TU WIEN Informatics

Foundations for Sustainable and Trustworthy Edge Data Analytics

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Ivan Lujić, MSc
Registration Number 01529645

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Ivona Brandić

The dissertation has been reviewed by:

Professor Guillaume Pierre	Assoc.-Prof. Alessandro Vittorio Papadopoulos

Vienna, 20th December, 2021

Ivan Lujić

Informatics

Foundations for Sustainable and Trustworthy Edge Data Analytics

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Ivan Lujić, MSc
Matrikelnummer 01529645

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Ivona Brandić

Diese Dissertation haben begutachtet:

Professor Guillaume Pierre

Assoc.-Prof. Alessandro Vittorio Papadopoulos

Wien, 20. Dezember 2021

Ivan Lujić

Erklärung zur Verfassung der Arbeit

Ivan Lujić, MSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 20. Dezember 2021

Ivan Lujić

Acknowledgements

This dissertation is the result of my research work that has been partially funded through (i) the Rucon project (Runtime Control in Multi Clouds), FWF Y 904 START-Programm 2015, (ii) 5G Use Case Challenge InTraSafEd5G (Increasing Traffic Safety with Edge and 5G) funded by the City of Vienna, and (iii) my netidee scholarship by the Internet Foundation Austria (IPA). Part of the research in this thesis was carried out during my internship at IBM Research-Ireland.

I would like to thank the following people, who have helped me to complete this thesis, and without whom this thesis would have not been the same!

First and foremost, I would like to express my very great appreciation to my supervisor Univ. Prof. Dr. Ivona Brandić for her patient guidance, enthusiasm for the topic, and invaluable support. Her feedback always pushed me to bring my research and work to a higher level. I would also like to thank Professor Guillaume Pierre and Assoc.-Prof. Alessandro V. Papadopoulos for providing insightful feedback as reviewers of this thesis.

Advice given by Prof. Stipe Čelar, Dr. Toni Mastelić, and Dr. Maria Calatrava Moreno has been a great and valuable help when I began my doctoral studies. I would like to especially thank my colleague Dr. Vincenzo De Maio for his support through most of my research work and writings, which helped me to improve the quality of this work.

For many good meetings, conversations, and collaborations I am very thankful to my colleagues and project group members especially to Dr. Atakan Aral, Fani Bašić, and Josip Žilić. I would also like to thank Dr. Srikumar Venugopal for keeping my progress and making my internship at IBM Research-Ireland a productive experience. I also thank Stephanie Wogowitsch for all the help in administrative tasks over the last several years.

Finally, I wish to thank my parents (Mirko and Silvija) who offered their support, understandings, and belief in me. I acknowledge all those who helped me in some ways during various stages of my PhD but whose names have not appeared on this page.

Thank you and may God bless you all!

Ivan Lujić
December 2021
Vienna, Austria

Abstract

Massive amounts of data are continuously generated from a growing number of Internet of Things (IoT) devices. Based on the insights obtained through the analysis of collected data, different data-driven decisions are made to manage IoT systems. Traditionally, managing such systems includes data processing in the cloud. However, performing data processing in centralized cloud data centers brings serious challenges, including the transfer of huge amounts of sensor data over the network and new strict requirements (e.g., latency, accuracy, privacy) from IoT applications (e.g., smart buildings, smart traffic). For these reasons, edge computing has been introduced.

Edge computing represents a promising methodology and solution to execute analytics close to data sources using much smaller edge servers and devices. However, scalable and centralized cloud services cannot be generalized and directly applied to edge infrastructures. Also, IoT decision-making processes require timely and accurate data processing, bringing a new set of challenges to design sustainable and trustworthy edge data analytics. This is because performing edge data processing needs to deal with problems such as limited computational and storage resources; sensor data that can often be incomplete leading to inaccurate analytics; decentralized data locations posing difficulties for latency-critical analytics placement.

We address these problems by targeting data-centric perspectives for sustainable and trustworthy edge analytics. We introduce an adaptive data recovery mechanism for incomplete sensor data, improving the accuracy of data analytics and decision-making processes. Further, we propose an efficient edge storage management mechanism for keeping only the most relevant data in limited edge storage. We also propose a self-adaptive and data locality-aware edge analytics placement mechanism that minimizes the latency for performing edge analytics. We show the integration of edge computing and modern network communication technologies in a prototype deployment, ensuring reliability and privacy for critical IoT applications.

Finally, we evaluate the proposed solutions using real-world datasets, applications, self-designed benchmarks and testbeds using physical edge infrastructures. Besides novel edge data services and approaches making the theoretical contribution, we also show their practical applicability in IoT systems. The proposed solutions can help IoT systems to produce more accurate, reliable, and timely decisions, thus, contributing to foundations for sustainable and trustworthy edge data analytics.

Kurzfassung

Riesige Datenmengen werden kontinuierlich von einer wachsenden Zahl von Internet of Things (IoT) Geräten generiert. Basierend auf den Erkenntnissen, die durch die Analyse der gesammelten Daten gewonnen werden, werden verschiedene datengesteuerte Entscheidungen zur Verwaltung von IoT-Systemen getroffen. Traditionell umfasst die Verwaltung solcher Systeme die Datenverarbeitung in der Cloud. Die Datenverarbeitung in zentralisierten Cloud-Rechenzentren bringt jedoch ernsthafte Herausforderungen mit sich, darunter die Übertragung riesiger Sensordatenmengen über das Netzwerk und neue strenge Anforderungen (z. B. Latenz, Genauigkeit, Datenschutz) von IoT-Anwendungen (z. B. intelligente Gebäude, intelligentes Verkehrsmanagement). Aus diesen Gründen wurde Edge Computing eingeführt.

Edge Computing stellt eine vielversprechende Methodik und Lösung dar, um Analysen in der Nähe von Datenquellen mit viel kleineren Edge-Servern und -Geräten durchzuführen. Skalierbare und zentralisierte Cloud-Dienste können jedoch nicht generalisiert und direkt auf Edge-Infrastrukturen angewendet werden. Darüber hinaus verlangen IoT-Entscheidungsprozesse auf eine zeitnahe und genaue Datenverarbeitung, was eine Reihe neuer Herausforderungen bei der Gestaltung nachhaltiger und vertrauenswürdiger Edge-Datenanalysen mit sich bringt. Dies liegt daran, dass bei der Durchführung von Edge-Datenverarbeitung Probleme berücksichtigt werden müssen, zum Beispiel begrenzte Rechen- und Speicherressourcen; Sensordaten können oft unvollständig sein und zu ungenauen Analysen führen; dezentralisierte Datenstandorte können die Platzierung von latenzkritischen Analysen erschweren.

Wir behandeln diese Probleme, indem datenzentrierte Perspektiven für eine nachhaltige und vertrauenswürdige Edge-Datenanalyse angewendet werden. Wir führen einen adaptiven Datenwiederherstellungsmechanismus für unvollständige Sensordaten ein, der die Genauigkeit von Datenanalysen und Entscheidungsprozessen verbessert. Darüber hinaus schlagen wir einen effizienten Verwaltungsmechanismus für die Edge-Datenspeicherung vor, um nur die relevantesten Daten in einem begrenzten Edge-Speicher zu halten. Wir schlagen auch einen selbstadaptiven und datenlokalitätsbewussten Platzierungsmechanismus für Edge-Analysen vor, der die Latenz für die Durchführung von Edge-Datenanalysen minimiert. Wir zeigen die Integration von Edge Computing und modernen Netzwerkkommunikationstechnologien in einer Prototypbereitstellung, um Zuverlässigkeit und Datenschutz für kritische IoT-Anwendungen zu gewährleisten.

Schließlich evaluieren wir die vorgeschlagenen Lösungen mit realen Datensätzen, Anwendungen, selbst entworfenen Benchmarks und Testumgebungen mit physischen Edge-Infrastrukturen. Neben neuartigen Edge-Datendiensten und Ansätzen, die den theoretischen Beitrag leisten, zeigen wir auch deren praktische Anwendbarkeit in IoT-Systemen. Die vorgeschlagenen Lösungen können IoT-Systemen dabei helfen, genauere, zuverlässigere und zeitnahe Entscheidungen zu treffen und so zu den Grundlagen für nachhaltige und vertrauenswürdige Edge-Datenanalysen beizutragen.

Contents

Abstract ix

Kurzfassung xi

Contents xiii

List of Figures xv

List of Tables xvii

List of Algorithms xix

Previous Publications xxi

1 Introduction **1**
- 1.1 Problem Statement . . . 4
- 1.2 Emerging Fields in Edge Data Analytics . . . 6
- 1.3 Research Questions . . . 6
- 1.4 Scientific Contributions . . . 8
- 1.5 Significance of the Study . . . 11
- 1.6 Thesis Organization . . . 11

2 Background **15**
- 2.1 Research Focus and Main Use Cases . . . 18
- 2.2 Methods and Analytics . . . 20
- 2.3 Edge Hardware and Network Resources . . . 24
- 2.4 Environment and Tools . . . 27
- 2.5 Research Contributions Roadmap . . . 29

3 Data Management Strategies for Near Real-Time Edge Analytics **31**
- 3.1 Background on Edge Data Management . . . 32
- 3.2 Adaptive Data Recovery Mechanism . . . 35
- 3.3 Efficient Edge Storage Management . . . 41
- 3.4 Mediator Component for Supporting Data Recovery . . . 48

4 Elastic Edge Data Services for Supporting Decision Making 51
4.1 Importance of Elasticity in Edge Data Services 51
4.2 Motivational Use Case 53
4.3 Analysis of Edge Storage Services 54
4.4 Engineering Principles for Edge Data Services 58

5 Deployment of Edge Video Analytics Systems 65
5.1 Increasing Traffic Safety with Real-Time Edge Analytics and 5G . . . 66
5.2 Data Locality-aware Edge Analytics Placement 69

6 Evaluation 81
6.1 Edge Data Management Services Evaluation 81
6.2 System Deployment and a Performance Evaluation of Traffic Safety with Edge and 5G 94
6.3 Self-adaptive and Locality-aware Edge Analytics Placement Evaluation 103

7 Related Work 113
7.1 IoT Data and Resource-limited Edge Systems 113
7.2 Edge Data Management 114
7.3 Elastic Edge Data and Storage Services for Decision Making 117
7.4 Time-Critical Edge Analytics Systems 118
7.5 Data Locality-Aware Edge Analytics Placement 119

8 Conclusion 121
8.1 Summary 121
8.2 Limitations 123
8.3 Future Work 124

Glossary 127

Bibliography 129

A Curriculum Vitæ 149

List of Figures

1.1 IoT connected devices worldwide in 2019 and 2030, by technology 2
1.2 Placing cloud functionalities to the edge of the network for near real-time decision-making 3
1.3 A smart city edge analytics scenario 4
1.4 Organization and structure of the thesis 12

2.1 Research focus - edge data analytics for near real-time decision-making . 19
2.2 Time series with different characteristics and applied forecasting methods . 21
2.3 Evaluation of the forecast accuracy 22
2.4 An example of applied object detection technique on a real-world image . 24
2.5 Edge hardware including Raspberry Pi, USB accelerator and camera module 25
2.6 A testbed infrastructure for evaluation of edge data analytics approaches 27
2.7 A high-level overview of the application deployment in Kubernetes 29
2.8 Organization of scientific research contributions 30

3.1 A high-level architecture model overview for edge data management framework 33
3.2 Adaptive edge data recovery mechanism of incomplete datasets - flowchart 37
3.3 Forecasting process of adaptive data recovery for multiple gaps 40
3.4 Edge storage management flowchart and design principles 42

4.1 Traditional single analytics system for university smart buildings use case 53
4.2 End-to-end monitoring metrics of elastic edge services through four data stages 59
4.3 Application-specific data flows through a new edge architecture with corresponding components 60
4.4 Elasticity management determining customized data flows and storage services 62

5.1 An example traffic scenario to illustrate the critical situations when pedestrians and cyclists appear in the driver's blind spots 67
5.2 An overview of the architecture design for increasing traffic safety with edge 69
5.3 An example smart city scenario to illustrate problems of tracking dataset, and timely placement of critical on-demand video analytics 71
5.4 SEA-LEAP architecture overview 73
5.5 SEA-LEAP monitoring service including registration of changes of data locations and tracking data movements 75

6.1 Adaptive data recovery of multiple gaps (G_1^1, G_2^2, G_3^{17} and G_4^{30}) on dataset *h_1*, employing ARIMA, as STR approach 83
6.2 Results of semi-automatic recovery of multiple gaps 85
6.3 Running time of recovering all gaps 86
6.4 MAPE accuracy measure for three recovered gaps of missing values among 4 datasets 87
6.5 Absolute percentage error behavior along recovered gaps comparing with proposed *AdaptOpt* case 88
6.6 Observed dataset and forecast accuracy observation with stable accuracy clusters detection 90
6.7 Evaluation of edge storage management on *h_1* dataset - cycles 1-3 91
6.8 Released/retained data, appropriate cluster accuracy and clustered forecast accuracies percentage of Algorithm 5 after 5 cycles 92
6.9 Projection recovery maps for six evaluated time series datasets 95
6.10 Inference time observation for different edge node configurations and two object detection models 97
6.11 Confidence score observation for two object detection models on RPi 4 with Edge TPU (threshold set to 0.5) 97
6.12 MQTT network latency observations for QoS=$\{0, 1, 2\}$ over different network types (3G, 4G and 5G) 99
6.13 MQTT network latency for QoS=1 w.r.t. edge/cloud broker placement . 100
6.14 Edge nodes setup on Vienna's chosen intersection and the integration into the traffic-signal chambers 101
6.15 Real-time edge analytics demonstration for increasing traffic safety 102
6.16 Network latency for cloud/edge meta-server placement 106
6.17 SEA-LEAP testbed configuration 107
6.18 SEA-LEAP deployment YAML file example 108
6.19 SEA-LEAP placement calculation of node location candidates in different edge sites, driven by *GuideMe* and *FollowMe* actions 110
6.20 SEA-LEAP placement decision and network bound 112

List of Tables

3.1 EDMFrame main notations and definitions 35

4.1 Examples of data/system characteristics and impact on architectural design 55
4.2 Examples of application contexts and their impact on architectural design 56

5.1 SEA-LEAP main notations and definitions 74

6.1 Main characteristics of datasets for the EDMFrame experimental evaluation 82
6.2 Adaptive recovery dataset information 84
6.3 Stable clusters of forecast accuracy 90
6.4 Mean results of 5 storage management cycles per dataset 93
6.5 Comparison of PRM-based Multiple Technique Recovery (MTR) vs Single Technique Recovery (STR) for four gaps on six datasets 93
6.6 Edge node configurations 96
6.7 Comparative study of IoT protocols. Pub/Sub = Publish/Subscribe; R/R = Request/Reply; P2P = Point to Point 98
6.8 Edge node types used in the experimental setup, technical details and inference latency benchmarks for each node type 104
6.9 Main characteristics of datasets for SEA-LEAP evaluation 104
6.10 Network latency and bandwidth benchmark (Vienna's suburb) 107

7.1 An overview of relevant works for architecturing elastic edge data services 118

List of Algorithms

1 Data preparation 38
2 Gap identification 39
3 Detection of stable clusters 44
4 Detection of the appropriate cluster 46
5 Adaptive algorithm 46
6 Agent-based dataset registration 76
7 Meta scheduler 78
8 Placement optimizer 80

Previous Publications

This thesis is based on work published in peer-reviewed scientific journals, conferences and workshops. The following core papers build the foundation of this thesis. They are listed here once and will generally not be explicitly referenced again. Parts of these papers are contained in verbatim. Please refer to Appendix A for a full list of publications by the author of this thesis.

Refereed Publications in Journals

1. **Ivan Lujic**, Vincenzo De Maio, Srikumar Venugopal and Ivona Brandic. *SEA-LEAP: Self-adaptive and Locality-aware Edge Analytics Placement.* IEEE Transactions on Services Computing, 2021. DOI: 10.1109/TSC.2021.3104458 (Early Access)

2. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Resilient Edge Data Management Framework.* IEEE Transactions on Services Computing, vol. 13, no. 4, pp. 663-674, 1 July-Aug, 2020. DOI: 10.1109/TSC.2019.2962016

Refereed Publications in Conference Proceedings

3. **Ivan Lujic** and Truong Hong-Linh. *Architecturing Elastic Edge Storage Services for Data-Driven Decision Making.* In: Bures T., Duchien L., Inverardi P. (eds) Software Architecture, European Conference on Software Architecture (ECSA), Lecture Notes in Computer Science, vol 11681, pp. 97-105, Springer, Cham, 2019. DOI: 10.1007/978-3-030-29983-5_7.

4. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Adaptive Recovery of Incomplete Datasets for Edge Analytics.* IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), 2018. DOI: 10.1109/CFEC.2018.8358726.

5. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Efficient Edge Storage Management Based on Near Real-Time Forecasts.* IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 21-30, 2017. DOI: 10.1109/ICFEC.2017.9.

Refereed Publications in Workshop Proceedings

6. **Ivan Lujic**, Vincenzo De Maio, Klaus Pollhammer, Ivan Bodrozic, Josip Lasic and Ivona Brandic. *Increasing Traffic Safety with Real-Time Edge Analytics and 5G*. ACM Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys), pp. 19–24, April, 2021. DOI: 10.1145/3434770.3459732

CHAPTER 1

Introduction

In recent years, we have seen the proliferation of the **Internet of Things (IoT)** systems [1], representing the networking of physical objects with the purpose of collecting and sharing data. The adoption and usage of IoT sensors and devices are rapidly increasing, improving people's daily lives through various applications such as smart buildings and homes [2], traffic management and safety [3], eHealth [4], smart cities [5], and smart agriculture [6]. Such applications rely on monitoring different parameters such as temperature, air quality, humidity, health status (e.g., heart rate, blood pressure), proximity, electricity consumption, and video images, coming from plenty of IoT devices.

As reported by Statista and Transforma Insights [7], the overall number of IoT connected devices worldwide is forecast to almost triple from 7.74 billion in 2019 to 24 billion in 2030 (see Figure 1.1), at a compound annual growth rate (CAGR) of 11%. By 2030, most IoT connected devices (72%) are projected to be enabled by short-range technologies (such as WiFi, Bluetooth and Zigbee), while the public (dominated by cellular networks) and private networks will account for 20% and 8% respectively, overall generating an enormous amount of data. Based on collected sensor data, IoT systems are often designed to be intuitive and to make decisions like human bodies, allowing systems and applications to perform timely actions. Accordingly, managing any IoT system generally includes three steps, namely, sensor data collection, data processing, and acting based on the obtained results. Still, IoT has limited capabilities, and thus traditionally relies on distant, geographically distributed and massive cloud data centers to perform needed processing [8, 9].

Cloud computing represents a model for enabling on-demand network access to infrastructure, platform, and application services. Such cloud services are based on a shared pool of resources (e.g., data storage and servers) that can be elastically provisioned using the pay-as-you-go model [10]. The integration of IoT and cloud computing has been derived to manage the increasing amount of sensor data more efficiently while enhancing complex decision-making capabilities [11].

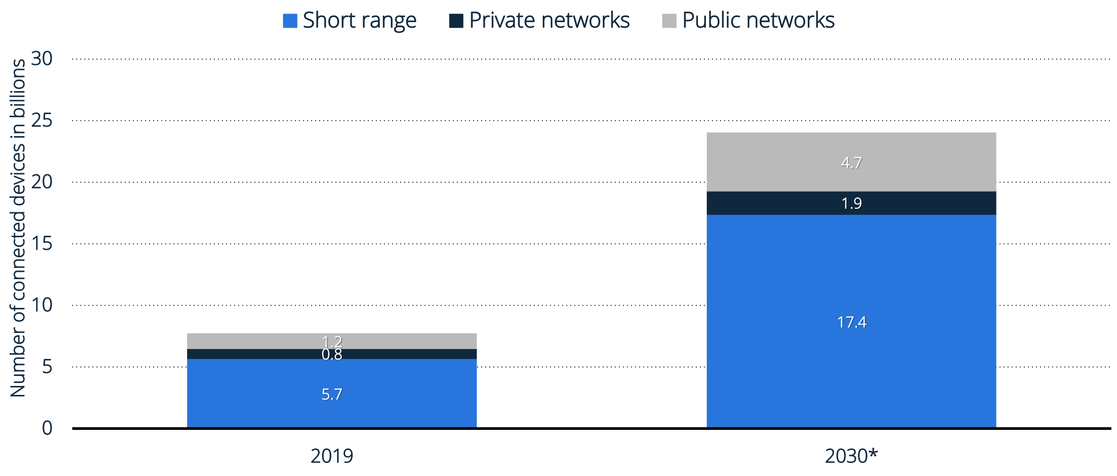


Figure 1.1: IoT connected devices worldwide in 2019 and 2030, by technology [7].

However, traditional cloud data processing imposes several challenges for managing emerging IoT systems. Critical applications like eHealth, smart buildings, or road safety have to process a big amount of collected data with strict accuracy and latency requirements [12, 13, 14, 15]. For example, smart building systems have requirements for automatic management of heating and cooling systems, either to foster energy efficiency or to support energy demand management systems integrated with smart grids [16]. Latency becomes a key factor in the performance of such critical IoT applications, requiring processing and communication latency in the range of milliseconds [17, 18].

As geographically distributed massive data centers are usually not in the vicinity of the IoT systems, performing analytics of IoT sensor data in the cloud brings the following issues: (i) the overall transfer of large amounts of IoT sensor data over the network including the execution of analytics can result in high and insufficient latency for decision-making processes [19]; (ii) future cloud data processing of IoT sensor data may be infeasible, since current bandwidth capabilities and network infrastructures cannot easily scale with the growing amount of IoT sensors and generated data, causing network communication bottlenecks [20]. Thus, reducing the time needed for collecting and processing the IoT sensor data is of paramount importance. Recent technological advances have disrupted the current centralized cloud computing model by moving resources close to data sources, as it is examined in [21, 22].

Edge computing has emerged as one of the most promising methodologies and solutions for providing data processing capabilities that are sufficient for critical IoT systems [23]. Edge computing is a computing paradigm that enables the data generated by IoT sensors and devices to be processed in the proximity of deployed systems. It is referred to as *any computing and network resources along the path between data sources and cloud data centers* [24], enabling timely insights for latency-sensitive applications while reducing data transfers and communication bottlenecks.

Edge computing has gained widespread attention from researchers and practitioners from both academia and industry. According to [25], by 2025, Gartner predicts that around 75% of enterprise-generated data will be processed at the edge, outside the traditional centralized data center or cloud, compared to only 10% in 2018. Consequently, the edge computing market size is rapidly growing worldwide and is expected to increase in revenues from USD 3.6 billion in 2020 to USD 15.7 billion by 2025, at a CAGR of 34.1% during the forecast period [26].

In edge data analytics, as illustrated in Figure 1.2, collection and analysis of data coming from IoT devices are performed using edge nodes. Edge nodes are designed as much smaller embedded systems to handle necessary processing tasks in the vicinity of IoT systems. Using edge nodes such as micro data centers, edge gateways and servers, or single-board computers (e.g., Raspberry Pi), it is possible nowadays to exploit computation resources across cloud boundaries and extend cloud functionalities to the network edge for necessary data processing [19, 27]. By placing data processing close to the source of data, edge architectures can significantly reduce the amount of data traversing the network and minimize the overall latency of decision-making processes [24]. Cloud layer offers more resources, and it can still be used for long-term and batch analytics considering either the most important sensor data or results from edge analytics.

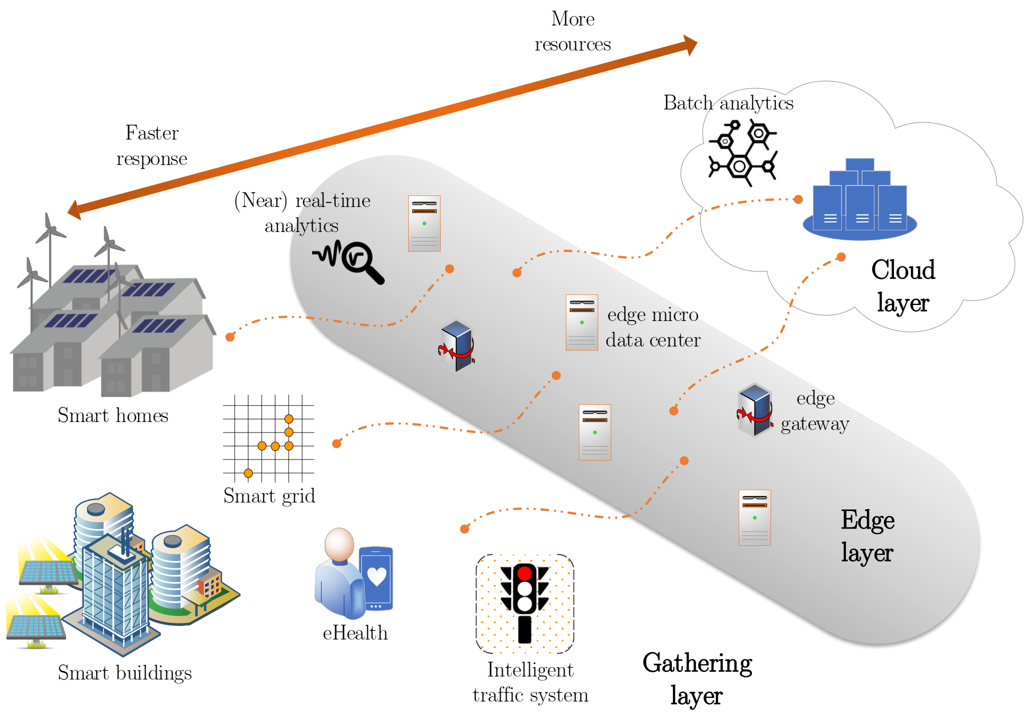


Figure 1.2: Placing cloud functionalities to the edge of the network for near real-time decision-making.

Decision-making processes of IoT systems rely heavily on the quality of the sensor data collected and their analysis [28, 29]. The requirement for handling the emerging IoT systems is causing rapid growth of the edge ecosystem. Accordingly, different data services and design features of edge infrastructures are required to enable sustainable and trustworthy edge data analytics [30]. This dissertation deals with self-adaptive and reliable data services as foundations for **sustainable** and **trustworthy** data analytics to ensure accurate and timely decision-making at the edge of the network.

In the following Section 1.1, we describe the research problems and motivate our research direction of decision-making in resource-constrained edge environments. Section 1.2 discusses sustainability and trustworthiness concerns of edge data analytics and how research problems in this thesis relate to such concerns. In Section 1.3, we provide an overview of addressed research questions, while Section 1.4 presents the main scientific contributions. We discuss the importance of the research and contributions to the field in Section 1.5. The thesis organization is outlined in Section 1.6.

1.1 Problem Statement

In this section, we examine more closely an edge analytics scenario in the context of a smart city to frame the research problems of this thesis. Based on the gathering (IoT sensors), edge, and cloud layers shown in the Figure 1.2, Figure 1.3 illustrates two typical IoT systems, namely, smart buildings and smart traffic, producing different sensor data. In Step 1 data are collected and transferred to edge nodes. In Step 2, the near real-time analytics are performed in edge nodes and based on these results, different actions can be executed, e.g., timely sending commands to actuators or safety notifications to manage IoT systems in Step 3. The most relevant subsets of data or results of edge analytics can be transferred to the cloud for batch analytics and long-term decisions in Step 4. However, current approaches for performing such near real-time data processing in edge

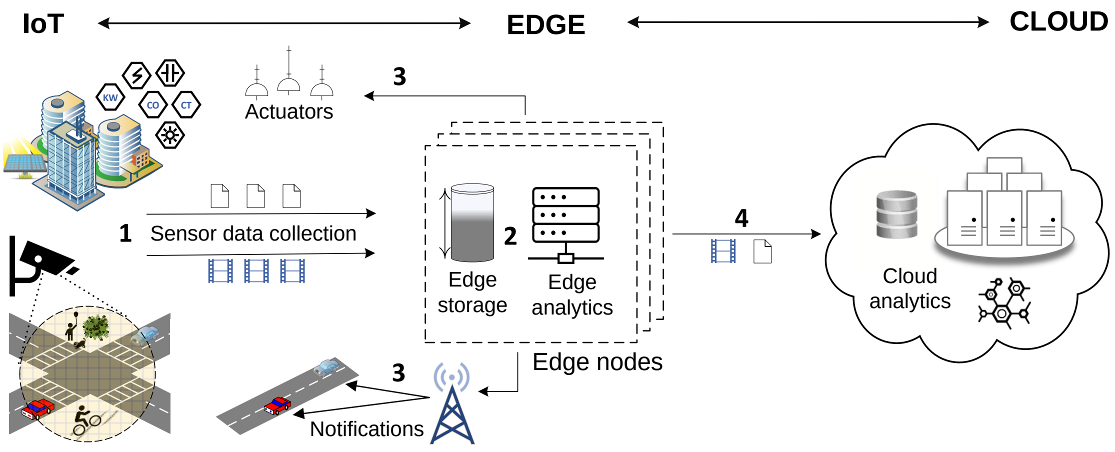


Figure 1.3: A smart city edge analytics scenario.

nodes face the challenges of achieving sustainable and trustworthy data analytics. To achieve sustainable and trustworthy edge data analytics, especially for latency-critical IoT applications like smart buildings or smart traffic, we address different perspectives with the emphasis on enabling accurate and timely data-driven decision-making.

- From a **data quality** perspective, data coming from IoT sensors can often be incomplete including missing and invalid measurement values, compromising the quality of edge data analytics. Errors, missing values and outliers may appear in data collected by IoT sensors, due to (i) the highly distributed nature of IoT systems; (ii) monitoring system failures; (iii) data packet loss in sensor networks; (iv) aging of the sensor; (v) changes in external conditions; or (vi) periodic sensor failures [31], affecting both near real-time and batch analytics. Traditional approaches for data management do not consider the impact of data quality on the accuracy of the decision-making processes, which is of paramount importance to ensure resilient and accurate analytics for IoT systems.
- From a **data storage** perspective, edge nodes contain limited storage capabilities while dealing with the rapidly growing amount of sensor data, making it infeasible to store all generated data [32]. Such edge limitations allow keeping only a certain amount of relevant data at the network edge, affecting the quality of data analysis as well as data-driven decision-making processes for IoT systems. As near real-time analytics require handling both historical and new data to perform accurate decision-making, especially when targeting proactive IoT systems, efficient edge storage management becomes an important problem at the edge [31].
- From a **data management** perspective, IoT devices and edge nodes can be highly decentralized and heterogeneous, making data collection, storage, and timely analysis difficult. Current state-of-the-art addresses data storage management problems only from individual perspectives such as data characterization [33, 34] or system operations [35, 36], and do not consider the impact on decision-making for IoT applications. Integration of strategies for the engineering of adaptive data and storage services is currently an unsolved problem [37]. Managing such adaptive edge data services within resource-constrained storage systems at runtime is fundamental for achieving sustainable edge analytics and data-driven decision-making.
- From an **architectural design** perspective, in critical real-world IoT applications such as increasing traffic safety, different notifications to road users need to be delivered with strict latency and accuracy requirements. Emerging challenges include (i) placement of edge nodes respecting urbanistic space constraints; (ii) selection of detection algorithms to identify dangerous situations based on deployed sensors; (iii) selection of a network technology allowing fast aggregation and delivery of information; (iv) providing real-time notifications to road users in critical intersections. Existing works target mostly vehicles with V2X communication capabilities [38]. The state-of-the-art solutions either do not take into account safety of road users (e.g., pedestrians) [39] or do not consider strict latency constraints [40].

- From a **data locality** perspective, input datasets required for edge analytics applications can often be subject to replication or transfer due to limited edge capacities, edge failure probabilities, or meeting certain service level objectives. One of the difficulties is how to automatically place analytics to the most appropriate edge nodes while considering both data locality and strict time requirements. Many state-of-the-art approaches for analytics placement address the data locality from aspects such as edge-cloud workload balance [41], a trade-off between resource usage and query accuracy [42], or the fairness of cloud resource allocation [43]. Still, they do not consider the adaptive placement of on-demand analytics for low-latency access to user-requested data in the distributed edge environment.

1.2 Emerging Fields in Edge Data Analytics

Despite the maturity of IoT and cloud systems, emerging edge data analytics raise new **sustainability** and **trustworthiness** challenges [44, 45]. In this context, **sustainability** represents one of the concerns in terms of (i) electricity consumption and costs involved in transferring data over the network to the cloud [46]; (ii) sending data through the network and choosing different design choices for IoT data processing that affect the carbon emissions [47]; (iii) having the ability for continuous and maintainable edge data processing without interruptions or problems. **Trustworthiness** represents a dimension of trust in terms of different aspects such as privacy concerns, reliability, resilience, safety, and correct functioning of the system [48, 44]. Assuring trustworthy systems always comes with costs, and finding the right balance in which users are ready to pay for trustworthy edge data analytics is an important challenge.

We consider data-oriented solutions to assure foundations for future sustainable and trustworthy edge data analytics that could unlock a positive impact in practical applicability for real-world IoT applications. We target foundations for **sustainability** from edge *data storage*, *data management*, and *data locality* perspectives. The massive growth of the Internet traffic by end IoT devices to and from data centers can be significantly reduced by targeting edge storage mechanisms, managing elastic edge data services, and adaptive analytics placements. We target foundations for **trustworthiness** from the *data quality* perspective (i.e., improving sensor data quality can make the decision-making processes more accurate and increase the resilience to monitoring failures), an *architectural design* perspective (i.e., selecting the right design choices can increase privacy and helpfulness of the latency-critical systems), a *data locality* perspective (i.e., exploiting data locality and self-adaptive analytics placements can improve reliability and ensure more trusted decision-making processes).

1.3 Research Questions

After describing research problems and their relation to sustainable and trustworthy edge data analytics, in this section, we outline five research questions that we address in the thesis. The first three questions are related to the near real-time edge analytics and resource limitations of edge nodes. The last two questions are related to the practical applicability and deployment of edge analytics systems.

Research Question 1
How can we efficiently recover incomplete sensor data, while dealing with strict accuracy and latency requirements of IoT applications?

IoT systems rely on data, collected from sensors and devices, that are either used for near real-time decisions or stored for long-term analysis. However, in highly distributed IoT systems, missing or invalid data may appear because of different reasons including sensor failures, monitoring system failures and network failures. Analyzing incomplete datasets can lead to inaccurate results and imprecise decisions, with negative effects on the target systems. Also, due to the increasing size of such systems and the consequently increasing amount of generated data, timely recovery of incomplete datasets is important for managing emerging IoT applications. Efficient recovery of incomplete datasets at the edge is a critical task, especially when dealing with strict latency and accuracy requirements from IoT systems.

Research Question 2
How can we store only the most relevant amount of data in space-limited edge storage, while keeping the quality of near real-time decisions?

The utilization of edge infrastructures, such as edge nodes, can reduce the amount of data traversing the network, minimizing latency and overall costs for IoT systems. Based on historical and new sensor data, edge nodes need to perform data analytics with certain accuracy requirements for decision-making processes. However, processing huge amounts of data at the edge is not possible due to the limited resources, i.e., storage capacity is not scalable to keep growing amounts of data. Limited edge capacities represent one of the critical bottlenecks for accurate edge analytics, requiring novel approaches to balance the quantity of data stored at the edge with the quality of near real-time decisions.

Research Question 3
How can we achieve elastic data services that rely on resource-constrained edge systems?

In the IoT era, a massive number of smart devices continuously produce a wide variety of data. To analyze these data for timely decision-making, data analytics at the network edge is a promising solution. Nevertheless, edge nodes contain limited storage and processing capacities posing a crucial challenge for efficient edge data analytics. Currently, these problems are addressed by considering traditional cloud-based database perspectives, including storage optimization and resource elasticity, while separately investigating data analytics approaches and system operations. To provide efficient edge data analytics, novel elastic approaches for data and storage services in decentralized edge have to be explored, while meeting latency objectives of time-sensitive IoT applications.

Research Question 4
How can we design edge settings to enable real-time analytics and support decision-making in critical real-world situations?

The emerging real-world IoT applications, such as increasing traffic safety, require strict latency and accuracy requirements. Despite advances in vehicle technology and road modernization, traffic accidents are a huge global issue, causing deaths and injuries, especially among pedestrians and cyclists. This often happens due to pedestrians and cyclists in drivers' blind spots or distractions delaying drivers' reactions. Therefore, timely warning drivers about critical situations is important to increase traffic safety. New edge computing and communication technologies have been proposed to reduce latency in critical IoT systems. However, state-of-the-art solutions either do not focus on traffic safety or do not consider low-latency requirements in this context. Thus, a new architecture design is necessary to address these challenges.

Research Question 5
How can we improve the overall latency of decision-making processes in which data analytics are dependent on data locality?

Performing edge analytics requires dealing with the rapidly growing amount of data, limited resources, and high failure probabilities of edge nodes. Therefore, data replication is of vital importance to meet Service Level Objectives (SLOs) such as service availability and failure resilience. In decentralized edge environments, it becomes difficult to track data movements or locations, and adaptively guide the placement of on-demand analytics tasks. Exploiting data locality in decentralized edge systems becomes crucial for improving decision-making processes. This is typical in applications such as object detection in video surveillance where the dataset of a specific camera can at different time points be stored in different locations (distributed edge storage nodes), preventing developers from timely and accurately executing queries or other analytic tasks. A new system design should consider data locality and provide low-latency access to requested input data.

1.4 Scientific Contributions

The state-of-the-art systems lack efficient data management strategies for time-sensitive edge analytics, especially targeting data-oriented solutions for edge computing and critical decision-making processes. In this thesis, we address (i) novel edge concepts and strategies showing the theoretical contributions and (ii) their practical applicability in real-world latency-sensitive IoT systems and applications. In this context, considering different edge computing infrastructures, driven by data characteristics and IoT requirements, **scientific contributions** of this thesis are the following.

Scientific Contribution 1
Edge data mechanism for adaptive recovery of incomplete datasets

To deal with incomplete IoT sensor data at the edge, while satisfying strict accuracy and latency requirements of IoT applications, we propose a generic mechanism for adaptive recovery of multiple gaps in incomplete time series. The proposed mechanism is able to remove outliers, detect and recover multiple gaps in datasets by employing different forecasting techniques, based on user specifications. Two approaches are shown considering user specifications, namely, using single-technique recovery (STR) and multi-technique recovery (MTR) by involving projection recovery maps (PRMs). PRMs are used to automate recovery of incomplete data, detecting an optimal trade-off between the gap size (number of missing values) and a range of historical data necessary to keep at the edge for accurate data analytics. We evaluate the proposed approaches by using real-world traces coming from IoT workloads such as smart homes and smart buildings. Our system shows the ability to adaptively recover various multiple gaps in datasets while reducing both the forecasting errors and computation time for different edge settings. This contribution has been previously published in [49, 50] and is detailed in Chapter 3.

Scientific Contribution 2
Edge data mechanism for efficient edge storage management

Regarding the challenge of storing only the most relevant data in space-limited storage, we propose a three-layer edge architecture model. In this model, we present an adaptive algorithm that dynamically finds a trade-off between providing high forecast accuracy necessary for efficient near real-time decisions, and minimizing the amount of data stored in the limited edge storage. We focus on time series data, typical in sensor-based IoT monitoring systems. The simulation results show that the proposed adaptive algorithm can reduce the amount of data in each computational cycle while satisfying demands for prediction accuracy. Therefore, our mechanism shows the potential for saving edge storage space and supporting reliable decision-making processes based on predictive analytics, in the context of storage-limited edge nodes. This contribution has been previously published in [51, 50] and is detailed in Chapter 3.

Scientific Contribution 3
Engineering principles of elastic data services for resource-constrained edge systems

For the better design of future data processing that relies on resource-constrained edge nodes, we provide a detailed analysis of elasticity and scalability of edge data and storage requirements together with edge analytics support. We propose a novel, holistic approach for architecting elastic edge storage services, featuring three aspects, namely, (i) data/system characterization (e.g., metrics and key properties), (ii) system operations (e.g., data filtering and sampling), and (iii) data processing utilities (e.g., data recovery, approximation and prediction). In this regard, we present seven principles for the architecture design and engineering of edge data services. Our contributions can have a direct impact on the quality of decision-making processes in time-sensitive IoT systems and can help researchers and developers to utilize proposed principles in edge data services. This contribution has been previously published in [52] and is detailed in Chapter 4.

Scientific Contribution 4
Real-world system for increasing traffic safety with edge and 5G

Regarding the challenge to design edge settings for enabling real-time analytics and supporting decision-making in critical real-world situations, we propose a system for increasing traffic safety with edge and 5G. It performs real-time edge analytics to detect critical situations and deliver early warnings to drivers. After describing our design choices, we provide a prototype implementation and evaluate its performance in a real-world setup. The evaluation shows that our system can (i) detect critical situations in real-time and (ii) notify affected drivers using 5G within expected latency requirements of road safety IoT applications. This work preserves privacy and ensures low latency, representing a promising step towards increasing overall traffic safety and supporting real-time decision-making in future edge-deployed applications. This contribution has been previously published in [53] and is detailed in Chapter 5.

Scientific Contribution 5
Self-adaptive and data locality-aware edge analytics placement system

Considering the challenge of executing on-demand analytics that depend on data locations, we propose a self-adaptive and data locality-aware edge analytics placement system. It includes a new mechanism for tracking data movements, on top of which we devise a generic control mechanism. Our system enables on-the-fly placement of on-demand analytics considering the most appropriate dataset location that minimizes overall analytics requests execution time. The experiments were conducted using real-world (i) object detection application, (ii) image datasets as input, (iii) self-designed benchmarks, and (iv) heterogeneous edge infrastructure using Kubernetes. The main novelty lies in the combination of data movement tracking and self-adaptive control logic for analytics placement on different hardware, facilitating both code-to-data and data-to-code movements. This can help users and developers to efficiently and timely deploy requested analytics across different edge infrastructures, indicating a promising solution for geo-distributed edge multi-cluster and hybrid environments. This contribution has been previously published in [54] and is detailed in Chapter 5.

1.5 Significance of the Study

This work represents an important cornerstone for future IoT systems and edge applications development, aiming to efficiently cope with (i) rapidly growing amounts of IoT sensor data; (ii) limited computation and storage capabilities of edge infrastructures; and (iii) strict requirements from IoT system to perform time-sensitive decision-making processes. The findings of this work should provide important insights for **system integrators** and **developers** that are responsible for combining different system subcomponents into the edge-cloud pipelines while creating dynamic adaptations and actions for efficient edge data processing.

This dissertation shows (i) **fundamental research** focusing on principles and theories as well as (ii) **applied research** focusing on practical solutions in real-world edge analytics settings. To solve identified research problems, this dissertation mainly involves the quantitative research methodology using different measurements and observations. The proposed solutions are available to the **scientific community** through several published journal and conference papers, while most of the developed code and implementation details are publicly available as open-source on GitHub, to facilitate greater reproducibility of research findings. This work demonstrates practical values that can serve as a guide for **industry practitioners**, e.g., enabling detection of critical situations at intersections (e.g., pedestrians and cyclists outside drivers' field of vision) by running real-time object detection on resource-constrained edge nodes integrated with smart traffic lights.

IoT solution architects can also directly benefit from given insights and improve decision-making processes in IoT systems considering data collection, storage, communication, data management and analytics. Consequently, this will help current data-driven systems to be more intuitive, (i) making the necessary transformation from reactive to proactive IoT systems and (ii) initiating the production of novel applications through different IoT/edge **research and development (R&D)** departments.

1.6 Thesis Organization

Figure 1.4 shows the thesis organization. The rest of the thesis is structured as follows:

- Chapter 2 describes our research focus in the thesis, addressing two fundamental use cases that motivate this work. Based on illustrated use cases, we introduce data types and edge hardware as well as define needed methods and environments that are used in proposed solutions in the following chapters.
- Chapter 3 presents an edge data management framework featuring different data services for (i) the recovery of incomplete time series coming from IoT sources and (ii) efficient edge storage management. We describe motivational scenarios and introduce a three-layer architecture with details about each of its components. This work covers the scientific contributions 1 and 2, and is derived from TSC 2020 [50], ICFEC 2018 [49] and ICFEC 2017 [51].

