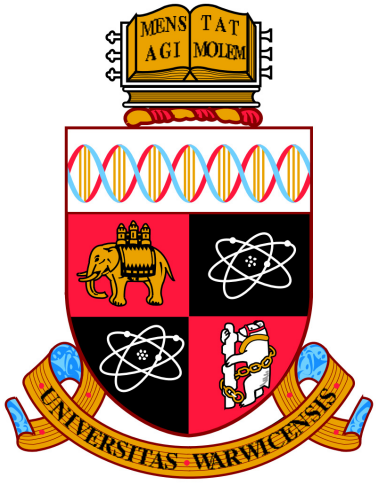


Machine Learning at the Edge for Air Quality Prediction

by

I Nyoman Kusuma Wardana

Thesis

Submitted to the University of Warwick

for the degree of

Doctor of Philosophy

School of Engineering

January 2024

Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
AQG	Air Quality Guidelines
AQI	Air Quality Index
ARIMA	Auto Regressive Integrated Moving Average
ARMA	Auto Regressive Moving Average
ASIC	Application-Specific Integrated Circuit
AURN	Automatic Urban and Rural Network
BAN	Binary Activation Network
Bi-LSTM	Bi-Directional Long Short-Term Memory
BLE	Bluetooth Low Energy
BNN	Binary Neural Network
BWN	Binary Weight Network
CLB	Configurable Logic Block
ClustME	Clustered Model Exchange
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAE	Denoising Autoencoder

DL	Deep Learning
FedAvg	Federated Averaging
FF	Flip-Flop
FL	Federated Learning
FPU	Floating-Point Unit
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
I2C	Inter-Integrated Circuit
I/O	Input/Output
IC	Integrated Circuit
ICT	Information and Communication Technology
IoT	Internet of Things
KNN	k-Nearest Neighbours
LAQN	London Air Quality Network
LoRa	Long-Range
LPDDR	Low-Power Double Data Rate
LR	Linear Regression
LSTM	Long Short-Term Memory
LUT	Look-Up-Table
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MAR	Missing at Random
MCAR	Missing Completely at Random
MICE	Multiple Imputation by Chained Equation

MNAR Missing Not at Random

ML Machine Learning

MLP Multi Layer Perceptron

MLR Multiple Linear Regression

MQTT Message Queuing Telemetry Transport

NB-IoT NarrowBand-Internet of Things

NERC Natural Environment Research Council

NN Neural Network

NPU Neural Processing Unit

OLS Ordinary Least Squares

OS Operating System

PM Particulate Matter

PM_{10} Particulate matter that are 10 μm or less in diameter

$PM_{2.5}$ Particulate matter that are 2.5 μm or less in diameter

PSU Power Supply Unit

QNN Quantised Neural Network

RAM Random-Access Memory

RBF Radial Basis Function

ReLU Rectified Linear Unit

RIR Rate of Improvement on RMSE

RMSE Root Mean Square Error

RNN Recurrent Neural Network

RTC Real-Time Clock

RPi Raspberry Pi

SBC	Single-Board Computer
SDG	Sustainable Development Goals
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System-on-a-Chip
SpaTemp	Spatiotemporal Data Exchange
SPI	Serial Peripheral Interface
SSH	Secure Shell
STE	Straight-Through Estimator
TF	TensorFlow
TFLite	TensorFlow Lite
TFLM	TensorFlow Lite for Microcontrollers
TinyML	Tiny Machine Learning
UART	Universal Asynchronous Receiver-Transmitter
UN	United Nations
VFP	Vector Floating Point
Wi-Fi	Wireless Fidelity
WHO	World Health Organization

Acknowledgments

I am deeply indebted to my supervisors, Prof. Julian Gardner and Dr. Suhaib Fahmy, for their unwavering support, patience, and motivation throughout my time at the University. Their invaluable feedback has significantly contributed to developing my research skills and personal growth. Their guidance and expertise have been instrumental in the completion of this thesis.

I would like to acknowledge the Indonesia Endowment Fund for Education (LPDP), Ministry of Finance, Republic of Indonesia, for providing me with financial support under grant number Ref: S-1027/LPDP.4/2019. Without LPDP's support, studying abroad would not have been possible.

My sincere gratitude to all the individuals I have encountered during my studies, particularly the WARC lab members (Ryan, Alex, Lenos, and Nidhin), PhD students (Augusta, Naser, Dyah, Joseph, and Satria), and colleagues in the Connected System group and at the Microsensors and Bioelectronic Laboratory. Their companionship and insightful discussions have greatly enriched my daily life.

I extend special thanks to Amanda Billingsley, CEO/Managing Director of Environmental Instruments Ltd, for generously donating an AQmesh picture and providing valuable insights about air pollution monitoring studies. I sincerely thank my colleagues at Politeknik Negeri Bali for granting me permission and supporting me in pursuing a doctoral degree at the University of Warwick.

My heartfelt appreciation goes to my family, especially my parents (Bapak Surim and Ibu Martina), my brothers (Suwardika and Dwipayana), my sisters-in-law (Wulan and Hiroe), and Hartawan's family (Bapak Hartawan, Ibu Swandewi, Fufe and DeAgus) for their unwavering support and understanding.

I cannot fail to express my deepest gratitude to my beloved wife, Juliani, and my daughter, Gauri, whose constant encouragement and unconditional support. They have sprinkled moments of joy and happiness throughout this PhD journey. I apologise for the occasions when my physical presence may have been there, yet I acknowledge that my true availability was lacking.

All praises and glories to Sri Guru and Sri Gauranga for giving me the blessing and strength to complete my study and thesis. I offer my obeisances to Srila Subhag Swami Guru Maharaj and all the Vaishnavas.

Declarations

This thesis is submitted to the University of Warwick to support my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. All work presented was carried out by the author, except where otherwise indicated. Parts of this thesis have been published in peer-reviewed journals and international conferences.

Publications

The author has published parts of this thesis as follows:

Journal Papers

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Optimising Deep Learning at the Edge for Accurate Hourly Air Quality Prediction," *Sensors*, vol. 21, no. 4, p. 1064, Feb. 2021 [1].

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Estimation of Missing Air Pollutant Data Using a Spatiotemporal Convolutional Autoencoder," *Neural Comput & Applic*, vol. 34, no. 18, pp. 16129–16154, Sep. 2022 [2].

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Collaborative Learning at the Edge for Air Pollution Prediction," *IEEE Transactions on Instrumentation & Measurement*, vol. 34, pp.1-12, Dec. 2023 [3].

- I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "TinyML Models for a Low-cost Air Quality Monitoring Device," *IEEE Sensors Letters*, vol. 7, no. 11, pp. 1-4, Sep. 2023 [4].

Conference

- I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "TinyML with Meta-Learning on Microcontrollers for Air Pollution Prediction," *Eurosensors 2023 Conference*, Lecce, Italy, 10 - 13 September 2023 [5].

Posters

- I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising Tiny Machine Learning with Binary Weight Network for a Low-cost Air Quality Monitoring Device," *The 3rd Imperial Workshop on Intelligent Communications*, Imperial College London, 19 - 20 June 2023 [6].

- I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising TinyML Using Binary Weight Network and Meta-Learning for a Low-cost Air Quality Monitoring Device," *Warwick Secure and Intelligent Communications (WSIC) Workshop*, University of Warwick, 31 July 2023 [7].

Sponsorships and Grants

I was supported with a PhD scholarship from the Indonesia Endowment Fund for Education (LPDP), Ministry of Finance, Republic of Indonesia, under grant number Ref: S-1027/LPDP.4/2019.

Abstract

Air pollution has emerged as a notable worldwide threat to public health, emphasising the importance of monitoring air quality status. A recent development involves establishing an extensive network comprising low-cost sensor nodes to facilitate air quality monitoring. Typically, these affordable devices are linked to cloud infrastructure. Nevertheless, processing data near its source presents a potential solution to the latency, privacy, and scalability challenges often encountered in cloud-based systems. The considerable amount of data generated also enables the potential implementation of machine learning for air quality research. Current research has yet to delve extensively into how best to leverage machine learning on edge devices for air quality monitoring applications. This thesis addresses this gap by presenting new approaches encompassing various aspects: overcoming missing data, optimising machine learning models, facilitating collaborative learning across devices, and deploying models on resource-constrained devices.

