not consider spatial data, limited analytics using MapReduce.

Negru et al. [123] propose a two-layer storage based architecture for managing heterogeneous smart city data. The proposed architecture ingests variety of data from IoT, WSN, spatial data into Hadoop data lake as the bottom layer and maintains various types of databases atop the data lake, such as Oracle spatio-temporal database, NoSQL databases, document and tabular databases. The limitations of this work include, lack of proof-of-concept implementation, no fusion of spatial and IoT data, two-layer storage may lead to data inconsistencies.

Chen et al. [124] proposes VAUD for spatio-temporal analysis of urban data. This work proposes to use multi-source urban data, such as spatial, POIs, GPS trajectories, and so on for exploratory analysis i.e., which, when, where queries for urban data. This work stores the spatio-temporal data using Space-Time-cube (ST cube) storage and runs exploratory queries. The limitations of this work include, non-cloud architecture, did not tap big data technologies, the cube-based data storage on single-node will face scalability issues, and based on non-IoT data.

Rathore et al. [29] propose a four-tier architecture for smart city big data management. The proposed architecture is based on Hadoop ecosystem that stores data in HDFS and uses MapReduce, Spark, VoltDB, and Storm to analyze the urban IoT data. The limitations of this work include, non-cloud architecture, does not consider spatial data.

Table 2.5 taxonomizes the reviewed smart city big data frameworks on the basis of framework design and Table 2.6 taxonomizes the reviewed smart city big data frameworks on the basis of data management.

Table 2.5: Taxonomy of Smart City Big Data Frameworks Based on Framework Design

Author(s)	Nature of Data	IoT and Geospatial Fusion	Distributed Handling of Big Data	Implemented Case Study
Ramos et al., [112] 2023	IoT, external databases	✗(used zip-code as textual column in databases)	✓	Zip-code wise analysis of no. of citizens using 3 different smart city apps for Alagoas state (Brazil)
Mete et al., [43] 2023	open data (csv), geospatial data (geopackage)	✓	✓	Spatial analysis of Buildings Energy-efficiency data for England and Wales
Raif et al., [113] 2022	IoT devices, drones, and video cameras	✗	✓	None
Riberio et al. [114] 2021	real-time data stream	✗	✓	None
Ordonez-Ante et al. [115] 2020	IoT data stream	✗	✓	Air quality monitoring for Antwerp city in Belgium
Santos et al. [116] 2020	IoT data stream, GIS spatial data	✗	✓	Traffic data from Aarhus, Denmark
Li et al. [117] 2020	spatio-temporal dataset	✗	✓	Logistics trajectory dataset analysis
Alic et al. [118] 2019	Public transport GTFS trajectories (non-IoT data)	✗	✓	Public Transport Trajectories Analysis for Brazil
Anejionu et al. [119] 2019	Social, economic data	✗	✗	Urban Metrics analysis for major cities of U.K.
Osman et al. [120] 2019	IoT, WSN, social media	✗	✓	None
Rathore et al. [53] 2018	IoT, Graph data	✗	✓	Graph processing of transport data using Giraph
Silva et al. [39] 2018	IoT	✗	✓	Analysis of IoT datasets from Aarhus, Denmark
Bellini et al. [122] 2018	Static POI data, real-time mobility data, spatial data of city, Km4City ontology	semantic-based fusion	✓	Integration of Mobility trajectories, static POIs, spatial data for the city of Florence/Tuscany area
Lv et al. [121] 2018	3D spatial data from GIS	✗	not specified	only theoretical use-cases, lacks practical implementation
Babar et al. [30] 2017	IoT	✗	✓	Analysis of Aarhus city data
Negru et al. [123] 2017	IoT, Spatial, WSN, external APIs	✗	✗	None

Continued on next page

Table 2.5 – *Continued from previous page*

Author(s)	Nature of Data	IoT and Geospatial Fusion	Distributed Handling of Big Data	Implemented Case Study
Chen et al. [124] 2017	non-IoT (GPS trajectories, POIs, social media)	✗	✗	Analyzing traffic jam, find lost phone use cases for a city
Rathore et al. [29] 2016	IoT	✗	✓	Analysis of IoT datasets from several cities

Table 2.6: Taxonomy of Smart City Big Data Frameworks Based on Holistic Big Data Management

Author(s)	Data Storage	Data Processing	Data Visualization	Data Management
Ramos et al., [112] 2023	Hadoop Data Lake	Descriptive Querying (PrestoDB, Tableau)	Streamlit (map)	HANDLE, a 3rd-party metadata management tool
Mete et al., [43] 2023	Data Warehouse	Spatial Clustering (compared Dask-GeoPandas with Apache Sedona)	Datashader (map)	✗
Raif et al., [113] 2022	Hadoop Data Lake	Descriptive Querying (Apache HBase)	✗	Hive metastore for metadata management
Riberio et al. [114] 2021	Hadoop Data lake	Descriptive and Predictive (Apache kafka, Spark)	Apache Superset, Kibana	metadata catalog in NoSQL
Ordonez-Ante et al. [115] 2020	3D space-time data structure, PostgreSQL, RocksDB	Descriptive Querying (Apache Kafka)	Matplotlib	✗
Santos et al. [116] 2020	Spatial Data warehouse	Descriptive Spatial Querying (GeoSpark)	QGIS	✗
Li et al. [117] 2020	HDFS, Hive, Apache HBase	Descriptive Spatio-Temporal Querying (JustQL)	✗	metadata table in MySQL
Alic et al. [118] 2019	Ophidia (data cube abstraction)	Descriptive: trajectory matching, Predictive: crowdedness prediction (Apache Spark)	map	✗
Anejionu et al. [119] 2019	Data lake, PostGIS, Snowflake warehouse	Descriptive BI Queries (Snowflake warehouse)	geoserver	only for non-spatial data
Osman et al. [120] 2019	HDFS, Apache HBase, Kudu (NoSQL databases)	Data Mining, Machine Learning	✗	✓
Rathore et al. [53] 2018	HDFS	Spark over Hadoop, Giraph	line plots, bar plots	✗

Continued on next page

Table 2.6 – *Continued from previous page*

Author(s)	**Data Storage**	**Data Processing**	**Data Visualization**	**Data Management**
Silva et al. [39] 2018	HDFS, Apache HBase	Batch Processing (MapReduce), real-time (Spark)	line and bar plots	✗
Bellini et al. [122] 2018	HDFS, HBase, Pheonix, Virtuoso	Queries (SPARQL), REST APIs	✗	✗
Lv et al. [121] 2018	Cloud storage	3D spatial processing	GIS	✗
Babar et al. [30] 2017	HDFS	Batch Processing (MapReduce)	line, bar plots	✗
Negru et al. [123] 2017	two-layer: multiple databases atop Hadoop data lake	Not specified	✗	✗
Chen et al. [124] 2017	Space-Time cube	Exploratory queries (which, when, where, what)	map-based	✗
Rathore et al. [29] 2016	HDFS, HBase, Hive, SQL	Batch (MapReduce), Real-Time (Spark, Storm)	line, bar plots	✗

2.2.2 Analysis of Smart City Frameworks

This section analyzes state-of-the-art smart city frameworks for big data management from two aspects – framework design and data management approaches as follows:

- **Framework Design**
  Smart city frameworks must be scalable to handle large scale urban data. Scalable infrastructure and resources to manage the urban big data are provided by the resource-rich cloud computing platforms. However, few works [115, 124, 122, 29, 53, 120, 39, 30, 114, 112] did not consider cloud-based architecture for their smart city frameworks. In addition to scalable resources, the cloud computing also offers fault-tolerance, security, and privacy features. Thus, an ideal smart city framework must adopt cloud-based architecture for data-centric smart city applications. Next, distributed storage and processing are recommended to address scalability [115, 125]. Single-node data storage approaches [119, 115, 124] will face issues like lack of storage space and reduced processing speed as the data from smart city increases.

  The second major design concern for smart city frameworks is the integration of heterogeneous data. Some works [126, 127, 128, 122] consider semantic approaches for integration of Points of Interest (POI) information with the IoT data. However, the semantic approaches store data as Resource Description Framework (RDF) triples and use semantic query language SPARQL for data retrieval. Semantics-based for data integration is unsuitable for smart city applications, as SPARQL retrieves data slower than curated databases, warehouses and do not support Machine Learning (ML) and Artificial Intelligence (AI) analysis unlike data lakes. On the other side, if additional information (POI, and so on) is integrated during the Extract, Transform, and Load (ETL) pipeline while storing the IoT data, it would incur the processing latency only once, rather than impacting the latency for every single retrieval query. Thus, novel approaches for heterogeneous data integration at ETL level must be investigated.

- **Data Management Approaches**
  Data management involves urban big data storage, processing, and management for smart city applications. Current approaches to store smart city data can be classified as database storage (NoSql [117, 39], spatial databases [119, 115], multiple databases atop HDFS [123, 122, 120]); custom data structures [115, 124, 118]; data warehouse [119, 116]; and hadoop-based data lakes [121, 116, 118, 29, 53, 123, 30, 114, 113, 112]. The database storage approach is limited by I/O bottlenecks, lack of parallelism, and scalability issue for smart city applications [125]. Few works

[115, 124, 118] implemented custom data structure in the form of three-dimensional cube for data storage and are limited by scalability, lack of open-source APIs and cloud implementation. The data warehouse [119, 116] storage is optimized for storing large scale structured big data that allows fast query retrieval. However, the limitations of warehouse storage include lack of support for unstructured data and poor retrieval performance for ML/AI workloads [129]. Such limitations i.e., ability to store unstructured big data storage and optimized data retrieval for ML/AI analytics are addressed in data lake storage. Data lakes store big data as distributed part files either using the Hadoop File System (HDFS) or using the cloud-based storage. However, data lakes do not address data quality and metadata management issues. Nevertheless, a novel data storage framework – data lakehouse [130] is emerging that combines the benefits of both data warehouse and data lakes.

Data lakehouse [130] is a recent data storage and management architecture that offers reliable big data storage, high-performance retrieval, in-built metadata management, data quality and governance features. This recent data storage framework, not implemented by existing works, is a promising direction for smart city big data management. Table 2.7 provides a comparative analysis of traditional big data management architectures i.e., data warehouses and data lakes with the lakehouse architecture.

Data processing extracts valuable knowledge and insights from the stored urban big data using various data analytics approaches. Data analytics approaches can be classified into three types, descriptive, predictive, and prescriptive [44]. Descriptive analytics derive summaries, extract hidden patterns, present visual reports for answering 'what happened in the past' queries. Descriptive analytics approaches to process the urban big data include, batch processing i.e. MapReduce [121, 123, 29, 30]; batch + stream processing [118, 120, 39, 53, 114]; data retrieval queries (spatial [119, 116], spatio-temporal [115, 117, 124], others [122, 123, 118, 114, 113, 112]). Predictive analytics learn hidden dynamics in large amount of historical data to predict future events like 'what will happen in the future'. Predictive analytics include Machine Learning (ML) [120, 118, 114, 131, 132] and Deep Learning (DL) [44, 133, 134, 135]. Prescriptive analytics suggests future decisions, such as 'what should be done to make this happen'. This type of analytics has not been undertaken in existing works. The type of analytics in a smart city use case depends on the nature of data and the underlying use case. Ideally, various kinds of analytics, at least both descriptive and predictive analytics, should be seamlessly supported within a single smart city framework. In addition, data management features, such

as add and and analyze latest IoT data and data quality management are crucial for smart city big data frameworks.

Table 2.7: Comparison of Traditional Data Management Architectures with the Lakehouse Architecture

Features	**Data Warehouse (DW)**	**Data Lake (DL)**	**Data Lakehouse (DLH)**
Nature of Data	Structured data	Semi-structured, unstructured data	All: structured, semi-structured, and structured data
Data Quality	Highly curated, structured data, reliable	Raw data in native formats, not reliable	Raw and curated data in open data formats, reliable
Type of Analytics	Querying, BI and reporting	Data Science: ML and AI	Both, querying and ML/AI
Support for ACID transactions	Yes	No	Yes
Schema	Schema-on-read	Schema-on-write	Schema-on-read
Cost	High	Low	Low
Data governance and metadata management	Can be integrated in DW	Need to use 3rd party tools	In-built support for metadata handing and data governance
Target Users	Enterprise users	Data scientists and engineers	Both, data engineers and enterprise users

2.2.3 Research Gaps in Smart City Frameworks

From the state-of-the-art analysis of smart city frameworks for big data management, following research gaps have been identified:

RG1 *No Fusion of IoT data with City's Geospatial Data:* Integrating IoT big data (spatio-temporal sensor data) with geospatial data (city locations and maps data) is crucial in analyzing urban dynamics from space-time dimensions at multiple scales. Previous works did not embrace this promising integration and they consider only one type of urban data in their work. For instance, [29, 53, 120, 39, 30, 114, 113, 112] consider only IoT data without geospatial data, while [119, 117, 118, 124] consider only geospatial data without IoT data. However, considering both types of urban data, as few works [121, 116, 115, 123, 122] have started, is crucial for an ideal smart city framework.

RG2 *Non-Cloud Architecture:* Many works [115, 124, 122, 29, 53, 120, 39, 30, 114, 112] did not consider cloud-based architecture for their smart city frameworks.

RG3 *Scalability:* Apparently, urban big data will pose scalability issue when it is stored and analyzed using single-node technologies like using spatial databases or grid-based custom data structures [119, 115, 124]. On the contrary, big data infrastructures based on distributed storage and processing [115, 125] are recommended to address scalability.

RG4 *Inefficient Storage Architecture:* Storage architectures in previous works include

Hadoop-based data lake [121, 116, 118, 29, 53, 123, 30, 114, 113, 112]; Databases: NoSql [117, 39], spatial databases [119, 115], multiple databases atop HDFS [123, 122, 120]; Data Warehouse [119, 116]; and custom data structures [115, 124, 118]. However, novel efficient storage architectures like the Lakehouse architecture [130] that features the combined benefits of both, the data lake and data warehouse is not explored for smart city frameworks yet.

RG5 *Add/Store New Data:* Smart cities will generate new data over time, which should be efficiently cleaned, pre-processed and loaded into framework's storage layer. None of the previous works address this challenge.

RG6 *Limited Analytics Types:* Types of analytics in previous works include: Batch processing i.e. MapReduce [121, 123, 29, 30]; Batch + Stream [118, 120, 39, 53, 114]; Retrieval queries: Spatial [119, 116], Spatio-Temporal [115, 117, 124], others [122, 123, 118, 114, 113, 112]; Machine Learning [120, 118, 114]. Ideally, various kinds of analytics should be seamlessly supported within a single smart city framework.

RG7 *Spatio-Temporal Perspective & Queries Taxonomy:* As mentioned earlier, majority of works neglected spatio-temporal analysis of urban big data. While few works [123, 124, 115, 117] consider spatio-temporal aspects of urban IoT data, they analyze the urban data for a few, yet different queries. Nevertheless, there exist a knowledge gap in literature for a standard taxonomy or list of baseline Spatio-Temporal (ST) queries for smart city applications.

RG8 *Multi-Resolution Analysis:* Space and time information can be represented at multiple resolution scales. For example, space can have different scales like point (longitude-latitude site), small area (neighborhood, postal district etc.), large area (zone, governance area), and entire city and time can have different scales like hour, day, week, month, year etc. The framework should be flexible enough to support analytics for varying space-time scales. Only [115] considered multi-resolution analysis, however, this work store aggregate data for each resolution scale, which limits number of resolution scales and is inefficient. Rather, aggregates/ statistics for multiple resolutions can be computed upon query demand to avoid lot of pre-computation and save extra storage.

RG9 *Data Quality & Governance:* Many works [121, 116, 115, 124, 123, 29, 53, 120, 39, 30] ignored data quality, metadata management and data governance challenges, which must be addressed for an application like smart city.

RG10 *Data Versioning & Time Travel:* Smart Cities will generate new data over time, which should be integrated into the smart city framework to support analytics on fresh as well as any previous version of data selectively. Therefore, the framework should support

schema validation while adding latest data, data versioning and time travel to different data versions in order to effectively enable analytics for chosen version of data. None of the previous works address the data versioning and time travel challenge, which is an important requirement for smart city frameworks.

Table 2.8 provides a summary of identified research gaps in previous works on big data management in smart city frameworks.

Table 2.8: Research Gaps in Previous Works w.r.t Big Data Management in Smart City Frameworks

Elements of SC Framework	**Research Gaps**	**Status of Previous Works**	**Desired for Smart City Frameworks**
Framework Design	Nature of Data	Ignored either IoT or geospatial urban data: IoT data without geospatial data [29, 53, 120, 39, 30, 114, 113, 112]; geospatial data without IoT data [119, 117, 118, 124]. Only few works consider both, IoT and geospatial data [121, 116, 115, 123, 122]	Both types of data, IoT and geospatial data of the city are crucial for an ideal Smart City Framework
	Non Cloud Architecture	Many works did not consider cloud-computing [115, 124, 122, 29, 53, 120, 39, 30, 114, 112]	Ideally, the Smart City framework should follow a cloud-based architecture [1, 38, 136, 125]
	Scalability	Few works [119, 124] did not adopt distributed processing to address scalability	Distributed processing is recommended to address the scalability challenge [125, 115]
Data Storage	Inefficient Storage Architecture	Storage architectures include: hadoop-based data lake [121, 116, 118, 29, 53, 123, 30, 114, 113, 112]; Databases: NoSql [117, 39], spatial databases [119, 115], multiple databases atop HDFS [123, 122, 120]; Data warehouse [119, 116]; custom data structures [115, 124, 118]	Novel efficient storage architectures like the Lakehouse architecture [130] that features the combined benefits of data lake and warehouse should be explored for Smart City frameworks
	Add/Store New Data	None of the previous works address this challenge	Smart Cities will generate new data over time. Thus, the Smart City framework should accommodate addition of latest data
Data Processing	Limited Analytics Types	Analytics include: Batch Proc. (MapReduce) [121, 123, 29, 30]; Batch + Stream [118, 120, 39, 53, 114]; Retrieval queries: Spatial [119, 116], Spatio-Temporal [115, 117, 124], others [122, 123, 118, 114, 113, 112]; Machine Learning [120, 118, 114]	Various kinds of analytics should be seamlessly supported within a single Smart City framework

Continued on next page

Table 2.8 – *Continued from previous page*

Elements of SC Framework	**Research Gaps**	**Status of Previous Works**	**Desired for Smart City Framewoks**
	Spatio-Temporal Perspective & Queries Taxonomy	Many works did not consider spatio-temporal analysis [29, 53, 39, 120, 30, 113, 114]; only spatial analysis [122, 121, 119, 116, 112]. Few exceptions are: [123, 124, 115, 117, 118]	Analyzing urban data from spatio-temporal perspective is crucial to derive useful data-driven insights about urban phenomena. Moreover, there is a need to propose a taxonomy of ST Queries for smart city applications
	Multi-Resolution Analysis	Previous works neglected multiple space-time resolutions of data except [115]	Multi-level urban decision making (operational, managerial, policy making etc.) requires considering multiple space-time resolutions for fine-grain to coarse-grain analysis of data
Data Management	Data Quality & Governance	Many works did not address data management at all: [121, 116, 115, 124, 123, 29, 53, 120, 39, 30]. While, few addressed as: data management [119]; metatdata management [117, 118, 114, 112]; data quality [122, 113]	Efficient data management in a Smart City framework should include metadata management alongwith data quality and governance
	Data Versioning & Time Travel	None of the previous works address the data versioning and time travel challenge	Smart City framework should include new data periodically over time. Thus, efficient data versioning is crucial to enable time travel among different versions of urban data

The critical analysis of state-of-the-art smart city frameworks identifies these existing research gaps, which must be addressed to design and develop an efficient smart city framework.

2.3 Smart City: Edge Computing Support for Real-Time IoT Applications

This section focuses on real-time smart city IoT applications. Such applications are latency-critical, thus, to satisfy Quality of Service (QoS) criteria for such applications edge and fog computing [137, 17, 18, 138, 139] are used, in collaboration with the cloud, to run such applications in a multi-tier architecture. Research works on optimal service placement for edge-based architecture are reviewed to enable real-time smart city IoT applications.

2.3.1 State-of-the-art Service Placement in Edge-based Architecture

The things-centric smart city IoT applications should exploit edge-based architecture for QoS-aware service delivery of real-time smart city IoT applications [17, 138]. The Multi-access Edge Computing (MEC) paradigm uses its widespread mobile (i.e., cellular) network, in collaboration with the remote cloud, to realize the low-latency and high-bandwidth real-time IoT applications [138]. To do so, the Mobile Network Operators (MNOs) provide access of their network devices (gateways, edge servers, and so on) to the IoT applications for running suitable services on their infrastructure. Thus, the MEC-cloud collaboration seems to be a promising solution for enabling real-time smart city IoT applications. However, the decisions for which service to place at which compute node must be taken optimally to satisfy an optimization criteria (minimize latency, energy, and so on) under given constraints (cpu, memory, and so on). Thus, the SPP is formulated and solved as a mathematical optimization problem with optimization criteria and set of constraints [36]. Reinforcement Learning (RL)-based approaches have been proposed in literature to solve the SPP. The RL algorithms can be classified as meta-heuristics, Value-Based (VB), Policy Gradient (PG), and model-based algorithms [140]. The meta-heuristic SPP approaches [141, 142, 143, 144, 145, 146, 147] use a *searching-based* algorithm like Particle swarm optimization (PSO), Ant Colony Optimization (ACO), Non-dominated Sorted Genetic Algorithm-II (NSGA-II), and so on to search the near-optimal solution in the random search space for solving the placement problem. The searching-based algorithms find the near-optimal solution but does not guarantee to find the optimal

solution. However, the *learning-based* algorithms, such as VB, PG, and model-based RL algorithms interact with the stochastic environment and learn from their experience to provide an optimal solution to the placement problem. Furthermore, a recent paradigm named as, Deep RL (DRL) algorithms has emerged that combine RL and Deep Learning to solve the optimal decision-making for problems with large state-space and action-space. Therefore, the service placement for smart city applications must also be solved using DRL algorithms. Thus, many recent works consider *learning-based* DRL algorithms as, VB DRL algorithms [148, 149, 150, 151, 152, 153, 154, 155], PG DRL algorithms [156, 157, 158, 159, 160, 161], and model-based DRL algorithms [162, 163] to solve the SPP.

- **Meta-Heuristic Approaches for Service Placement**
  Deng et al. [141] places the services (i.e., tasks) of mobile users either locally on User Equipment (UE) or the edge computational nodes. This approach assumes that mobile users will pay for the consumed edge resources and proposes a service pricing model. This work proposes 0-1 weights-based improved weighted PSO algorithm to solve the SPP that minimizes latency, energy consumption and price for the user. Ma et al. [142] performs service placement for dependent tasks of Directed Acyclic Graph (DAG) applications in the three-tier user-MEC-cloud architecture. This work proposes a queue-based improved multi-objective PSO that minimizes task completion time and execution cost for multiple DAG applications. Peng et al. [143] applies a Non-dominated Sorted Genetic Algorithm-II (NSGA-II) to optimize energy and cost for workflow applications of mobile users. Xu et al. [144] applies Ant Colony Optimization (ACO) for privacy-preserving service placement in smart city by solving the trade-off between privacy entropy value and placement performance i.e., time, energy consumption and load balance variance. In another work, Xu et al. [145] performs trust-oriented service placement in smart city that determines a privacy conflict set to represent non-trusted edge nodes for a given service, and then, applies Strength Pareto Evolutionary Algorithm (SPEA2) to solve the trade-off between privacy conflicts and various placement metrics. Dubey et al. [146] combines Cuckoo Search Optimization (CSO) and PSO to solve the SPP for fog-cloud environment to minimize application makespan, cost and energy consumption. Jojoa et al. [147] proposes an ACO-based model – MAACO that integrates mobility-awareness and application-priority to minimize latency while dynamically placing services for mobile users running critical applications in a smart city.

- **Value-Based Approaches for Service Placement**

Xiong et al. [148] extends DQN with multiple buffers to optimize completion time and resources requested for placing services of a single IoT application in a single-server MEC. Ale et al. [149] applies DQN to determine the MEC placement server and its cpu frequency to optimize energy consumption and no. of tasks completed for multiple independent deadline-based tasks of many users. Gazori et al. [150] proposes a Double DQN based algorithm for placing independent IoT tasks in a hierarchical fog-cloud network to optimize delay and cost under the resource and deadline constraints. Tang et al. [151] uses DQN for placing dependent services of an application in edge computing to optimize transmission and computation cost. Lu et al. [152] proposes a DQN with LSTM as the last layer to optimize the energy consumption, cost and load balancing for mobile users in multi-server MEC. Panda et al. [153] proposes a DRL based scheme to optimize the IoT devices' energy consumption in edge computing. Chen et al. [154] proposes a Double DQN based algorithm for placing services of mobile users to optimize execution time and energy consumption. Wei et al. [155] uses DQN for placing single deadline-based task from multiple mobile users in a single-server MEC to optimize latency and energy computation.

- **Policy Gradient Approaches for Service Placement**
  Goudarzi et al. [156] uses distributed actors-learners by combining importance weighted actor-learner architecture (IMPALA) with recurrent neural network (RNN) to place dependent services of DAG applications in fog-cloud computing environment. Wang et al. [157] uses sequence-to-sequence neural network to represent binary-offloading policy in MEC, which is trained using Proximal Policy Optimization (PPO) to minimize total latency for all dependent tasks of mobile users. Zheng et al. [158] applies the DQN, PPO and Deep Dterministic Policy Gradient (DDPG) algorithms to minimize total delay and failure rate for placing independent, delay-constrained services of mobile users in MEC and find that DQN outperforms both, PPO and DDPG. Zhan et al. [159] applies PPO to determine sequence of placement actions in vehicular edge computing to minimize latency and energy consumption. Sadiki et al. [160] applies PPO to determine two continuous actions: cpu frequency and user's transmit power for minimizing power consumption and offloading delay in multi-antenna MIMO MEC environment. Liu et al. [161] applies Actor-Critic for placing multiple, independent services of a single mobile user in MEC to minimize the latency and energy consumption.

- **Model-Based Approaches for Service Placement**
  Brogi et al. [162] uses Monte Carlo simulation to estimate QoS assurance under

varying QoS of communication links to place services of a single IoT application in QoS and cost-aware manner. Chen et al. [163] applies Monte Carlo Tree Search (MCTS) to construct the environment model for SPP in MEC for multiple mobile devices, which is then, solved by using DNN to optimize the latency and power consumption.

2.3.2 Analysis of Service Placement in Edge-based Architecture

Service placement using meta-heuristic algorithms generate a random set of policies and determine the best and worst policy. Then, the algorithm converges to a (near) optimal policy using advanced random search or evolutionary approaches. However, the major drawback is that these approaches require the mathematical model of problem at hand, which is not practical for the dynamic MEC environment, where channel conditions keep on varying [149]. The searching-based meta-heuristic approaches assure to find good near-optimal solutions, but the optimal solutions might not be explored in their search strategy. Thus, searching-based meta-heuristic algorithms are not recommended to solve the SPP for dynamic environments like MEC networks, and hence, the dynamic smart city environment.

The VB and PG classes of RL algorithms are called as *model-free* algorithms, as they do not require the model of environment in advance, rather they find the optimal solution by learning from the experience gained by interacting with the stochastic environment. For very large state-space and action-space, the Deep RL (DRL) variants of these algorithms combine the Deep Neural Networks (DNN) with the RL algorithms. Among these, the VB algorithms directly optimize the action-value Q-function i.e., obtain optimal $Q^*(s, a)$, which indirectly optimize the agent policy. Whereas, the PG algorithms directly optimize the policy using policy parameters, which indirectly optimizes the value function (Q-function) [140]. Thus, model-free DRL algorithms (both, VB and PG) have the potential to solve the SPP for smart city applications.

The model-based DRL algorithms operate on the model of environment i.e., either the world dynamics are explicitly given, or they are obtained from outputs corresponding to black-box interactions with the environment [140]. Therefore, the model-based approaches are suitable only for small-scale service placement. For large-scale service placement scenario such as a smart city, these approaches tend to be highly compute intensive in generating large number of diverse samples to simulate the model of the environment with reasonable accuracy. Thus, model-based algorithms are not suitable for SPP for smart city applications.

2.4 Conclusion

This chapter provides an in-depth review of literature for smart city enablement. First, big data management in IoT is reviewed across several application domains, followed by cloud computing support for IoT and IoT big data management. Then, smart city frameworks for big data management are reviewed and the existing smart city frameworks are taxonomized on the basis of framework design and big data management approaches. These smart city frameworks are analyzed to identify research gaps in existing smart city frameworks. Finally, for enabling real-time smart city IoT applications, edge-computing based service placement works from literature are reviewed.

Next chapter presents the design of the proposed smart city framework that considers the integration of heterogeneous IoT and geospatial data of the city.

Chapter 3

Cloud4IoTCity: Proposed Smart City Framework

The previous chapter reviews the existing literature on the subject under study i.e., technologies for enablement of smart city framework. It undertakes to study in detail the IoT, cloud computing, and big data management approaches for handling smart city data. It gave a chance to understand various aspects and issues of the current study. It facilitated the identification of gaps in this research area, such as lack of data fusion approaches to integrate IoT big data with geospatial data of the smart city, inability to accommodate the latest urban data over time, lack of cloud computing technologies, not considering the application requirements for spatio-temporal analysis of urban data, and so on. These research gaps motivate the design of the proposed framework.

This chapter presents the proposed 'Cloud4IoTCity' framework that addresses the identified research gaps for holistic big data management for a smart city. Cloud4IoTCity is a cloud framework based on the fusion of geospatial and IoT big data for spatio-temporal analysis and efficient management of smart city data. Cloud4IoTCity is designed as a layered framework that comprises of four layers namely, storage layer, processing layer, service layer, and application layer. This work is novel and applicable to any smart city IoT dataset, which has space-time qualified sensor readings and also provides the spatial dataset comprising of (longitude, latitude) coordinates for sensor locations.

This chapter is organized as follows: Section 3.1 outlines major design features that the proposed smart city framework must support. Based on these design features, the proposed smart city framework is designed in Section 3.2. Then, Section 3.3 discusses the significance of proposed framework to smart city managers. Section 3.4 discusses how the framework addresses the research gaps. Section 3.5 proposes a taxonomy of spatio-temporal queries supported by the proposed framework. Finally, Section 3.6 concludes the chapter.

3.1 Framework Design Features

This section outlines major design features that must be supported by a smart city big data management framework:

Design Feature 1 (DF1): Integration of Heterogeneous Data
Smart cities generate heterogeneous data. An efficient framework for smart city big data management must perform meaningful integration of multi-source heterogeneous urban data, such as IoT data with disparate urban data (example, geospatial data) to enhance the *value* of insights drawn from data-driven analysis.

Design Feature 2 (DF2): Data Management Service to Add New Data
Smart cities generate new IoT data over time. The smart city frameworks should allow the addition and analysis of latest data, while ensuring data quality.

Design Feature 3 (DF3): Scalable Framework Design
Smart cities are large IoT environments with thousands of sensors generating IoT big data. Thus, distributed big data storage and processing on the scalable cloud infrastructure must be considered for scalable framework design.

Design Feature 4 (DF4): Spatio-Temporal Perspective for Urban Data
Smart city applications are classified into several types based on their application design (Section 1.1.3). The smart city applications present diverse application requirements. For example, government-centric analytics-driven urban planning applications have requirements for urban data analysis in space and time dimensions. Thus, the space-time information within IoT sensing data should be tapped to enable spatio-temporal analysis of urban big data.

3.2 Cloud4IoTCity: Framework Design

In this section, Cloud4IoTCity, an IoT and cloud based smart city framework has been proposed for holistic big data management of smart city data on a cloud platform. The framework considers the integration of IoT big data with geospatial data of the city, enables data analysis from spatio-temporal perspective, and supports design features sketched in Section 3.1. The proposed Cloud4IoTCity framework is shown in Figure 3.1. Cloud4IoTCity is designed as a cloud-centric framework, which makes it accessible across the internet for data ingestion from the smart city infrastructure layer and for delivering knowledge-driven services to the user layer. The framework comprises of four layers: storage layer, processing layer, service layer, and application layer on the cloud as

discussed next.

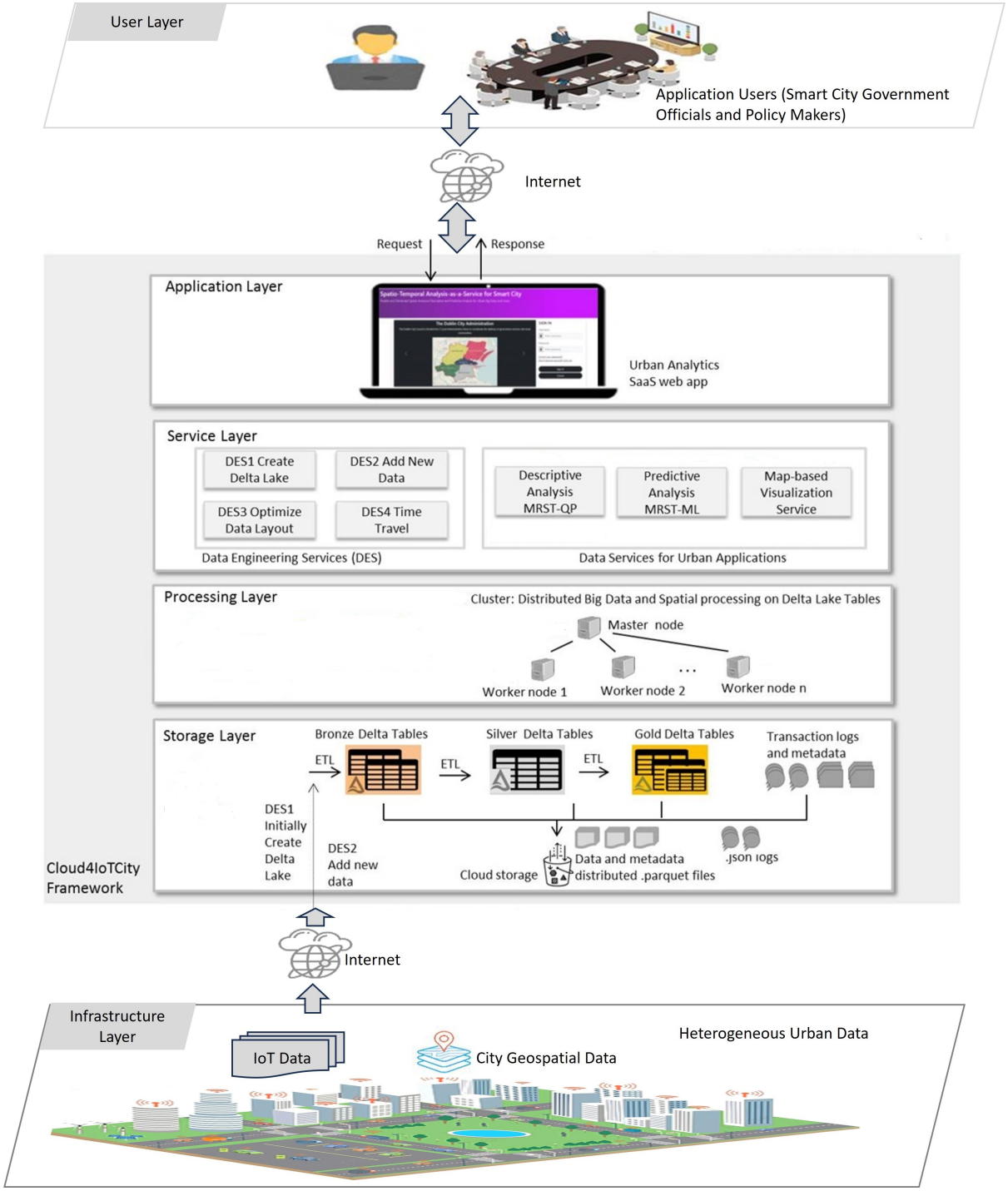


Figure 3.1: The Proposed Cloud4IoTCity Framework

3.2.1 Storage Layer: Lakehouse Storage with Delta Tables

Storage layer in the proposed Cloud4IoTCity framework is based on the recent data lakehouse architecture [130], which combines the benefits of both the data warehouses and data lakes into a single storage architecture. Among data storage architectures, data warehouses store curated and structured data into reliable warehouse storage, which is optimized for structured queries and Business Intelligence (BI). The downside of warehouse storage is that it is not suitable for ML/AI analysis due to lack of support for semi-structured and unstructured big data. Henceforth, data lake storage architecture emerged with the idea to store all sorts of structured, semi-structured, and unstructured big data in their native formats as distributed files on either the cloud object store (eg. AWS S3), or on hadoop file system. The downside of data lakes is that it looses data quality and reliability, which necessitates the use of third-party metadata management tools. Thanks to the recent research, the recent lakehouse storage architecture [130] has emerged, which adds a metadata layer to the data lake storage, such that both heterogeneous data and its metadata are physically stored together as .parquet (an open format) files and logically accessible as tabular storage.