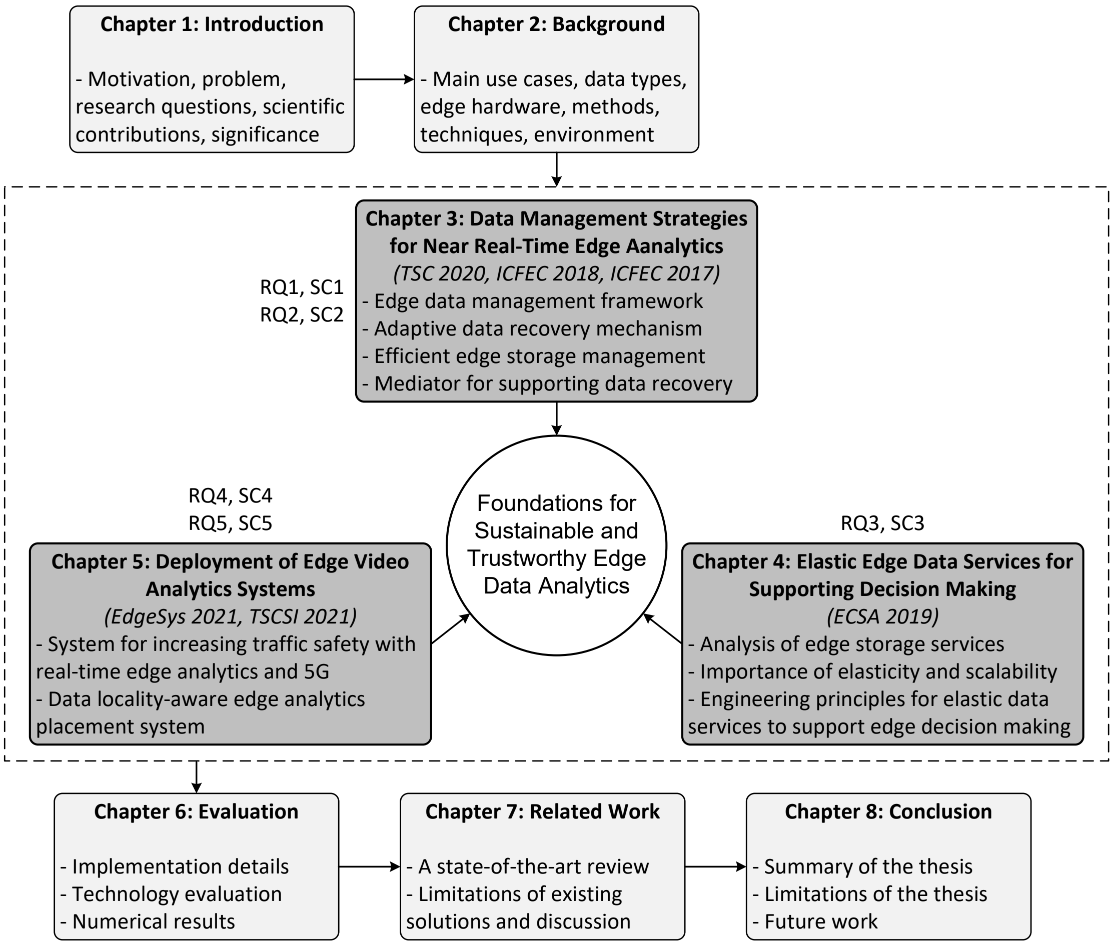

Chapter 1: Introduction
- Motivation, problem, research questions, scientific contributions, significance
Chapter 2: Background
- Main use cases, data types, edge hardware, methods, techniques, environment
RQ1, SC1
RQ2, SC2
Chapter 3: Data Management Strategies for Near Real-Time Edge Aanalytics
(TSC 2020, ICFEC 2018, ICFEC 2017)
- Edge data management framework
- Adaptive data recovery mechanism
- Efficient edge storage management
- Mediator for supporting data recovery
Foundations for Sustainable and Trustworthy Edge Data Analytics
RQ4, SC4
RQ5, SC5
Chapter 5: Deployment of Edge Video Analytics Systems
(EdgeSys 2021, TSCSI 2021)
- System for increasing traffic safety with real-time edge analytics and 5G
- Data locality-aware edge analytics placement system
RQ3, SC3
Chapter 4: Elastic Edge Data Services for Supporting Decision Making
(ECSA 2019)
- Analysis of edge storage services
- Importance of elasticity and scalability
- Engineering principles for elastic data services to support edge decision making
Chapter 6: Evaluation
- Implementation details
- Technology evaluation
- Numerical results
Chapter 7: Related Work
- A state-of-the-art review
- Limitations of existing solutions and discussion
Chapter 8: Conclusion
- Summary of the thesis
- Limitations of the thesis
- Future work

Figure 1.4: Organization and structure of the thesis.

- Chapter 4 provides the analysis of edge data and storage services and illustrates a motivational use case. We show the importance of elasticity and scalability in distributed and resource-constrained edge systems. Based on the analysis of edge requirements, we propose a set of engineering principles for elastic data services to support data-driven decision-making and describe potential implementation choices. This work covers the scientific contribution 3, derived from ECSA 2019 [52].

- Chapter 5 presents two deployments of edge video analytics systems. The **first** deployment presents the conceptual design of *InTraSafEd 5G*, a system for increasing traffic safety with edge and 5G, corresponding to the scientific contribution 4, derived from EdgeSys 2021 [53]. We provide the motivational use case and design requirements of a prototype implementation. An overview of the proposed architecture design is presented, and two main components are described in detail. The **second** deployment describes *SEA-LEAP*, a self-adaptive and locality-aware edge analytics placement system. This work covers the scientific contribution 5, derived from TSC 2021 [54]. We describe first a motivational use case and the importance of data locality. Then, a system design is proposed and the main components (tracking and control mechanisms) with corresponding algorithms are discussed in detail.

- Chapter 6 provides experimental evaluations of the proposed approaches. For each approach, we describe testbed setups as well as techniques and data that are used during experiments. We show and discuss numerical results and outcomes.
- Chapter 7 presents an overview of the state-of-the-art, their limitations and discussion. We organize related work into five categories, namely, (i) IoT data and resource-limited edge systems, (ii) edge data management, (iii) elastic edge data and storage services for decision making, (iv) time-critical edge analytics systems, and (v) data locality-aware edge analytics placement.
- Chapter 8 summarizes the thesis, provides conclusions and outlines limitations of the proposed solutions as well as possible future research directions.

CHAPTER 2

Background

In this chapter, we present the background information and discuss the main use cases and requirements for proposed approaches and implementations of edge data services. We describe used data types, research methods, and techniques necessary to evaluate the research aims and objectives of this thesis. We first explain the basic concepts of IoT-based systems, cloud, and edge computing.

Internet of Things

In the *Internet of Things* concept, "Things" refers to different types of physical devices (e.g., sensors, actuators, machines) able to transmit and share monitored data over networks. IoT devices represent a part of a bigger infrastructure allowing the collection, storage, processing and access to the produced data by users or other systems [55]. Regarding functionality, there are two types of IoT sensors, namely, (i) *sensors with memory*, in which data are temporarily stored and sent periodically to defined destination points (e.g., considering mobile IoT devices and intermittent network connectivity in remote areas); and (ii) *sensors without memory*, in which data is constantly generated and streamed to defined destination points (e.g., considering real-time environmental monitoring and feeding data into processing systems).

Some of the key properties and characteristics of IoT include [56, 57, 58]:

- **Interconnectivity**. IoT devices are interconnected over private or public networks, allowing autonomous interaction of sensors and other parts of IoT systems. To satisfy requirements for latency, reliability, and energy efficiency, there are different underlying wired and wireless communication technologies such as Ethernet, Wi-Fi, ZigBee, Bluetooth, cellular networks (GSM, UMTS, LTE).
- **Heterogeneity**. A heterogeneous distributed IoT system is usually made of various sub-systems. Therefore, the IoT devices can be highly heterogeneous in terms

of different hardware platforms, multiple network interfaces, and communication protocols for exchanging data.

- **Sensing**. Different IoT sensor types detect and measure the characteristics of their surroundings and environmental changes. Collaborative sensing plays an important role in enabling more intelligent decisions and better situational awareness.
- **Enormous scale**. The number of connected IoT devices that need to be managed has been rapidly growing over years and will continue to grow. IoT adoption offers new opportunities to improve decision-making processes, but at the same time raising challenges for handling interoperability and effective data processing.

Cloud computing

Transferring and analyzing historical IoT sensor data in resource-rich and centralized data centers enabled detecting threatening patterns in system behavior for improved management and decisions. According to NIST [10], *cloud computing represents a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing (e.g., networks, servers, storage) and software (e.g., applications and services) resources that can be dynamically provisioned and released.* Accordingly, the five fundamental characteristics of cloud computing are:

- on-demand self-service, meaning that users can access needed resources automatically without involving human interaction with different service providers;
- broad network access, meaning that computation resources are available for access over the network from multiple locations and a wide range of user devices;
- resource pooling, meaning that providers use a cost-effective pooling of computing resources to serve multiple consumers according to their demands;
- rapid elasticity, meaning that computing capabilities are elastically provisioned, i.e., dynamically increasing or shrinking the number of physical resources according to workload demands;
- measured service, meaning that cloud systems automatically control and optimize usage of infrastructure resources by monitoring the provision of services, providing transparency for both providers and consumers.

With the advent of IoT and its emerging applications, the amount of generated sensor data continues to explode aggravating the transfer, storage, and processing of collected data. Exceeding network capacity with enormous data traffic might cause serious challenges including degradation of Quality of Services (QoS) and slowing down the response time in critical IoT systems such as smart cities. For these reasons, cloud capabilities and resources are partially moving to the edge of the network, shaping the future of collecting and processing generated data.

Edge computing

Edge computing is a computing paradigm that enables data processing closer to the source of data. According to ETSI (European Telecommunications Standards Institute) [59], *edge computing is an evolution of cloud computing bringing application hosting from centralized data centers down to the network edge, closer to consumers and the data generated by applications.* However, there are different variants and visions of edge computing implementations [60] and the most common are:

- *fog computing (FC)* proposed by Cisco [23] as a highly virtualized platform that provides computing resources between end devices and traditional cloud data centers, while delivering a set of new services and applications at the edge of the network. As a new computing layer, it leverages devices like M2M gateways and wireless routers, to store and compute data;
- *cloudlet computing (CC)*, proposed by Satyanarayanan et al. [61] as a concept that involves proximate computing infrastructure with one-hop network latency that can be leveraged by mobile devices. They represent dedicated devices with capacities similar to a data center, but on a lower scale;
- *mobile edge computing or multi-access edge computing (MEC)* proposed by ETSI [62], as a key technology to enable ultra low-latency requirements and a rich computing environment for value-added services closer to end-users. It proposes the deployment of intermediate nodes with storage and processing capabilities in the base stations of cellular networks.

To perform data analytics based on IoT sensor data, we consider edge nodes (e.g., Raspberry Pi, edge gateways, or micro data centers) residing between IoT sensors and centralized cloud data centers. By keeping data analytics close to the source of data, edge nodes can minimize the latency for decision-making processes as well as to improve the energy efficiency of data transmission networks [63]. The fundamental characteristics that make edge computing extension of the cloud include [64]:

- **Location awareness and low latency**. Edge nodes are aware of their logical location regarding the entire system, and the latency for communicating with other interconnected nodes. Data analysis and response time are much faster than from remote and centralized cloud services.
- **Geographical distribution**. The services and applications targeted by edge computing often require geographically distributed deployments. Thus, edge nodes play an important role in delivering data processing capabilities close to end-users.
- **Heterogeneity**. Edge nodes can be heterogeneous regarding both hardware and software configurations as well as network communication capabilities. Further, they can be deployed in different environments.

- **Support for time-sensitive applications**. Edge computing applications involve near real-time and real-time interactions compared to traditional batch processing in the cloud (described further on this page).

Still, to deal with (i) a big volume of IoT data, (ii) limited edge resources, and (iii) low latency requirements, edge methodology should include additional components, modules, techniques, and algorithms, that we address in this thesis. We target edge nodes that are responsible for critical tasks, emergency monitoring, and response functionalities, e.g., to improve IoT systems, support decision-making processes, or reduce network traffic [65], with different time-sensitivity aspects.

Time-sensitivity of data processing

Regarding time-sensitivity of data processing, there are three aspects, namely:

- *batch* processing, meaning that processing is less time-sensitive including aggregation and processing of huge amounts of data in centralized data locations while taking more time to get insights over past historical data. For example, it can be in the range of minutes and hours for big cloud data analytics in healthcare systems [66].
- *near real-time* processing, meaning that the processing time is important for decision-making, but without guarantees of completing a data processing task within specific time constraints. For example, it can be in the range of seconds and minutes for sustainable energy management systems for smart buildings [67].
- *real-time* processing, meaning that the critical time period requirement needs to be fulfilled for data processing, i.e., technically, it must guarantee response times within specific constraints (deadlines) for real-time applications. Depending on the system, it can be in the range of milliseconds and seconds for time-sensitive cases (e.g., ensuring traffic safety [18] or autonomous industrial machines and cars [68]).

In this thesis, we target both near real-time (to address RQs 1 and 2 in Chapter 3, RQ 3 in Chapter 4, and RQ 5 in Chapter 5, Section 5.2) and real-time (to address RQ 4 in Chapter 5, Section 5.1) data analytics, through different edge use cases.

2.1 Research Focus and Main Use Cases

Figure 2.1 shows the research focus of this thesis (green shaded area), in which we address data management approaches for efficient edge analytics and latency-critical decision-making. We can see that IoT applications require latency-sensitive analytics based on data produced by different IoT sensors. Nonetheless, near real-time edge analytics is just a part of the bigger picture of distributed cloud/edge architecture, where data analyzed at the edge can be also further anonymized and used for other applications in the cloud. In this context, cloud computing can be used for non-time-critical analytics

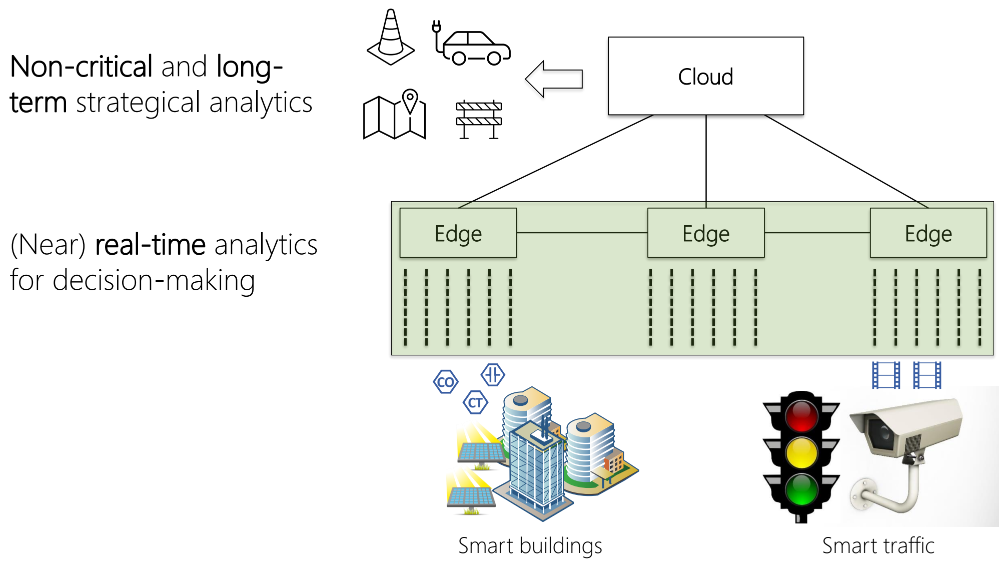

Non-critical and long-term strategical analytics
Cloud
(Near) real-time analytics for decision-making
Edge
Edge
Edge
CO
CT
Smart buildings
Smart traffic

Figure 2.1: Research focus - edge data analytics for near real-time decision-making.

and long-term strategic analytics (e.g., optimization of actions in case of accidents for police and ambulance, strategic planning of low emission zones for polluting vehicles, intelligent traffic management during congestion, creating high definition maps for self-driving cars). Contributions and insights from this work can help in improving emerging IoT applications in distributed and decentralized edge environments. In particular, it opens space for geographically distributed machine learning in a wider picture where captured data and results of edge analytics can be further used to maintain global models in the cloud for long-term strategical analytics.

As illustrated in Figure 2.1, in this thesis, we target two fundamental use cases **smart buildings** and **smart traffic**, producing two different data types, namely, sensor-based time series, and video data, and we refer to them throughout this thesis.

Time series

We partially performed evaluations of this work by collaborating with the Federal Real Estate Company (BIG) that provided us real-world data coming from TU Wien's Plus-Energy Office High-Rise Building, and thereby showing potential for improvement in the overall building management system and similar smart building scenarios.

We consider time series data that are very common in IoT data sources for applications like smart buildings and homes [2], in contrast to other data types [69] such as audio, status, or location data. Generally, data generated by sensor-based monitoring systems are classified as time series [70].

Definition 1. *A time series is a sequence of data points made over a continuous time interval, where each data point consists of a timestamp and one or multiple values.*

Time series can be collected from systems having either equal or unequal monitoring time intervals between data points. In the first case, they are made periodically, e.g., by reading outdoor temperature measurements, so-called *evenly-spaced* or *regular time series*. In the second case, they can occur when data production is triggered by certain events or while dynamically changing the static monitoring frequency to avoid redundant data during steady runs of the system [71], so-called *unevenly-spaced* or *irregular time series*. In this thesis, we focus on regular time series, since in our use cases data analytics rely particularly on regularly time-stamped measurements.

Univariate and multivariate time series. Univariate time series refers to a single observing variable, while multivariate time series includes two or more variables where each variable depends on its past values and on other variables. In this work, we target only univariate time series data, i.e., observations including only time stamps and measured values.

Video data

We partially consider evaluations of this work based on our *InTraSafEd 5G* project, funded by the city of Vienna to explore 5G use cases for a better connected smart city. We collected video data (video frames) coming from traffic cameras deployed in a typical edge smart traffic scenario used in the project. This type of data is typical in many different applications including increasing security and public safety.

The video frames streamed from IoT cameras need to be fed into video analytics applications in real-time to automate the analysis of the monitored surroundings. Managing such an amount of video data represents one of the major challenges from the video analytics perspective, representing a computationally expensive task including the execution of neural network inferences and other machine learning models in resource-constrained environments [72]. Therefore, performing edge video analytics is considered as the killer app for edge computing [73].

2.2 Methods and Analytics

IoT systems are designed to make decisions based on data collected from sensors and their analysis. To be able to store and analyze historical data timely before making any near real-time decisions at edge nodes, we need to make the transition from exact data analysis and reactive approaches to statistical paradigm and proactive approaches, incorporating different data science methods. The research challenges and problems for data-centric services and data management (described in the previous chapter) require different research methodologies that we detail in this section.

2.2.1 Data Science

Data-centric approaches of this work involve different phases such as data collection and preparation including data observations, sampling, and analysis of data. We apply data science techniques for time series analysis (RQ 1, RQ 2, RQ3) to improve the effectiveness of forecasting models as shown to be critical for many time-sensitive IoT systems [74]. Different statistical methods can be used to determine data characteristics, trends, and patterns in collected sensor data. Statistic measurements are fundamental in typical experiments that involve time series. For example, to choose appropriate forecasting methods in predictive analytics, it is necessary to explore data characteristics such as data stationarity and non-stationarity.

Stationary time series implies that the statistical properties (such as mean, variance, auto-correlation) of time series and the shape of its distribution are constant over time. *Non-stationary time series* implies that the statistical properties of data change over time, e.g., data points can include trends, seasonality, random walks.

Figure 2.2 illustrates representative examples of time series types with different characteristics. The first top three graphs show non-stationary patterns with seasonality and trends, while the bottom graph shows a dataset with stationary characteristics.

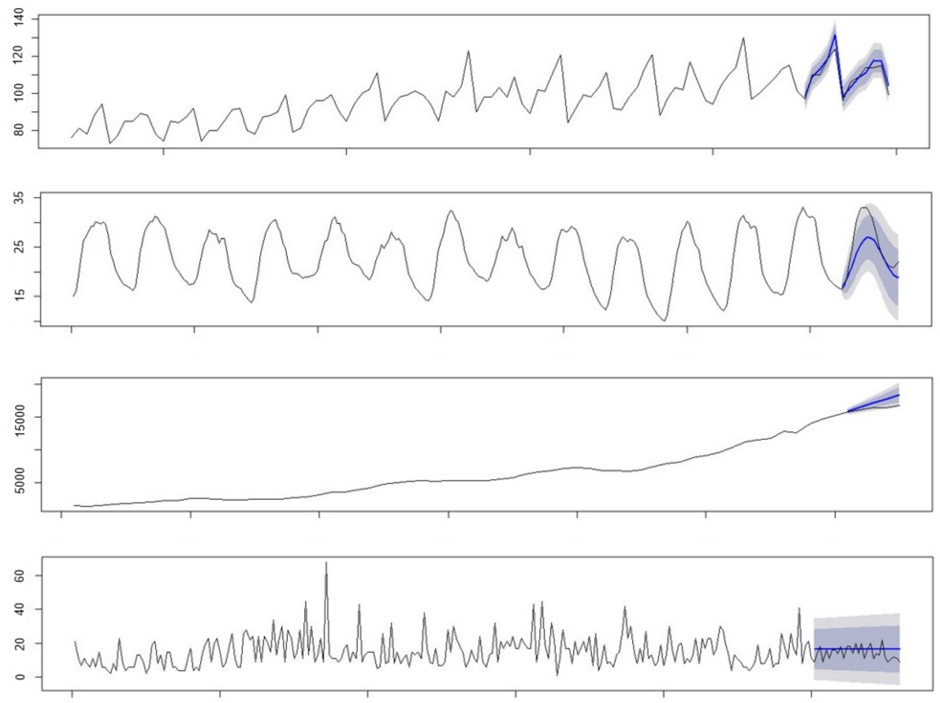


Figure 2.2: Time series with different characteristics and applied forecasting methods.

2.2.2 Time Series Forecasting

To deal with data management strategies (RQ 1, RQ 2, and RQ 3), we target the analysis and forecasting based on time series. Time series data analysis has been a very active research area over the past few decades. Using historical data it is possible to predict the future values of time series data. Based on time series forecasting and analysis of unusual data behavior, life-threatening situations and system failures can be detected even before they occur [70].

Predictive analytics has a huge potential to revolutionize critical and reactive IoT applications. We use different forecasting methods that rely on previously described statistical learning of data at the core. Based on characteristics and recognized patterns of historical data, predictive analytics and forecasting methods can help us to predict future data behaviors and consequently improve decision-making processes for managing IoT systems. Therefore, we target predictive analytics in the context of proactive decision-making. When historical numerical data are available, we can use different forecasting models to observe a forecast horizon.

A *forecast horizon* is the number of data points representing prediction length, i.e., the amount of data into the future for which the forecasts will be calculated. Figure 2.2 shows calculated forecast horizons in different datasets (blue lines) with prediction confidence intervals (shaded areas) within which calculated forecasts are expected with a certain probability. Different lengths of historical data and forecast horizons can impact the performance of predictive analytics in terms of error and inference time. Determining the accuracy of time series forecasting is an essential step in many decision-making processes. To evaluate the forecast horizon, accuracy measures are used.

Accuracy measures evaluate the forecast by considering how well the forecasting model performs on test data that were not included during the process. Accordingly, time series prediction evaluation is consisted of dividing datasets into two parts, namely, training and a test set. As illustrated in Figure 2.3, training data are used to estimate different parameters of the forecasting method to calculate forecast horizon, while test data are used to evaluate the forecast accuracy.

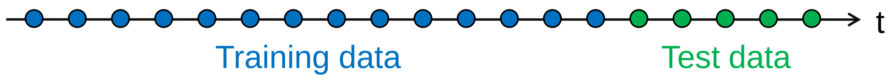


Figure 2.3: Evaluation of the forecast accuracy.

One of the most widely used measures for evaluating the forecast accuracy is Mean Absolute Percentage Error (MAPE), defined as:

$$\texttt{MAPE} = mean(|p_i|) = \frac{100}{n} \sum_{i=1}^{n} \frac{|(y_i - \hat{y}_i)|}{y_i} \tag{2.1}$$

where p_i is the forecast percentage error, n is the number of data points, $y_i - \hat{y}_i$ is the forecast error, y_i and $\hat{y}_i$ are respectively the i-th actual value of y and its forecast. MAPE is based on percentage error, meaning it is scale independent, enabling to compare forecast performances between different data sets.

Further, different IoT sensors can have different patterns in collected time series data (considering stationary and non-stationary characteristics). There are several widely used forecasting methods appropriate in our context. When data analytics rely particularly on regularly time-stamped measurements, the choice of automatic forecasting techniques, i.e., without constant user interactions, is appropriate for predicting critical events at the edge. Since we are faced with near real-time decisions and data handling operations in our algorithms, there are commonly used methods such as ARIMA and ETS. They are implemented in different forecast packages, featuring automatic parameter estimations.

AutoRegressive Integrated Moving Average (ARIMA) [75] models aim to describe the autocorrelations in the historical data based on their characteristics. To make the model fit the data, ARIMA captures three aspects, namely, *AR: Autoregression*, using the dependency relationship between an observation and its own lagged observations; *I: Integrated*, using the differentiation of observed data values (i.e., subtracting the observations from the previous values) to make the time series stationary; *MA: Moving Average*, using the dependency between an observation and a residual error from a moving average applied to lagged observations [76]. The automatic ARIMA method can obtain different and unusual patterns (e.g., high seasonality) [77], estimate needed parameters, and automatically compute suitable forecasts [78].

ExponenTial Smoothing (ETS) [79] models aim to describe important data components, such as *Trend* and *Seasonality* in combinations with an additive and multiplicative *Error* term. The modeling combination of these three terms results in 30 ETS models. Different models fit different sensor data from different applications [80], and can be used for capturing multiple complex seasonality in time series data.

2.2.3 Video Analytics

Video analytics refers to applications for processing video streams from cameras to detect or identify specific objects, events, or behaviors in the monitored environment. Such obtained information can help to perform certain actions and decision-making processes in IoT systems. To automatically complete processing tasks, video analytics often use artificial intelligence by applying different computer vision techniques and machine learning (ML) models. In this thesis, we target object detection, a popular computer vision technique.

Video analytics workflow for object detection requires the training of ML models, such as convolutional neural networks (CNN) [81], on manually annotated images. The standard Common Objects in Context (COCO) [82] is one of the large-scale datasets containing pre-annotated images for training computer vision models. In this work, we use pre-trained object detection models to detect different object classes in images.

For example, Figure 2.4 shows a result of the object detection workflow. A model trained to detect the presence and location of object classes (such as persons) is provided with an input image. The shown image is collected during our *InTraSafEd 5G* project demonstration (described later in Chapter 5). Once the input image is provided to the model, it will result in a list of objects, their spatial location (with a bounding box), and a confidence score. A confidence score is a number indicating the confidence of the model

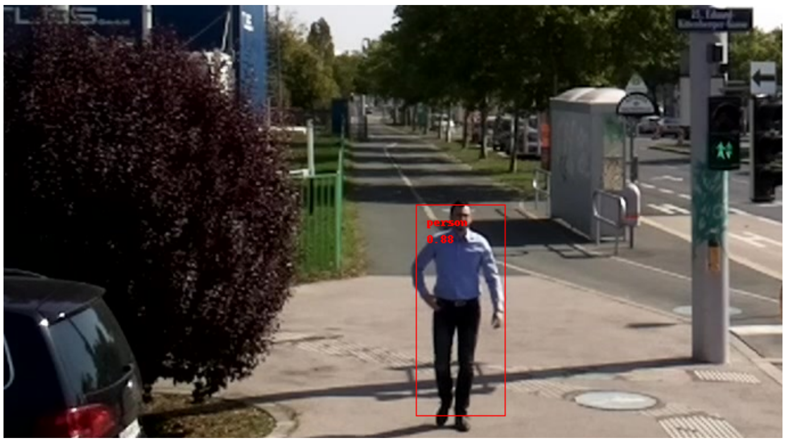


Figure 2.4: An example of applied object detection technique on a real-world image.

that the object was detected correctly, i.e., that the predicted bounding box contains the object. In the application, it can be also set as a cut-off threshold for accepting or discarding detection results.

Further, TensorFlow Light [83], a lightweight version of the popular TensorFlow platform, is proposed to build and deploy ML models on edge or IoT devices that contain limited resource capabilities. Therefore, to run ML models on edge devices, it is required to reduce the size of the model (i.e., affecting less storage space and memory usage).

MobileNetSSD v1 and v2. When building object detection networks, an existing network architecture is used within the object detection pipeline, and can result in a very large size in the order of hundreds of MB. For this reason we use MobileNetSSD v1 [84] and v2 [85], lightweight and pre-trained CNN-based object detection models trained using the standard COCO dataset. MobileNet represents the base network using convolution to produce high-level features (i.e., recognition and classification), and SSD (Single Shot MultiBox Detector) represents a detection network, i.e., an algorithm using the last convolutional layer on the base network for the detection task. Models such as MobileNetSSD v1 and v2 are optimized to run on resource-constrained edge hardware.

2.3 Edge Hardware and Network Resources

In this section, we describe edge hardware resources and network communication protocols used in our evaluation testbeds.

2.3.1 Hardware

Raspberry Pi (RPi). RPi is a single-board computer, enabling a small scale computing, e.g., in the deployment of IoT systems. The latest version RPi model 4B comes with Quad core Cortex-A72 (ARM v8) at 1.5GHz and 8GB RAM. It also features Gigabit Ethernet,

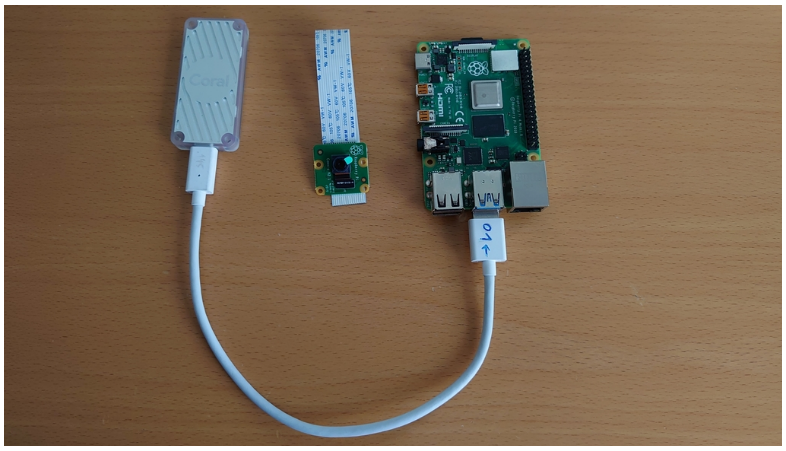


Figure 2.5: Edge hardware including Raspberry Pi, Edge TPU and camera module.

onboard wireless networking, Bluetooth, USB components, and enables connecting different peripheral devices such as camera modules and sensors. Considering its relatively small physical size, this RPi can be used in edge environments (e.g., homes, factories, weather stations), robotics, or attached to traffic lights. Figure 2.5 shows one RPi, its camera module V2, and the USB accelerator to speed up the inferencing of ML models.

Edge accelerators. Since edge nodes have constrained resources and limited power, they often include a TPU (Tensor Processing Unit) coprocessor to accelerate ML workloads. TPU represents application-specific integrated circuits (ASIC) capable to execute CNN, enabling different vision-based ML applications. We use Coral USB Accelerator that can be attached to RPi (as shown in Figure 2.5). Edge TPU requires TensorFlow Lite models (e.g., MobileNetSSD v1 and v2) that are quantized and compiled specifically for the Edge TPU to speed up video analytics. Before running an inference, the model is loaded into Edge TPU RAM to increase the performance.

2.3.2 Network Communication

Advances in connectivity, communications, and networking technologies are one of the key enabling points for the deployment of IoT systems through data exchange and derivation of actions. Together with the evolution of network technologies, edge data analytics (i.e., the potential of performing data processing at the edge of the network), are driving the growth of IoT forward to be more efficient and reliable. From the *network perspective*, edge means that the destination point is only a few network hops from the source point, while cloud means that data packets are transferred across the Internet backbone, often including different geographic regions.

Low latency delivered by modern communication networks like 5G is becoming a condition for latency-sensitive IoT applications [18] (e.g., to increase traffic safety in smart cities). With *5G-enabled phone* we were able to obtain different network measurements such as network latency and available bandwidth for various network types. Based on obtained measurements, it is possible to emulate real-world test conditions of an edge testbed in the lab. To improve experimental evaluation setup (Chapter 6), we target *Iperf*, a popular tool for performing real-time throughput measurements of different network types. To address RQ 4 and RQ 5 in Chapter 5, we used Huawei 5G router and Samsung S20 5G-enabled phone as shown in the lower part of Figure 2.6.

IoT Network Protocols

In the IoT environment, IoT devices need to communicate with other devices, systems, and services to exchange messages, data, or commands for actuators. Typical network communication models include:

- *Request/Response* or *Request/Reply* model [86] in which client software sends a request for data or services, and server software responds to it. One of the request/response-based application-layer protocols is *Constrained Application Protocol (CoAP)* [87] designed for resource-constrained IoT devices and limited network connectivity. CoAP is a one-to-one protocol mostly used in machine-to-machine (M2M) applications, and runs over the UDP transport layer, i.e., does not provide secure communication (e.g., handshakes) and guaranteed delivery of messages.
- *Publish/Subscribe (Pub/Sub)* model [88, 89] in which entities can exchange messages in a many-to-many style of communication. Three important components in Pub/Sub protocol are publisher, broker, and subscriber. Publishers are entities that send data to the topic managed by the broker. Subscribers are entities that subscribe to certain topics to receive and consume data distributed by brokers. Pub/Sub is widely used as an IoT communication pattern because of the distributed nature of IoT and its ability to support a rapidly growing number of IoT devices and services. For these reasons, we utilize Pub/Sub model in the edge system design in this thesis (while addressing RQ 4 and RQ 5 in Chapter 5).

Message Queue Telemetry Transport (MQTT) is a popular IoT network protocol that follows the Pub/Sub pattern. It is designed as a lightweight, client-server messaging protocol, in which publishers and subscribers represent clients and a broker represents a server. It runs on top of the TCP transport layer enabling many-to-many communication and allowing messages to be broadcast to different system components asynchronously. MQTT is suitable for small sensors and mobile devices, e.g., in M2M industry applications [90]. It offers fast response time, reliability, and different Quality of Service (QoS) levels for guaranteeing delivery of messages. We used open-source MQTT Eclipse Mosquito broker and MQTT Paho client for android application development.

2.3.3 Testbed

Figure 2.6 shows the heterogeneous edge infrastructure used in our testbed setup for experimentation (Chapter 5, Section 5.2). There are 12 RPis model 3B+ separate into 3

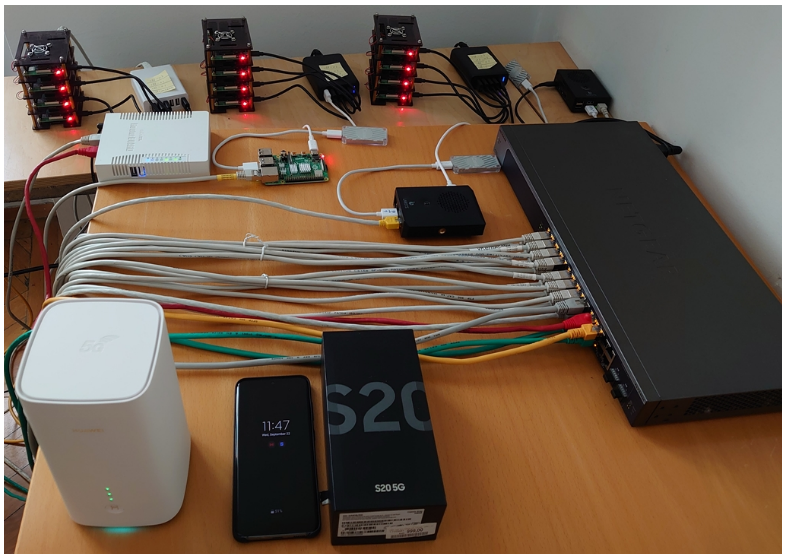


Figure 2.6: A testbed infrastructure for evaluation of edge data analytics approaches.

stackable cases and 3 RPis model 4B available in the lab. Further, all three RPis 4B have attached Coral Edge TPU (USB accelerators) and 8MP camera modules. All RPis are connected to the network with Netgear 24-Port 10-Gigabit Switch and Ethernet router.

2.4 Environment and Tools

In this section, we describe programming environments, platforms and tools together with their features that helped us to evaluate proposed approaches.

2.4.1 R Environment

R[1] represents a software environment and programming language for statistical computing, graphics, and data visualizations. Based on different packages it enables applying statistic functions and models, data manipulations, and predictive analytics. Some of the forecasting methods and models are implemented and available to users, including automatic ARIMA or ETS techniques.

To implement algorithms and approaches by addressing RQ 1 and RQ 2 in Chapter 3, we use *R* environment that can be extended with different libraries and packages for effective data handling and manipulations. Some of the used packages include *tseries* [91] (providing methods for analyzing time series data), *forecast* [78] (providing methods for univariate time series and forecasting techniques), *zoo* [92] (providing functions for regular and irregular time series), *ggplot2* [93] (providing functions for data visualizations and improving the quality of graphics).

[1]https://www.r-project.org/

2.4.2 Python

Python[2] represents a high-level programming language used for data science, handling huge amounts of data and building data science workflows. It is also often used by developers to build and run machine learning models. To address RQ 4 and RQ 5 (described in Chapter 5) we implemented different algorithms by using python libraries such as *numpy* (providing different mathematical functions for multidimensional arrays), *pandas* (providing methods for quantitative data analysis and manipulation), *cv2* (providing computer vision and machine learning algorithms), *PIL* (Python Imaging Library featuring image processing functionalities).

2.4.3 Docker

To support adaptive data analytics placement dependent on data locality from RQ 5, in the experimental evaluation, we consider the design methodology of microservices for more efficient and easily maintainable IoT analytic systems (as examined in [94]). Microservices represent an architectural approach to structure complex applications into a collection of smaller independent services. Such services offer easier deployment, maintenance, and scalability, compared to traditional monolithic architecture that aims to make an application as a single unified unit.

Docker[3] is the software containerization platform that enables the encapsulation of such microservices into containers. In this thesis, we consider data analytics services and IoT applications to be packaged up into docker containers and can be run across different machines, i.e., edge nodes. Considering the deployment and management of containerized applications, many researchers and enterprises are revealing nowadays the rapid adoption of Kubernetes platform. Docker provides integration with such existing platforms.

2.4.4 Kubernetes

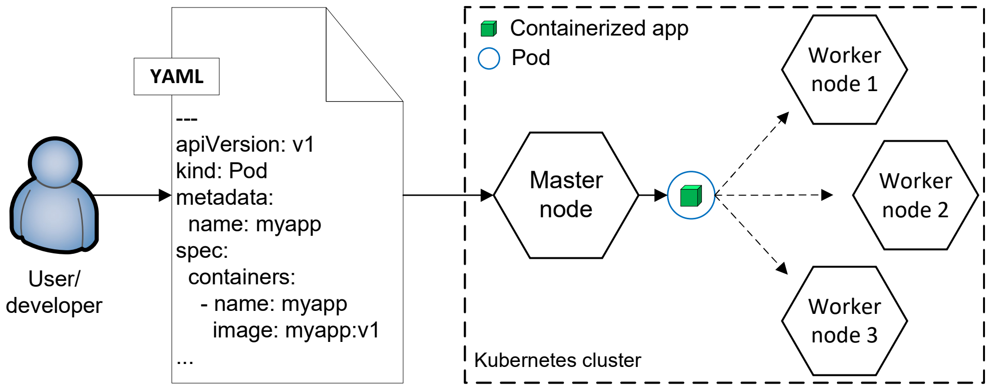
Kubernetes[4] represents one of the widely used open-source orchestration platform that automates the deployment and management of containerized services and applications across computational infrastructures. It is used by a broad community of researchers and industry practitioners.

Kubernetes uses a *pod* as a basic object of execution, referring to a single or group of related application-specific containers. So, the data analytics application (e.g., object detection) can be represented as a pod that will be deployed to one of the edge nodes, i.e., workers. We use Kubernetes as a platform for container orchestration, aiming to realize and support important edge data analytics approaches in this thesis. To deploy an application instance in Kubernetes, a set of specifications must be included to describe desired characteristics of the Kubernetes objects such as pods.

[2] https://www.python.org/
[3] https://www.docker.com/
[4] https://kubernetes.io/

YAML (Yet Another Markup Language). Kubernetes objects, such as pods, deployments and services, are created by using expressions in YAML configuration files, also known as Kubernetes manifests. Using various fields in the YAML simple text format, users can describe basic details such as the type of the Kubernetes resource, metadata, and a desired state of the application. Description of a desired state include specifications such as executable docker image containing the application software and resource requests (e.g., CPU, memory).

Figure 2.7 illustrates a simple example of deploying an application using Docker and Kubernetes. The typical way to deploy applications in Kubernetes include passing YAML file directly to the API server component on the master node. Based on the scheduler component, master node will then assign pods to worker nodes based on specification from YAML file, e.g., specified container image of the application, constraints and available resources of worker nodes. Based on the testbed setup from Figure 2.6, we created different single and multi-node Kubernetes clusters.



```
---
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - name: myapp
    image: myapp:v1
...
```

Figure 2.7: A high-level overview of the application deployment in Kubernetes.

2.5 Research Contributions Roadmap

Lastly, Figure 2.8 illustrates the research contributions of the thesis through the following three main chapters. As described in this chapter, we focus on two main uses cases and corresponding IoT sensor data types, namely, time series and video frames. Chapter 3 presents edge data management framework featuring adaptive recovery of incomplete data and efficient edge storage mechanisms. Chapter 4 addresses data and storage management perspectives of distributed edge nodes with the emphasis on elastic data services for achieving sustainable edge analytics and data-driven decision-making. Section 5.1 in Chapter 5 proposes a system design for integrating edge infrastructures and modern communication technologies for real-world critical IoT scenarios such as increasing traffic safety. Finally, Section 5.2 in Chapter 5 proposes a self-adaptive placement of edge analytics based on data locality, featuring tracking and control mechanisms.

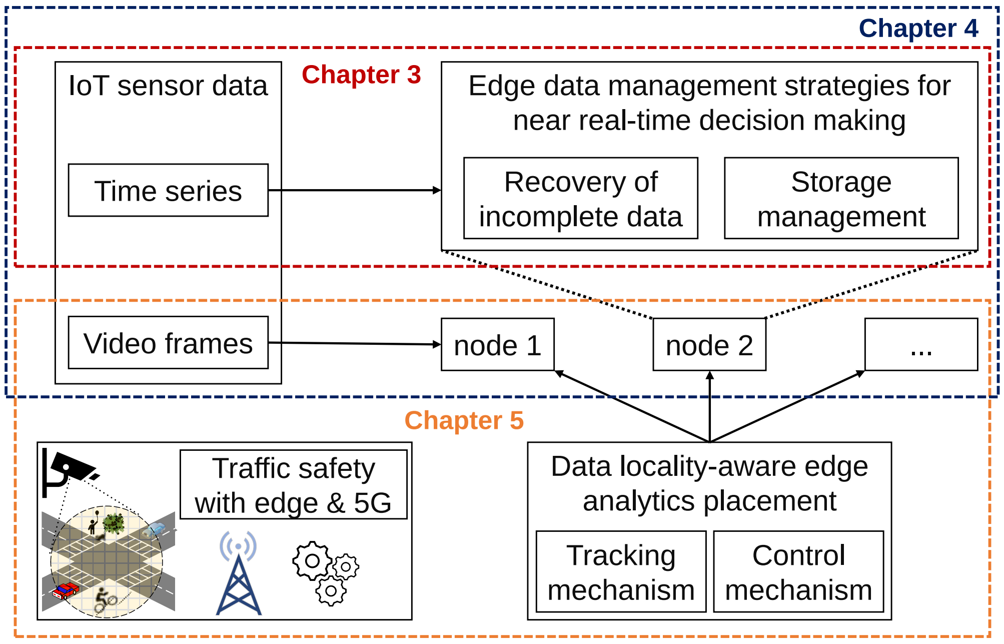

Chapter 4
Chapter 3
IoT sensor data
Time series
Video frames
Edge data management strategies for near real-time decision making
Recovery of incomplete data
Storage management
node 1
node 2
...
Chapter 5
Traffic safety with edge & 5G
Data locality-aware edge analytics placement
Tracking mechanism
Control mechanism

Figure 2.8: Organization of thesis research contributions.

CHAPTER 3

Data Management Strategies for Near Real-Time Edge Analytics

In the previous chapter we presented background on edge data analytics, our research focus, necessary research methods and environments. In this chapter, we first address challenges of limited resources of edge nodes, while dealing with strict latency and accuracy requirements from IoT applications. Further, we focus on the problem of incomplete sensor data and its impact on near real-time decision-making processes.

We introduce Edge Data Management Framework (*EDMFrame*[1]), a three-layer architecture model for resilient edge data management. The main contributions include:

- **a novel, generic mechanism for adaptive recovery** of incomplete time series, incorporating a recovery cycle that ensures outliers removal, detection, and forecasting of each gap, using single-technique recovery (STR);
- **edge storage management mechanism** that achieves a trade-off between the amount of data stored at the edge and high accuracy for predictive analytics;
- **a mediator component** featuring Projection Recovery Maps (PRMs) that detect the necessary range of historical data to recover different gaps in datasets, as well as to recommended recovery techniques, enabling multiple-technique recovery (MTR).

We propose a three layer architecture model in Section 3.1.2. Section 3.2.2 presents data recovery mechanism, while Section 3.3.1 shows the edge storage management algorithms. Section 3.4 describes the novel mediator component. Experimental evaluation and discussion are shown in the evaluation Chapter 6, in Sections 6.1, 6.1.2, 6.1.3 and 6.1.4.

[1]`https://github.com/lujic/EDMFrame`

3.1 Background on Edge Data Management

The IoT has recently attracted attention from both academia and industry. Billions of devices are getting connected to the Internet [95], generating huge amounts of data. Today, IoT sensors are used in many applications, like eHealth [4, 13], smart manufacturing [96], smart home and building systems [2], and smart cities [97, 98]. These systems require sensors data collection, data analysis, and acting based on the results of the analysis. Usually, IoT data are processed in geographically distributed and distant cloud data centers [8, 9]. However, cloud data processing performance is affected by the increased size of data and due to the limited scalability of current network infrastructures [20]. Also, respecting critical predictive analytics in modern IoT systems, meeting the strict latency and accuracy requirements [12] of decision-making processes imposes new issues.

Edge analytics is a promising solution to latency and network size challenges by employing edge nodes, i.e., smaller scale cloud data centers deployed closer to IoT sensors [22]. Performing data processing in edge nodes allows near real-time decisions for IoT systems [23]. However, edge analytics has many open challenges. **First**, missing or invalid data may appear due to different reasons such as monitoring system failures, data packet loss, sensor aging, or changes in external conditions [31]. Performing analytics on incomplete data can lead to inaccurate decisions [2, 99]. **Second**, compared to cloud data centers, edge nodes have limited storage and scalability affecting the accuracy of predictive analytics and, consequently, decisions for critical applications such as smart buildings [2] or manufacturing systems [96].

3.1.1 Edge Data Management Solutions and Limitations

There are different data management strategies for IoT and edge systems, considering IoT requests offloading [100], IoT resource management [4] and IoT security mechanism [101]. However, mentioned solutions focus on QoS for distributed edge data processing, workload management, and ensuring the security of IoT sensitive data rather than focusing on data reconstruction and storage management. Although some works propose various reconstruction methods of incomplete datasets [102, 103], they do not distinguish recovery of various gaps, despite diverse data characteristics.

We argue that for timely and accurate data recovery in modern IoT systems, it is important to combine different recovery techniques, even within the same datasets. Predictive analytics has a huge potential to revolutionize critical and proactive IoT applications, such as accurate diagnosis of patients in eHealth, maintenance services, and failures prevention in smart manufacturing and building systems. For decision-making processes, Sensor-Cloud Infrastructure [104] is a promising solution. Other works like [105, 106] discuss IoT sensor data reduction and dynamic compression techniques, focusing on network optimizations. Still, there is a lack of solutions for accurate predictive analytics while dealing with incomplete data and limited edge storage capabilities. Traditional approaches for IoT and cloud data management address the challenges related to incomplete datasets and storage limitations [107, 8, 9], without considering the impact of data quality and data resilience on decision-making processes, which is of paramount importance to ensure efficient and accurate analytics in IoT systems [108]. In the following sections, we describe our proposed edge data management architecture model with all components in detail.