Based on the conducted experiments, the proposed autoencoder model outperforms commonly used univariate imputations to deal with missing data that can occur in such networks. This results in root mean square error improvement rates of approximately 50% to 65% against univariate and about 20% to 40% against multivariate imputation. The thesis also introduces a hybrid deep learning model that combines 1D Convolutional and Long Short-Term Memory networks for accurately predicting $PM_{2.5}$ pollutant levels by leveraging spatiotemporal data from neighbouring stations. The proposed model outperformed the other 19 models. Dynamic range quantisation was found to be a beneficial solution. Furthermore, this thesis discusses three collaborative learning methods that can be applied in such a distributed sensor setting: federated learning, learning with model sharing, and learning with spatiotemporal data exchanges, showing that the latter minimises loss during training across all participating air quality monitoring stations. Finally, developing a real low-cost air quality device is discussed, including implementing tiny machine learning on microcontrollers by investigating techniques such as binary weight networks and meta-learning.

These contributions address the key challenges of data integrity, neighbourhood correlation, collaborative methods for building models, and hardware considerations for real deployments. This holistic approach has identified further challenges for applying machine learning in this important area.

Keywords: air pollution prediction, machine learning, missing data imputation, edge computing, collaborative learning, tinyML.

Chapter 1

Introduction

1.1 Air Pollution as a Global Threat

The quality of the air we breathe significantly affects our health [8, 9]. Nevertheless, in 2019, 99% of the global population resided in areas where air quality did not meet the levels recommended by the World Health Organization (WHO) Air Quality Guidelines (AQG) [10]. The adverse impacts of pollutants released into the atmosphere impact local environments, ecosystems, human health, and climate on a global scale. Air pollution affects individuals of all ages, regardless of socioeconomic status, across diverse communities worldwide. Air pollution poses a significant global risk and requires immediate attention and comprehensive solutions. Addressing the complex challenge of air pollution requires collaborative efforts, innovative technologies, and proactive policies to reduce its far-reaching impacts.

The primary source of air pollution worldwide stems from the combustion of fossil fuels such as coal, diesel fuel, gasoline, oil, and natural gas [11]. This occurs predominantly in activities such as electricity production, heating, transportation, and industrial processes. Several factors have been linked to the heightened combustion of fossil fuels, including population growth, urbanisation, industrial expansion, and economic development [12]. The world population continues to grow, and the United Nations estimates that the world population will grow to around 8.5 billion by 2030 and 9.7 billion by 2050 [13]. The population growth has led to increased demand for energy, transportation, and resources, thereby increasing emissions of pollutants into the atmosphere. Urbanisation, city development, and industrialisation increase pollutant concentrations through vehicle exhaust gases and industrial processes. While economic development is a pivotal catalyst for national progress, it frequently entails an associated cost in elevated pollution levels. This occurs as

industries expand and the demand for energy experiences a surge. Among the pollutants are particulate matter (PM), nitrogen dioxide (NO_2), carbon monoxide (CO), ozone (O_3), and sulphur dioxide (SO_2).

The impact of diseases caused by air pollution is comparable to other global health risks, such as unhealthy eating patterns and smoking [14]. Air pollution is increasingly recognised as the greatest environmental hazard to human health and is becoming a major threat to public health on a global scale. The effects of air pollutants on the human body vary depending on the type, quantity, and duration of exposure to the contaminants. Regarding human health, air pollution is associated with lung cancer [15, 16], cardiovascular diseases [17, 18, 19], impaired cognitive function, and human emotion [20, 21]. Furthermore, in 2017, air pollution was responsible for approximately 4.9 million deaths [8]. In addition to negatively affecting human health, it also influences socioeconomic activity [22, 23], one of them is reducing physical activity during periods of high pollution or discouraging a person from engaging in physical activity at all, especially in highly polluted environments [24]. Other socioeconomic consequences of air pollution include premature mortality, adverse social and educational outcomes, and a catastrophic climate [25]. Research also indicates a connection between air pollution and poverty [26].

Some countries suffer more from air pollution. In Central and Southern Asia and Sub-Saharan Africa, populations continue to face escalating levels of air pollution [27]. India, Nepal, Bangladesh, and Pakistan rank among the most polluted countries globally [28]. Bangladesh was the most polluted country, while New Delhi, India, was recognised as the world's most polluted capital. The nations with the highest levels of air pollution, surpassing WHO guidelines by tenfold, included India, Pakistan, and Bangladesh [14, 29]. In China, prior economic growth heavily depended on fossil fuels, leading to significant air pollution problems [30]. Fortunately, there have been developments in air pollution control in China, emphasising a strategic shift from focusing only on emissions control to comprehensive air quality management [31].

1.2 Air Pollution Assessment

Monitoring and assessing pollution levels and air quality is important in maintaining public health, preserving the environment, and guiding policy decisions [32]. The importance of assessing air quality can be exemplified by the following:

- Public Health Protection: Harmful pollutants can adversely affect human health [33]. Thus, efforts to assess air quality are pivotal in measuring con-

centrations of harmful pollutants. This information is crucial for informing the public and decision-makers about the necessary actions to protect public health. Moreover, early detection of high pollution levels allows for timely public health interventions, reducing the risk of pollution-related diseases.

- Climate Change and Environmental Protection: Air pollution and climate change are closely linked because the substances that contribute to deteriorating air quality often share emissions with greenhouse gases [34, 35, 36]. Thus, monitoring and assessing air quality can improve efforts to address climate change and protect the environment.

- Policy Development: Accurate air quality data is crucial for formulating policies and regulations to control emissions and enhance air quality standards. Making policies in the field of air pollution has fundamental benefits, primarily aimed at preventing and controlling pollution originating from emission sources. The goal is to improve air quality and prevent negative impacts on health [37].

- Resource Allocation: A crucial aspect of regional air quality management involves allocating resources for air pollution mitigation to maximise environmental and human health benefits [38, 39]. Efficient resource allocation can be facilitated through monitoring, targeting areas with the most prominent air quality problems, and ensuring interventions are directed where they are most needed.

- Research, Technology and Innovation: Considerable research attention has been devoted to technological innovation and environmental pollution [40]. Air quality data helps researchers understand how pollution affects health, leading to technological innovations and policies addressing these issues.

1.3 Initiatives to Reduce Air Pollution Impact

Intervention is required to mitigate the negative impacts of air pollutants [41], and implementation of policy interventions has proven effective in improving air quality [42]. The United Nations introduced the Sustainable Development Goals (SDGs), also called the Global Goals, in 2015. These goals serve as a worldwide initiative to eliminate poverty, safeguard the planet, and ensure peace and prosperity for everyone by 2030 [43]. The SDGs aim to promote sustainable living for humanity, and naturally, addressing air pollution is intertwined with these goals in several ways.

Advancing specific SDGs can improve air quality as a positive result, and initiatives to reduce emissions will directly contribute to several SDGs [44].

The World Health Organization (WHO) published the Air Quality Guidelines (AQG), which are global standards for national, regional, and municipal governments. These guidelines aim to improve their citizens' health by minimising air pollution [45]. The WHO AQG updated in September 2021 recommends targeting annual mean concentrations of $PM_{2.5}$ not exceeding 5 $\mu g/m^3$, NO_2 not exceeding 10 $\mu g/m^3$, and a peak season mean 8-hour ozone concentration not surpassing 60 $\mu g/m^3$ [14]. Moreover, in December 2023, the United Nations Climate Change Conference (COP28) concluded with an agreement marking the start of the transition away from fossil fuels [46]. The agreement aims for a rapid, fair, and equitable shift, supported by substantial reductions in emissions and increased financial support. Following this two-week-long conference, there is a consensus that global greenhouse gas emissions need to be cut by 43% by 2030, compared to 2019 levels [46]. This development raises optimism for improved global air quality.