In the proposed framework, storage layer is implemented using Delta Lake [164], which implements the lakehouse storage architecture in the form of high-performance tables that are stored on any cloud object store, such as AWS S3. Delta lake creates three levels of structured tables – bronze, silver, and gold delta tables with increasing levels of data quality and aggregation. Each delta table is physically stored as distributed .parquet files on the underlying cloud storage. In addition, delta lake offers in-built metadata management and data governance. To do so, it maintains transaction logs and checkpoints, maintains dataset versions and updates table's metadata whenever the delta tables are written, overwritten, or appended.

The three types of delta tables are created by executing three ETL pipelines namely, raw-to-bronze, bronze-to-silver, and silver-to-gold respectively, which are discussed ahead in Section 3.2.3.

3.2.2 Processing Layer: Distributed and Parallel Processing on Compute Cluster

Processing layer in the proposed Cloud4IoTCity is based on a compute cluster that follows the master-worker architecture to perform distributed processing of big data in parallel over the worker machines. Apache Spark [165, 166], which is a high-performance cluster-based big data processing engine is used for distributed processing in the proposed

framework. In addition, delta lake supports integration with Spark, which enables the spark cluster to directly access delta tables from the storage layer without any 3rd-party data connectors.

However, spark does not provide native support for handling geospatial data, which is available in spatial formats like .geojson, .shapefile, .kml, and so on. The geospatial data encodes the (longitude, latitude) information in the form of spatial objects, such as point, line, polygon, multi-polygon, and many more with geometry datatype. The proposed framework relies on geospatial data of the smart city. Thus, a cluster-based spatial processing engine is required that can be integrated with spark. Therefore, Apache Sedona (formerly GeoSpark) [167, 168, 169], which is built on top of spark for distributed spatial processing on a compute cluster is used by the proposed framework. Moreover, the superior performance of Apache Sedona for spatial processing relative to its counterparts is demonstrated experimentally in earlier works [170, 43].

To sum up, processing layer in the Cloud4IoTCity framework comprises a compute cluster with open-source softwares: Delta Lake, Apache Spark, and Apache Sedona (abbreviated as DSS cluster in rest of the text). The DSS cluster is able to directly access the delta tables of the storage layer, which store the raw and fused geospatial and IoT big data of the smart city to perform big data and spatial processing using Spark and Sedona APIs.

3.2.3 Service Layer: Data Engineering and Data Processing Services

Service layer in the proposed framework implements two kinds of data services:

i) Data Engineering Services (DES): these services deal with creation, updation, governance and management of storage layer data. DES services include data fusion, add new data, optimize data layout, and dataset versioning for time travel

ii) Data Processing Services: these services deal with analysis and visualization of the data. These include spatio-temporal descriptive analysis, predictive analysis, and map-based visualization.

Data Engineering Services

These services are implemented as ETL pipelines to create, update, manage and govern the data in storage layer of the framework. The proposed framework provides four DES services as follows:

- DES1: Fusion of Geospatial and IoT Big Data into Structured Delta Lake Tables

- DES2: Add New IoT Data
- DES3: Data Layout Optimization
- DES4: Dataset Versioning and Time Travel

DES1: Fusion of Geospatial and IoT Big Data into Structured Delta Lake Tables

This service takes as input the raw heterogeneous geospatial and IoT data to create the storage layer delta tables. For creating the bronze, silver, and gold delta tables, three ETL pipelines, namely, *raw-to-bronze, bronze-to-silver, and silver-to-gold* are used respectively.

The ***raw-to-bronze*** pipeline reads each type of urban data, i.e., the geospatial and IoT data in their native formats, such as .csv, .json, .geojson, .shapefile, and so on into Spark's Structured DataFrames and writes them as individual bronze delta tables without doing any pre-processing or data cleaning.

The ***bronze-to-silver*** pipeline reads all the bronze delta tables to perform data cleaning, datatype transformation, co-ordinate system transformation, and performs spatial and non-spatial data joins (spatial containment joins, joining non-spatial data with spatial data) to write a single silver delta table, which is of high quality than the bronze tables.

The ***silver-to-gold*** pipeline reads the silver delta table and performs data aggregation based on common query columns to enable quick retrieval for most of the business queries. In the proposed framework, the analysis will be done from spatio-temporal perspective. Therefore, few commonly used pairs of space-time scales are used as aggregation columns to write multiple space-time aggregated gold tables that represent high-quality aggregated business data.

Note that, as part of in-built metadata management, delta lake assigns version number 0 to all three types of delta tables upon their successful creation using this DES1 service.

DES2: Add New Data

This service enables the proposed framework to add the recent smart city IoT data generated over time. This service ensures that only quality data can be added into existing storage layer tables by performing a schema validation test. The schema validation test checks if the schema of the input file(s) (i.e., new data) matches with the schema of the bronze table, only then does it proceed to apply the corresponding ETL steps to append this new data into the three types of delta tables. Otherwise, it gives a schema mismatch

error and does not update any of the delta tables.

Note that delta lake automatically updates the transaction log, delta tables' versions (previous version number + 1), and their metadata whenever new data is successfully added to the delta tables.

DES3: Data Layout Optimization
This service optimizes the layout for a delta table, which is stored physically as distributed .parquet files on the cloud storage. Delta lake supports z-ordering to optimize the layout for a delta table by re-arranging its data within small .parquet files on the basis of a specified z-ordering column. This layout optimization uses the value of z-ordering column to pick up matching records from separate files and store them together in a same file.

Z-ordering a given delta table is preferred for a column that has many distinct values and occurs frequently in query predicates (for example, site_id for a sensor). For instance, z-ordering on the site_id column will collect rows for site=1 distributed across several small files and place them together into a file, and a similar re-arrangement is done for all values of z-ordering column. This co-location of similar information enables data-skipping algorithms to skip many (.parquet) files while searching for site_id value(s), which results in faster query retrieval. This layout optimization service can be run after a write or an append operation on delta tables, i.e., after DES1 or DES2.

DES4: Dataset Versioning and Time Travel
This service leverages delta lake's in-built dataset versioning and time travel feature. The write, append, and optimize operation on a delta table creates a new version of the delta table. Thus, the services DES1, DES2, and DES3 create newer versions of delta tables, which is reflected in their metadata. The versions' metadata information includes version_id, version_ timestamp, user metadata description, data operation, engine information, and so on for all the versions of delta tables.

A read operation on the delta table retrieves its most recent version by default. However, delta lake supports time travel to any specific version by specifying either the version_id or version_timestamp. Time travel to a specified version enables the retrieval of an older snapshot of urban data in the proposed framework. In the proposed framework, version-specific data analysis services are supported that analyze data w.r.t retrieved snapshot of the data.

Data Processing Services

These services perform data analysis and generate map-based visualization for analysis results. The fusion of geospatial and IoT big data in the proposed framework enables

data analysis from a spatio-temporal perspective. The proposed framework provides three services for data analysis and visualization as follows:

- Descriptive Analysis Service
- Predictive Analysis Service
- Map-based Visualization Service

Descriptive Analysis Service
This service performs descriptive analysis of data in delta tables using structured queries for retrieving urban attribute information from spatio-temporal perspective. In the proposed framework, this service is named as Multi-Resolution Spatio-Temporal Query Processing (MRST-QP). The MRST-QP service allows the end-user, i.e., smart city managers to run fifteen types of spatio-temporal queries for urban data analysis. This taxonomy is proposed in this work and is explained in detail in Section 3.5.

The choice for various space-time scales depends on the smallest scale of space-time in the given smart city IoT dataset, as higher resolution space-time scales can be derived from lower-resolution column values by applying suitable operations. For example, date, week, month, and other higher time-scales can be derived from the datetimne (i.e., timestamp) column. However, spatial resolutions depends on the available geospatial data resolutions. For example, small area neighborhoods, large area admin zones, and entire city represent three different resolutions on space scale. The spatial join functions, such as ST_CONTAINS, ST_WITHIN from Sedona API can be used to map the sensor (longitude, latitude) points with higher-resolution spatial areas. In this way, the proposed framework generates multiple space-time scales from the given geospatial and IoT datasets.

Since delta lake stores the fused urban big data as structured delta tables, Spark SQL – the distributed SQL Query Engine of Spark for structured data retrieval is used to implement the descriptive analysis service in the proposed framework. A Spark SQL session on a DSS (Delta Lake, Spark, Sedona)-enabled compute cluster can run Spark+Sedona SQL queries on the delta tables. The DSS cluster distributes the delta table's .parquet files among its worker nodes for distributed query processing. Both Spark and Sedona offer Python APIs, so PySpark script files are written to implement the business logic for multi-resolution spatio-temporal queries for each of the fifteen queries for smart city taxonomized in section 3.5.

Predictive Analysis Service
This service runs the ML models to predict the urban attribute for chosen spatial site(s) for a futuristic time. Such a spatio-temporal predictive analysis is based on the ML

models trained per-site for the urban data. In the proposed framework, this service is named as Multi-Resolution Spatio-Temporal Machine Learning (MRST-ML) that allows the end-user, i.e., smart city manager to run per-site trained ML models to predict the urban phenomena for a future time.

The choice of space-time scales, whether fine-grained or coarse-grained scales to use for prediction, depends on the nature of urban data i.e., the smart city use case. For instance, fine-grained predictions, such as per-site hourly/ daily predictions, are suitable for highly dynamic domains such as traffic data, and coarse-grained predictions, such as area-wise monthly/quarterly predictions, are ideal for less dynamic domains such as air quality monitoring data.

It is worth noting that training the ML models for all sites of the city is compute-intensive compared to running the trained model for predictions. Thus, to train a large number of ML models for per-site predictions, the training of ML models is done in a distributed manner using Pandas User Defined Function (UDF). When the smart city manager user uses this MRST-ML service to make predictions for the chosen sites, the corresponding trained models of that site are loaded and run to give predictive analysis results to the user.

Map-based Visualization Service

The proposed framework is designed for smart city domain, hence, the analysis results for both, descriptive and predictive analysis are presented in the form of map-based visualization. The proposed framework uses Plotly Express [171] python library to create interactive map-based visualizations. These interactive maps support pan, zoom-in/out, and save features.

In the proposed framework, scattered mapbox plots are used for site-level plotting to show spatial points and choropleth plots are used for for area-level plotting to show spatial polygons. The visualization service displays a single map-based plot for an instant of time (any scale) and displays an animation of time frames for a range of time. When the descriptive or predictive analysis, i.e., MRST-QP or MRST-ML services are ready with their analysis results, they call this service to render the map-based plots along with the textual-tables to present the analysis results to the user.

3.2.4 Application Layer: Urban Analytics SaaS Web App

Application layer of the proposed framework enables the development of applications for spatio-temporal analysis of the urban data on the basis of services from the service layer. The proposed framework proposes a cloud-based software-as-a-service (SaaS) application

for smart city government managers, officials and policy makers to analyze the smart city IoT data using MRST-QP service for spatio-temporal descriptive analysis and MRST-ML service for spatio-temporal predictive analysis.

3.3 Significance of the Proposed Smart City Framework

IoT-based smart city applications generate *geolocated and timestamped* sensor readings for the measured urban attribute like traffic volume, air quality, noise levels, and many more. Such smart city IoT data has space and time information, such that the sensor's (longitude, latitude) location is the spatial information and the timestamp of a reading is the temporal information. Such spatio-temporal information about the measured urban attribute paves the way for spatio-temporal analysis of smart city IoT data.

Spatio-temporal analysis of smart city IoT data can unfold useful data-driven insights, such as analyzing trends and behavior in urban data, detecting epicenters of certain phenomena, observing spatial distribution of urban phenomena for a specific instant or period of time, summarizing urban data over space-time horizon, predicting urban phenomena for a site in futuristic time, and many others. Such spatio-temporal insights are useful for smart city managers, government officials, city planners, and policymakers for making policy design and urban planning decisions.

Next section discusses how the proposed framework outshines existing smart city frameworks by addressing the research gaps found in current smart city frameworks for big data management.

3.4 Mapping of the Designed Framework with Identified Research Gaps

The designed Cloud4IoTCity smart city framework addresses the research gaps for smart city big data management identified in the section 2.2.3 as follows:
RG1 *No Fusion of IoT data with City's Geospatial Data*: The proposed framework considers the fusion of smart city IoT data and the geospatial data of the city
RG2 *Non-Cloud Architecture*: Cloud4IoTCity is designed and implemented as a *cloud-based framework*
RG3 *Scalability*: The proposed framework performs distributed storage and distributed processing in the cloud to address the scalability issue for urban big data

RG4 *Inefficient Storage Architecture*: The proposed framework implements the *efficient Lakehouse architecture* [130] for data storage, which offers the combined benefits of data lakes and data warehouses with other add-on features
RG5 *Add/Store New Data*: New data can be easily added into our framework with schema validation and data versioning
RG6 *Limited Analytics Types*: The proposed framework supports various analytics services, such as data engineering service and data processing services. Among these, data engineering comprises Extract-Transform-Load (ETL) pipelines for the framework's internal tasks, such as storage layer creation and other data management features, while data processing services include data-driven analysis and visualization services to support application development. Data processing services in the proposed framework include: descriptive analysis, predictive analysis, and visualization services.
RG7 *Spatio-Temporal Perspective & Queries Taxonomy*: A *taxonomy of 15 spatio-temporal queries* for smart city applications is proposed in section 3.5 and is used to perform the descriptive analysis of urban IoT data from space-time perspective
RG8 *Multi-Resolution Analysis*: The proposed 15 spatio-temporal queries are implemented for *multiple space-time resolutions*
RG9 *Data Quality & Governance*: Major advantange of lakehouse architecture over traditional data lakes is its built-in features for *efficient metadata management and data governance*. The lakehouse architecture stores the transactional metadata along with the large scale big data to ensure data quality, metadata management, and governance
RG10 *Data Versioning & Time Travel*: Data versioning and time travel is an important feature offered by delta lake storage architecture [164], which is used by the proposed framework for version-specific urban data analysis.

Next section presents the proposed taxonomy for spatio-temporal queries for smart city applications. These queries are used by Cloud4IoTCity framework for descriptive analysis of past urban data using query-based retrieval.

3.5 Proposed Taxonomy of Spatio-Temporal Queries for Smart City Applications

A taxonomy of spatio-temporal queries for smart city applications is proposed in this section, as there exist a knowledge gap in literature regarding standard list of baseline spatio-temporal queries for smart city domain. The proposed taxonomy is adapted from a taxonomy of eleven query types for spatio-temporal data retrieval given by Yuan et al. [172]. This taxonomy [172] is for generic applications and does not consider the smart

city analysis requirements. Hence, this taxonomy is enhanced with respect to smart city managers analysis needs. Therefore, a novel taxonomy of fifteen spatio-temporal queries for smart city applications is proposed and is shown in Figure 3.2. The proposed taxonomy categorizes fifteen queries into three types: five spatial, four temporal, and six spatio-temporal queries. Each of these query types with multi-resolution query examples and their significance for smart city analytics are summarized in Table 3.1.

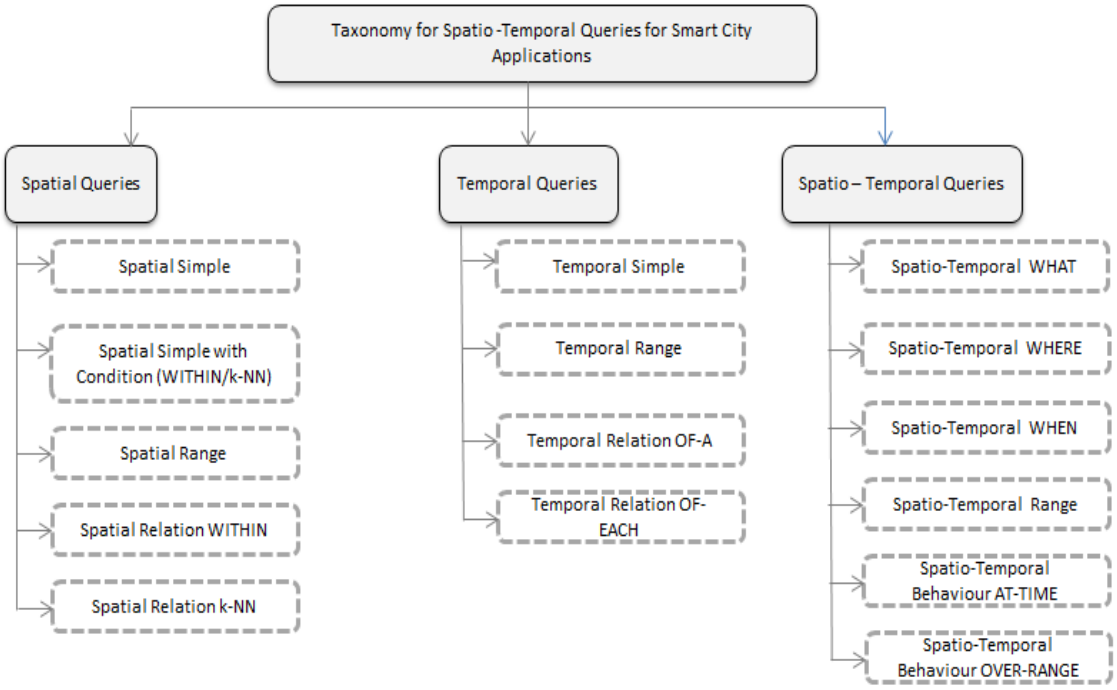


Figure 3.2: Taxonomy for Spatio-Temporal Queries for Smart City Applications

3.5.1 Spatial Queries

Spatial queries apply spatial processing operations like spatial object (point, polygon, and so on) retrieval, k-NN (k-nearest neighbor), spatial range, and many more. The spatial queries for smart city domain include:

i) Spatial Simple: the basic address retrieval query to *retrieve the points or polygon spatial objects*

ii) Spatial Simple with Condition: *spatial objects retrieval based on evaluation of a spatial-condition* (`WITHIN` or `k-NN`). If the spatial condition is WITHIN then, retrieve spatial points that are located `WITHIN` the selected polygon objects else, if the spatial condition is k-NN then, retrieve 'k' number of spatial points, which are nearest neighbors of the input query point

iii) Spatial Range: retrieve *aggregated attribute (e.g., traffic volume) information over*

spatial polygons to get the statistical overview over range of spatial addresses
iv) Spatial Relation WITHIN: retrieve *attribute information for those spatial points, which lie WITHIN selected polygon objects.* This query involves solving the spatial containment join for getting the contained points and retrieving attribute information for those points
v) Spatial Relation k-NN: retrieve *attribute information for those k spatial points, which are closest neighbors of the input query point*

To summarize, the first two query types help the smart city officials get information on spatial addresses (points/polygons) responsible for generating data in their application; then, the spatial range helps to get a statistical attribute summary over spatial polygons helpful in making district or area-wise comparisons and decisions. Finally, the last two spatial relation queries provide fine-grained per-site attribute information, either w.r.t WITHIN relation, which is helpful for district/area managers to get per-site information for their districts/areas, or w.r.t k-NN relation, which helps in querying the attribute information closest to a hotspot or POI in the city.

3.5.2 Temporal Queries

Temporal queries analyze urban data from the temporal dimension without applying any spatial processing functions. The temporal queries for smart city domain include:
i) Temporal Simple: the basic *attribute retrieval for chosen spatial sites at any time scale* (temporal scales: datetime, date, week, month, quarter, year, all)
ii) Temporal Range: retrieve *aggregated attribute information for spatial sites' during a range of time*, wherein the querying user defines the time range on an hourly/ daily/ weekly/ monthly/ quarterly/ yearly basis to support multiple time resolutions. This query captures the statistical attribute information for the spatial sites *at each time frame* during the chosen range of time
iii) Temporal Relation OF-A: retrieve *attribute information for spatial objects for a range of time instances, which are part OF-A bigger time moment*, for example, retrieve attribute information for chosen spatial objects (points/ polygons) for 08:00 to 18:00 Hours OF-A Day 2020-12-30
iv) Temporal Relation OF-EACH: retrieve *attribute information for spatial objects for a range of time instances, which are part OF-EACH bigger time scale*, for example, retrieve attribute information for chosen spatial objects (points/ polygons) for Mon to Fri OF-EACH Month

To summarize, the temporal queries enable smart city officials to analyze the temporal dimension of urban data, such that the first query provides *temporal snapshot* of attribute

information at any time resolution. In contrast, the range query displays an *animation of temporal snapshots* for fine-grained site-level statistical attribute information. Furthermore, the animation of temporal snapshots may involve heterogeneous time scales, which implies solving the complex temporal relationships such as *OF-A, OF-EACH* in choosing a smaller time slice taken from a bigger time slice, such as working hours (8 AM to 6 PM) *of-a* week, all weeks *of-a* quarter, weekends (Sat to Sun) *of-each* month, and many more and that too for either fine-grained spatial sites or coarse-grained spatial polygons. The last two temporal relation queries address such querying needs for urban data analysis.

3.5.3 Spatio-Temporal Queries

Spatio-temporal queries analyze the urban data from spatial and temporal dimensions with multiple resolution scales for both dimensions and may involve spatial processing. These queries include:

i) Spatio-Temporal WHAT: basic *attribute retrieval for spatial objects for temporal value at any chosen space and time resolution*

ii) Spatio-Temporal WHERE: retrieve *spatial objects that satisfy the given temporal and attribute conditional-threshold criteria.* This query retrieves information about spatial objects, i.e., WHERE (spatial scales: points/ polygons) the attribute value satisfies the chosen threshold condition $(>, \geq, <, \leq, =)$ for the chosen time scale

iii) Spatio-Temporal WHEN: retrieve *time instances that satisfy the given spatial and attribute conditional-threshold criteria.* This query retrieves temporal information, i.e., WHEN (DateTime, date, week, month, quarter, year) the attribute value satisfies the chosen threshold condition $(>, \geq, <, \leq, =)$ for the chosen spatial scale

iv) Spatio-Temporal Range: retrieve *aggregated attribute information over range of both space and time* i.e. get animation of statistical attribute information over time range for chosen spatial polygons

v) Spatio-Temporal Behaviour AT-TIME: retrieve *statistical attribute pattern for spatial objects at chosen time* for any space-time resolution

vi) Spatio-Temporal Behaviour OVER-RANGE: retrieve *statistical attribute pattern for spatial objects for a range of time* for any space-time resolution

To summarize, the first three spatio-temporal queries enable the smart city officials to query the attribute (`what`) information, location information `where` the attribute satisfies the threshold condition, and time information `when`, the attribute satisfies the threshold condition, respectively. Next, the spatio-temporal range query provides a temporal animation of spatial polygons, which is quite helpful in assessing district or area-wise

policies/decisions over a range of time. Finally, the last two behavior queries summarize the urban dynamics (attribute), either as a temporal snapshot (AT-TIME) or as an animation of temporal snapshots (OVER-RANGE).

These 15 taxonomized queries form the basis for multi-resolution spatio-temporal queries for descriptive analysis in the proposed Cloud4IoTCity framework. Each of these fifteen query types with multi-resolution query examples and their significance for smart city analytics are summarized in Table 3.1.

Table 3.1: Description of Proposed Spatio-Temporal Queries for Smart City Data Analysis

Query Type	**Description**	**Multi-Resolution Query Examples**	**Significance for Smart City Analytics**
Spatial Simple	Multi-resolution Spatial Location(s) (`Points`, or `Polygons` spatial objects) Information *Points: Sites* *Polygons: District/ Area/ City*	1. Get s1, s210, s450 Sites 2. Get D1, D2, D6 Districts 3. Get North West, Central, and South East Areas 4. Get ALL Areas	Know the sensor locations in the city or the city's spatial areas
Spatial Simple with Condition	Spatial Point(s) information either WITHIN spatial polygons or NEAREST to a point-of-interest (POI) in the city *Polygon: District/ Area/ City* *POI: (long,lat)*	`Within:` 1. Get Sites WITHIN D2, D6, D6W districts 2. Get Sites WITHIN Central, South Central Areas 3. Get Sites WITHIN City `k-NN:` 1. Get 10 Nearest Neighbour Sites from (-6.2341, 53.3124)	IoT Devices/ Sensors' Spatial Sites Information based on a spatial condition
Spatial Range	Statistical attribute (eg. Traffic) summary over spatial polygon(s) at given timestamp (datetime)	1. Get agg. traffic for D2, D5, D6W Districts at 2020-12-25 12:00:00 time 2. Get agg. traffic for ALL AREAS at 2021-01-01 09:00:00 time 3. Get agg. traffic for city at 2022-04-05 20:00:00 time	Spatial Distribution analysis of urban phenomena (i.e., the attribute eg. traffic volume) for area-wise evaluations and comparisons
Spatial Relation WITHIN	Fine-grained attribute information for sites WITHIN polygon(s) for any time scale *(datetime, date, week, month, quarter, year etc.)*	1. Get traffic for sites WITHIN D8 district at 2020-12-25 date 2. Get traffic for sites WITHIN North Central Area for the week 2022-W52 3. Get traffic for sites WITHIN ALL districts for the month 2022-12	District/ Area/ City managers get fine-grained Per-Site Attribute information for Sites in their domain
Spatial Relation k-NN	Attribute information for sites nearest to a POI in the city for any time scale *(datetime, date, week, month, quarter, year etc.)*	1. Get traffic for 20 nearest neighbour sites from (-6.4356, 53.3214) at 2020-12-25 date 2. Get traffic for 8 nearest neighbour sites from (-6.3214, 53.3124) for the week 2022-W52	Attribute Information Spread for k Nearest Sites from a POI
Temporal Simple	Fine-grained Attribute Information Snapshot for Site(s) for any time scale *(datetime, date, week, month, quarter, year etc.)*	1. Get traffic for site s110 at 2020-12-25 09:00:00 time 2. Get traffic for sites s243, s343, s540 for the date 2022-02-28 3. Get traffic for ALL sites for the quarter 2022-Q3	Temporal Snapshot for Fine-grained Sites at chosen time

Continued on next page

Table 3.1 – *Continued from previous page*

Query Type	**Description**	**Multi-Resolution Query Examples**	**Significance for Smart City Analytics**
Temporal Range	Summarized Attribute information (sum, avg, min, max) for Site(s) at time instants over range of time *Time Range Scales: datetime, date, week, month, quarter, year*	1. Get traffic for site s110 from 2020-12-25 08:00:00 to 2020-12-25 06:00:00 2. Get traffic for sites s243, s546 from 2022-01-01 to 2022-01-15 3. Get traffic for ALL sites from 2021-W40 to 2021-W52	Temporal Snapshots Animation for site-level Attribute Summaries over Range of Time
Temporal Relation OF-A	Attribute information for selected spatial object(s) involving heterogeneous time scales such as, smaller time slice OF-A bigger time instant *1. HOURS OF-A Day/ Week/ Month/ Quarter/ Year* *2. DAYS OF-A Week/ Month/ Quarter/ Year* *3. WEEKS OF-A Month/ Quarter/ Year* *4. MONTHS OF-A Quarter/ Year* *5. QUARTERS OF-A Year*	1. Get traffic for sites s220, s532 for 08:00 to 10:00 (morning) Hours OF-A day 2022-12-25 2. Get traffic for D2,D6,D6W districts for Monday to Friday Days OF-A Quarter 2021-Q3 3. Get traffic for ALL Areas for Weekdays Week OF-A Quarter 2020-Q4 4. Get traffic for ALL Sites for All Weeks OF-A Month 2020-02 5. Get traffic for City for ALL Quarters OF-A Year 2021	Temporal Snapshots for a smaller scale time slice 'of a' bigger scale temporal value
Temporal Relation OF-EACH	Attribute information for selected spatial object(s) involving heterogeneous time scales such as, smaller time slice OF-EACH bigger time scale *1. HOURS OF-EACH Day/ Week/ Month/ Quarter/ Year* *2. DAYS OF-EACH Week/ Month/ Quarter/ Year* *3. WEEKS OF-EACH Month/ Quarter/ Year* *4. MONTHS OF-EACH Quarter/ Year* *5. QUARTERS OF-EACH Year*	1. Get traffic for sites s220, s532 for 08:00 to 10:00 (morning) Hours OF-EACH day 2. Get traffic for D2, D6, D6W districts for Monday to Friday Days OF-EACH Quarter 3. Get traffic for ALL Areas for Weekdays Week OF-EACH Quarter 4. Get traffic for ALL Sites for All Weeks OF-EACH Month 5. Get traffic for City for ALL Quarters OF-EACH Year	Temporal Snapshots for a smaller scale time slice 'of each' bigger time scale
Spatio-Temporal WHAT	Multi-resolution space-time attribute (e.g. traffic) information *Space scales: Site/ District/ Area / City* *Time scales: Datetime, Date, Week, Month, Quarter, Year, All*	1. Get traffic for sites s24, s67, s156 at 2020-12-25 15:00:00 time 2. Get traffic for All sites for 2022-06-30 date 3. Get traffic for D2, D6, D8 districts for week 2021-W34 4. Get traffic for City for all time	Fine-to-Coarse grained WHAT information i.e., attribute information for chosen spatial and temporal values
Spatio-Temporal WHERE	Retrieve WHERE i.e., Spatial object(s) (`Points`, or `Polygons`) based on attribute threshold condition $(<, \leq, >, \geq, ==)$ for chosen temporal value *Time Scales: Datetime, Date, Week, Month, Quarter, Year*	1. Get sites with traffic < 1000 at time 2020-12-15 09:00:00 2. Get districts with traffic ≥ 50000 for date 2022-01-01 3. Get areas with traffic $== 8000000$ for week 2021-W52	Get Spatial Hotspots in the City based on Attribute Intensity Condition at the chosen time value

Continued on next page

Table 3.1 – *Continued from previous page*

Query Type	**Description**	**Multi-Resolution Query Examples**	**Significance for Smart City Analytics**
Spatio-Temporal WHEN	Retrieve WHEN i.e., Temporal instants (for chosen time scale) that satisfy an attribute threshold condition criteria ($<, \leq, >, \geq, ==$) for given spatial object(s)	1. Get datetimes when was traffic < 1000 at sites s13, s456 2. Get dates with traffic ≥ 50000 for districts D1, D6, D9 3. Get weeks with traffic == 8000000 for areas Central, South East 4. Get months with traffic >= 1500000 for City	Get Important Time Instances when the Attribute Threshold Condition is satisfied at chosen spatial objects
Spatio-Temporal Range	Summarized aggregate (avg, sum, min, max) attribute information for range of both, space and time *Space Scales: Polygons (District/ Area/ City)* *Time Scales: Datetime, Date, Week, Month, Quarter, Year, ALL*	1. Get traffic for districts D2, D6, D8 from 2020-12-25 08:00:00 to 2020-12-25 06:00:00 2. Get traffic for ALL districts from 2022-01-01 to 2022-01-15 3. Get traffic for ALL areas from 2021-W40 to 2021-W52	Coarse-grained Temporal Snapshots Animation of Attribute Summaries for Spatial Polygons over range of time
Spatio-Temporal Behaviour AT-TIME	Attribute summary i.e., behaviour for chosen spatial object(s) at chosen time instance	1. Get traffic behaviour for ALL sites for the week 2022-W03 2. Get traffic behaviour for D1,D5,D7 districts for date 2021-10-30 3. Get traffic behaviour for Central area for the month 2020-11 4. Get traffic behaviour for City for the quarter 2021-Q4	Fine-to-Coarse grained Attribute Summary Statistics for Spatial Objects at Point of Time
Spatio-Temporal Behaviour OVER-RANGE	Attribute summary i.e., behaviour for chosen spatial object(s) over range of time	1. Get traffic behaviour for ALL sites during the weeks 2022-W03 to 2022 W15 2. Get traffic behaviour for D1, D5, D7 districts during the days 2021-10-10 to 2021-10-30 3. Get traffic behaviour for Central area during the months 2020-05 to 2020-11 4. Get traffic behaviour for City during the quarters 2021-Q2 to 2021-Q4	Fine-to-Coarse grained Attribute Summary Statistics for Spatial Objects over Range of Time

3.6 Conclusion

This chapter presents Cloud4IoTCity, a cloud based framework for IoT big data management for a smart city. The proposed framework considers the integration of heterogeneous urban data viz. IoT big data and multi-resolution geospatial datasets of the city. The proposed framework provides data engineering services namely, data fusion, add new data, data layout optimization, and time travel for efficient management of the urban big data. In addition, the framework provides data processing services namely, descriptive analysis, predictive analysis and map-based visualization for developing user-centric applications based on the framework.

The next chapter discusses the implementation and validation of the proposed framework through a case study on Urban Traffic Analysis for Dublin City. The case study uses real IoT dataset from Dublin along with multiple geospatial datasets of the city to facilitate their integration towards multi-resolution spatio-temporal analysis of IoT data.

Chapter 4

Validation of the Proposed Framework

Previous chapter presented design of Cloud4IoTCity smart city framework. The proposed framework will assist the smart city government users in policy making and urban planning based on insights drawn by multi-resolution spatio-temporal analysis of the urban data. It also presented a taxonomy of spatio-temporal queries that is used for descriptive analysis by the proposed framework. The proposed framework and queries taxonomy are used as a basis for implementation and validation of the proposed framework in this chapter.

This chapter validates the proposed framework through a Case Study. The case study uses traffic volume data from sensors for Dublin city. The case study implementation fuses the IoT traffic volume data with Dublin's geospatial data at multiple spatial resolutions that are Dublin Districts (small postal areas), Dublin Admin Areas (city administrative zones), and Dublin city to enable multi-resolution spatio-temporal analysis, whose map-based results are presented in this chapter. This chapter also presents experimental results regarding distributed processing and z-order indexing for all queries across different cluster sizes.

Chapter organizations is as: Section 4.1 describes about implementation of the proposed framework through case study with details on datasets. Section 4.2 discusses implementation of each layer of the proposed framework and Section 4.3 presents experiment results for framework validation and finally, Section 4.4 concludes the chapter.

4.1 Framework Implementation through Case Study

This chapter implements the proposed Cloud4IoTCity framework through a case study on urban traffic analysis for Dublin city. The details of the case study are as follows:

4.1.1 Datasets Used

Following datasets are used to implement the undertaken case study on Urban Traffic Analytics for Dublin City:

i) **IoT Dataset:** this work uses the SCATS Traffic Volumes data from Dublin City [173, 174]. The Sydney Coordinated Adaptive Traffic System (SCATS) is an intelligent transportation system for intelligent traffic light management implemented in Dublin. The SCATS technology installs inductive-loop vehicle sensors embedded in-roads to detect vehicle presence at more than 500 sites in the city. The SCATS system pre-processes the obtained sensor readings to hourly readings for traffic volume across all sites and publishes the dataset as open data on the government portals [173, 174]. The SCATS IoT data for each month is available as a structured .csv file containing spatio-temporal sensor reading (i.e., site_id of the sensor, hourly timestamp, traffic volume value). Traffic volume data from 2020 onwards is used to implement the case study. Specifically, three years of data from Jan 2020 to Dec 2022 is used for DES1 service to create the initial Delta Lake tables. Later, as new data for the year 2023 was available, it was gradually added the incremental urban data using the DES2 service, creating newer versions of Delta Lake tables.

ii) **IoT Sensors' Location Dataset:** This dataset gives the spatial geometry (longitude, latitude) points for sensors' locations. This spatial dataset for SCATS sensor locations is available in many formats: .csv, .geojson, and .kml [175]. The site_id, site_address_description, (longitude, latitude) geometry attributes in this locations dataset identify a sensor location. In this case study, the .geojson spatial format is used to work with the geometry datatype in spatial formats.

iii) **Geospatial Datasets of the Smart City:** The fusion of city geospatial datasets with IoT data widens the scope of spatial dimension for spatio-temporal urban data analysis. Usually, the target city can provide the geospatial data at multiple spatial scales, such as small area neighborhoods, large area admin zones, and the entire city. For Dublin city, multiple spatial scales considered are as i) Dublin's Postal Districts, which represent small area neighborhoods for the city and follow an Eircode-based addressing system for the city's postal service, such as Dublin1

(D1), Dublin2 (D2), up to Dublin 24 (D24) [176], ii) Dublin Administrative Areas represent the five large area zones for city that the city government defines for city administrative purposes [177], and iii) the entire city for city-level analysis. The Dublin's Postal District geospatial dataset in shapefile format at [178] and the Dublin Administrative Areas geojson dataset provided by the Dublin city council at [179] were available for use in the case study. However, the spatial dataset for Dublin city could not be found, so this work applies the spatial union function of Sedona ST_UNION on the five admin areas in geojson [179] to get the spatial polygon geometry to represent the entire Dublin city. Thus, this case study offers four scales for the spatial dimension of the city: (longitude, latitude) points as sensor sites, Dublin Postal Districts, Dublin Admin Areas, and the entire Dublin city for multi-resolution spatio-temporal urban analysis in the undertaken case study.