3.1.2 Edge Data Management Architecture Model

Figure 3.1 shows an overview of the *EDMFrame* architecture model. At the time we write, several frameworks for IoT data processing have been proposed, such as Eclipse Kura[2], Node-RED[3] and Flogo[4]. Most of these frameworks focus on integrating heterogeneous IoT devices and on the interplay between the edge and cloud layers. They either do not provide methods for data recovery and predictive analytics, or they focus only on a specific technique for performing these tasks. Complementary to these works, we design *EDMFrame* as a service built on top of these or similar IoT data processing services, to enhance the data recovery and analytics features they offer. The aim is to devise a mechanism to deliver accurate near real-time decisions while coping with (i) incomplete data, (ii) a big volume of data, and (iii) limited storage resources at the network edge. The architecture includes three software layers, namely: gathering layer, edge layer, and cloud layer. Even though the main focus of this paper is the edge layer, we describe all of them for completeness.

Gathering layer transmits IoT measurements to the edge layer to reduce communication costs, save bandwidth and meet latency requirements in distributed sensor networks. Gateways at this layer can aggregate sensor data sending them in an appropriate format

[2]https://www.eclipse.org/kura/
[3]https://nodered.org/
[4]https://www.flogo.io/

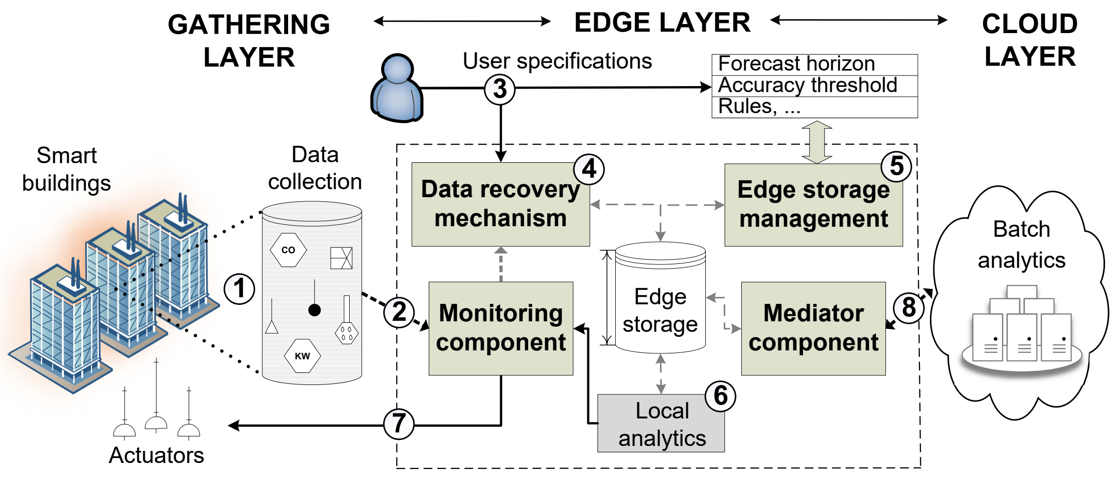


Figure 3.1: A high-level architecture model overview for edge data management framework, based on smart buildings use case.

and size to the monitoring component. In step (1) (see Figure 3.1), data are collected from smart buildings and then in step (2) transferred to the edge layer.

Edge layer manages data through different stages of *EDMFrame*, to perform accurate and timely analytics. It is composed of edge nodes, e.g., edge servers and micro data centers [22], aiming to perform data processing closer to data sources. *EDMFrame* includes the following elements:

Monitoring component. This component (i) receives and analyses data to detect outliers and missing values, (ii) notifies the mediator component about incomplete data, (iii) prepares data for the data recovery mechanism and (iv) triggers IoT actuators based on local edge analytics. It can also extrapolate data characteristics for further analytics.

Specification list. Once data are transmitted to the edge layer, user specifications are checked in step (3). Specification list contains application-dependent and user-defined parameters, useful for both data recovery and edge data management process (e.g., forecast horizon, monitoring frequency, forecast method, accuracy threshold, conditions).

Data recovery mechanism. The adaptive recovery process is performed in step (4). It receives data from the monitoring component and performs semi-automatic recovery of multiple gaps incorporating recovery cycles (see Section 3.2.2). The output is a dataset without gaps and cleaned from outliers.

Storage. Edge storage carries limited capacities. It stores data coming from the data recovery mechanism and communicates with the edge storage management, mediator component, and local edge analytics processes.

Edge storage management. In step (5), the edge storage management mechanism maintains limited storage keeping only data relevant for near real-time decisions. It checks available data, validates the specification list, and implements the edge storage management phases (see Section 3.3.1). The available data are used in step (6) for local analytics, whose output is forwarded either to the storage or to the monitoring component sending commands to actuators in step (7).

Mediator component. The mediator manages projection recovery maps to support the data recovery mechanism (see Section 3.4). In step (8), the mediator component communicates with the cloud data repository. It transfers the necessary data from/to the cloud. It can also perform data filtration and data transformation to improve data transfer between edge and cloud layers.

Cloud layer contains the data repository, storing historical data collected from IoT systems. It performs compute-intensive big data analytics based on entire datasets.

Sections 3.2, 3.3 and 3.4, detail all edge layer components. We design an experimental implementation as a pipeline with the core algorithms: monitoring component (Algorithm 1), data recovery mechanism (Algorithm 2) and edge storage management (Algorithms 5 utilizing Algorithms 3 and 4). Table 3.1 lists the main notations used hereafter.

Table 3.1: EDMFrame main notations and definitions.

Notation	**Description**
D_{in}	Matrix that represents incomplete input data.
D_{fr}	Matrix that represents framed (prepared) data to be stored.
ω	Vector that contains all indexes of missing values.
nom	Variable that counts number of missing values (based on ω).
$\hat{\omega}$	Vector that stores indexes of the current gap ($\hat{\omega} \subset \omega$).
γ	Vector containing forecast accuracies from the iteration phase.
v_γ	Standard deviation (volatility) of the entire vector γ.
s_f	Scaling factor dividing v_γ to set threshold for finding clusters.
CL_{th}	Threshold in identifying stable accuracy clusters.
Δ_v^γ	Set of standard deviations calculated from the sampled γ.
$\mathcal{C}$	Matrix containing detected stable accuracy clusters.
f_{th}^{ac}	Forecast accuracy threshold.
CL_{ap}	Appropriate cluster with stable forecast accuracy.
f_h	Forecast horizon - the amount of data as prediction length.
S_d	Array representing available dataset in storage.
d_f	Decrement factor that decreases available dataset S_d.
df_{pct}	Decrement factor percentage.

3.2 Adaptive Data Recovery Mechanism

In this section, we focus on the data recovery mechanism based on the architecture model proposed in the previous section. We propose a novel mechanism for the efficient recovery of incomplete time series datasets. We first provide background and motivation for the recovery of incomplete datasets. Then we introduce our data recovering mechanism, describing each component and related algorithms. We also provide an analysis of the algorithms' complexity. The experimental evaluation is presented in Section 6.1.2.

3.2.1 Motivation for Recovery of Incomplete Datasets

The use of IoT architectures is constantly increasing, and consequently also the amount of data collected by IoT smart devices [20]. Collected datasets can bring valuable information by performing timely data analytics. However, to extract meaningful information from the data, we need to ensure that datasets are complete and cleaned from outliers.

One of the use cases we select is *smart home/building applications*, where energy-efficient smart homes and buildings are equipped with automated energy management systems integrating different components such as renewable energy generation (e.g., solar and wind turbines), smart meters, and smart sockets [16]. We consider the impact of incomplete datasets in this scenario from two perspectives: (i) batch (long term), and (ii) near real-time (short term) analytics. Incomplete datasets may affect batch analytics on historical data, affecting management systems (e.g., heating and cooling management), and thus decreasing energy efficiency while increasing operational costs. Moreover, it can affect near real-time power management systems in the case of power fluctuation caused by intermittent renewable sources of electricity. Analytics performed on incomplete datasets may affect also load balancing in smart grids, and reliability of energy supplies [99].

We can find such challenges in other real-world scenarios. In intelligent traffic management systems [109], the traffic situation is constantly monitored by different types of IoT environmental traffic sensors [3] to optimize/control traffic flow and avoid collisions and congestions. Inaccurate analysis due to incomplete data may affect traffic monitoring, with negative effects on collision and congestion avoidance systems. Concerning eHealth [4], where patients' health is constantly monitored by different types of sensors to treat different diseases, incomplete data may affect the accuracy of health monitoring and prevent a timely reaction in case of problems. Therefore, it is important to efficiently recover incomplete data before processing them.

In our approach, we consider data analytics performed on the edge nodes to deal with the increasing scale of systems such as smart buildings and homes. However, the edge layer has limited resources in comparison with the cloud layer, where resource-demanding batch analytics can be performed. Solutions such as resource management mechanisms [14] and optimized service placement [110] have been proposed to deal with the resource constraints of the edge. However, such approaches do not propose efficient and adaptive solutions for data quality improvement. Additionally, state-of-the-art works do not consider the time-critical demands in the context of IoT applications and the improvement of data quality by using different forecasting techniques. Therefore, we bridge this gap by introducing user-defined and condition-based recovery in the choice of different forecasting techniques for adaptive recovery of incomplete data on the edge. In this work we focus on the edge layer, leaving the interplay between cloud and edge for future work.

3.2.2 Adaptive Mechanism Components

We present an adaptive mechanism for sensor time series data recovery. First, we define a *gap* as a sequence of one or more missing or invalid consecutive values, distributed in time series. Missing values occur due to sensor or monitoring failures, while invalid data represent outliers due to measurement errors.

Definition 2. *A gap G_n^k represents the n-th gap in an incomplete dataset with k missing/invalid values.*

For example, G_2^{17} refers to the second gap with 17 missing values. In Figure 3.2, we provide a flowchart of the proposed recovery mechanism. First, data are prepared in the monitoring component. To this end, data indexes from each gap in the dataset are

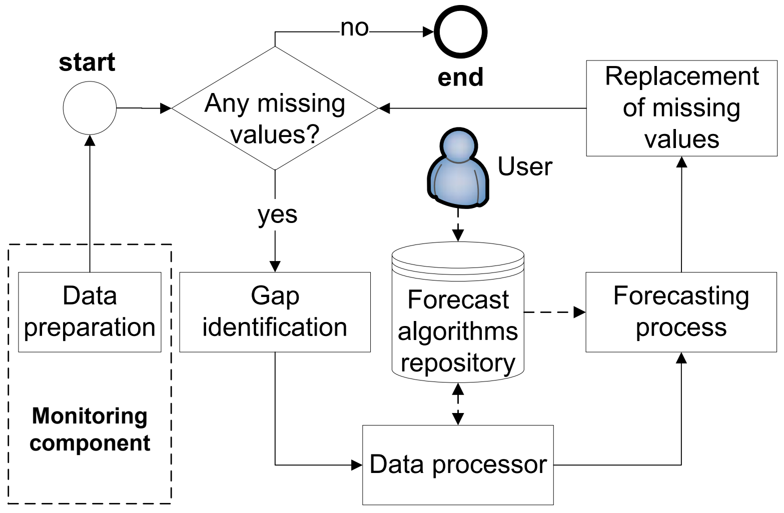


Figure 3.2: Adaptive edge data recovery mechanism of incomplete datasets - flowchart.

detected and marked in the data preparation module. Then, the recovery cycle starts by detecting the amount of missing/invalid values. The cycle terminates when there are no more missing values. Otherwise, the gap identification component detects the size of the current gap, selecting it for the current recovery cycle. The data processor component analyses data points preceding the current gap, that are important for the setup of the forecasting process, including user specifications. Then, necessary data and techniques selected from the repository are forwarded to the forecasting process. Once the gap is recovered, conditions for the next recovering cycle are checked. The following subsections describe all components in detail.

Data preparation

The goal of this component is to prepare an incomplete dataset for the recovery process. To this end, we apply a set of operations that detect each gap in the dataset. The data preparation process is described in Algorithm 1. First, line 1 creates an empty vector for indexes of missing values in the dataset. Outliers are identified according to minimum and maximum values, for particular sensors, that can be either application-dependent or predefined by the user. If a data value is out of bounds, it is replaced by a missing value indicator such as *NA* (Not Available) (line 2), so that the correct value can be efficiently estimated in the recovering cycle. Missing values can occur for different reasons, like system or sensor failures. Once the system/sensor is recovered, the next received data point is stored right after the last generated timestamp. Therefore, to identify a gap, it is necessary to check timestamps. We propose a solution where the monitoring component receives data and stores either corresponding data value or *NA* (a missing value indicator) for each created timestamp (lines 4-13). Counters i and j (line 3) count data from input and prepared D_{fr}, respectively. If timestamps from D_{fr} and D_{in} match (line 5), the data point is moved to D_{fr} beside the corresponding timestamp (line 6). Otherwise, *NA* is stored (line 9), and the index of a missing data point is moved to the created vector ω (line 10). Once the *while* loop terminates, the vector ω contains all indexes of missing data, the amount of which is placed in the variable *nom* (line 14).

Algorithm 1 `DataPreparation`

```
Input: $D_{in}[timestamp, value]$, $D_{fr}[timestamp, \_]$
Output: vector $\omega$
 1: Create vector $\omega$
 2: Replace all outliers by a missing value indicator $NA$ (based on thresholds)
 3: $i \leftarrow 1$; $j \leftarrow 1$
 4: while $i \leq length(D_{in})$ do
 5:     if $D_{fr}[j, 1] = D_{in}[i, 1]$ then
 6:         $D_{fr}[j, 2] \leftarrow D_{in}[i, 2]$
 7:         $i \leftarrow i + 1$; $j \leftarrow j + 1$
 8:     else
 9:         $D_{fr}[j, 2] \leftarrow NA$
10:         Add index $j$ in vector $\omega$
11:         $j \leftarrow j + 1$
12:     end if
13: end while
14: $nom \leftarrow length(\omega)$
```

Gap identification

The gap identification phase (see Algorithm 2) is responsible for detecting multiple gaps in a given dataset. It identifies the limits of each gap, using this information for the recovery process. Each gap is processed separately, to enable the selection of an appropriate forecasting technique based on characteristics of previous data or user predefined specifications. The counter i (line 1) is used to iterate over the vector ω, while data index of the first missing value is copied to the beginning of the vector $\hat{\omega}$ and stored also in the temporary variable t (lines 2-3). As long as there are missing values in *nom* (line 4), the counter i looks for the next index of missing value, while the index stored in the variable t is incremented by 1 (line 5). It allows checking whether a gap of consecutive missing values exists (line 6). If indexes are consecutive, the corresponding index is copied to the vector $\hat{\omega}$ (line 7). Otherwise, all missing values from the current gap are detected (line 8), and the vector ω is updated in line 9.

Data processor

This component performs the extrapolation of data characteristics and parameters needed for the utilization of particular forecasting methods. Necessary characteristics are obtained during the analysis of available data preceding the first missing index of the current gap,

Algorithm 2 `GapIdentification`

```
Input: Vector $\omega$, variable $nom$
1: $i \leftarrow 1$
2: $\hat{\omega}[i] \leftarrow \omega[i]$        ▷ Create vector $\hat{\omega}[]$ storing missing indexes of current gap
3: $t \leftarrow \omega[i]$        ▷ Create temporary variable $t$ and store first missing index
4: while $nom > i$ do
5:     $i \leftarrow i + 1$; $t \leftarrow t + 1$
6:     if $t = \omega[i]$ then
7:         $\hat{\omega}[i] \leftarrow \omega[i]$
8:     else
9:         Remove indexes of $\hat{\omega}$ from $\omega$
10:        break;
11:    end if
12: end while
```

that is identified in the previous component. To efficiently forecast NA marked gaps, predecessor data are analyzed to derive parameters necessary to the forecasting process. Parameter selection depends on the forecasting technique. Semi-automatic mechanism allows two scenarios: (i) *single-technique recovery* (STR) and (ii) *multiple-technique recovery* (MTR). In the first scenario, a single technique, that can be specified by users, is used to recover all gaps. In the latter scenario, a technique is selected for different gaps. Currently, we assume that techniques are predefined by users in the algorithm repository.

Forecasting process

In this component, a forecasting technique is selected from the repository and applied to the current gap. Figure 3.3 shows the adaptive recovery process including the results of the aforementioned components. After corresponding missing indexes are stored by the preparation component and the first gap identified by the gap identification component, the data processor component analyzes predecessor data before the gap. The selected forecasting technique is then applied for the recovering process.

Figure 3.3 shows our approach, where forecasting process component applies different techniques (*t1*, *t2*, and *t3*) for different gaps. The choice of suitable techniques depends on data characteristics and forecast objectives as described in [111]. For example, we can select different techniques according to different dataset characteristics: (i) the Autoregressive Integrated Moving Average (ARIMA) method can be used if data contain stationary characteristics, such as trend stationarity, that can be explored by methods proposed in [112]; (ii) the Exponential Smoothing method (ETS), although overlapping in some cases with ARIMA model, can be used for short-term seasonal series or with multiple complex seasonality [113]; (iii) the TBATS forecasting model can be used for long seasonal periods.

If seasonality occurs in time series, by checking periodicity, the data processor component

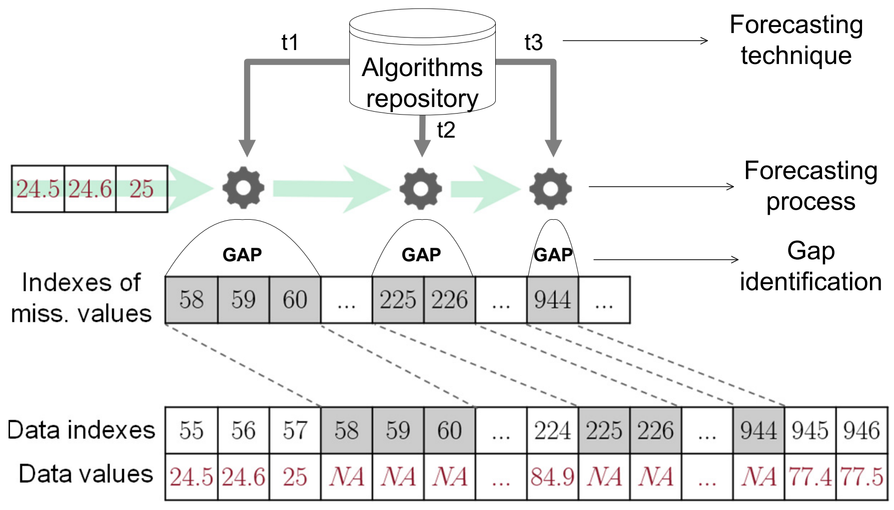


Figure 3.3: Forecasting process of adaptive data recovery for multiple gaps.

can forward that information to the next component. Users can also specify additional information about the data, such as a monitoring frequency, e.g., if temperature data are collected every five minutes, then the seasonal parameter value 288, representing the expected daily seasonality ($12 \cdot 24$), is included in the forecasting procedure. Once all necessary parameters are forwarded from the data processor, the forecasting process can start. Missing values are replaced in the original dataset, and their indexes are removed from the vector ω. Once the current gap is recovered, the next gap (if exists) is considered in a new cycle. The recovering process stops when no more missing values are left in *nom* variable.

Algorithm complexity

By looking at the *while* loop in line 4 of Algorithm 1, we see it iterates over available dataset making it *O(n)*, where n represents a number of data points in the acquired dataset. Further, entering the *while* loop of Algorithm 2 (line 4), it iterates the vector of indexes of missing values that are always less than the amount of data in array S_d. The other lines require $O(1)$. In case the forecasting process uses automatic ARIMA method, the time complexity is $O(n^2)$, where n is the size of data, resulting in the overall complexity $O(n^2)$.

Running time is affected by different factors such as the size of the gap that has to be recovered, the amount of available historical data (finding an optimal trade-off between the gap size and necessary amount of historical data is given by the mediator component in Section 3.4) or seasonal complexity of time series. Since the proposed mechanism targets resource-limited edge nodes and analysis for near real-time decisions, we expect that the input size and dimensionality of incompleteness will not cause a violation of latency requirements.

3.3 Efficient Edge Storage Management

In this section, we focus on the edge storage management process, describing the adaptive algorithm and its interaction between specification list and limited edge storage. We first introduce edge storage management phases and principles, followed by a detailed description of the proposed adaptive algorithm. We also provide an analysis of the algorithms' complexity. We evaluate the applicability of our approach in the experimental scenario by utilizing different real-world datasets, as presented in the evaluation Chapter 6 in Section 6.1.3.

Storing constantly-produced sensor data on the edge can result in several problems due to the storage limitations on the edge nodes [32]. As decision-making processes rely on real-time analytics requiring historical data to perform accurate predictions, storage efficient real-time analytics becomes an important issue on the edge [31].

Storage management problem has been discussed by works like [105, 106] but they consider neither limitation of storage capacity on the edge nor accuracy of near real-time analytics for sensitive edge systems. Due to the always increasing use of data exchanged by IoT devices [114] and the growing tendency of using edge nodes for performing real-time analytics on them, proposing a way to reduce the amount of data stored on edge nodes without affecting forecast accuracy can bring substantial benefits in this context.

3.3.1 Edge Storage Mechanism Design

The effectiveness of limited edge storage nodes depends on the ability to determine the amount of necessary data to perform accurate near real-time decisions. Hence, an edge node should keep only relevant data for local data analytics, discarding or transferring the rest if they are irrelevant. Based on the architectural model (see Figure 3.1), we describe edge storage management phases, as shown in Figure 3.4, namely:

Learning phase. This phase derives information about data, such as time series pattern recognition, used to determine the most appropriate method for that specific pattern [115], or seasonality over a certain period, used to set up a forecast method [116]. This phase is executed only once and provides information used by all the other phases.

Validation of the specification list. This phase checks the user-defined specification list. During the execution of the proposed algorithm, users can update the specification list anytime, e.g., setting forecast accuracy threshold, a new forecast horizon, or different forecast methods. This list has to be checked each time a cycle starts since any changes made to it can affect the whole edge storage management.

Multiple forecast iteration on the available dataset. This phase takes one of the forecasting methods (in our case ETS or ARIMA) with accuracy threshold (f_{th}^{ac}) and forecast horizon (f_h) from the specification list. The available dataset is divided into training and test data. Test data are equal to the number of data points specified by the user in the specification list (i.e., the forecast horizon f_h). The amount of training data is reduced in each iteration by a certain amount of data to find parts of the dataset resulting in

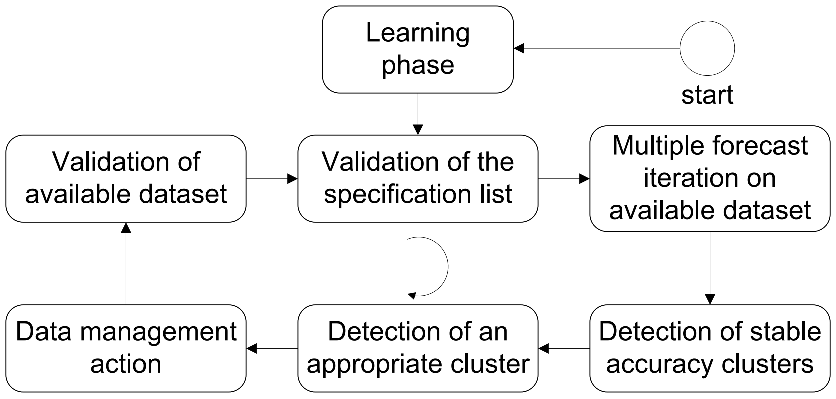

Learning phase
start
Validation of available dataset
Validation of the specification list
Multiple forecast iteration on available dataset
Data management action
Detection of an appropriate cluster
Detection of stable accuracy clusters

Figure 3.4: Edge storage management flowchart and design principles.

required forecast accuracy. At the end of each iteration, forecast accuracy measures are added in the vector γ to be used in the next phase.

Detection of stable accuracy clusters. Here, stable clusters of accuracy values have to be found in the vector γ.

Definition 3. *We define a stable cluster CL_{st} as a set of subsequent data points in the vector γ whose standard deviation for contained values is less than a given percentage CL_{pct} of the standard deviation of entire vector γ, that is,*

$$CL_{st} \subset \gamma \quad AND \quad sd(CL_{st}) < CL_{pct} * (sd(\gamma)) \tag{3.1}$$

We define that a cluster contains at least three members. To provide reliable information regarding future system behaviors, our predictions must be stable. When the multiple forecast iteration process is finished, cluster detection is applied on the vector, which consists of measuring forecast accuracy from each of forecast iterations. The method finds stable clusters of forecast accuracies close to the threshold defined in the specification list (detailed in Section 3.3.1).

Detection of an appropriate cluster. The previous step can return more than one stable cluster, therefore we define how to select the most appropriate one. Stable clusters can differ in mean value and the amount of used data. Therefore, selection priorities have to be set. We propose the twofold priority for cluster selection and a corresponding algorithm. The priority is to satisfy the user-specified threshold. Formally:

Definition 4. *A stable cluster $\mathcal{C}[i]$ of forecast accuracies is appropriate if its mean value ($\mathcal{C}[i]^{m\text{—}v}$) is the closest to the accuracy threshold f_{th}^{ac} from the specification list, namely,*

$$CL_{ap} = \underset{\mathcal{C}[i]}{\arg\min} \left(|\mathcal{C}[i]^{m\text{—}v} - f_{th}^{ac}| \right) \tag{3.2}$$

We select, as appropriate cluster CL_{ap}, the one with minimum absolute difference between f_{th}^{ac} and $\mathcal{C}[i]^{m\text{—}v}$. If there are clusters with higher accuracy, we select the one using less data, regarding second priority (detailed in Section 3.3.1).

Data management action. This step releases irrelevant data from the edge storage. According to the appropriate cluster, we define that the central data index of this cluster indicates a border between relevant and irrelevant data. There are three possible cases, namely: (i) we can reach an appropriate cluster among stable clusters respecting the desired accuracy threshold with fewer amounts of data. In this case, all data not needed to obtain the observed accuracy cluster are released; (ii) none of the resulting stable clusters meets the specified accuracy threshold by the client. In this case, data management action will select the one with fewer data points; (iii) forecast accuracy of stable clusters is higher with an increased amount of training data, e.g., forecast based on all available data from the storage. In that case, the mediator component can retrieve more data from the cloud repository, if available.

Validation of available dataset. The adaptive algorithm is continuously repeated and in each cycle checks storage for newly collected data. Depending on the application and data generation rate, the next cycle of edge storage management can act as a triggered event. In the next cycle, data received from the recovery mechanism and, potentially, from the mediator component, are included. The relevance of old data can be lower unless the prediction accuracy level shows that some stable clusters occur based on these data. In that case, if algorithm feedback shows that accuracy increases with historical data and the amount of currently stored data exceeds edge storage limitations, this data processing can be performed in the cloud. Hence, the approach requires monitoring accuracy rate variations for performed forecasts.

According to the aforementioned phases, we developed an algorithm that is capable of reducing the amount of stored data while keeping required forecast accuracy for predictive analytics. All phases are performed in three main algorithms: the *DetectionOf-StableClusters* (Algorithm 3), the *DetectionOfTheAppropriateCluster* (Algorithm 4) and the *AdaptiveAlgorithm* (Algorithm 5). They are detailed in the following sections.

Detection of Stable Clusters

Detection of smooth behaviors for consecutive forecast accuracies, calculated in the forecast iteration phase, presents the cornerstone of our algorithm. There are many clustering techniques [117] such as partitioning, hierarchical or density-based, but they are not suitable for our case, because often they require the specification of a certain number of clusters beforehand as well as separating the entire dataset based on similarity. Our case requires a dynamic approach that discovers as few clusters as possible based on Definition 3, and considering only corresponding parts of the entire dataset. The process consists of three steps. **First**, we calculate the overall standard deviation for all forecast accuracies and mark it as a baseline. **Second**, forecast accuracies are grouped into clusters of fixed length and the standard deviation is calculated per cluster. **Third**, obtained deviations are compared to the baseline considering the previously calculated threshold. Consequently, stable clusters show where the forecast accuracies are stable.

Pseudo-code for detecting stable accuracy clusters is presented in Algorithm 3. The method requires vector γ consisting of forecast accuracy measures (MAPA) from the forecast iteration process and scaling factor s_f that is used for the threshold calculation. The threshold CL_{th} differs among different datasets, because each measurement has its scale of values with unpredicted volatility. Based on experiments, by default, the scaling factor is always equal to 5 in the first attempt of stable clusters detection. This means that only stable clusters with CL_{pct} equal to 20% (see Definition 3) of the baseline standard deviation will be selected. Still, even with a fixed threshold, it is possible to have no clusters. In case it is impossible to meet any stable clusters for the specified threshold, i.e., since forecast accuracies show greater dispersion, the threshold is increased and the process is repeated. For example, by decreasing the scaling factor from 5 to 4, the CL_{pct} becomes 25% of the baseline for detecting stable clusters, by setting in Algorithm 5.

Algorithm 3 `DetectionOfStableClusters`

```
Input: Vector of iteration results $\gamma$, scaling factor $s_f$
Output: Matrix $\mathcal{C}$
 1: $v_\gamma \leftarrow sd(\gamma)$        ▷ Calculate standard deviation (volatility) of entire vector $\gamma$
 2: $CL_{th} \leftarrow \frac{v_\gamma}{s_f}$        ▷ Calculate threshold $CL_{th}$
 3: Create vector $\Delta_v^\gamma$ storing STDs of sliding windows (length 3) on vector $\gamma$
 4: $i \leftarrow 1; j \leftarrow 1$        ▷ Initialize counters $i$ and $j$
 5: Create matrix $\mathcal{C}$ forming mean value, start and end index of stable clusters
 6: while $i < length(\Delta_v^\gamma)$ do
 7:     if $\Delta_v^\gamma[i] < CL_{th}$ then        ▷ Satisfying conditions in Equation 3.1
 8:         Add cluster in $\mathcal{C}$ such that
 9:         $\mathcal{C}[j,1] \leftarrow$ mean value of corresponding range in $\gamma$
10:         $\mathcal{C}[j,2] \leftarrow$ start data index of corresponding range
11:         $\mathcal{C}[j,3] \leftarrow$ end date index of corresponding range
12:         $i \leftarrow i + 1$        ▷ Incrementing $i$ to check next potential cluster member
13:         if $\Delta_v^\gamma < CL_{th}$ then        ▷ Satisfying conditions in Equation 3.1
14:             while $\Delta_v^\gamma[i] < CL_{th}$ do
15:                 Update mean value $\mathcal{C}[j,1]$
16:                 Update end index $\mathcal{C}[j,3]$
17:                 $i \leftarrow i + 1$
18:             end while
19:             $j \leftarrow j + 1$
20:         else
21:             $i \leftarrow i + 1$; $j \leftarrow j + 1$
22:         end if
23:     else
24:         $i \leftarrow i + 1$
25:     end if
26: end while
27: Return $\mathcal{C}$
```

In line 1, the algorithm calculates the standard deviation of the entire vector γ and divides the result (line 2) by scaling factor s_f to set a threshold CL_{th} for finding clusters. In line 3, standard deviations will be calculated for each grouped iteration results in a sliding window in vector γ and then stored to vector Δ_v^γ. Before searching for stable clusters, the algorithm initializes two counters and creates an empty matrix in lines 4-5, i.e., counter i will count clusters in Δ_v^γ and counter j will denote recognized stable clusters in matrix $\mathcal{C}$ including attributes: mean value of forecast accuracies and corresponding ranges of cluster indexes. To detect stable clusters, the algorithm starts from the beginning of Δ_v^γ (line 6) and checks if a standard deviation of the first cluster is below threshold CL_{th} (line 7). If a cluster is recognized as stable, corresponding data will be added to $\mathcal{C}$ (lines 8-11). In some cases, stable clusters can be wider, so it is necessary to check the neighbor cluster (line 12), and if the new one is recognized as stable (line 13), it will continue to check other neighbors (line 14). For each next cluster recognized as stable, the existing cluster is extended updating its corresponding mean value and end index (lines 15-16) and checks the next one (line 17). When there are no more stable clusters in a row, a place for a new stable cluster is prepared increasing counter j in line 19. In case the next cluster is not recognized as stable (line 20), the algorithm will simply close the existing cluster and check the next one (line 21). For each non-stable cluster (line 23), the algorithm will increment counter i (line 24) and loop back to line 6. Finally, the matrix $\mathcal{C}$, whose rows represent stable clusters with the attributes, is returned.

Detection of the Appropriate Cluster

The cluster selection process is described in Algorithm 4. It starts by checking the number of stable clusters. The *else* branch in line 11 is executed only if one stable cluster is recognized and it will become the appropriate cluster (line 12), otherwise, the algorithm will find the appropriate cluster (lines 2-10). Considering the priority, the appropriate cluster becomes the one that is closest to the specified accuracy threshold (line 2). Further, all stable clusters (line 3) that have better accuracy than selected CL_{ap}, i.e., higher mean value, and whose start index begins after end index of selected CL_{ap} (line 4), become potential appropriate clusters (line 5). If there are such clusters (lines 8-10), the one including less data, i.e., which has the lowest start index (line 9), is selected as a new appropriate cluster CL_{ap}. Finally, CL_{ap} is returned in line 14.

3.3.2 Adaptive Algorithm for Edge Storage

The adaptive algorithm integrates all design principles shown in Figure 3.4, and includes calls on described Algorithms 3 and 4. Algorithm 5 requires a forecast horizon f_h and accuracy threshold f_{th}^{ac} that are specified in the specification list, and array S_d that denotes data available in storage. As shown in Figure 3.4, the learning phase helps to select and set up the appropriate method. One of the possibilities is to find periodicity as a necessity to determine the seasonality and thereby to make a better forecast, as described in [118].

Algorithm 4 `DetectionOfTheAppropriateCluster`

```
Input: Matrix $\mathcal{C}$, accuracy threshold $f_{th}^{ac}$
Output: Appropriate cluster $CL_{ap}$
 1: if $\mathcal{C}$ has more than 1 cluster then
 2:     Compute $CL_{ap}$ using Equation 3.2
 3:     for each cluster $i \in \mathcal{C}$ do
 4:         if $\mathcal{C}[i]^{m\_v} > CL_{ap}^{m\_v}$ AND $\mathcal{C}[i]^{start\_index} > CL_{ap}^{end\_index}$ then
 5:             Add $\mathcal{C}[i]$ to temporary matrix $\mathcal{A}$
 6:         end if
 7:     end for
 8:     if $\mathcal{A}$ is not empty then
 9:         $CL_{ap} \leftarrow \mathcal{A}_i$ with minimum starting index
10:     end if
11: else
12:     $CL_{ap} \leftarrow \mathcal{C}[0]$
13: end if
14: Return $CL_{ap}$
```

Algorithm 5 `AdaptiveAlgorithm`

```
Input: forecast horizon $f_h$, accuracy threshold $f_{th}^{ac}$, storage data $S_d$
 1: Calculate $d_f$ using Equations 3.4 and 3.5
 2: while $length(S_d) > 2 * f_h$ do
 3:     Perform method $(S_d, f_h)$
 4:     Calculate MAPA (See Equation 6.1)
 5:     Add MAPA to vector $\gamma$
 6:     $S_d \leftarrow S_d$ decreased by $d_f$
 7: end while
 8: $s_f \leftarrow 5$                    ▷ Set threshold on 20% of overall standard dev., i.e., $\frac{1}{5}$)
 9: $\mathcal{C} \leftarrow$ DetectionOfStableClusters$(\gamma, s_f)$
10: while $\mathcal{C}$ is empty do
11:     $s_f \leftarrow s_f - 1$                              ▷ Decrease scaling factor $s_f$
12:     $\mathcal{C} \leftarrow$ DetectionOfStableClusters$(\gamma, s_f)$
13: end while
14: $CL_{ap} \leftarrow$ DetectionOfTheAppropriateCluster$(\mathcal{C}, f_{th}^{ac})$        ▷ Find $CL_{ap}$
15: Release data from $S$ in range between the oldest and the central index of the
    appropriate cluster $CL_{ap}$, retaining only relevant data in the storage.
```

At the beginning of the algorithm, the decrement factor d_f is calculated utilizing Equations 3.4 and 3.5. The calculated d_f will be decreasing storage data S_d in the multiple forecast iteration phase. Forecast iterations (lines 2-7) will continue until the amount of data in S_d becomes less than the two lengths of a forecast horizon (more details in Section 3.3.2). Appropriate forecast method uses storage data S_d and other attributes (e.g., periodicity), to make prediction for defined forecast horizon f_h and calculates Mean Absolute Percentage Accuracy (MAPA) (see Equation 6.1) in lines 3-4. At the end of each iteration, the MAPA is stored in vector γ (line 5), and a certain amount of old data is removed (line 6) based on decrement factor d_f. Next, for the detection of stable accuracy clusters phase (lines 9-13), scaling factor s_f is set to number 5 representing the impact of 20% in determining the threshold for finding stable clusters in Algorithm 4. If any stable cluster is recognized, the matrix $\mathcal{C}$ gets corresponding information (line 9): mean value, start and end index of the cluster. Otherwise, if stable clusters cannot be found (line 10), the algorithm will decrease the s_f and keep looking for the clusters (lines 11-12). Line 14 finds the appropriate cluster CL_{ap}. Finally, data in the array S_d are released in the range between the oldest index and the central index of the appropriate cluster CL_{ap} (line 15). The adaptive algorithm repeats itself based on demands in the specification list.

Optimal parameter settings

Our goal is to set up necessary parameters enabling continuous operation of the edge storage management mechanism. To allow proper performance of proposed algorithms, storage must always contain enough data for training, plus additional test data (amount of which is equal to defined forecast horizon), and there must be enough number of forecast iterations in each cycle to find stable clusters. This is ensured by keeping training data as:

$$T = 2 * f_h + k, \quad k \geq 3 \tag{3.3}$$

where T denotes the amount of training data points, f_h denotes a forecast horizon that is application dependent and given in the specification list (see Figure 3.1), and k is a natural number. Based on Equation 3.3, there will always be at least 4 forecast iterations resulting in 4 MAPA measures as a basis for finding at least one stable cluster (see Definition 3.2).

Also, running time depends on the number of multiple forecast iterations, which is affected by the decrement factor d_f, calculated using Equations 3.4 and 3.5:

$$\psi = round(df_{pct} * (T - 2 * f_h)) \tag{3.4}$$

$$d_f = \begin{cases} \psi, & \text{if } \psi >= 1. \\ 1, & \text{otherwise,} \end{cases} \tag{3.5}$$

where *round()* is a function that rounds the result half away from zero to integer and df_{pct} is decrement factor percentage. In order to set optimal value for df_{pct}, we performed experiments using all possible ranges of df_{pct} on different datasets. The evaluation is done on 144 data points, representing data collected every 5min over 12h with 1h forecast horizon, satisfying Equation 3.3 to have enough iterations to find stable clusters. Thus, maximum df_{pct} of 30%, which is a representative from a range of 26%-33%, gives the necessary 4 forecast iterations. Results showed that as df_{pct} becomes very low (1%-2%) and very high (8%-30%), the algorithm cannot always find stable clusters in the first run of calculation (Algorithm 5, line 8) while at the same time having a number of clustered

MAPA measures near 100%. Among the rest df_{pct} values, to find a setting that releases more data without significantly decreasing the accuracy of the appropriate cluster, we set df_{pct} at 3% resulting in 34 iterations on average.

Further, we assume that the prediction of data, potentially containing seasonality, requires at least twice as many data points compared to the forecast horizon. This assumption is derived from the constraint that the prediction of one period of seasonal time series requires at least two periods of prior data. Therefore, to calculate the d_f, the training dataset is reduced by two lengths of the f_h.

Algorithm complexity

Considering Algorithm 5, its complexity is $O(n^2)$, where n represents the size of data. ARIMA method (line 4) has $O(n^2)$ complexity and the outer *while* loop (lines 2-7) iterates the entire dataset until n is equal to the two lengths of the specified f_h. In the worst case, it is decreased by 1 at each iteration, leading to a $O(n)$. Further, both Algorithms 3 and 4 have the complexity of $O(n)$. The *while* loop (line 6) from the Algorithm 3 iterates over the size of the vector Δ_v^γ. The inner *while* loop (line 14) uses the same counter as the outer loop resulting in the same $O(n)$ and decreasing the space complexity by not creating new objects in line 8. Algorithm 4 instead iterates over each cluster in the for loop in line 3, whose number is always less than n. Other operations have either a $O(1)$ or a $O(n)$, resulting in a $O(n^2)$ time complexity. Such complexity can be reduced by using less accurate forecasting methods. Even though $O(n^2)$ is not suitable for big datasets, it provides acceptable response time in this context since we target edge storage.

3.4 Mediator Component for Supporting Data Recovery

Storage space limitations on edge nodes prevent keeping all historical data collected from IoT sensors. However, some IoT systems require both local edge and global batch data analytics [119, 23]. Consequently, an edge node can keep only relevant data, and send all acquired data to be integrated into the cloud data repository. Once historical data are available in the cloud, batch analytics can be applied. By using historical data it is possible to calculate the Projection Recovery Map (PRM), which recommends ranges of data for recovering gaps of various lengths, as well as the appropriate recovery technique for each dataset. In this context, the proposed mediator component can either employ cloud-based PRMs to improve recovery of gaps detected by the monitoring component (see Figure 3.1), or transfer data considering local analytics requirements. In both cases the mediator can retrieve necessary data from the cloud, storing them locally when needed. Here, we describe the first case.

Regarding PRMs, recommended ranges of data for certain lengths of detected gaps can be found by slightly modifying the edge storage management mechanism (presented in Algorithm 4). Here, selecting the appropriate cluster CL_{ap_med} means selecting a cluster with the highest accuracy, namely:

$$CL_{ap\_med} = \underset{\mathcal{C}[i]}{\arg\max}\,(\mathcal{C}[i]^{mean\_value}) \tag{3.6}$$

In this case, we employ priority criteria different from Section 3.3.1. Once the CL_{ap_med} is detected, bounds (upper and lower) for the number of used data points and corresponding MAPA measures are stored. This process is repeated with multiple-recovery techniques (MTR case) over consecutive gap lengths and then merged in a PRM for each dataset. In extreme cases where big gap lengths are expected, PRM can be calculated over non-consecutive number of missing values and, if required, interpolate ranges that are not calculated. To check the accuracy of PRM-based MTR, we test it on available historical data (evaluated in Section 6.1.4).

CHAPTER 4

Elastic Edge Data Services for Supporting Decision Making

On top of the proposed edge data management framework in the previous chapter, we provide a thorough analysis of scalability challenges in edge environments. In the first step, we focus on architectural requirements and the design of elastic edge storage and data services. This chapter contributes:

- a detailed analysis of edge storage services with application-specific analytics support, including different utilities and analytics requirements;
- a specification of a set of necessary principles for engineering elastic strategies, leading to highly customized software-defined elastic storage services, suitable for dynamic data workload characterizations at the edge.

In Section 4.1, we describe the importance of elasticity regarding data services at the network edge. In Section 4.2 we provide a motivational scenario on elastic edge storage services. Detailed analysis of strategies for elastic edge storage services and their requirements are provided in Section 4.3, on top of which we propose engineering principles for elastic edge storage in Section 4.4.

4.1 Importance of Elasticity in Edge Data Services

Today, owing to the evolution of sensor and communication technologies, IoT systems are shaping the world. It is possible to make data-driven decisions by monitoring different parameters from "Things" present in, e.g., smart buildings [120], smart grids [121], manufacturing systems [122], or road infrastructure and vehicles [123]. The increased amount of IoT sensor data, leading to these emerging IoT application areas, requires near real-time analytics executed on a set of distributed edge nodes (e.g., lightweight devices, micro data centers, and edge/fog servers) that store and process data. The introduction of edge computing can help to deal with time-sensitive requirements for accurate decisions based on IoT data [34].

However, unlike scalable cloud data repositories, edge systems have limited storage capacities, whereas a certain amount of IoT sensor data have to be stored and processed in the proximity of the data sources [124]. Consequently, any edge data service must store only the most relevant data for edge analytics, whereas non-relevant data either have to be discarded or moved to cloud data centers. But the *relevancy* is determined by analytics contexts: new edge infrastructure conditions and application analytics requirements, regarding explosive growth of IoT data [125], force us to explore novel architectural design and further implementations of edge data services. By investigating edge data services, we consider strategies, methods, mechanisms, and operations for handling and storing constantly generated data at the network edge. We observe that even *within a single edge analytics system*:

- (**O1**) IoT data are categorized into different model types representing multi-model data, in particular near real-time streaming data and log-based data, thus, requiring different storage types and governance policies. They also include different levels of *significance* regarding storage and edge analytics. This is especially the case for critical applications, such as healthcare [126] (e.g., by keeping most important and relevant data close to the source for effective treating of diseases) and smart manufacturing [127] (e.g., considering data significance levels among data streams coming from industrial equipment for maintenance purposes). Hence, *all applications and sensors do not have equal importance*;
- (**O2**) Different IoT sensors include *various errors* such as missing data, outliers, noises, and anomalies, affecting the designs of edge analysis pipelines and corresponding different to decision-making processes. In this context, incomplete and noisy data can be critical, e.g., for traffic-dependent near real-time route guidance [128], but can be tolerated by intelligent weather forecasts [129];
- (**O3**) Data from different IoT sensors appear with *different data generation speeds, consequently producing different data volumes for the same time interval.* Simultaneously, different types of monitored sensors require different data volumes to make meaningful analytics. In systems like smart cities, it is crucial, for example, to have big amount of frequent traffic measurements for managing traffic flow in real-time. On the other hand, due to lower volatility, a weather station requires much fewer data volumes from its sensors for accurate predictions [36].

Currently, all these highlighted issues are solved outside edge storage services. Solutions for these issues are not included in existing designs of edge data services because, as one might argue, such issues are *analytic context-specific.* However, we argue that they are generic enough that can be customized and must be incorporated into the design of new edge data storage systems. These observations indicate crucial changes for enhancing traditional approaches, which have assumptions on consistent low latency, high availability, and centralized storage solutions, that cannot be generalized to the edge storage services and unreliable IoT distributed systems.