Many nations have created and implemented plans to decrease air pollution, specifically emphasising reducing transportation, industry, and energy production emissions. For example, in 2013, the Chinese State Council introduced the Air Pollution Prevention and Control Action Plan (APPCAP) to reduce particulate matter (PM) levels [47]. Since 2013, China's National Environmental Monitoring Centre has increased its air pollution monitoring network, now consisting of over 2000 stations nationwide, measuring pollutant concentrations [48]. China has taken effective measures to significantly lower air pollution in recent years. However, achieving nationwide air quality standards remains a long-term challenge for the country [49]. Another country, India, launched the National Clean Air Programme (NCAP) in 2019, aiming to enhance air quality in 122 cities [50]. This initiative comprises regulations, policies, and programs focused on identifying cost-effective measures to reduce emissions from various sources, ultimately improving air quality and public health. The NCAP aims to develop clean air action plans to reduce $PM_{2.5}$ pollution by 20–30% by 2024 compared to 2017 levels in designated cities [50, 51].

1.4 Machine Learning for Air Quality Research

A large network of low-cost sensor nodes has recently been proposed for monitoring air quality [52]. This new paradigm aims to gather spatial and temporal data on air pollution using multiple sensing devices, incorporating the established methodology with more accurate and expensive instrumentation [53]. Additionally, these sensors

are often connected to the Internet, allowing remote air quality monitoring. The Internet of Things (IoT) is a rapidly growing field that has become vital to our everyday lives. By leveraging cloud computing capabilities, IoT technology can link many heterogeneous devices to the cloud with different communication technologies, allowing data to be processed and represented in various ways. IoT technology has benefited numerous domains, including home automation, personal health care, environmental monitoring, and industry [54]. While the IoT offers many benefits, it provokes a critical situation for time-sensitive applications. IoT applications encompass devices linked to a network. The duration required to transmit data between end devices and the server, in both directions, may induce delays, presenting potential challenges for time-sensitive applications. Additionally, IoT applications frequently rely on a consistent internet connection, and a failure in connectivity could result in significant delays or even system faults.

A significant increase in the volume of data generated, stored, and transmitted has been attributed to the rapid growth of these sensing devices [55]. Due to the large amount of data being collected, Machine Learning (ML) techniques have greater potential for predicting air pollution [56]. With its ability to extract spatial and temporal features from data, Deep Learning (DL), a subset of machine learning, offers a promising approach to predicting air quality status. A deep learning model comprises multiple layers containing neurons and extracts meaningful patterns from large quantities of input data to support inference. By leveraging the availability of air quality data, DL techniques can be effectively utilised in various areas of air quality research. These include imputing missing air pollution data, developing accurate deep learning models, optimising embedded deep learning models, and facilitating collaborative learning among sensing devices.

1.5 Moving Machine Learning Towards the Edge

The future of machine learning is shifting towards edge computing, and a recent survey sheds light on how developers are driving innovation in tiny machine learning (tinyML) [57]. ML technology has become a viable option for endpoint devices within a few years. The key advantages of embedded machine learning, summarised as **BLERP** [58, 59], are as follows:

- **B**andwidth: By reducing the need to send data to the cloud, processing data locally at the edge minimises bandwidth requirements.
- **L**atency: Performing data processing at the edge enables quicker decision-

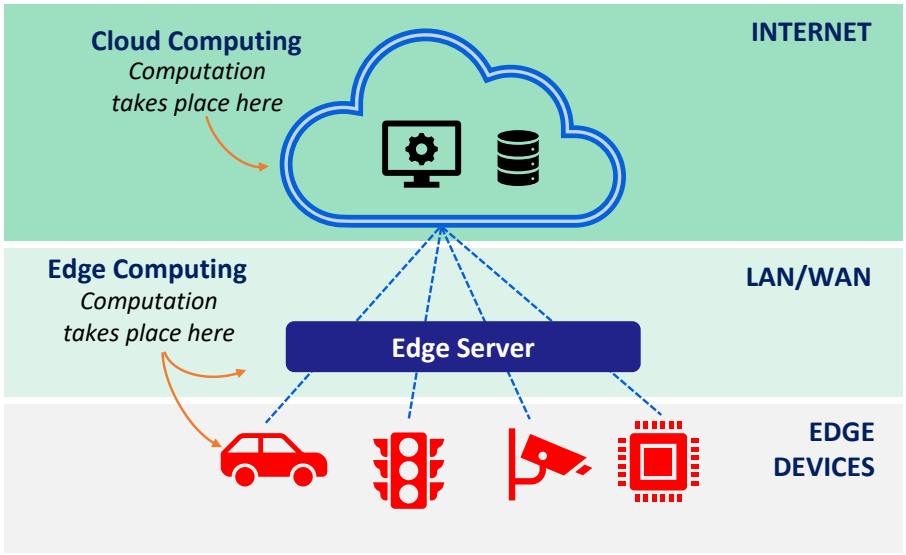


Figure 1.1: Processing data in the cloud versus at the edge.

making, as there is no delay in transmitting data to a remote server.

- **E**conomics: Local processing at the edge eliminates the need for cloud services, resulting in reduced operational costs.
- **R**eliability: Edge computing does not rely on an internet connection, making it more reliable for time-critical applications.
- **P**rivacy: Concerns over privacy increase when sending sensitive data to the cloud. Edge computing allows for local processing and consumption of data, reducing the risk of privacy breaches.

The integration of ML processing at the edge is commonly known as *edge computing*. Fig. 1.1 depicts that edge computing involves deploying computation closer to the data sources, as opposed to a more centralised approach seen in cloud computing. This approach effectively tackles latency, privacy, and scalability challenges often encountered in cloud-based systems.

1.6 Thesis Aims and Objectives

This thesis aims to develop methods related to some aspects of air quality domains using machine learning (or deep learning) techniques, emphasising edge devices as deployment targets. Leveraging edge devices such as single board computers (SBCs)

and microcontrollers requires machine learning models to be tailored to the available computational capabilities and memory capacity. With the increasing number of air quality monitoring devices and the presence of spatiotemporal correlations among these devices in a specific region, it becomes imperative to explore strategies for implementing collaborative learning among edge devices. Such approaches can potentially enhance the performance of models used in predicting air quality data. Based on the aforementioned aims, the thesis objectives can be summarised as follows:

1. To develop a method for imputing missing values on measurement data, considering spatiotemporal behaviour of air quality status.
2. To develop a deep learning model to predict air pollution levels accurately with model optimisations for edge devices.
3. To develop collaborative learning strategies among edge devices and evaluate the proposed strategies in terms of model accuracy, device performance, and communication cost.
4. To deploy tiny machine learning models on resource-constrained microcontrollers as target devices to address air quality issues.

1.7 Thesis Organisation

This thesis can be organised in the following order:

- **Chapter 1 Introduction** discusses the effects of air pollution on human health, assessments of air pollution, initiatives to minimise its impact, the application of machine learning in air quality research, and the transition of machine learning to the edge. This chapter covers the development of low-cost air quality monitoring devices. The significant air quality data generated and transmitted to the cloud by these devices may result in data congestion. Thus, edge devices have become crucial in computing tasks, including executing machine learning algorithms. This chapter also describes the research aims and objectives.
- **Chapter 2 Background and Literature Review** covers relevant research background. This chapter examines a significant paradigm shift in air pollution monitoring. This approach has evolved from relying solely on standard government-operated networks to combining reference-level monitors and

emerging sensor technologies. The following discussion investigates the feasibility of integrating low-cost sensor nodes into an air quality monitoring system. Recent research on air quality prediction using machine learning methods is presented, followed by research related to edge computing. It also covers the most commonly used deep learning models for air quality prediction and methods for quantising these models. Furthermore, this chapter addresses the issue of missing data, a common occurrence when collecting air quality data. The chapter concludes by explaining air quality datasets and evaluation metrics utilised in this thesis.

- **Chapter 3 Deep Learning for Missing Data Imputation** discusses the implementation of deep learning for missing data imputation. Since air quality status exhibits spatiotemporal correlation, incorporating data from nearby stations can aid in estimating measurement values at a station with missing data. The background to this chapter includes an explanation of the types of missing data and the utilised dataset. The contributions of the chapter involve extracting air pollution features using a deep convolutional denoising autoencoder with spatiotemporal considerations, introducing a method well-suited for both short-period and long-interval consecutive scenarios, and minimising data exchange by incorporating only the pertinent pollutant data from neighbouring stations. Finally, the performance of the proposed model is evaluated and compared to commonly used imputation techniques.