4.2 Implementation of the Case Study

This section discusses the layer-wise implementation of the proposed Cloud4IoTCity framework for the undertaken case study.

4.2.1 Cloud4IoTCity Storage Layer: Fusion of Geospatial and IoT Data into Delta Lake Tables

The DES1 service implements the framework storage layer by fusing the geospatial and IoT big data to create Delta Lake tables on the cloud object store (AWS S3). The DES1 service comprises three ETL pipelines: raw-to-bronze, bronze-to-silver, and silver-to-gold, which run on the DSS cluster. Figure 4.1 shows the detailed steps in these three ETL pipelines for the case study on Urban Traffic Analysis for Dublin City.

As shown in Figure 4.1, in the **raw-to-bronze** pipeline, the non-spatial urban traffic IoT dataset is loaded into a structured spark dataframe. After that, the Sedona spatial format readers are used to load Dublin's spatial data, i.e., traffic sensor locations, Dublin Postal Districts, and Dublin Admin Areas, into spark dataframes. Next, the delta-spark integration enables writing each spark dataframe as a bronze delta table. Note that the path of the AWS S3 bucket is specified to store the delta tables on the cloud.

Then, in the **bronze-to-silver** pipeline, the bronze delta tables are loaded into the DSS spark session and perform data pre-processing, enrichment, and fusion of spatial data with non-spatial data to save the resulting processed data as a single silver delta table. As shown in Figure 4.1, step 2 of this ETL performs data cleaning to remove unwanted rows/

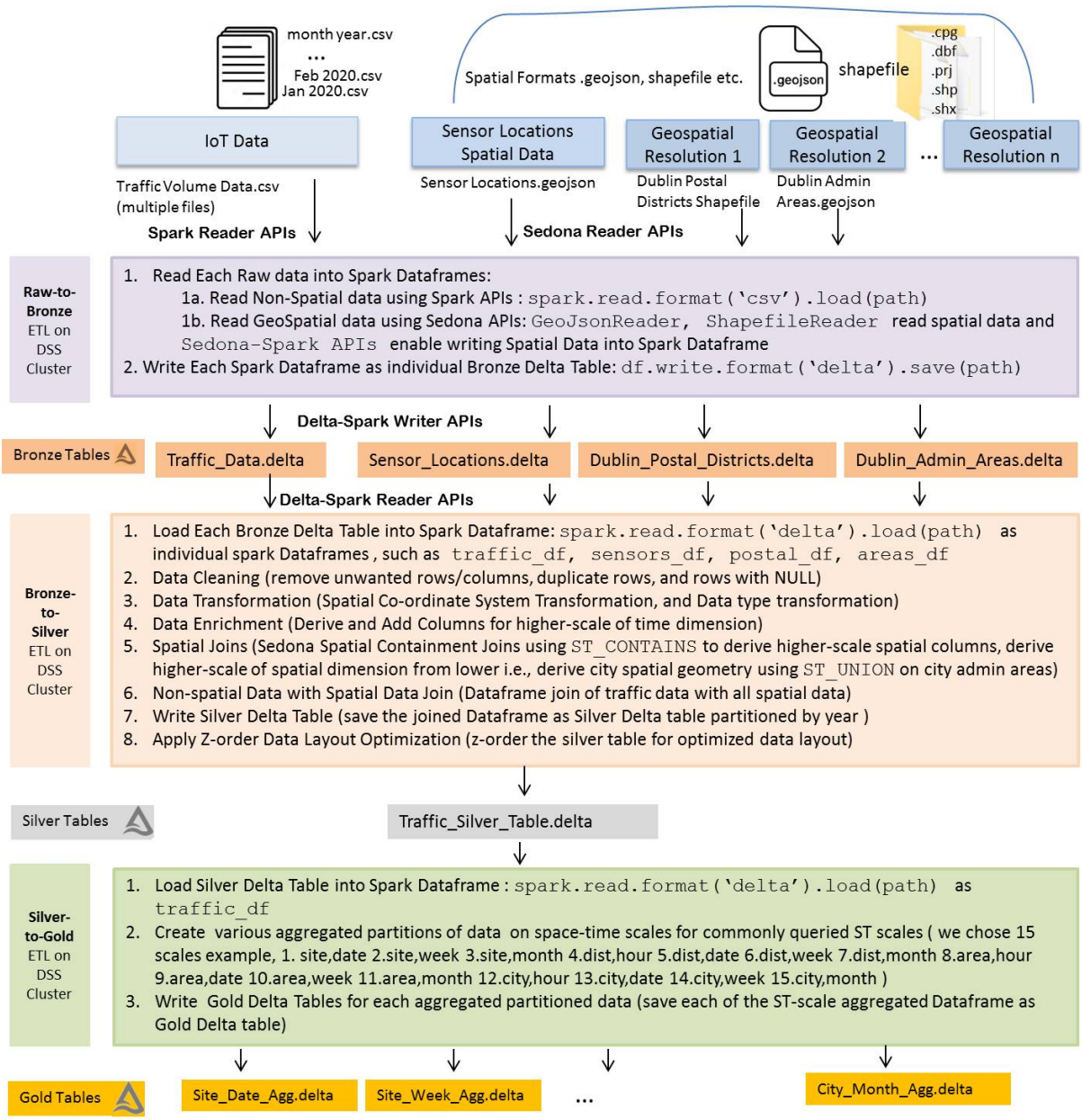


Figure 4.1: Fusion of Geospatial and IoT Big Data of a Smart City to Create Delta Lake Tables

columns, followed by datatypes and coordinate system transformation in step 3. After that, step 4 enriches the traffic dataframe from time-dimension by adding higher-scale temporal columns: date, week (ISO format), month, quarter, and year derived from the hourly timestamp in the traffic dataframe. Then, step 5 derives the city dataframe by the spatial union on Dublin Admin Areas and performs spatial containment joins on pairwise spatial dataframes (sensor locations with Dublin Postal Districts followed by Dublin Admin Areas, followed by Dublin city) to create the enriched spatial dataframe with all spatial-scale columns. Then, step 6 joins the enriched non-spatial traffic dataframe with

the enriched spatial dataframe on the sensor_id column. The joined dataframe contains columns for all space-time scales representing overall processed data, which is saved as a silver delta table (step 7). Then, z-ordering is applied to optimize the data layout of the silver traffic delta table (step 8).

Finally, the **silver-to-gold** pipeline aggregates data w.r.t fifteen commonly space-time column pairs to serve as gold delta tables. Note that all these ETL pipelines are written as Pyspark scripts to run on DSS compute cluster.

4.2.2 Cloud4IoTCity Processing Layer: Compute Cluster for Distributed Processing

The processing layer is implemented as a DSS compute cluster – Delta Lake, Spark, Sedona cluster to perform distributed big data and spatial processing of the service layer Pyspark scripts. The AWS cloud platform offers the AWS Elastic Map Reduce (EMR) service to create and launch customized compute clusters for big data workloads. These clusters spun on the AWS infrastructure allow the developer to customize the software installation and configuration for these clusters. To implement the case study, the AWS EMR Clusters were created using the EMR 6.10.1 Release, which installs Spark 3.3.1 and Hadoop 3.3.3, and a custom bootstrap script was written to install other softwares and libraries namely, geopandas, shapely, attrs, fsspec, s3fs, Apache Sedona 1.3.1, and Delta Spark 2.2.0 on all nodes of the EMR cluster.

4.2.3 Cloud4IoTCity Service Layer: Data Engineering and Data Processing Services

The service layer of the Cloud4IoTCity is in the form of Pyspark script files.

For the DES1 service, three Pyspark scripts implement *raw-to-bronze, bronze-to-silver,* and *silver-to-gold* ETL pipelines to create the Bronze, Silver, and Gold Delta Lake tables, respectively (refer to Section 4.2.1).

The DES2 service appends the city's latest IoT data (traffic data in undertaken case study) into the existing Delta tables subject to the schema validation check to ensure the data quality of Delta tables. This schema validation check is done automatically by the Delta Lake tables. It appends the new data into the Bronze traffic table only if the schema of new traffic data matches the Bronze Table schema. Otherwise, it gives a schema mismatch error and does not append the given set of records into the bronze delta table. Suppose the new data is appended to bronze tables successfully upon schema

validation. In that case, DES2 proceeds to append this new data into silver and gold tables by following the steps in *bronze-to-silver* and *silver-to-gold* pipelines.

The DES3 service performs z-ordering on the silver delta table to optimize the data layout w.r.t commonly queried column(s). In the case study, to implement spatio-temporal analysis, so the temporal column 'year' was chosen as the partitioning column and the spatial column 'site_id' as the z-ordering column to optimize the silver table's layout in space and time dimensions. Partitioning the silver table w.r.t year distributes the table's parquet files among different directories identified by the value of the year column, and z-ordering w.r.t site_id column enables co-locating records within parquet files for the same value of site_id.

The DES4 service allows time travel to a specific version by specifying version number or version timestamp. It is known from Section 3.2.3 that a successful write, append, or optimize operation on Delta tables, i.e., DES1, DES2, or DES3 services, creates a newer version of Delta Lake tables. In this case study, the Delta Lake's time travel feature is used to analyze urban traffic data for the chosen version of Delta Lake tables. The implemented case study enables time travel to a specific version for descriptive and predictive analysis. Maintaining dataset versions and analyzing urban data for chosen dataset version is a novel and valuable feature for smart city frameworks.

The data processing services namely, descriptive analysis and predictive analysis are implemented following a Multi-Resolution spatio-temporal (MRST) perspective. In this regard, the four spatial resolutions and seven temporal resolution are considered for descriptive analysis. The four spatial resolutions are as: i) sensor sites i.e., (longitude, latitude) points for traffic sensor locations, ii) small area polygons for Dublin Postal Districts, iii) large area polygons for Dublin Admin Areas, and iv) the complete city polygon. The seven temporal resolutions are as: i) hour (datetime), ii) date, iii) week, iv) month, v) quarter, vi) year, and vii) all. The descriptive analysis service implements the fifteen taxonomized spatio-temporal urban queries (Section 3.5) in fifteen Pyspark script files containing the business logic for each query type to run Spark+Sedona SQL query on the silver delta table w.r.t the chosen space-time scales.

For predictive analysis service, the site is considered for spatial scale, and hour and date are considered as two temporal scales for traffic prediction in the case study. Since traffic data is highly dynamic, which often requires predictions at fine-granular space-time scales, such as hourly or daily traffic prediction for chosen sites, thus we considered only these scales. Next, Prophet time-series forecasting ML model [180] is used to implement the per-site hourly and daily ML models for traffic prediction in Dublin. The Prophet is chosen as the baseline ML model in this case study because the Prophet allows to consider holidays,

both the country's (Ireland in the undertaken case study) public holidays and custom-specified date ranges for holidays while fitting the time-series forecasting model. This case study data for urban traffic (2020-2022) includes the COVID-19 pandemic lockdowns, which exhibit an entirely different traffic pattern for the city. Such circumstances as public holidays and government-announced holidays or severe lockdowns happen in smart cities, affecting urban phenomena such as traffic. Therefore, the ML model for predicting urban phenomena shall allow configuring city holidays and lockdowns while fitting the ML model to urban data. So, the traffic prediction is implemented by configuring Ireland's public holidays and pandemic lockdowns as holidays in the Prophet-based per-site hourly and daily urban traffic prediction ML models.

A visualization service is implemented for this case study using the Plotly Express [171] Python package. As the analysis services receive the descriptive or predictive analysis results dataframe, they call the visualization services to plot the results visually on city maps. While creating map plots, the size and color is kept proportional to traffic magnitude at displayed sites and provide additional data as hover text in the maps. For the descriptive analysis queries involving time range, an animation of maps over the duration is displayed with play and pause controls for the running animation.

4.2.4 Cloud4IoTCity Application Layer: 'Urban Ananlytics' SaaS Web App on AWS

To implement the framework's application layer, a SaaS web application named 'Urban Analytics' is developed to enable authorized city managers and analysts to perform urban traffic analysis for Dublin City through the web app deployed on the AWS cloud. This work uses Flask [181] (web-app development framework in python) and HTML5 to develop the web application. The Flask application launches the DSS cluster's spark session to enable the running of service layer Pyspark scripts, which comprise the application's business logic for distributed spatio-temporal urban analysis on Delta Lake tables supported with map-based visuals obtained from the visualization service. To deploy the web application on the AWS cloud, an AWS EC2 (Elastic Compute Cloud) instance is spinned and deployed the 'Urban Analytics' Flask application on that EC2 instance. Figures 4.2 and 4.3 shows screenshots from the developed web app. Figure 4.2 shows an application snapshot while configuring the descriptive query and Figure 4.3 shows an application snapshot displaying descriptive analysis query results.

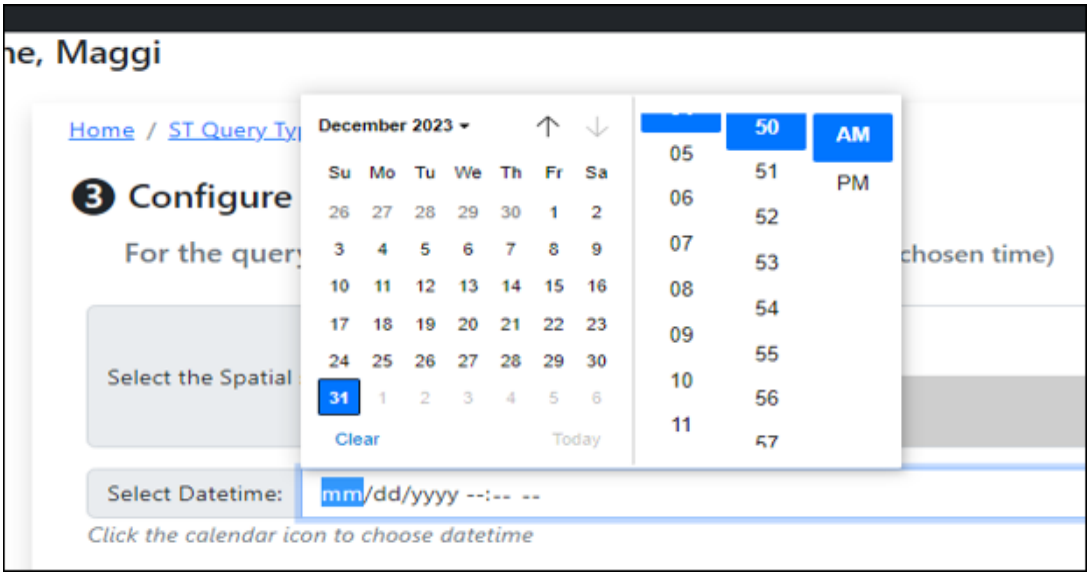


Figure 4.2: Query Configuration for Descriptive Analysis in Urban Analytics App

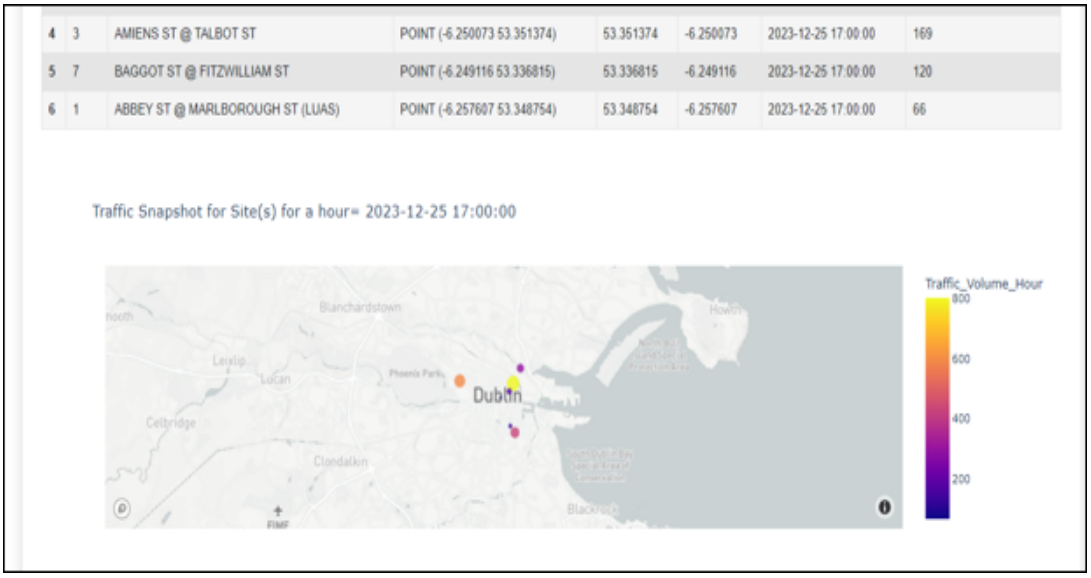


Figure 4.3: Descriptive Analysis Results in Urban Analytics App

4.3 Experimental Results for Cloud4IoTCity Validation

To carry out the experiments for framework validation, the AWS Elastic MapReduce (EMR) service is used to create custom DSS cluster with supporting libraries. The technical specifications along with its versions are given in Table 4.1. The AWS EMR service provide compute cluster for distributed processing of the big data in a master-worker architecture. To conduct the experiments, EMR 6.10.1 release is used that comes

with Spark 3.3.1 and Hadoop 3.3.3. To this cluster, Delta Spark 2.2.0,Apache Sedona 1.3.1, and supporting libraries geopandas, shapely, attrs, s3fs, and fsspec are installed during the cluster bootstrap. Further, to assess the results for distributed processing for different cluster sizes, three cluster configurations are created with the number of worker nodes as 1, 4, and 8, respectively. Each cluster node is a m5.xlarge instance in each cluster configuration. The m5.xlarge AWS instance has four virtual cpus (vcpus), which enables four executors per worker node. Using the m5.xlarge instance for each node in a cluster implies that the total number of executors available for distributed processing in a cluster configuration is four times the number of worker nodes in that configuration. Cloud4IoTCity is validated in subsequent sections as follows:

Table 4.1: Technical Specifications for Experimental Testbed

Technical Specification	**Version/ Remarks**
AWS EMR	6.10.1 (Spark 3.3.1 and Hadoop 3.3.3 pre-installed)
Delta Spark	2.2.0
Apache Sedona	1.3.1
geopandas, shapely, attrs	pre-requisite libraries for Apache Sedona
s3fs, fsspec	libraries required for accessing AWS S3 files
m5.xlarge	AWS VM Type with 4 vcpus for cluster nodes (both, master and worker)
Cluster Size	three configurations with no. of worker nodes as 1, 4, and 8

Test Case 1: Multi-Resolution Spatio-Temporal Descriptive Analysis (Section 4.3.1)
Test Case 2: Multi-Resolution Spatio-Temporal Predictive Analysis (Section 4.3.2)
Test Case 3: Add and Analyze Newly Data while ensuring Data Quality (Section 4.3.3)
Test Case 4: Impact on Query Runtime by Distributed Processing of the Queries (Section 4.3.4)
Test Case 5: Impact on Query Runtime by Optimizing Data Layout. (Section 4.3.5)

4.3.1 Cloud4IoTCity Validation for Multi-Resolution Spatio-Temporal Descriptive Analysis

This section validates the Cloud4IoTCity framework for descriptive analysis based on the fifteen spatio-temporal queries. To perform multi-resolution spatio-temporal descriptive analysis, the end user can choose among four spatial resolutions – sensor sites, Dublin postal districts, Dublin admin areas, Dublin city; and seven temporal resolutions – hour, date, week, month, quarter, year, and all through the developed Urban Analytics web

app.

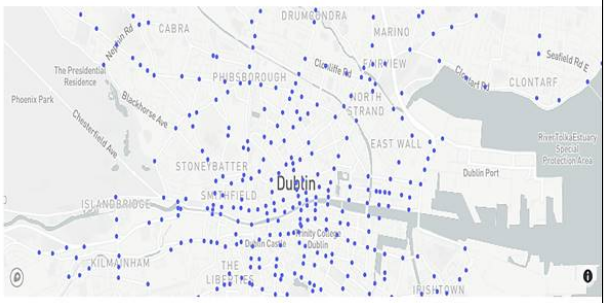
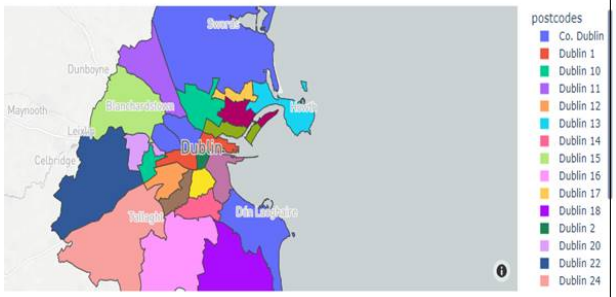
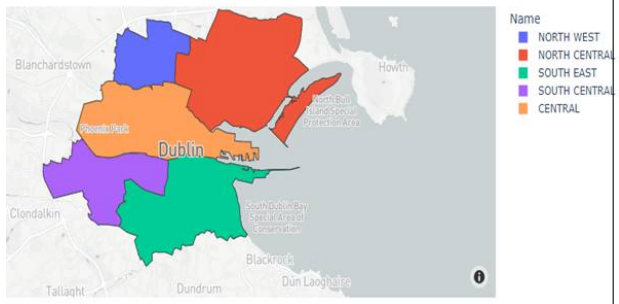
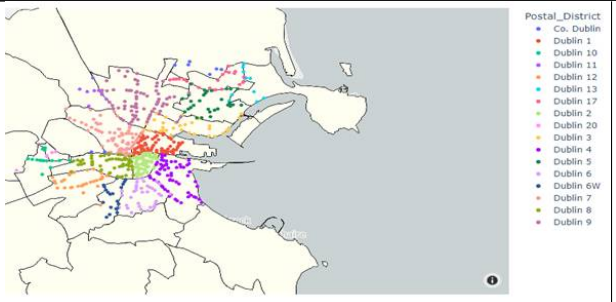
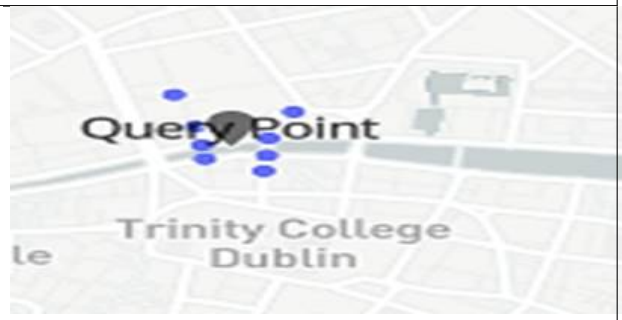
The Urban Analytics app offers the MRST-QP service to the city managers to perform multi-resolution spatio-temporal descriptive analysis in three steps:
i) select query type
ii) choose space and time scales
iii) configure query parameters

The developed app enables users to select single, multiple, or ALL values for the spatial parameter w.r.t the spatial scale while configuring the query.

Each of the fifteen descriptive queries is run three times, selecting different space-time scales with varying numbers of spatial parameters (single/ multiple/ ALL). Note that selecting different space-time scale combinations can configure many queries. However, three examples per query are shown here for conciseness while demonstrating the variety of space-time scales and spatial parameter plurality for enabling fine-granular to coarse-granular urban data analysis.

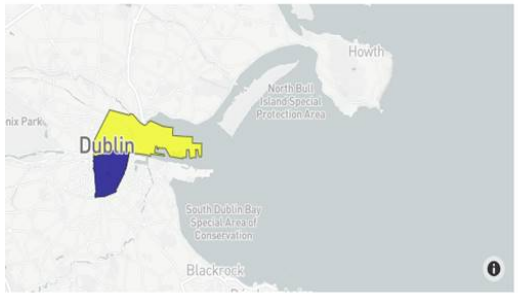
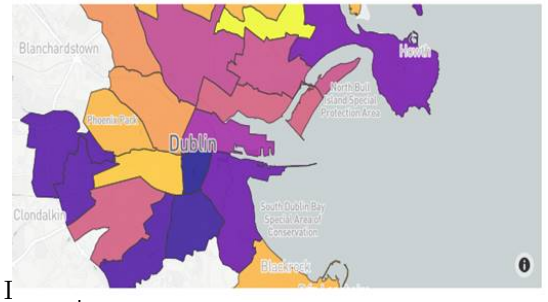
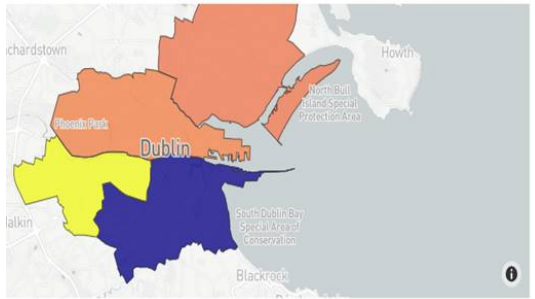
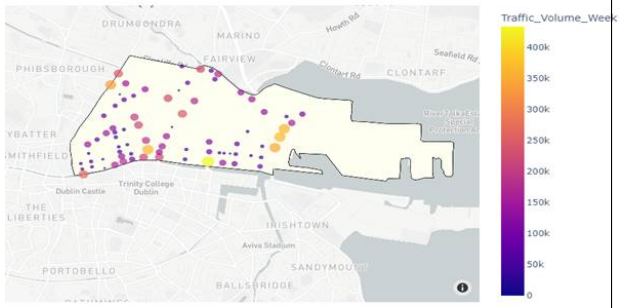
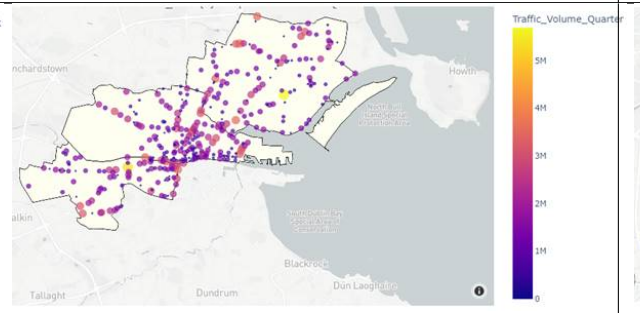
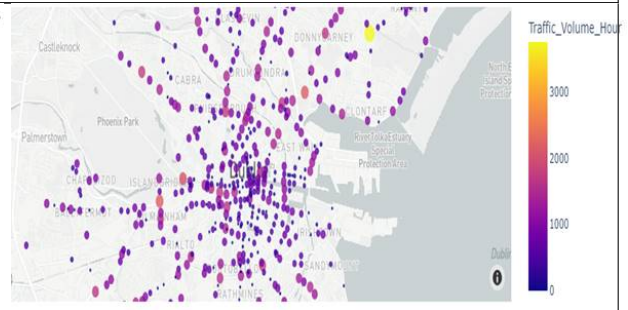
Table 4.2 shows the configured queries and their visualization results for the three examples of each query type. The map-based results for descriptive queries in Table 4.2 validate the framework's MRST-QP service for multi-resolution spatio-temporal descriptive analysis of urban data, enabled by geospatial and urban IoT data fusion.

Table 4.2: Multi-Resolution Spatio-Temporal Query Processing Results for the Case Study on Urban Traffic Analysis for Dublin City

Query Type	**Example 1**	**Example 2**	**Example 3**
Spatial Simple	 Description: Get selected spatial addresses (sensor points or city polygons) Get (ALL) ***Sites*** for IoT sensor locations	 Get (ALL) ***Postal Districts*** of Dublin city	 Get (ALL) ***Admin Areas*** of the city
Spatial Simple with Condition	 Description: Spatial object retrieval based on WITHIN or k-nn Condition Get sites WITHIN (ALL) ***Districts*** of city	 Condition: get sites WITHIN polygon Get sites WITHIN (multiple) selected ***Areas*** of city	 Condition: find k-NN sites Get (k=8) nearest *Sites* from the input query point (long, lat)= (-6.2564,53.3487)

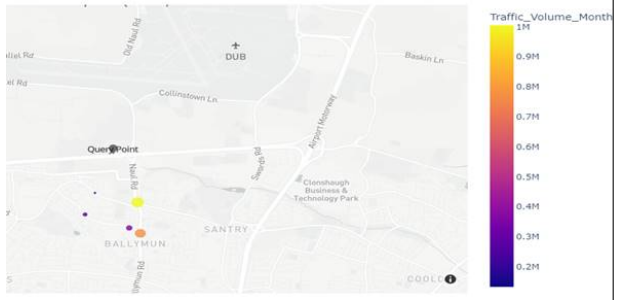
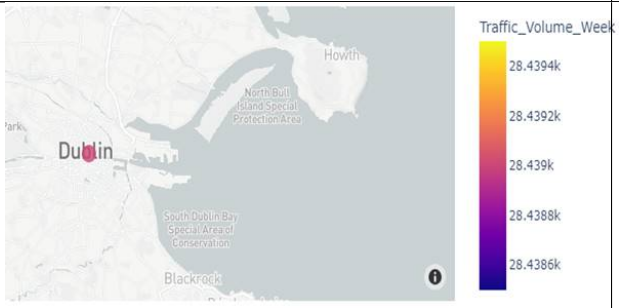
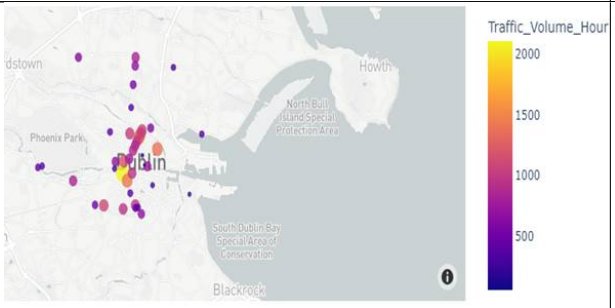
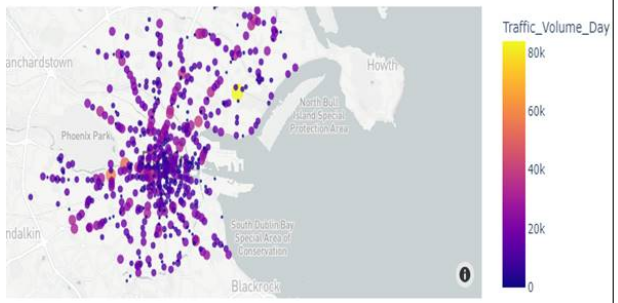
Continued on next page

Table 4.2 – *Continued from previous page*

Query Type	Example 1	Example 2	Example 3
Spatial Range	 Attribute summary over spatial polygons for a datetime Get traffic summary for (multiple) selected ***Districts*** at 2023-12-25 18:00	 Get traffic summary for (ALL) ***Districts*** at 2023-12-25 18:00	 Get traffic summary for (ALL) ***Areas*** at 2023-12-25 18:00
Spatial Relation WITHIN	 Description: Attribute info. for Sites WITHIN Polygons at any temporal scale Get traffic for ***Sites WITHIN District*** (single) D1 for the ***Week*** 2023-W10	 Get traffic for ***Sites WITHIN Areas*** (multiple) selected for the ***Quarter*** 2020-Q4	 Get traffic for ***sites WITHIN City*** (Dublin) at ***Datetime*** 2021-04-05 17:00

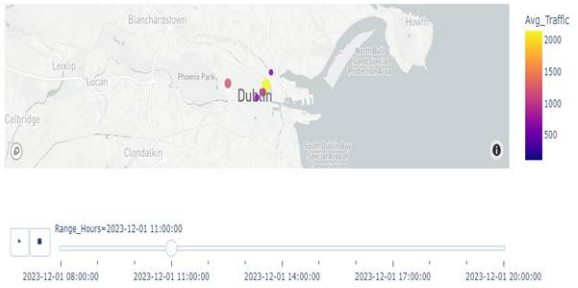
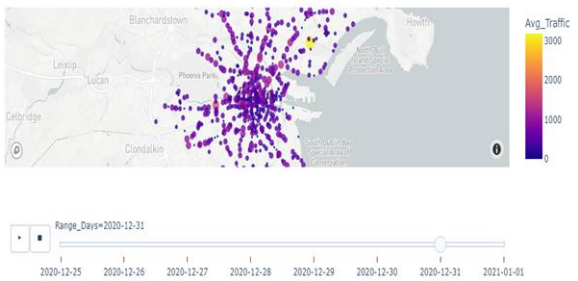
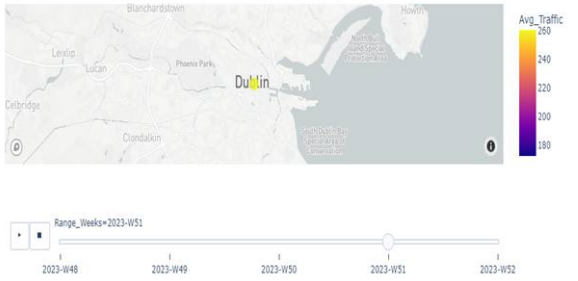
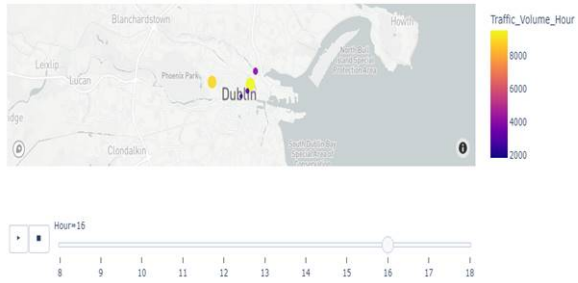
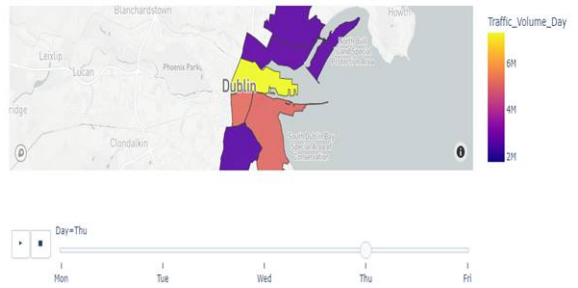
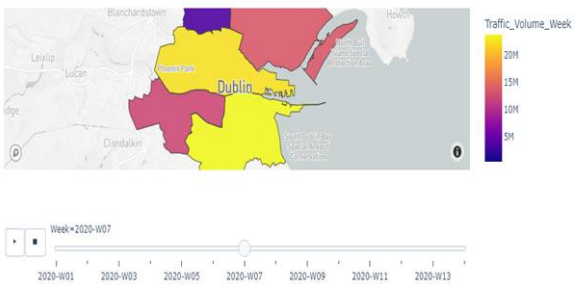
Continued on next page

Table 4.2 – *Continued from previous page*

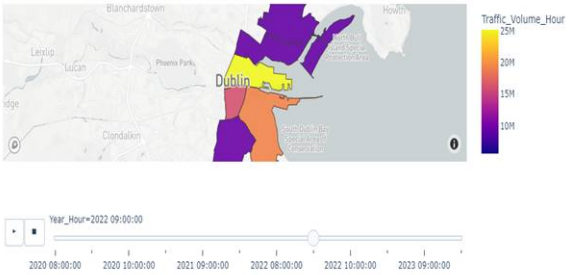
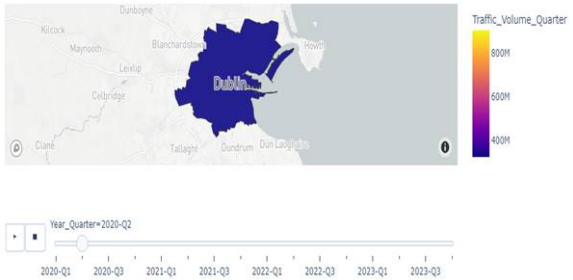
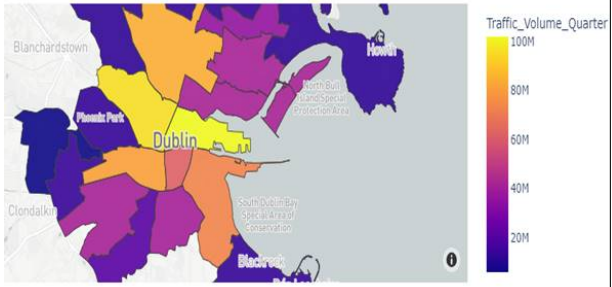
Query Type	Example 1	Example 2	Example 3
Spatial Relation k-NN	 Description: Attribute info. for k-nn sites from query pt at any temporal scale Get traffic for (k=10) ***nearest sites*** from query point= (-6.2534,53.3421) at ***Datetime*** 2022-01-01 13:00	 Get traffic for (k=7) ***nearest sites*** from query point= (-6.23,53.35) for ***Date*** 2022-02-28	 Get traffic for (k=5) ***nearest sites*** from query point= (-6.27,53.41) for the ***Month*** 2020-01
Temporal Simple	 Description: Attribute snapshot for chosen site(s) at any temporal scale Get traffic for ***Sites*** (single) site1 for the ***Week*** 2021-W04	 Get traffic for ***Sites*** (multiple) selected at ***Datetime*** 2022-12-25 19:00	 Get traffic for ***Sites*** (ALL) for the ***Date*** 2020-07-04