4.2 Motivational Use Case

IoT systems produce and rely on huge amounts of sensor-based time series data [51, 105]. In an IoT sensor environment, such as an exemplified university smart building shown in Figure 4.1, we can observe data workloads from different IoT applications and decide whether to (i) push data to the cloud data storage, (ii) keep relevant data for local edge analytics, or (iii) discard data if they are not useful for future analytics. In the first case, traditionally, all data are transferred to resource-rich cloud data centers where storage and compute-intensive workloads can be handled, resulting in necessary control commands for IoT actuators. However, increasing data streams and latency requirements arising from IoT applications make distant cloud data transfer often impractical. Recent solutions for making crucial fast decisions in IoT systems have brought up the second case employing edge nodes.

In an IoT system, such as a university smart building equipped with many sensors measuring internal subsystems, it is obvious that data from HVAC (Heating, Ventilation, and Air Conditioning) sensors do not have the same importance as data from smart meters

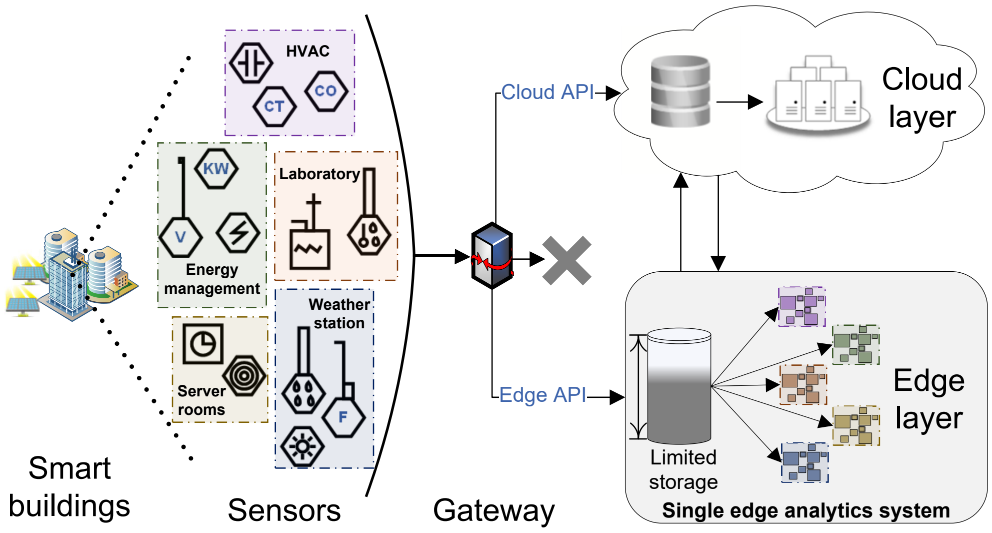


Figure 4.1: Traditional single analytics system for university smart buildings use case. Different shading colors describe sensors that belong to different subsystems and show corresponding data analytics modules in the edge layer.

and solar panels essential for energy management (O1); incomplete data from weather stations can occur due to external conditions while missing data coming from server room sensors can be caused by some internal failures (O2); an energy management subsystem has higher data generation frequency than a laboratory subsystem (O3). Accordingly, each of these subsystems requires a different approach to sensor data analysis, although the same edge storage system is used to integrate data for edge analytics. In addition, limited storage capacities at the network edge prevent us from keeping all generated data. Several proposed approaches [122, 130] established the data communication between cloud and edge nodes without an efficient solution to these complex issues.

In the third case, due to the limited underlying network infrastructure, some data can be filtered or reduced to save bandwidth usage and storage space [105], but impacting later degradation of Quality of Service (QoS) and causing data integrity problems.

Edge deployed systems must maintain multi-model data. Sensor-based time series might conform to a common IoT data model, while it is important to support other data types across different data stores by triggering certain functions, in particular, maintaining logs for state information of IoT devices/equipment, data exchange, and combination between different storage types. Considering present multi-model data, resource-limited edge nodes, and complex data management at the network edge, the next-generation storage service has to deal with many trade-offs to make sure that the edge analytics is always served with the most relevant and suitable data. Edge analytics have to meet certain quality of analytics [131], including amounts of data available, timely decisions, and certain levels of data accuracy. Therefore, developing such reliable and elastic edge storage services is of paramount importance. From an architectural design viewpoint, we must identify which data should be kept at the edge nodes, how long to store them, and which data processing utilities can assist these problems. We provide a detailed analysis of elastic storage services in the following section.

4.3 Analysis of Edge Storage Services

Based on the contemporary research, we view an edge storage system including a set of storage nodes. In a basic model, storage nodes in an edge storage service can interact and transfer data to each other to improve QoS properties for data applications and decision-making processes at the edge. To achieve optimized designs for the storage service management, we analyze requirements from edge data and system characterization (Section 4.3.1), application-specific analytics context (Section 4.3.2), and edging system operations (Section 4.3.3).

4.3.1 Edge Data and System Characterization

To provide an elastic edge storage system, storage nodes have to be aware of *edge data workload characteristics*. Table 4.1 summarizes key characteristics. Incoming data might have quality issues due to the presence of missing values or outliers making collected data

Table 4.1: Examples of data/system characteristics and impact on architectural design.

Characteristics	**Impact on architectural design**
Incomplete data	efficient & adaptive *data recovery* mechanisms;
Storage availability	decentralised *monitoring* components & data-centric *metrics*;
Data incidents	incident *tracing* & *adaptation* mechanisms for data handling modules;
Different data stores	on-demand data *integration* support & load balancing *optimization.*

incomplete. It is important for storage services to decide how to deal with particular incoming data, with the help of metrics such as data quality, data volume, and frequency. These metrics can include different statistic techniques for important insights such as anomaly detection.

The current status of available capacities from storage nodes must always be up-to-date for efficient management of idle capacities. Hence, the storage service should rely on *system characterization metrics.* The combination of different metrics from system and data collection processes represents composite metrics and it can uncover useful failure information of underlying infrastructure, leading to proactive decisions on how to collect and store data.

Having complete end-to-end metrics from data sources to data analytics represents the main building block of the elastic storage system. An efficient incident analysis mechanism can be of great importance for recovery and decision-making. To determine the cause of missing data, the metrics should provide insights into the communication network, sensor status, or storage activities. For improved detection, authors in [132] provided incidents characterization in IoT and points of necessary instrumentation for collecting data for incident analysis.

Example: Data from sensors are collected from smart buildings. Because of the changed sensor generation rate, some sensor failures can result in the form of outliers and missing data. Edge data and system metrics indicate several things to storage services: (i) data quality metrics indicate the persistence of errors and require data recovery before storing data; (ii) data similarity metrics indicate which correlated dataset can be used as a baseline for recovery; (iii) sensor generation rate and incoming data volume metrics reveal sensor failure appearance, leading to necessary adaptation of data frequency; (iv) based on storage availability, re-routing is performed to the storage node with needed capacity.

4.3.2 Application Context and Requirements for Edge Analytics

From an application-specific context, there are various aspects, such as application information, processing utilities, and security, related to edge analytics. Increasing awareness of these aspects can significantly improve overall designs for storage strategies and data management. We highlight such aspects in this section.

Application information

Domain-specific application information can represent direct relation to the storage system. Different types of application contexts have strong influences on the architectural design, as shown in Table 4.2. Based on the internal configuration, storage systems map certain data types to sensors. It is important, from the storage standpoint, to know which data types belong to a certain application and what is the nature of IoT data collection, that is, batch or stream data characteristics. Having information about the nature of data collection enables decisions for performing data processing before or after storing data.

Another challenge for edge storage is how to adapt to unpredictable sensor workloads in cases when data collection is related to physical events or periods. Also, fault tolerance levels can be related to sensor *id*s, in case of significant applications, requiring replication of sensor data across interconnected edge storage nodes to ensure availability. Concerning different data model types, different data stores should be supported, e.g., combining time series and document-oriented/log data. Accordingly, it is also possible to reveal hidden relationships by combining different data models (e.g., sensor logs, documents), but significantly increasing the complexity of data handling and analytics.

Table 4.2: Examples of application contexts and their impact on architectural design.

Application context	**Impact on architectural design**
Data analytics mode (batch, stream)	dynamic storage *configuration policies*, *plugins* for application & edge-cloud *data connectors*;
Fault-tolerance level	*replication* mechanism & *re-routing* strategy;
Data-sensitive applications	*secure* & *verifiable* support for data storage/exchange;
Multi-model data types	runtime customization of *data operations* & *processing utilities.*

Example: Collection and storage of sensor data from IoT devices can differ regarding storage and application-specific analytics. For instance, instead of constantly streaming data from smart building subsystems, IoT sensors can be set to push logs every 30 minutes to an edge storage node. Specifying information about data collection nature in applications, like smart building monitoring and HVAC stream data, can be forwarded to data cleaning or anomaly detection mechanisms. If a certain level of data sensitivity is attached to data from the laboratory, then before storing data, an additional protection level is attached through encryption algorithms, and limited access to stored data is set.

Data processing utilities

Novel edge storage services should be supported by different possibilities of data processing utilities that are nowadays overlooked in storage management research. Such possibilities can include data cleaning techniques, anomaly detection, data normalization, and different data recovering techniques for missing values and outliers that often appear in distributed

IoT systems. Further, providing data approximation methods and codecs can reduce data dimensionality and numerosity while retaining essential features of interest for effective data analytics [133].

Further, predictive maintenance represents another perspective. Performing estimations on future generated data volumes and availability of storage capacities, some proactive actions can be taken to improve storage service at the edge such as (i) deciding when historical data should be discarded or moved to cloud repository; (ii) redirecting/moving sensor data to a storage node with idle resources.

Example: In smart buildings, incomplete data might occur for many reasons, such as monitoring system failures, sensor failures, and external environmental conditions. However, no common techniques can be appropriate for different data sources [51]. In sensor-based time series data, one technique can be suitable for high volatile data values, but not good for non-volatile and stationary data [112].

Verification and auditing support

Incoming sensor data at the edge often contain sensitive information about the monitored system, raising the importance to address data reliability within storage services. Security vulnerabilities of sensitive sensor data are overlooked in today's IoT data exchange and storage posing important challenges for their handling. Another important aspect is to audit flows of data into the storage and related analytics. Thus, future storage service architectures must consider building verifiable and secure storage of large-scale IoT sensor data [134]. Integrating blockchain technology can enable distributed and decentralized storage services on heterogeneous edge nodes (IoT devices, edge gateways, micro data centers). However, increased data generation represents a great barrier for its integration with blockchain technology [135]. Therefore, careful architectural designs for the integration between blockchain and storage must be done in the view that blockchain is only used for selective types of data for verification and auditing purposes.

Example: Smart building systems manage sensitive data, such as laboratory records or renewable energy traces important for smart grids. Integrating blockchain technology in storage services systems can ensure authorized access to data, distribute sensitive data among storage nodes, and trace changes of data.

4.3.3 Edging System and Service Operations

Elasticity operations and constraints

To provide reliable edge data storage, elastic storage service should have predefined operations such as (i) the change in sampling frequency, (ii) data filtering operation, (iii) functions for efficient data integration from multiple sensors, (iv) data sharding operations and (v) data re-routing to interconnected edge storage nodes to avoid storage overhead or network bottlenecks. Further, elasticity operations can be directly dependent on a set of Service Level Objectives (SLOs) for a specific application. Based on a set of objectives, multi-objective optimization algorithms can be applied to define a set of appropriate solutions. The aim of the multi-objective optimization module is to recommend a set of operations for storage services based on system constraints and data characteristics, and thereby maximizing possibilities of data-driven IoT decisions at runtime.

Example: Smart building systems can include a multi-objective optimization algorithm to deal with QoS conditions for response time, storage costs, available resources and data quality, triggering different aforementioned elastic operations.

Data importance

IoT systems specify different levels of importance for different types of monitored sensors. From a storage viewpoint, such sensor types can be coupled with additional services, such as outliers and anomalies detection or data cleaning, before passing the data to the storage or analytics. IoT systems can change the data importance at runtime (context-aware relevance). Some system functionalities become critical in certain periods of year or day/night time, requiring elastic adaptation and support from edge storage services.

Example: Data sources with less importance can utilize elastic operations such as stream filters to omit redundant data. If data are redundant (e.g., temperature measurements in smart building offices do not change often), a dynamic monitoring frequency can be set [71], leading to increased data generation in case of high data volatility, and thus impacting storage services.

4.4 Engineering Principles for Edge Data Services

Based on aforementioned challenges and examples regarding three important aspects of distributed edge storage nodes, namely, edge data/system characterization, application context and edging operations, we present seven principles (P) as guidelines for engineering of elastic edge storage services.

P1: Define and Provide Needed Metrics

To enable efficient customization and adaptation among elements of edge storage systems, it requires a clear definition and flexible monitoring of end-to-end metrics regarding data workloads, application context, and system activities.

How: Figure 4.2 shows end-to-end metrics that can assist in elastic edge storage management. We present end-to-end metrics in 4 stages of the data lifecycle, namely, data collection, data preprocessing, storage service analysis, and data analytics. The storage system must also allow the definition of new metrics at runtime, depending on application-specific requirements.

Tooling: There are many tools for monitoring cloud systems, e.g., Prometheus[1], and Fluentd[2], but few able to monitor edge data metrics. These tools should be equipped with

[1]https://prometheus.io/
[2]https://www.fluentd.org/

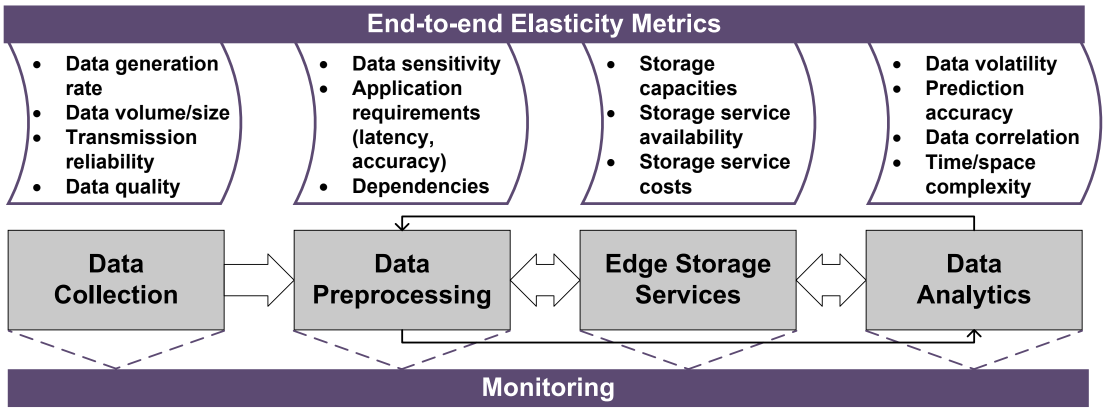


Figure 4.2: End-to-end monitoring metrics of elastic edge services through four data stages. Managing data in each stage can depend on metrics from other stages, e.g., data analytics might depend on metrics collected before data are passed to storage nodes.

additional features including pluggable components for edge systems, such as fluentbit[3], providing AI support and tracing instrumentation, as a promising solution for providing end-to-end metrics for elastic storage services.

P2: Support Application-specific Requirements

Based on sensor-specific metrics and relevancy, we can combine different solutions to deliver appropriate data to local analytics, while meeting application conditions. For example, sensor data have requirements to be clean, complete or normalized before stored and analyzed. Further, customization for secure and verifiable storage is required for applications dealing with sensitive data.

How: As shown in Figure 4.3, depending on application information, different sensor data have corresponding data flow routed through the edge architecture to appropriate edge analytics, namely, descriptive, predictive, or prescriptive. Interconnected storage nodes, with features including *data recovery* and edge *storage management* mechanisms, ensure access to the relevant data at the right time for different purposes. An algorithm repository contains a set of predefined processing utilities, which usage and order are application-specific and dynamically set at runtime in the elasticity management component. In addition, the blockchain integrator component can capture certain types of application-specific data and pass them to the edge blockchain network for verification and auditing.

Tooling: A repository of available and pluggable microservices can speed up the DevOps of storage services by supplying needed utilities. Different microservices can be used to enable elastic activities, such as data cleaning, normalization, and data integration [94]. To keep relevant and complete data in space-limited storage, nodes might incorporate an adaptive algorithm for efficient edge storage management [51] and an automatic mechanism for recovery of incomplete datasets [49].

[3]https://fluentbit.io/

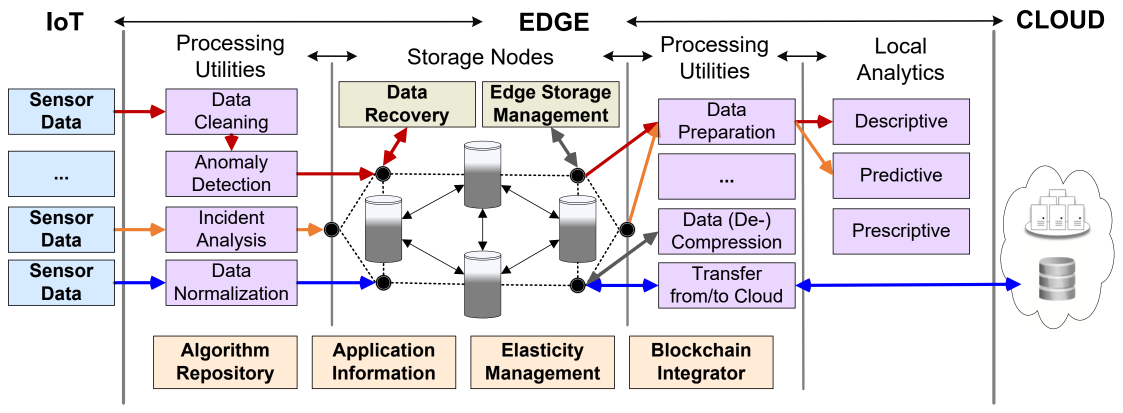


Figure 4.3: Application-specific data flows through a new edge architecture with corresponding components. Different arrows show several scenarios of sensor data flows across edge layer, IoT-edge and edge-cloud continuum, based on supportive modules (yellow) for edge data services, algorithm repository and blockchain integrator.

P3: Enable Adaptive Data Handling

From a software management viewpoint, we need to cope with heterogeneous data workloads including changing data streams, batch transfers, QoS critical requirements. Also, storage services should ensure that stored data are always available, relevant, and complete, i.e., keeping data integrity by utilizing different system and data operations.

How: In this context, critical software technology running on the edge can play an important role in storage resources abstraction, supporting communications, and configuring suitable data handling features and on-demand data transfers. Techniques for auto-switch data handling algorithms/components should be explored (e.g., data reduction [105]).

Tooling: Fogger[4] could be used to support dynamic allocation and contextual location awareness of storage resources in a distributed environment, and featuring blockchain technology. Microservices-based design concepts, such as Edgex[5] open-source platform, might enable decentralized and independent data handling as well as reliable data integration supported by on-demand data services.

P4: Highly Customized System Bundling

Edge storage features should be highly customized and application-aware regarding data and system characterization. Considering data workloads and deployment conditions, traditional inflexibility in software modules bundling can produce over- or under-bundled features for supporting edge application analytics. Thus, flexible storage configurations need to meet deployment situations.

[4]https://fogger.io/
[5]https://www.mainflux.com/

How: Based on application-specific information and internal constraints (e.g., available capacities and performance of resources), the build and deployment process [136] should bundle only components to match these constraints for the right infrastructures. This forces us to develop an optimizer for bundling and deploying different software modules. As shown in Figure 4.3, different utilities and software modules should be available for customized bundling.

Tooling: Existing deploying tools like Docker Compose[6], Ansible[7], and Terraform[8], allow us to bundle and deploy a stack of services but they do not enable needed optimization. This requires us to leverage existing work and develop novel algorithms based on edge node characteristics [137]. Developed algorithms should select application-specific and customized services to build dependent components (e.g., creating a snap on Docker, while selecting components based on application-specific requirements).

P5: Runtime Software-defined Customization

Different inputs, such as application information and data workload characteristics, have to be combined to support runtime customization of elastic operations and data processing utilities. A way of combining these inputs must enable dynamic, software-defined components for the overall system management. A multi-objective optimization mechanism should enable dynamic prioritization of IoT data and condition evaluation from SLOs at runtime and thus would impact provided storage service.

How: Figure 4.4 illustrates potential control flow for elastic storage services. It incorporates a loop for managing internal storage systems initially taking valid application information and current storage system metrics. To evaluate a set of defined objectives, dynamic workload characteristics are combined with static knowledge (e.g., predefined elastic operations and processing utilities). To decide situational trade-offs for data quality and storage capacities, and utilizing edging system operations, we need to derive an optimization strategy for customized storage with core software-defined APIs for data management and service operations.

Tooling: We need to provide approaches of dynamic configuration, runtime code change (e.g., model@runtime [138]), and services mesh, to combine different inputs from distributed storage nodes. The Kinetic Edge SDN[9], could enable efficient load balancing between distributed storage locations. Multi-objective optimization of customized objectives, e.g., data quality and storage capacities, can be well addressed by leveraging proposed strategies to optimize data placement in multi-cloud storage [139].

P6: Support IoT-Edge Continuum

This principle looks at impacting constant data flows between IoT systems and edge storage services, while supporting underlying protocols. According to edge storage

[6]https://docs.docker.com/compose/
[7]https://www.ansible.com/
[8]https://www.terraform.io/
[9]https://www.vapor.io/kinetic-edge/

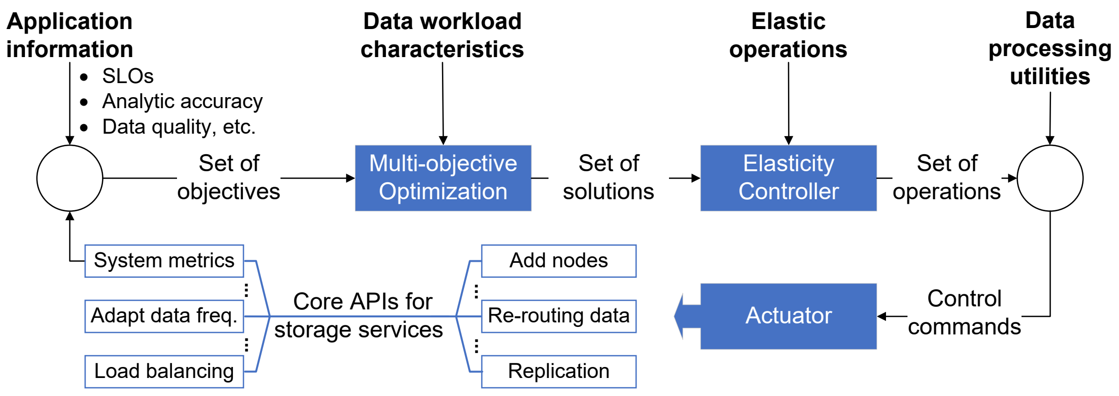


Figure 4.4: Elasticity management determining customized data flows and storage services at runtime in proposed elastic edge storage architecture.

performances, it requires triggering different actions with changing data generation frequency on-demand.

How: Both IoT and edge nodes require developing an edge-IoT connector to control data flows that can often be unpredictable. This connector should be able to (i) discard incoming sensor data (e.g., in the case of bad data quality with outliers or missing values); (ii) apply different sampling commands for collecting only relevant data; (iii) trigger actions for turning off/on sensors in producing data, highly impacting the overall performance of edge storage services.

Tooling: From a data viewpoint, novel mechanisms can be considered for allowing IoT sensors to securely receive and perform actuation requests from edge nodes, while from a programmability viewpoint supporting actuation capabilities for remote IoT device programmability [140] (e.g., building standard actuation APIs). New design patterns for data pipelines should be implemented to control unpredictable data flows and prevent low-quality data.

P7: Support Edge-Cloud Continuum

This principle looks at inter-operation and data transmission between edge and cloud storage systems (see Figure 4.3). Despite the advantages of edge nodes, it is obvious that for many applications, cloud repositories still have to keep large datasets for complex data mining and big data analytics [141]. Thus, we need to support efficient and secure data transfer of large data sets. With an increasing number of data-intensive applications and having bandwidth constraints, it will be crucial to reduce data traffic between edge and cloud. Further, once large datasets are available in the cloud, machine learning models can be trained and then deployed at the edge for better decision-making.

How: For efficient edge-cloud cooperation we need to build an edge connector to the cloud, supporting: (i) operation viewpoint featuring timely techniques for data approximation, (de)compression and encryption/decryption; (ii) network viewpoint featuring mechanism to avoid excessive data traffic through limited network infrastructure; (iii) analytics viewpoint featuring coordination mechanism for consistent ML models employing elasticity and deployment strategies.

Tooling: The approaches of push and pull data on-demand can be investigated for edge-cloud data transfer. Impact of symbolic data representation [133] can be considered as a good starting point to avoid excessive data traffic. There is a need for a model to support secure data migration among multi-location data stores (such as OceanStore [142] utility infrastructure model). Also from an analytics viewpoint, different algorithms and metrics for scheduling and synchronizing ML model updates are required [119].

CHAPTER 5

Deployment of Edge Video Analytics Systems

In this chapter, we show two deployments of edge video analytics systems, namely, (i) increasing traffic safety with real-time edge analytics and 5G, and (ii) data locality-aware edge analytics placement.

First, we show the potential of edge computing and modern communication technologies such as 5G in a practical real-world application, i.e., to increase traffic safety. We propose *InTraSafEd 5G* (Increasing Traffic Safety with Edge and 5G), a system for detecting pedestrians and cyclists in drivers' blind spots at critical traffic intersections and reporting their presence to drivers. Detection of pedestrians is performed by applying an object detection algorithm on video frames streamed from cameras that are attached to edge devices, and integrated on traffic lights. Detection results are delivered in real-time using low-latency 5G communication. *InTraSafEd 5G* is designed to provide audio and visual notifications on the drivers' mobile phones, thereby providing early warnings about critical situations in the driver's path.

Based on the first deployment, in which we discuss and demonstrate an edge analytics prototype that enables real-time execution of critical tasks (independent of input data that are always available locally), we further consider a scenario for self-adaptive placements of edge analytics based on data locality. We aim to efficiently place the critical analytics respecting the location of the user-required input dataset. To ensure efficient placement of on-demand analytics considering data locality and lower analytics execution time, we propose *SEA-LEAP*, a framework for Self-adaptive and Locality-aware Edge Analytics Placement, featuring:

- **a new architecture**, enabling the exploitation of data locality based on the *tracking mechanism* that manages event-triggered registration of dataset movements across different edge nodes;

- **a generic control mechanism**, allowing on-the-fly adaptation and autonomous placement of on-demand analytics to node locations storing required input datasets;
- **a placement optimizer**, enabling on-demand analytics placement to the most appropriate dataset location that minimizes overall request execution time.

We first present the deployment of an edge analytics system for increasing traffic safety in Section 5.1. Then, we present the deployment of self-adaptive and data locality-aware edge analytics placement system in Section 5.2.

5.1 Increasing Traffic Safety with Real-Time Edge Analytics and 5G

The motivational scenario is illustrated and describe including design requirements and properties for the prototype implementation in Section 5.1.1. Section 5.1.2 describes the proposed architecture design with two main parts, namely, computation and communication components, ensuring necessary traffic safety requirements and properties. Technology evaluation of the proposed solution is described in Chapter 6.

5.1.1 Background on Time-Critical Traffic Safety Systems

Ensuring road traffic safety represents an important challenge [143]. According to [144], at least 51300 pedestrians and 19450 cyclists were killed on EU roads between 2010 and 2018, accounting for deaths among pedestrians and cyclists for 29% of all EU road deaths, as reported by the European Transport Safety Council (ETSC). Causes of accidents involve distractions and poor visibility. Early warning systems can improve traffic safety by promptly notifying drivers about critical situations (e.g., pedestrians or cyclists in drivers' blind spots [143]), allowing them to take actions to avoid accidents [145]. However, since drivers' brake reaction time is measured around 1500ms on average [146], early warning systems must detect and send timely notifications respecting strict latency constraints.

Motivational scenario

In the last 10 years, the city of Vienna has experienced around 50000 casualties from traffic accidents, resulting in more than 100 deaths [1]. Many casualties are caused due to blind spots in the drivers' sight or distractions while driving, causing inabilities to brake on time. Figure 5.1 illustrates our motivational scenario. We consider a typical example of a crosswalk situation on the intersections, in which both drivers and users (e.g., pedestrians, cyclists) can not detect potentially threatening situations on time. Such situations can include roadside obstacles and conditions that can reduce traffic safety such as (i) roadside bus stations and buses covering cyclists approaching the intersection, (ii) bushes and trees hiding children and animals that are moving towards the crosswalk

[1] https://www.wien.gv.at/english/administration/statistics/traffic-accidents.html

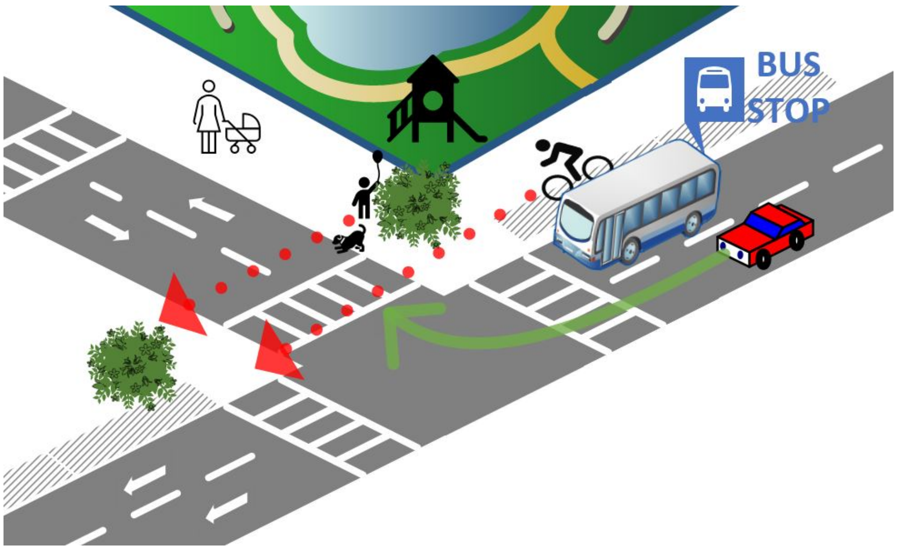


Figure 5.1: An example traffic scenario to illustrate the critical situations when pedestrians and cyclists appear in the driver's blind spots (e.g., behind roadside bus station, bushes).

(e.g., a child running after a dog, out of parents' sight), (iii) bad weather conditions causing poor visibility (e.g., fog, heavy rain). Also, pedestrian-vehicle conflicts can often happen due to unexpected behaviors of pedestrians near crosswalks, such as sudden acceleration and deceleration [147].

To increase traffic safety and support drivers and users on critical intersections to avoid accidents, we consider (i) exploiting modern AI and computer vision techniques to detect critical situations and, (ii) exploring possibilities of edge computing paradigm and emerging 5G network connection. We design our traffic safety solution in the scope of *InTraSafEd 5G* project, funded by the city of Vienna to explore 5G use cases for a better connected smart city.

Design requirements

We identified a specific intersection in the city of Vienna that is considered critical for traffic accidents, due to roadside obstacles and blind spots. Swarco Futurit, a Vienna traffic infrastructure provider, enabled access to empty traffic lights chambers, in which the corresponding edge hardware is installed. We consider the following properties and requirements in the *InTraSafEd 5G* design:

- *Low latency.* Low latency is an important requirement to increase traffic safety, since delivering timely notifications to drivers is important to avoid accidents. Total latency is given by the sum of (i) computation time, (ii) communication (data transfer or message) overhead, (iii) reliability overhead. The sum of these values must respect the drivers' average brake reaction time [146], therefore notification latency must be ranging in the order of tens to hundreds of milliseconds [18].

- *Privacy preservation.* Many edge sensors and devices can collect sensitive data about people (e.g., people's faces). In this context, any network transfer of sensitive data should be minimized or anonymized (e.g., through aggregation, numbers). Compared to traditional scenarios of sending sensitive data over the network to the cloud, we should ensure that no sensitive data are stored or sent over the network [148].
- *Space limitations.* Edge-relevant hardware must be deployed close to critical intersections, respecting urbanistic space constraints. In our setup, the city of Vienna infrastructure provider provided access to empty traffic light chambers for the deployment of edge hardware. Provided traffic lights chambers can be found in most of the traffic infrastructures worldwide [149].
- *Low cost.* Edge hardware (e.g., AI device, camera) has to be installed on many crossroads of a metropolitan area. Therefore, hardware costs must be contained. In our use case, the traffic infrastructure provider of the city of Vienna estimated the costs for a single edge node installation to be in the range of a few hundred euros.

5.1.2 System Design

Figure 5.2 shows an overview of the proposed architecture design, based on the challenges and requirements identified in Section 5.1.1. We illustrate two main components, namely:

Computation component is responsible for the real-time processing of data coming from edge sensing devices. It includes both hardware and software parts. The hardware part includes adapted configurations of edge devices, considering space requirements, and plugged cameras to constantly monitor critical intersections and crosswalks. The software part includes edge processing, which processes video frames using deep learning object detection modules. Object detection workflows are based on a neural network, trained to detect pedestrians and cyclists.

Communication component is responsible for real-time delivery of the edge processing output that is critical for early warnings of drivers. It includes both network and application parts. The network part includes a mechanism to efficiently and timely broadcast such information to a multiple and a dynamic number of vehicle drivers approaching the intersection. The application part includes implemented modules for tracking the vehicle movements as well as the application interface with warning features.

In Figure 5.2, Step 1 shows vulnerable road users (e.g., pedestrians and cyclists) captured in video frames, and collected by a deployed traffic camera. Input video frames are constantly analyzed by the edge processing module in Step 2, searching for target users in the critical crosswalk or intersection area. Once users approaching this monitored area are detected (Step 3), a notification message is generated in Step 4. The message is

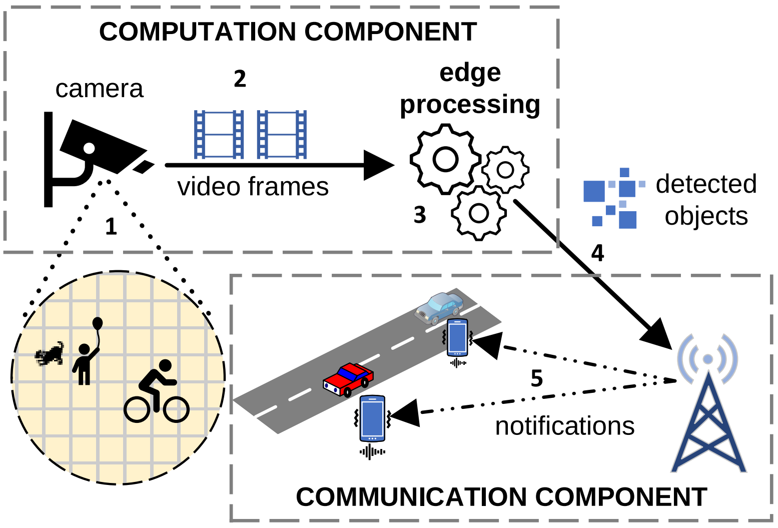


Figure 5.2: An overview of the architecture design for increasing traffic safety with edge.

then forwarded to an app installed on the driver's mobile device, which shows visual and audio notifications (Step 5).

Regarding the privacy property, the video frames, captured by cameras, are analyzed immediately at the edge device, ensuring that sensitive data are neither transferred nor stored over the network. Only the output of data analytics, i.e., a number of objects detected (i.e., pedestrians, cyclists, or animals), is transmitted to drivers' mobile devices. Technology evaluation and design choices for the prototype deployment are shown and discussed in the evaluation Chapter 6 in detail.

5.2 Data Locality-aware Edge Analytics Placement

For the second deployment use case, we propose *SEA-LEAP* framework for self-adaptive and data locality-aware edge analytics placement. We first describe the importance of data locality in Section 5.2.1 and the motivational use case in Section 5.2.2, illustrating a problem of tracking data movements and timely placement of critical on-demand analytics. In Section 5.2.3, we propose *SEA-LEAP* system design, while tracking and control components are detailed in Section 5.2.4 and Section 5.2.5, respectively.

5.2.1 Importance of Data Locality Exploitation in Edge Environments

Modern IoT applications such as public safety video surveillance [150], predictive maintenance in smart manufacturing [151], and traffic management in smart cities [152], are characterized by strict latency requirements. Due to the rapidly increasing number of

IoT sensing devices, data production is growing exponentially [153], with negative effects on the latency of analytics required by IoT applications.

Although edge computing, i.e., moving cloud processing closer to data sources, has been proposed as a solution to address these issues [154], the rapidly growing amount of data produced at the edge affects traditional centralized data collection and processing. Data can be transferred and replicated due to (i) limited storage capacities [50]; (ii) edge failure probabilities [155]; (iii) meeting certain SLOs (e.g., data loss tolerance [156]); (iv) workload balancing [157]. Consequently, data can reside in locations different from where they were initially produced.

Typical examples are edge video analytics applications. Performing video analytics (e.g., object detection to extract specific information of video frames) close to the source of data (e.g., on edge servers such as traffic cameras or micro data centers) is considered as the killer app for edge computing [73]. For example, video analytics on traffic footages of a specific area could be submitted to detect a suspect's vehicle. However, sampled footage frames from a traffic camera system can be stored at locations different from the source node to ensure fault tolerance, affecting the latency of on-demand analytics. This problem is present in other event-driven scenarios, where critical decision-making processes strongly depend on the timely placement of analytics requests, such as finding lost children or pets [150], and failure prevention in smart manufacturing [151]. Therefore, making self-adaptive analytics placement to the most suitable location is an important step toward improving the overall latency of decision-making processes.

In typical placement strategies for data processing applications available in the literature, researchers focus on traditional centralized data collection and analytics solutions [158, 22], or placing data processing based on resource-cost trade-offs [159], but do not discuss critical latency requirements of such analytics applications. Others propose strategies for latency-aware placement configurations of data stream processing applications [160] and low-latency data management on the distributed edge [161]. Still, data movement tracking and locality-awareness for scheduling on-demand analytics across distributed edge infrastructures are currently unsolved problems [162, 163].

5.2.2 Background on Data Locality in Edge Analytics

Motivational scenario

In the context of edge nodes, we consider our *InTraSafEd 5G* project, i.e., Increasing Traffic Safety with Edge and 5G (presented in the previous Section 5.1), as the motivational example described in Figure 5.3. It aims to improve traffic and pedestrian safety through video analytics running on edge nodes, deployed near data sources (e.g., urban area, traffic infrastructure). Once data are collected, different analytics applications can query collected data. Example of these analytics ranges from (i) locating lost children or pets [150], (ii) timely locating suspects, (iii) finding a vehicle suspected of violating rules in captured footages (e.g., by recognizing license plates captured for accessing restricted central business districts, low emission zones), in a smart city. In such on-

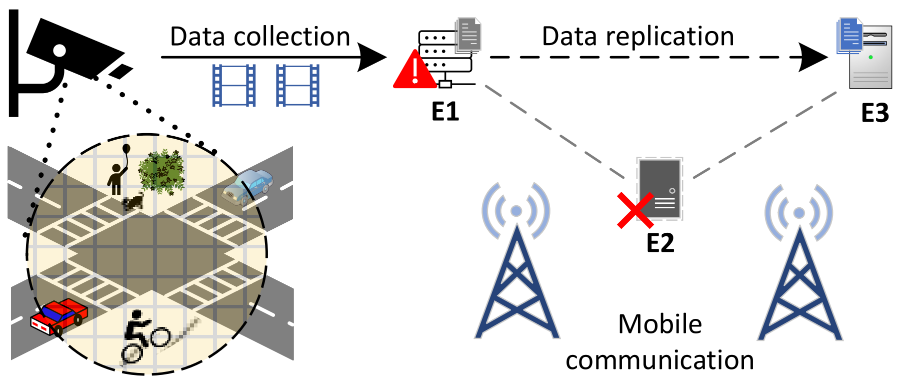


Figure 5.3: An example smart city scenario to illustrate the problems of (i) tracking dataset movements/replications due to high edge failure probabilities, and (ii) timely placement of critical on-demand video analytics.

demand scenarios, critical decision-making processes strongly depend on the strict latency execution of edge analytics requests using specific input datasets.

However, edge servers can fail to execute analytics services for different reasons such as limited resources, power outages, or network failures [155]. Consequently, in the context of edge servers, it becomes necessary to replicate relevant data and services to other nodes (black dashed arrow), to meet SLOs, e.g., service and data availability. Based on calculated failure probabilities, some datasets can be replicated (partially or completely) to different locations to avoid data loss or interruption of running analytics services. To meet the low-latency requirements of on-demand analytics [164] (e.g., finding suspects), they should be placed at the same node where the dataset is stored to reduce the impact of network latency. However, in a geographically distributed edge infrastructure, replication causes the required dataset to be present in location(s) different from where it is produced. Consequently, our challenges are to (i) keep track of datasets, and identify where they are located at a specific time point; (ii) identify which node location is the most suitable to reduce the latency of analytics requests. Inspired by the *InTraSafEd 5G*, we consider its benefits for running on-demand analytics on heterogeneous edge servers such as Raspberry Pi roadside devices with cameras attached to smart traffic lights.

Definition 5. *On-demand data analytics represents data processing applications that are submitted to a computational infrastructure $\mathcal{I}$ in response to specific user requests $\mathcal{R}$.*

The main characteristics of on-demand data analytics are: (i) they refer to specific input datasets [151], and (ii) they have low-latency requirements [164]. In this work, on-demand data analytics are represented as container-based applications running services that process input data and can be placed in different nodes of computational infrastructure (as in [165] and [166]). The input of on-demand analytics is a finite dataset that can be stored in different locations. We focus on a set of sampled video frames generated from video-camera systems.

Locality-aware edge analytics placement

Edge computing is a paradigm where computation is performed on edge nodes, deployed near data sources. Edge computing is the key to exploit *data locality*, i.e., processing data closer to its origin, instead of collecting and processing data far from its source [124]. Data locality is considered of paramount importance to meet low-latency requirements of on-demand analytics [167, 168]. However, identifying the correct dataset location to timely perform processing in the distributed edge environments is a challenging issue due to the possible data transfers and replications. Therefore, we introduce *SEA-LEAP*, a new framework allowing users and developers to easily deploy on-demand analytics applications without knowing the current location of the required datasets. Once required datasets are located by *SEA-LEAP*, on-demand analytics are automatically deployed to the most appropriate edge nodes, allowing analytics requests to meet low-latency requirements and significantly improving decision-making processes.

5.2.3 SEA-LEAP System Design

This section introduces a high-level architecture model overview of the *SEA-LEAP* system, aiming to illustrate the main components and steps for achieving self-adaptive analytics placement at the edge. The design concept of *SEA-LEAP* is driven by the following properties for on-demand edge analytics placement, namely,

- *Data locality-awareness*: as shown in the motivational scenario, data can change its locations over geographically distributed and heterogeneous edge nodes for different reasons, making it challenging to exploit data locality. Therefore, the framework should be able to keep track of dynamic data movements and enable efficient locality-aware data management.
- *Autonomy*: on-demand analytics requests, as shown in the motivational scenario, often have low-latency requirements, making it difficult to timely identify the node minimizing overall request execution time. For this reason, the framework should be able to (i) find the most appropriate node location for analytics placement and (ii) handle numerous requests on time, with little or no human intervention in the deployment process. To this end, we need to enable autonomous analytics placements.
- *Genericity*: the computational edge infrastructure can be heterogeneous regarding both hardware resources and software configurations. Thus, the framework design should be generic and applicable to work on top of existing systems by customizing the logic of proposed components and services, improving the overall reusability.

Overall execution time represents the completion time of the user analytics requests, which includes execution of tracking and control mechanisms to find the location that guarantees the lowest latency. Figure 5.4 provides an overview of the proposed *SEA-LEAP* architecture. We envision the on-demand analytics placement scenario based on data locality. To manage data locality-awareness with autonomous analytics placements, we illustrate three main parts, namely:

Edge sites represent sets of geographically distributed and resource-limited edge nodes capable of executing on-demand analytics on data coming from IoT devices. IoT devices constantly generate data, which are transmitted to the edge infrastructure for temporary storage and future analytics.

Tracking mechanism is a component used for event-triggered registration of datasets and for tracking their future movements. It includes a monitoring service and meta-dataset that stores location-related details about datasets, enabling their dynamic tracking in the distributed edge environments. We focus on datasets with fixed sizes, which are generated, processed, and stored at the edge for future analytics. This is typical in storage-limited edge nodes since in many cases it is enough to have a subset of data to preserve the analytics accuracy [169]. To minimize overall completion time, the control mechanism can trigger adaptive data movements.

Control mechanism is a component performing placement of on-demand analytics. It contains two main sub-components: the *meta scheduler* and the *placement optimizer*. The meta scheduler receives the description of on-demand analytics with the requested dataset name, and communicates with both the meta-dataset and the placement optimizer. The placement optimizer computes the most appropriate location for on-demand analytics to minimize overall execution time. Finally, the meta scheduler performs the actual placement to a target node and commands the analytics execution.

In Step 1, data generated from different IoT sensing devices are transferred to edge nodes, where they can be processed or stored for later analysis. In Step 2, datasets can be moved or replicated to other nodes due to different reasons such as fault tolerance. Any dataset generation or its replication or movement are registered and updated constantly within

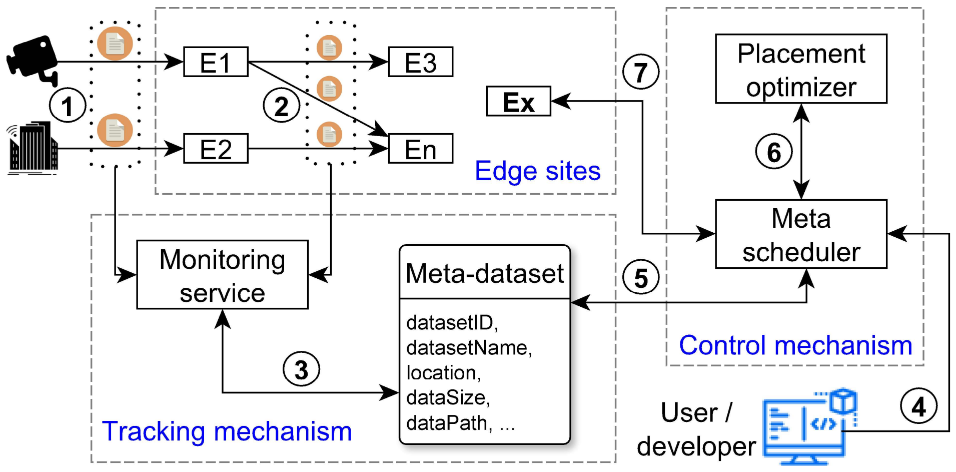


Figure 5.4: SEA-LEAP architecture overview.

the tracking mechanism (Step 3). For each dataset, current location-related details are stored in a database called *meta-dataset*. Meta-dataset can include information such as dataset id, dataset name, corresponding cluster, location path, and data size. Once a user submits the request description (Step 4), the meta-scheduler initiates the automatic placement adaptation. In Step 5, the meta scheduler extracts the required dataset name and retrieves location-related details of the required dataset from the database. We assume that a user knows the target dataset id or name needed as an input for requested analytics. Some of the proposed solutions include (i) access to a list of already generated and existing dataset names (e.g., based on a dataset catalog explained in Section 5.2.4), (ii) a consistent and regulated, easy-to-remember naming of datasets.