- **Chapter 4 Optimising Deep Learning at the Edge** presents a novel deep learning model for air quality prediction. Unlike models that rely solely on a station's local data, the proposed model considers local and spatiotemporal data, resulting in more precise predictions. To reduce file size and execution time performed on edge, some post-training quantisation techniques are introduced. Finally, the original and quantised model characteristics and performances are assessed.

- **Chapter 5 Collaborative Edge Learning** discusses collaboratively- and locally-trained deep learning models. The application scenario includes Raspberry Pi boards and a Jetson Nano Developer kit as edge devices. The chapter further examines losses during training, model performances, and communication costs. Finally, expanding the number of participating edge devices and deriving mathematical formulations for each learning scenario is also explained.

- **Chapter 6 Tiny Machine Learning for Microcontroller Applications** focuses on developing machine learning for resource-constrained devices, such as microcontrollers. Despite their limited memory capacity and computing capability, microcontrollers are well-suited for low-power applications that involve sensors. The chapter discusses the implementation of tiny machine learning on a low-cost air quality monitoring device. This chapter also includes optimisations using binary weight networks and meta-learning approaches.

- **Chapter 7 Conclusions and Further Work** provides a summary of all preceding chapters. This chapter also presents an overview of the objectives stated in Chapter1 and highlights the corresponding accomplishments. Finally, potential areas for future work are also discussed.

Chapter 2

Background and Literature Review

2.1 Introduction

The global population is increasing, and the United Nations estimates the number will reach around 9.7 billion in 2050 [13]. A growing global population can contribute to urbanisation. Urbanisation is a major global trend that has significantly changed how humans and the environment interact in recent decades [60]. Rapid urbanisation has increased environmental problems, such as toxins and signs of climate change, affecting both developed and developing countries [61]. Globally, most people reside in urban areas, comprising 55% of the world's population in 2018. In 1950, only 30% lived in cities, and projections anticipate that by 2050, 68% of the global population will be living in urban areas [62]. Currently, the regions with the highest levels of urbanisation include Northern America (82%), Latin America and the Caribbean (81%), Europe (74%), Oceania (68%), and Asia (50%). In Africa, most people live in rural areas, with 43% in cities [63].

The growth in economic activity from urbanisation and improved living standards has led to environmental problems. Industrial and urban growth makes balancing environmental protection and economic growth increasingly difficult [64, 65]. Excessive releases of greenhouse gases from industrial activities have exacerbated environmental damage and climate change, endangering global security and human well-being [64]. Urban air pollution, which results from expanding urban areas, improving industrial technology, and improving transportation, poses health risks and contributes to the disruption of the atmosphere and ecosystems. Urban air pollution continues to be a significant issue in many cities worldwide [66]. In pursuing

environmental sustainability, many interested parties have developed air pollution monitoring systems to measure, analyse and predict the concentration of most critical air pollutants [2].

Measuring air pollution and its related factors is frequently integrated into a smart city framework [67]. The smart city concept emerged by combining Information and Communication Technology (ICT) with fixed/mobile sensors placed throughout the city, and this approach has transformed into a sustainable urban data source [66]. Under the smart city framework, environmental health scientists and other parties can relate the measured air contaminants with potential exposures to health impacts [68]. For example, potential exposures tend to occur during traffic congestion. The smart system can propose the best route for drivers towards their target destinations by avoiding traffic congestion and minimising the health impact caused by air pollution [69]. Another example can be seen in smart hospital concepts [70]. Intelligent air pollution sensors located in the field can be connected to the hospital system to improve the quality of medical care. Furthermore, the vast majority of individuals, approximately 80-90%, spend a significant portion of their time indoors, encompassing various settings like homes, schools, and offices [71]. Therefore, monitoring indoor air quality becomes an important aspect to ensure overall well-being and health. Understanding and addressing the factors that contribute to indoor air quality is critical to creating an environment that supports the health and comfort of its occupants [72, 73, 74]. Nowadays, smart city initiatives are being adopted globally [75].

As discussed in Chapter 1.2, monitoring and assessing pollution levels and air quality are crucial for maintaining public health, preserving the environment, and guiding policy decisions. These activities are integrated into the smart city framework, involving the use of air quality monitoring instruments. Precise and traceable air quality monitoring devices built to industrial standards are costly. The prices of these devices can vary significantly, typically ranging from €5,000 to €30,000 [76]. These devices generally have large dimensions (not optimised for mobile instruments) and require dedicated installation space. Regular maintenance and calibration are essential to ensure the accuracy of data. The data produced by these devices serve as a reference for other devices. While these devices deliver precise results, their high costs and stringent maintenance requirements restrict their widespread deployment, resulting in a region's sparse network of air quality monitors. In small cities, the availability of standard air quality monitoring instruments may be limited or nonexistent.

There has been a paradigm shift in air pollution monitoring, transitioning

from reliance on standardised government-operated networks (using standard air quality monitoring instruments) to a combination of reference-grade monitors and emerging sensor technologies [77, 78]. Recent research has demonstrated the feasibility of low-cost sensor nodes for air quality monitoring systems. The growing utilisation of low-cost air quality devices is attributed to their ability to lower production costs, compact size, and enhanced mobility [79, 80, 81]. This emerging sensor-based air quality monitoring field can provide high-density spatiotemporal pollution data, supplementing the established methodology with more precise and expensive devices [53]. The global deployment of small, low-cost, interconnected air pollution sensors has spurred communities worldwide to expand city-wide sensor networks. This aims to comprehensively understand the air pollution levels citizens encounter daily [67].

In their work, Snyder *et al.* (2013) summarised the paradigm shift in air pollution monitoring, as depicted in Fig. 2.1. The new paradigm emphasises using low-cost air quality monitoring devices for collecting air quality data. This shift is not limited to government agencies; it includes communities, hobbyists, and individuals interested in monitoring air pollutants. Citizen science initiatives leverage community-based participatory monitoring and crowd-sourcing, where individuals voluntarily gather extensive data that is later compiled and analysed [78]. An example of a low-cost air quality monitoring device is shown in Fig. 2.2

2.2 Machine Learning for Air Pollution Prediction

Pollutant level prediction is usually associated with various meteorological factors, such as wind speed, wind direction, relative humidity, precipitation, barometric pressure, and solar radiation. All these recorded factors can be categorised as a time-series problem. The existing methods for time-series forecasting are mainly categorised into three methods: *deterministic* methods, *statistical* methods, and *machine learning methods*. Deterministic methods use mathematical equations to determine the numerical model of air pollution. It is based on the understanding of aerodynamic, physicochemical and environmental knowledge. This method needs high-speed calculation and simulation to predict the atmospheric pollutant concentration [83], which is considered not viable for edge devices due to their limited energy capacity and computation resource [84].

Statistical methods implement statistic-based models by trying to find the relationship between influencing factors (meteorological data, spatiotemporal factors, and others) and air pollutants [85]. There are some commonly used statistical

methods. These methods include the multiple linear regression (MLR) methods [86, 87, 88] the autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) methods [89], [90]. However, these models are based on linear assumptions that affect their prediction accuracy for commonly non-linear real problems. To overcome these problems mentioned above, researchers implement a non-linear machine learning approach, such as multilayer perceptron (MLP) model [91], [92], radial basis function (RBF) model [93], support vector machine (SVM) model [94], and artificial neural network (ANN) model [95], and neuro-fuzzy model [96].