Continued on next page

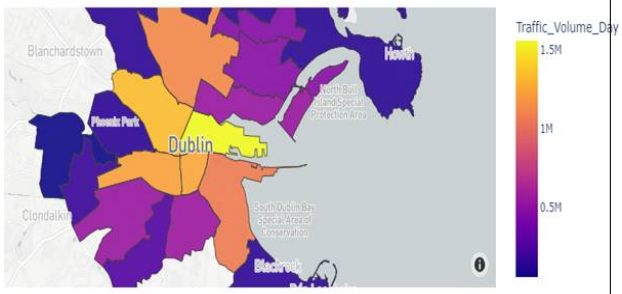
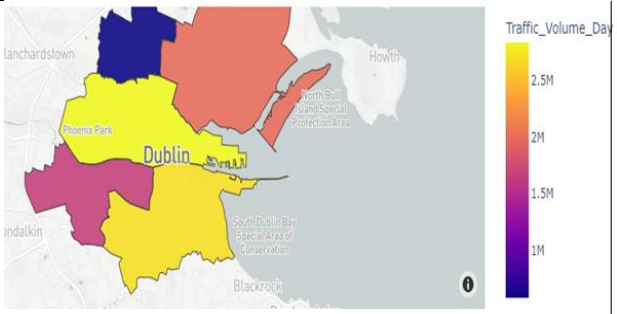
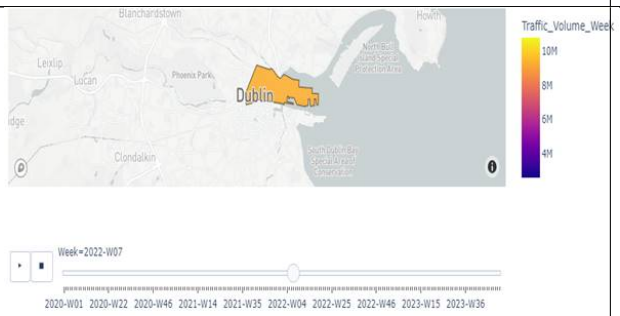
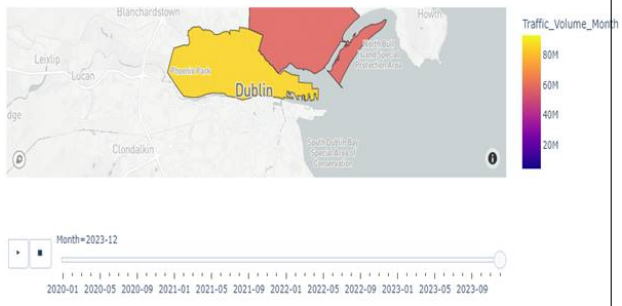
Table 4.2 – *Continued from previous page*

Query Type	Example 1	Example 2	Example 3
Temporal Range	 Description: Attribute info for sites for the range of time defined at any temporal scale Get traffic for ***Sites*** (multiple) selected for the ***Datetime*** range from 2023-12-01 08:00 to 2023-12-01 20:00	 Get traffic for ***Sites*** (ALL) for the ***Date*** range from 2020-12-25 to 2021-01-01	 Get traffic for ***Sites*** (single) site1 for the ***Week*** range from 2023-W48 to 2023-W52
Temporal Relation OF-A	 Description: Attribute info at any spatial scale for smaller time range OF-A bigger temporal value Get traffic for (multiple) selected ***Sites*** for ***HOURS OF-A WEEK*** for 08:00 to 18:00 Hours of-a Week 2023-W50	 Get traffic for (multiple) selected ***Districts*** for ***DAYS OF-A MONTH*** for Mon to Fri Days of-a Month 2022-06	 Get traffic for (ALL) ***Areas*** for ***WEEKS OF-A QUARTER*** for all Weeks of-a Quarter 2020-Q1

Continued on next page

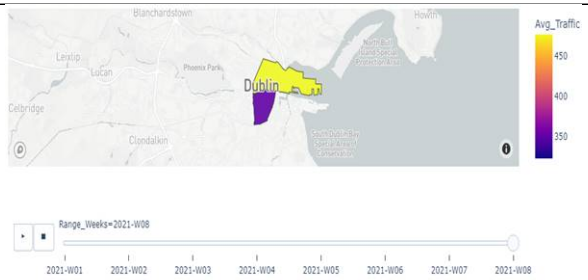
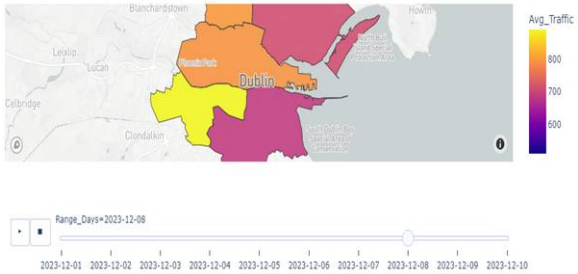
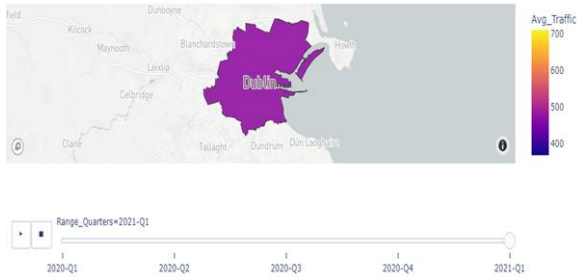
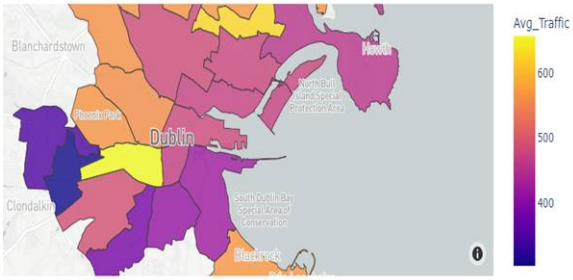
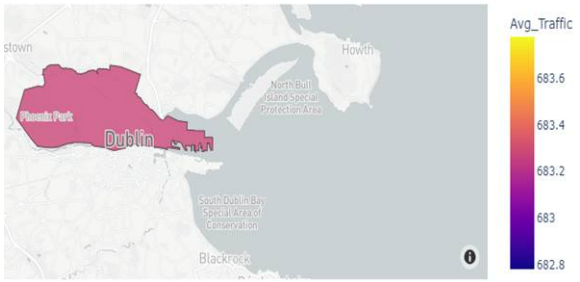
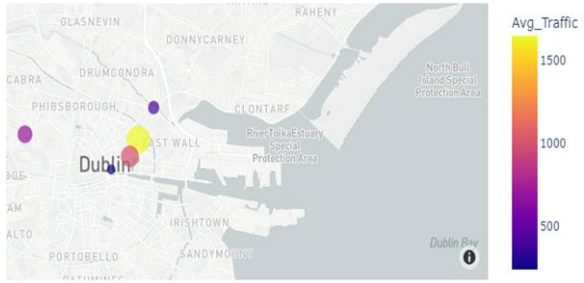
Query Type	Example 1	Example 2	Example 3
Temporal Relation OF-EACH	 Description: Attribute info at any spatial scale for smaller time range OF-EACH bigger temporal value Get traffic for (multiple) selected ***Districts*** for 08:00 to 10:00 Morning Hours ***HOURS OF EACH YEAR***	 Get traffic for (single) selected ***Site*** for (ALL) ***MONTHS OF EACH QUARTER***	 Get traffic for ***City*** (Dublin city) for (ALL) ***QAURTERS OF EACH YEAR***
Spatio-Temporal WHAT	 Description: Attribute info for any spatial and temporal scales Get traffic for (multiple) selected ***Sites*** for the ***Date*** 2023-12-25	 Get traffic for (ALL) ***Districts*** for the ***Quarter*** 2021-Q2	 Get traffic for ***City*** (Dublin city) for the ***Datetime*** 2023-12-12 19:00

Continued on next page

Query Type	Example 1	Example 2	Example 3
Spatio-Temporal WHERE	 Description: Find spatial locations of any scale where the attribute meets specified threshold condition for any temporal scale Get ***Sites*** WHERE traffic is ($>$ 2500) at ***Datetime*** 2023-12-31 20:00	 Get ***Districts*** WHERE traffic is ($\geq$ 10000) for ***Date*** 2020-02-05	 Get ***Areas*** WHERE traffic is ($>$ 80000) for ***Date*** 2023-07-04
Spatio-Temporal WHEN	 Description: Get times of any scale WHEN the attribute meets specified threshold condition at any spatial scale Get ***Dates*** WHEN traffic ($>$ 5000) for (single) site1 ***Site***	 Get ***Weeks*** WHEN traffic ($\geq$ 80000) for (single) D1 ***District***	 Get ***Months*** WHEN traffic ($\geq$ 1000000) for the (multiple) selected ***Areas***

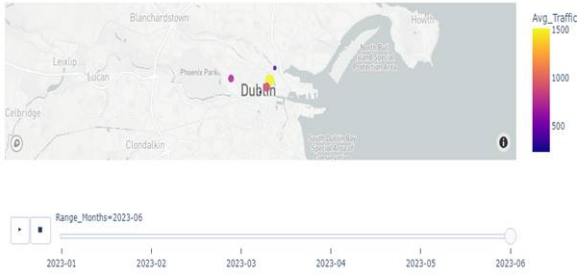
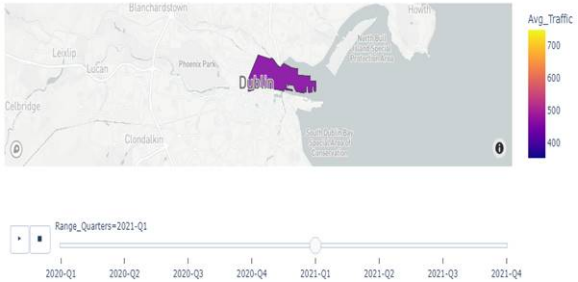
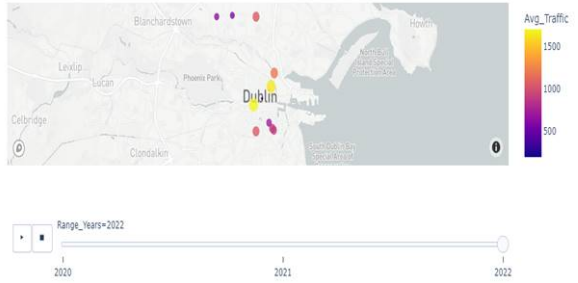
Continued on next page

Table 4.2 – *Continued from previous page*

Query Type	Example 1	Example 2	Example 3
Spatio-Temporal Range	 Description: Attribute summary over spatial polygons for time range at any temporal scale Get traffic summary over spatial sites of (multiple) selected ***Districts*** for the time range over ***Weeks*** from 2021-W01 to 2021-W08	 Get traffic summary over spatial sites of (ALL) ***Areas*** for the time range over ***Dates*** from 2023-12-01 to 2023-12-10	 Get traffic summary over spatial sites of ***City*** (Dublin) for the time range over ***Quarters*** from 2020-Q1 to 2021-Q1
Spatio-Temporal Behaviour AT-TIME	 Description: Attribute behavior AT-TIME for any spatial and temporal scales Get traffic behavior for (ALL) ***Districts*** for the ***Date*** 2023-12-31	 Get traffic behavior for (single) Central ***Area*** for the ***Month*** 2023-07	 Get traffic behavior for (multiple) selected ***Sites*** for the ***Quarter*** 2020-Q1

Continued on next page

Table 4.2 – *Continued from previous page*

Query Type	**Example 1**	**Example 2**	**Example 3**
Spatio-Temporal Behaviour OVER-RANGE	 Description: Attribute behavior OVER-RANGE of time for any spatial and temporal scales Get traffic behavior for (multiple) selected ***Sites*** for the time range over ***Months*** from 2023-01 to 2023-06	 Get traffic behavior for (single) D1 ***District*** for the time range over ***Quarters*** from 2020-Q1 to 2021-Q4	 Get traffic behavior for (multiple) selected ***Sites*** for the time range over ***Years*** from 2020 to 2022

4.3.2 Cloud4IoTCity Validation for Multi-Resolution Spatio-Temporal Predictive Analysis

This section presents the results for MRST-ML service to perform predictive analysis of urban data. In the undertaken case study, traffic prediction is implemented at finer space-time resolutions (refer 4.2.3) to enable per-site hourly or daily traffic prediction. Since traffic is a highly dynamic urban attribute, it is recommended to train ML models at the site level for spatial scale to get fine-granular traffic predictions for selected sites. Higher spatial scales are not favorable for an urban attribute like traffic because, within an area, different sites experience different patterns of traffic. Therefore, the sensor's site is considered for spatial scale, while hour and date are two temporal scales for urban traffic prediction.

Figures 4.4, 4.5, 4.6, and 4.7 shows the visualization results for site-level hourly and daily traffic prediction. Furthermore, the Urban Analytics app enables city managers to predict traffic either for a single site or multiple sites. From the visualization results in Figures 4.4, 4.5, 4.6, and 4.7, it is concluded that the MRST-ML service enables multi-resolution spatio-temporal predictive analysis by predicting hourly or daily traffic for selected sites.

Figure 4.4: Hourly Traffic Prediction for Single Site (site=1 hour=2024-07-04 17:00)

The results of MRST-ML analysis service in Figures 4.4, 4.5, 4.6, and 4.7 validates the Cloud4IoTCity framework for enabling spatio-temporal predictive analysis at multiple scales, such as hourly and daily temporal scales for traffic prediction in the city.

Figure 4.5: Hourly Traffic Prediction for Multiple Sites (sites=1,2,3,4 hour=2024-07-04 17:00)

Figure 4.6: Daily Traffic Prediction for Single Site (site=1 date=2024-07-04)

Figure 4.7: Daily Traffic Prediction for Multiple Sites (sites=1,2,3,4,5 date=2024-07-04)

4.3.3 Cloud4IoTCity Validation for Addition and Analysis of Latest IoT Data

This section validates the Cloud4IoTCity framework for adding and analyzing the latest IoT data. The web app allows the end-user to upload single or multiple .csv files (as the SCATS traffic data is in .csv) for adding the uploaded recent data to the storage layer, the webpage shown in Figure 4.8 below.

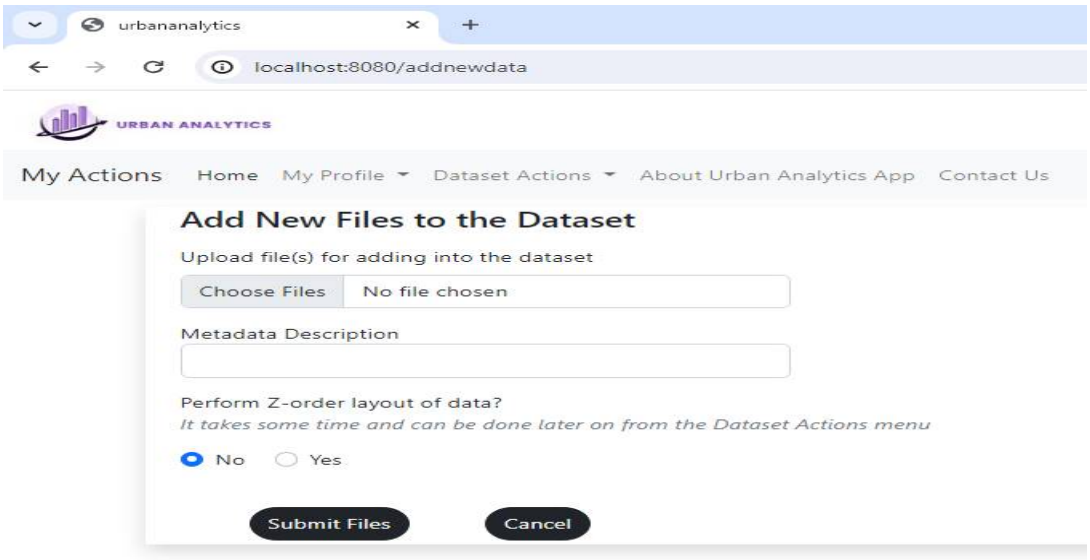


Figure 4.8: Add New Data in Urban Analytics app

Dataset Versions and Time Travel

Currently, loaded dataset version is: 9

Dataset Version	Created at time	Operation	User Metadata	Engine Info	Time Travel
9	2024-02-22 18:40:35.788	WRITE	December 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
8	2024-02-21 16:32:47.276	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
7	2024-01-23 12:47:45.548	WRITE	April 2023-Nov 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
6	2024-01-15 12:17:50.731	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
5	2024-01-15 12:12:18.227	WRITE	March2023	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
4	2024-01-12 19:47:05.834	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
3	2024-01-12 19:41:37.435	WRITE	February 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
2	2024-01-12 19:37:03.300	WRITE	January 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
1	2023-07-12 14:22:19.145	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
0	2023-07-12 14:06:35.296	WRITE	Jan2020-Dec2022 complete traffic data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing

Figure 4.9: Dataset Versions created by successful addition of new data from time to time

Delta Lake performs a schema validation check, allowing only quality data to be writ-

ten/appended to the delta tables. Suppose the uploaded file(s) schema matches the bronze delta table (traffic bronze table). In that case, it appends the latest IoT data from .csv files to the bronze table and runs bronze-to-silver and silver-to-gold pipelines to append the latest data to silver and gold delta tables. This step creates a new dataset version for the delta tables. Figure 4.9 shows the newer dataset versions in app created upon successfully adding new SCATS data to the storage layer, while the figures 4.10 and 4.11 show the failure for bad writes. These results from the app answer the first part of second validation feature, that the framework allows the addition of the latest smart city IoT data while ensuring data quality.

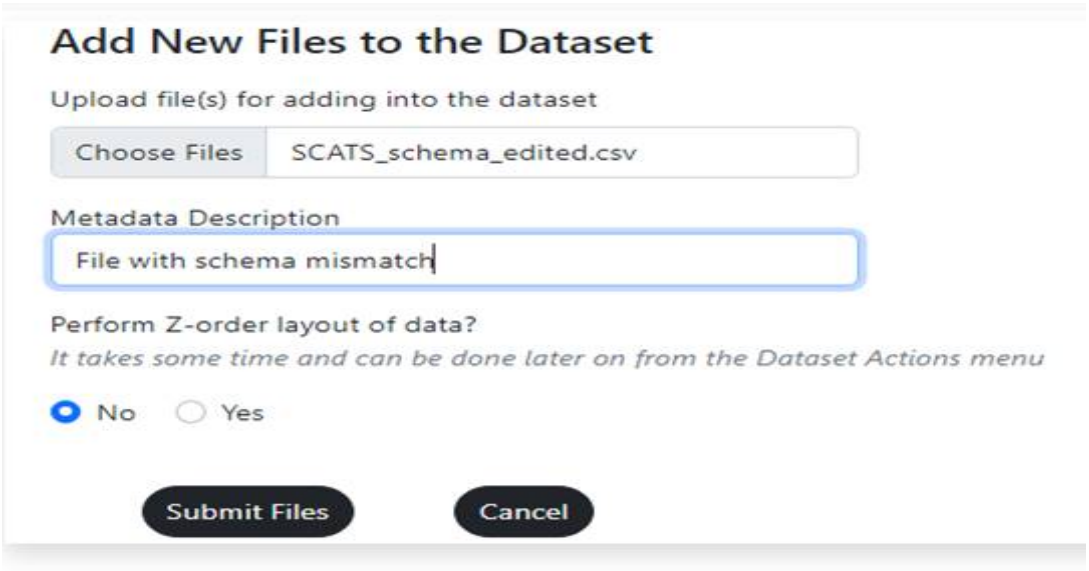


Figure 4.10: Attempt to add data bad data, whose schema do not match

Now, the second part of validation test i.e., to analyze newly added data is presented. The proposed framework leverages the time travel feature of Delta Lake to facilitate analysis of the newly added data by time travel to the most recent version of the delta table.

Both MRST-QP and MRST-ML services in the proposed framework support version-specific urban data analysis. By default, the MRST-QP service reads the most recent version of delta tables. The MRST-QP service allows time travel to any version by clicking the 'time travel' link for a version from the 'Dataset versions and Time Travel' webpage (refer Figure 4.12, which reads the delta table for the chosen version and now the user proceeds to run descriptive queries. After selecting the query type and choosing space-time scales for the query, the user configures space-time parameters, wherein the temporal parameter picker (calendar) allows the choice of a time value within the min-max time range of the chosen dataset version. For example, Figure 4.12 selects time travel

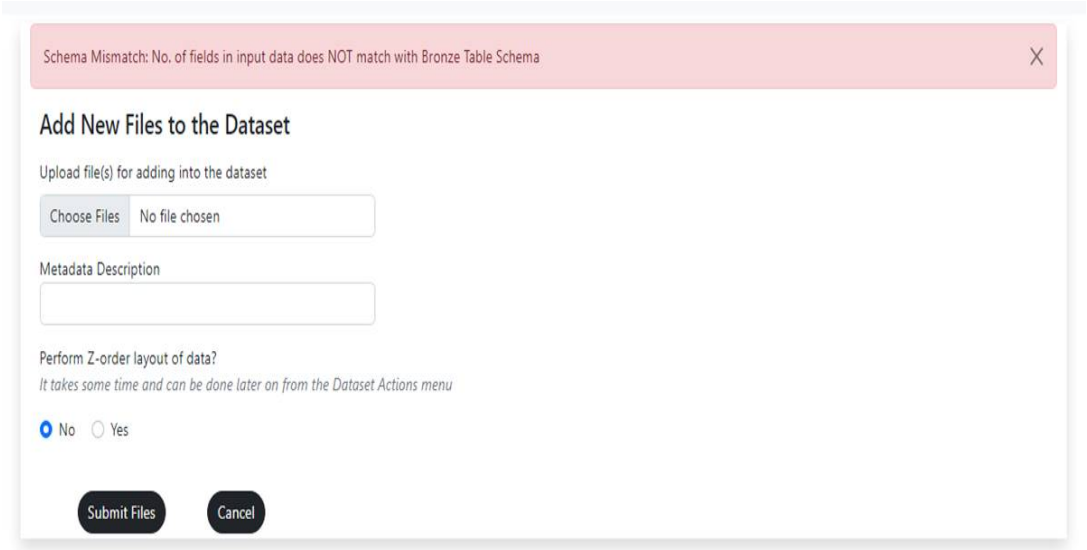
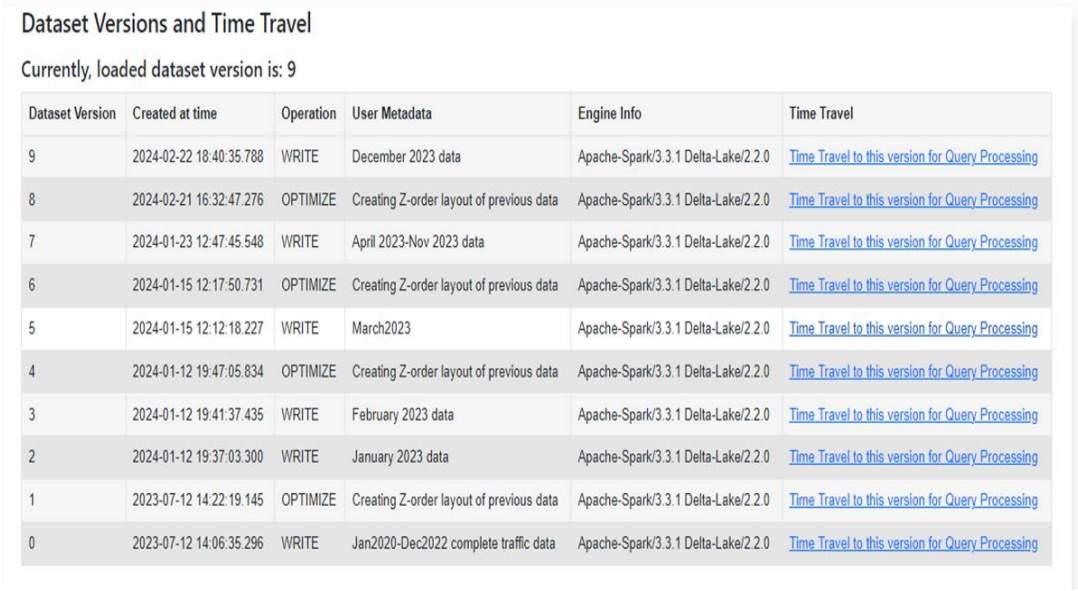


Figure 4.11: Schema mismatch error returned to end-user ensuring Data Quality

to version 5, which added March 2023 data, and Figure 4.13 shows subsequent querying for the chosen version 5, allowing the max time range up to March 2023. However, the later data exists in storage layer delta tables added by subsequent versions. Figure 4.14 shows time travel again to the most recent version 9, which allows the selection of temporal values corresponding to this version up to Dec 2023. These results validate that the MRST-QP service enables analyzing newly added data or any previous snapshot of data by using data versioning and time travel features.



Dataset Versions and Time Travel

Currently, loaded dataset version is: 9

Dataset Version	Created at time	Operation	User Metadata	Engine Info	Time Travel
9	2024-02-22 18:40:35.788	WRITE	December 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
8	2024-02-21 16:32:47.276	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
7	2024-01-23 12:47:45.548	WRITE	April 2023-Nov 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
6	2024-01-15 12:17:50.731	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
5	2024-01-15 12:12:18.227	WRITE	March2023	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
4	2024-01-12 19:47:05.834	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
3	2024-01-12 19:41:37.435	WRITE	February 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
2	2024-01-12 19:37:03.300	WRITE	January 2023 data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
1	2023-07-12 14:22:19.145	OPTIMIZE	Creating Z-order layout of previous data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing
0	2023-07-12 14:06:35.296	WRITE	Jan2020-Dec2022 complete traffic data	Apache-Spark/3.3.1 Delta-Lake/2.2.0	Time Travel to this version for Query Processing

Figure 4.12: Time Travelling to version 5, which added March 2023 data (css color change on row select)

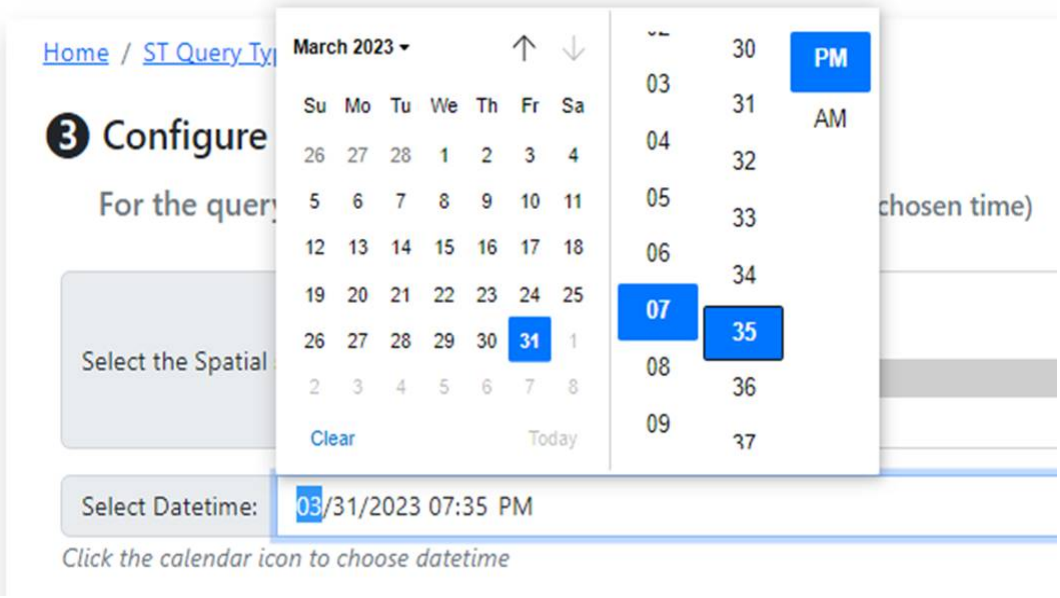


Figure 4.13: Querying on version 5 does not allow to pick a time beyond its version i.e., March 2023

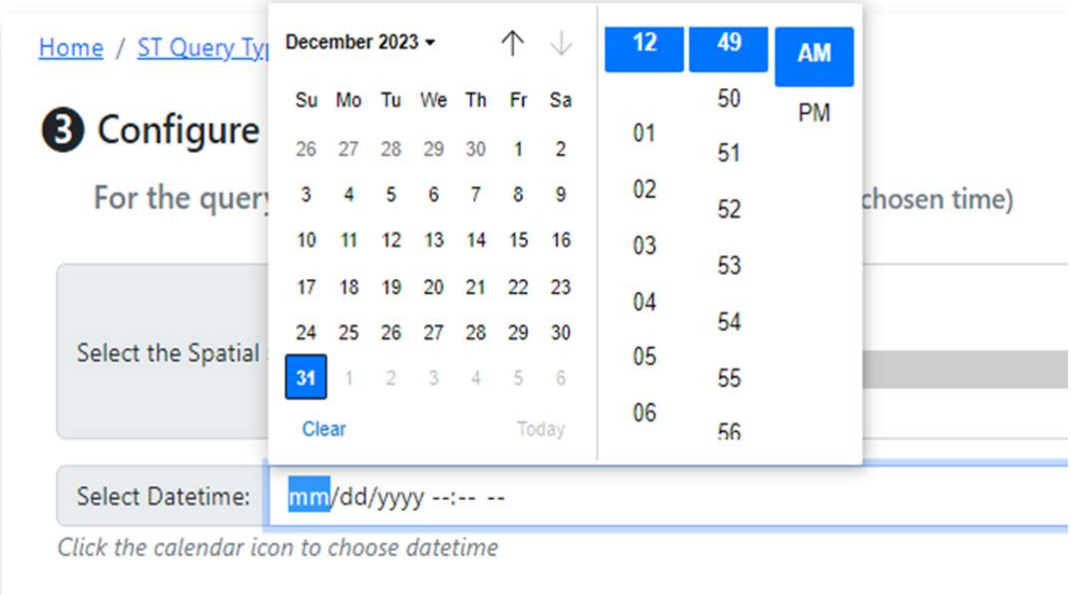


Figure 4.14: Querying on latest version 9 allows to pick a time w.r.t. version 9 i.e., upto Dec 2023

The MRST-ML service also allows predictive analysis by considering the latest data added to the delta tables. The MRST-ML service also enables version-specific analysis corresponding to a selected data version. This service trains ML models w.r.t. a chosen version and, while running the trained models for analysis, enables users to select the underlying version for the ML models. The version-specific per-site hourly and daily trained ML models follow a naming convention, such as 'hourly_site1_ver0', 'daily_site2_ver9', and so

on, for allowing the selection of the ML model w.r.t dataset version used for training the model. Figures 4.15 and 4.16 show the results of hourly prediction for a site using version 0 and version 9, respectively, then figures 4.17 and 4.18 show the results of daily prediction for multiple sites using version 0 and version 9, respectively. Thus, both scales for prediction hourly and daily have their models trained w.r.t dataset version, allowing the user to train models for newer versions as well. Therefore, the proposed framework supports predictive analysis over multiple versions of data. These results validate the framework's MRST-ML service for enabling analysis by considering the latest IoT data.

Figure 4.15: Hourly Traffic Prediction for Single Site for Version 0 (ML model trained on 3-year data from 2020-2022)

Figure 4.16: Hourly Traffic Prediction for Single Site for Version 9 (ML model training included 4-year data from 2020-2023)

The results in this subsection answer the later part of validation test case 3 that the

Figure 4.17: Daily Traffic Prediction for Multiple Sites for Version 0 (all models trained on 3-year data)

Figure 4.18: Daily Traffic Prediction for Multiple Sites for Version 9 (all models training included 4-year data)

proposed framework allows adding new data successfully and avoids bad writes to ensure data quality. In addition, the proposed framework supports the analysis of newly added data for both descriptive and predictive analysis.

4.3.4 Cloud4IoTCity Validation for Distributed Data Processing

This section evaluates the impact of distributed query processing on the query runtime. To obtain experiment results, each of the fifteen taxonomized spatio-temporal descriptive analysis queries are run in a distributed manner. Each query is run ten times on the three cluster configurations to avoid the bias effect of selecting space-time scales and

their parameter values. For each query type on a given cluster size,the average runtime is computed for ten iterations as the average query runtime. The query runtime results for each category of queries viz. Spatial Queries, Temporal Queries, and Spatio-Temporal Queries are shown in Figures 4.19, 4.20, and 4.21 respectively.

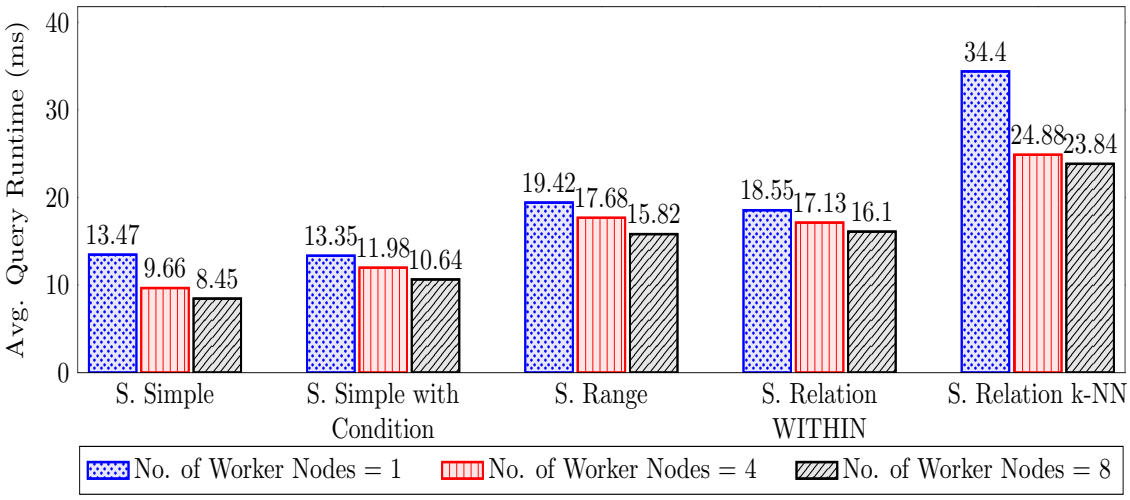


Figure 4.19: Average Query Runtime (millisec) for Spatial Queries

From Figure 4.19, it is inferred that the proposed Cloud4IoTCity framework is efficiently able to process the *Spatial Queries* on Urban Big Data in *distributed* manner as for each spatial query the *average query runtime decreases with increasing no. of worker nodes* in the cluster. As expected, this observed trend is attributed to high data parallelism in cluster configurations with a higher number of worker nodes, wherein the higher number of executors *(4 times the worker nodes)* are processing the query in parallel and distributed manner resulting in lesser query runtime.

Secondly, the query runtime for each spatial query varies w.r.t query complexity. It is observed that the simplest *filtering-based* where clause queries (S. Simple, S. Simple with Condition) run fastest, followed by *aggregation-based* groupby, aggregation queries (S. Relation WITHIN, S. Range), followed by the in place spatial processing spatial distance computation (S. Relation k-NN) query. On a further note, within the first *filtering-based* category (S. Simple, S. Simple with Condition), the S. Simple with Condition query gives slightly higher runtime due to involvement of expensive spatial distance processing function ST_DISTANCE when the randomly selected condition for an iteration is k-NN instead of WITHIN condition; in the second *aggregation-based* category (S. Relation WITHIN, S. Range), the S. Range query gives slightly higher runtime due to additional data shuffling across cluster executors involved in collecting the per-executor group aggregates (avg, sum, min, max) from all executors and reducing these per-executor group aggregates to single-valued group aggregates; finally, the third *in place spatial pro-*

cessing category (S. Relation k-NN) results in highest query runtime as it applies the ST_DISTANCE *Spatial function at runtime* to compute the spatial distance between the query point's (longitude, latitude) and each site's (longitude,latitude) point for finding the k-nearest neighbouring points (sites). However, the other spatial queries did not involve any spatial processing at runtime within the query as the spatial containment functions (ST_CONTAINS) and both spatial, and non-spatial data joins were already computed in pre-processing steps of data engineering while creating the Silver Delta Tables.