Considering that (i) requested dataset can be present in multiple nodes and (ii) different nodes can comply with analytics requirements (e.g., resource capabilities), in Step 6, the meta scheduler queries the placement optimizer to find the most appropriate location for analytics among node location candidates. Finally, the analytics application is placed to the most appropriate node (Step 7). The proposed *SEA-LEAP* follows the service-oriented architecture (SOA), featuring multiple parts and services that can be maintained independently. The following subsections describe all proposed parts in detail, while Table 5.1 lists the main notations used in our approaches.

Table 5.1: SEA-LEAP main notations and definitions.

Notation	**Description**
α	An analytics application that requires input data.
d_{loc}	Variable representing the current dataset location.
d_{name}	Variable representing name of the dataset during the data generation at the data source
dma	A data management action (e.g., replication).
$meta_{db}$	Meta dataset database with location-related info.
KB	A knowledge base containing edge-relevant information.
rcv_{msg}	Variable containing request description for data mgmt.
rcv_f	Description containing the request for analytics placement.
L_{ap}	The most appropriate location for analytics placement.
$\mathcal{L}$	Matrix storing about node candidates for the placement.
$\mathcal{D}$	The set of datasets.
$\mathcal{N}$	The set of locations.
Λ	The set of nodes where dataset d is stored.
$l(n_i, n_j)$	Latency of network connection between n_i and n_j.
$b(n_i, n_j)$	Bandwidth of network connection between n_i and n_j.
$hops(n_i, n_j)$	Number of network hops between n_i and n_j.
$size(d)$	The overall size of a dataset d.
inf_time	An average inference time on a target node.
no_frames	Number of image frames in a target dataset d.

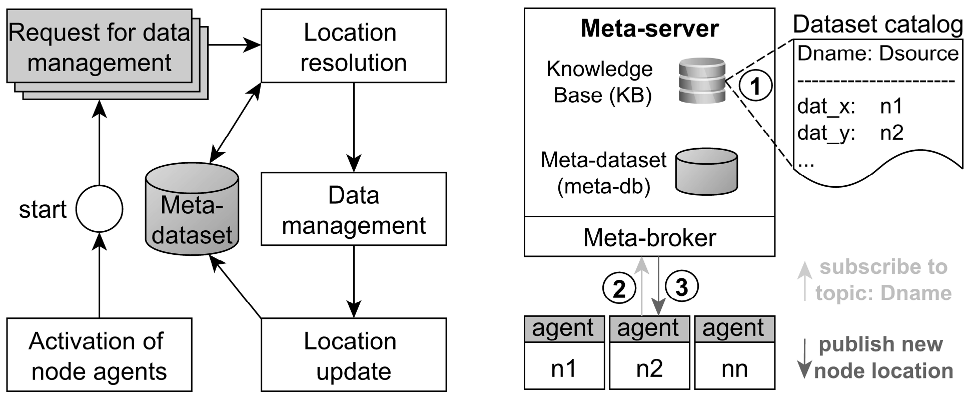
5.2.4 SEA-LEAP Tracking Mechanism

Based on the architectural model, we describe the tracking mechanism that is focused on data management and registration of edge data movements. Figure 5.5 shows *SEA-LEAP* agent-based monitoring service, which incorporates an event-triggered registration of changes of data locations and publish/subscribe-based tracking of data movements, while Algorithm 6 shows pseudocode and the concept behind the agent-based monitoring. Regarding the *data locality-awareness* and *autonomy* properties, an autonomous software agent is employed on top of each node, constantly monitoring and acting upon data management events. Location-related details are stored in the meta-dataset database indicating where the datasets are currently available and accessible. The event-triggered data registration mechanism (Figure 5.5a) consists of several consecutive phases:

Activation of node agents. Every node has an agent listening to a known port (line 1, Algorithm 6). The *while* loop (line 2) serves one client request for data management action. This phase is executed only once on each node and used in all the other phases. The received description of a data management action triggers the following phases.

Request for data management. In this phase, different requests for data management can be initiated (lines 3-4). We define data management as any data manipulation process including (i) generation of a new dataset, (ii) dataset replication or movement. Data management requests employ location-related details about data and can be initiated from (i) meta scheduler (Section 5.2.5), (ii) edge nodes, (iii) edge providers, or (iv) other incorporated mechanisms maintaining edge systems (e.g., load balancing, replication, fault tolerance).

Location resolution. In this phase, based on the target dataset, the location details are either (i) produced for newly generated datasets or (ii) checked in the meta-dataset before further actions. In the first case, a dataset is generated and stored in an edge node.



(a) Event-triggered data registration (b) Pub/sub based data tracking

Figure 5.5: SEA-LEAP monitoring service including (a) registration of changes of data locations and (b) tracking data movements due to data management events.

Algorithm 6 `Agent-basedDatasetRegistration`

```
1: Listening for data management requests
2: while TRUE do                                          ▷ serving one client request
3:     Connect to client
4:     $parse(rcv_{msg})$                                 ▷ analyse received message
5:     $data_{mgmt}(dma, d_{name}, d_{loc})$              ▷ data management applied
6:     Update $meta_{db}$, setting new node location $d_{loc}$ for given $d_{name}$
7:     Disconnect from client
8: end while
```

Details about dataset location are saved in the meta-dataset and partially in the dataset catalog (Figure 5.5b). Dataset catalog (DC), a lightweight key-value store contains pairs of all generated dataset names and initial locations where they are created (Step 1). It supports the submission of the user's analytics request (see Section 5.2.5), and can be accessed from meta-server or keeping a synchronized copy locally. In the latter case (ii), the required dataset already exists and meta-dataset database is queried, if needed, for retrieving location details.

Data management. In this phase, requested data management action (dma) is performed (line 5). Node agents complete data management requests. A trivial example $data_{mgmt}(fetch, dat\_x, n1)$ would fetch dataset dat_x from location $n1$. This phase is designed in a generic way, so it can also be adapted with other data management operations by edge infrastructure providers or deployed edge systems.

Location update. Once the previous data management actions are done, it is required to update new information in the meta-dataset. In this phase, a new connection to the meta database is established and corresponding dataset entries are updated (line 6). Once the database is updated, the source node of the corresponding dataset is notified via the pub/sub (publish/subscribe) channel enabling data movement tracking.

In case of failures, error messages will be returned in each phase. Furthermore, Figure 5.5b shows the pub/sub based tracking of data management. First, as shown in the *location resolution* phase, DC stores existing and unique dataset names (Step 1) included in the Knowledge Base (KB). KB represents prior obtained and edge-relevant details such as edge-specific network topology, node characteristics, and analytics benchmarks. They provide a collection of information necessary for efficient analytics placement (explained further in the evaluation Section 6.3). Edge node locations and produced datasets can be numerous and highly distributed. Thus, regarding the scalability of the tracking mechanism, the monitoring service includes a pub/sub messaging system in which every edge node ($n_1, n_2, ..., n_n$) can be subscribed to topics representing dataset names or *id*s that were initially produced at these nodes (Step 2). Once the dataset location is changed in the meta-dataset during the *location update* phase, the meta-broker publishes the change to a specific topic (Step 3). As a result, each node has information about the current location of its datasets, facilitating further edge data management actions. We assume a pub/sub system, such as MQTT (Message Queuing Telemetry Transport), due to its communication scalability and minimum resource requirements [170].

Note that the tracking mechanism is essential for enabling data locality-awareness, while the following control mechanism primarily ensures the self-adaptive and timely placement of edge analytics based on data locality. The scalability of the tracking mechanism will be investigated in future work.

5.2.5 SEA-LEAP Control Mechanism

The control mechanism is the cornerstone of *SEA-LEAP* architecture. The goal is to enable self-adaptive placement of an analytics application α considering a dataset location d_{loc}, based on two actions, namely,

- *GuideMe:* static placement of an analytics application to the source node candidate initially storing the requested input dataset. If the input dataset is simultaneously present on multiple locations, the node ensuring the lowest estimated analytics execution time is selected as the target one;
- *FollowMe:* dynamic placement of an analytics application to an alternative node candidate that minimizes overall request execution time. In this case, an adaptive dataset movement from the source to the alternative node is necessary, before the analytics placement and execution.

These actions can offer a continuous adaptation of analytics placements in highly distributed and networked edge servers. Both actions rely on two important services of the control mechanism, namely, the *meta-scheduler* and the *placement optimizer*, described in the following sections.

Meta Scheduler

Accessing the dataset locations can be done by storing location details in the meta-dataset (described in Section 5.2.4), while placement adaptations are managed on-the-fly within the meta scheduler. We assume that the meta scheduler is accessible, and there can be multiple instances serving. Algorithm 7 describes the meta scheduler life cycle in detail. The scheduler continuously listens for new requests in lines 1-2. Once a user sends an analytics request description, containing details for application execution, its format is checked. If its format is valid (lines 3-5), the meta scheduler extracts information such as the name of the required dataset *id* and analytics process α (line 6). Next, the meta-dataset is checked and relevant information is retrieved (lines 7-8). If corresponding information exists, the meta scheduler will send details to the placement optimizer (line 9). The output of the placement optimizer represents the node location that guarantees the lowest total latency for the request and it is stored in the L_{ap} (lines 10-11). In case the usage of L_{ap} requires adaptive data movement, the meta scheduler will follow the procedure from Algorithm 6 (lines 12-14). Finally, the meta scheduler performs the placement of α in the node L_{ap} (line 15).

Algorithm 7 `MetaScheduler`

```
1: while TRUE do                                        ▷ loop serving one client's requests
2:     $rcv_f \leftarrow waitConnection()$                         ▷ waiting for incoming connections
3:     if $rcv_f = fmt$ then                           ▷ checking the format of analytics request
4:         continue
5:     end if
6:     $parse(rcv_f)$                        ▷ extracting needed information $(d_{name}, \alpha)$
7:     Create matrix $\Lambda$ with location-related information about $d$
8:     $\Lambda \leftarrow retrieve(d_{name})$                         ▷ retrieving details from $meta_{db}$
9:     if $\Lambda \neq empty$ then
10:        $L_{ap} \leftarrow$ PlacementOptimizer($\Lambda$)
11:        Adapt deployment templates with the $L_{ap}$ and other info
12:        if $L_{ap}$ is not one of the initial location from $\Lambda$ then
13:            Replicate dataset $d$ to $L_{ap}$ using Algorithm 6
14:        end if
15:        deploy($R, L_{ap}$)                        ▷ placing analytics to node location $L_{ap}$
16:    end if
17: end while
```

Placement Optimizer

The goal of the placement optimizer is to find an edge location that minimizes the total latency. It is designed to satisfy users' latency requirements for timely decision-making processes. Total latency is impacted by (i) analytics execution time that depends on node and dataset characteristics, (ii) data transfer that is affected by network characteristics.

We consider analytics placement as a minimization problem with latency as an objective. We assume that users can submit requests for executing data analytics applications over different input datasets, whose location is unknown to the user. Computational infrastructure is modeled as a graph $\mathcal{I} = (\mathcal{N}, \mathcal{E})$ (as used in [171]), such that $\mathcal{N}$ is a set of different nodes where applications and datasets can be placed, and $\mathcal{E}$ models the network connections between nodes. For each $(n_i, n_j) \in \mathcal{E}$, with $n_i, n_j \in \mathcal{N}$, we define both latency $l(n_i, n_j)$ and bandwidth $b(n_i, n_j)$ measurements of network connection (described later in Section 6.3.3).

We also define a set $\mathcal{D}$ of different generated datasets, that can be initially stored in one or multiple nodes $n \in \mathcal{N}$ over a geographical area. In the latter case, we assume that those datasets are always synchronized. Further, each dataset d is defined by its size $size(d)$ and the set $\Lambda(d)$ of nodes where d is stored. Users submit a request $\mathcal{R} = (\alpha, d_{name})$ to $\mathcal{I}$, where α is an analytics application, and d_{name} is the name of the input dataset for α. For each $\mathcal{R}$, *SEA-LEAP* goal is to identify location L_{ap} for α and d that minimizes $\mathcal{R}$

total latency, i.e.,

$$L_{ap} = \arg\min_{\mathcal{L}[i]} (\mathcal{L}[i]_R^{t\_lat}), \ \ L(\alpha) = L(d). \tag{5.1}$$

$\mathcal{L}$ is a matrix that contains location candidates with calculated total latency, including a potential data transfer and the analytics execution time on the specific node candidate:

$$TL = t_{(n_i,n_j)}^{mv_d} + t(R, n_j), \tag{5.2}$$

where n_i initially stores the dataset d ($n_i \in \Lambda(d)$), and n_j is an alternative node candidate ($n_j \in \mathcal{L}(d)$). In case of n_i as the initial candidate, i.e., $n_i = n_j$, then $TL = t(R, n_i)$. Otherwise, we define $t_{(n_i,n_j)}^{mv_d}$ as the time required to send d from n_i to n_j, i.e.,

$$t_{(n_i,n_j)}^{mv_d} = l(n_i, n_j) + hops(n_i, n_j) \cdot \frac{size(d)}{b(n_i, n_j)}, \tag{5.3}$$

where $hops(n_i, n_j)$ is the number of hops between n_i and n_j. Further, we define $t(R, n_j)$ as the estimated time required to complete analytic request $R(\alpha, d)$ on node n_j, i.e.,

$$t(R, n_j) = inf\_time(n_j) \cdot no\_frames(d), \tag{5.4}$$

where $inf\_time(n_j)$ is an average inference time per frame, and $no\_frames(d)$ is the corresponding number of frames in target dataset d (see Section 6.3.3).

Algorithm 8 describes the placement optimizer. First, two data structures are created to store location candidates for analytics placement (lines 1-2): $\mathcal{L}$, containing a total estimated latency, and $\mathcal{L}_{KB}$, which stores potential candidates retrieved from KB if they satisfy certain conditions, e.g., nodes with better inference time compared to the source node(s) (line 3). Next, each node containing the requested dataset (line 4) will initially become a candidate for placing the required analytics (lines 5-7). Then, even if the required dataset is present on multiple nodes, the placement depends on the total latency of each candidate (line 6). In lines 8-12, we calculate total latency for each new candidate, since due to the heterogeneity of the infrastructure it is possible to achieve a lower total latency on a node with more resources, despite the data transfer (line 10). Consequently, the most appropriate node location L_{ap} is the one offering the lowest total latency (line 14), returned in line 15. We consider three scenarios: (i) the dataset is stored on a single node with the lowest estimated analytics latency; (ii) the dataset is stored on multiple and heterogeneous nodes, therefore the most powerful will run the analytics; and (iii) edge node(s) storing the required dataset do not have resource capabilities to meet latency requirements, thus, the dataset will be placed to a node which minimizes overall request execution time.

Theorem 1. *SEA-LEAP complexity is $O(n \cdot m + k)$.*

Proof. SEA-LEAP complexity is determined mainly by two components, `MetaScheduler` and `PlacementOptimizer`. `MetaScheduler` complexity (see Algorithm 7) depends

Algorithm 8 `PlacementOptimizer`

```
Input: Set of nodes storing req. dataset $\Lambda$
Output: The most appropriate location $L_{ap}$
 1: Create matrix $\mathcal{L}$ forming location candidates with est. total latency
 2: Create matrix $\mathcal{L}_{\mathcal{KB}}$ for potential location candidates from KB
 3: $\mathcal{L}_{\mathcal{KB}} \leftarrow KB(n_{type} > \Lambda(n_{type}))$        ▷ retrieving alternative candidates
 4: for each $n_{init} \in \Lambda$ do
 5:     Add node $n_{init}$ to $\mathcal{L}$
 6:     Calculate $TL(n_{init})$ for the initial node using (5.4)
 7:     $\mathcal{L}(n_{init}, TL) \leftarrow TL(n_{init})$
 8:     for each $n_{new} \in \mathcal{L}_{\mathcal{KB}}$ do
 9:         Add node $n_{new}$ to $\mathcal{L}$
10:         Calculate $TL(n_{new})$ for new nodes locations using (5.2), (5.3) and (5.4)
11:         $\mathcal{L}(n_{new}, TL) \leftarrow TL(n_{new})$
12:     end for
13: end for
14: $L_{ap} \leftarrow \mathcal{L}_i$ with minimum total latency        ▷ compute $L_{ap}$ using (5.1)
15: Return $L_{ap}$
```

on `PlacementOptimizer` (see Algorithm 8), since all other lines have complexity $O(1)$. The *for* loop (line 4) from Algorithm 8 iterates over the set of node locations Λ that simultaneously store the requested dataset. Entering the inner *for* loop in line 8, it iterates over each new node location candidate and calculates the estimated total latency in line 10, resulting in complexity of $O(n \cdot m)$, where n is the number of replicas, representing the initial node candidates, and m is the number of alternative node candidates. Next, searching the candidate with the lowest total latency in line 14 has the complexity of $O(k)$, where k is the total number of node candidates. Other lines are $O(1)$, resulting in the overall complexity of $O(n \cdot m + k)$. $\square$

$O(n \cdot m + k)$ is acceptable in our context, since (i) n is expected to be either 1 due to limited edge storage capacities [50] or small while still guaranteeing the resilience with fewer replicas as showed in [155], and (ii) m is limited to both nodes whose types match the types from KB benchmarks and that have lower inference time per frame than initial nodes from the set Λ.

CHAPTER 6

Evaluation

This chapter shows the experimental evaluation of the proposed approaches in the previous chapters. We first evaluate the proposed edge data management framework in Section 6.1 featuring three mechanisms, namely, adaptive data recovery (Section 6.1.2), edge storage management mechanism (Section 6.1.3), and projection recovery maps for supporting an automatic data recovery (Section 6.1.4). In Section 6.2, we evaluate the technology and system components for increasing traffic safety with real-time edge analytics and 5G, including details about design choices of the real-world prototype deployment. In Section 6.3, we show the experimental evaluation of self-adaptive and locality-aware edge analytics placement, including details about testbed configuration, input datasets and target application.

6.1 Edge Data Management Services Evaluation

In Chapter 3, we described a complete edge data management framework (*EDMFrame*), featuring different edge data services. We evaluate *EDMFrame* experimentally using six sensor-based time series datasets from smart buildings and homes. Results show that *EDMFrame*: (i) can automatically recover gaps of various lengths with single-technique recovery and achieving less error employing PRM-based multi-technique recovery (as shown later in Section 6.1.4; and (ii) reduce the amount of data stored adaptive cycles, while keeping high prediction accuracy, therefore storing only data relevant for predictive analytics at the network edge.

The proposed *EDMFrame* is implemented and simulated using R language on a 64-bit Windows 10 PC, configured with a 2.70-2.90 GHz Intel i7-7500U CPU and 16 GB RAM.

6.1.1 Data and Measures

We evaluate our approach on sensor-based time series data, typical in IoT applications like smart buildings and homes [2]. The main characteristics of selected time series datasets are presented in Table 6.1.

- Datasets *h_1-3* contain traces from the Smart* project [172] for optimization of energy consumption in smart homes, obtained by UMass Trace Repository [173]. They represent various environmental and electrical data, such as temperature, relative humidity, wind information, and heat index.
- Datasets *b_1-3* come from the monitoring system of Austria's largest Plus-Energy Office High-Rise Building [174]. These datasets contain various measurements used for automatic HVAC systems (Heating, Ventilation, and Air Conditioning), cooling, energy management (e.g., temperature, indoor air quality, electricity production and consumption).

These traces represent real-world data, whose near real-time analysis can bring valuable hints for IoT actuators, making these datasets appropriate samples for our experiments. We have selected six datasets targeting different characteristics and behavior, to show the applicability of our algorithms.

Further, we use Mean Absolute Percentage Error (MAPE) [175] for forecast accuracy evaluation among different datasets, due to its scale independence. Based on MAPE, we also define Mean Absolute Percentage Accuracy (MAPA) as in Equation 6.1:

$$\texttt{MAPA}(Y, \hat{Y}) = 100 - \texttt{MAPE}(Y, \hat{Y}) \tag{6.1}$$

where $\texttt{MAPE}(Y, \hat{Y})$ is equal to $\frac{100}{n}\sum_{i=1}^{n}|\frac{(y_i - \hat{y}_i)}{y_i}|$, Y and $\hat{Y}$ are respectively the sets of actual values and their forecasts, n is the number of data points, $y_i - \hat{y}_i$ is the forecast error, y_i and $\hat{y}_i$ are respectively the i-th value of y and its forecast.

Table 6.1: Main characteristics of datasets for the EDMFrame experimental evaluation.

Source	Dataset	Sensor type	Range of values	Volatility [SD]
	h_1	heat index [F]	52.11-88.79	8.74
1	h_2	room temp. [F]	68.90-79.70	2.51
	h_3	RH [%]	30.2-92.7	18.52
	b_1	el. meter [kWh]	19.86-19.97	0.04
2	b_2	flow temp. [C]	41.3-48.1	1.32
	b_3	IAQ [ppm]	458.11-707.88	62.53

6.1.2 Adaptive Data Recovery Evaluation

Recovery of multiple gaps simulation 1

We first evaluate the applicability of the proposed approach (see Section 3.2) by recovering multiple gaps in different datasets, utilizing the forecast package [176] in R. We see an example in Figure 6.1. The upper graph shows an incomplete subset of the dataset *h_1* before the recovery, while the lower graph shows a complete dataset after recovery. Gray shaded areas indicate four gaps. In all datasets by source 2, we observe several real gaps in collecting data that can affect the accuracy of data analytics. Therefore, we identify these gaps and artificially make gaps with same sizes in the dataset *h_1* (precisely, gaps with 1 (G_1^1), 2 (G_2^2), 17 (G_3^{17}) and 30 (G_4^{30}) consecutive missing/invalid data values). Then, these gaps are recovered using the proposed mechanism, and forecast accuracy is evaluated using MAPE measure. The black solid line represents the actual behavior of a dataset using data points connected by trend lines. The black dashed line shows actual data for corresponding gaps, while the red solid line represents predicted values of missing/invalid data, calculated using forecasting techniques.

In this example, multiple gaps are automatically recovered using ARIMA, representing the

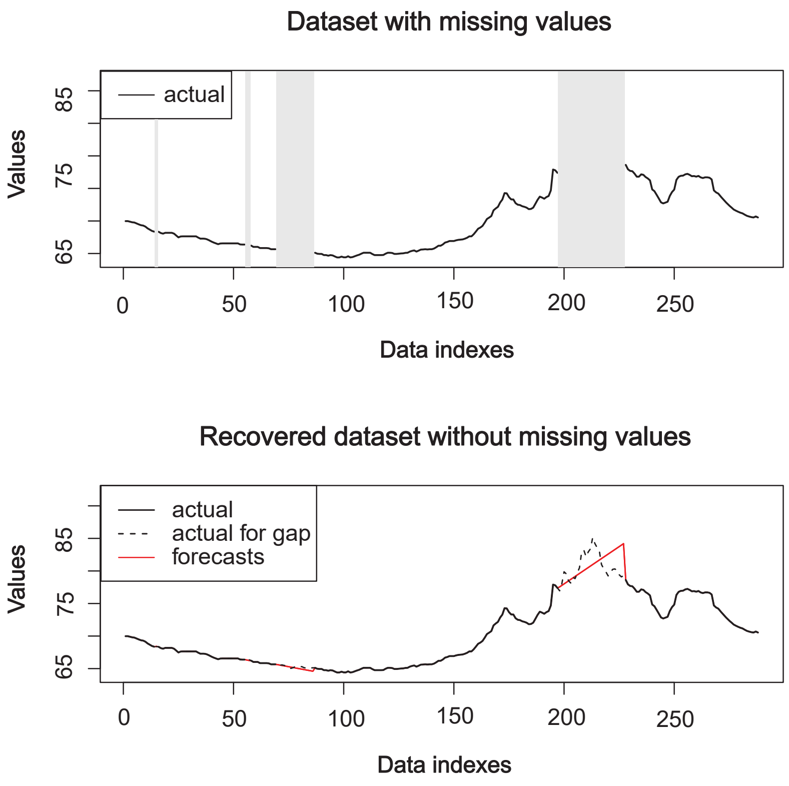


Figure 6.1: Adaptive data recovery of multiple gaps (G_1^1, G_2^2, G_3^{17} and G_4^{30}) on dataset *h_1*, employing ARIMA, as STR approach.

STR (single-technique recovery) scenario. Sensor-based time series, as shown in the first sub-figure, can contain different behaviors from stationary to trend and volatile patterns. For this reason, the MAPE of reconstructed gaps G_1^1, G_2^2 and G_3^{17} are significantly low, 0.1843%, 0.1317% and 0.3366% respectively, while the MAPE for the G_4^{30} is 2.6797%. We also notice a direct relationship between gap length and forecasting error. Results show that our mechanism is able to recover all gaps with a running time of $0.68s$. Moreover, based on the proposed mechanism, multiple techniques can be involved in the recovery process of each gap separately. MTR is employed by the mediator component in Section 6.1.4.

Recovery of multiple gaps simulation 2

Further, we extend the evaluation using larger subsets of the complete datasets (Table 6.1) from each source, targeting characteristics such as sensor type and a wider range of values, as it is shown in Table 6.2. This time, each dataset contains around 14600 data points for the second part of the experiments. To evaluate the proposed approach of adaptive and automatic near real-time data recovery, after the analysis of datasets we identify larger gaps and artificially made those gaps with approximately same sizes, precisely, gaps with 238 (G_1^{238}), 3 (G_2^3) and 5016 (G_3^{5016}) consecutive missing/invalid data values. Then, these gaps are recovered using the proposed mechanism, and forecast accuracy is evaluated by comparing predicted data with actual data from each gap. This choice has been made to ensure a fair comparison between all techniques, comparing them according to the forecast error they achieve on the same gap sizes and the same data.

We apply the proposed mechanism and show two representative examples among used datasets from each of data sources as it is illustrated in Figure 6.2. Each example is represented in a separated sub-figure. Each sub-figure consists of two graphs: the upper graph indicates incomplete dataset before recovering process, while the lower graph indicates complete dataset after applying recovering procedures. Gray shaded areas indicate three gaps, precisely, 238, 3 and 5016 missing values in a sequence. The black solid line represents existing values, that is, the actual state of a collected dataset on the edge. The black dashed line shows actual data for corresponding gaps, while the red solid line represents predicted values of missing data points as an output of applied

Table 6.2: Adaptive recovery dataset information.

Source	Dataset	Sensor type	Range of values
1	bsmart1	el. meter [kWh]	21.71-23.37
	bsmart2	room temp. [C]	21.06-23.78
2	hsmart1	room temp. [F]	65.89-83.30
	hsmart2	heat index [F]	27.86-107.64

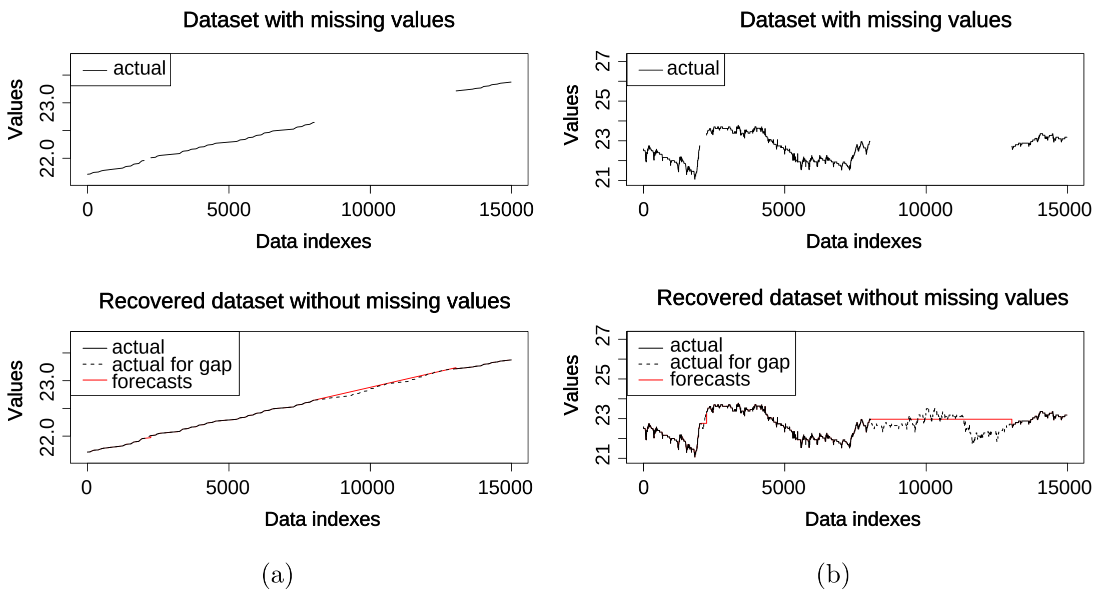
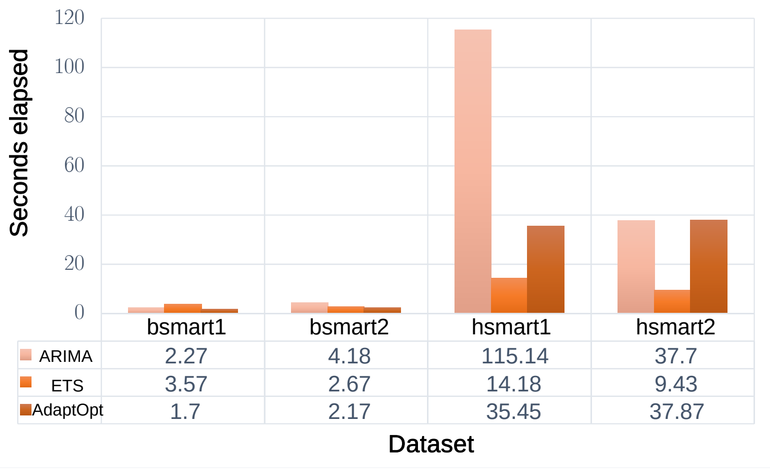


Figure 6.2: Results of semi-automatic recovery of multiple gaps. Sub-figures (a) and (b) are representative examples of recovered gaps for datasets *bsmart1* and *bsmart2*, respectively. It shows results of *STR*.

forecasting techniques. All gaps are again recovered using ARIMA technique from R forecast package, representing STR scenario, instead of selecting the most appropriate recovery method for each of three gaps, as in *condition-based recovery* of MTR. It can be seen that time series in the first sub-figure contains deterministic linear trend pattern that makes ARIMA forecasting appropriate and more accurate in comparison to the behavior of time series shown in the second sub-figure (i.e., flat forecasts for the third gap of Figure 6.2b). This dataset shows an irregular pattern making forecasting process more difficult. For this type of time series, where no trend or seasonality can be captured, the algorithm calculates best suitable straightforward prediction line with 95% confidence interval, i.e., the same possible mean value for missing data. For this reason, the MAPE of reconstructed gap G_3^{5016} by *bsmart2* is around 1.68315%, while the MAPE for the same gap by *bsmart1* is around 0.11999%. It can be also seen that, as the gap increases, the forecasting error increases too.

Finally, the graph confirms that the proposed automatic data recovery mechanism is able to efficiently cope with multiple gaps by forecasting all gaps with running time of 2.27s and 4.18s (see Figure 6.3), for *bsmart1* and *bsmart2*, respectively. These scenarios simulate users that specify a particular forecasting technique for the whole recovering process of incomplete dataset (see Subsection 3.2.2). On the other hand, based on the proposed mechanism, it is possible to specify different forecasting techniques involved in recovering process of each gap separately, i.e., MTR scenario, that is shown in the following section.



	bsmart1	bsmart2	hsmart1	hsmart2
ARIMA	2.27	4.18	115.14	37.7
ETS	3.57	2.67	14.18	9.43
AdaptOpt	1.7	2.17	35.45	37.87

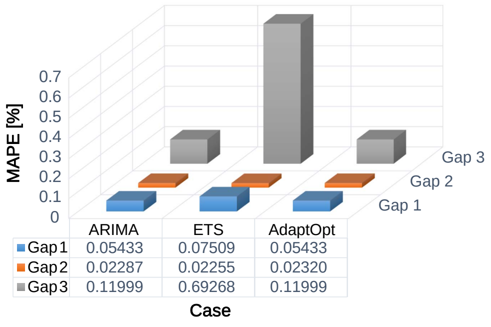
Figure 6.3: Running time of recovering all gaps.

Discussion on adaptive recovery of multiple gaps

We further evaluate the applicability of the proposed mechanism by recovering multiple gaps in different datasets and measuring accuracy and runtime for three cases: (1) ARIMA, (2) ETS, and (3) AdaptOpt (Adaptive Option), respectively. In the first two cases, the same technique is used for all gaps, that is, single-technique recovery (STR), while in the last case, condition-based MTR (multi-technique recovery), based on the combination of different forecasting techniques on different gaps, is employed.

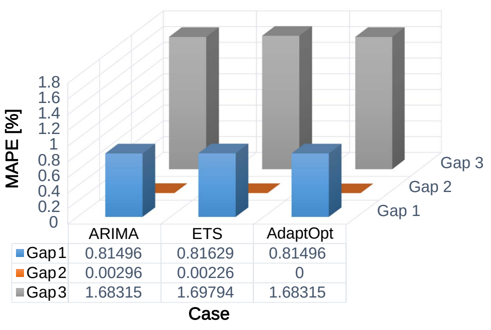
We compare forecast accuracy measures (see Figure 6.4) and perform code running time analysis (see Figure 6.3), showing the efficiency of the proposed mechanism with the selection of different forecasting techniques from the repository. Besides the possibility of using different techniques for each time series, it is also possible to select different techniques for each gap. This allows to avoid computation expensive techniques for small gaps or individual missing values and thus improving performance.

Figure 6.4 depicts the proposed mechanism applied on each of four different datasets containing the same sizes of multiple gaps. Each of the four sub-graphs represents one dataset. The recovering process is tested by utilizing different forecasting techniques in three cases shown on the horizontal axis, (1) ARIMA, (2) ETS and (3) AdaptOpt, respectively. In the first two cases, the same technique is used for recovering all gaps, among which some graphs have been shown previously. The latter (Adaptive Option) depicts the second scenario, where we utilize a combination of different forecasting techniques, showing the benefits of the recovering procedure. The vertical axis shows MAPE for corresponding gaps. Comparing the MAPE between same gaps within one particular dataset, the error increases as size of gap increases in *bsmart1* and *bsmart2*, while for *hsmart1* and *hsmart2* is the opposite. The reason lies in a wider range of values and more volatile behavior for data from source 2, where, e.g., for *bsmart2*, the error increases by 107.02% on average, while for *hsmart2* it decreases by 46.47% on average,



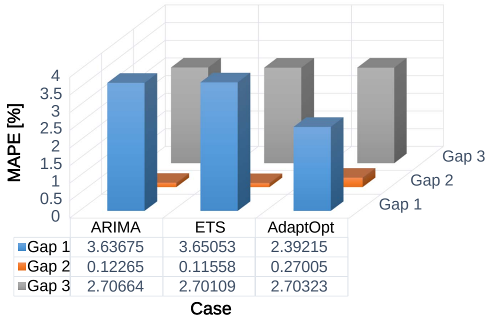
	ARIMA	ETS	AdaptOpt
🟦 Gap1	0.05433	0.07509	0.05433
🟧 Gap2	0.02287	0.02255	0.02320
⬜ Gap3	0.11999	0.69268	0.11999

(a) Data recovery in dataset *bsmart1*.



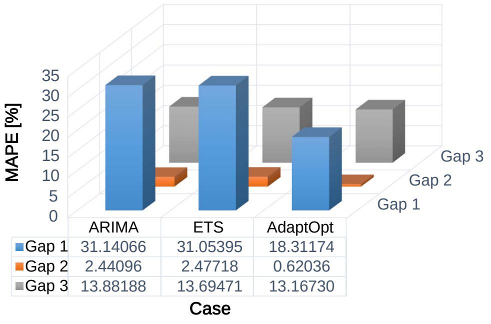
	ARIMA	ETS	AdaptOpt
🟦 Gap1	0.81496	0.81629	0.81496
🟧 Gap2	0.00296	0.00226	0
⬜ Gap3	1.68315	1.69794	1.68315

(b) Data recovery in dataset *bsmart2*.



	ARIMA	ETS	AdaptOpt
🟦 Gap 1	3.63675	3.65053	2.39215
🟧 Gap 2	0.12265	0.11558	0.27005
⬜ Gap 3	2.70664	2.70109	2.70323

(c) Data recovery in dataset *hsmart1*.



	ARIMA	ETS	AdaptOpt
🟦 Gap 1	31.14066	31.05395	18.31174
🟧 Gap 2	2.44096	2.47718	0.62036
⬜ Gap 3	13.88188	13.69471	13.16730

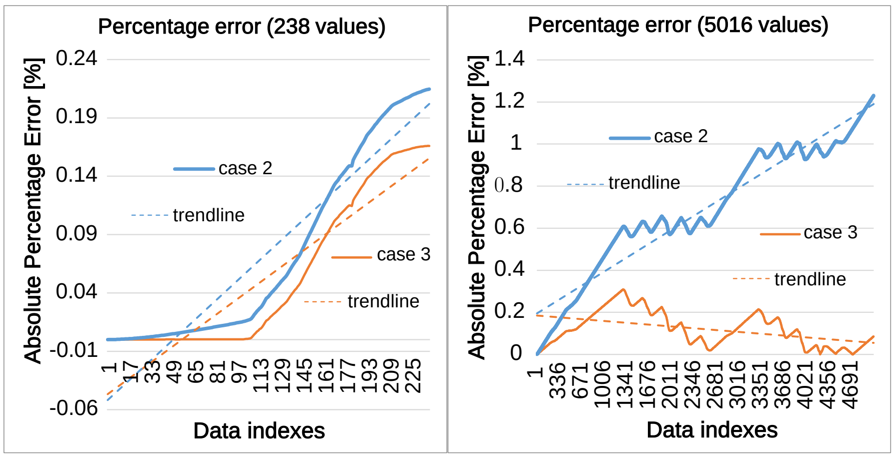
(d) Data recovery in dataset *hsmart2*.

Figure 6.4: MAPE accuracy measure for three recovered gaps of missing values among 4 datasets. All three gaps (that is, G_1^{238}, G_2^3 and G_3^{5016}, respectively), are recovered by utilizing different forecasting models in three cases: ARIMA, ETS and AdaptOpt.

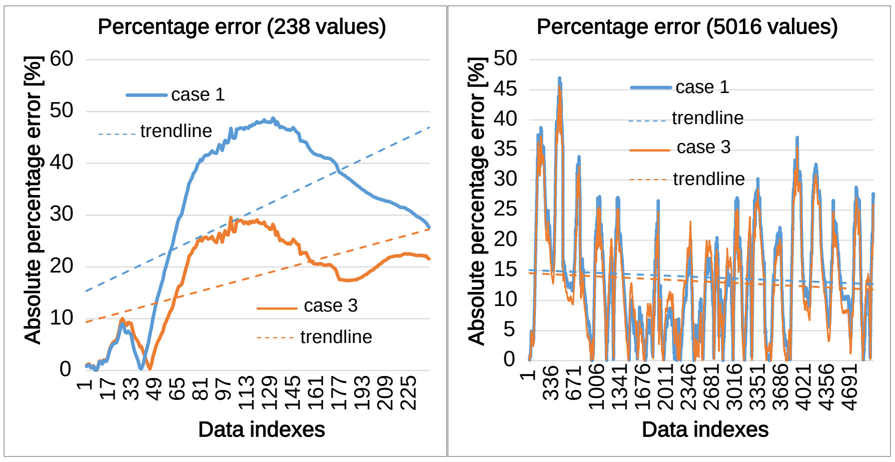
between gaps G_1^{238} and G_3^{5016}.

For the case 3, in each dataset, three gaps are recovered, respectively, by applying techniques ARIMA, ETS, ARIMA for *bsmart1*, ARIMA, n-point average, ARIMA, for *bsmart2*, TBATS, n-point average, ETS, for both, *hsmart1* and *hsmart2*. With this setup, we are able to reduce overall running time by 25.11% and 52.38% for *bsmart1* comparing case 3 to cases 1 and 2, respectively, and at the same time having 82.68% less forecasting error for gap G_3^{5016} comparing with case 2. For *bsmart2*, we have a slightly better forecast accuracy, but reduced overall running time by up to 48.09% in comparison to cases 1 and 2. Additionally, by performing n-point average for gap G_2^3, we were able to completely recover that gap, having 0 for the MAPE value, because of the presence of redundant predecessor values. Although the running time for the case 3 is not the lowest for *hsmart1* and *hsmart2*, the error decreases up to 41.2% for the first gap of dataset *hsmart2*. By performing the proposed *AdaptOpt* in case 3, depending on the type of datasets, we are able to reduce either only forecast error or both, the error and the running time.

Figure 6.5 shows absolute percentage error behavior for two datasets (*bsmart1* and *hsmart2*) along two bigger gaps (gaps G_1^{238} and G_3^{5016}) of the incomplete data. It



(a) Recovered gaps from *bsmart1*.



(b) Recovered gaps from *hsmart2*.

Figure 6.5: Absolute percentage error behavior along recovered gaps comparing with proposed *AdaptOpt* (case 3).

illustrates that application of the proposed *AdaptOpt* approach, overall, results in a smaller error which is shown also by calculated trendlines. Although the gap to recover can be significantly bigger (5016 values), we see that the MAPE increases only by up to 0.31% for *bsmart1* and to the peak value of 46.57% for *hsmart2*. Because of the volatile behavior of data from *hsmart2*, the error can come to the higher peak values. However, with the appropriate methods, it can be greatly improved, e.g., using ARIMA instead of ETS and TBATS instead of ARIMA for *bsmart1* and *hsmart2*, respectively. Considering unlikely cases with big gaps at the edge, the experimental results confirm the benefits of our approach, and especially of the selection of different techniques for recovering different gaps within the same incomplete dataset.

6.1.3 Edge Storage Management Evaluation

Based on the proposed edge storage management mechanism and principles in Section 3.3, we evaluate the accuracy of the forecasts in predictive analytics for decision-making processes and the amount of data that have to be stored on the edge node. First, we show an example and application of the proposed algorithm, focusing on the recognition and selection of clusters with stable forecast accuracy. Then we perform an experimental evaluation of the edge storage mechanism, showing the benefits of implementing our algorithm in edge-limited storage.

Example of stable forecast accuracy cluster detection

To explore the full potential of the proposed adaptive algorithm (described in Section 3.3.2), we show a simple dataset with a seasonality pattern that has more than one cluster with stable forecast accuracy. We provide a step-by-step evaluation of the algorithm. For the experiments, we manually defined rules to simulate the specification list. The forecast period is set to 24, the same as the periodicity of the dataset. The forecast period stays fixed in a current cycle, while a client can change the desired forecast period in the specification list anytime. Then, we consider a high threshold for the forecast accuracy of 90%, since we expect high forecast accuracy results due to the small forecasting period in comparison with the available dataset size.

In Figure 6.6, the upper graph represents the original dataset, while the lower graph represents the result after applying ETS forecast method validating the principle for multiple forecast iterations on the available dataset. The original dataset is composed of 336 data points. As we already defined the forecast period, the last 24 data points become test data, while the others will be used in different ranges to predict the same forecast period.

Blue bold points in the lower graph indicate forecast accuracies for the corresponding number of used data, i.e., the number of data that is artificially decreased in each iteration to capture different forecast values estimating the same forecast period. For example, the first blue point shows us that in the forecasting process 312 data points are used and the forecast accuracy is slightly below 93%. In the lower graph of Figure 6.6, some stable behaviors of forecast accuracies are visible, and their automatic recognition is performed by Algorithm 3. Two stable clusters are selected and shown in Table 6.3. The resulting table provides mean values of accuracies in each cluster and corresponding indexes of data points from the storage for each cluster. Additionally, an interval of available data points shows how many data points remain between the start/end index of the cluster. The sum of the corresponding cells gives the total number of test data.

The selection of the appropriate cluster is done according to Algorithm 4. The forecast accuracy threshold for the current observations is compared with the mean value of each cluster. Therefore, the first cluster is selected at first as an appropriate cluster. Further, checking the conditions for a better accuracy value with a smaller dataset, the second cluster replaces the old one and becomes a new selected cluster.

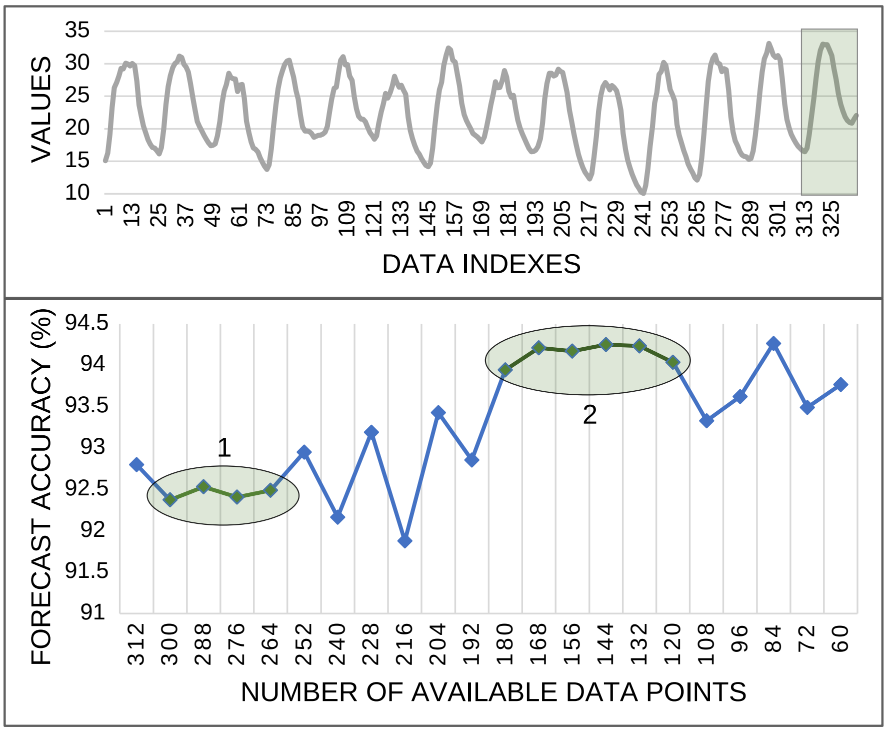


Figure 6.6: Observed dataset and forecast accuracy observation with stable accuracy clusters detection.