The immense volume of collected spatiotemporal data from low-cost air quality monitoring devices has provided a better opportunity to apply machine learning (ML) techniques in air quality areas, such as air contaminant predictions [1, 97], missing data imputations [2, 98], classification tasks [99], or even personal pollutant exposure [100, 101]. In recent years, deep learning (DL), a subset of machine learn-

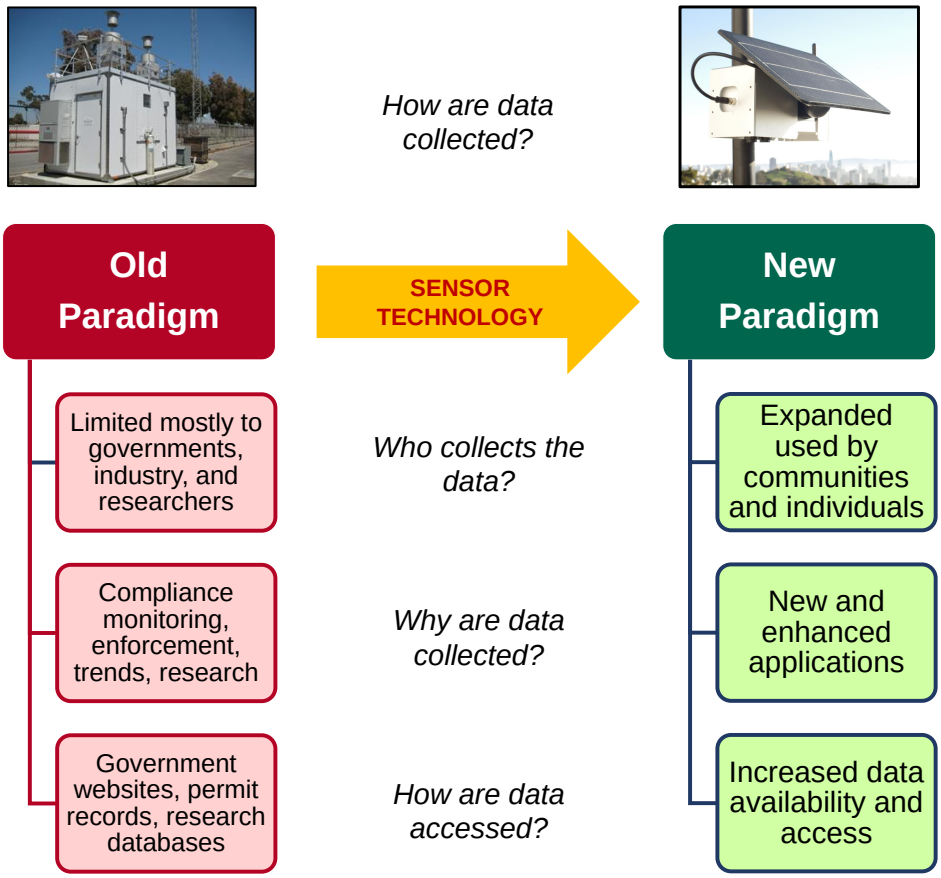


Figure 2.1: Shifted paradigm of air pollution monitoring [78].

Figure 2.2: An example of a low-cost air quality monitoring device called AQmesh installed at an airport (donated by Environmental Instruments Ltd.) [82].

ing, has been actively explored by many researchers. Deep learning methods can effectively learn the features from a complex and large amount of data [102]. This thesis exclusively focuses on deep learning-based methods as the foundation for all proposed techniques.

2.3 Machine Learning at the Edge

2.3.1 Edge Computing

Machine learning (ML), especially deep learning (DL), has gained popularity across various applications, such as image recognition and pattern matching. By learning features from input sets, deep learning enables the discovery of good representations, often achieved on multiple levels. Deep learning is more resilient to noise and able to deal with non-linearity. Instead of relying on hand-crafted features, deep learning automatically extracts the best possible features during training. During training, the deep neural network architecture can extract very coarse low-level features in its first layer, recognise finer and higher-level features in its intermediate layers and achieve the targeted values in the final layer [103]. However, deep learning models require substantial computational resources and often rely on cloud computing platforms for training and evaluation. However, in recent years, a new trend in deep learning has emerged, bringing computation to the *edge* [103].

Edge computing refers to deploying computation closer to data sources rather

than more centrally, as is the case with cloud computing [104]. It can address latency, privacy and scalability issues faced by cloud-based systems [105, 106]. In terms of latency, moving computation closer to the data sources decreases end-to-end network latency. Regarding privacy, the computation performed at the edge or a local trusted edge server prevents data from leaving the device, potentially reducing the chance of cyber-attacks. Regarding scalability, edge computing can avoid network bottlenecks at central servers by enabling a hierarchical architecture of edge nodes [107]. Moreover, edge computing can address energy-aware and bandwidth-saving applications [108]. Incorporating intelligence directly into edge devices is now feasible to facilitate data processing and information inference. This can be achieved by leveraging machine learning (ML) or deep learning algorithms [109, 110]. Deep learning [111] can now be implemented on edge devices, such as mobile phones, wearables and the Internet of Things (IoT) nodes.

2.3.2 Machine Learning Platform

Embedded machine learning is the application of machine learning technology on hardware with limited resources. This involves the development of models that can work efficiently on devices with small memory and processing power [112]. This discipline is experiencing substantial expansion in scale and scope, driven by advances in system performance and the development of more sophisticated machine learning models. This growth is increasingly driven by increased affordability and greater accessibility. Consequently, there is a marked improvement in the quality, power consumption efficiency and overall effectiveness of these systems [113].

A machine learning platform is a comprehensive software setup that provides tools, libraries, and resources to make it easy to build, deploy, and manage machine learning models and applications. This platform is designed to increase the efficiency of the machine learning process, from preparing data and building models to training, evaluation, deployment, and monitoring. Many machine learning platforms are accessible, including TensorFlow [114, 115], PyTorch [116], Keras [117], Caffe [118], and MATLAB [119], among others. The choice of a machine learning platform depends on the developer or researcher's project requirements, skill level, and preferences. Each platform has its own strengths and weaknesses, and the choice depends on the context of use. For example, TensorFlow is a robust and mature deep-learning library renowned for its formidable visualisation capabilities, production-ready deployment solutions, and extended support to mobile platforms. Conversely, PyTorch provides flexibility, robust debugging capabilities, dynamic computational graphs, efficient memory usage, and shorter training dura-

tions [120, 121].

This thesis constructs all deep learning models using the TensorFlow (TF) framework, which can be freely downloaded from the official website: `https://www.tensorflow.org`. The versions utilised were 2.2, 2.4, and 2.12. TensorFlow is a comprehensive platform designed to simplify the process of building and deploying machine learning models. It provides solutions to enhance the efficiency of machine learning tasks at all workflow stages. With TensorFlow, users have the flexibility to build machine learning models using features such as the Keras Functional and Model Subclassing Application Programming Interfaces (APIs). Additionally, TensorFlow supports an extensive ecosystem of add-on libraries and models, offering the opportunity to explore and experiment with various advanced capabilities [115].

TensorFlow offers TensorFlow Lite (TFLite), a lightweight version specifically designed to facilitate the deployment of TensorFlow models on edge devices. TensorFlow Lite has been instrumental in popularizing the use of machine learning in mobile and IoT devices, making the technology more accessible and applicable in various real-world scenarios. TensorFlow Lite is optimised to run on mobile and embedded devices with high efficiency. This includes support for ARM processors and lower resource usage than standard TensorFlow [122]. Bringing the standard concept to the lite version typically involves building, training, testing, and optimising deep learning models on a desktop computer. Once an optimised deep learning model is obtained through these steps, the model is deployed to the edge devices. To execute the model on the target hardware, the TensorFlow Lite Interpreter library is utilised, enabling seamless portability and execution of the deep learning model.

2.3.3 Quantised Neural Networks

Quantised Neural Networks (QNNs) can operate with lower precision for weights and activations than full-precision models. Generally, neural network models use high precision, such as floating-point 32-bit, to represent weights and activations [123]. In QNN, this precision is reduced to integers with fewer bits, such as 8-bit integers. The primary goal of QNN is to reduce the computational and memory resource requirements needed to run a neural network model without significantly sacrificing the model's performance. One approach to achieving quantised models is by performing *post-training quantisation*, meaning that the quantisation process is performed after the full-precision model has been trained. Figure 2.3 depicts the post-training provided by the TensorFlow framework.