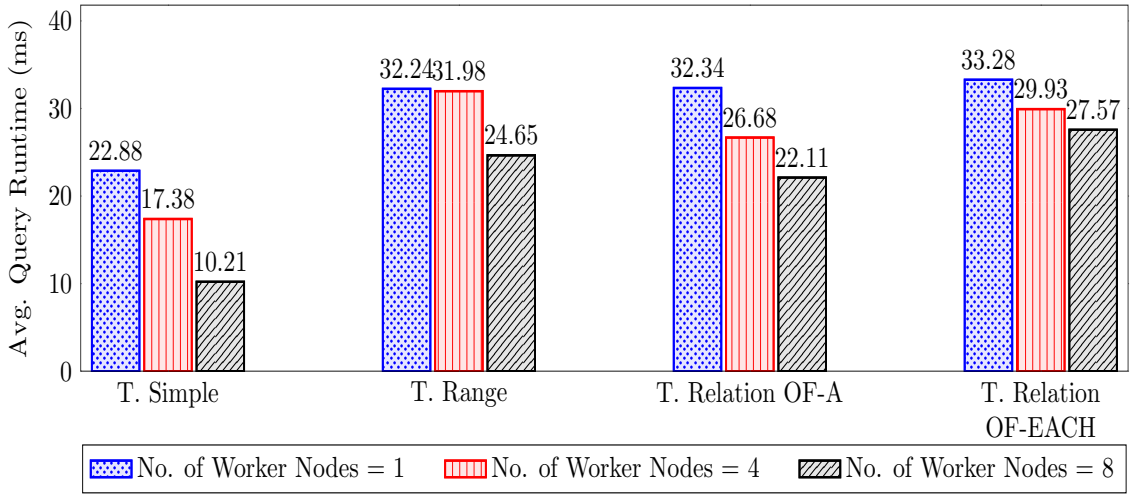


Figure 4.20: Average Query Runtime (millisec) for Temporal Queries

Next, Figure 4.20 showcases the efficacy of Cloud4IoTCity framework for *distributed* processing of *Temporal Queries* on Urban Big Data. From Figure 4.20, it is seen that for each temporal query *the average query runtime decreases with an increase in no. of worker nodes* due to high data parallelism achieved with higher no. of workers that leads to reduced query runtime.

Secondly, the query runtime for each temporal query varies w.r.t. query complexity. Interestingly, all four queries are *aggregation-based* queries that group data based on both spatial and temporal columns. Among these, the simple grouping-on-filtered-records without any aggregates (T. Simple) runs fastest, followed by group-aggregation queries (T. Range, T. Relation OF-A, T. Relation OF-EACH) that involve data (partial results') shuffling across cluster executors to collect per-executor group aggregates and reduce them to single-valued group aggregates. Furthermore, among the *group-aggregation* queries, the large number of records filtered before grouping and lesser no. of group aggregate functions to compute in T. Relation queries (only sum function) resulted in T. Relation queries experiencing lesser runtime as compared to the T. Range query (computing avg, sum, min, max functions).

From Figure 4.21, it is seen that Cloud4IoTCity efficiently processes each spatio-temporal

query in a distributed manner as *the average query runtime decreases with an increase in no. of worker nodes* due to high data parallelism achieved with higher no. of workers leading to reduced query runtime.

Secondly, the single-function (sum) group-aggregation queries (ST. WHAT, ST. WHERE, ST. WHEN) runs faster as compared to the multiple-functions (sum, avg, min, max) group-aggregation queries (ST. Range, ST. Behaviour AT-TIME, ST. Behaviour OVER-RANGE). Besides the number of aggregate functions to compute, the number of records filtered before grouping also impacts the query runtime, as for ST. WHAT and ST. WHERE queries, temporal filtering reduced a much larger number of records than in ST. WHEN query that could only do spatial and threshold filtering without temporal filtering (as time info is to be retrieved); similarly, temporal filtering for a specific time in ST. Behaviour AT-TIME query reduced much larger number of records than in range-based temporal filtering in ST. Behaviour OVER-RANGE and ST. Range queries.

These results answer third validation feature that distributed processing positively impacts query runtime. Results demonstrate that the query runtime reduces as the cluster size increases, as data gets distributed among more executors in larger clusters, and parallel query processing on all executors results in lesser query runtime.

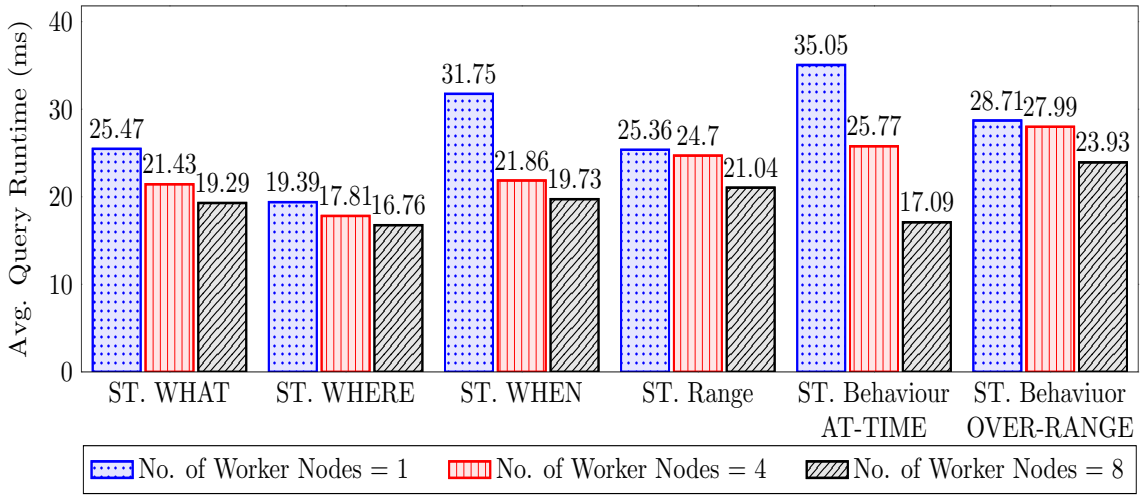


Figure 4.21: Average Query Runtime (millisec) for Spatio-Temporal Queries

4.3.5 Cloud4IoTCity Validation for Efficient Data Layout

This section evaluates the impact of optimized data layout on query runtime. For this, each query is run ten times on each cluster configuration, once for z-ordered data and once for non-z-ordered data, to obtain their query runtimes. Figures 4.22, 4.23, and 4.24 show the query response time results for the three types of queries, comparing the

z-order optimized data layout vs. without z-ordered data layout. Each of these plots shows reduced query response time for z-order optimized data layout compared to when the data storage is without z-order layout optimization. These results answer the fourth validation feature that z-ordering delta tables to optimize the data layout proves beneficial as it results in reduced query runtime.

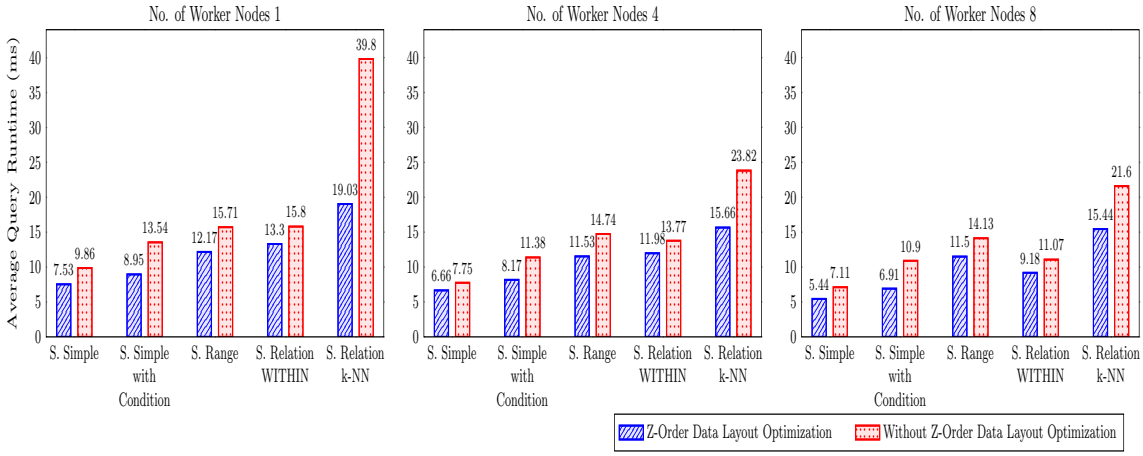


Figure 4.22: Comparison of Data Layout Efficiency for Spatial Queries

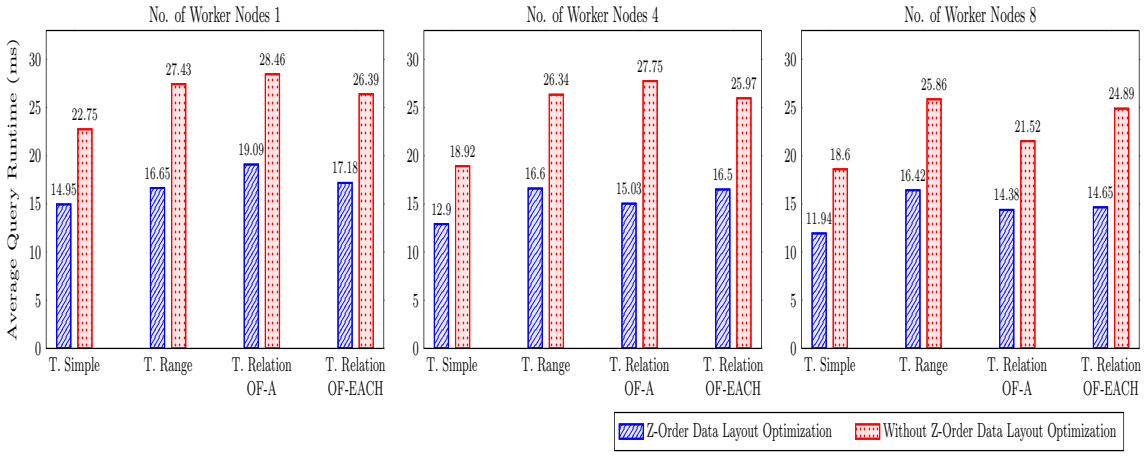


Figure 4.23: Comparison of Data Layout Efficiency for Temporal Queries

4.4 Conclusion

This chapter implements a case study on urban traffic analysis for Dublin city to validate the proposed Cloud4IoTCity framework. The proposed framework provides multi-resolution spatio-temporal descriptive and predictive analysis services that can be accessed through application layer of the framework. For the undertaken case study, a web application named as, 'Urban Analytics' is developed that allows users to access the app

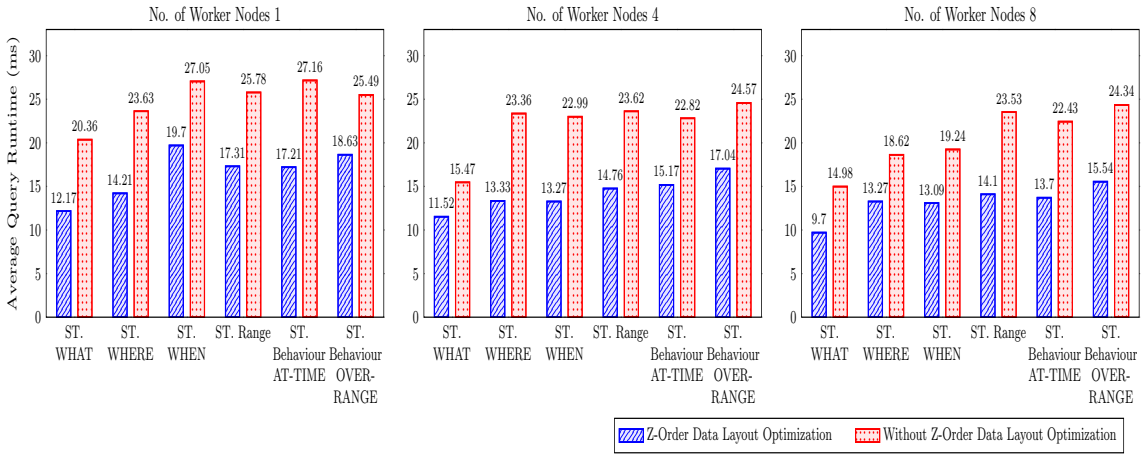


Figure 4.24: Comparison of Data Layout Efficiency for Spatio-Temporal Queries

in a two-tier client-server manner. The case study undertaken in this chapter demonstrates the analysis of smart city IoT data, which is an open data available as batch dataset on Dublin city portal [174].

Next chapter presents efficient service delivery for real-time smart city IoT applications. As, in a two-tier IoT-cloud architecture, real-time applications face unbearable round-trip network latency during uplink and downlink communication between IoT devices and the cloud. Therefore, recent research addresses this issue by using edge computing, which proposes to run suitable application services on edge compute nodes while communicating with the remote cloud, thereby resulting in reduced overall latency for real-time IoT applications [138]. Thus, for efficient service delivery of real-time smart city IoT applications, suitable services of the applications must run on network edge layer to reduce latency i.e., improve application QoS. Therefore, optimal decisions must be taken regarding which service to place at which compute node in the three-tier IoT-edge-cloud architecture to minimize latency and other parameters (example, energy, cost, and so on), which is undertaken in the next chapter.

Chapter 5

Efficient Service Delivery for Real-Time Smart City IoT Applications

Previous chapter validates the proposed framework through a case study on traffic data analysis for Dublin city. The case study uses offline IoT data, so the two-tier client-server architecture is sufficient as there is no incoming data in real-time that could be processed on edge compute nodes. However, real-time smart city IoT applications that operate in a continuous 'sense-process-actuate' control loop must leverage edge compute nodes to run selected services within network edge in order to satisfy latency constraints. Thus, efficient service delivery for such applications empowered with edge computing is undertaken in this chapter.

Efficient service delivery for IoT applications must consider placing application services in a three-tier IoT-edge-cloud architecture in an optimal manner, such that the objective criteria (such as, minimize latency, energy) is achieved, while satisfying resource requirement constraints. Thus, the Service Placement Problem (SPP) reduces to a mathematical optimization problem. This chapter formulates the SPP specifically from smart city perspective, by modeling urban IoT-tier, modeling per-flow latency/bandwidth network constraints, considering federation of multiple telecommunication operators in a smart city. This chapter undertakes to solve the SPP for a set of real-time smart city IoT applications with an aim to minimize both, latency and energy consumption, while satisfying resource, per-flow network, and IoT locality requirements of application services. A Deep Reinforcement Learning (DRL) model is proposed in this chapter to solve the SPP for smart city IoT applications.

The chapter is organized as follows: Section 5.1 presents the system model for service placement in a smart city. Section 5.2 presents the problem formulation. Section 5.3 introduces the proposed DRL model to solve the formulated placement problem and Section 5.4 discusses the experimental results. Finally, Section 5.5 concludes the chapter.

5.1 System Model: Service Placement in a Smart City Scenario

This section describes the system model for service placement in a smart city scenario. Following points shed light on the undertaken SPP for a smart city. Figure 5.1 shows a sample IoT application based on the sense-process-actuate model with diverse application requirements.

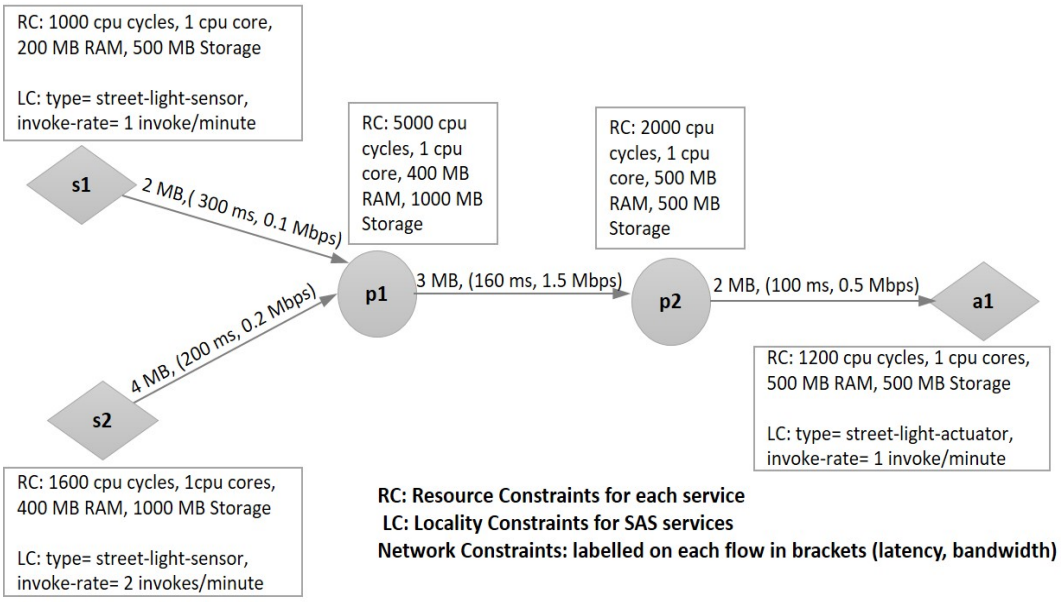


Figure 5.1: A sample sense-process-actuate IoT application

i) This work considers the problem of optimally placing application services of real-time smart city IoT applications. Such applications operate in a *sense-process-actuate* control loop. A small example of smart city IoT application is shown in Figure 5.1. There are two types of services: i) Sensing and Actuation Services (SAS), shown in diamond shape, these services can only be placed in the IoT-tier, ii) processing services, shown in circular shape, these services can be placed on any node in the three-tier architecture.

ii) The smart city applications have three types of constraints: resource constraints (RC in terms of, cpu, memory), network constraints (NC in terms of, per-flow latency, bandwidth), and IoT Locality constraints (LC in terms of, type and invoke rate of sensor/actuator) (all three labelled in Figure 5.1)

iii) This work considers placing services for multiple IoT applications

iv) This work considers the the Multi-access Edge Computing (MEC) [138] as the edge

computing paradigm among the various edge computing paradigms [17]. Because the telecommunication networks have matured over the years and are seen widely deployed across every city.

v) This work consider three-tier IoT-edge-cloud architecture, wherein federation of multiple MEC operators in a city is envisioned. Multiple telecommunications vendor federate with each other to allow cross-operator service placement

vi) This work models the urban IoT-tier comprising of Urban Smart Things (USTs) in a smart city.

To formulate and solve the placement problem, the system model is presented in the next section.

5.1.1 Architecture Model: The 'Urban IoT -Federated MEC - Cloud' architecture

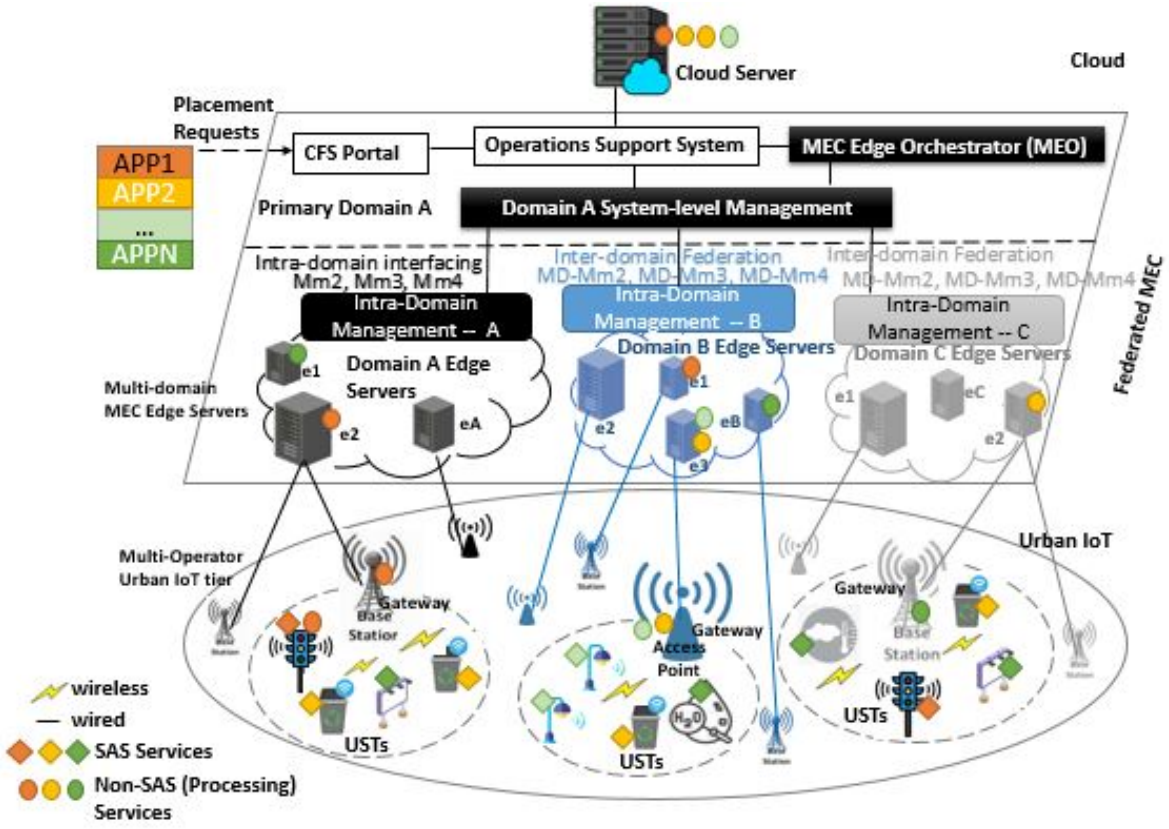


Figure 5.2: System Model for the 'Urban IoT - Federated MEC - Cloud' architecture

The placement nodes in the Urban IoT-Federated MEC-Cloud architecture are mathematically modelled by their respective sets as: set of USTs as $\mathcal{U} = \{1, 2, ..., U\}$, the

set of GWs as $\mathcal{V} = \{1, 2, ..., V\}$, the set of federated multi-domain MEC edge servers as $\mathcal{E} = \{1, 2, ..., E\}$, and the cloud server as *cloud*. The overall set of placement nodes is collectively denoted as $\mathcal{K} = \{0, 1, 2, ..., K\}$ where, the index $k = 0$ denotes the *cloud* server and index $k \in \{1, 2, ..., K\}$ represents the MEC hosts for placement in urban IoT and federated MEC tiers i.e., $\mathcal{U} \cup \mathcal{V} \cup \mathcal{E}$. The placement server k is defined as, $server_k = \langle type_k, id_k, N_k^{services}, R_k \rangle$, which specify type of server, index in its respective category set $\mathcal{U} or \mathcal{V} or \mathcal{E} or cloud$, no. of services currently placed and available compute resources for server k respectively. Here, the available **compute resources** for the server k are defined as, $R_k = \langle r_k^{cpu-freq}, r_k^{n-cores}, r_k^{ram}, r_k^{storage} \rangle$, which specify cpu clock frequency, no. of cpu cores, available ram, and available storage respectively.

To model the **IoT resources** i.e., sensing/actuating capability of USTs (denoted by set $\mathcal{U}$), it is assumed that a single UST device can host multiple sensors/actuators. Thus, the device profile of a UST is mathematically modelled as $u \in \mathcal{U}$ as, $dev_u = \langle sens_u, srates_u, act_u, arates_u \rangle$, which specify list of sensors, list of corresponding sensing rates, list of actuators, and list of corresponding actuating rates respectively. In addition, the *wireless connectivity* of USTs to GWs is defined by the matrix GW of size UxV, such that $GW_{uv} = 1$ specifies that the UST u is connected to the GW v. Also, b_{uv}^{UL}, b_{uv}^{DL} denote the uplink (UL) and downlink (DL) bandwidth for the wireless link between UST u and the GW v respectively. The GWs connect to the MEC system using wired communication links. For wireless-to-wired (or vice-versa) communication, τ represents the propagation delay. Further, for modeling the energy consumed in wireless transmission, P_u^{TX}, P_u^{RX} are defined as the transmit and receive power of the UST u, and P_v^{TX}, P_v^{RX} as the transmit and receive power of the GW v.

5.1.2 Application Model: Real-time Smart City IoT Applications with Resource, Network, and Locality Constraints

The set of smart city IoT applications, received by the CFS portal for placement is denoted as $\mathcal{N} = \{1, 2, ..., N\}$. The placement request of application n is defined as $APP_n = (\mathcal{M}_n, RR_n, \mathcal{F}_n, SAS_n)$, which specify the set of constituent services, matrix of resource constraints, matrix of communication flows between services with their network constraints, set of SAS services with their locality constraints for the application n respectively. For the sake of simplicity, it is assumed the set of applications have same number of services M in each placement request APP_n and the set of services is denoted as $\mathcal{M}_n = \{1, 2, ...M\}$.

The matrix RR_n specify **resource constraints** for all M services of application n. The i^{th} row in the matrix is defined as $RR_{ni} = (rr_{ni}^{cpu-cycles}, rr_{ni}^{cpu-cores}, rr_{ni}^{ram}, rr_{ni}^{storage})$ to

specify no. of cpu cycles, no. of cpu cores, ram, and storage needed for the service i of application n.

The $\mathcal{F}_n$ is a matrix of inter-service communication flows for the application n, such that each $\mathcal{F}_{nij}$ entry is either 0 or $f_{nij} = (\alpha_{nij}, \beta_{nij}, \gamma_{nij})$ to respectively indicate the absence or presence of a communication flow from service i to service j. The non-zero f_{nij} entries in $\mathcal{F}_n$ specify the **per-flow network constraints**, where $\alpha_{nij}, \beta_{nij}, \gamma_{nij}$ respectively denote the amount of data to be transferred (in bytes), the minimum required bandwidth (in Mbps), and the maximum tolerable latency (in msec) for the uplink communication from service i to j for application n.

The set of all SAS services for application n is defined as, $\mathcal{SAS}_n = \{sas_1^n, sas_2^n, ...sas_l^n, ..., sas_L^n\}$, where the l^{th} SAS service in this set defined as, $sas_l^n = \langle id_l^n, type_l^n, sens_l^n, srate_l^n, act_l^n, arate_l^n \rangle$, which specify index in set $\mathcal{M}_n$, type of service as *sensing or actuating*, type of sensor needed, required sensor rate, type of actuator needed, and required actuation rate respectively. The $sens_l^n, srate_l^n, act_l^n, arate_l^n$ values for a SAS service l specify its **locality constraints** for its placement among appropriate UST $u \in \mathcal{U}$. For example, $sas_1^n = \langle 1, sensing, LightSensor, 2, None, 0 \rangle$ represents a sensing service with locality constraints as "light sensor with atleast 2 invokes/min sensing rate", similarly, $sas_3^n = \langle 9, actuating, None, 0, StreetLightActuator, 1 \rangle$ represents an actuating service with locality constraints as "street light actuator at a rate of 1 invoke/min".

The key notations used in the system model are given in Table 5.1. It should be noted that for placing the M_n services of application n, the resource constraints RR_{ni}, the per-flow network constraints $f_{nij} \in \mathcal{F}_n$, and locality constraints for SAS_n should be satisfied for all $\forall i, j \in \mathcal{M}_n$. To *simultaneously place all services of an application in a decision-step*, the multi-action service placement decision for application n is defined as, $a_n = \{a_{n1}, a_{n2}, ...a_{ni}, ...a_{nM}\}$ corresponding to its M services. Note that each $a_{ni} \in \mathcal{K}$ and value for i^{th} placement decision $a_{ni} = k$ represents that server k is chosen for placing the service i of application n.

Table 5.1: Key Notations in the System Model

Symbol(s)	**Meaning**
$u, U, \mathcal{U}$	index, cardinality, and the Set of USTs (Urban Smart Things) in the city
$v, V, \mathcal{V}$	index, cardinality, and the Set of GWs (Gateways) in the city
$e, E, \mathcal{E}$	index, cardinality, and the Set of MEC edge servers of the federated MEC network
	Continued on next page

Symbol(s)	Meaning
$cloud$	the cloud server
$k, K+1, \mathcal{K}$	index, cardinality, and the Set of overall placement servers such that $\mathcal{K} = \mathcal{U} \cup \mathcal{V} \cup \mathcal{E} \cup cloud$
$server_k$	the placement server k defined as, $server_k = \langle type_k, id_k, N_k^{services}, R_k \rangle$, specify type of server, index in its respective category set $\mathcal{U} or \mathcal{V} or \mathcal{E} or cloud$, no. of services currently placed and available resources for server k respectively
R_k	available resources of placement server k defined as, $R_k = \langle r_k^{cpu-freq}, r_k^{n-cores}, r_k^{ram}, r_k^{storage} \rangle$ represent cpu clock frequency, no. of cpu cores, available ram, and available storage respectively
dev_u	device profile of UST u defined as, $dev_u = \langle sens_u, srates_u, act_u, arates_u \rangle$ specify list of sensors, list of corresponding sensing rates, list of actuators, and list of corresponding actuating rates respectively
P_u^{TX}, P_u^{RX}	transmitting and recieving power of the UST u
P_v^{TX}, P_v^{RX}	transmitting and recieving power of the GW v
GW	matrix of UST-to-GW connectivity where, $GW_{uv} = 1$ specifies that the UST u is connected to the GW v
b_{uv}^{UL}, b_{uv}^{DL}	uplink and downlink bandwidth for the wireless link between UST u and the GW v respectively
C_{niuv}^{UL}, C_{niuv}^{DL}	uplink and downlink wireless data transfer rate between the UST u and the GW v for communicating the data of n^{th} application's service i, which is placed on the UST u
τ	propagation delay for the wired communication
$n, N, \mathcal{N}$	index, cardinality, and the Set of urban IoT Applications for placement
$m, M, \mathcal{M}_n$	index, cardinality, and the Set of constituent services in n^{th} IoT application
APP_n	placement request of application n defined as, $APP_n = \langle \mathcal{M}_n, RR_n, \mathcal{F}_n, SAS_n \rangle$ specifies set of constituent services, matrix of resource requirements, matrix of communication flows between services, set of sensing-actuating services for the application n respectively
$\mathcal{F}_n$	matrix of communication flows directed from service i to service j for n^{th} application, such that $\mathcal{F}_{nij}$ is either 0 or f_{nij}
f_{nij}	a communication flow for application n directed from service i to service j defined as, $f_{nij} = \langle \alpha_{nij}, \beta_{nij}, \gamma_{nij} \rangle$ specify amount of data to transfer, min. bandwidth network-constraint, and the max. latency network-constraint for the flow f_{nij} respectively
Continued on next page	

Symbol(s)	Meaning
$\mathcal{SAS}_n$	set of all Sensing Actuating Services for application n defined as, $\mathcal{SAS}_n = \{sas_1^n, sas_2^n, ...sas_l^n, ..., sas_L^n\}$
sas_l^n	l^{th} SAS service for application n defined as, $sas_l^n = \langle id_l^n, type_l^n, sens_l^n, srate_l^n, act_l^n, arate_l^n \rangle$ specify index in set $\mathcal{M}_n$, type of service as *sensing or actuating*, type of sensor needed, required sensor rate, type of actuator needed, and required actuation rate respectively
RR_n	matrix of resource requirements for all the M services of application n, the i^{th} row is defined as $RR_{ni} = (rr_{ni}^{cpu-cycles}, rr_{ni}^{cpu-cores}, rr_{ni}^{ram}, rr_{ni}^{storage})$ specify no. of cpu cycles, no. of cpu cores, ram, and storage needed for the service i of application n
a_n	the multi-action service placement decision for the application n, taken simultaneously for all M services defined as, $a_n = \{a_{n1}, a_{n2}, ...a_{ni}, ...a_{nM}\}$
a_{ni}	service placement decision for a given service i of application n such that $a_{ni} \in \mathcal{K}$ $a_{ni} = k$ represents that server k is chosen for placing the service i of application n
T_n, E_n	latency and energy consumption for the application n corresponding to the placement action a_n
T_{nij}^{Comm}	per-flow communication latency for the flow f_{nij}
E_{nij}^{Comm}	per-flow energy consumption for the flow f_{nij}
E_{per_bit}	energy consumed per-bit for wired communication in the core and cloud network
λ^t, λ^e	scalar weights for latency and energy consumption respectively

5.2 Problem Formulation

5.2.1 Latency Model for an IoT Application

The total latency T_n for the n^{th} IoT application is summation of latency incurred during communication and computation, as follows:

$$T_n = T_n^{Comm} + T_n^{Comp} \tag{1}$$

The communication latency for the application n, T_n^{Comm} is calculated by summation of each **per-flow latency** T_{nij}^{Comm} for all i, j pairs of communicating services $f_{nij} \in \mathcal{F}_n$ as follows:

$$T_n^{Comm} = \sum_i \sum_j T_{nij}^{Comm} \quad \forall i, j \in \mathcal{M}_n \text{ if } \exists f_{nij} \in \mathcal{F}_n \tag{2}$$

The per-flow latency T_{nij}^{Comm} in the above equation depends on the type of communication link between the placement servers k and k', where the communicating services i and j are placed respectively. Given, $a_{ni} = k, a_{nj} = k'$ there are four cases corresponding to the type of communication i) end-to-end wireless i.e., $k, k' \in \mathcal{U} \cup \mathcal{V}$ connected to the same gateway v ii) the k-to-GW uplink wireless and GW-to-k' wired communication iii) the k-to-GW wired and GW-to-k' downlink wireless communication iv) k, k' is the same server. Thus, to calculate the latency T_{nij}^{Comm}, first the **uplink wireless latency** T_{nikv}^{UL-WS} and **downlink wireless latency** $T_{njk'v}^{DL-WS}$ for the flow f_{nij} are defined. For wireless communication the servers $k, k' \in \mathcal{U} \cup \mathcal{V}$ are connected to the gateway v and depending upon whether $k \in \mathcal{U}$ or $k \in \mathcal{V}$ and $k' \in \mathcal{U}$ or $k' \in \mathcal{V}$, there are two cases for each UL-WS latency and DL-WS latency as follows:

$$T_{nikv}^{UL-WS} = \begin{cases} \frac{\alpha_{nij}}{C_{niuv}^{UL}} GW_{uv} & (k \in \mathcal{U} \wedge id_k = u) \\ \frac{\alpha_{nij}}{C_{nivv}^{UL}} GW_{vv} & (k \in \mathcal{V} \wedge id_k = v) \end{cases} \tag{3}$$

$$T_{njk'v}^{DL-WS} = \begin{cases} \frac{\alpha_{nij}}{C_{nju'v}^{DL}} GW_{u'v} & (k' \in \mathcal{U} \wedge id_{k'} = u') \\ \frac{\alpha_{nij}}{C_{njvv}^{DL}} GW_{vv} & (k' \in \mathcal{V} \wedge id_{k'} = v) \end{cases} \tag{4}$$

In eq. (3) case 1, C_{niuv}^{UL} denotes the uplink wireless data transfer rate between UST u and the GW v for sending data from service i of application n application placed on the UST u, and it is calculated as:

$$C_{niuv}^{UL} = b_{uv}^{UL} log_2(1 + \frac{P_u^{TX} h_{uv}^{UL}}{\omega_0}) \tag{5}$$

Similarly, in eq. (4) case 1, the $C_{nju'v}^{DL}$ denotes the downlink wireless data transfer rate between GW v and the UST u' for receiving data for service j of application n application placed on the UST u', and it is calculated as:

$$C_{nju'v}^{DL} = b_{u'v}^{DL} log_2(1 + \frac{P_v^{TX} h_{u'v}^{DL}}{\omega_0}) \tag{6}$$

In eqs. (5) and (6), $h_{uv}^{UL}, h_{u'v}^{DL}$ denote the channel gain for the uplink (UST u to GW v) and downlink (GW v to UST u') connections respectively, and ω_0 is the white noise power. Other notations in these equations for bandwidth and transmit power are already defined in subsection 5.1.1. Considering the case 2 in eqs. (3) and (4) i.e., when the sending service i is placed on GW v, i.e., $id_k = v$, then the uplink v-to-v data rate is $C^{UL}nivv = +\infty$, and if the receiving service j is placed on GW v i.e., $id_{k'} = v$, then the downlink rate from v-to-v is $C^{DL}njvv = +\infty$.