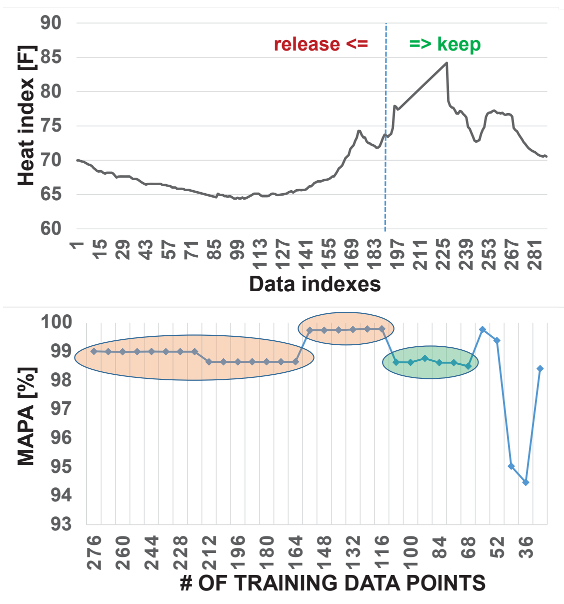
Table 6.3: Stable clusters of forecast accuracy.

#	Mean value	Range of cluster indexes		Interval of available data points	
		Start	End		
1	92.44903	12	48	300	264
2	94.13858	132	192	180	120

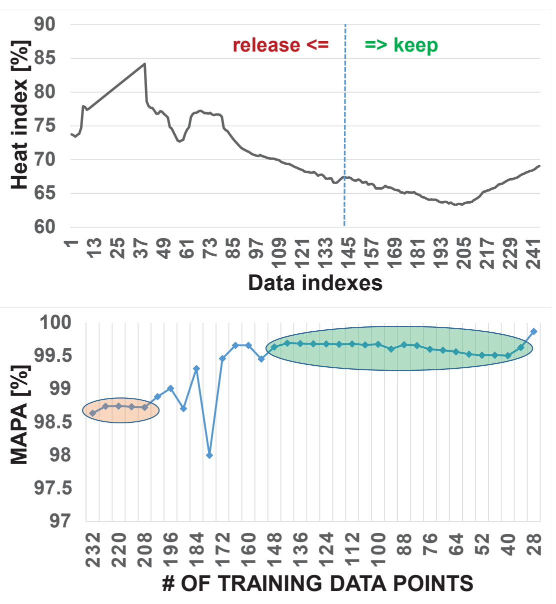
Finally, at the end of each cycle, a data management action will release data points starting from the oldest data point, i.e., with index 1, until the middle of an appropriate cluster, i.e., by index 162, thus keeping 174 available data points in the storage. In this demonstrating example, based on our proposed approach, it is possible to reduce the amount of stored data by 48% in one cycle, while keeping the accuracy of our predictions above a specified threshold.

Edge storage management simulation

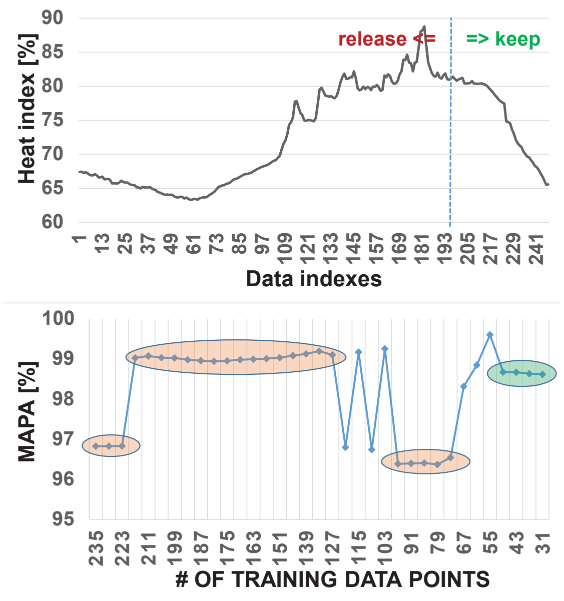
We simulate edge storage management with a fixed amount of data. We use Algorithm 4 for appropriate cluster selection. Subsets of a dataset *h_1* is shown in Figure 6.7 and



(a) Cycle 1



(b) Cycle 2



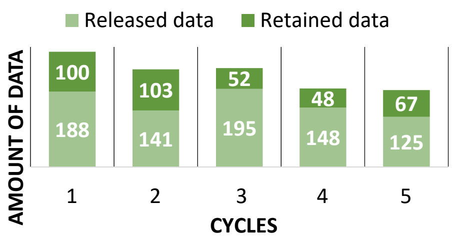
(c) Cycle 3

Figure 6.7: Evaluation of edge storage management on *h_1* dataset - cycles 1-3 showing available dataset (upper graphs) and stable clusters of forecast accuracies (lower graphs). Vertical blue dotted line represents retained data points.

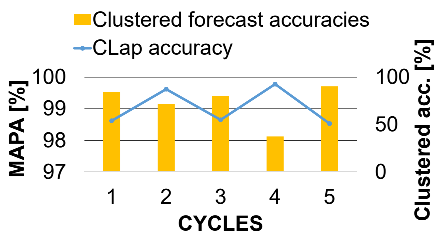
represents 3 cycles of edge storage management. In the first cycle, the same data from the data recovery process are used. Further, in the second and third cycles, we set, in addition to data coming from the previous cycle, the upcoming amount of data on 144 (5min interval for 12h) data points per turn. Also, we define certain rules to simulate the specification list. Forecast horizon f_h is set to 12 data points, representing one hour $(12 \cdot 5\text{min})$ for which forecasts are calculated. Forecast horizon f_h is fixed in the current cycle, while the user can change the desired f_h in the specification list. Then, we consider the threshold CL_{th} for identifying stable accuracy clusters of 90%, since the proposed framework targets near real-time decision-making in IoT applications and we expect accurate MAPA measures due to the short f_h comparing to the available dataset size.

Regarding Figure 6.7, the upper graph represents the original dataset. The vertical blue dotted line represents the data management decision after the procedures from lower graphs. The lower graphs represent the result after applying the forecast method and validating the principle for multiple forecast iterations on the available dataset. In MAPA measurements, orange areas are stable clusters, while green areas are selected appropriate clusters. The last 12 data points are test data, while the rest is used in different variations to predict f_h. In the first cycle, the algorithm finds an appropriate cluster in a range between 68 and 108 available data points, corresponding to the cluster between data indexes 168 and 208 in our original dataset (upper graph). The central index of that cluster indicates that data management will release data points in range 1-188, respectively, indexes in range 189-288 will retain in the edge storage. The process repeats for each next cycle. Figure 6.8 summarizes all 5 cycles.

Figure 6.8a shows both, released and retained data per each cycle, while Figure 6.8b shows the accuracy of the selected appropriate cluster CL_{ap} and the percentage of clustered MAPA values (lower graphs in Figure 6.7) per cycle. In Table 6.4, all 5 cycles are averaged and compared among datasets in Table 6.1. The results show that on average 39.9% of data points can be retained, while keeping the appropriate cluster CL_{ap} accuracy, depending on algorithms' parameters. Generally, we can find appropriate clusters with accuracy around 98.83% and clustered MAPA measurements above 50%, based on approximately 34 multiple forecast iterations. Results show that our mechanism can achieve user-defined high accuracy using less data.



(a) Data management



(b) Output details

Figure 6.8: Released/retained data, appropriate cluster accuracy and clustered forecast accuracies percentage of Algorithm 5 after 5 cycles.

Table 6.4: Mean results of 5 storage management cycles per dataset.

Dataset	**Retain.**	**Retain. [%]**	**Clusters**	**Clust. [%]**	**Iterations**	CL_{ap}	**Time**
h_1	74	31.4	3.2	72.73	33	99.04	2.48
h_2	63.4	33.4	3.8	65.06	33.8	99.77	2.53
h_3	94.6	42.8	4	59.89	34.6	96.90	3.09
b_1	69.8	35.2	3.2	73.88	33.8	99.99	2.26
b_2	97.2	44.6	3.6	81.24	34.4	99.20	3.10
b_3	137.2	52	3.4	61.43	33.6	98.05	2.62

Table 6.5: Comparison of PRM-based Multiple Technique Recovery (MTR) vs Single Technique Recovery (STR) for four gaps on six datasets.

Datasets	h_1 STR	h_1 PRM	h_2 STR	h_2 PRM	h_3 STR	h_3 PRM	b_1 STR	b_1 PRM	b_2 STR	b_2 PRM	b_3 STR	b_3 PRM
Runtime [s]	0.68	2.47	0.98	0.83	0.97	1.62	0.89	0.75	1.2	1.1	1.08	0.92
MAPE	0.8331	0.8061	0.4919	0.3476	13.3229	11.789	0.0084	0.0029	0.9377	0.9321	0.767	0.7836

6.1.4 Projection Recovery Maps Evaluation

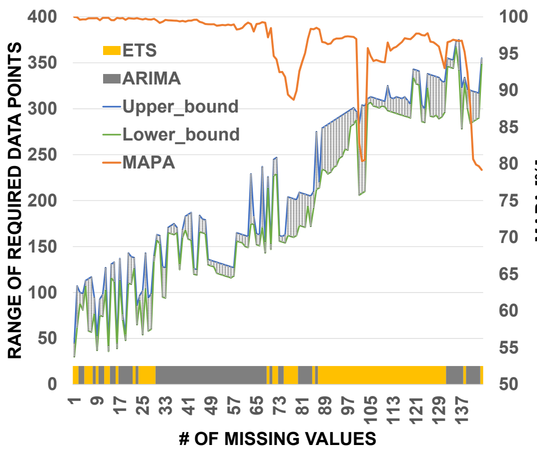
In Section 3.3.2, we defined two methods for data recovery, namely, STR and MTR. Section 6.1.2 first describes two examples of the STR scenario, where each gap is recovered with a single technique, that uses data points preceding each gap as input. We also demonstrated the benefits of the MTR method in which applying different techniques for various gaps within the incomplete dataset can reduce either only the forecast error or both the error and computation time. To make the second method MTR self-adaptive and automatic, we employ PRMs (Projection Recovery Maps), used by the adaptive recovery mechanism to select both the recommended range of required data points and recommended recovery method.

In Figure 6.9, for each number of missing values from indexes 1 to 144, the algorithm finds recommended range of required data points by finding stable clusters (described in Algorithm 3) and calculating the most accurate cluster. The original algorithm is modified by considering stable clusters with the highest forecast accuracy (see Definition 3.6). The blue line shows the upper border of the cluster, while the green line shows its lower border. We apply ETS and ARIMA forecast methods, and the method with the highest accuracy is selected. Figure 6.9(a-f) shows PRMs for all datasets. ETS results are in yellow, while ARIMA is in grey. For each selected range of required data points, we show the MAPA value in orange.

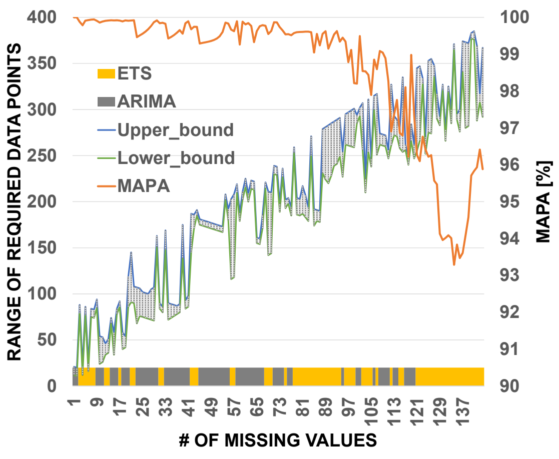
The mediator component can store the completed PRMs, and once the monitoring component detects a gap, the mediator recommends a range of required data points and a forecasting method for recovery. In case there is not enough data in edge storage, the mediator component retrieves data from the cloud, storing them for the time of data recovery. We test data recovery with 4 defined gaps (see Section 6.1.2) using STR as a baseline. In Table 6.5, we compare running time and MAPE between STR and PRM-based MTR. STR achieves a running time of $0.97s$ on average, while PRM-based recovery can achieve up to 65.48% less error (e.g., dataset b_1) and only 31.96% more time on average compared to STR. Using PRMs, we can improve at least one objective or both in some cases.

6.2 System Deployment and a Performance Evaluation of Traffic Safety with Edge and 5G

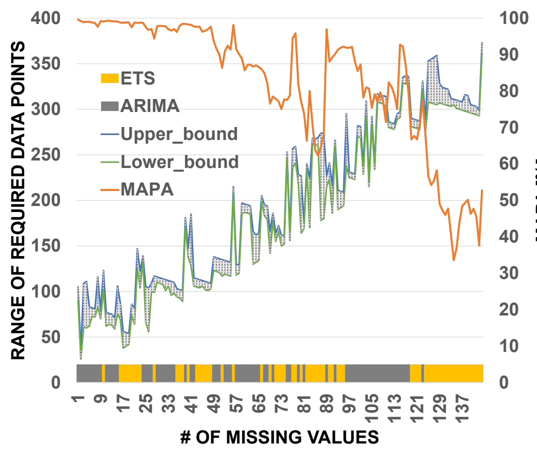
Based on the architecture design discussed in Section 5.1, we implemented and deployed a prototype on a critical intersection in the city of Vienna. We evaluate the system performance showing its capability to detect critical situations and deliver time-critical warnings, e.g., timely sending notifications to drivers. We first provide technology evaluation of both computation (including hardware and software) and communication (including network protocol, quality of service and protocol setup) components.



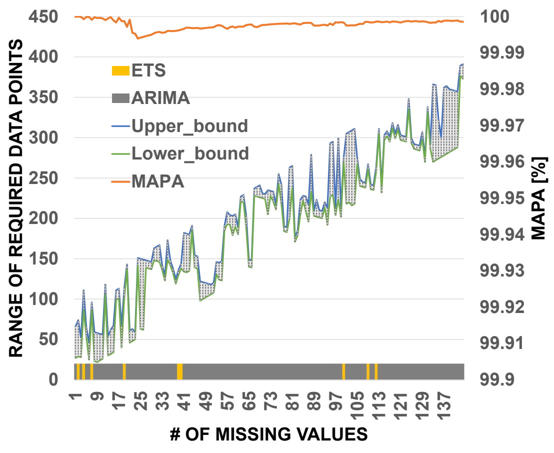
(a) h_1



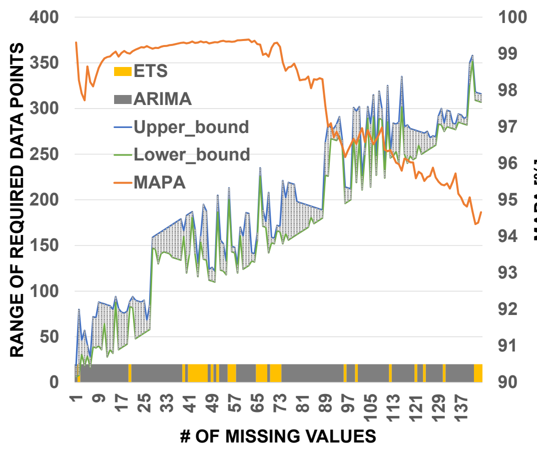
(b) h_2



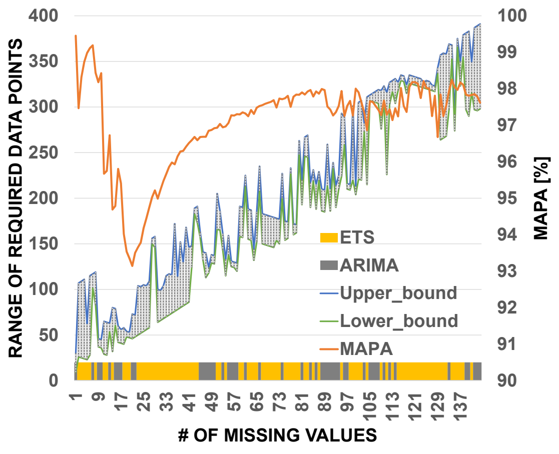
(c) h_3



(d) b_1



(e) b_2



(f) b_3

Figure 6.9: Projection recovery maps for six evaluated time series datasets.

6.2.1 Technology Evaluation

Computation component

Hardware. Object detection performance depends on selected hardware. Due to physical space limitations of the traffic light chamber, we employ single-board Raspberry Pi (RPi) edge devices, to which we attach co-processors to improve the performance of neural network inference. As co-processors, we evaluate Google's Coral Edge TPU accelerator, and Intel's Neural Compute Stick 2 (NCS2) since both can be plugged via USB and used for vision-based ML applications. We select Coral Edge TPU, since it supports TensorFlow Lite models (lightweight solution to run ML TensorFlow models on edge devices). Considering space and low-latency requirements, we evaluate different RPi models with/without Coral's Edge TPU. Table 6.6 shows technical details. To capture video frames from the target intersection, we use RPi 8MP Camera Module V2.

Software. The software module is developed using TensorFlow Lite, a version of the popular TensorFlow framework optimized for limited IoT devices, including RPi (Raspberry Pi). To select the best-performing edge configuration, we first collected video frames from the chosen intersection, on which the prototype should be deployed. Then, we evaluated the performance of both quantized MobileNet SSD v1 [84] and v2 [85], lightweight and pre-trained convolutional neural network (CNN) based object detection models, trained using the standard COCO [82] dataset, on our collected dataset.

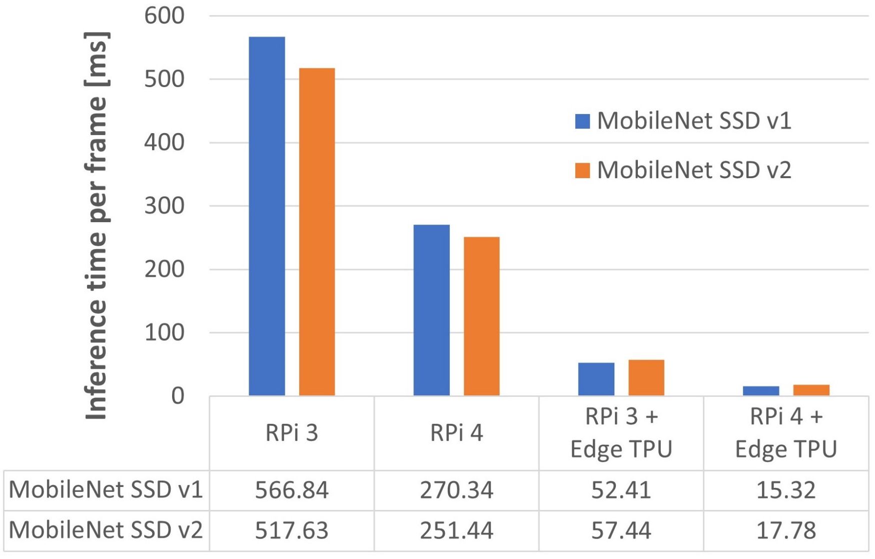
Figure 6.10 shows the average inference time per single frame on different edge node configurations. The results are averaged over 100 frames for statistical significance, since by adding more frames the difference in the standard deviation of inference times is below 39 μs on average. We observe in Table 6.6 that RPi 4 with Edge TPU has overall the lowest inference time for both models.

Furthermore, we check confidence scores of the model, calculated using Tensorflow confidence function[1], for both models in Figure 6.11. We consider quantized model versions, to further improve latency with limited effect on inference score. Here, the object detection module is set to only detect a class "person" from a collected dataset with a confidence threshold of 0.5 (i.e., a cut-off threshold for accepting detection results). Although MobileNet SSD v1 has a slightly lower inference time by 2.46ms (or 13.84%) on

[1]https://www.tensorflow.org/lite/models/object_detection/overview

Table 6.6: Edge node configurations.

Node type	**CPU**	**RAM [GB]**	**Edge TPU**
RPi 3 B+	Quad-core Cortex-A53 (ARMv7) at 1.4GHz	1	no
RPi 3 B+	Quad-core Cortex-A53 (ARMv7) at 1.4GHz	1	yes
RPi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4	no
RPi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4	yes



	RPi 3	RPi 4	RPi 3 + Edge TPU	RPi 4 + Edge TPU
MobileNet SSD v1	566.84	270.34	52.41	15.32
MobileNet SSD v2	517.63	251.44	57.44	17.78

Figure 6.10: Inference time observation for different edge node configurations and two object detection models.

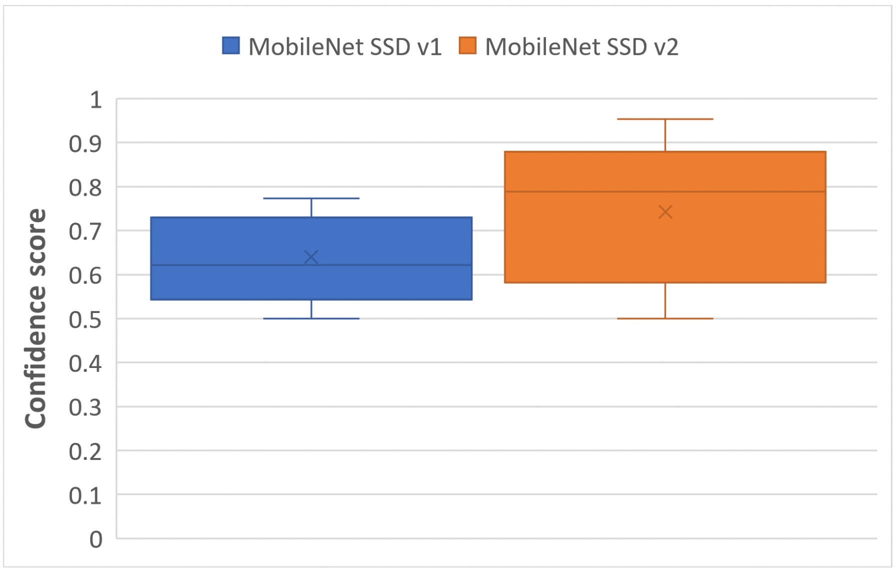


Figure 6.11: Confidence score observation for two object detection models on RPi 4 with Edge TPU (threshold set to 0.5).

average, MobileNet SSD v2 shows a better inference score of 16.12% on average. Thus, we select as the best option MobileNet SSD v2 running on RPi 4 with Edge TPU.

Communication component

InTraSafEd 5G is designed to work in the mobile context, thus we consider 3G, 4G and 5G physical and transport layers. We describe now all design choices related to the communication component.

Network Protocol. Since the main goal of *InTraSafEd 5G* is to provide notifications of critical situations within specific time frames, the selected communication protocol should (i) have a minimal message overhead to keep the transfer time and latency low, (ii) offer guaranteed delivery of messages to users, and (iii) avoid unnecessary network flooding. Traditional client-server communication (e.g., HTTP in cloud-based web applications), is not suited for this scenario, due to high message overhead and the necessity of polling to be notified about new events (e.g., a pedestrian detected in a blind spot). We focus then on publish/subscribe (Pub/Sub) protocols, which allow event-based notification and dynamical targeting of drivers close to a particular crossroad.

We select the following protocols for evaluation, used in different contexts requiring near real-time latency: CoAP, DDS, AMQP, MQTT. In Table 6.7 we show the result of our comparative study. First, we focus on Pub/Sub protocols, to enable event-triggered data transmission and avoid unnecessary polling. Also, since accurate detection of pedestrians and cyclists in our scenario needs to aggregate sensor data in a single processing point, a protocol designed for centralized processing is more desirable than a distributed protocol. Another desirable feature is the possibility to choose between different Quality of Service (QoS) levels according to provided latency. A good level of security should also be provided since no malicious users should be able to inject fake detection data and potentially causing trouble. Finally, we want to ensure that the header size of each message is low, to reduce network load. Based on our analysis, we do not select CoAP and AMQP since they do not offer a publish/subscribe communication model. Between publish/subscribe protocols (DDS and MQTT) we select MQTT, due to its lower overhead and the fact that it is most suited for centralized processing.

Quality of Service. Besides strict low-latency of delivery, Intrasafed communication layer has also to provide guarantees on delivery of the notification. Indeed, the potential effects of unreliable communication might cause the loss of important updates, with negative effects on traffic safety. Since our communication layer is based on the MQTT protocol,

Table 6.7: Comparative study of IoT protocols. Pub/Sub = Publish/Subscribe; R/R = Request/Reply; P2P = Point to Point.

Protocol	**Paradigm**	**Processing**	**QoS levels**	**Security**	**Header size [bytes]**
CoAP	R/R	Distributed	4	DTLS	4
DDS	Pub/Sub	Distributed	5	TLS	48
AMQP	P2P	Impl. specific	3	TLS	8
MQTT	Pub/Sub	Centralized	3	TLS/SSL	2

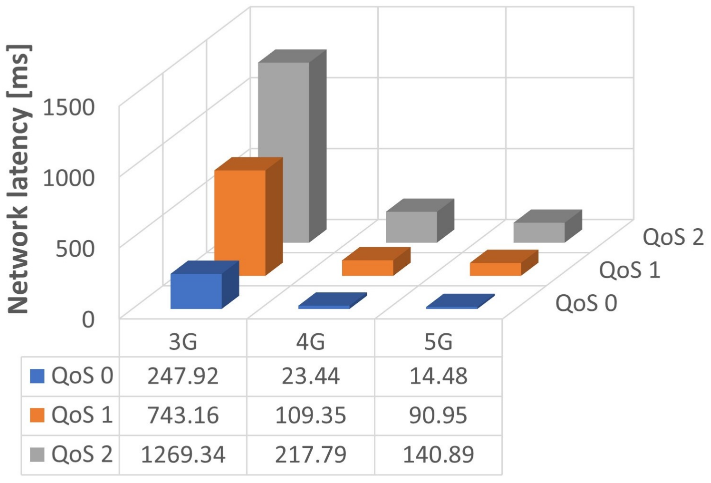
we focus on MQTT QoS layers.

MQTT offers three different QoS levels: *QoS 0*, where messages are delivered AT MOST ONCE (no guarantee on message delivery); *QoS 1*, where messages are delivered AT LEAST ONCE (message delivery is guaranteed, but replications might occur); and *QoS 2*, where messages are delivered EXACTLY ONCE. Since selected QoS level affects the latency of notifications, we measure the latency of MQTT messages in our scenario using different QoS levels.

Latency measurements are performed employing a self-developed Android mobile app, which first subscribes to a specific topic, then it sends a message with a unique id using the same topic. Once it receives the same messages, it calculates the latency based on the round-trip time of the message. The payload of the message is set to 64 bytes that is realistic for the amount of data managed by the application (the number of pedestrians and cyclists in a blind spot).

Figure 6.12 shows network latency offered by different QoS levels on different network layers (3G, 4G and 5G). Results are obtained by calculating the average latency of 100 messages, for statistical significance. From the plot, we see that in most of the cases average notification latency falls within the requirements of tens to hundreds of milliseconds for road safety applications [18]. However, when selecting QoS 2, the 95% confidence interval of our measurements for 3G includes values above 1000ms. For this reason, we select QoS 1, which ensures latency below 800ms in the worst case (3G, QoS 1). Also, in our scenario, the average notification latency is 109.35ms and 90.95ms, respectively for 4G and 5G.

Protocol Setup. Finally, we describe the general MQTT protocol setup. MQTT message routing relies on a software component called MQTT broker, which receives messages



	3G	4G	5G
■ QoS 0	247.92	23.44	14.48
■ QoS 1	743.16	109.35	90.95
■ QoS 2	1269.34	217.79	140.89

Figure 6.12: MQTT network latency observations for QoS=$\{0, 1, 2\}$ over different network types (3G, 4G and 5G).

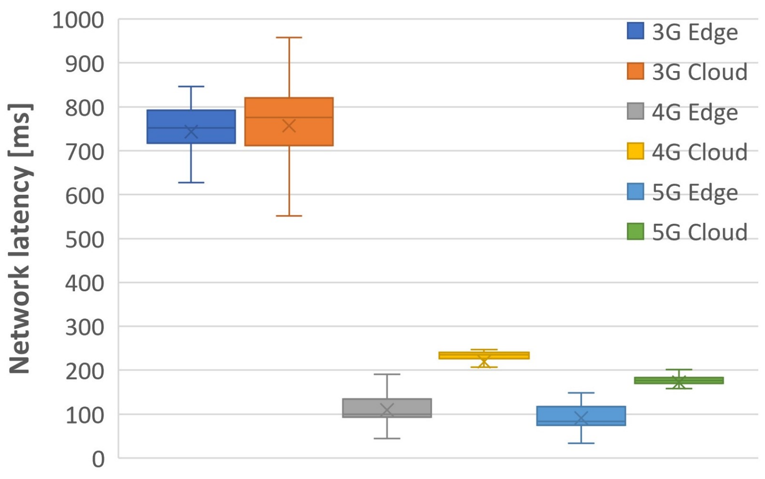


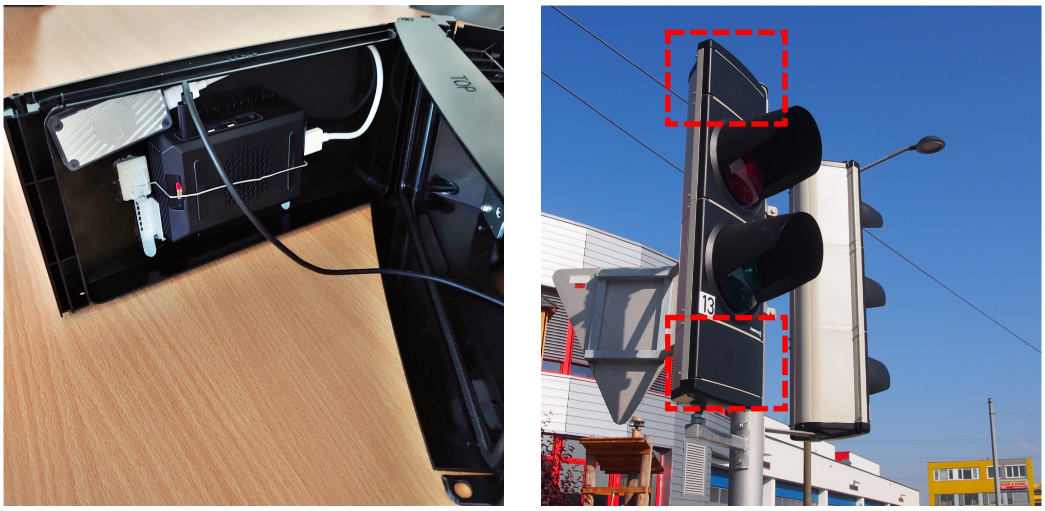
Figure 6.13: MQTT network latency for QoS=1 w.r.t. edge/cloud broker placement.

published by different clients (publishers) and forwards them to clients subscribed to the messages' topic (subscribers). In the target scenario, each camera with RPi represents a publisher, and a mobile app is a subscriber. In the current prototype, topic subscriptions are set up when the application is started. In future work, we plan to investigate location-based subscriptions, as described by [177]. We select Mosquitto MQTT broker [178], which offers enough security and flexibility for our requirements.

Each client has to be able to access the MQTT broker over the Internet. Since latency is the main requirement of our application, we evaluate the latency of deploying the broker at the edge (inside TU Wien's network infrastructure) or using a cloud service instead. Evaluation of latency of both deployments is shown in Figure 6.13. We can see that edge deployment significantly reduces the latency (up to 50.20% and 47.18% for 4G and 5G networks, respectively), making edge broker placement as the best choice for our latency-critical scenario.

6.2.2 Design Choices

Based on the proposed system design, *InTraSafEd 5G* collects video frames from cameras deployed on smart traffic lights, targeting drivers' blind spots. Video frames are then processed by an object detection algorithm, implemented using Python 3.7, to identify critical situations (e.g., pedestrians and cyclists outside drivers' field of vision). To this end, we (i) deployed edge nodes using RPi 4B, to which we attached a camera and Edge TPU (Coral USB Accelerator) to speed up object detection (see Figure 6.14a) and (ii) set up a network connection using a HTC 5G Hub (5G network coverage is set exclusively for the prototype deployment). We placed the edge node (above) and 5G Hub (under) specific traffic lights (see Figure 6.14b). Figure 6.15a illustrates the detection of pedestrians in drivers' blind spots (e.g., behind bushes or bus stations) by installed



(a) Edge node deployment. (b) Smart traffic light integration.

Figure 6.14: Edge nodes setup on Vienna's chosen intersection and the integration into the traffic-signal chambers.

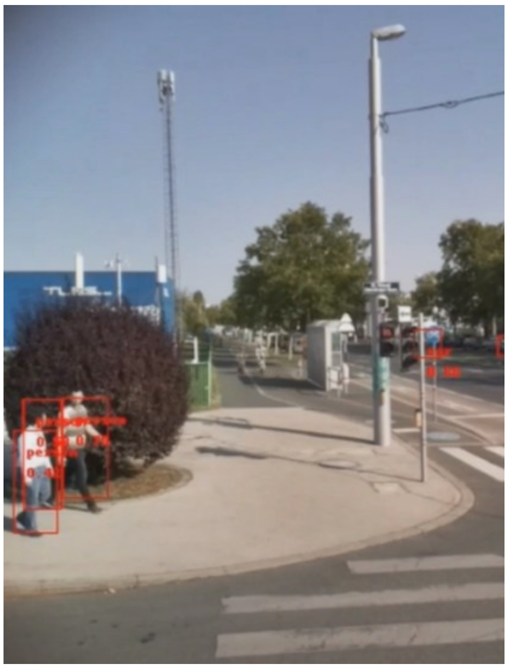
edge nodes. Results from different edge nodes are then aggregated and audio and visual notifications are sent to the driver's mobile device if a critical situation is detected while a driver approaches the covered intersection (see Figure 6.15b).

6.2.3 Mobile Client

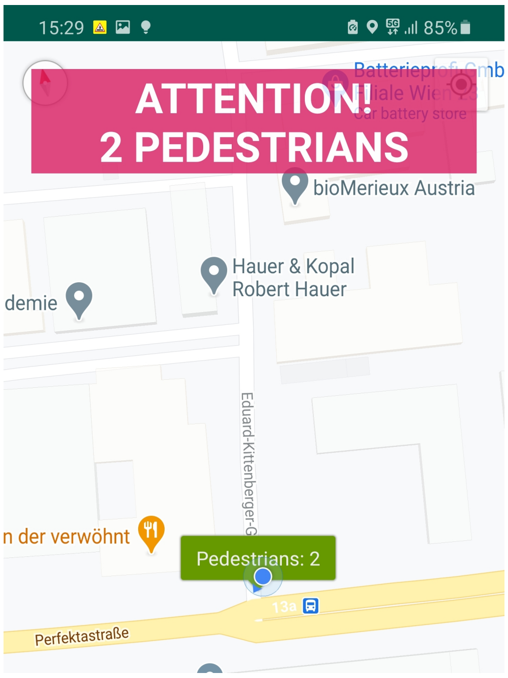
Notifications of critical situations are sent to a mobile client installed on the driver's mobile phone. *InTraSafEd 5G* mobile client is developed for Android 10 using Kotlin. We leave the implementation of iOS and Windows Phone versions for future work. The application works as follows: first, the app subscribes to the topic representing a monitored critical area and registers to the broker. Once the message is received, the app visualizes a message using overlays and playing an audio notification. Communication with the MQTT broker is performed through the PAHO MQTT library 1.1 for Android, while the location is obtained using the mobile phone's GPS and Google Maps SDK v3.1.0. An example of the mobile client interface is shown in Figure 6.15b.

6.2.4 Discussion

Transmission latency on 5G is measured to be around 90.95ms for QoS 1, which guarantees enough time for timely notifications. Regarding the computational performance on the edge device, object detection on camera-collected frames takes around 17.78ms using a lightweight RPi 4 with Edge TPU. This solution not only reduces cost while providing high-level performance, but also allows increased privacy, as data taken by cameras do not travel over the network nor need to be stored on the devices. Overall, in our *InTraSafEd*



(a) Pedestrian detection.



(b) Screenshot of user application.

Figure 6.15: Real-time edge analytics demonstration for increasing traffic safety. Subfigure (a) shows the output of object detection (during the project demonstration day) on edge node Raspberry Pi 4B, while Subfigure (b) shows the application running on the driver's Samsung S20 5G phone.

5G use case, detection of pedestrians on the crossroad together with the required latency to notify drivers results in 108.73ms on average.

We also observed that the proposed system with QoS level 0 results in overall latency of 32.26ms on average for 5G. However, although this QoS level does not guarantee the delivery of notifications, it is appropriate in situations where (i) the connection is reliable (e.g., cellular antennas deployed close to the target area), and (ii) message loss on a small scale would not affect the early warning system (e.g., capturing and processing higher frame rates). Still, in future work we plan to investigate the scale of message loss using QoS level 0.

The installation of the proposed solution on a higher scale of a metropolitan area, would require (i) identification of important intersections, (ii) installation of 5G network coverage, (iii) integration of the proposed design choices, and (iv) owning a 5G-enabled phone. Although using 5G connection can offer the full benefits of the proposed solution, (i) owning a 5G-enabled phone can be costly for users, as well as (ii) installing 5G coverage in all areas. However, as we showed, the proposed solution is designed to work also with common 3G/4G enabled phones and network types, (e.g., 3G and 4G only available in rural areas of the city), still within acceptable total latency of 760.94ms (3G) and 127.13ms (4G) on average for QoS 1. Furthermore, the cost of edge hardware integration should be in the range of € 200 on average for edge configuration such as RPi 4B, 8MP Camera Module V2 and Edge TPU.

6.3 Self-adaptive and Locality-aware Edge Analytics Placement Evaluation

SEA-LEAP system presented in Section 5.2 is implemented using Python, and has been evaluated conducting experiments on (i) real-world video analytics application; (ii) real-world sets of video frames as input for the application; (iii) obtained network and inference benchmarks, and (iv) heterogeneous edge infrastructure using Kubernetes platform. Our emulation-based evaluation utilizes RuconLiveLab, our physical edge infrastructure consisting of 11 Raspberry Pi (RPi) single-board computers, available in three different configurations (see Table 6.8). Experimental results showed that the proposed *SEA-LEAP* can (i) autonomously deploy on-demand analytics requests (e.g., object detection) considering data locality, and (ii) reduce overall request execution time.

6.3.1 Implementation Details

We first introduce technologies used for the experimental evaluation of the proposed *SEA-LEAP*. Considering the deployment of analytics applications, many researchers and industries are revealing nowadays the rapid adoption of Kubernetes orchestration platform [165, 166], relying on master-worker architecture. The master node is, in our scenario, responsible to assign a container-based analytics application to one of the available nodes in the corresponding cluster. Containerized applications are typically using Docker, a container platform used to build and isolate applications with a relevant stack of services. Here, to deploy an analytics application to edge nodes, a docker image has to be included in the Kubernetes manifest, i.e., deployment file, typically defined in YAML (described later in Section 6.3.5 and shown in Figure 6.18).

6.3.2 Target Application

We consider as our target application object detection, a typical video analytics processing in which an input set of video frames is analyzed. Analytics output is a list of detected objects with confidence levels and their positions on the image. We assume that edge keeps only a limited number of frames, e.g., sampling an industry-standard frame rate of 30fps and filtering only frames with significant changes or object movements, due to the limited capacity of edge nodes and efficient bandwidth usage [73].

In this experimental setup, we used the computation logic from our real-world application *InTraSafEd 5G* (introduced in Section 5.1 and evaluated in Section 6.2), used to perform object detection analytics to increase traffic and pedestrian safety with edge and 5G in the city of Vienna. The analytics application runs a quantized version of the SSD

Table 6.8: Edge node types used in the experimental setup, technical details and inference latency benchmarks for each node type.

Node label	**Node type**	**CPU**	**RAM**	**# of nodes**	**Edge TPU**	**Inference/frame [ms]**
A	Raspberry Pi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4GB	2	yes	17.81
B	Raspberry Pi 4	Quad-core Cortex-A72 (ARMv7) at 1.5GHz	4GB	1	no	250.74
C	Raspberry Pi 3 B+	Quad-core Cortex-A53 (ARMv7) at 1.4GHz	1GB	8	no	500.62

Table 6.9: Main characteristics of datasets for SEA-LEAP evaluation.

Dataset name	**Frames**	**Size [MB]**	**Size/frame [MB]**	**Dimensions**
Intrasafed	600	91.4	0.15	1280x720
Penn-Fudan	60	25.2	0.42	various
Sherbrooke	1800	154	0.09	800x600
René-Lévesque	3600	1011.8	0.28	1280x720

MobileNet v2 model [85], a lightweight and pre-trained convolutional neural network (CNN) based object detection. We dockerized the object detection logic and expose it as a service running in a container. Docker images for all node types, with and without edge TPU attached (Coral USB Accelerator enabling high-performance neural network inference), are publicly available on the Docker hub repository[2], while the *SEA-LEAP* implementation is accessible on the GitHub repository[3]. Further, we used a PostgreSQL database running in a docker container to store metadata, i.e., location-related details about existing datasets.

6.3.3 Input Datasets

We evaluate proposed approaches using datasets typically used in computer vision analytics applications such as object detection and recognition [124]. To perform a complete evaluation, we select datasets of a different average size of image files, which allows having a wide diversity in terms of resolution, dimensions, and color depth. The main characteristics of datasets are presented in Table 6.9. For each dataset, we show the average size-frame ratio (γ) calculated as $\gamma(d) = size(d)/no\_frames(d)$, impacting *SEA-LEAP* placement optimizer (explained in Section 6.3.5).

- Dataset *Intrasafed* comes from the *InTraSafEd 5G* project, containing sampled video frames from the chosen Vienna's intersection used for the real-time detection of critical situations and to support drivers in avoiding accidents. The frames are taken by traffic cameras and show critical situations where objects like pedestrians, cyclists, and pets, can appear in drivers' blind spots when turning on intersections.
- Dataset *Penn-Fudan* comes from an image database used for object detection and recognition on areas around campus and streets around the University of Pennsylvania and Fudan University [179]. Selected frames represent various image qualities and angles of captured objects (pedestrians, bikes, and cars).
- Datasets *Sherbrooke* and *René-Lévesque* come from the cameras monitoring different intersections, used for detecting and tracking multiple objects of various types in outdoor urban traffic surveillance [180]. Selected image frames represent different camera angles and resolutions, namely, a low camera monitoring cars, trucks, and pedestrians moving at an intersection (*Sherbrooke*) and a high camera covering three intersections with cars and bikes (*René-Lévesque*).

6.3.4 Testbed Configuration

In the experimental setup, we emulated a real-world system from our *InTraSafEd 5G* project, where node communication is handled by the MQTT broker, communicating to distant edge nodes deployed on traffic lights near a short-range cellular base station. Since

[2]https://hub.docker.com/r/ilujic/inference-arm32v7/

[3]https://github.com/lujic/sea-leap

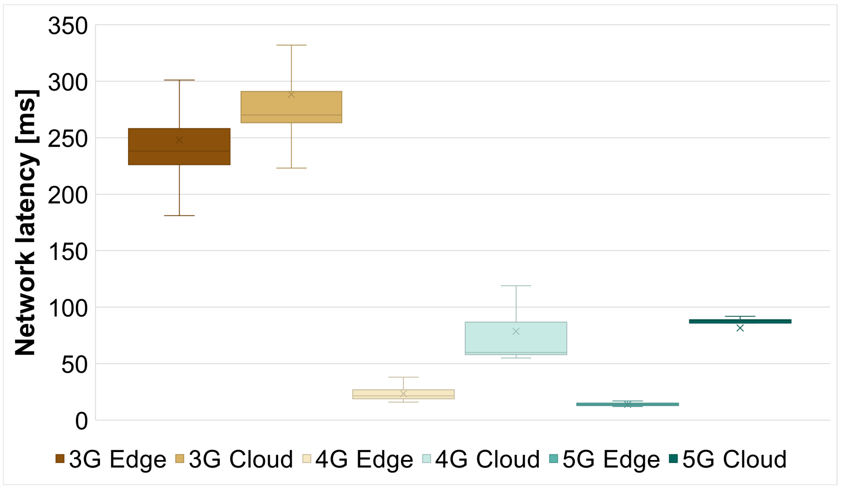
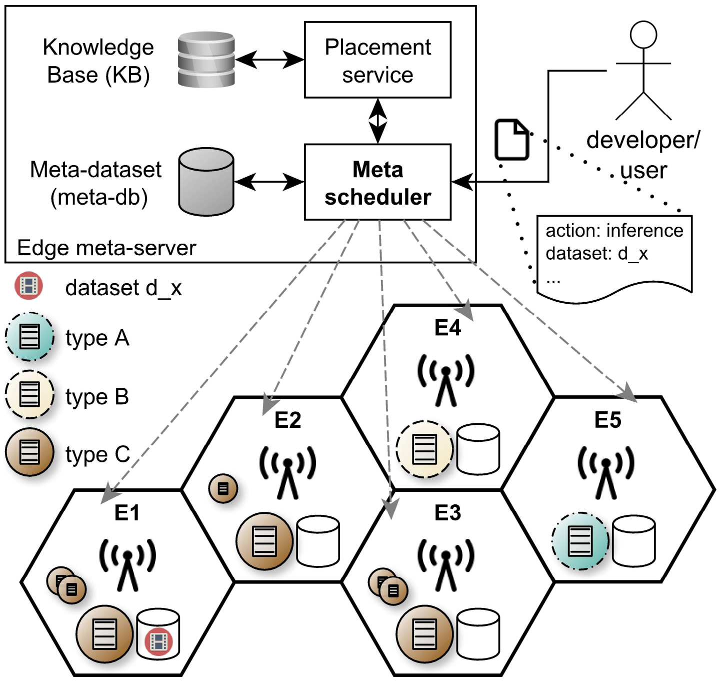


Figure 6.16: Network latency for cloud/edge meta-server placement.