In post-training quantisation, optimisation occurs after the training process has been completed. There are three post-training quantisation methods pro-

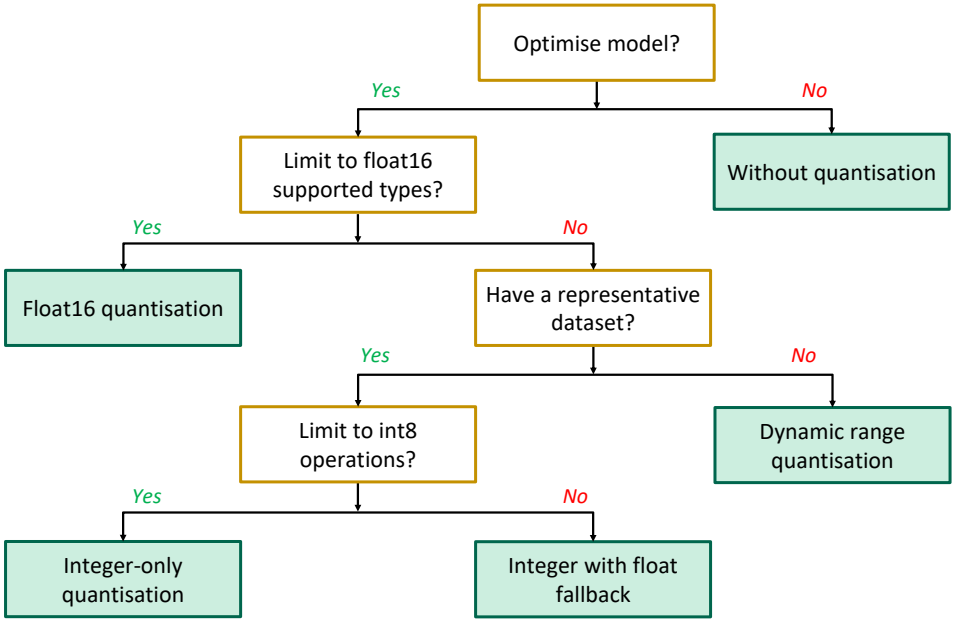


Figure 2.3: Post-training optimisation methods provided by TensorFlow (adapted from [124]).

vided by TensorFlow, namely *dynamic range quantisation*, *full integer quantisation* and *float16 quantisation*. Dynamic range quantisation statically quantises only the weights, from floating-point (32 bits) to integer (8 bits). During inference, weights are converted back from 8 bits to 32 bits and computed using floating-point kernels. Compared to dynamic range quantisation, full integer quantisation offers latency improvements.

Full integer quantisation supports two methods, namely *integer with float fallback* and *integer-only conversions*. The integer with float fallback means that a model can be fully integer quantised, but the execution falls back to float32 when operators do not have an integer implementation. The integer-only method is appropriate for 8-bit integer-only devices, such as microcontrollers and accelerators, e.g., EdgeTPU. In this method, the conversion fails if the model has unsupported operations. Finally, float16 quantisation converts weights to float16 (16-bit floating-point numbers).

2.3.4 Tiny Machine Learning

Tiny Machine Learning (TinyML) is an innovative field within machine learning focusing on edge devices such as microcontrollers and low-power embedded systems [125]. Unlike powerful desktop computers or cloud servers, microcontrollers

typically have limited memory capacity and computing power. Effective deployment of tinyML models requires a thorough understanding of hardware, software, algorithms, and applications. However, they offer the advantage of sensing environmental parameters through sensors. Microcontrollers can be employed to generate data, forming the fundamental backbone of machine learning disciplines. From these data, microcontrollers can perform decisions based on ML algorithms [126].

As an illustration in air pollution research, microcontrollers can be equipped with various sensors to capture physical quantities of air pollutants, such as particulate matter, carbon monoxide, sulfur dioxide, ozone, etc. These physical parameters are translated by an analog-to-digital converter, one of the unique peripherals that microcontrollers possess. Microcontrollers collect data from sensors periodically, forming time series data. The collected data usually includes the concentration of pollutants in the air at a specific time. Microcontrollers can perform initial data processing, such as calibration, normalisation, or noise filtering. The data recorded by the microcontroller can then be stored through storage media, such as an SD card. This recorded data can then be used to create a machine learning model. This modelling is generally not done directly on the microcontroller, as microcontrollers have limited computing capabilities. The pre-trained model obtained is then brought to the microcontroller using the tiny machine learning libraries for microcontrollers. The embedded deep learning model in the microcontroller then receives this input data from direct measurements and can produce the desired results, for prediction purposes or classification of certain pollutant data.

An example of work related to tinyML for air quality monitoring has been published by Botero-Valecia *et al.* [127]. They developed an affordable, open-source station for measuring pollution, suitable for outdoor and indoor environments. This station can assess air pollution, noise, light, relative humidity and ambient temperature. It employs tinyML technology to linearise the variables against reference values, effectively minimising measurement errors. In another instance, Sakr *et al.* [128] implemented machine learning models on a range of STM 32-bit microcontrollers. Their study utilised multiple datasets, including one focused on the air quality index (AQI). To address different research questions, they explored various machine learning algorithms, such as Linear Support Vector Machine (SVM), k-nearest Neighbours (k-NN), and Decision Trees. These questions encompassed comparative analysis of inference performance, training duration, data pre-processing, and hyperparameter tuning.

2.4 Edge Devices

The term *edge* refers to devices positioned close to data sources. Based on the specific applications in use, various devices can be classified within this category, including but not limited to software programmable platforms, Application Specific Integrated Circuits (ASICs), and Field-Programmable Gate Arrays (FPGAs) [129].

2.4.1 Software Programmable Platforms

Software programmable platforms encompass a range of devices, notably Central Processing Units (CPUs) and Graphics Processing Units (GPUs). CPUs provide the computational infrastructure for executing machine learning models. The widespread prevalence of generic CPU architectures and their cost-effectiveness has established this category as advantageous for neural network (NN) execution. It is important to note that CPU capabilities can exhibit substantial variability based on the specific computing platform they are integrated into. For instance, server-grade CPUs generally boast higher capabilities than personal computers, surpassing the computational power of embedded microprocessors like single-board computers or microcontrollers.

Initially developed for accelerating computer graphics and image processing, Graphics Processing Units (GPUs) have found their utility in various electronic devices necessitating graphical processing. These include video cards, personal computers, workstations, mobile phones, game consoles, and more. In the neural networks (NN) domain, GPUs have substantial attention for training and inference tasks. Their parallel computing architecture has significantly cut execution times for training and inference while maintaining the user-friendly nature of software programming. NVIDIA, in particular, has emerged as a key player in manufacturing GPUs tailored for general-purpose computing applications [130].

2.4.2 Application Specific Integrated Circuits

An ASIC is a special computer chip that integrates several circuits onto a single chip. This unique architecture allows customisation for specific tasks rather than catering to general-purpose applications. ASICs are specifically designed for particular tasks. This allows maximum optimisation in terms of performance. ASIC chips can be customised for the desired task, resulting in better performance than general-purpose chips or FPGAs. Because ASICs are designed for a specific purpose, they can be optimised for highly efficient power use. This is important in applications that require low power consumption, such as battery-operated or mo-

bile devices. ASICs usually have a minimal physical size because they only contain the components needed for a specific task. This makes it ideal for devices that require a compact form factor [131]. When applications require mass production, ASICs can be a more cost-effective option in the long run. In the machine learning field, ASICs can also serve as accelerators for training machine learning or deep learning models [132].

2.4.3 Field-Programmable Gate Arrays

FPGAs are semiconductor integrated circuits (ICs) that can be customised and reconfigured to meet specific needs. Its computing role sets it apart from other devices in the computing world (CPUs and GPUs) and dedicated accelerators such as ASICs. Unlike CPUs and GPUs, which adhere to a fixed hardware architecture for program execution, FPGAs and ASICs empower the creation of purpose-built hardware optimised for specific program tasks. Although ASICs generally outperform FPGAs in certain assignments, ASIC development requires substantial time and financial commitment. On the other hand, FPGAs offer a more budget-friendly and quickly accessible option, allowing reprogramming for each new application [133].