Now, per-flow latency T_{nij}^{Comm} based on four types of communication links between servers k, k' is calculated as follows:

$$T_{nij}^{Comm} = \begin{cases} T_{nikv}^{UL-WS} + T_{njk'v}^{DL-WS} & (k, k' \in \mathcal{U} \cup \mathcal{V}) \\ T_{nikv}^{UL-WS} + \tau & (k \in \mathcal{U} \cup \mathcal{V} \wedge k' \in \mathcal{E} \cup cloud) \\ \tau + T_{njk'v}^{DL-WS} & (k \in \mathcal{E} \cup cloud \wedge k' \in \mathcal{U} \cup \mathcal{V}) \\ 0 & (k = k') \end{cases} \tag{7}$$

Now, the computation latency T_n^{Comp} from equation (1) is modelled, which is calculated as a summation of compute-latency T_{ni}^{Comp} for each service $i \in \mathcal{M}_n$ for the given application n, as follows:

$$T_n^{Comp} = \sum_i T_{ni}^{Comp} \quad \forall i \in \mathcal{M}_n \tag{8}$$

The compute-latency for i^{th} service T_{ni}^{Comp} depends on two factors: cpu cycles requested by that service $rr_{ni}^{cpu-cycles}$, and the fraction of cpu cycles allocated to the service i on the shared placement server k denoted as $r_{nik}^{cpu-cycles}$. Given that $a_{ni} = k$, the T_{ni}^{Comp} is calculated as:

$$T_{ni}^{Comp} = \frac{rr_{ni}^{cpu-cycles}}{r_{nik}^{cpu-cycles}} \tag{9}$$

where, $r_{nik}^{cpu-cycles}$ is calculated by dividing the cpu frequency of the shared server $r_k^{cpu-freq}$ by no. of services placed on it $N_k^{services}$, as follows:

$$r_{nik}^{cpu-cycles} = \frac{r_k^{cpu-freq}}{N_k^{services}} \tag{10}$$

After modeling the overall latency for an IoT application (eqs 1-10), now an application's energy consumption is modelled.

5.2.2 Energy Consumption Model for an IoT Application

The total energy-consumption E_n for the n^{th} IoT application is a summation of energy consumed during communication and computation, as follows:

$$E_n = E_n^{Comm} + E_n^{Comp} \tag{11}$$

The E_n^{Comm} energy consumption during communication for the application n is calculated by summation of each **per-flow** energy consumption E_{nij}^{Comm} for all i, j pairs of communicating services $f_{nij} \in \mathcal{F}_n$ as follows:

$$E_n^{Comm} = \sum_i \sum_j E_{nij}^{Comm} \quad \forall i, j \in \mathcal{M}_n \text{ if } \exists f_{nij} \in \mathcal{F}_n \tag{12}$$

The per-flow energy consumption E_{nij}^{Comm} in the above equation depends on the type of communication link between servers k and k', where the communicating services i and j are placed respectively. There are the same have four cases as eq. (7) corresponding to various type of communication links between placement servers. Given that $a_{ni} = k$ and $a_{nj} = k'$, the per-flow energy consumption E_{nij}^{Comm} is calculated as follows:

$$E_{nij}^{Comm} = \begin{cases} E_{nikv}^{UL-WS} + E_{njkv}^{DL-WS} & (k, k' \in \mathcal{U} \cup \mathcal{V}) \\ E_{nikv}^{UL-WS} + E_{nij}^{wired} & (k \in \mathcal{U} \cup \mathcal{V} \wedge k' \in \mathcal{E} \cup cloud) \\ E_{nij}^{wired} + E_{njkv}^{DL-WS} & (k \in \mathcal{E} \cup cloud \wedge k' \in \mathcal{U} \cup \mathcal{V}) \\ 0 & (k = k') \end{cases} \tag{13}$$

where, the E_{nikv}^{UL-WS} and $E_{njk'v}^{DL-WS}$ respectively denote the UL-WS and DL-WS energy-consumption for the access network communication, and E_{nij}^{wired} is the energy consumed in wired backbone network. As per [182], the energy-consumption for access network and wired backbone network communication are respectively calculated by time-based model and flow-based model. First, the UL-WS and DL-WS energy consumption are calculated using time-based model. Since, from eq.(3) the time (latency) for UL-WS communication $T_{nikv}^{UL-WS} = 0$ for case 2 i.e., when $id_k = v$ and from eq.(4) DL-WS latency $T_{njk'v}^{DL-WS} = 0$ for case 2 when $id_{k'} = v$, so will be the corresponding UL-WS and DL-WS energy-consumption, which is defined as follows:

$$E_{nikv}^{UL-WS} = \begin{cases} P_u^{TX} T_{nikv}^{UL-WS} + P_v^{RX} T_{nikv}^{UL-WS} & (k \in \mathcal{U} \wedge id_k = u) \\ 0 & (k \in \mathcal{V} \wedge id_k = v) \end{cases} \tag{14}$$

$$E_{njkv}^{DL-WS} = \begin{cases} P_v^{TX} T_{njkv}^{DL-WS} + P_u^{RX} T_{njkv}^{DL-WS} & (k' \in \mathcal{U} \wedge id_k = u') \\ 0 & (k' \in \mathcal{V} \wedge id_k = v) \end{cases} \tag{15}$$

In eqs. (14) and (15) $P_u^{TX}, P_u^{RX}, P_v^{TX}, P_v^{RX}$ are the transmit and receive powers of UST u and GW v respectively. Now, E_{nij}^{wired}, the energy consumed in the wired network for the communication flow f_{nij} is calculated, as follows:

$$E_{nij}^{wired} = E_{per\_bit}(\alpha_{nij} * 8) \tag{16}$$

where, E_{per_bit} is the energy consumed per-bit in the shared transport and cloud networks, and α_{nij} is multiplied with 8 for bytes-to-bits conversion. The per-bit energy is calculated as $E_{per\_bit} = h_e E_{edge} + h_c E_{core} + E_{cent\_cloud}$, where h_e, h_c respectively denote the no. of edge and core routers, and $E_{edge}, E_{core}, E_{cent\_cloud}$ respectively denote the energy consumed per-edge element, per-core element, and for the centralized cloud. As per [182], $h_e = 3, h_c = 5, E_{edge} = 37nJ/bit, E_{core} = 12.6nJ/bit,$ and$E_{cent_cloud} = 20\mu J/bit$ in both, uplink (i.e., edge-to-cloud) and downlink (i.e., cloud-to-edge) directions.

Now, the computation energy consumption E_n^{Comp} from equation (11) is modelled, which is calculated as summation of energy consumed in computing E_{ni}^{Comp} for each service $i \in \mathcal{M}_n$ for the given application n, as follows:

$$E_n^{Comp} = \sum_i E_{ni}^{Comp} \quad \forall i \in \mathcal{M}_n \tag{17}$$

The compute-energy for i^{th} service E_{ni}^{Comp} depends on two factors: cpu cycles requested by that service $rr_{ni}^{cpu-cycles}$, and the server characteristic ρ_k for the k^{th} server, where service i is placed. Thus, given that $a_{ni} = k$, the E_{ni}^{Comp} is calculated as:

$$E_{ni}^{Comp} = \rho_k rr_{ni}^{cpu-cycles} \tag{18}$$

where, ρ_k represents the energy consumed per cpu cycle by the placement server k, and $\rho_k = 10^{-26}(r_k^{cpu-freq})^2$ [149].

After modeling the overall energy consumption for an IoT application (equations 11-18), now the service placement optimization problem for a set of IoT applications with relevant constraints is formulated.

5.2.3 The Service Placement Optimization Problem

The SPP is formulated as a multi-objective latency and energy minimization problem over a given set of IoT applications $\mathcal{N}$ with relevant constraints as follows:

$$\min_{a_n} \lambda^t \sum_{n=1}^{N} T_n + \lambda^e \sum_{n=1}^{N} E_n \tag{19}$$

s.t.

$$
\begin{aligned}
C1 :& R_k \geq RR_{ni} \quad (\forall i \in \mathcal{M}_n) \\
C2a :& T_{nij}^{Comm} \leq \gamma_{nij} \quad (\forall f_{nij} \in \mathcal{F}_n) \\
C2b :& C_{niuv}^{UL} \geq \beta_{nij} \quad (\forall f_{nij} \in \mathcal{F}_n, \text{if } k \in \mathcal{U}) \\
C2c :& C_{nju'v}^{DL} \geq \beta_{nij} \quad (\forall f_{nij} \in \mathcal{F}_n, \text{if } k' \in \mathcal{U}) \\
C3 :& a_{nm} = u \quad (\forall sas_l^n \in SAS_n, id_l^n = m \exists u \in \mathcal{U}) \\
& \text{s.t. if } type_l^n = sensing, \text{ then }, \exists x \text{ s.t.} \\
& \quad sens_l^n = sens_u[x] \text{ and } srate_l^n \leq srates_u[x] \\
& \quad \text{else, } type_l^n = actuating, \text{ then }, \exists x \text{ s.t.} \\
& \quad act_l^n = act_u[x] \text{ and } arate_l^n \leq arates_u[x] \\
C4 :& \lambda^t + \lambda^e = 1 \\
& \quad a_n = \{a_{n1}, a_{n2}, ...a_{nM} | a_{ni} \in \mathcal{K}, 1 \leq i \leq M, n \in \mathcal{N}\}
\end{aligned}
$$

where, a_n represents the multiple decision variables – one for each of the M services of the application n. In this way, the optimization problem for one application is solved in each decision step. In eq. 19, the constraint $C1$ specifies resource constraints, while $C2a, C2b, C2c$ denote the per-flow network constraints for latency, uplink and downlink bandwidth respectively, and the constraint $C3$ denotes satisfaction of locality constraints for all SAS services for their placement among appropriate UST subject to validation for type and traffic-rate criteria. In addition, $C4$ constraints the sum of relative weights for latency and energy-consumption objectives (λ^t, λ^e) to be 1. The goal of the formulated optimization problem is to solve the placement for a set of applications with the objective to minimize the weighted sum of latency and energy consumption while satisfying the specified constraints for all applications.

To solve a stochastic optimization decision-problem using the DRL framework, the underlying optimization problem should be formulated as a Markov Decision Process (MDP). Therefore, the SPP in eq. 19 is formulated as a MDP named as, UrbanEnQoSMDP that is presented in the next section.

5.2.4 Proposed MDP Formulation: UrbanEnQoSMDP

A MDP [183] is defined as 5-tuple $< \mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \gamma >$ discrete time stochastic control process, where $\mathbb{S}$ is the state-space, $\mathbb{A}$ is the action-space, $\mathbb{T}$ is the transition probability $p(s'|s, a,)$ that is unknown for model-free DRL agents, $\mathbb{R}$ is the reward function that outputs immediate scalar reward r_t (on the basis of current state s_t, action taken a_t, and the next state s_{t+1}), and $\gamma \in [0, 1)$ is the discount factor to assess the present value

of future rewards. Note that the concept of discount-factor enables the DRL models to maximize the agent reward in the *long-term* as it guides the DRL model to select an optimized action in each timestep – the action with maximum value of *discounted-reward*. This maximization of discounted-reward per timestep enables the DRL model to achieve the maximum total reward for the episode. The DRL models solve the MDP by solving the Bellman's-optimality equation for the optimal action-value function $Q^*(s,a)$ [184] defined as:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{20}$$

where, r is the immediate reward, γ is the discount factor, s, a denote the current state and current action, and s', a' denote the next state and next optimal action.

The SPP in eq. (19) is formulated as an episodic MDP, named as **UrbanEnQoS-MDP**, whose objective is to maximize the total reward (i.e., minimize the latency and energy-consumption) it receives during an episode. An episodic MDP is a MDP with two additions: the terminal state that marks the end of episode and the episode length T that defines the no. of discrete decision timesteps for each episode. In each timestep $t \in \{1, 2, ..., T\}$ the DRL agent is in current state s_t takes action a_t, then receives reward r_t and reaches in the next state s_{t+1}. In the SPP (eq. 19) the placement problem is solved for one application in one decision-step for a total of N applications. Thus, in the proposed UrbanEnQoSMDP, the **episode length** is set as $T = N$, such that the DRL model solves for the application $n = t$ in the t^{th} timestep. Further, the **episode termination** is defined as *successful termination* when the episode ends after N timesteps by solving for N applications, or *failure termination* when at any timestep $t \in T$ the chosen action a_t leads to any of the constraints violation (C1 to C4 eq. 19) for the current application n.

UrbanEnQoSMDP is defined as 6-tuple: $\langle \mathcal{S}, \mathcal{A}, N, r, \gamma, s_1 \rangle$, where $\mathcal{S}$ is the state-space, $\mathcal{A}$ is action-space, N is the episode length, r is the reward function, γ is the discount-factor and s_1 is the initial state. The state-space $\mathcal{S}$, action-space $\mathcal{A}$, and reward function r for the proposed UrbanEnQoSMDP are defined as follows.

State Space $\mathcal{S}$: During each timestep t, the primary domain MEO observes the 'Urban IoT-Federated MEC-Cloud' environment as o_t and constructs $mask_t$ (detailed in algo. 5.1) to form the current state $s_t = (o_t, mask_t)$. The current observation of the (multi-tier placement) environment o_t includes: i) the application index n, which is $n = t$ that represents the application to be solved for placement in the current timestep ii) the placement request of the current urban IoT application n, defined as $APP_n = (\mathcal{M}_n, RR_n, \mathcal{F}_n, SAS_n)$ iii) the current information about the 'Urban IoT-Federated MEC-

Cloud' architecture i.e., placement servers and connectivity information i.e., $network_t = (\mathcal{K}, \mathcal{U}, \mathcal{V}, \mathcal{E}, cloud, GW, b_{uv}^{UL}, b_{uv}^{DL})$. Thus, the current observation o_t is defined as:

$$o_t = (n, APP_n, network_t) \tag{21}$$

and current state s_t for timestep t as:

$$s_t = (o_t, mask_t) \tag{22}$$

The state-space over the episode of length N is defined as:

$$\mathcal{S} = \{s_t | t = n \forall n \in \mathcal{N}\} \tag{23}$$

Action Space $\mathcal{A}$: From the SPP eq. 19, the placement solution at each decision-step is a *multi-action decision to simultaneously place all the* M *services* of the current application n. Thus, in timestep t, the current action a_t is defined as:

$$a_t = a_n = \{a_{n1}, a_{n2}, ..., a_{nM} | a_{ni} \in \mathcal{K}, 1 \leq i \leq M\} \tag{24}$$

The DRL model calculates $M\text{x}K + 1$ values for $Q(s, a)$ estimates over the action-space $\mathcal{K}$ for each of the M services of application undertaken in the current timestep and chooses *one optimal placement action* out of $K + 1$ possible actions as – the action with maximum value of Q-function estimate ($maxQ^*(s, a)$) for each of M services. In this way, the DRL model simultaneously solve the placement for all application services and outputs M optimal placement actions, one for each service. Recall, that for the sake of simplicity, it is assumed to have the same no. of services per application as M, however, this assumption can be easily generalized for varying no. of services per application by adding dummy actions for applications with lesser no. of services. The action-space over the episode of length N is defined as:

$$\mathcal{A} = \{a_t | t = n \forall n \in \mathcal{N}\} \tag{25}$$

Reward Function r: The reward function r defines the immediate reward r_t that the DRL model receives in the current timestep, in response to the action a_t taken by the DRL model for the current state s_t. Since, the DRL models solve the MDP, which is a reward maximization problem whereas, the objectives latency and energy-consumption in eq. 19 are of minimization type. Thus, the negative of eq. 19 is taken as the immediate reward for an intermediate timestep ($t < N$) if the current action a_t does not result in any constraints violation, i.e. when $t < N \wedge done = False \wedge violate =$

$False$. Here, *done* represents episode termination, which is set to $True$ when the current episode terminates either with success after solving the last timestep or with failure at any timestep (last/intermediate). For the terminal state (when $done = True$), the r_t is defined as a huge +ve reward $r_t = 50N$ when the current episode terminates with success after solving the placement for N episodes without incurring constraints violation, or r_t is set as a huge -ve penalty as $r_t = -50N$ when the episode terminates with failure i.e, encountered constraints violation in current timestep. Thus, the reward for the current timestep r_t is defined as:

$$r_t = \begin{cases} -(0.5T_n + 0.5E_n) & (t < N \wedge done = False \wedge violate = False) \\ 50N & (t = N \wedge done = True \wedge violate = False) \\ -(50N) & (t \leq N \wedge done = True \wedge violate = True) \end{cases} \tag{26}$$

Performance Measure: The DRL algorithms optimize discounted reward at each timestep to maximize the cumulative reward (i.e., total reward) for an episode, which is also known as return $G = \sum_{t=1}^{T} \gamma^t r_{t+1}$ [183]. The DRL algorithms are evaluated either at **episodic-level to evaluate the amount of reward** it receives (i.e., cumulative reward for an episode) or at **timestep-level to evaluate how good a policy** it finds (i.e., average cumulative reward) [185]. For the UrbanEnQoSMDP, the Average Cumulative Reward (ACR) i.e., accumulated reward per timestep is used, which clearly shows the value of policy learned and whether the algorithm has stopped learning (i.e., converged when ACR is flat). The average cumulative reward is calculated by dividing the cumulative reward by episode length N:

$$Avg.CumulativeReward(ACR) = \frac{\sum_{t=1}^{N} \gamma^t r_{t+1}}{N} \tag{27}$$

5.3 Proposed DRL Model: UrbanEnQoSPlace

The key ideology of dueling DQN to separately estimate the state-value function $V(s)$, so as to avoid the exhaustive exploration of the action-space for lesser valuable states resulting in faster training and better convergence [186], motivates the design of proposed UrbanEnQoSPlace DRL model. Because, in the urban placement scenario it would be helpful to have an estimate of lesser valuable state(s) as when there are insufficient resources for a service, then all placement actions in the action-space except the cloud are meaningless, in such case, using the dueling architecture avoids exhaustive exploration based on its learning that the current state is less valuable. Standard DQN [184] does not have this feature and so it takes longer to converge. The Dueling DQN [186] is discussed in brief before describing the proposed UrbanEnQoSPlace model.

Dueling Deep Q-Networks: The dueling DQN estimates $Q(s,a)$ values as per eq. 28 [186] based on two separate functions: the action-independent state-value function $V(s)$ that indicates the value of a state, and the action-dependent advantage function $A(s,a)$ that estimates the advantage of choosing an action relative to all other actions in that state. To implement eq. 28, the dueling architecture first splits its deep network into two branches of Fully Connected Layers (FCL) that separately estimate $V(s)$ and $A(s,a)$ respectively, then a forward-pass in the advantage module subtracts the mean of the advantage function from each $A(s,a)$ value, which is then aggregated with $V(s)$ to output $Q(s,a)$ estimates for all actions in the action-space $\mathcal{A}$ for the given input state s.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a} A(s,a';\theta,\alpha)\right) \tag{28}$$

where, θ, α, β denote the parameters for convolutional layers, advantage function FCLs, and value function FCLs respectively and $|\mathcal{A}|$ denotes the dimensionality of action-space. Based on dueling DQN, the UrbanEnQoSPlace DRL model is proposed for solving the formulated UrbanEnQoSMDP.

5.3.1 UrbanEnQoSPlace DRL Model: Proposed Deep Network Architecture for Multiple Actions

This section describes the proposed UrbanEnQoSPlace DRL model with customized deep network architecture to support multi-actions output. Then, in next section (section 5.3.2) a customized mask-based policy: "ϵ-greedy with mask" is proposed, which will be used by the proposed DRL model. Then, the optimal service placement done by the proposed UrbanEnQoSPlace DRL model is discussed in detail (section 5.3.3).

The architecture of the UrbanEnQoSPlace model shown in Figure 5.3, comprises three key components: i) Main Dueling Deep Q-Network (MDDQN), ii) Experience Replay Buffer, and iii) Target Dueling Deep Q-Network (TDDQN). For MDDQN, to output $Q(s,a)$ estimates simultaneously for M number of actions $\{a_1, a_2, ..., a_M\}$, the deep network's advantage module is constructed with M independent streams parameterized by their respective weights $w_1, w_2, ...w_M$ – such that each of the i^{th} advantage stream $A(s,a_i;w_i)$ outputs advantage-function estimates over its action-space $\mathcal{K}$ for placing the i^{th} service of the current application n inferred from the input state. Since the value-function $V(s)$ is independent of actions, it is constructed as an independent single stream of FCL layers with weights w_0. As the SPP has no image features, the convolutional layers from the standard dueling architecture are omitted. The Q-estimates for placing i^{th} service are obtained by aggregating $V(s,w_0)$ with mean-adjusted $A(s,a_i;w_i)$ to output

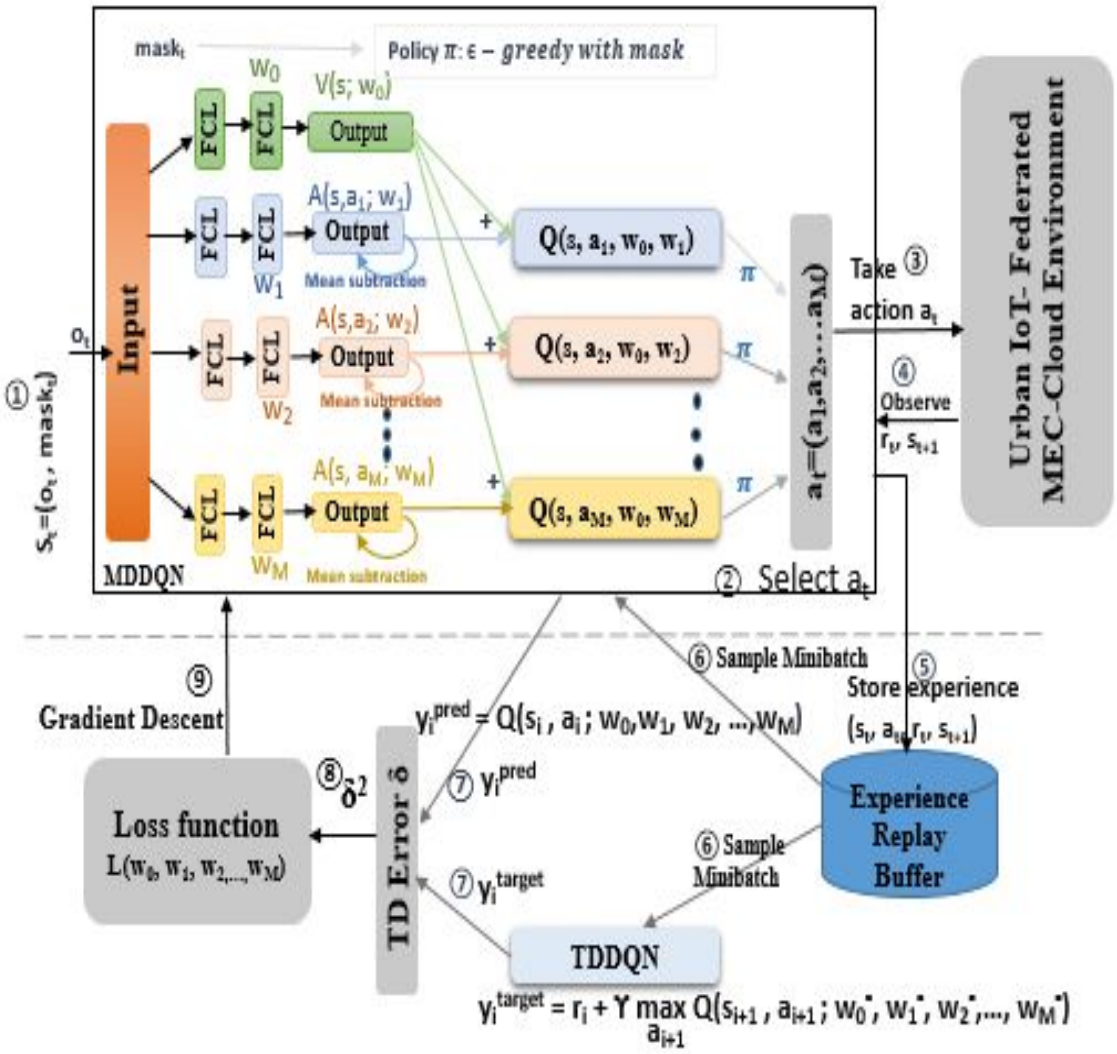


Figure 5.3: The Proposed UrbanEnQoSPlace DRL Model

$Q(s, a_i; w_0, w_i)$ over the action-space $\mathcal{K}$. The architecture of the target network TDDQN is exactly the same as the main network MDDQN and the role of experience replay is elaborated in the training phase for the proposed DRL model.

5.3.2 UrbanEnQoSPlace: Proposed Policy based on Action-Masking

This section presents the proposed novel policy "ϵ-greedy with mask" given in Algorithm 5.2 that integrates a boolean action mask with the popularly used ϵ-greedy policy, such that the $True, False$ values in the mask respectively correspond to valid, invalid placement actions as per locality constraints for SAS services. To better appreciate the

proposed policy, first the construction of action-mask by the primary domain's MEO presented in Algorithm 5.1 is discussed. During each timestep t, the UrbanEnQoSPlace model solves the placement problem for one application $n = t$ so, the MEO takes as input the current timestamp t, the current application APP_t, and the set of USTs $\mathcal{U}$ to determine the action-mask $mask_t$ (line 1). The $mask_t$ is a boolean matrix of size Mx$K + 1$, where rows and columns represent application services and placement servers respectively. During each timestep, the $mask_t$ is initialized as $False$ for all rows and columns (lines 3-4) and then it is updated (row-wise) for each service such that if the service i is a non-SAS service (line 5), then all servers are valid as-per locality constraints, hence, all columns (placement servers) for that row (service) are set to $True$ (line 6). However, if the service i is a SAS-service (line 7), then the MEO iterates over the set of USTs (line 8) to find those valid USTs $u \in \mathcal{U}$ (where $id_k = u$) that satisfy the senor/actuator type and traffic-rate criteria for the given SAS service i (lines 10, 13) to unmask the server k for service i by setting the column k for row i in $mask_t$ as $True$ (lines 11, 14). Note that the servers (columns) that do not match the locality criteria are already masked for the SAS service i, as the $mask_t$ matrix already holds the value $False$ for them. After evaluating the valid servers for all services, the MEO returns the $mask_t$ to the policy implementation module (line 21). Note that, only the locality constraints could be integrated into the model policy, since they specify static information, whereas the resource and network constraints specify dynamic information, so those constraints must be handled during model training.

Now, the proposed policy "ϵ-greedy with mask" is discussed, which takes as input the current state $s_t = (o_t, mask_t)$ (eq. 22) that includes current system state o_t and the mask for the current application $mask_t$ and outputs simultaneous multiple actions $\{a_1, a_2, ...a_M\}$ for placing M services of the current application. In lines 4-10, the $mask_t$ is processed for each service i to determine the set of candidate-servers-per-service, denoted by $cand\_server_{ni}$ by adding the valid server k if $mask_t[i][k] = True$. Clearly, for a SAS service i, the $cand\_server_{ni} \subset \mathcal{K}$, thus, it reduces the search-space for corresponding placement actions, and also ensures the *apriori satisfaction* of locality constraints. Now, the policy solves the explore-exploit dilemma as: with ϵ probability random action(s) are explored for placing all M services (lines 12-15), while with probability $1 - \epsilon$ the current best action is exploited for all M actions by taking arg max operator over Q-values of *only* $cand\_servers_{ni}$ (lines 16-20). Finally, the policy outputs the action to be taken in the current timestep for the application n as: $a_t = \{a_1, a_2, ..., a_M\}$ (line 21).

Algorithm 5.1 Construction of the Action Mask by the Primary MEC Domain's MEO

1: **function** ENV_CALC_MASK($t, APP_t, \mathcal{U}$)
2: **Initialization**: Initialize application index to solve in current_timestamp as: $n = t$; Extract $SAS_n = SAS_t$ from APP_t; Initialize a boolean matrix $mask_t$ with M rows and $(K + 1)$ columns
3: **for** $i \leftarrow 1$ to M **do**
4: $mask_t[i][0 : K] \leftarrow False$
5: **if** $\forall sas_l^n \in SAS_n, \quad id_l^n \neq i$ **then**
6: $mask_t[i][0 : K] \leftarrow True$
7: **else if** $\exists\, sas_l^n \in SAS_n \quad s.t.\ id_l^n = i$ **then**
8: **for** $u \in \mathcal{U} \wedge (\exists\, k \in \mathcal{K} \mid id_k = u)$ **do**
9: **if** $type_l^n = sensing, \quad \exists\, x$ **then**
10: **if** $sens_l^n = sens_u[x] \wedge srate_l^n \leq srates_u[x]$ **then**
11: $mask_t[i][k] \leftarrow True$
12: **else if** $type_l^n = actuating, \quad \exists\, x.$ **then**
13: **if** $act_l^n = act_u[x] \wedge arate_l^n \leq arates_u[x]$ **then**
14: $mask_t[i][k] \leftarrow True$
15: **end if**
16: **end if** {\\End of line 12 else-if block}
17: **end if** {\\End of line 9 if-elseif block}
18: **end for** {\\End of line 8 for block}
19: **end if** {\\End of line 5 if- elseif block}
20: **end for** {\\End of line 3 for block}
21: **return** $mask_t$
22: **end function**

5.3.3 UrbanEnQoSPlace: Optimal Service Placement

The placement process initiates when the primary domain MEO receives the request for placing a set of applications from the CFS portal. Practically, for service placement in the real MEC system, the MEC operator would first need to train the UrbanEnQoSPlace DRL model by implementing Algorithm 5.3 on its MEO, so that the model learns by taking actions in the MEC operator's real network environment and adjusts the model parameters (weights and biases of the deep network) accordingly. Then, after training when the model has converged, it needs to be saved with its weights (either as a full tensorflow model, or only the weights in numpy/HDF5) to load the trained model with the updated weights into the MEO for making optimized placement decisions that maximize reward (i.e., minimize latency and energy-consumption) for the target MEC operator. The training procedure for the UrbanEnQoSPlace model, outlined in algorithm 5.3 is as follows: first, the MDDQN weights are initialized randomly and TDDQN weights are copied from MDDQN, the experience replay buffer $\mathbb{D}$ with capacity C is initialized, and the episode length is set as N (line 1). Then in each training iteration, the model first gathers the current state s_t from MEO (line 3), which also involves a call to algo. 5.1 to get the mask $mask_1$ for the current (1st) application. The current state in its raw-form

Algorithm 5.2 Algorithm for the Proposed "ϵ-greedy with mask" Policy

1: **Input:** current state: $s_t \leftarrow (o_t, mask_t)$, where $mask_t \leftarrow$ ENV_CALC_MASK$(t, APP_t, \mathcal{U})$ is obtained from algo. 5.1; o_t is observed by the MEO, and $\epsilon \in [0, 1]$
2: **Output:** current policy: $\pi(a_t|s_t; w_0, w_1, ...w_M) = \pi(a_i|s_t, 1 \leq i \leq M$ and $\forall a_i \in \mathcal{K})$
3: **Initialization:** Initialize current_app $n = t$; initialize M sets to store candidate/unmasked servers for each service i denoted as, $cand\_server_{ni} = \{\}$
4: **for** $i \leftarrow 1$ to M **do**
5: **for** $k \leftarrow 0$ to K **do**
6: **if** $mask_t[i][k] = True$ **then**
7: add server k to the candidate set for service i as, $cand\_server_{ni} \leftarrow cand\_server_{ni} \cup k$
8: **end if**
9: **end for**
10: **end for**
11: Choose a random number $r \in [0, 1]$
12: **if** $r < \epsilon$ **then**
13: **for** $i \leftarrow 1$ to M **do**
14: $a_i \leftarrow$ random k chosen from $\in cand\_server_{ni}$
15: **end for**
16: **else**
17: **for** $i \leftarrow 1$ to M **do**
18: $k \leftarrow \arg\max_{a_i} Q(s_t, a_i; w_0, w_1, ..., w_M | a_i \in cand\_server_{ni})$
19: **end for**
20: **end if**
21: **return** $\{a_1, a_2, ...a_M\} \leftarrow \pi(a_i|s_t, 1 \leq i \leq M)$ the derived ϵ-greedy policy with $mask_t$

is not suitable for stable learning, so linear-normalization pre-processing is applied, and also the mask $mask_1$ is input to the policy module (algo. 5.2) to get the current policy, which specifies the chosen simultaneous multiple actions for the current state (line 4). For each timestep t, the composite action a_t obtained from policy is implemented in the environment and the resulting reward and next state are observed (lines 6-7). The next state is then pre-processed (line 8) and the experience tuple (s_t, a_t, r_t, s_{t+1}) is stored into the experience replay buffer $\mathbb{D}$ (line 9). These steps are repeated many times, until the buffer has atleast $\mathbb{D}_b$ (batch size) number of samples for each timestep for further training. The model then sends a uniformly sampled minibatch $\mathbb{D}_b$ from the replay buffer to both networks TDDQN and MDDQN (line 11) that respectively calculates the target y_i^{target} and predicted y_i^{pred} Q-values for each i^{th} experience tuple $\mathbb{T}_i \in \mathbb{D}_b$ as per eq. 29 and 30:

$$y_i^{target} = \begin{cases} r_i & \text{if } episode \text{ terminates at timestep } i+1 \\ r_i + \gamma \underset{a}{max} Q(s_{i+1}, a_{i+1}, w_0^-, w_1^-, \ldots, w_M^-) & \text{otherwise} \end{cases} \tag{29}$$

$$y_i^{pred} = Q(s_i, a_i, w_0, w_1, \ldots, w_M) \tag{30}$$

The goal of training is to make MDDQN predictions y_i^{pred} as close as possible to the real values obtained from environment y_i^{target}. In other words, the loss-function should be

Algorithm 5.3 Optimal Service Placement by the Proposed UrbanEnQoSPlace DRL Model

1: **Initialization:** Initialize replay memory $\mathbb{D}$ with capacity C; MDDQN parameters $w_0, w_1, w_2, ..., w_M$ with random weights; TDDQN parameters as: $w_0^- = w_0$, $w_1^- = w_1$, $w_2^- = w_2$, ..., $w_M^- = w_M$; episode length as N; no. of training episodes as *num_episodes*
2: **for** *episode* $\leftarrow 1$ to *num_episodes* **do**
3: Get Initial state from environment $s_1 \leftarrow (o_1, mask_1)$, which internally calls algo. 5.1 to get $mask_1$
4: Pre-process initial state: $\phi_1 \leftarrow \phi(s_1)$ and derive the policy "ϵ-greedy with mask" using algo. 5.2
5: **for** timestep $t \leftarrow 1$ to N **do**
6: Choose an action a_t as per proposed policy (algo. 5.2)
7: Execute action a_t in the environment and observe reward r_t and next state s_{t+1}
8: Pre-process next state: $\phi_{t+1}(s_{t+1}) \leftarrow \phi(s_{t+1})$
9: Store the experience tuple $(\phi_t, a_t, r_t, \phi_{t+1})$ into the replay buffer $\mathbb{D}$
10: Uniformly sample a random minibatch $\mathbb{D}_b$ of tuples $(\phi_i, a_i, r_i, \phi_{i+1})$ from $\mathbb{D}$
11: Use TDDQN and MDDQN to calculate the target and predicted Q-value respectively for each sample in $\mathbb{D}_b$ as per eq. 29 and 30
12: Perform a Gradient descent on Loss function $L(w_0, w_1, w_2, ..., w_M)$ (defined in eq. 31) and update all weights of MDDQN as per eq. 32
13: Every T_0 steps update weights of TDDQN as: $w_0^- = w_0$, $w_1^- = w_1$, $w_2^- = w_2$, ..., $w_M^- = w_M$
14: Store reward, latency and energy-consumption for current timestep
15: **end for**
16: Calculate and store the average cumulative scores for reward, latency and energy-consumption for the current episode
17: **end for**
18: **return** The trained DRL network MDDQN with its weights $w_0, w_1, w_2, ..., w_M$

minimized, which represents the Mean-Squared Error (MSE) between the two values over the minibatch (eq. 31). For this, the TD-error for each i^{th} experience tuple is defined as $\delta_i = y_i^{target} - y_i^{pred}$ and the MSE over batch is taken to define the loss function as:

$$L(w_0, w_1, w_2, ...w_M) = \frac{1}{\mathbb{D}_b} \sum_{i=1}^{\mathbb{D}_b} (\delta_i)^2 \tag{31}$$

As the loss-function is parameterized by $M + 1$ weights, gradient descent on the loss-function w.r.t each of these network weights $w_l \forall l \in \{0, 1, 2, ..., M\}$ is applied and the Adam optimizer is used to update all the $M + 1$ weights of MDDQN network (line 12) as eq. 32:

$$w_l = w_l + \alpha \nabla_{w_l} L(w_0, w_1, w_2, ...w_M) \quad \forall l \in \{0, 1, 2, ..., M\} \tag{32}$$

where, α is the learning rate and ∇_{w_l} denotes the partial derivative w.r.t l^{th} weight. The weights of MDDQN are updated instantly, while the weights of TDDQN are copied periodically from MDDQN after every T_0 steps (line 13). The metrics (reward, latency, and

energy-consumption) are recorded during each timestep (line 14) and the average cumulative score for each of them for each episode (line 16) is calculated. After completing the training for *num_episodes*, the MDDQN with its current weights is now a learned DRL model that can be used to output optimal service placement actions for the UrbanEnQoSMDP that maximize reward (i.e., minimize latency and energy-consumption).