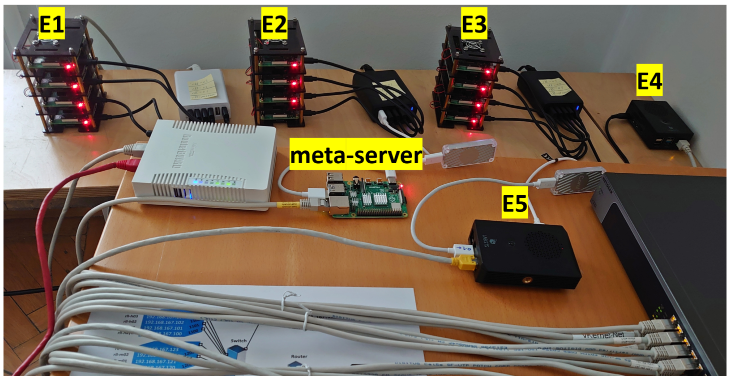
latency is a critical requirement for our scenario, we evaluated the latency of deploying the broker either at the edge (inside the TU Wien's infrastructure) or using cloud service (hosted on MyQttHub). Network latency evaluation is shown in Figure 6.16. We can see that edge deployment significantly reduces the latency (by 14.01%, 70.18%, 83.04% on average for 3G, 4G and 5G, respectively), making the edge meta-server placement as the best option for our setup.

Emulating and extending this real-world scenario, Figure 6.17 shows our testbed configuration as well as an initial setup based on different edge sites (E1-E5). Edge sites represent small cells in a cellular network, featuring short-radius coverage of a small cell base station (as used in [171]), providing specific network connection types. Each site can contain one or multiple edge clusters, where our setup contains multi-node (i.e., E1 and E3 including 3-node clusters, E2 including 2-node cluster) and single-node (i.e., E4 and E5) Kubernetes clusters. Edge meta-server represents a more reliable node or edge micro data center, able to communicate with edge nodes from different sites. Meta-scheduler receives from a user the description of an analytics request with a specific dataset.

We evaluate our emulation-based approach with the containerized analytics application, where as a baseline, we measured inference times on the real-world datasets using our physical edge nodes, as described in Table 6.8. For each node type, we show the average inference time per single image. Results are averaged over 100 image frames for statistical significance since by adding more images the differences in inference time show a deviation of $39\mu s$ on average. These results are saved in the KB, which is used by Algorithm 8 for selecting the node which minimizes the latency of edge analytics placement. Further, Table 6.10 shows different latency and bandwidth measurements obtained using standard *iperf* application. The representative values are weighted averages of bandwidth collected on different network types in a suburb area of Vienna from the *InTraSafEd 5G* project and will be used in the placement optimizer (Section 6.3.5).



(a) The emulation-based scenario and network topology.



(b) Edge infrastructure overview.

Figure 6.17: SEA-LEAP testbed configuration.

Table 6.10: Network latency and bandwidth benchmark (Vienna's suburb).

Network type	Latency [ms]	Bandwidth [Mbps]
3G	247.92	8.81
4G	23.44	41.43
5G	13.83	66.55

6.3.5 SEA-LEAP Evaluation

Static placement evaluation

Based on the *GuideMe* action (Section 5.2.5), the *SEA-LEAP* placement optimizer will enable the execution of the user's request on a node storing the required dataset, i.e., without considering alternative candidates (considered in Section 6.3.5). In the case of multiple locations storing the dataset, a node showing better performances (node type with a lower inference benchmark observation) will be prioritized. Otherwise, the algorithm will randomly select one of them. However, to enable the analytics execution on a target node using the requested dataset, the meta-scheduler needs to add a set of placement-specific details into a Kubernetes deployment file.

In our design, the meta-scheduler already stores different deployment templates that will be adapted with a specific set of information such as the node location, appropriate container image of the analytics application, and the dataset path on the target node (using *hostPath* as a volume). A simple example of an adapted deployment file is showed in Figure 6.18. Based on specific lines from this description (i.e., the one including keyword *nodeName*), a distant master node will know where to place the analytics application in its cluster using the default scheduler. Beforehand, the edge meta-server is created as a single-node Kubernetes cluster and enabled to communicate to multiple clusters, using so-called Kubernetes configuration files with needed details (e.g., IP addresses of master nodes from our testbed edge sites).

Followed by a meta-scheduler command to process a certain input dataset on the exposed analytics application, the obtained results can be forwarded back to a user. That said, the proposed *SEA-LEAP* meta-scheduler is designed as a new service that can be used on top of existing schedulers as a feature in different edge scenarios that require data locality-aware analytics placement.

```yaml
...
spec:
  containers:
  - name: inference
    image: inference-notpu:latest
    ports:
    - containerPort: 5000
    volumeMounts:
    - mountPath: /data
      name: input-data
  volumes:
  - name: input-data
    hostPath:
      path: /data
      type: Directory
  nodeName: E1-m01
```

Figure 6.18: SEA-LEAP deployment YAML file example.

Adaptive data movement evaluation

In this experiment, we want to evaluate the *FollowMe* action (see Section 5.2.5), *SEA-LEAP* placement optimizer will consider alternative location candidates different than the initial node storing the required dataset, and select the option with the lowest total latency. Thus, the placement algorithm (Algorithm 8) estimates the total latency (based on details from the knowledge base KB) including the transfer of requested data from the source to an alternative node location.

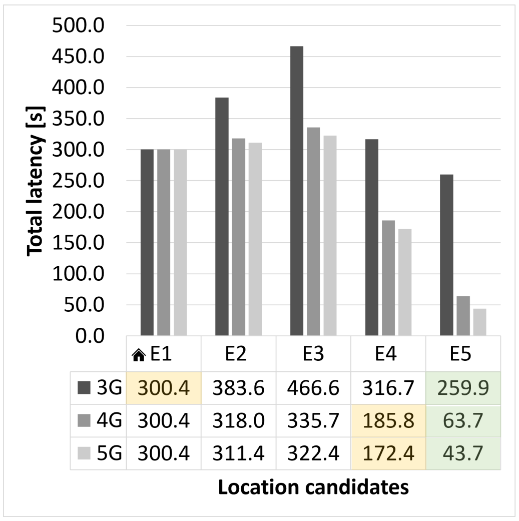
Figure 6.19 shows the results of the *SEA-LEAP* placement optimizer applied to each dataset from Table 6.9. In this representative example, the source node location of a dataset is set to the edge site E1. For that reason, the source location from E1 represents at the same time an initial candidate for analytics placement. Other alternative candidates will include additional network latency due to needed dataset transfer. Green and yellow shaded locations show first and second-best candidates, respectively.

For dataset *Intrasafed* (see Figure 6.19a), the selected appropriate node location for analytics placement in each network type results in moving the dataset from the source node storing the dataset (to an alternative candidate location from E5). In all cases of 3G (low), 4G (medium), and 5G (high) network conditions, we are able to decrease the total latency by 13.47%, 78.81%, and 85.46%, respectively, by moving the dataset to a candidate location in E5.

For dataset *Penn-Fudan* (see Figure 6.19b), in low network conditions, the selected appropriate node location for analytics placement will be in the source location storing the dataset, while for medium and high network conditions the total latency becomes 47.77% and 66.14% lower by moving dataset to an alternative location candidate. At the same time, even in the scenarios where the most appropriate location cannot host the analytics application or the dataset for different reasons (e.g., limited capacity, high failure probability), selecting the second-best candidate can still achieve 17.44% and 29.70% lower total latency by selecting a location candidate in E4, for medium and high network conditions, respectively.

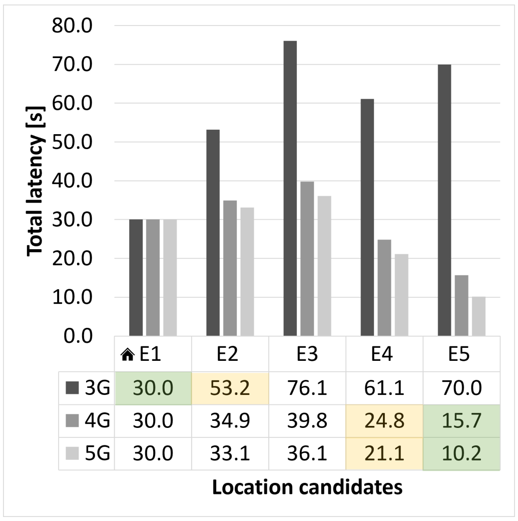
For dataset *Sherbrooke* (see Figure 6.19c), for all networks types, moving dataset can bring 49.86% (3G), 86.54% (4G), and 90.28% (5G) lower latency than in the source node location initially storing the dataset. The dataset *René-Lévesque* (see Figure 6.19d) with the largest number of frames and overall size can benefit from better network conditions, achieving 63.92% and 76.20% lower total latency for medium and high bandwidth availability, respectively.

To evaluate the optimizer's applicability to near real-time systems, we measure its runtime, included in the total latency, averaged over 100 times for statistical significance. We observe an average runtime of 1.41ms for the real testbed (see Figure 6.17). We also evaluate average runtime by increasing the number of candidate nodes up to 100, resulting in 13.89ms and 26.29ms respectively with 1 or 2 source node locations.



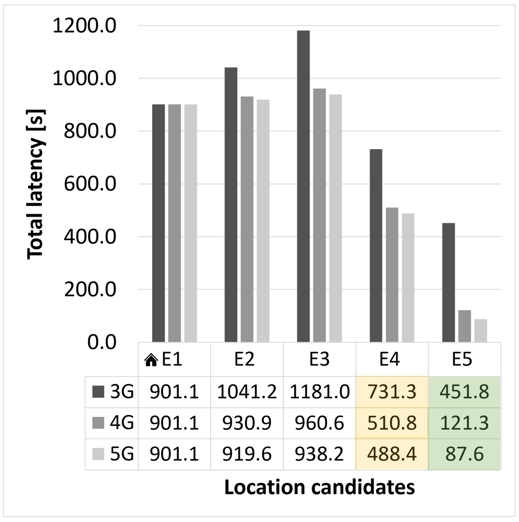
	E1	E2	E3	E4	E5
3G	300.4	383.6	466.6	316.7	259.9
4G	300.4	318.0	335.7	185.8	63.7
5G	300.4	311.4	322.4	172.4	43.7

(a) Intrasafed



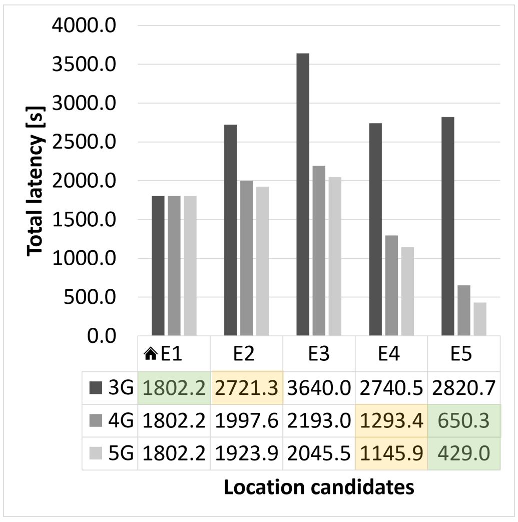
	E1	E2	E3	E4	E5
3G	30.0	53.2	76.1	61.1	70.0
4G	30.0	34.9	39.8	24.8	15.7
5G	30.0	33.1	36.1	21.1	10.2

(b) Penn-Fudan



	E1	E2	E3	E4	E5
3G	901.1	1041.2	1181.0	731.3	451.8
4G	901.1	930.9	960.6	510.8	121.3
5G	901.1	919.6	938.2	488.4	87.6

(c) Sherbrooke



	E1	E2	E3	E4	E5
3G	1802.2	2721.3	3640.0	2740.5	2820.7
4G	1802.2	1997.6	2193.0	1293.4	650.3
5G	1802.2	1923.9	2045.5	1145.9	429.0

(d) René-Lévesque

Figure 6.19: SEA-LEAP placement calculation of node location candidates in different edge sites, driven by *GuideMe* and *FollowMe* actions. It is based on network connection benchmarks from a real-world edge location. For all cases, the source location of the dataset is set to E1.

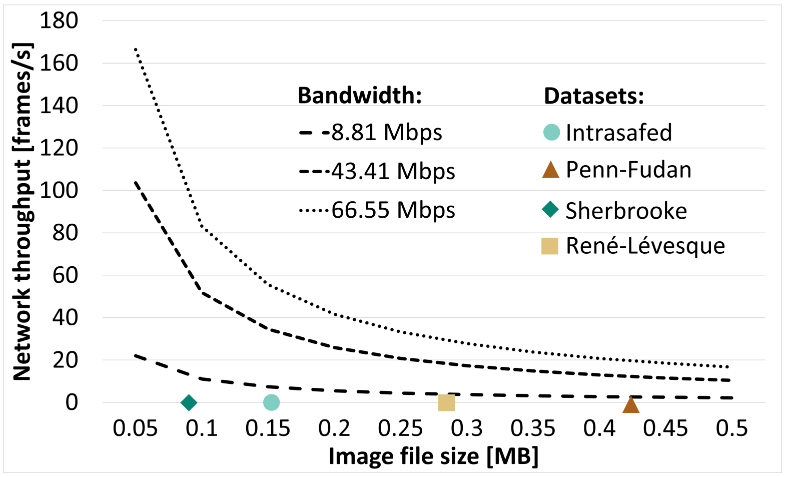
Discussion

The results show benefits of the proposed *SEA-LEAP*, it allows (i) an autonomous placement of analytics requests, and (ii) the self-adaptation to data locality by considering both network and node candidate characteristics. For example, although node types A and B from our experimental setup have respectively 28x and 2x lower inference time per frame compared to the source node type C (see Figure 6.17), not all datasets benefit from their movements if available bandwidth is low.

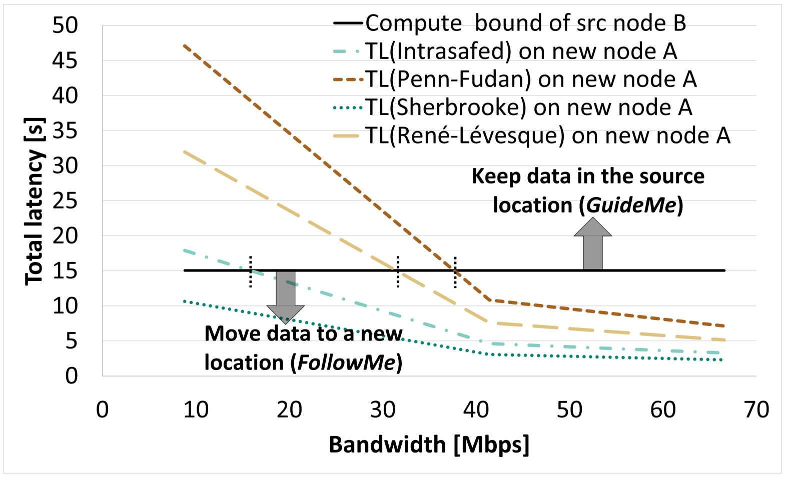
Despite different network characteristics of different edge sites, the network performance will depend on the available network bound of a node location storing the dataset. As described in Section 5.2.5, the total latency of placing analytics to new location candidates will be also impacted by other factors such as network latency, number of hops, analytics execution time, and dataset size. Still, based on the experimental results, moving the dataset to another location we can reduce total latency by 65.85% on average. To determine these cases, Figure 6.20 shows the *SEA-LEAP* placement decision rule and which aspects have strong impacts on it. The estimated total latency of analytics placement is largely affected by two main aspects, namely, network bound (available bandwidth) and compute (node performance) bound.

Figure 6.20a shows the relation between the average image file size and network throughput. We see that the higher the bandwidth, the higher is the network bound for transferring a certain number of image frames for a specific dataset. However, this relation does not hold for the compute-bound of a specific node candidate in this context. This is because the target object detection application resizes each input frame to the same dimension due to performance reasons. Since resizing does not affect the accuracy of object detection, the average inference per frame (i.e., compute-bound) will stay the same, independent of the dimension or size of a single image frame. Consequently, the computation time for a specific edge node type will depend exclusively on the number of input image frames.

For example, in Figure 6.20b, the number of input image frames for each dataset is set to 60, while the initial candidate for analytics placement is a source node type B, and only alternative node candidates with lower inference time per frame are considered, i.e., node type A. The solid black line represents the compute-bound of the source node B as the baseline, i.e., the total latency of running requested analytics on the dataset in the source location is equal to $\sim 15s$ $(60 \cdot 250.74ms)$. Dashed lines represent the estimated total latency of running the analytics on the datasets in alternative node candidates, including data transfer over different network characteristics with two hops. Depending on the available bandwidth in the source location, the self-adaptive SEA-LEAP placement optimizer will decide whether to place analytics to (i) the initial location storing the dataset (*GuideMe*) for all results above the baseline, or (ii) a new location where the dataset is also moved (*FollowMe*) for all results below the baseline. Our self-adaptive solution shows that considering data locality in edge analytics placement can significantly improve overall analytics requests execution time, and thus impacting the timely placement of on-demand analytics applications.



(a) Network bound



(b) Placement decision

Figure 6.20: SEA-LEAP placement decision. Subfigure (a) shows network bounds for various image file sizes. Subfigure (b) shows a borderline of placing analytics between the source node B and a new node A, among different datasets (60 frames) and available bandwidths with two hops.

CHAPTER 7

Related Work

In this chapter, we provide a comprehensive overview and discussions of the state-of-the-art on performing data analytics in heterogeneous and resource-constrained edge environments. We investigate related work and limitations of previous solutions considering five main aspects, namely, (i) IoT data and resource-limited edge systems (Section 7.1), (ii) edge data management (Section 7.2), (iii) elastic edge data and storage services for decision making (Section 7.3), (iv) time-critical edge analytics systems (Section 7.4), (v) data locality-aware edge analytics placement (Section 7.5).

7.1 IoT Data and Resource-limited Edge Systems

Regarding integration of IoT sensors and cloud computing systems, Villari et al. [9] discussed scenarios of large streams of data coming from IoT sensors while at the same time IoT systems require a short response time of data analytics. Further, they provide an overview of four macro objectives in interconnecting IoT and cloud concepts including integration, configuration, communication, and security. Since IoT devices are usually not equipped with powerful resources needed for data processing, edge computing has been proposed as a novel methodology for exploitation of computation, storage and networking resources across cloud boundaries, as discussed also in Bittencourt et al. [19] and Baccarelli et al. [27].

De Maio et al. [181] discussed enabling IoT and mobile devices to extend their computing power and storage by offloading computation or data to more powerful servers on single-hop proximity such as edge nodes. Lewis et al. [182] proposed the first catalog of architectural tactics for cyber-foraging. Cyber-foraging depicts architecture elements helping to extend battery life and improve computational capabilities of mobile sensors and devices in the future. To enable latency-sensitive applications and to minimize data transmission over the network, movement of functionality from the cloud to the edge devices requires a reliable data management system with the support from IoT layer.

Recent technological advances have disrupted the current centralized cloud computing model by moving cloud resources close to users, as it is examined in Villari et al. [21]. Proposed osmotic computing depicts the dynamic management of services and microservices across cloud data centers and edge nodes. A comprehensive analysis of the advantages of having the intermediate layer consisting of these edge micro data centers close to the network edge has been done also by Mehta et al. [183]. The results show that for data-intensive applications it is inevitable to place data processing tasks close to the source of data. Additionally, this approach can reduce energy consumption affected by data transmission, storage and processing of big data. Edge computing can provide a distributed infrastructure at the edge of the Internet where edge nodes have limited resources. Bittencourt et al. [14] examined scheduling strategies to cope with different application classes according to demands coming from mobile users. Although there are many studies in processing IoT data in resource-constrained edge systems, the state-of-the-art still lacks contributions in efficient data-centric services for near real-time edge analytics.

7.2 Edge Data Management

Regarding the proposed edge data management mechanisms in Chapter 3, we provide the related work through four parts, namely, (i) time-series sensor data and predictive analytics, (ii) recovery of incomplete datasets, (iii) edge storage management, and (iv) edge data management systems and frameworks.

Time-series sensor data and predictive analytics

In the last few decades, time series forecasting gained much attention due to necessary predictive analytics that rely on the increasing amount of time-stamped measurements. Herbst et al. [111] provided an extensive overview of existing time series forecast methods, including a self-adaptive approach for optimized forecasting method selection based on users' forecasting objectives. Regarding contribution in time series forecasting and statistical research, Hyndman et al. [184] proposed a forecast package that includes implementation of automatic versions of most popular forecasting methods such as Autoregressive Integrated Moving Average (ARIMA) and ExponenTial Smoothing (ETS). Considering the proposed approaches in this thesis, the use of different forecasting methods is motivated by the fact that forecast accuracy depends on the characteristics and behavior of data points preceding each gap. Therefore, we considered methods like ARIMA and ETS, and TBATS, described in [78, 113], which are also suitable for near real-time decisions in IoT since they do not require constant user interaction. We use these methods for adaptive recovery of multiple gaps, due to the near real-time requirements of systems, running on self-contained edge nodes. Regarding data mining in time series, Ding et al. [185] provided a comprehensive validation of representation methods for dimensionality reduction and similarity measures in time series datasets.

Recovery of incomplete datasets

Wellenzohn et al. [186] propose Top-k Case Matching (TKCM) for imputing missing values in time series. Based on principles of correlation between time series, it is possible to impute missing values in streams of data by comparing incomplete series with a set of reference time series, and additionally allowing phase shifts. In [187], a program for imputing missing data in multivariate time series is proposed for handling missing data from stationary processes. Since most of the IoT measured processes are non-stationary, the proposed approach cannot be easily applied and can be inefficient for univariate time series. Also, none of the proposed approaches allows recovering multiple gaps separately giving the possibility to select different models for recovery. Honaker et al. [188] describe the reconstruction of missing values in time series cross-section data proposing multiple imputation models, focusing only on imputing a small number of public-use datasets in surveys with many variables. Duan et al. [189] design an adaptive model selection based on Hidden Markov models aiming to constantly validate mean percentage error in a prediction algorithm, resulting in higher accuracy in time series of stock price prediction. The approach constantly validates mean percentage error to find the best model. However, machine learning-based models for the prediction of missing data are time-consuming and require training, making these approaches difficult for data recovery on most of the resource-constrained edge devices.

Furthermore, in [190], univariate imputation is used for air pollution data, but this approach target gaps of fixed size. The challenge of recovering missing data has been investigated by many researchers, providing methods relying on cubic interpolation [191], Singular Spectrum Analysis [192] or Lomb-Scargle method [193]. However, these works either have not been validated on IoT sources or propose approaches targeting only specific cases of time series. In this thesis, we show how to combine different forecasting techniques for the recovery of different gaps affecting both, the accuracy and performance of recovering processes. Additionally, the proposed approach allows users to specify preferred conditions and define other algorithms for recovering incomplete data. Accordingly, our proposed edge data management framework in this thesis is designed in a generic way, such that depending on the application context and sensor data characteristics, different methods can be utilized for both data recovery mechanism and adaptive edge storage management.

Edge storage management

Many applications can benefit from edge computing, but existing solutions are usually not made for the edge layer, leaving many research issues for novel data management approaches. The problem of reducing data transmission on the edge has been surveyed by several works. Papageorgiou et al. [105] proposed a solution for network-edge data reduction targeting IoT devices, but does not consider storage problems and latency requirements of IoT applications. It automates the switching between different data handling algorithms at the network edge and evaluates specifically for IoT devices and datasets. Further, Abu-Elkheir et al. [194] focus on data management solutions, proposing a comprehensive description of components of IoT data management framework, focusing on data collection, storage, and processing. The authors mostly target design elements for efficient data handling. Ukil et al. [106] propose a dynamic compression-based technique for sensor datasets. Other existing works, like [195], focus instead on data storage structure, memory allocation strategy and data compression in order to efficiently use storage capacity. All these works try to reduce datasets size by maximizing compression of generated data while reducing information loss, but do not consider the impact on real-time needs for sensor-based monitoring systems and corresponding actuators.

According to Sensor-Cloud Infrastructure [104], it is possible to employ service innovation to accelerate decision-making. These challenges have been considered from traditional IoT and cloud perspectives [8, 31], but cannot be used due to edge limited storage. Existing works, like [196, 197] propose data staging for edge nodes including a control loop where the data staging manager takes care of retrieving and caching necessary data from the cloud. Proposing predefined fetching algorithms, these approaches are suitable for a scenario where data are available to the mobile devices when it is needed, but it is not suitable for our case, where we need to obtain fast and accurate responses. Plus, their contributions consider neither the performance of storage limited edge nodes, nor the possibility of targeting actuators with reliable predictions. In comparison with traditional batch data analysis, data stream classification has many open issues. In [198], holdout accuracy is estimated using prequential accuracy. In other works, two types of methods are used: a sliding window with the most recent observations or fading factors that weigh observations using a certain decay factor [199, 198]. However, these approaches do not consider the prediction of upcoming data streams together with historical data. Our approach proposes an adaptive algorithm that constantly checks newly received data and then together with the remaining old data performs proposed principles for efficient edge storage management.

Edge data management systems and frameworks

Montella et al. [200] target the resilience and privacy of sensitive data in delay tolerant networks. Other solutions target resilient edge systems for IoT data management and cloud, focusing on system resource management [4], location-based energy control [2], network congestion [100], security [101] or data integrity [201]. Others [202, 9, 107], describe the interplay and communication models for cloud and IoT resources due to growing data streaming and increasing latency issues of smart sensors. These works do not discuss critical decision-making processes at the edge.

Regarding industrial frameworks, collection and data analysis at the edge is the basis of industrial cloud platforms such as AWS IoT Greengrass[1], focuses on event-driven computing and performs data storage on the cloud, rather than on the edge; Azure IoT Edge[2] employs containers to package modules and custom logic at the edge. AWS IoT Analytics[3] offers remote device management, optimized IoT data storage, and time

[1]https://aws.amazon.com/greengrass/
[2]https://azure.microsoft.com/en-us/services/iot-edge/
[3]https://docs.aws.amazon.com/iotanalytics/

series analytics, enabling end-to-end workflow automation for large amounts of data and connecting IoT devices with cloud applications. Eclipse Kura[4] represents a reference IoT Edge framework for building IoT gateways, incorporating networking protocols, and data services, allowing connectivity of IoT devices to their cloud platform. These approaches focus on fully managed workload services instead of edge data management on limited storage and adaptive data recovery with multiple techniques.

7.3 Elastic Edge Data and Storage Services for Decision Making

We surveyed existing works on elastic edge data and storage services for decision making from three aspects, namely, (i) system viewpoint, (ii) application viewpoint, and (iii) design viewpoint. Further, we also show a comparative overview of relevant works regarding analyzed requirements from Section 4.

System viewpoint

Various system operations have been used to build efficient edge storage. For example, Ali et al. [35] discussed a data life cycle while investigating the optimization of storage mechanisms and data management system design for the IoT. To improve the cost and efficiency of the data collection and robustness in IoT systems, Sinaeepourfard et al. [36] proposed distributed data storage subsystems instead of centralized data management solutions. The concept of data-centric communication [34], looked for enhancing network connectivity with local storage services, proposing different management strategies to handle stored data from a system viewpoint.

Application viewpoint

According to [203], it is possible to assign dynamic routes for IoT data based on application context information, considering four objectives, namely, lifetime, delay, reliability and data delivery. However, in this approach, only the network viewpoint is examined. He et al. [123] proposed a Storage-as-a-Service model where unused storage space can be shared as a cloud-based service for different applications. In most IoT applications, collected data are sent to a remote server for storage and analysis [120, 204]. Existing controllers for auto-scaling of resources in cloud data centers mostly rely on a set of predefined rules and static threshold-based policies [205]. Jamshidi et al. [206] proposed a self-learning controller of rules adaption for cloud-deployed applications based on reinforcement learning. However, due to the resource limitations at the network edge, it would be infeasible to perform compute-intensive training processes.

[4]https://www.eclipse.org/kura/

Design viewpoint

Some of the high-level requirements for dealing with a new design of the edge storage service are in line with IoT common design principles [207], but such IoT common principles do not dig into edge storage services and analytics scenarios. Weyns et al. [208] discusses architecture designs for IoT management and adaptation, but not about the design for edge data services. However, they have certain tools to evaluate the design models, while we have focused on requirements, architectural models and engineering principles. High-level self-adaptation for edge computing has been discussed in [209]. But it is too high-level and does not focus on edge storage services for application contexts. Table 7.1 summarizes a comparative overview of other works that address relevant edge storage-oriented characteristics. In our approach, we bridge the aforementioned gaps leading to customized software-defined elastic edge storage services.

Table 7.1: An overview of relevant works for architecturing elastic edge data services.

Work	**Edging system operations**	**Edge/system characterization**	**Application context**
[36]	√		
[210]			√
[33]		√	
[35]	√		
[34]	√	√	
Our approach	√	√	√

7.4 Time-Critical Edge Analytics Systems

We address existing works considering the proposed system for increasing traffic safety with real-time edge analytics, as described in Section 5.1. Danish et al. [40] addressed multi-object tracking in urban settings using a network of edge devices. However, the discussed use case only considered privacy, accuracy and performance. Liu et al. [159] propose EdgeEye, a service enabling the development and execution of video analytics applications. Luo et al. [158] show EdgeBox, an architecture for improving automatic event detection in edge near real-time video analytics. Anandhalli et al. [211] discussed video image processing algorithms for real-time tracking and counting vehicles using edge devices. However, these works either do not consider strict low-latency requirements, or have analytics placement using cloud, or target different problems than traffic safety.

Vehicular networks and Vehicle-To-Everything (V2X) communication capabilities based on 5G are investigated in [38]. It introduces a novel system design, targeting vehicles and roadside infrastructure with V2X capabilities. To increase traffic safety, the concept of [39] looked at the problem from a perspective of intelligent driving vehicles and real-time lane-change recognition. We focus on real-time edge analytics and high transmission mobile networks in designing a real-world prototype for detecting critical situations of pedestrians and cyclists appearing in drivers' blind spots on intersections.

7.5 Data Locality-Aware Edge Analytics Placement

Researchers proposed different frameworks and solutions for the placement of edge data analytics, regarding the proposed approach in Section 5.2. We separate related work into three parts, namely, (i) analytics placement and data management, (ii) latency-aware scheduling and data locality, and (iii) placement decision and edge data analytics.

Analytics placement and data management

Analytics placement at the edge has been discussed in several recent works. Aazam et al. [22] propose a service-oriented resource management framework for fog computing focusing on service reliability, different types of service customers and their unpredictable relinquish probabilities. Liu et al. [159] propose EdgeEye, a service enabling the development and execution of video analytics applications. EdgeBox [158] is an architecture to improve automatic event detection in edge near real-time video analytics. However, these works limit placement to specific and centralized cloud/edge locations. Satyanarayanan et al. [124] discuss the decentralized and federated edge infrastructure, focusing on a scalable approach to perform data collection and video analytics at the edge of the network. Still, these approaches do not consider data locality and adaptive analytics placement.

Current data management approaches adopt storage services configured toward centralized data aggregation [212] or geo-distributed data storage [213]. Diène et al. [214] survey existing solutions for IoT data management, including techniques for managing IoT data gathering, their transfer and storage. In [73], the relationship between resource availability and accuracy of edge-cloud video analytics are investigated, focusing on high computational costs and CPU demands, but without considering data locality. In [162], Firework system is described facilitating distributed data processing requests at different connected edge nodes, considering only specific locations.

Latency-aware scheduling and data locality

Yi et al. [215] addressed the offloading of computation-intensive tasks on edge nodes as an optimization problem. The proposed scheduling approach minimizes latency by static offloading of dependent tasks according to input data. In [160] latency-aware placement of data stream analytics applications is proposed, while Gupta et al. [161] performed low-latency data management over geo-distributed and heterogeneous edge infrastructures. We focus on a latency-aware placement of on-demand analytics using data locality.

The exploitation of data locality has been considered by other works in literature. For example, Venugopal et al. [163] discuss the concept of Semantic Cache, which employs a caching technique for edge analytics while reducing latency compared to cloud-only inference. In [216], the spatio-temporal locality of analytics is used to improve workload balancing between edge and cloud servers. Other works for analytics placement exploit data locality considering the edge-cloud workload balance perspectives [41], the trade-off between the resource usage and query accuracy [42], or the fairness of cloud resource allocation [43]. However, these works either do not consider the autonomous placement of on-demand analytics or focus on different aspects than minimizing requests execution time. We bridge these gaps by considering data locality in the self-adaptive placement of on-demand edge analytics.

Placement decision and edge data analytics

The service placement decisions and trade-offs between local execution and remote execution are discussed in voluntary-based computing environments [217] and micro-cloud infrastructures [218]. Further, Hamrouni et al. [219] proposed replica management and replica selection strategies in data grids, based on data mining techniques. Furthermore, concerning network service chaining at the edge, Kathiravelu et al. [220] looked at latency-aware service execution through software-defined approaches. Most of these placement decisions are based on either network performance or resource allocation aspects. In this thesis, we target a data locality-aware, generic and self-adaptive mechanism that facilitates the edge analytics deployment across different edge infrastructures, while minimizing overall execution time for on-demand analytics.

In [221], big data processing and analytics at the edge is discussed from the point of view of energy-efficient edge scheduling and the impact of energy on QoS. Luo et al. [222] established a framework for offloading tasks to the edge, focusing mostly on minimizing the delay and cost of the computation. In [223], the bandwidth-adjustment problem in video-streaming is addressed, proposing a framework to reduce network traffic and adapt to conditions of mobile users. Other works address caching of mobile big data traffic at the edge [28], the container migration problem of mobile application tasks [224] or enable federated query evaluations across cloud and fog nodes to reduce communication [225], but data locality-aware analytics placement is not considered.

CHAPTER 8

Conclusion

In this chapter, we provide conclusions from the research done as well as the limitations and constraints of the proposed solutions. We also outline possible future directions.

8.1 Summary

Edge computing has been proposed as a solution for near real-time decision-making in rapidly growing IoT systems. Unlike performing data processing in highly reliable and centralized cloud data centers, there are sustainability and trustworthiness concerns regarding edge data analytics. This is because of the edge-specific problems such as limited computational resources of edge nodes (i.e., edge servers, micro data centers, single-board computers) and decentralized data locations posing difficulties for time-critical analytics execution. Further, edge analytics faces us with the problems of making accurate and near real-time decisions based on limited and often incomplete data. In this thesis, we aim to improve decision-making processes for IoT systems proposing data-centric services and approaches for edge infrastructures, making foundations for sustainable and trustworthy edge data analytics.

First, to increase the accuracy of data analytics, particularly for time-sensitive IoT applications, we introduced *EDMFrame*, a framework for edge data management featuring (i) a mechanism for recovery of multiple gaps using both STR and PRM-based MTR approaches, and (ii) a mechanism for accurate decision-making using limited storage resources. Considering the self-adaptive data recovery, we are able to efficiently remove outliers, detect and recover gaps of various lengths by incorporating recovery cycles in incomplete datasets. Experimental results showed that *EDMFrame* is able to reduce both the forecasting error and improve the overall running time of the recovering process.

Second, based on the presented architecture model for edge data storage management, we proposed the adaptive algorithm to cope with constant streams of data and limited edge storage. The proposed approach is able to continuously monitor forecast accuracy and capture stable forecast accuracy clusters. Based on the proposed adaptive algorithm, our simulation results showed that we are able to reduce the amount of edge-stored data while satisfying demands for forecast accuracy in predictive analytics, and thereby showing potential for saving limited edge storage space.

Third, IoT data-intensive applications pose big challenges to satisfy strict requirements for timely and accurate data-driven decision-making and at the same time relying on resource-constrained edge nodes. Thus, we aimed to dynamically define highly customized optimization strategies to handle incoming data from different perspectives. Based on a detailed analysis of edge storage requirements, we proposed a novel architectural design incorporating elasticity in edge data services. Considering dynamic edge data workloads and strict latency and accuracy requirements, we presented engineering principles and how they can potentially be implemented. In this context, proposed approaches can help researchers and practitioners to utilize resulted dependencies for edge data services at runtime, achieving more sustainable edge analytics and data-driven decision-making.

Fourth, by deploying *InTraSafEd 5G* system we addressed the challenge of using the potential of edge data analytics and modern communication technology (such as 5G) in a real-world scenario, to increase traffic safety. We target critical situations on intersections, e.g., pedestrians and cyclists appearing in drivers' blind spots. After analyzing design choices, we developed and evaluated a real-time edge analytics prototype. The proposed system enables real-time (i) detection of critical situations by running object detection on lightweight edge nodes; (ii) delivery of resulted critical information to vehicle drivers with 5G. The proposed system is designed to preserve privacy and ensure low latency with other network types, representing a promising step for trustworthy edge applications and communication technologies to support real-time decision-making.

Finally, we looked at executing on-demand analytics requests and critical challenges of data locality. We proposed *SEA-LEAP*, a Self-adaptive and Locality-aware Edge Analytics Placement framework to (i) identify locations of requested input datasets, and (ii) determine the most appropriate edge computational node. *SEA-LEAP* includes a tracking mechanism for event-triggered data management and registration of data movements. On top of it, we presented a generic control mechanism featuring a meta-scheduler and placement optimizer. *SEA-LEAP* allows self-adaptive, on-the-fly placements of on-demand analytics based on data locality, and minimizes overall request execution time by performing adaptive data movements. We evaluated *SEA-LEAP* by considering video analytics using our physical and heterogeneous edge infrastructure and obtained benchmarks. Results showed benefits for users and developers, automating the placement of analytics requests and reducing the total latency for certain network and node characteristics. We believe that *SEA-LEAP* is a valuable step towards sustainable and trustworthy on-demand analytics execution for edge multi-cluster or hybrid environments.

8.2 Limitations

In this section, we state limitations and constraints of the work presented in this thesis.

Time sensitive edge analytics systems

Regarding edge data management and incomplete data, the recovering process depends on multiple elements including the number of missing/invalid data points, gap sizes, data characteristics, and the performance of used methods. Currently, our approach is a semi-automatic approach, requiring certain predefined specifications by a user (e.g., selecting recovering methods in advance). In both data recovery and storage management mechanisms, utilized methods such as ARIMA and ETS, expect historical and regularly time-stamped data, and they are not often appropriate for some IoT streaming cases in which data generation and collection are triggered by certain events. Also, the current PRM calculation requires a predefined number of consecutive gap lengths, posing obstacles in extreme cases of big gap lengths expected.

Practical applicability and deployments

The main limitations of the *InTraSafEd 5G* system design are (i) the driver's app automatically subscribes to topics of certain intersections and (ii) the real-time notifications start showing when drivers' distance is around 100m from the critical intersection. The first limitation can cause a scalability problem in the case of a high number of intersections, while in the second, the distance parameter can vary for different intersections causing an important challenge in the real-world setup.

When considering analytics placements based on data locality, the *SEA-LEAP* placement optimizer estimates total latency for initial and alternative candidate locations based on prior obtained and managed edge-related details such as network characteristics (e.g., network types, number of hops) and analytics benchmarks on node types. Also, we assume in our setup that edge nodes have access to existing docker images of analytics applications. Still, we partially address these issues by designing *SEA-LEAP* parts as generic services, which can be easily extended to consider (i) other analytics applications and datasets (e.g., time series analytics for failure prevention in smart cities), (ii) different network topologies and characteristics, and (iii) additional conditions for filtering location candidates. In the *SEA-LEAP*, a centralized edge meta-server represents a single point of failure, which could affect *SEA-LEAP* availability. Also, in the current version, we assume a network of edge servers managed by trusted infrastructure providers that control access to edge resources as well as handle data security issues. Even though accessing metadata via the proposed meta-scheduler or through the tracking mechanism already provides access control, we believe that additional protection measures could be taken, especially in contexts where security is critical (e.g., use cases with sensitive information). Lastly, we consider only data locality, while the resource allocation can be handled by Kubernetes. As shown, our proposed approach can be integrated on top of existing systems such as Kubernetes, facilitating data locality-aware edge analytics placement.

8.3 Future Work

Based on the presented work in this thesis and stated limitations in the previous section, we look beyond the current context and outline potential future work.

In the case of a full-automatic mechanism of data recovery, based on data characteristics and without user interaction, it would be interesting to explore how to automatically trigger conditions for using different forecasting techniques. Moreover, it would be interesting to investigate the possibility of recovering incomplete data based on correlated time series. We also plan to extend our approach to scenarios with strict latency requirements, such as eHealth and intelligent traffic management systems. In the first case, we plan to use our approach for recovering data in monitoring illnesses such as diabetes and heart diseases, helping such systems to timely react to the change of patients' conditions. In the latter case, our mechanism can be used to recover datasets coming from road sensors, helping to accurately and timely react before accidents or collisions happen. Still, considering that these real-world cases can have event-triggered data collection, in our *EDMFrame* we plan to address incomplete data issues by exploring recurrent neural networks for time series with irregular timestamps (as in [226]), and look at the interpolation of ranges to allow PRM calculation for such dynamic IoT cases. As discussed, for future IoT services it is crucial to make fast decisions as in smart cities requiring distributed ML at the edge, e.g., in the case of consistent ML models that must be updated when data streams evolve over time, posing critical issues to observe correct data at the right time [119]. Hence, we plan to explore using *EDMFrame* for reliable distributed ML at the edge.

Considering edge storage management, we would like to extend our solution in different ways. It would be interesting to explore methods for dealing with multiple data sources and evaluate our application with different datasets, to show the wide applicability of the proposed approaches. Considering other application contexts (e.g., smart cities and industrial systems), we plan to further explore the design requirements of elastic edge storage services to support data-driven decision-making, and focus on implementing a prototype based on proposed principles in this thesis. Also, the combination of elasticity in compute and network resources as well as implications of involved computation resources during the runtime we left for future research.

Further, we plan to extend our two deployments of edge video analytics systems. In *InTraSafEd 5G* system design for increasing traffic safety, we plan to explore location-based subscriptions (as in [177]) to avoid the app's automatic subscription to topics of intersections when the app started. We plan to investigate the effect of distance and warning timings on drivers' brake reaction time, as in [145]. It would be also interesting to improve the resilience of critical edge processing to network/node failures (e.g., using container-based approaches for edge analytics). In the *SEA-LEAP* system design, we plan to consider the single point of failure of the *SEA-LEAP* meta-scheduler by improving its overall scalability and reliability, e,g., by using multiple instances and implementing replication strategies of meta-dataset. We also plan to investigate the scalability of the tracking mechanism by experimenting with different edge storage platforms such as Ceph, Minio, or other object storage technologies. Although the concept of accessing dataset metadata via the meta-scheduler or using the proposed tracking mechanism for data management actions can already reduce access to sensitive data, it would be interesting to further investigate the privacy and the data protection of *SEA-LEAP*, and extend the management of status data by adding an improved monitoring infrastructure.

Finally, it would be interesting to focus on collaborative data analytics and different data representations. Since edge nodes are designed to perform data processing tasks for (near) real-time analytics, cloud resources can utilize analytics results from the edge layer and perform needed batch analytics or maintain global models on top of geographically distributed edge infrastructures. Thus, it would be interesting to investigate how to enable collaborative data analytics between cloud and edge. Considering that performing data analytics on big IoT data can represent computationally expensive tasks, it would be interesting to explore how to distribute data analytics among connected edge nodes, where we expect an increase in data processing speed and better exploitation of edge computational resources. The challenge of increased data processing speed could be investigated from other aspects. In addition to classical sensor-based time series, one of the related fields can include other data representations such as symbolic data representation [185]. Symbolic data representation can reduce data dimensionality while keeping its main characteristics. Due to the increasing production of time-series data, such representation can impact new research directions of approximate analytics, further reduce communication bottlenecks, improve data transfer and storage.

Glossary

CAGR *Compound Annual Growth Rate* is the average rate of return, one of the most accurate measures to project an investment's annual growth rate over time. 1, 3

CNN *Convolutional Neural Networks* is a class of artificial neural network used in image processing for computer vision systems. It is composed of multiple layers of artificial neurons (mathematical functions) with learnable weights to extract features from images. 23–25, 96, 105

GSM The Global System for Mobile Communications (GSM) is a standard to describe the protocols for the second-generation cellular network technology known as 2G, developed by the European Telecommunications Standards Institute (ETSI). 15

hybrid environment refers to a cloud computing environment that combines or integrates on-premises infrastructures, private and public cloud services with orchestration management capabilities between these platforms. 10, 122

IoT *Internet of Things* is a system of networked physical objects, so called "things", connected to the Internet with the purpose of collecting and sharing data with other objects and systems. 1, 2, 32, 51

Kubernetes is a platform, developed by Google, representing one of the widely used open-source orchestrators that automates deployment and management of containerized applications across multiple machines. 28

LTE The Long Term Evolution (LTE) is a standard to describe protocols for the fourth generation cellular network technology known as 4G. 15

M2M *Machine-to-machine* refers to direct communication and automatic data exchange between networked mechanical or electronic devices. 17, 26

monolithic architecture is a unified architectural model to make self-contained and complex applications by composing all components in one piece. 28

non-stationarity in *time series* means that the statistical properties of data change over time, e.g., data points can include trends, seasonality, random walks. 21

QoS *Quality of Service* refers to a description or measurement of the performance of provided service, and it is usually included in the agreement between the service provider and user to guarantee a certain level of performance. 16, 54, 58

quantized Quantization refers to the process of approximating neural networks by executing the operations with low bit width numbers (i.e., integer) instead of floating point numbers. 25, 96, 103

SLO *Service-level objectives* represent objectives that must be achieved within a service-level agreement (SLA) between a service provider and service users, utilizing different performance metrics. 8, 57, 61, 70, 71

SOA *Service-oriented architecture* defines a type of software design for making services that are independent, reusable and interoperable across different platforms. 74

stationarity in *time series* means that the statistical properties (such as mean, variance, auto-correlation) of time series and the shape of its distribution are constant over time. 21

UMTS The Universal Mobile Telecommunications System (UMTS) is a standard to describe protocols for the third generation mobile cellular network technology known as 3G. 15

Bibliography

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of things (iot): A vision, architectural elements, and future directions,” *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] J. Pan, R. Jain, S. Paul, T. Vu, A. Saifullah, and M. Sha, “An internet of things framework for smart energy in buildings: designs, prototype, and experiments,” *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 527–537, 2015.