Figure 2.4 illustrates the internal architecture of the FPGA. These arrangements consist of a limited set of general-purpose routing resources and Configurable Logic Blocks (CLBs), also known as slices. Each CLB consists of Look-Up-Table (LUT) memories capable of executing various logic functions and synchronous memory elements, known as Flip-Flops (FFs). To facilitate connectivity, each CLB is

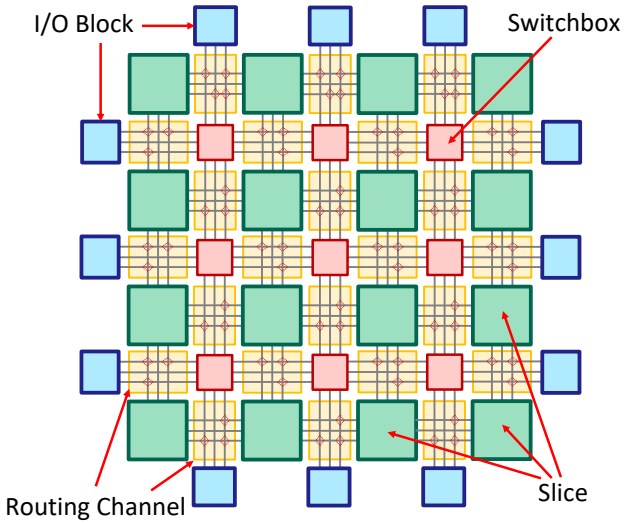


Figure 2.4: Internal structure of an FPGA (adapted from [134]).

connected to a routing channel that can be configured to connect the I/O CLB to a programmable interconnect. This routing channel interacts with the switchboxes, facilitating connections between accessible routing channels.

In the field of air quality research, Ramírez-Montañez *et al.* engineered a modified LSTM (Long Short-Term Memory) neural network, executed on an FPGA board for modelling and forecasting pollutants like nitrogen dioxide, carbon monoxide, and particulate matter [135]. This method achieved an 11% enhancement in performance relative to the standard LSTM network architecture. The findings highlight that the modified architecture effectively retains its operational capabilities even with a reduced neuron count in the initial layers. In a separate study, Abbasi *et al.* developed and implemented a device for atmospheric testing in confined spaces, utilising FPGA technology [136]. This device integrates various gas sensors into an FPGA-based system, including those for oxygen, methane, and nitrogen dioxide. This system is designed to provide field users with warning signals based on the sensor readings.

2.4.4 Computing Platform Selection

This thesis focuses on using development boards as edge devices. The development board can be categorised into *single-board computers* (SBCs) and *microcontrollers*. Single-board computers (SBCs) can incorporate both Central Processing Units (CPUs) and Graphics Processing Units (GPUs), whereas microcontrollers predominantly rely on CPUs as their primary processor. More recently, microcontrollers have seen the integration of neural network accelerators, called Neural Processing Units (NPUs), into their systems. For example, the Ethos-U NPU can be connected to a Cortex-M series CPU, effectively minimising inference time and memory demands essential for the execution of machine learning models [137].

In this study, using FPGA and ASIC is considered too complicated and demanding. In contrast, widely used single-board computers can smoothly run machine learning models created using TensorFlow, while a microcontroller board can efficiently manage light versions with minimal effort. Hence, this thesis exclusively employs Single-Board Computers (SBCs) and microcontrollers as edge devices for conducting experiments. Furthermore, all the utilised microcontrollers can execute machine learning models seamlessly without requiring a dedicated NPU.

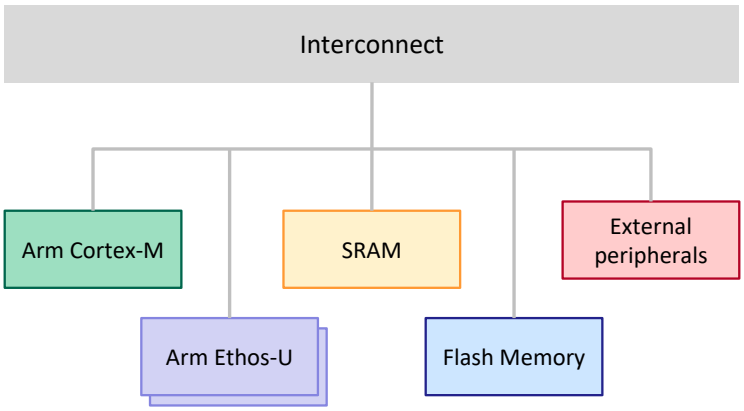


Figure 2.5: Ethos-U system coupled with a Cortex-M series CPU (adapted from [137]).

2.4.4.1 Single Board Computers

Single-board computers (SBCs) consolidate all the components to operate a functional computer onto a single board. With a compact form factor (roughly the size of a credit card), SBCs offer a streamlined solution for edge computing applications. Specifically, the SBCs utilised in this research are from two companies: Nvidia [138] and Raspberry Pi [139]. Chapter 5 of this thesis utilises the Nvidia Jetson Nano 2GB developer board. For the experimentation conducted in Chapter 4 and Chapter 5, the Raspberry Pi 4 Model B (RPi 4B) and Raspberry Pi 3 Model B+ (RPi 3B+) boards are utilised. Additionally, the Raspberry Pi Zero W (RPi 0W) boards are exclusively used in Chapter 5. Fig. 2.6 shows the physical appearance of the single-board computers used in this thesis. The standard size, measuring 85.6mm $\times$ 56.5mm, is consistent across all Model B versions of the Raspberry Pi, while the zero size, measuring 65mm $\times$ 30mm, is uniform for all Raspberry Pi Zero versions.

All single-board computers are equipped with specific operating systems (OS). In this thesis, the Jetson Nano board utilises Ubuntu 18.04 as its operating system, whereas the Raspberry Pi boards employ Raspberry Pi OS (Buster version) as their respective operating system [140]. While these single-board computers can support essential input/output peripherals such as keyboards and display monitors when equipped with an operating system, they are operated in headless mode during experiments. In this mode, users interact with the SBCs remotely using the SSH (Secure Shell) protocol to send commands and access its functionality. The SSH approach eliminates the need for a physical interface, making SBCs suitable for applications where space constraints or remote operation are priorities. Most importantly, these SBCs can run TensorFlow, a widely-used framework for building

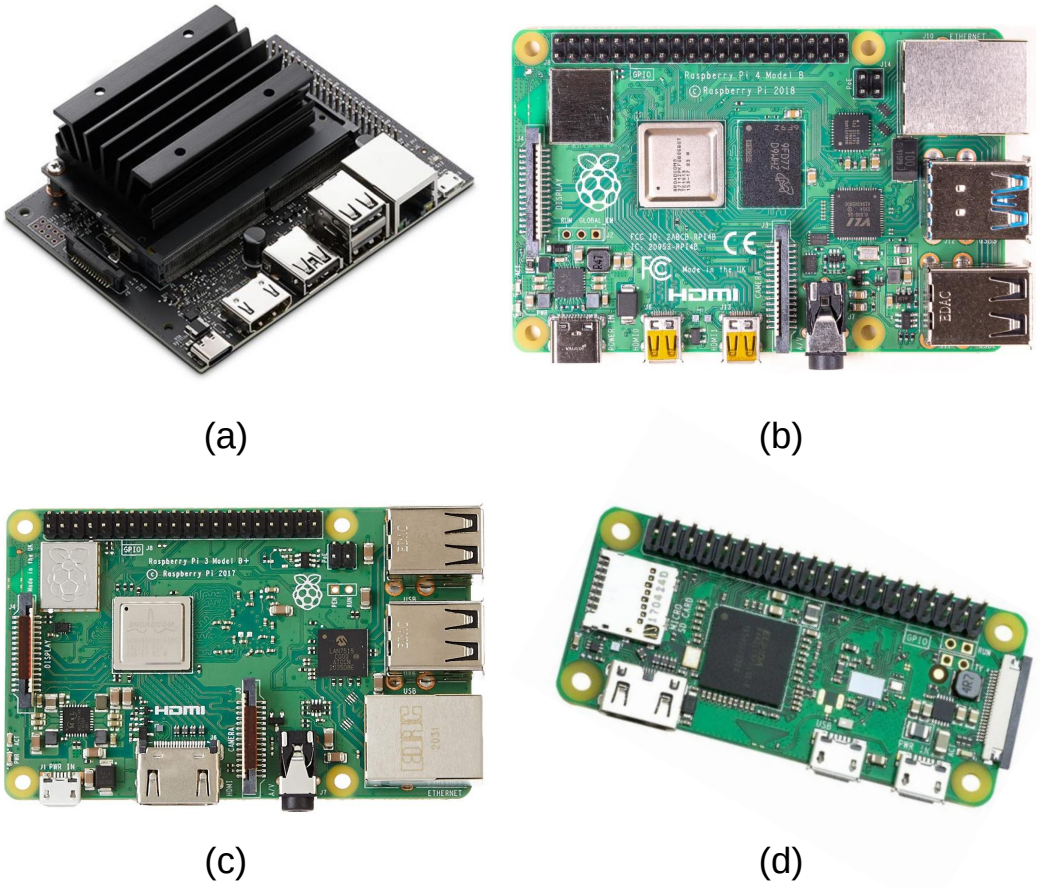


Figure 2.6: Edge devices: (a) Jetson Nano 2GB, (b) RPi 4B, (c) RPi 3B+, and (d) RPi 0.

and deploying deep learning models [114].