5.4 Experimental Results

The performance of the proposed UrbanEnQoSPlace DRL model is analyzed from various aspects - performance of the DRL model, scalability of the model, and efficacy of the model policy (ϵ-greedy with mask). Python and Tensorforce [187] are used for carrying out experiments. Tensorforce is an open-source DRL framework built on the top of Google's Tensorflow and allows to built custom RL environments quite similar to openAI Gym. All experiments are done on a Dell laptop with intel i3 processor operating at 2.30 GHz with 4 GB ram and windows 10 operating system. First simulation and hyperparameter settings are described and then, the results obtained for each experiment are discussed.

5.4.1 Simulation and Hyperparameter Settings

A custom RL environment is developed in Tensorforce to model the UrbanEnQoSMDP simulation scenario comprising of: i) $K+1$ placement servers (one cloud server and rest K servers as sum of ESs, USTs, APs), ii) N number of urban IoT applications with each application having M constituent services. The values $N = 10$, $M = 5$ and $K = 20$ are set and also the proposed UrbanEnQoSPlace DRL model is implemented. The details for various MDP entities and proposed model are as follows:

Placement Servers

The compute resources for cloud server are set as $(1000GHz, 8, 256GB, 1000GB)$ and the K servers for IoT-MEC tiers are randomly partitioned into 4 *heterogeneous* categories as: tiny $(0.3GHz, 1, 2GB, 20GB)$, small $(0.5Ghz, 1, 4GB, 40GB)$, medium $(0.7GHz, 2, 8GB, 80GB)$, and large $(1GHz, 4, 16GB, 160GB)$ [162]. Further, $\frac{1}{3}K$ servers are chosen randomly from the set of $tiny \cup small \cup medium$ to denote USTs and APs (GWs) and remaining servers denote the MEC edge servers. To model the urban IoT access network enabled by public WiFi, the chosen USTs are randomly distributed in two groups- near-to-AP and far-from-AP, such that the bandwidth (both, UL and DL) and latency between each UST and the AP is randomly assigned in the range of $[15, 20]$ Mbps and $[1, 5]$ msec for group 1; and $[8, 12]$ Mbps and$[8, 12]$ msec for group 2 respectively. In addition, each UST is

modelled with 5 sensors and 3 actuators. The other parameters for wireless access network are as: the transmit and receive powers for USTs $P_k^{TX} = P_k^{RX} = 100$ mW, for APs $P_{AP}^{TX} = P_{AP}^{RX} = 25$ mW, channel gain 4 dBi, white noise power $\omega_0 = -100$ dBm, round-trip propagation delay 15 msec [155]. The weights are set as: $\lambda^e = \lambda^t = 0.5$

Urban IoT Applications

For each application, its application graph is manually designed based on the *sense-process-actuate* model (refer Figure 5.1 for a small example) with a total of M services. The resource demands for each service are generated randomly in the range of $[1000, 10000]$ cpu-cycles, $[1, 4]$ cores, $[50, 500]$ MB ram, and $[200, 8000]$ MB storage respectively. And, the per-flow communication parameters are set randomly in the given ranges as: amount of data $\alpha_{nij} = [2, 6]$ MB, bandwidth constraint $\beta_{nij} = [0.1, 1.0]$ Mbps, and latency constraint $\gamma_{nij} = [100, 1000]$ msec [162]. Similarly, for SAS services the traffic rates are chosen randomly in the range [1,3] invoke/min.

Hyperparameters for UrbanEnQoSPlace DRL Model

The UrbanEnQoSPlace DRL model comprises $M + 1$ streams – M streams for the advantage module and one stream for value module for both the networks ie., MDDQN and TDDQN. Each stream in the neural network comprises of 2 FCLs with 64 neurons each and tanh activation. The capacity of replay memory is set as 10000, no. of training episodes as 3000, and episode length (max. episode timesteps) as N. According to [148], various DRL hyperparameters are set as given in table 5.2 below.

Table 5.2: DRL Hyperparameters

Hyperparameter	Value
Discount factor γ	0.99
Optimizer	Adam
Learning rate α	0.001
Batch Size $\mathbb{D}_b$	64
Update Frequency for target network T_0	16

5.4.2 Performance Analysis

In this section, the results for performance analysis of UrbanEnQoSPlace are analyzed against the following state-of-the-art DRL algorithms:

i) **Value Based DRL Algorithms:** These include the standard Deep Q-Network (DQN) [184] and Double DQN [188] algorithms. The UrbanEnQoSPlace model, based on Dueling DQN [186] also falls in this category. Service placement works that use VB DRL algorithms include [148, 152, 153, 150, 154, 149, 151, 155, 158].

ii) **Policy Gradient DRL Algorithms:** The algorithms PPO [189], Trust Region Policy Optimization (TRPO)[190], and Advantage Actor-Critic (A2C)[191] are chosen for performance comparison. Service placement works that use PG DRL algorithms include [157, 158, 159, 160, 161, 156].

iii) **Random Heuristic:** In this approach, the placement servers are assigned randomly to the services from the action-space. Numerous works [141, 144, 150, 159, 160, 163] compare their work with random heuristic.

Note that the results of proposed DRL model could not be reported against any specific research work due to underlying differences in application, network, or system models. Thus, the most commonly used DRL algorithms for SPP from literature are enlisted and the performance of the proposed model is compared against those standard DRL algorithms in solving the formulated placement MDP, the UrbanEnQoSMDP. Also, note that for performance evaluation, the multi-action output and action-masking features are implemented in all these baseline algorithms for fairness and also, similar to [152] the model hyperparameters are kept same for all baselines to ensure fairness of results.

Convergence Analysis

The convergence performance of the UrbanEnQoSPlace model against the VB DRL algorithms is shown in figure 5.4. Recall that a DRL agent receives a huge negative/positive reward as penalty/reward when the episode terminates with failure/success respectively. As it is known that DRL algorithms learn by experience, so during the initial episodes when the DRL agent knows nothing it is expected to have huge negative rewards and slowly as the DRL agent undergoes learning the algorithm converges towards successive positive rewards. From the results, it is inferred that all the VB algorithms confirm to such a trend – initially huge negative penalties followed by successive positive rewards upon convergence after few hundred episodes. And the occasional dips upon convergence are due to the random exploration done by the DRL agent to balance the explore-exploit dilemma for solving RL problems. Additionally, as far as the performance of our model relative to DQN and Double DQN is concerned, it is inferred from the results that *UrbanEnQoSPlace performs superior to both, DQN and Double DQN* as it exhibits quicker convergence and stable training (i.e., less exploration upon convergence) as compared to the other two VB algorithms.

Next, the convergence performance of UrbanEnQoSPlace against the PG DRL algorithms is shown in figure 5.5. From the results, it is inferred that *UrbanEnQoSPlace outperforms all the chosen PG algorithms* as it converges after few hundred episodes

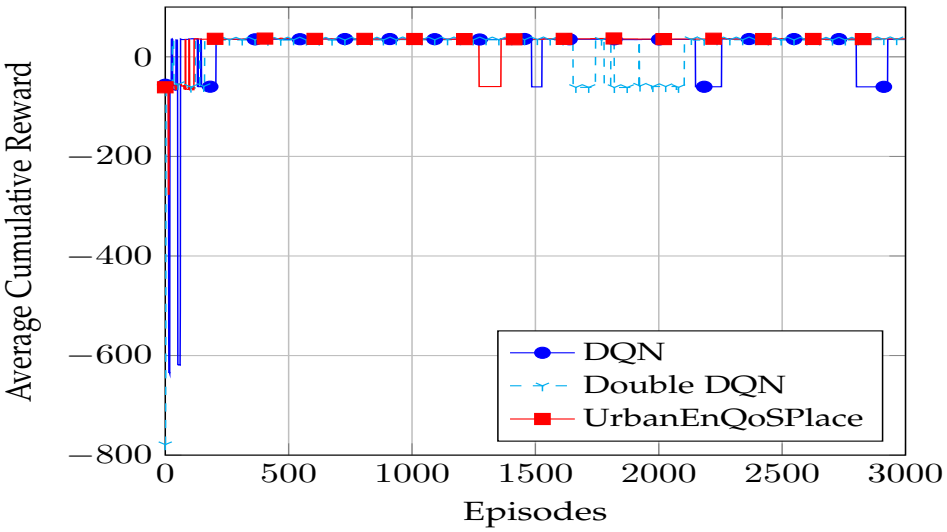


Figure 5.4: UrbanEnQoSPlace vs. Value-Based DRL algorithms

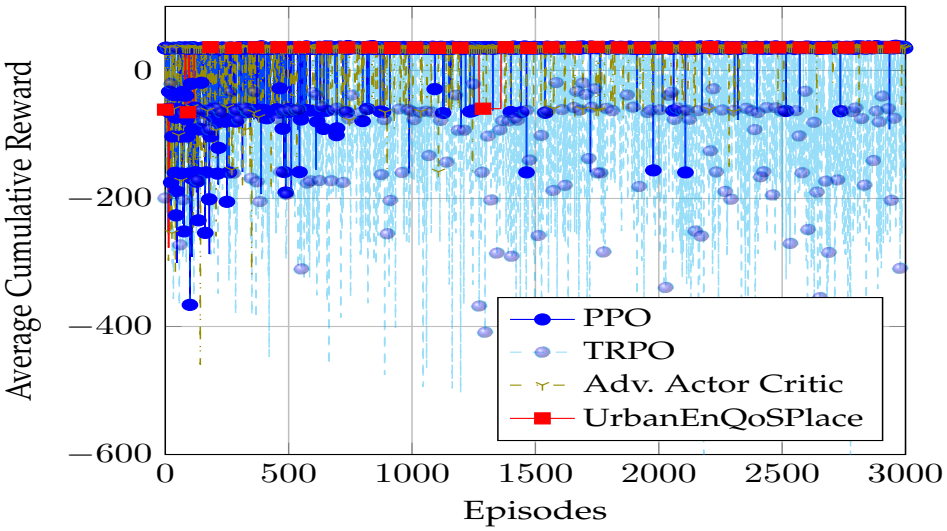


Figure 5.5: UrbanEnQoSPlace vs. Policy-Gradient DRL algorithms

with continuous higher positive rewards, while the PG algorithms keep oscillating between negative and positive rewards throughout the entire training process and could not lead to continuous positive rewards even until the end of experiment. Hence, it is concluded that the PG DRL algorithms could not converge/learn an optimal placement policy for solving the placement MDP. However, among the PG algorithms, the A2C rewards are less negative (slightly better) than the PPO rewards, which are less negative than TRPO rewards. One plausible reason for the poor performance of PG class of algorithms is that they are well suited for RL problems with continuous actions, while the UrbanEnQoSMDP problem comprises discrete actions for which the VB class of algorithms outperform the PG class of algorithms. Finally, figure 5.6 shows the convergence performance of the learning-based UrbanEnQoSPlace model against the random heuris-

tic that does not undergo any learning. The obtained results clearly indicate the quicker convergence of UrbanEnQoSPlace in comparison to the random heuristic .

Note that due to the significantly poor performance of the chosen PG and random algo-

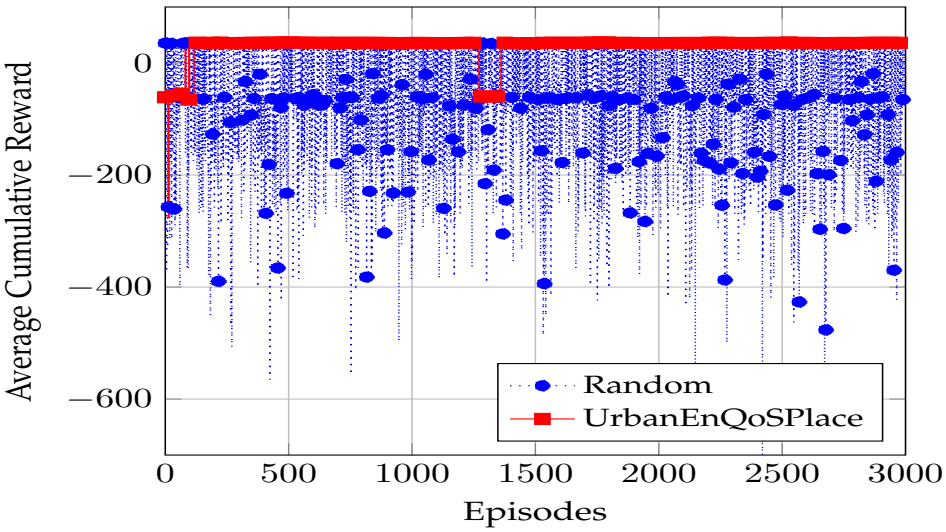


Figure 5.6: UrbanEnQoSPlace vs. Random Heuristic

rithms, these algorithms are excluded from further analysis in subsequent sections.

Reward Analysis

The rewards obtained in terms of, average cumulative reward (eq. 27) by the proposed UrbanEnQoSPlace model relative to DQN, and Double DQN algorithms in solving the placement MDP are shown in figure 5.7. From the results, it is inferred that the Ur-

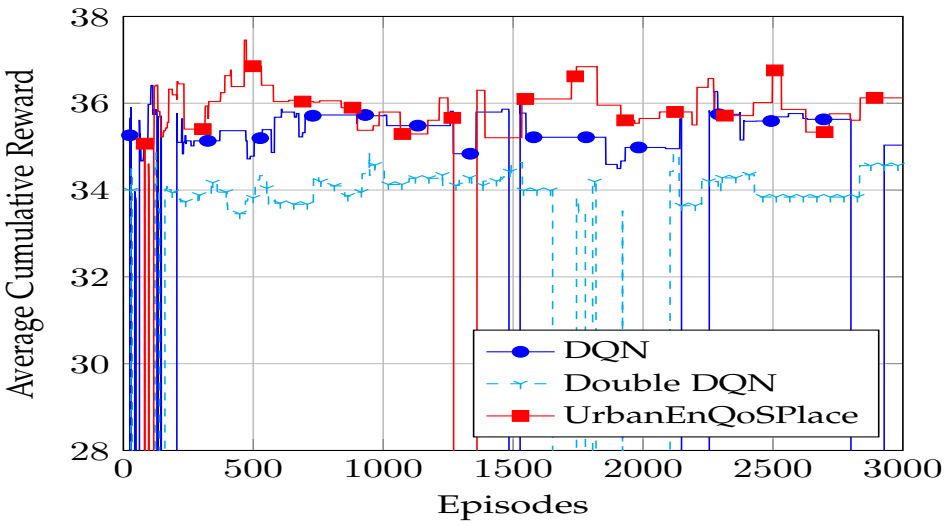


Figure 5.7: Reward Analysis

banEnQoSPlace model *consistently achieves the highest reward* and stable training with

lesser exploration as compared to the DQN and double DQN algorithms. Secondly, the performance of DQN is better than the Double DQN with higher rewards.

Runtime Analysis

The results for runtime analysis of UrbanEnQoSPlace model relative to DQN and Double DQN are shown in figure 5.8. In this experiment, each model's runtime for both the training and testing phases is analyzed. In the training phase, the underlying DRL model undergoes learning to update the model weights, while the testing phase uses the trained model without any weights' updation. The training process is run for 3000 episodes and testing process for 500 episodes for each DRL model. Each experiment is repeated five times to obtain the average training and testing times. From the results, it is inferred that UrbanEnQoSPlace incurs the minimum average training and testing times as compared to the DQN and Double DQN algorithms. Note that the drastic variation in training and testing times is due to the fact that the model training involves the compute-intensive gradient calculation and weights updating steps during each episode, which, of course, are not performed in the testing phase.

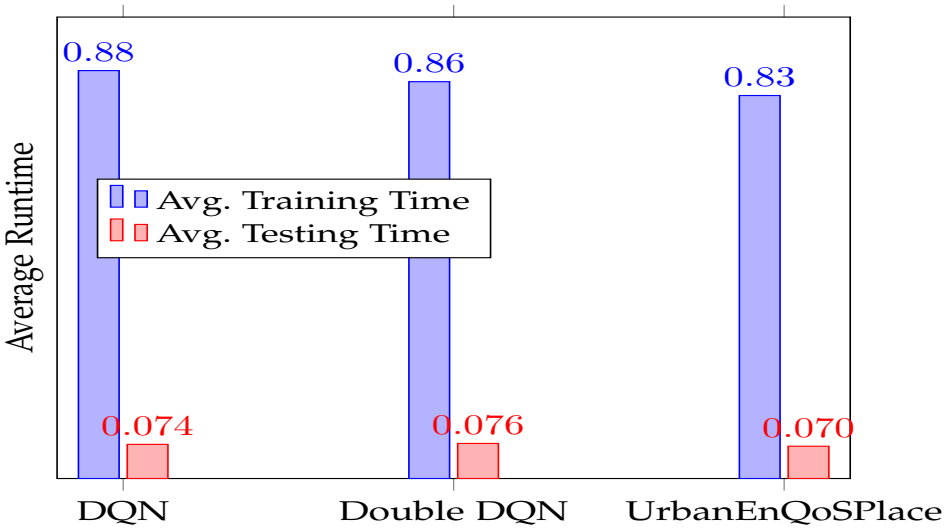


Figure 5.8: DRL Models Runtime Analysis

5.4.3 Scalability Analysis

In this set of experiments, performance of the proposed UrbanEnQoSPlace model is evaluated against VB algorithms at different scales of the placement problem.

Impact of Number of Applications (N)

In this section, the impact of no. of applications N on the obtained reward is analyzed by varying N from 10 to 50 applications, increased by 10 in each experiment. As a higher

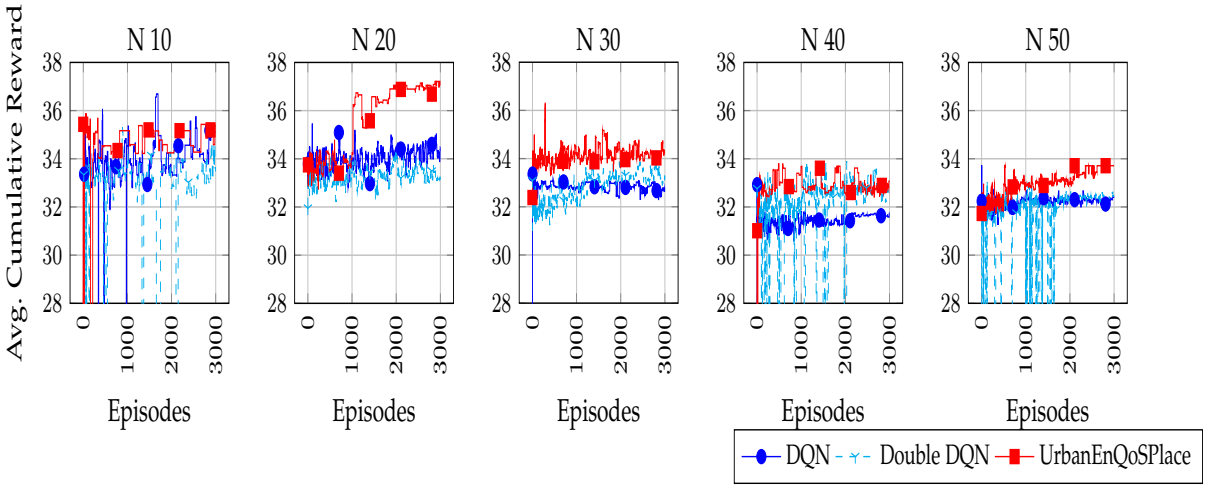


Figure 5.9: Scalability Analysis: Impact of Varying No. of Applications N (Fixed: M=5, K=60)

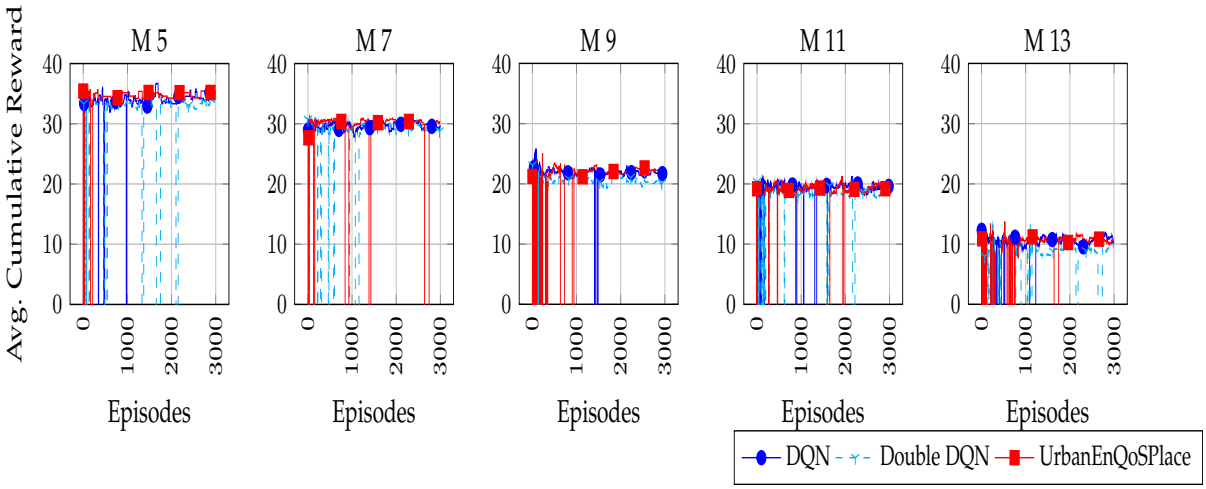


Figure 5.10: Scalability Analysis: Impact of Varying No. of Application Services M (Fixed: N=10, K=60)

no. of applications will require more servers and USTs, servers are set $K = 60$ and no. of USTs $\frac{3}{4}K$ with the remaining simulation settings unchanged. Figure 5.9 shows the reward for the UrbanEnQoSPlace model and the chosen VB algorithms across different numbers of applications. It is found that (i) ***UrbanEnQoSPlace*** consistently outperforms DQN and double DQN at all scales of application number; (ii) double DQN achieves higher rewards than DQN at larger scales – as it uses double estimates to overcome overestimation problem of DQN, however, it cannot perform better than UrbanEnQoSPlace; (iii) UrbanEnQoSPlace performs far better when the application number increases. This is due to the fact that with an increase in N, the no. of states in the underlying MDP (UrbanEnQoSMDP) also increases and the dueling approach for separate estimation of state-value

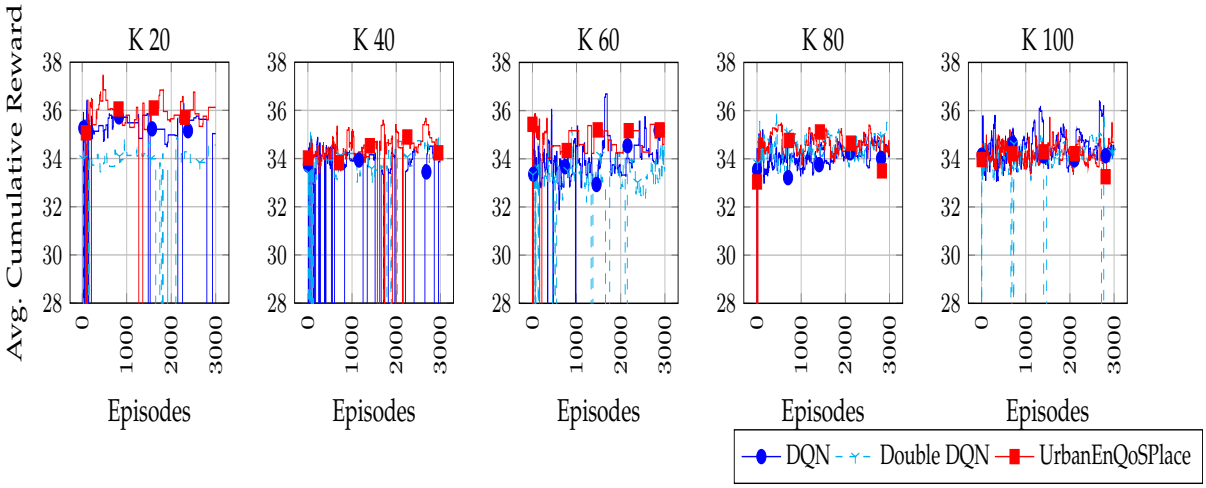


Figure 5.11: Scalability Analysis: Impact of Varying No. of Placement Servers K (Fixed: N=10, M=5)

function $V(s)$ and action-value function $A(s,a)$ in the UrbanEnQoSPlace results in better performance i.e., stable training and higher rewards than DQN and double DQN as both of these lack the state-value separation mechanism that could limit unnecessary exploration for lesser valuable states.

Impact of Number of Application Services (M)

In this section, the impact of the number of application services M on the achieved reward is analyzed by varying M from 5 to 13, increased by 2 in each experiment. The values $N = 10, K = 60$, no. of USTs $\frac{3}{4}K$ are set; with the remaining simulation settings unchanged. Figure 5.10 shows the rewards obtained for UrbanEnQoSPlace and VB algorithms across different scales of M. It is inferred that (i) ***UrbanEnQoSPlace*** outperforms both DQN and Double DQN at all scales of M; (ii) there is negligible difference in the performance of double DQN and DQN, however DQN exhibits less exploration and stable training; (iii) the converged value of the reward metric is gradually decreasing with increasing values of M for all the approaches. Specifically, the converged value of the reward metric dropped from 35 for $M = 5$ to 10 for $M = 13$. This is because with increasing M, the application becomes more complex i.e., resource intensive in terms of compute, communicate and IoT resources that leads to an increase in total latency and energy-consumption at the application-level by the large no. of application services. In the nutshell, an application with large no. of services will incur higher latency and energy consumption and lower rewards as reward is inversely related to the weighted sum of latency and energy consumption.

Impact of Number of Placement Servers (K)

In this section, the impact of varying no. of placement servers K from 20 to 100 is

analyzed, increased by 20 in each experiment. Figure 5.11 shows the reward for UrbanEnQoSPlace and the VB algorithms across different values of K. From the results, it is inferred that (i) ***UrbanEnQoSPlace*** outperforms both DQN and Double DQN across all scales of K; (ii) all the approaches experience faster convergence at higher scales of K. This is because with an increase in K the action-space for each of multiple actions also increases and all the DRL agents efficiently estimate the action-value function $Q(s, a)$ that leads to less exploration and quick convergence.

5.4.4 Policy Evaluation

In this section, the efficacy of the proposed "ϵ-greedy with mask policy" is evaluated against the widely used ϵ-greedy policy that works without any action-mask. In this experiment, each DRL model is run five times using these two policies. In each experiment, how many times the episode terminate with a failure on account of any of the constraints violation (C1 to C3 eq. 19) is counted out of 3000 training episodes and then, the average value over five such runs is computed to denote the average number of constraints violation. The results obtained for average number of constraints violation for each model by using each policy are shown in Figure 5.12. As expected, a *significant*

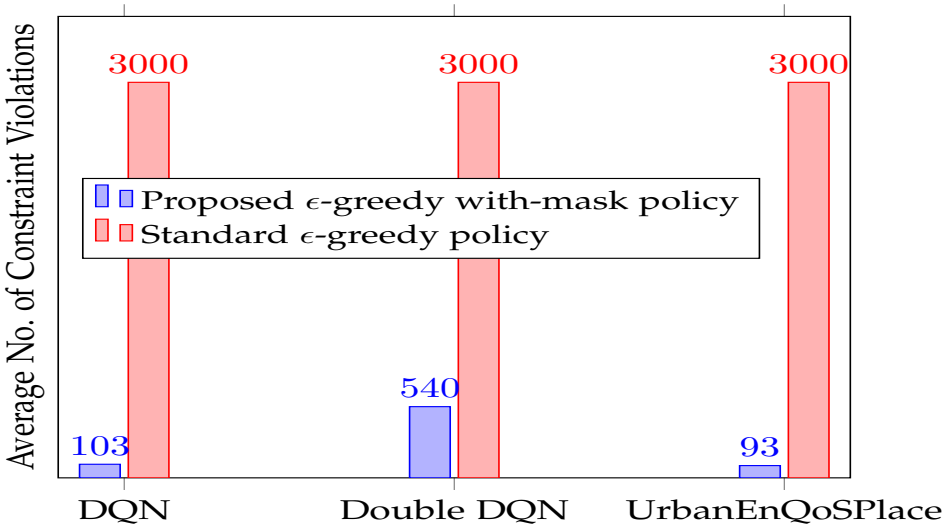


Figure 5.12: Policy Evaluation

lesser no. of constraint violations are observed for the proposed ϵ-greedy with mask policy as compared to the standard ϵ-greedy policy *for each DRL model*. This is due to fact that using the proposed policy the underlying DRL model encounters only two types of constraint violations – insufficient compute or communicate resources as the IoT resource satisfaction is guaranteed by the policy by means of the mask it uses, on the other hand for ϵ-greedy policy the DRL model encounters all three types of constraint violations –

insufficient compute, communicate and IoT resources.

Additionally, the policy-wise results for different DRL models indicate that i) using the standard ϵ-greedy policy, *none of the DRL models converge* to an optimal placement policy as they keep on encountering constraint violation in each of the 3000 training episodes; ii) using the proposed policy over different DRL models our proposed *UrbanEnQoSPlace DRL model outperforms both* DQN and Double DQN in terms of average number of constraint violations. Thus, *both, the proposed DRL model and the proposed policy are the most efficient* in achieving the optimal placement solution with highest reward incurring minimum number of constraint violations.

5.5 Conclusion

This chapter proposes UrbanEnQoSPlace, a novel DRL model that minimizes latency and energy consumption for optimal service placement of application services from multiple smart city IoT applications while satisfying their compute, communicate, and IoT resource requirements in an 'Urban IoT - Federated MEC - Cloud' architecture. The proposed DRL model runs on the MEC edge orchestrator to make optimal placement decisions with respect to both, latency and energy consumption for a set of smart city applications. The experiment results demonstrate the efficacy of proposed DRL model against state-of-the-art DRL models. In addition, a masking-based model policy ϵ-greedy with mask is proposed for the DRL model that ensures satifaction of IoT constraints and enables the model to converge faster as compared to the standard ϵ-greedy policy.

The next chapter concludes the entire thesis with directions for future work.

Chapter 6

Conclusions and Future Scope

Previous chapter addresses the enablement of efficient service delivery for real-time IoT applications. In this regard, a Deep RL model named as, UrbanEnQoSPlace is proposed along with a model policy named as, ϵ-greedy with mask, in order to place application services in the multi-tier placement architecture byb minimizing latency and energy consumption, while satisfying diverse resource requirements of a set of IoT applications in a smart city.

This chapter concludes the entire thesis and proposes future directions to extend this work. Section 6.1 concludes the research work presented in this thesis by highlighting the main contributions. Section 6.2 presents future directions for extending this research work.

6.1 Conclusions

The objective of this research work is to develop an IoT and cloud based framework for enabling a smart city. Smart city enablement involves two aspects: cloud-based urban big data management for analytical applications and edge-based service delivery for real-time IoT applications. The key findings of this research work are discussed below:

This thesis begins with an introduction on smart cities in **Chapter 1** that provides an overview of smart cities, its enabling technologies, application design, deployment architectures and challenges. It presents the problem statement, research motivation and objectives to pursue research in the area of smart cities. At last, the chapter discusses contributions of the research done and organization of the rest of this thesis.

Chapter 2 provides literature survey of various smart city enabling technologies. Firstly, big data management in IoT are analyzed across several IoT domains that classified prevalent big data management techniques in IoT and identified 13 V's challenges for big data in IoT. Secondly, big data management specific to smart city frameworks are analyzed that identified research gaps in existing smart city frameworks for urban big data management. Third, edge computing support for real-time applications are analyzed that identified research gaps for deploying real-time smart city applications in the IoT-edge-cloud multi-tier architecture.

In **Chapter 3**, an IoT and cloud based smart city framework named as, Cloud4IoTCity is proposed. The framework is designed to support the fusion of IoT big data with geospatial dataset(s) of the city to perform multi-resolution spatio-temporal analysis of IoT data. Cloud4IoTCity is designed as a cloud-based layered framework that comprises of four layers namely, storage layer, processing layer, service layer, and application layer. The storage layer stores the fused IoT and geospatial data in Delta Lake tables on the cloud object store (AWS S3). The processing layer uses a customized compute cluster for distributed big data and spatial processing. The service layer provides two types of services: i) data engineering services - for efficient data management and ii) data processing services - for spatio-temporal analysis (both, descriptive and predictive) and visualization of urban data. The application layer allows end users (smart city managers) to interact through the developed application to run various data management and analysis services.

Further, to enable descriptive analysis of smart city IoT data in the form of space-time queries, a taxonomy of fifteen spatio-temporal queries is proposed. The proposed taxonomy comprises of five spatial queries, four temporal queries and six spatio-temporal

queries. These queries enable query-based retrieval for descriptive analysis service in the proposed Cloud4IoTCity framework, which assist smart city managers in analyzing the urban attribute (measured as IoT readings) from space-time dimensions for smart governance purposes.

In **Chapter 4**, the proposed framework is validated through a case study on urban traffic analysis for Dublin city. The case study is implemented for a real IoT-based smart city dataset 'SCATS Traffic Volumes Dataset for Dublin', which contains hourly sensor readings for traffic volume at more than 500+ sites across the city. This dataset is generated by in-road inductive-loop sensors implemented under smart traffic management system. The spatial (longitude, latitude) information of sensors is also provided along with traffic readings. Further, geospatial datasets of Dublin city at three spatial resolutions: i) Dublin Postal districts ii) Dublin Administrative Areas and iii) Dublin city are obtained and integrated with the SCATS traffic dataset for implementing the framework's storage layer. Then, the compute cluster of processing layer is used to implement various data engineering services: add new data, optimize data layout, and time travel. Various data processing services: descriptive analysis using proposed taxonomy of spatio-temporal queries, predictive analysis for per-site hourly and daily traffic prediction, and map-based visualization for analysis services are implemented. Experimental results show that distributed processing of queries reduces the average query runtime by 15.25% on a cluster with 4 worker nodes and by 28.12% on a cluster with 8 worker nodes as compared to a cluster with 1 worker node. Experimental results show that employing the z-order data layout optimization on data reduces the average query runtime by 34.12% as compared to the non z-ordered layout of data.

In **Chapter 5**, second aspect of smart city enablement i.e., edge-based efficient service delivery for real-time applications in a smart city is presented. In this chapter, an AI-based DRL model named as, UrbanEnQoSPlace is proposed that solves the optimal service placement problem for a set of IoT applications in a smart city. The proposed DRL model optimally places application services among the three-tier 'Urban IoT - Federated MEC - Cloud' architecture, with an aim to optimize (minimize) both, latency and energy consumption for the set of IoT applications while satisfying IoT resource, compute resource, and per-flow latency, bandwidth requirements of the application services. The proposed DRL model is adapted from the Dueling Deep-Q Network to output multiple actions simultaneously, unlike the original Dueling DQN that outputs one action at a time.

Further, a novel policy named as, 'ϵ-greedy with mask' is proposed for use by the DRL model while solving the placement problem. The proposed policy adds a boolean mask on

the standard ϵ-greedy policy, such that the valid placement nodes (model actions) for a sensing/actuating service are set to true, while others are set to false. This policy satisfies the IoT constraints by masking invalid placement nodes, which enables UrbanEnQoSPlace to converge faster. The DRL model uses the proposed policy to ensure the satisfaction of IoT locality constraints, which leads to lesser constraints violations and faster model convergence. Experimental results show that the proposed DRL model achieves faster convergence than state-of-the-art DRL algorithms and achieves higher rewards. Further, experiments are carried out to assess the scalability of the proposed model by varying i) no. of applications ii) no. of application services and iii) no. of placement servers. Finally, experiments show that the proposed ϵ-greedy with mask policy results in 96.9% reduction in number of constraints violation as compared to the standard ϵ-greedy policy for solving the placement problem using UrbanEnQoSPlace model.

6.2 Future Directions

The contributions of this thesis leads to new research in enabling smart cities which can be extended by further research. Following future directions are suggested for the research community:

i) The proposed Cloud4IoTCity framework can be extended to develop new applications in the application layer. For instance, a mobile app for daily traffic prediction for citizens can be developed by using the predictive analysis services from the framework's service layer.

ii) The proposed Cloud4IoTCity framework is implemented and validated using the traffic data from Dublin city, which is available periodically as a batch dataset. The proposed framework can be extended to enable ingestion, analysis and overall management for real-time data as well.

iii) The proposed framework is implemented for IoT data from a single domain. The framework can be extended to acquire and manage data from multiple IoT domains.

iv) New AI-based approaches can be proposed to enhance the proposed service placement DRL model.