[3] P. Pyykönen, J. Laitinen, J. Viitanen, P. Eloranta, and T. Korhonen, “Iot for intelligent traffic system,” in *Intelligent Computer Communication and Processing (ICCP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 175–179.

[4] S. Oueida, Y. Kotb, M. Aloqaily, Y. Jararweh, and T. Baker, “An edge computing based smart healthcare framework for resource management,” *Sensors*, vol. 18, no. 12, p. 4307, 2018.

[5] Y. Sun, H. Song, A. J. Jara, and R. Bie, “Internet of things and big data analytics for smart and connected communities,” *IEEE Access*, vol. 4, pp. 766–773, 2016.

[6] K. A. Patil and N. R. Kale, “A model for smart agriculture using iot,” in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, Dec 2016, pp. 543–545.

[7] Transforma Insights. In Statista, “Internet of things (iot) connected devices worldwide in 2019 and 2030, by technology (in billions) [graph],” https://www.statista.com/statistics/1183463/iot-connected-devices-worldwide-by-technology/, May, 19 2020, [Online; accessed 01-September-2021].

[8] B. P. Rao, P. Saluia, N. Sharma, A. Mittal, and S. V. Sharma, “Cloud computing for internet of things & sensing based applications,” in *International Conference on Sensing Technology.* IEEE, 2012, pp. 374–380.

[9] M. Villari, A. Al-Anbuky, A. Celesti, and K. Moessner, “Leveraging the internet of things: Integration of sensors and cloud computing systems,” 2016.

[10] P. Mell, T. Grance *et al.*, “The nist definition of cloud computing,” 2011.

[11] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and internet of things: a survey," *Future generation computer systems*, vol. 56, pp. 684–700, 2016.

[12] A. Artikis, C. Baber, P. Bizarro, C. Canudas-de Wit, O. Etzion, F. Fournier, P. Goulart, A. Howes, J. Lygeros, G. Paliouras *et al.*, "Scalable proactive event-driven decision making," *IEEE Technology and Society Magazine*, vol. 33, no. 3, pp. 35–41, 2014.

[13] C. Doukas and I. Maglogiannis, "Bringing iot and cloud computing towards pervasive healthcare," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 922–926.

[14] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[15] S. Krishnan and T. Balasubramanian, "Traffic flow optimization and vehicle safety in smart cities," *Traffic*, vol. 5, no. 5, 2016.

[16] C. Keles, A. Karabiber, M. Akcin, A. Kaygusuz, B. B. Alagoz, and O. Gul, "A smart building power management concept: Smart socket applications with dc distribution," *International Journal of Electrical Power & Energy Systems*, vol. 64, pp. 679–688, 2015.

[17] N. Kherraf, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Latency and reliability-aware workload assignment in iot networks with mobile edge clouds," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1435–1449, 2019.

[18] P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S. A. Ashraf, B. Almeroth, J. Voigt, I. Riedel *et al.*, "Latency critical iot applications in 5g: Perspective on the design of radio interface and network architecture," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 70–78, 2017.

[19] L. F. Bittencourt, O. Rana, and I. Petri, "Cloud computing at the edges," in *International Conference on Cloud Computing and Services Science.* Springer, 2015, pp. 3–12.

[20] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[21] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[22] M. Aazam and E.-N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops).* IEEE, 2015, pp. 105–110.

[23] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *MCC workshop on Mobile cloud computing.* ACM, 2012, pp. 13–16.

[24] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[25] R. van der Meulen, “What edge computing means for infrastructure and operations leaders,” https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders/, 2018, [Online; accessed 15-June-2021].

[26] K. Mlitz, “Market size for edge computing worldwide 2025,” https://www.statista.com/statistics/948762/worldwide-edge-computing-market-size-forecast/#statisticContainer, 2021, [Online; accessed 15-June-2021].

[27] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, “Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study,” *IEEE access*, vol. 5, pp. 9882–9910, 2017.

[28] Y. Liu, Q. He, D. Zheng, X. Xia, F. Chen, and B. Zhang, “Data caching optimization in the edge computing environment,” *IEEE Transactions on Services Computing*, 2020.

[29] J. Byabazaire, G. O’Hare, and D. Delaney, “Data quality and trust: A perception from shared data in iot,” in *2020 IEEE International Conference on Communications Workshops (ICC Workshops).* IEEE, 2020, pp. 1–6.

[30] D. O’Keeffe, T. Salonidis, and P. Pietzuch, “Frontier: Resilient edge processing for the internet of things,” *Proceedings of the VLDB Endowment*, vol. 11, no. 10, pp. 1178–1191, 2018.

[31] C. C. Aggarwal, N. Ashish, and A. Sheth, *The Internet of Things: A Survey from the Data-Centric Perspective.* Boston, MA: Springer US, 2013, pp. 383–428.

[32] M. Taneja and A. Davy, “Poster abstract: Resource aware placement of data stream analytics operators on fog infrastructure for internet of things applications,” in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, 2016, pp. 113–114.

[33] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, “A storage solution for massive iot data based on nosql,” in *2012 IEEE International Conference on Green Computing and Communications.* IEEE, 2012, pp. 50–57.

[34] I. Psaras, O. Ascigil, S. Rene, G. Pavlou, A. Afanasyev, and L. Zhang, “Mobile data repositories at the edge,” in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[35] N. A. Ali and M. Abu-Elkheir, “Data management for the internet of things: Green directions,” in *2012 IEEE Globecom Workshops.* IEEE, 2012, pp. 386–390.

[36] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, E. Marín-Tordera, J. Cirera, G. Grau, and F. Casaus, “Estimating smart city sensors data generation,” in *2016 Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net).* IEEE, 2016, pp. 1–8.

[37] S. Davy, J. Famaey, J. Serrat, J. L. Gorricho, A. Miron, M. Dramitinos, P. M. Neves, S. Latré, and E. Goshen, “Challenges to support edge-as-a-service,” *IEEE Communications Magazine*, vol. 52, no. 1, pp. 132–139, 2014.

[38] Z. Shang, “Low latency v2x application of mec architecture in traffic safety,” in *The 2020 International Conference on Machine Learning and Big Data Analytics for IoT Security and Privacy*, J. MacIntyre, J. Zhao, and X. Ma, Eds. Springer International Publishing, 2021, pp. 735–739.

[39] K. Yu, L. Lin, M. Alazab, L. Tan, and B. Gu, “Deep learning-based traffic safety solution for a mixture of autonomous and manual vehicles in a 5g-enabled intelligent transportation system,” *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–11, 2020.

[40] M. Danish, J. Brazauskas, R. Bricheno, I. Lewis, and R. Mortier, “Deepdish: Multi-object tracking with an off-the-shelf raspberry pi,” in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys ’20. ACM, 2020, p. 37–42.

[41] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, “User allocation-aware edge cloud placement in mobile edge computing,” *Software: Practice and Experience*, 2019.

[42] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, “Videoedge: Processing camera streams using hierarchical clusters,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC).* IEEE, 2018, pp. 115–131.

[43] J. Ru, Y. Yang, J. Grundy, J. Keung, and L. Hao, “An efficient deadline constrained and data locality aware dynamic scheduling framework for multitenancy clouds,” *Concurrency and Computation: Practice and Experience*, p. e6037, 2020.

[44] W. Toussaint and A. Y. Ding, “Machine learning systems in the iot: Trustworthiness trade-offs for edge intelligence,” in *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI).* IEEE, 2020, pp. 177–184.

[45] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, “6g white paper on edge intelligence,” *arXiv preprint arXiv:2004.14850*, 2020.

[46] B. Varghese, E. De Lara, A. Y. Ding, C.-H. Hong, F. Bonomi, S. Dustdar, P. Harvey, P. Hewkin, W. Shi, M. Thiele *et al.*, "Revisiting the arguments for edge computing research," *IEEE Internet Computing*, 2021.

[47] B. Ramprasad, A. da Silva Veith, M. Gabel, and E. de Lara, "Sustainable computing on the edge: A system dynamics perspective," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, 2021, pp. 64–70.

[48] M. Liyanage, P. Porambage, A. Y. Ding, and A. Kalla, "Driving forces for multi-access edge computing (mec) iot integration in 5g," *ICT Express*, 2021.

[49] I. Lujic, V. De Maio, and I. Brandic, "Adaptive recovery of incomplete datasets for edge analytics," in *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC).* IEEE, 2018, pp. 1–10.

[50] I. Lujic, V. D. Maio, and I. Brandic, "Resilient edge data management framework," *IEEE Transactions on Services Computing*, vol. 13, no. 04, pp. 663–674, oct 2020.

[51] I. Lujic, V. De Maio, and I. Brandic, "Efficient edge storage management based on near real-time forecasts," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC).* IEEE, 2017, pp. 21–30.

[52] I. Lujic and H.-L. Truong, "Architecturing elastic edge storage services for data-driven decision making," in *European Conference on Software Architecture.* Springer, 2019, pp. 97–105.

[53] I. Lujic, V. D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, and I. Brandic, "Increasing traffic safety with real-time edge analytics and 5g," in *Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 19–24.

[54] I. Lujic, V. De Maio, S. Venugopal, and I. Brandic, "Sea-leap: Self-adaptive and locality-aware edge analytics placement," *IEEE Transactions on Services Computing*, no. 01, pp. 1–1, 2021.

[55] B. Dorsemaine, J.-P. Gaulier, J.-P. Wary, N. Kheir, and P. Urien, "Internet of things: a definition & taxonomy," in *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies.* IEEE, 2015, pp. 72–77.

[56] O. Vermesan, P. Friess *et al.*, *Internet of things-from research and innovation to market deployment.* River publishers Aalborg, 2014, vol. 29.

[57] K. K. Patel, S. M. Patel *et al.*, "Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges," *International journal of engineering science and computing*, vol. 6, no. 5, 2016.

[58] J. Cecílio, P. Martins, and P. Furtado, "Planning for heterogeneous iot with time guaranties," *Procedia Computer Science*, vol. 109, pp. 249–256, 2017.

[59] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, pp. 1–28, 2018.

[60] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.

[61] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[62] F. Giust, X. Costa-Perez, and A. Reznik, "Multi-access edge computing: An overview of etsi mec isg," *IEEE 5G Tech Focus*, vol. 1, no. 4, p. 4, 2017.

[63] IEA, "Data centres and data transmission networks," IEA, 2020, (accessed: 2021-09-21). [Online]. Available: https://www.iea.org/reports/data-centres-and-data-transmission-networks

[64] M. Iorga, L. Feldman, R. Barton, M. Martin, N. Goren, and C. Mahmoudi, "Fog computing conceptual model," 2018-03-14 2018.

[65] O. C. A. W. Group *et al.*, "Openfog reference architecture for fog computing," *OPFRA001*, vol. 20817, p. 162, 2017.

[66] N. C. Taher, I. Mallat, N. Agoulmine, and N. El-Mawass, "An iot-cloud based solution for real-time and batch processing of big data: Application in healthcare," in *2019 3rd International Conference on Bio-engineering for Smart Technologies (BioSMART)*. IEEE, 2019, pp. 1–8.

[67] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, "Iot big data analytics for smart homes with fog and cloud computing," *Future Generation Computer Systems*, vol. 91, pp. 563–573, 2019.

[68] I. Yaqoob, L. U. Khan, S. A. Kazmi, M. Imran, N. Guizani, and C. S. Hong, "Autonomous driving cars in smart cities: Recent advances, requirements, and challenges," *IEEE Network*, vol. 34, no. 1, pp. 174–181, 2019.

[69] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[70] V. Gularnik and J. Srivastava, "Event detection from time series data," in *In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1999, pp. 33–42.

[71] T. Mastelic and I. Brandic, “Data velocity scaling via dynamic monitoring frequency on ultrascale infrastructures,” in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on.* IEEE, 2015, pp. 422–425.

[72] D. Rivas, F. Guim, J. Polo, and D. Carrera, “Performance characterization of video analytics workloads in heterogeneous edge infrastructures,” *Concurrency and Computation: Practice and Experience*, p. e6317, 2021.

[73] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, “Real-time video analytics: The killer app for edge computing,” *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[74] G. Zhang, “Time series forecasting using a hybrid {ARIMA} and neural network model,” *Neurocomputing*, vol. 50, pp. 159 – 175, 2003.

[75] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control.* Holden-Day, Incorporated, 1990.

[76] M. Zhang, “Time series: Autoregressive models ar, ma, arma, arima,” *University of Pittsburgh*, 2018.

[77] R. J. Hyndman, E. Wang, and N. Laptev, “Large-scale unusual time series detection,” in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, Nov 2015, pp. 1616–1619.

[78] R. Hyndman and Y. Khandakar, “Automatic time series forecasting: The forecast package for r,” *Journal of Statistical Software, Articles*, vol. 27, no. 3, pp. 1–22, 2008.

[79] P. McSharry, “Stream analytics for forecasting,” *Foresight: The International Journal of Applied Forecasting*, no. 24, pp. 7–12, 2012.

[80] C. A. Jofipasi *et al.*, “Selection for the best ets (error, trend, seasonal) model to forecast weather in the aceh besar district,” in *IOP conference series: materials science and engineering*, vol. 352, no. 1. IOP Publishing, 2018, p. 012055.

[81] R. L. Galvez, A. A. Bandala, E. P. Dadios, R. R. P. Vicerra, and J. M. Z. Maningo, “Object detection using convolutional neural networks,” in *TENCON 2018-2018 IEEE Region 10 Conference.* IEEE, 2018, pp. 2023–2027.

[82] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in *European conference on computer vision.* Springer, 2014, pp. 740–755.

[83] S. Li, “Tensorflow lite: On-device machine learning framework,” *Journal of Computer Research and Development*, vol. 57, no. 9, p. 1839, 2020.

[84] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[85] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[86] I. A. Khan, M. Safdar, F. Ullah, S. R. Jan, F. Khan, and S. Shah, “Request-response interaction model in constrained networks,” *International Journal of Advance Research and Innovative Ideas in Education, Online ISSN-2395*, vol. 4396, 2016.

[87] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, “A survey on application layer protocols for the internet of things,” *Transaction on IoT and Cloud computing*, vol. 3, no. 1, pp. 11–17, 2015.

[88] O. Blazy, E. Conchon, M. Klingler, and D. Sauveron, “An iot attribute-based security framework for topic-based publish/subscribe systems,” *IEEE Access*, vol. 9, pp. 19 066–19 077, 2021.

[89] I. M. Wirawan, I. D. Wahyono, G. Idfi, and G. R. Kusumo, “Iot communication system using publish-subscribe,” in *2018 International Seminar on Application for Technology of Information and Communication.* IEEE, 2018, pp. 61–65.

[90] B. Mishra and A. Kertesz, “The use of mqtt in m2m and iot systems: A survey,” *IEEE Access*, vol. 8, pp. 201 071–201 086, 2020.

[91] A. Trapletti and K. Hornik, *tseries: Time Series Analysis and Computational Finance*, 2020, r package version 0.10-48. [Online]. Available: https://CRAN.R-project.org/package=tseries

[92] A. Zeileis and G. Grothendieck, “zoo: S3 infrastructure for regular and irregular time series,” *Journal of Statistical Software*, vol. 14, no. 6, pp. 1–27, 2005.

[93] H. Wickham, *ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York, 2016. [Online]. Available: https://ggplot2.tidyverse.org

[94] S. Ali, M. A. Jarwar, and I. Chong, “Design methodology of microservices to support predictive analytics for iot applications,” *Sensors*, vol. 18, no. 12, p. 4226, 2018.

[95] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” *CISCO white paper*, 2011.

[96] F. Tao, Q. Qi, A. Liu, and A. Kusiak, “Data-driven smart manufacturing,” *Journal of Manufacturing Systems*, vol. 48, pp. 157–169, 2018.

[97] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Sensing as a service model for smart cities supported by internet of things,” *Transactions on Emerging Telecommunications Technologies*, vol. 25, no. 1, pp. 81–93, 2014.

[98] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, “An information framework for creating a smart city through internet of things,” *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 112–121, 2014.

[99] A. Mikulec and V. Mikuličić, “Influence of renewable energy sources on distribution network availability,” *International journal of electrical and computer engineering systems*, vol. 2, no. 1, pp. 37–48, 2011.

[100] M. Al-Khafajiy, T. Baker, A. Waraich, D. Al-Jumeily, and A. Hussain, “Iot-fog optimal workload via fog offloading,” in *International Conference on Utility and Cloud Computing Companion.* IEEE, 2018, pp. 359–364.

[101] N. Abbas, M. Asim, N. Tariq, T. Baker, and S. Abbas, “A mechanism for securing iot-enabled applications at the fog layer,” *Journal of Sensor and Actuator Networks*, vol. 8, no. 1, p. 16, 2019.

[102] S. Liu and P. C. Molenaar, “ivar: A program for imputing missing data in multivariate time series using vector autoregressive models,” *Behavior research methods*, vol. 46, no. 4, pp. 1138–1148, 2014.

[103] K. Wellenzohn, M. H. Böhlen, A. Dignös, J. Gamper, and H. Mitterer, “Continuous imputation of missing values in streams of pattern-determining time series.” in *EDBT*, 2017, pp. 330–341.

[104] M. Yuriyama, T. Kushida, and M. Itakura, “A new model of accelerating service innovation with sensor-cloud infrastructure,” in *Annual SRII Global Conference.* IEEE, 2011, pp. 308–314.

[105] A. Papageorgiou, B. Cheng, and E. Kovacs, “Real-time data reduction at the network edge of internet-of-things systems,” in *2015 11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 284–291.

[106] A. Ukil, S. Bandyopadhyay, and A. Pal, “Iot data compression: Sensor-agnostic approach,” in *2015 Data Compression Conference*, April 2015, pp. 303–312.

[107] F. Van den Abeele, J. Hoebeke, I. Moerman, and P. Demeester, “Integration of heterogeneous devices and communication models via the cloud in the constrained internet of things,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 10, pp. 1–16, 2015.

[108] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, “Machine learning for internet of things data analysis: a survey,” *Digital Communications and Networks*, vol. 4, no. 3, pp. 161 – 175, 2018.

[109] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics.* Springer International Publishing, 2014, pp. 169–186.

[110] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec 2017.

[111] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. ACM, 2013, pp. 187–198.

[112] L. Tang, L. Yu, F. Liu, and W. Xu, "An integrated data characteristic testing scheme for complex time series data exploration," *International Journal of Information Technology & Decision Making*, vol. 12, no. 03, pp. 491–521, 2013.

[113] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, "Forecasting time series with complex seasonal patterns using exponential smoothing," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1513–1527, 2011.

[114] "Ericsson mobility report, june 2016," http://www.ericsson.com/res/docs/2016/ericsson-mobility-report-2016.pdf, 2016, [Online; accessed 30-August-2021].

[115] J. Lin, S. Williamson, K. Borne, and D. De Barr, *Pattern Recognition in Time Series, Advances in Machine Learning and Data Mining for Astronomy.* Chapman and Hall, 2012.

[116] X. Wang, K. Smith-Miles, and R. Hyndman, "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series," *Neurocomputing*, vol. 72, no. 10-12, pp. 2581–2594, 2009.

[117] T. W. Liao, "Clustering of time series data—a survey," *Pattern recognition*, pp. 1857–1874, 2005.

[118] X. Wang, K. Smith, and R. Hyndman, "Characteristic-based clustering for time series data," *Data mining and knowledge Discovery*, vol. 13, no. 3, pp. 335–364, 2006.

[119] A. Aral and I. Brandic, "Consistency of the fittest: Towards dynamic staleness control for edge data analytics," in *European Conference on Parallel Processing.* Springer, 2018, pp. 40–52.

[120] D. Minoli, K. Sohraby, and B. Occhiogrosso, "Iot considerations, requirements, and architectures for smart buildings—energy optimization and next-generation building management systems," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 269–283, 2017.

[121] M. Jaradat, M. Jarrah, A. Bousselham, Y. Jararweh, and M. Al-Ayyoub, “The internet of energy: smart sensor networks and big data management for smart grid,” *Procedia Computer Science*, vol. 56, pp. 592–597, 2015.

[122] D. Georgakopoulos, P. P. Jayaraman, M. Fazia, M. Villari, and R. Ranjan, “Internet of things and edge cloud computing roadmap for manufacturing,” *IEEE Cloud Computing*, vol. 3, no. 4, pp. 66–73, 2016.

[123] W. He, G. Yan, and L. Da Xu, “Developing vehicular data cloud services in the iot environment,” *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1587–1595, 2014.

[124] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, “Edge analytics in the internet of things,” *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[125] A. Zaslavsky, C. Perera, and D. Georgakopoulos, “Sensing as a service and big data,” *arXiv preprint arXiv:1301.0159*, 2013.

[126] D. V. Dimitrov, “Medical internet of things and big data in healthcare,” *Healthcare informatics research*, vol. 22, no. 3, pp. 156–163, 2016.

[127] P. O’Donovan, K. Leahy, K. Bruton, and D. T. O’Sullivan, “An industrial big data pipeline for data-driven analytics maintenance applications in large-scale smart manufacturing facilities,” *Journal of Big Data*, vol. 2, no. 1, p. 25, 2015.

[128] R. Lederman and L. Wynter, “Real-time traffic estimation using data expansion,” *Transportation Research Part B: Methodological*, vol. 45, no. 7, pp. 1062–1079, 2011.

[129] L. L. Lai, H. Braun, Q. P. Zhang, Q. Wu, Y. N. Ma, W. C. Sun, and L. Yang, “Intelligent weather forecast,” in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, vol. 7, Aug 2004, pp. 4216–4221 vol.7.

[130] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, “Fogflow: Easy programming of iot services over cloud and edges for smart cities,” *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, 2018.

[131] H.-L. Truong, A. Murguzur, and E. Yang, “Challenges in enabling quality of analytics in the cloud,” *Journal of Data and Information Quality (JDIQ)*, vol. 9, no. 2, pp. 1–4, 2018.

[132] H.-L. Truong and M. Halper, “Characterizing incidents in cloud-based iot data analytics,” in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 2018, pp. 442–447.

[133] J. Lin, E. Keogh, L. Wei, and S. Lonardi, “Experiencing sax: a novel symbolic representation of time series,” *Data Mining and knowledge discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[134] R. Li, T. Song, B. Mei, H. Li, X. Cheng, and L. Sun, "Blockchain for large-scale internet of things data storage and protection," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[135] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," *Future generation computer systems*, vol. 88, pp. 173–190, 2018.

[136] P. Balamuralidhara, P. Misra, and A. Pal, "Software platforms for internet of things and m2m," *Journal of the Indian Institute of Science*, vol. 93, no. 3, pp. 487–498, 2013.

[137] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a fog and edge computing infrastructure," in *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXIII.* Springer, 2017, pp. 40–79.

[138] G. Blair, N. Bencomo, and R. B. France, "Models@ run. time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[139] M. Su, L. Zhang, Y. Wu, K. Chen, and K. Li, "Systematic data placement optimization in multi-cloud storage for complex requirements," *IEEE Transactions on Computers*, vol. 65, no. 6, pp. 1964–1977, 2016.

[140] A. Taivalsaari and T. Mikkonen, "A roadmap to the programmable world: software challenges in the iot era," *IEEE Software*, vol. 34, no. 1, pp. 72–80, 2017.

[141] D. Talia, "Clouds for scalable big data analytics," *Computer*, vol. 46, no. 5, pp. 98–101, 2013.

[142] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, "Oceanstore: An architecture for global-scale persistent storage," in *ACM SIGARCH Computer Architecture News*, vol. 28. ACM, 2000, pp. 190–201.

[143] K. Goniewicz, M. Goniewicz, W. Pawłowski, and P. Fiedor, "Road accident rates: strategies and programmes for improving road traffic safety," *European journal of trauma and emergency surgery*, vol. 42, no. 4, pp. 433–438, 2016.

[144] D. Adminaité-Fodor and G. Jost, *How safe is walking and cycling in Europe? PIN Flash Report 38.*, European Transport Safety Council (ETSC), 2020.

[145] X. Yan, Y. Zhang, and L. Ma, "The influence of in-vehicle speech warning timing on drivers' collision avoidance performance at signalized intersections," *Transportation research part C: emerging technologies*, vol. 51, pp. 231–242, 2015.

[146] M. Green, "" how long does it take to stop?" methodological analysis of driver perception-brake times," *Transportation human factors*, vol. 2, no. 3, pp. 195–216, 2000.

[147] W. K. Alhajyaseen and M. Iryo-Asano, “Studying critical pedestrian behavioral changes for the safety assessment at signalized crosswalks,” *Safety science*, vol. 91, pp. 351–360, 2017.

[148] J. Chen and X. Ran, “Deep learning with edge computing: A review.” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[149] SWARCO, “Signals for the future. led traffic lights from the global market leader.” SWARCO, 2018, (accessed: 2021-07-16). [Online]. Available: https://www.swarco.com/sites/default/files/public/downloads/2019-02/SWARCO_folder_signalgeber_EN_screen.pdf

[150] Q. Zhang, H. Sun, X. Wu, and H. Zhong, “Edge video analytics for public safety: A review,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1675–1696, 2019.

[151] P. Patel, M. I. Ali, and A. Sheth, “On using the intelligent edge for iot analytics,” *IEEE Intelligent Systems*, vol. 32, no. 5, pp. 64–69, 2017.

[152] Z. Ning, J. Huang, and X. Wang, “Vehicular fog computing: Enabling real-time traffic management for smart cities,” *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, 2019.

[153] E. Ahmed, I. Yaqoob, I. A. T. Hashem, I. Khan, A. I. A. Ahmed, M. Imran, and A. V. Vasilakos, “The role of big data analytics in internet of things,” *Computer Networks*, vol. 129, pp. 459–471, 2017.

[154] J. Ren, Y. Pan, A. Goscinski, and R. A. Beyah, “Edge computing for the internet of things,” *IEEE Network*, vol. 32, no. 1, pp. 6–7, 2018.

[155] A. Aral and I. Brandic, “Learning spatiotemporal failure dependencies for resilient edge computing services,” *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2020.

[156] C. Wang, C. Gill, and C. Lu, “Adaptive data replication in real-time reliable edge computing for internet of things,” in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*. IEEE, 2020, pp. 128–134.

[157] Q. Fan and N. Ansari, “Towards workload balancing in fog computing empowered iot,” *IEEE Transactions on Network Science and Engineering*, 2018.

[158] B. Luo, S. Tan, Z. Yu, and W. Shi, “Edgebox: Live edge video analytics for near real-time event detection,” in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, IEEE. IEEE, 2018, pp. 347–348.

[159] P. Liu, B. Qi, and S. Banerjee, “Edgeeye: An edge service framework for real-time intelligent video analytics,” in *Proceedings of the 1st international workshop on edge systems, analytics and networking*. ACM, 2018, pp. 1–6.

[160] A. da Silva Veith, M. D. de Assuncao, and L. Lefevre, “Latency-aware placement of data stream analytics on edge computing,” in *International Conference on Service-Oriented Computing.* Springer, 2018, pp. 215–229.

[161] H. Gupta, Z. Xu, and U. Ramachandran, “Datafog: Towards a holistic data management platform for the iot age at the network edge,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[162] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, “Firework: Data processing and sharing for hybrid cloud-edge analytics,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2004–2017, 2018.

[163] S. Venugopal, M. Gazzetti, Y. Gkoufas, and K. Katrinis, “Shadow puppets: Cloud-level accurate {AI} inference at the speed and economy of edge,” in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[164] B. Cheng, A. Papageorgiou, and M. Bauer, “Geelytics: Enabling on-demand edge analytics over scoped data sources,” in *2016 IEEE International Congress on Big Data (BigData Congress)*, 2016, pp. 101–108.

[165] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, “Towards network-aware resource provisioning in kubernetes for fog computing applications,” in *2019 IEEE Conference on Network Softwarization (NetSoft).* IEEE, 2019, pp. 351–359.

[166] P.-H. Tsai, H.-J. Hong, A.-C. Cheng, and C.-H. Hsu, “Distributed analytics in fog computing platforms using tensorflow and kubernetes,” in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS).* IEEE, 2017, pp. 145–150.

[167] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, “Context-aware data and task placement in edge computing environments,” in *2019 IEEE International Conference on Pervasive Computing and Communications (PerCom.* IEEE, 2019, pp. 1–10.

[168] C. Li, J. Tang, H. Tang, and Y. Luo, “Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment,” *Future Generation Computer Systems*, vol. 95, pp. 249–264, 2019.

[169] H. B. Pasandi and T. Nadeem, “Convince: Collaborative cross-camera video analytics at the edge,” in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops).* IEEE, 2020, pp. 1–5.

[170] T. Rausch, S. Nastic, and S. Dustdar, “Emma: Distributed qos-aware mqtt middleware for edge computing applications,” in *2018 IEEE International Conference on Cloud Engineering (IC2E).* IEEE, 2018, pp. 191–197.

[171] V. De Maio and I. Brandic, "Multi-objective mobile edge provisioning in small cell clouds," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 127–138.

[172] S. Barker, A. Mishra, D. Irwin, E. Cecchet, P. Shenoy, and J. Albrecht, "Smart*: An open data set and tools for enabling research in sustainable homes," *SustKDD, August*, vol. 111, p. 112, 2012.

[173] "Umass trace repository," http://traces.cs.umass.edu/, 2017, [Online; accessed 04-August-2021].

[174] T. Bednar, A. David, H. Schöberl, and G. Kratochwil, "University plus-energy office high-rise building innovations for buildings in practice," 12 2016.

[175] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, pp. 679–688, 2006.

[176] R. J. Hyndman, *forecast: Forecasting functions for time series and linear models*, 2017, r package version 8.2. [Online]. Available: http://pkg.robjhyndman.com/forecast

[177] J. Hasenburg and D. Bermbach, "Geobroker: A pub/sub broker considering geo-context information," *Software Impacts*, vol. 6, p. 100029, 2020.

[178] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[179] L. Wang, J. Shi, G. Song, and I.-F. Shen, "Object detection combining recognition and segmentation," in *Asian conference on computer vision.* Springer, 2007, pp. 189–199.

[180] J.-P. Jodoin, G.-A. Bilodeau, and N. Saunier, "Urban tracker: Multiple object tracking in urban mixed traffic," in *IEEE Winter Conference on Applications of Computer Vision.* IEEE, 2014, pp. 885–892.

[181] V. De Maio and I. Brandic, "First hop mobile offloading of dag computations," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).* IEEE, 2018, pp. 83–92.

[182] G. Lewis and P. Lago, "A catalog of architectural tactics for cyber-foraging," in *2015 11th International ACM SIGSOFT Conference on Quality of Software Architectures (QoSA).* IEEE, 2015, pp. 53–62.

[183] A. Mehta, W. Tärneberg, C. Klein, J. Tordsson, M. Kihl, and E. Elmroth, "How beneficial are intermediate layer data centers in mobile edge networks?" in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W).* IEEE, 2016, pp. 222–229.

[184] R. J. Hyndman, Y. Khandakar *et al.*, *Automatic time series for forecasting: the forecast package for R.* Monash University, Department of Econometrics and Business Statistics, 2007, no. 6/07.

[185] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1542–1552, Aug. 2008.

[186] K. Wellenzohn, M. H. Böhlen, A. Dignös, J. Gamper, and H. Mitterer, "Continuous imputation of missing values in streams of pattern-determining time series," in *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, 2017, pp. 330–341.

[187] S. Liu and P. C. Molenaar, "ivar: A program for imputing missing data in multivariate time series using vector autoregressive models," *Behavior research methods*, vol. 46, no. 4, pp. 1138–1148, 2014.

[188] J. Honaker and G. King, "What to do about missing values in time-series cross-section data," *American Journal of Political Science*, vol. 54, no. 2, pp. 561–581, 2010.

[189] J. Duan, W. Wang, J. Zeng, D. Zhang, and B. Shi, "A prediction algorithm for time series based on adaptive model selection," *Expert Systems with Applications*, vol. 36, no. 2, pp. 1308–1314, 2009.

[190] W. Junger and A. P. de Leon, "Imputation of missing data in time series for air pollutants," *Atmospheric Environment*, vol. 102, pp. 96–104, 2015.

[191] F. Cismondi, A. S. Fialho, S. M. Vieira, J. M. C. Sousa, S. R. Reti, M. D. Howell, and S. N. Finkelstein, "Computational intelligence methods for processing misaligned, unevenly sampled time series containing missing data," in *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, April 2011, pp. 224–231.

[192] J. P. Musial, M. M. Verstraete, and N. Gobron, "Comparing the effectiveness of recent algorithms to fill and smooth incomplete and noisy time series," *Atmospheric chemistry and physics*, vol. 11, no. 15, pp. 7905–7923, 2011.

[193] K. Hocke and N. Kämpfer, "Gap filling and noise reduction of unevenly sampled data by means of the lomb-scargle periodogram," *Atmospheric Chemistry and Physics*, vol. 9, no. 12, pp. 4197–4206, 2009.

[194] M. Abu-Elkheir, M. Hayajneh, and N. A. Ali, "Data management for the internet of things: Design primitives and solution," *Sensors*, vol. 13, pp. 15 582–15 612, 2013.

[195] Q. Yan and Y. Y. Wang, "A kind of efficient data archiving method for historical sensor data," in *ICISCE '16*, July 2016, pp. 44–48.

[196] G. A. Lewis and P. Lago, "A catalog of architectural tactics for cyber-foraging," in *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*, ser. QoSA '15. ACM, 2015, pp. 53–62.

[197] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanarayanan, "Data staging on untrusted surrogates." in *FAST*, vol. 3, 2003, pp. 15–28.

[198] J. a. Gama, R. Sebastião, and P. P. Rodrigues, "Issues in evaluation of stream learning algorithms," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. ACM, 2009, pp. 329–338.

[199] J. Read, A. Bifet, G. Holmes, and B. Pfahringer, "Streaming multi-label classification." in *WAPA*, 2011, pp. 19–25.

[200] R. Montella, M. Ruggieri, and S. Kosta, "A fast, secure, reliable, and resilient data transfer framework for pervasive iot applications," in *Conference on Computer Communications Workshops.* IEEE, 2018, pp. 710–715.

[201] T. Baker, M. Asim, Á. MacDermott, F. Iqbal, F. Kamoun, B. Shah, O. Alfandi, and M. Hammoudeh, "A secure fog-based platform for scada-based iot critical infrastructure," *Software: Practice and Experience*, 2019.

[202] Z. Maamar, T. Baker, M. Sellami, M. Asim, E. Ugljanin, and N. Faci, "Cloud vs edge: Who serves the internet-of-things better?" *Internet Technology Letters*, vol. 1, no. 5, p. e66, 2018.

[203] H. d. S. Araújo, J. J. Rodrigues, R. d. A. Rabelo, N. d. C. Sousa, J. V. Sobral *et al.*, "A proposal for iot dynamic routes selection based on contextual information," *Sensors*, vol. 18, no. 2, p. 353, 2018.

[204] M. Hassanalieragh, A. Page, T. Soyata, G. Sharma, M. Aktas, G. Mateos, B. Kantarci, and S. Andreescu, "Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges," in *2015 IEEE International Conference on Services Computing*, 2015, pp. 285–292.

[205] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of grid computing*, vol. 12, no. 4, pp. 559–592, 2014.

[206] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," in *Quality of Software Architectures (QoSA), 2016 12th International ACM SIGSOFT Conference on.* IEEE, 2016, pp. 70–79.

[207] B. Vogel and D. Gkouskos, "An open architecture approach: Towards common design principles for an iot architecture," in *Proceedings of the 11th European*

Conference on Software Architecture: Companion Proceedings, ser. ECSA '17. ACM, 2017, pp. 85–88.

[208] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying architecture-based adaptation to automate the management of internet-of-things," in *Software Architecture - 12th European Conference on Software Architecture, ECSA 2018, Madrid, Spain, Proceedings*, 2018, pp. 49–67.

[209] M. D'Angelo, "Decentralized self-adaptive computing at the edge," in *Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '18. ACM, 2018, pp. 144–148.

[210] Z. Wen, P. Bhatotia, R. Chen, M. Lee *et al.*, "Approxiot: Approximate analytics for edge computing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 2018, pp. 411–421.

[211] M. Anandhalli and V. P. Baligar, "A novel approach in real-time vehicle detection and tracking using raspberry pi," *Alexandria engineering journal*, vol. 57, no. 3, pp. 1597–1607, 2018.

[212] B. Guidi and L. Ricci, "Aggregation techniques for the internet of things: An overview," in *The Internet of Things for Smart Urban Ecosystems.* Springer, 2019, pp. 151–176.

[213] V. Moysiadis, P. Sarigiannidis, and I. Moscholios, "Towards distributed data management in fog computing," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[214] B. Diène, J. J. Rodrigues, O. Diallo, E. H. M. Ndoye, and V. V. Korotaev, "Data management techniques for internet of things," *Mechanical Systems and Signal Processing*, vol. 138, p. 106564, 2020.

[215] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2573–2574.

[216] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS).* IEEE, 2017, pp. 276–286.

[217] B. Ali, M. A. Pasha, S. ul Islam, H. Song, and R. Buyya, "A volunteer-supported fog computing environment for delay-sensitive iot applications," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3822–3830, 2020.

[218] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).* IEEE, 2017, pp. 401–410.

[219] T. Hamrouni, S. Slimani, and F. B. Charrada, “A survey of dynamic replication and replica selection strategies based on data mining techniques in data grids,” *Engineering Applications of Artificial Intelligence*, vol. 48, pp. 140–158, 2016.

[220] P. Kathiravelu, P. Van Roy, and L. Veiga, “Composing network service chains at the edge: A resilient and adaptive software-defined approach,” *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 11, p. e3489, 2018.

[221] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, “A double deep q-learning model for energy-efficient edge scheduling,” *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2018.

[222] Q. Luo, C. Li, T. Luan, and W. Shi, “Minimizing the delay and cost of computation offloading for vehicular edge computing,” *IEEE Transactions on Services Computing*, 2021.

[223] D. Wang, Y. Peng, X. Ma, W. Ding, H. Jiang, F. Chen, and J. Liu, “Adaptive wireless video streaming based on edge computing: Opportunities and approaches,” *IEEE Transactions on services Computing*, vol. 12, no. 5, pp. 685–697, 2018.

[224] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, “Migration modeling and learning algorithms for containers in fog computing,” *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[225] M. Malensek, S. L. Pallickara, and S. Pallickara, “Hermes: Federating fog and cloud domains to support query evaluations in continuous sensing environments,” *IEEE Cloud Computing*, vol. 4, no. 2, pp. 54–62, 2017.

[226] P. B. Weerakody, K. W. Wong, G. Wang, and W. Ela, “A review of irregular time series data handling with gated recurrent neural networks,” *Neurocomputing*, 2021.

APPENDIX A

Curriculum Vitæ

Ivan Lujić

Personal Information

Date of birth	November 26th, 1992
Place of birth	Livno, Bosnia and Herzegovina
Citizenship	Croatian
E-mail	ivan.lujic@tuwien.ac.at || ivan.lujic@gmail.com
Web	https://www.ec.tuwien.ac.at/ivan.lujic
Affiliation	Vienna University of Technology Institute of Information Systems Engineering Research Unit E-Commerce Favoritenstrasse 9-11/194, A-1040 Vienna, Austria

Education

2017 - present	**Doctoral Degree in Computer Science** at Vienna University of Technology, Vienna, Austria
2014 - 2016	**Master's Degree in Computer Science** at University of Split, Split, Croatia
2011 - 2014	**Bachelor's Degree in Computer Science** at University of Split, Split, Croatia
2007 - 2011	**Economic Technician** at Vocational school, Livno, Bosnia and Herzegovina

Professional Experience

2016 – present	**Project Assistant** at TU Wien
2020 – 2021	**Teaching Assistant** at TU Wien (Course: Energy-efficient Distributed Systems)
07/2019 – 09/2019	**Research Intern** at IBM Research Ireland, Dublin, Ireland
09/2015 – 10/2015	**Student Intern** at Ericsson Nikola Tesla, Split, Croatia

Honors, Awards and Scholarships

- *netidee* scholarship financed by the Internet Foundation Austria (IPA), for supporting PhD theses that promote research on Internet-related topics (10k Euro, 12/2018 - 05/2020)
- Short Term Scientific Mission (STSM) grant within the COST project IC1305: NESUS (Network for Sustainable Ultrascale Computing) (2018)
- Erasmus+ grant for student exchange at the Vienna University of Technology, Vienna, Austria (03/2016 - 07/2016)
- Two times student scholarship by the State Office for Croats Abroad, a government office of the Republic of Croatia (2014, 2015)

Certifications

- IBM Cloud: Deploying Microservices with Kubernetes (Coursera, 2019)
  Credential: https://www.coursera.org/account/accomplishments/verify/N68MT92A942C
- In-Memory Data Management (openHPI - Hasso Plattner Institute, 2015)
  Credential: https://open.hpi.de/verify/xohah-sipof-mylog-mehob-gesit
- SAP S/4HANA – Deep Dive (openSAP, 2015)
  Credential: https://open.sap.com/verify/xevoz-tehav-cahef-habuh-duzem
- Leadership in Digital Transformation (openSAP, 2015)
  Credential: https://open.sap.com/verify/xusev-lobir-lilis-nyhed-tubuz
- Sustainability and Business Innovation (openSAP, 2015)
  Credential: https://open.sap.com/verify/xucan-zilah-malos-mezov-zapet

Scientific Activities

Research Projects

- InTraSafEd 5G - Increasing Traffic Safety with Edge and 5G, financed by the City of Vienna, 2020 (2020) http://intrasafed.ec.tuwien.ac.at/
- RUCON - Runtime Control in Multi Clouds, FWF Y 904 START Programm, 2015 (2016 - present) http://rucon.ec.tuwien.ac.at/

Scientific Talks

- "Increasing Traffic Safety with Real-Time Edge Analytics and 5G", ACM 4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2021), April 26, 2021, [Online]
- "Architecturing Elastic Edge Storage Services for Data-Driven Decision Making", 13th European Conference on Software Architecture (ECSA 2019), September 9-13, 2019, Paris, France
- "Efficient Data Management for Near Real-Time Edge Analytics", PhD Symposium on Data Science and Heterogeneous Computing, Third NESUS Winter School (COST IC1305), January 22-25, 2018, Zagreb, Croatia
- "Adaptive Recovery of Incomplete Datasets for Edge Analytics". IEEE 2nd International Conference on Fog and Edge Computing (ICFEC 2018), May 3, 2018, Washington DC, USA
- "Efficient Edge Storage Management Based on Near Real-Time Forecasts". IEEE 1st International Conference on Fog and Edge Computing (ICFEC 2017), May 14, 2017, Madrid, Spain

Other Activities

- **Organizing Committee:** DCC 2020 (Web chair)
- **Reviewer for Conferences and Workshops:** ACM/SPEC ICPE 2022, IEEE CLUSTER 2021, IC2E 2021, EURO-PAR 2017-2018 & 2021, CCGRID 2019-2020, ICS 2020, UCC 2019-2020, IEEE CIM 2020, SBAC-PAD 2019, ICPADS 2017, WORKS 2017.

- **Co-advised Students:** Daniel Hofstätter (Bacchelor Thesis: "Evidential Deep Learning in Object Detection", October 2021), Daniel Suchan (Bacchelor Thesis: "Performance Benchmarking of Near Real-Time Inference on Heterogeneous Edge Devices", August 2021), Dominik Jandl (Bacchelor Thesis: "Near-Real-Time Deciscion-Making for Sensor-Based Edge Systems", July 2021), Ivan Bodrožić (Master Thesis: "Near Real-Time Object Detection in Resource-constrained Edge Environments", July 2020), Josip Lasić (Master Thesis: "Communication Models for Edge/5G", July 2020)

- **Memberships:** Student Member, IEEE, 2017-2021

Publications

Refereed Publications in Journals

1. **Ivan Lujic**, Vincenzo De Maio, Srikumar Venugopal and Ivona Brandic. *SEA-LEAP: Self-adaptive and Locality-aware Edge Analytics Placement.* IEEE Transactions on Services Computing, 2021. DOI: 10.1109/TSC.2021.3104458 (Early Access)

2. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Resilient Edge Data Management Framework.* IEEE Transactions on Services Computing, vol. 13, no. 4, pp. 663-674, 1 July-Aug, 2020. DOI: 10.1109/TSC.2019.2962016

Refereed Publications in Conference Proceedings

1. **Ivan Lujic** and Truong Hong-Linh. *Architecturing Elastic Edge Storage Services for Data-Driven Decision Making.* In: Bures T., Duchien L., Inverardi P. (eds) Software Architecture, European Conference on Software Architecture (ECSA), Lecture Notes in Computer Science, vol 11681, pp. 97-105, Springer, Cham, 2019. DOI: 10.1007/978-3-030-29983-5_7.

2. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Adaptive Recovery of Incomplete Datasets for Edge Analytics.* IEEE 2nd International Conference on Fog and Edge Computing (ICFEC), 2018. DOI: 10.1109/CFEC.2018.8358726.

3. **Ivan Lujic**, Vincenzo De Maio and Ivona Brandic. *Efficient Edge Storage Management Based on Near Real-Time Forecasts.* IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 21-30, 2017. DOI: 10.1109/ICFEC.2017.9.

Refereed Publications in Workshop Proceedings

1. **Ivan Lujic**, Vincenzo De Maio, Klaus Pollhammer, Ivan Bodrozic, Josip Lasic and Ivona Brandic. *Increasing Traffic Safety with Real-Time Edge Analytics and 5G.* ACM Proceedings of the 4th International Workshop on Edge Systems, Analytics and Networking (EdgeSys), pp. 19–24, April, 2021. DOI: 10.1145/3434770.3459732

Theses

- **Ivan Lujić**. Foundations for Sustainable and Trustworthy Edge Data Analytics. PhD thesis, Faculty of Informatics, Vienna University of Technology, 2022, Vienna, Austria. (to appear)
- **Ivan Lujić**. Analysis of unevenly spaced time series data in highly distributed and unreliable networks. Master's thesis, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture (FESB), University of Split, July, 2016, Split, Croatia.
- **Ivan Lujić**. Usluge Amazon računalnog oblaka (Amazon Web Services). Bachelor's thesis (in Croatian), Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture (FESB), University of Split, July, 2014, Split, Croatia.

Curriculum vitæ last updated: 10th January, 2022