References

[1] Jiong Jin, Jayavardhana Gubbi, Slaven Marusic, and Marimuthu Palaniswami. An information framework for creating a smart city through internet of things. *IEEE Internet of Things journal*, 1(2):112–121, 2014.

[2] Taewoo Nam and Theresa A Pardo. Conceptualizing smart city with dimensions of technology, people, and institutions. In *Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times*, pages 282–291, 2011.

[3] Tuba Bakıcı, Esteve Almirall, and Jonathan Wareham. A smart city initiative: the case of barcelona. *Journal of the knowledge economy*, 4:135–148, 2013.

[4] Diogo Correia, João Lourenço Marques, and Leonor Teixeira. The state-of-the-art of smart cities in the european union. *Smart Cities*, 5(4):1776–1810, 2022.

[5] Luis Sanchez, Luis Muñoz, Jose Antonio Galache, Pablo Sotres, Juan R Santana, Veronica Gutierrez, Rajiv Ramdhany, Alex Gluhak, Srdjan Krco, Evangelos Theodoridis, et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, 61:217–238, 2014.

[6] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE Internet of Things journal*, 1(1):22–32, 2014.

[7] Pu Liu and Zhenghong Peng. Smart cities in china. *IEEE Computer Society*, 16(7), 2013.

[8] Jung-Hoon Lee and Marguerite Gong Hancock. Toward a framework for smart cities: A comparison of seoul, san francisco and amsterdam. *Research Paper, Yonsei University and Stanford University*, 2012.

[9] Government of India. Smart cities. `https://smartcities.gov.in/`. Accessed on Oct 05, 2023.

[10] IBM. Smarter cities. `https://www.ibm.com/smarterplanet/us/en/smarter_cities/solutions/infrastructure_solutions/`. Accessed on Oct 05, 2023.

[11] Cisco. Cities and communities. `https://www.cisco.com/c/en/us/solutions/industries/smart-connected-communities.html`. Accessed on Oct 05, 2023.

[12] Microsoft. Microsoft for government. `https://www.microsoft.com/en-us/industry/government`. Accessed on Oct 05, 2023.

[13] Rudolf Giffinger, Christian Fertner, Hans Kramar, Robert Kalasek, Natasa Pichler-Milanovic, and Evert J Meijers. Smart cities. ranking of european medium-sized

cities. final report. 2007.

[14] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[15] 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.

[16] Peter Mell and Timothy Grance. The nist definition of cloud computing. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`, 2011. Accessed on Oct 05, 2023.

[17] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98:289–330, 2019.

[18] Charith Perera, Yongrui Qin, Julio C Estrella, Stephan Reiff-Marganiec, and Athanasios V Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys (CSUR)*, 50(3):1–43, 2017.

[19] Douglas Laney. 3D data management: Controlling data volume, velocity, and variety. Technical report, META Group, February 2001.

[20] Eiman Al Nuaimi, Hind Al Neyadi, Nader Mohamed, and Jameela Al-Jaroodi. Applications of big data to smart cities. *Journal of Internet Services and Applications*, 6:1–15, 2015.

[21] Ibrahim Abaker Targio Hashem, Victor Chang, Nor Badrul Anuar, Kayode Adewole, Ibrar Yaqoob, Abdullah Gani, Ejaz Ahmed, and Haruna Chiroma. The role of big data in smart city. *International Journal of information management*, 36(5):748–758, 2016.

[22] Kamran Soomro, Muhammad Nasir Mumtaz Bhutta, Zaheer Khan, and Muhammad A Tahir. Smart city big data analytics: An advanced review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(5):e1319, 2019.

[23] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, and et al. The rise of "big data" on cloud computing: Review and open research issues. *Information systems*, 47:98–115, 2015.

[24] Ammar Gharaibeh, Mohammad A Salahuddin, Sayed Jahed Hussini, Abdallah Khreishah, Issa Khalil, Mohsen Guizani, and Ala Al-Fuqaha. Smart cities: A survey on data management, security, and enabling technologies. *IEEE Communications Surveys & Tutorials*, 19(4):2456–2501, 2017.

[25] Gavin McArdle and Rob Kitchin. The dublin dashboard: Design and development of a real-time analytical urban dashboard. *ISPRS Annals of the Photogrammetry,*

Remote Sensing and Spatial Information Sciences, 4:19–25, 2016.

[26] London. Citydashboard london. `http:citydashboard.org/london`. Accessed on March 05, 2024.

[27] Gareth W Young, Rob Kitchin, and Jeneen Naji. Building city dashboards for different types of users. In *Sustainable smart city transitions*, pages 259–279. Routledge, 2022.

[28] Leonidas G Anthopoulos and Athena Vakali. Urban planning and smart cities: Interrelations and reciprocities. In *The Future Internet: Future Internet Assembly 2012: From Promises to Reality 9*, pages 178–189. Springer Berlin Heidelberg, 2012.

[29] M Mazhar Rathore, Awais Ahmad, Anand Paul, and Seungmin Rho. Urban planning and building smart cities based on the internet of things using big data analytics. *Computer networks*, 101:63–80, 2016.

[30] Muhammad Babar and Fahim Arif. Smart urban planning using big data analytics based internet of things. In *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, pages 397–402, 2017.

[31] Nikolaos Komninos, C Kakderi, Anastasia Panori, and P Tsarchopoulos. Smart city planning from an evolutionary perspective. *Journal of Urban Technology*, 26(2):3–20, 2019.

[32] Paul Melnyk, Soufiene Djahel, and Farid Nait-Abdesselam. Towards a smart parking management system for smart cities. In *2019 IEEE international smart cities conference (ISC2)*, pages 542–546. IEEE, 2019.

[33] Yunchuan Sun, Houbing Song, Antonio J Jara, and Rongfang Bie. Internet of things and big data analytics for smart and connected communities. *IEEE access*, 4:766–773, 2016.

[34] Alfonso Garcia-de Prado, GO Ortiz, Juan Boubeta-Puig, and David Corral-Plaza. Air4people: a smart air quality monitoring and context-aware notification system. *Journal of Universal Computer Science*, 24(7):846–863, 2018.

[35] Hadi Habibzadeh, Cem Kaptan, Tolga Soyata, Burak Kantarci, and Azzedine Boukerche. Smart city system design: A comprehensive study of the application and data planes. *ACM Computing Surveys (CSUR)*, 52(2):1–38, 2019.

[36] Farah Ait Salaht, Frédéric Desprez, and Adrien Lebre. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, 53(3):1–35, 2020.

[37] Eduardo Felipe Zambom Santana, Ana Paula Chaves, Marco Aurelio Gerosa, Fabio Kon, and Dejan S Milojicic. Software platforms for smart cities: Concepts, requirements, challenges, and a unified reference architecture. *ACM Computing Surveys*

(Csur), 50(6):1–37, 2017.

[38] Rida Khatoun and Sherali Zeadally. Smart cities: concepts, architectures, research opportunities. *Communications of the ACM*, 59(8):46–57, 2016.

[39] Bhagya Nathali Silva, Murad Khan, Changsu Jung, Jihun Seo, Diyan Muhammad, Jihun Han, Yongtak Yoon, and Kijun Han. Urban planning and smart city decision management empowered by real-time data processing using big data analytics. *Sensors*, 18(9):2994, 2018.

[40] Andreas Kamilaris and Frank O Ostermann. Geospatial analysis and the internet of things. *ISPRS international journal of geo-information*, 7(7):269, 2018.

[41] Poonam Sharma, Rashmi Singh, and Ankur Srivastava. Analyzing the role of geospatial technology in smart city development. *Geospatial Technology and Smart Cities: ICT, Geoscience Modeling, GIS and Remote Sensing*, pages 1–20, 2021.

[42] Daniel G Costa, João Carlos N Bittencourt, Franklin Oliveira, João Paulo Just Peixoto, and Thiago C Jesus. Achieving sustainable smart cities through geospatial data-driven approaches. *Sustainability*, 16(2):640, 2024.

[43] Muhammed Oğuzhan Mete. Geospatial big data analytics for sustainable smart cities. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 48:141–146, 2023.

[44] Safa Ben Atitallah, Maha Driss, Wadii Boulila, and Henda Ben Ghézala. Leveraging deep learning and iot big data analytics to support the smart cities development: Review and future directions. *Computer Science Review*, 38:100303, 2020.

[45] Md Whaiduzzaman, Alistair Barros, Moumita Chanda, Supti Barman, Tania Sultana, Md Sazzadur Rahman, Shanto Roy, and Colin Fidge. A review of emerging technologies for iot-based smart cities. *Sensors*, 22(23):9271, 2022.

[46] Michael Chui, Vasanth Ganesan, and Mark Patel. Taking the pulse of enterprise iot. Technical report, McKinsey & Company, 2017.

[47] Cisco News Release. Cisco survey reveals close to three-fourths of iot projects are failing, may 2017.

[48] Emir Husni, Galuh Boy Hertantyo, Daniel Wahyu Wicaksono, and et al. Applied internet of things (iot): car monitoring system using ibm bluemix. In *2016 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 417–422. IEEE, 2016.

[49] William Tärneberg, Vishal Chandrasekaran, and Marty Humphrey. Experiences creating a framework for smart traffic control using aws iot. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 63–69. IEEE, 2016.

[50] Yanxu Zheng, Sutharshan Rajasegarar, and Christopher Leckie. Parking availabil-

ity prediction for sensor-enabled car parks in smart cities. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6. IEEE, 2015.

[51] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang. Traffic flow prediction with big data: a deep learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):865–873, 2014.

[52] Jiafu Wan, Jianqi Liu, Zehui Shao, Athanasios V Vasilakos, and et al. Mobile crowd sensing for traffic prediction in internet of vehicles. *Sensors*, 16(1):88, 2016.

[53] M Mazhar Rathore, Anand Paul, Won-Hwa Hong, HyunCheol Seo, Imtiaz Awan, and Sharjil Saeed. Exploiting iot and big data analytics: Defining smart digital city using real-time urban data. *Sustainable cities and society*, 40:600–610, 2018.

[54] Muhammad Usman, Mian Ahmad Jan, Xiangjian He, and Jinjun Chen. A survey on big multimedia data processing and management in smart cities. *ACM Computing Surveys (CSUR)*, 52(3):1–29, 2019.

[55] Nazli Farajidavar, Sefki Kolozali, and Payam Barnaghi. A deep multi-view learning framework for city event extraction from twitter data streams. 2017.

[56] Ivana Podnar Žarko, Krešimir Pripužić, Martin Serrano, and Manfred Hauswirth. Iot data management methods and optimisation algorithms for mobile publish/subscribe services in cloud environments. In *2014 European conference on networks and communications (EuCNC)*, pages 1–5. IEEE, 2014.

[57] Hugo Hromic, Danh Le Phuoc, Martin Serrano, Aleksandar Antonić, and et al. Real time analysis of sensor data for the internet of things by means of clustering and event processing. In *2015 IEEE International conference on communications (ICC)*, pages 685–691. IEEE, 2015.

[58] Giuseppe Piro, Ilaria Cianci, Luigi Alfredo Grieco, Gennaro Boggia, and Pietro Camarda. Information centric services in smart cities. *Journal of Systems and Software*, 88:169–188, 2014.

[59] Maggi Bansal, Inderveer Chana, and Siobhán Clarke. Enablement of iot based context-aware smart home with fog computing. *Journal of Cases on Information Technology (JCIT)*, 19(4):1–12, 2017.

[60] Mahdi Kasmi, Faouzi Bahloul, and Haykel Tkitek. Smart home based on internet of things and cloud computing. In *2016 7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pages 82–86. IEEE, 2016.

[61] Luca Mainetti, Luigi Patrono, and Antonio Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *SoftCOM 2011, 19th international conference on software, telecommunications and computer networks*, pages

1–6. IEEE, 2011.

[62] Andreas P Plageras, Kostas E Psannis, Christos Stergiou, Haoxiang Wang, and Brij B Gupta. Efficient iot-based sensor big data collection–processing and analysis in smart buildings. *Future Generation Computer Systems*, 82:349–357, 2018.

[63] Abdulsalam Yassine, Shailendra Singh, M Shamim Hossain, and Ghulam Muhammad. Iot big data analytics for smart homes with fog and cloud computing. *Future Generation Computer Systems*, 91:563–573, 2019.

[64] Wassim Derguech, Eanna Bruke, and Edward Curry. An autonomic approach to real-time predictive analytics using open data and internet of things. In *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*, pages 204–211. IEEE, 2014.

[65] Young Jin Jung, Yang Koo Lee, Dong Gyu Lee, Yongmi Lee, and et al. Design of sensor data processing steps in an air pollution monitoring system. *Sensors*, 11(12):11235–11250, 2011.

[66] Sergio Trilles, Òscar Belmonte, Sven Schade, and Joaquìn Huerta. A domain-independent methodology to analyze iot data streams in real-time. a proof of concept implementation for anomaly detection from environmental data. *International Journal of Digital Earth*, 10(1):103–120, 2017.

[67] Rubén Mulero, Aitor Almeida, Gorka Azkune, Patricia Abril-Jiménez, and et al. An iot-aware approach for elderly-friendly cities. *IEEE Access*, 6:7941–7957, 2018.

[68] Diana C Yacchirema, David Sarabia-Jácome, Carlos E Palau, and Manuel Esteve. A smart system for sleep monitoring by integrating iot with big data analytics. *IEEE Access*, 6:35988–36001, 2018.

[69] Sandeep Singh Sandha, Mohammad Kachuee, and Sajad Darabi. Complex event processing of health data in real-time to predict heart failure risk and stress. 2017.

[70] Mohammed Al-Khafajiy, Thar Baker, Carl Chalmers, Muhammad Asim, Hoshang Kolivand, Muhammad Fahim, and Atif Waraich. Remote health monitoring of elderly through wearable sensors. *Multimedia Tools and Applications*, 78(17):24681–24706, 2019.

[71] NAM Alduais, J Abdullah, A Jamil, and L Audah. An efficient data collection and dissemination for iot based wsn. In *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 1–6. IEEE, 2016.

[72] Ngoc-Thanh Dinh and Younghan Kim. An energy efficient integration model for sensor cloud systems. *IEEE Access*, 7:3018–3030, 2018.

[73] Andrea Capponi, Claudio Fiandrino, Dzmitry Kliazovich, and Pascal Bouvry. Energy efficient data collection in opportunistic mobile crowdsensing architectures for smart cities. In *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 307–312. IEEE, 2017.

[74] Yogesh Simmhan, Srijith Nair, Sumit Monga, Ravi Sahu, Kuldeep Dixit, Ronak Sutaria, Brijesh Mishra, Anamika Sharma, SVR Anand, Malati Hegde, et al. Satvam: Toward an iot cyber-infrastructure for low-cost urban air quality monitoring. In *2019 15th International Conference on eScience (eScience)*, pages 57–66. IEEE, 2019.

[75] Lihong Jiang, Li Da Xu, Hongming Cai, Zuhai Jiang, and et al. An iot-oriented data storage framework in cloud computing platform. *IEEE Transactions on Industrial Informatics*, 10(2):1443–1451, 2014.

[76] Tingli Li, Yang Liu, Ye Tian, Shuo Shen, and Wei Mao. A storage solution for massive iot data based on nosql. In *2012 IEEE International Conference on Green Computing and Communications*, pages 50–57. IEEE, 2012.

[77] Juan Luis Pérez and David Carrera. Performance characterization of the servioticy api: an iot-as-a-service data management platform. In *2015 IEEE First International Conference on Big Data Computing Service and Applications*, pages 62–71. IEEE, 2015.

[78] Shanshan Wu, Liang Bao, Zisheng Zhu, Fan Yi, and Weizhao Chen. Storage and retrieval of massive heterogeneous iot data based on hybrid storage. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 2982–2987. IEEE, 2017.

[79] Yogendra Narain Singh and Sanjay Kumar Singh. Identifying individuals using eigenbeat features of electrocardiogram. *Journal of Engineering*, 2013(1):539284, 2013.

[80] Xiaomin Xu, Sheng Huang, Yaoliang Chen, Kevin Browny, and et al. Tsaaas: Time series analytics as a service on iot. In *2014 IEEE International Conference on Web Services*, pages 249–256. IEEE, 2014.

[81] Chi Yang, Deepak Puthal, Saraju P Mohanty, and Elias Kougianos. Big-sensing-data curation for the cloud is coming: A promise of scalable cloud-data-center mitigation for next-generation iot and wireless sensor networks. *IEEE Consumer Electronics Magazine*, 6(4):48–56, 2017.

[82] Limei Peng, Ahmad R Dhaini, and Pin-Han Ho. Toward integrated cloud–fog networks for efficient iot provisioning: Key challenges and solutions. *Future Generation Computer Systems*, 88:606–613, 2018.

[83] Shree Krishna Sharma and Xianbin Wang. Live data analytics with collaborative

edge and cloud processing in wireless iot networks. *IEEE Access*, 5:4621–4635, 2017.

[84] Jie Tang, Dawei Sun, Shaoshan Liu, and Jean-Luc Gaudiot. Enabling deep learning on iot devices. *Computer*, 50(10):92–96, 2017.

[85] Yoji Yamato, Hiroki Kumazaki, and Yoshifumi Fukumoto. Proposal of lambda architecture adoption for real time predictive maintenance. In *2016 fourth international symposium on computing and networking (CANDAR)*, pages 713–715. IEEE, 2016.

[86] DMC Dissanayake and KPN Jayasena. A cloud platform for big iot data analytics by combining batch and stream processing technologies. In *2017 National Information Technology Conference (NITC)*, pages 40–45. IEEE, 2017.

[87] Manuel Díaz, Cristian Martín, and Bartolomé Rubio. Lambda-coap: an internet of things and cloud computing integration based on the lambda architecture and coap. In *International conference on collaborative computing: Networking, applications and worksharing*, pages 195–206. Springer, 2015.

[88] Rahat Iqbal, Faiyaz Doctor, Brian More, Shahid Mahmud, and Usman Yousuf. Big data analytics: Computational intelligence techniques and application areas. *Technological Forecasting and Social Change*, 153:119253, 2020.

[89] Susan Gauch, Jianying Wang, and Satya Mahesh Rachakonda. A corpus analysis approach for automatic query expansion and its extension to multiple databases. *ACM Transactions on Information Systems (TOIS)*, 17(3):250–269, 1999.

[90] Altti Ilari Maarala, Xiang Su, and Jukka Riekki. Semantic reasoning for context-aware internet of things applications. *IEEE Internet of Things Journal*, 4(2):461–473, 2016.

[91] Chao-Hsien Lee, Zheng-Lin Wu, Yun-Ting Chiu, and Vi-Shu Chen. Heterogeneous industrial iot integration for manufacturing production. In *2019 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS)*, pages 1–2. IEEE, 2019.

[92] Abdulkadir Karaagac, Niels Verbeeck, and Jeroen Hoebeke. The integration of lwm2m and opc ua: an interoperability approach for industrial iot. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 313–318. IEEE, 2019.

[93] Hongming Cai, Yizhi Gu, Athanasios V Vasilakos, Boyi Xu, and Jun Zhou. Model-driven development patterns for mobile services in cloud of things. *IEEE Transactions on Cloud Computing*, 6(3):771–784, 2016.

[94] Shilpa Chaturvedi Anshu Shukla and Yogesh Simmhan. Riotbench: An iot benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, 29(21):e4257, 2017.

[95] Xing Chen, Aipeng Li, Wenzhong Guo, and et al. Runtime model based approach

to iot application development. *Frontiers of Computer Science*, 9(4):540–553, 2015.

[96] Jiafu Wan, Daqiang Zhang, Shengjie Zhao, Laurence T Yang, and Jaime Lloret. Context-aware vehicular cyber-physical systems with cloud support: architecture, challenges, and solutions. *IEEE Communications Magazine*, 52(8):106–113, 2014.

[97] Sukhpal Singh Gill, Minxian Xu, Carlo Ottaviani, Panos Patros, Rami Bahsoon, Arash Shaghaghi, Muhammed Golec, Vlado Stankovski, Huaming Wu, Ajith Abraham, et al. Ai for next generation computing: Emerging trends and future directions. *Internet of Things*, 19:100514, 2022.

[98] Ishana Attri, Lalit Kumar Awasthi, Teek Parval Sharma, and Priyanka Rathee. A review of deep learning techniques used in agriculture. *Ecological Informatics*, page 102217, 2023.

[99] Ishana Attri, Lalit Kumar Awasthi, and Teek Parval Sharma. Machine learning in agriculture: a review of crop management applications. *Multimedia Tools and Applications*, 83(5):12875–12915, 2024.

[100] Google. Google cloud iot.

[101] Android. Android things.

[102] Google. Bringing intelligence to the edge with cloud iot.

[103] Google. Edge tpu.

[104] AWS. Aws iot core.

[105] AWS. Aws iot button.

[106] AWS. Aws iot greengrass.

[107] Microsoft Azure. Azure iot solution accelerators.

[108] Microsoft Azure. Azure iot central.

[109] Microsoft Azure. Azure iot edge.

[110] Microsoft. Windows for internet of things.

[111] IBM. Watson internet of things.

[112] Geymerson S Ramos, Danilo Fernandes, Jorge Artur P de M Coelho, and Andre LL Aquino. Toward data lake technologies for intelligent societies and cities. In *Sustainable, Innovative and Intelligent Societies and Cities*, pages 3–29. Springer, 2023.

[113] Mourad Raif, Abdellah Chehri, Rachid Saadane, et al. Data architecture and big data analytics in smart cities. *Procedia Computer Science*, 207:4123–4131, 2022.

[114] Murilo Borges Ribeiro and Kelly Rosa Braghetto. A data integration architecture for smart cities. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados*, pages 205–216. SBC, 2021.

[115] Leandro Ordonez-Ante, Gregory Van Seghbroeck, Tim Wauters, Bruno Volckaert, and Filip De Turck. Explora: Interactive querying of multidimensional data in the

context of smart cities. *Sensors*, 20(9):2737, 2020.

[116] João Paulo Clarindo dos Santos, João Pedro de Carvalho Castro, and Cristina Dutra de Aguiar Ciferri. Solap query processing over iot networks in smart cities: A novel architecture. In *Proceedings XXI GEOINFO*, pages 118–129, 2020.

[117] Ruiyuan Li, Huajun He, Rubin Wang, Yuchuan Huang, Junwen Liu, Sijie Ruan, Tianfu He, Jie Bao, and Yu Zheng. Just: Jd urban spatio-temporal data engine. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1558–1569. IEEE, 2020.

[118] Andy S Alic, Jussara Almeida, Giovanni Aloisio, Nazareno Andrade, Nuno Antunes, Danilo Ardagna, Rosa M Badia, Tania Basso, Ignacio Blanquer, Tarciso Braz, et al. Bigsea: A big data analytics platform for public transportation information. *Future generation computer systems*, 96:243–269, 2019.

[119] Obinna CD Anejionu, Piyushimita Vonu Thakuriah, Andrew McHugh, Yeran Sun, David McArthur, Phil Mason, and Rod Walpole. Spatial urban data system: A cloud-enabled big data infrastructure for social and economic urban analytics. *Future generation computer systems*, 98:456–473, 2019.

[120] Ahmed M Shahat Osman. A novel big data analytics framework for smart cities. *Future Generation Computer Systems*, 91:620–633, 2019.

[121] Zhihan Lv, Xiaoming Li, Weixi Wang, Baoyun Zhang, Jinxing Hu, and Shengzhong Feng. Government affairs service platform for smart city. *Future Generation Computer Systems*, 81:443–451, 2018.

[122] Pierfrancesco Bellini, Paolo Nesi, Michela Paolucci, and Imad Zaza. Smart city architecture for data ingestion and analytics: Processes and solutions. In *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 137–144. IEEE, 2018.

[123] Catalin Negru, Florin Pop, Mariana Mocanu, and Valentin Cristea. Storage solution of spatial-temporal data for water monitoring infrastructures used in smart cities. In *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, pages 617–621. IEEE, 2017.

[124] Wei Chen, Zhaosong Huang, Feiran Wu, Minfeng Zhu, Huihua Guan, and Ross Maciejewski. Vaud: A visual analysis approach for exploring spatio-temporal urban data. *IEEE transactions on visualization and computer graphics*, 24(9):2636–2648, 2017.

[125] Md Mahbub Alam, Luis Torgo, and Albert Bifet. A survey on spatio-temporal data analytics systems. *ACM Computing Surveys*, 54(10s):1–38, 2022.

[126] Federica Paganelli, Stefano Turchi, and Dino Giuli. A web of things framework for restful applications and its experimentation in a smart city. *IEEE Systems Journal*,

10(4):1412–1423, 2014.

[127] Stefano Turchi, Federica Paganelli, Lorenzo Bianchi, and Dino Giuli. A lightweight linked data implementation for modeling the web of things. In *2014 IEEE International Conference on Pervasive Computing and Communication Workshops (PERCOM WORKSHOPS)*, pages 123–128. IEEE, 2014.

[128] Sefki Kolozali, Maria Bermudez-Edo, Daniel Puschmann, Frieder Ganz, and Payam Barnaghi. A knowledge-based approach for real-time iot data stream annotation and processing. In *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 215–222. IEEE, 2014.

[129] Athira Nambiar and Divyansh Mundra. An overview of data warehouse and data lake in modern enterprise data management. *Big Data and Cognitive Computing*, 6(4):132, 2022.

[130] Michael Armbrust, Ali Ghodsi, Reynold Xin, and Matei Zaharia. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*, volume 8, 2021.

[131] Amin Ullah, Syed Myhammad Anwar, Jianqiang Li, Lubna Nadeem, Tariq Mahmood, Amjad Rehman, and Tanzila Saba. Smart cities: The role of internet of things and machine learning in realizing a data-centric smart environment. *Complex & Intelligent Systems*, 10(1):1607–1637, 2024.

[132] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 4(3):161–175, 2018.

[133] Qi Chen, Wei Wang, Fangyu Wu, Suparna De, Ruili Wang, Bailing Zhang, and Xin Huang. A survey on an emerging area: Deep learning for smart city data. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 3(5):392–410, 2019.

[134] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):2923–2960, 2018.

[135] Qingchen Zhang, Laurence T Yang, Zhikui Chen, and Peng Li. A survey on deep learning for big data. *Information Fusion*, 42:146–157, 2018.

[136] Tanweer Alam. Cloud-based iot applications and their roles in smart cities. *Smart Cities*, 4(3):1196–1219, 2021.

[137] Amanjot Kaur and Nitin Auluck. Real-time trust aware scheduling in fog-cloud

systems. *Concurrency and Computation: Practice and Experience*, 35(10):e7680, 2023.

[138] Pawani Porambage, Jude Okwuibe, Madhusanka Liyanage, Mika Ylianttila, and Tarik Taleb. Survey on multi-access edge computing for internet of things realization. *IEEE Communications Surveys & Tutorials*, 20(4):2961–2991, 2018.

[139] Amanjot Kaur, Nitin Auluck, and Omer Rana. Real-time scheduling on hierarchical heterogeneous fog networks. *IEEE Transactions on Services Computing*, 16(2):1358–1372, 2022.

[140] Sergey Ivanov and Alexander D'yakonov. Modern deep reinforcement learning algorithms. *arXiv preprint arXiv:1906.10025*, 2019.

[141] Xiaoheng Deng, Zihui Sun, Deng Li, Jie Luo, and Shaohua Wan. User-centric computation offloading for edge computing. *IEEE Internet of Things Journal*, 8(16):12559–12568, 2021.

[142] Shuyue Ma, Shudian Song, Lingyu Yang, Jingmei Zhao, Feng Yang, and Linbo Zhai. Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing. *Applied Soft Computing*, 112:107790, 2021.

[143] Kai Peng, Maosheng Zhu, Yiwen Zhang, Lingxia Liu, Jie Zhang, Victor Leung, and Lixin Zheng. An energy-and cost-aware computation offloading method for workflow applications in mobile edge computing. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–15, 2019.

[144] Xiaolong Xu, Qihe Huang, Xiaochun Yin, Mahdi Abbasi, Mohammad Reza Khosravi, and Lianyong Qi. Intelligent offloading for collaborative smart city services in edge computing. *IEEE Internet of Things Journal*, 7(9):7919–7927, 2020.

[145] Xiaolong Xu, Xihua Liu, Zhanyang Xu, Fei Dai, Xuyun Zhang, and Lianyong Qi. Trust-oriented iot service placement for smart cities in edge computing. *IEEE Internet of Things Journal*, 7(5):4084–4091, 2019.

[146] Kalka Dubey, SC Sharma, and Mohit Kumar. A secure iot applications allocation framework for integrated fog-cloud environment. *Journal of Grid Computing*, 20(1):1–23, 2022.

[147] Christian Humberto Cabrera Jojoa, Sergej Svorobej, Andrei Palade, Aqeel Kazmi, and Siobhan Clarke. Maaco: A dynamic service placement model for smart cities. *IEEE Transactions on Services Computing*, 2022.

[148] Xiong Xiong, Kan Zheng, Lei Lei, and Lu Hou. Resource allocation based on deep reinforcement learning in iot edge computing. *IEEE Journal on Selected Areas in Communications*, 38(6):1133–1146, 2020.

[149] Laha Ale, Ning Zhang, Xiaojie Fang, Xianfu Chen, Shaohua Wu, and Longzhuang Li. Delay-aware and energy-efficient computation offloading in mobile-edge com-

puting using deep reinforcement learning. *IEEE Transactions on Cognitive Communications and Networking*, 7(3):881–892, 2021.

[150] Pegah Gazori, Dadmehr Rahbari, and Mohsen Nickray. Saving time and cost on the scheduling of fog-based iot applications using deep reinforcement learning approach. *Future Generation Computer Systems*, 110:1098–1115, 2020.

[151] Zhiqing Tang, Jiong Lou, Fuming Zhang, and Weijia Jia. Dependent task offloading for multiple jobs in edge computing. In *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9. IEEE, 2020.

[152] Haifeng Lu, Chunhua Gu, Fei Luo, Weichao Ding, and Xinping Liu. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Generation Computer Systems*, 102:847–861, 2020.

[153] Saroj Kumar Panda, Man Lin, and Ti Zhou. Energy efficient computation offloading with dvfs using deep reinforcement learning for time-critical iot applications in edge computing. *IEEE Internet of Things Journal*, 2022.

[154] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.

[155] Yifei Wei, Zhaoying Wang, Da Guo, and F Richard Yu. Deep q-learning based computation offloading strategy for mobile edge computing. *Computers, Materials and Continua*, 59(1):89–104, 2019.

[156] Mohammad Goudarzi, Marimuthu S Palaniswami, and Rajkumar Buyya. A distributed deep reinforcement learning technique for application placement in edge and fog computing environments. *IEEE Transactions on Mobile Computing*, 2021.

[157] Jin Wang, Jia Hu, Geyong Min, Wenhan Zhan, Qiang Ni, and Nektarios Georgalas. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Communications Magazine*, 57(5):64–69, 2019.

[158] Tao Zheng, Jian Wan, Jilin Zhang, and Congfeng Jiang. Deep reinforcement learning-based workload scheduling for edge computing. *Journal of Cloud Computing*, 11(1):1–13, 2022.

[159] Wenhan Zhan, Chunbo Luo, Jin Wang, Geyong Min, and Hancong Duan. Deep reinforcement learning-based computation offloading in vehicular edge computing. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[160] Abdeladim Sadiki, Jamal Bentahar, Rachida Dssouli, and Abdeslam En-Nouaary. Deep reinforcement learning for the computation offloading in mimo-based edge

computing. 2021.

[161] Yu Liu, Qimei Cui, Jian Zhang, Yu Chen, and Yanzhao Hou. An actor-critic deep reinforcement learning based computation offloading for three-tier mobile computing networks. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2019.

[162] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. Deploying fog applications: How much does it cost by the way? *small*, 1(2):20, 2018.

[163] Jienan Chen, Siyu Chen, Qi Wang, Bin Cao, Gang Feng, and Jianhao Hu. iraf: A deep reinforcement learning approach for collaborative mobile edge computing iot networks. *IEEE Internet of Things Journal*, 6(4):7011–7024, 2019.

[164] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, et al. Delta lake: High-performance acid table storage over cloud object stores. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, 2020.

[165] Apache Spark. Cluster mode overview. `https://spark.apache.org/docs/latest/cluster-overview.html`, 2023. Accessed on Oct 05, 2023.

[166] Apache Spark. Unified engine for large-scale data analytics. `https://spark.apache.org/`, 2023. Accessed on Oct 05, 2023.

[167] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, pages 1–4, 2015.

[168] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial data management in apache spark: the geospark perspective and beyond. *GeoInformatica*, 23:37–78, 2019.

[169] Apache Sedona. Apache sedona. `https://sedona.apache.org/1.3.1-incubating/`, 2023. Accessed on Oct 05, 2023.

[170] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. How good are modern spatial analytics systems? *Proceedings of the VLDB Endowment*, 11(11):1661–1673, 2018.

[171] Plotly. Plotly express in python. `https://plotly.com/python/plotly-express/`, 2023. Accessed on Oct 05, 2023.

[172] May Yuan and John McIntosh. A typology of spatiotemporal information queries. *Mining Spatio-Temporal Information Systems*, pages 63–81, 2002.

[173] data.gov.ie. Datasets. `https://data.gov.ie/dataset?q=SCATS+Traffic`, 2023. Accessed on Oct 05, 2023.

[174] Smart Dublin. Dublinked open data store. `https://data.smartdublin.ie/dataset/?category=Transport+and+Infrastructure&organization=`

dublin-city-council&tags=SCATS, 2023. Accessed on Oct 05, 2023.

[175] Smart Dublin. Traffic signals and scats sites locations dcc. https://data.smartdublin.ie/dataset/traffic-signals-and-scats-sites-locations-dcc, 2023. Accessed on Oct 05, 2023.

[176] Wikipedia. List of dublin postal districts. https://en.wikipedia.org/wiki/List_of_Dublin_postal_districts, 2023. Accessed on Oct 05, 2023.

[177] Dublin City Council. Your areas. https://www.dublincity.ie/council/council-explained/your-area, 2023. Accessed on Oct 05, 2023.

[178] Zenodo. Dublin postcode boundaries - shane mcguinness. https://zenodo.org/records/4284592, 2023. Accessed on Oct 05, 2023.

[179] Smart Dublin. Administrative areas dcc. https://data.smartdublin.ie/dataset/administrative-areas-dcc, 2023. Accessed on Oct 05, 2023.

[180] Facebook. Prophet. https://facebook.github.io/prophet/, 2024. Accessed on May 26, 2024.

[181] Flask. Flask. https://flask.palletsprojects.com/en/2.3.x/, 2023. Accessed on Oct 05, 2023.

[182] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S Tucker. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739, 2016.

[183] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[184] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[185] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[186] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[187] Alexander Kuhnle, Michael Schaarschmidt, and Kai Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. *Web page*, 9, 2017.

[188] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[189] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov.

Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[190] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[191] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

List of Publications

Published

1. **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "A Survey on IoT Big Data: Current Status, 13 V's Challenges, and Future Directions", *ACM Computing Surveys (CSUR)*, [**ACM**], vol. 53, no. 6, pp. 1-59, article no. 131, 2020. [SCI Indexed, Impact Factor = 10.282] `https://doi.org/10.1145/3419634` [**Citations = 104**]

2. **Maggi Bansal**, Inderveer Chana, and Siobhán Clarke, "UrbanEnQosPlace: A Deep Reinforcement Learning Model for Service Placement of Real-Time Smart City IoT Applications, *IEEE Transactions on Services Computing*, [**IEEE**], vol. 16, no. 4, pp. 3043-3060, 2023. [SCI Indexed, Impact factor = 5.5] `https://doi.org/10.1109/TSC.2022.3218044` [**Citations = 11**]

3. **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "Enablement of IoT based Context-Aware Smart Home with Fog Computing", *Journal of Cases on Information Technology (JCIT)*, [**IGI Global**], vol. 19, no. 4, pp. 1-12, 2017. [Web of Science ESCI and Scopus Indexed] `https://doi.org/10.4018/JCIT.2017100101` [**Citations = 21**]

4. **Maggi Bansal**, Inderveer Chana, and Siobhan Clarke, "Autonomic Occupancy Detection of an IoT-Based Smart Building Using Deep Neural Network", *2nd International Conference on Machine Intelligence for Research and Innovation (MAITRI 2024)*, Jammu & Kashmir, India, Proc. in [**Springer**] (Springer Nature), 2024. [Scopus Indexed]

Communicated

1. **Maggi Bansal**, Inderveer Chana, and Siobhán Clarke, "Cloud4IoTCity: A Cloud Framework based on Fusion of Geospatial and IoT Big Data for Efficient Management and Spatio-Temporal Analysis of Smart City Data", *Information Fusion*, [**Elsevier**], [SCI Indexed, Impact Factor = 14.7, Under Review]