Technical specifications for Jetson, RPi 4B, RPi 3B+, and RPi 0 boards are shown in Table 2.1, Table 2.2, Table 2.3, and Table 2.4, respectively. The RPi 4B stands out for its superior computing capabilities among the Raspberry Pi boards.

Table 2.1: Nvidia Jetson Nano 2GB Developer Kit technical specifications.

Selected Features	**Details**
CPU	Quad-core ARM A57 @1.43 GHz
GPU	128-core Maxwell^{TM} GPU
RAM	2 GB 64-bit LPDDR4 SDRAM
Storage	MicroSD card
Input Power	5V/2.5A DC (minimum) via USB-C connector

Table 2.2: Raspberry Pi 4 Model B technical specifications.

Selected Features	**Details**
SoC	Broadcom BCM2711
CPU	Quad-core Cortex-A72 (ARMv8) 64-bit @1.5 GHz
GPU	Broadcom VideoCore VI @500 MHz
FPU	VFPv4 + NEON
RAM	Up to 8GB LPDDR4 SDRAM
External storage	MicroSD card
Power	$\approx 3-4$W (idle), ≈ 6W (stress load)

Table 2.3: Raspberry Pi 3 Model B+ technical specifications.

Selected Features	**Details**
SoC	Broadcom BCM2837B0
CPU	Quad-core Cortex-A53 (ARMv8) 64-bit @1.4 GHz
GPU	Broadcom VideoCore IV @250 MHz
FPU	VFPv4 + NEON
RAM	1 GB LPDDR2 SDRAM
External storage	MicroSD card
Power	≈ 1.15W (idle), ≈ 3.6W (stress load)

Table 2.4: Raspberry Pi Zero W technical specifications.

Selected Features	**Details**
SoC	Broadcom BCM2835
CPU	Quad-core Cortex-A72(ARMv8) 64-bit @1.5GHz
GPU	Broadcom VideoCore IV @ 400 MHz
FPU	VFPv2
RAM	512 MB
Storage	MicroSD card
Power	≈ 0.5W (idle), ≈ 1W (stress load)

However, this board consumes the highest power compared to other models during idle periods and under stress load.

2.4.4.2 Microcontrollers

The microcontroller is specifically utilised in Chapter 6 of this thesis. One of the primary advantages of utilising microcontrollers is their ability to interface with sensors to **generate data**, which is a fundamental requirement for building a machine learning workflow. To generate data, microcontrollers are commonly equipped with an analogue-to-digital converter (ADC), which converts analogue quantities collected by sensors into digital forms suitable for internal microcontroller operations [141]. In addition to acquiring signals from analogue sensors, microcontrollers often include digital serial communication interfaces such as serial peripheral interface (SPI), Inter-integrated Circuit (I2C), or serial universal asynchronous receiver-transmitter (UART). These digital serial communication interfaces have become popular choices for commercial sensors.

This thesis employs microcontrollers to run the tiny machine learning (tinyML) algorithm. The term *tiny* is typically associated with small devices like microcontrollers, as they consume significantly less power than SBCs. For instance, even the smallest Raspberry Pi (RPi 0) can consume several hundred milliwatts, whereas the Nvidia Jetson can consume approximately 12 watts when operating at full speed [140]. Despite resource and computing power limitations, microcontrollers are highly popular and widely used devices for embedded and Internet of Things (IoT) applications. For instance, the Raspberry Pi Pico, utilised in this thesis, recorded sales of nearly two million units within the first 18 months of its release [142]. Another notable example is ST Microelectronics, which has shipped over 6 billion STM32 32-bit Arm Cortex-M-based microcontrollers worldwide since its introduction in 2007 (the report is accessed on 4 July 2023) [143]. These numbers highlight the extensive usage and significance of microcontrollers in various applications. An example of an Arm Cortex-M-based microcontroller is the Raspberry Pi Pico W microcontroller developed by Raspberry Pi in the UK [144]. RP2040 chip powers the development board and is currently priced at approximately £6.3 as of the time of writing this thesis. Fig. 2.7 shows the development board, and Table 2.5 presents the technical specifications for the RPi Pico W.

2.5 Neural Networks

2.5.1 Artificial Neuron

Artificial Neural Networks (ANNs) consist of interconnected layers of nodes, also called *neurons*. These nodes perform mathematical computations to recognise pat-

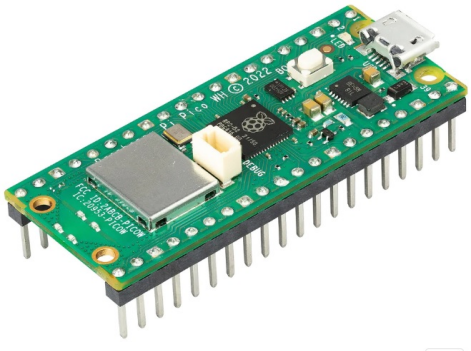


Figure 2.7: Raspberry Pi Pico W microcontroller development board.

Table 2.5: Raspberry Pi Pico W technical specifications.

Selected Features	**Details**
Chip	RP2040
Processor	Dual ARM Cortex-M0+ up to 133MHz
Memory	264kB of SRAM, 2MB of onboard Flash memory
Wireless	IEEE 802.11n wireless LAN
	Bluetooth 5.2
GPIO	26 multi-function GPIO pins
Peripherals	2 × UART, 2 × I2C, 2 × SPI, 16 × PWM channels

terns within data. ANNs are designed to simulate the behaviour of human neurons. An artificial neuron is summarised in Fig. 2.8. In an artificial neuron, multiple input values are received along with their respective weights. These weighted inputs are then summed, and an activation function is applied to generate the output. This output is subsequently passed on to other neurons in the network. In the case of the final layer, it serves as the overall output of the network.

2.5.2 Convolutional Neural Network

As depicted in Figure 2.9, a typical CNN model consists of two layers: 1) convolutional layers and 2) fully connected layers. The convolutional layers are situated immediately after the input layer, and their output is subsequently passed to the fully connected layer. The primary function of the convolutional layer is to extract features through convolutional operations and other operations. To classify or predict the desired values, the fully connected layer acts as a decision-making block, utilising the extracted features generated by the convolutional network.

Many articles focus on two-dimensional Convolutional Neural Network (2D CNN) models. The 2D CNNs work best for image classification problems. The same approach can be applied to data sequences (time-series data) by implementing one-dimensional (1D) CNN. A 1D CNN model learns to extract features from time-series data and maps the internal features of the sequence. This model efficiently gathers information from raw time-series data directly, especially from shorter (fixed-length) segments of the overall dataset.

Figure 2.10 illustrates how the feature detector (or kernel) of the 1D CNN slides across the features. In this thesis, these features are air pollutants and meteorological data. Examples of pollutant data are $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2 and O_3. Meteorological data, including temperature, air pressure, dew point, wind direction and wind speed, are often combined with pollutant data as the input features.

If the input data to the convolutional layer of length n are denoted as x, the kernel of size k as h and the kernel window is shifted by s positions, then the output y is defined as:

$$y(n) = \begin{cases} \sum_{i=0}^{k} x(n+i)h(i) & \text{if } n = 0 \\ \sum_{i=0}^{k} x(n+i+(s-1))h(i) & \text{otherwise} \end{cases} \tag{2.1}$$

For example, if $n = 6$, $k = 3$ and $s = 1$, then the output will be:
$y(0) = x(0)h(0) + x(1)h(1) + x(2)h(2)$
$y(1) = x(1)h(0) + x(2)h(1) + x(3)h(2)$
$y(2) = x(2)h(0) + x(3)h(1) + x(4)h(2)$
$y(3) = x(3)h(0) + x(4)h(1) + x(5)h(2)$

If it is assumed that there is no padding applied to the input data, then the

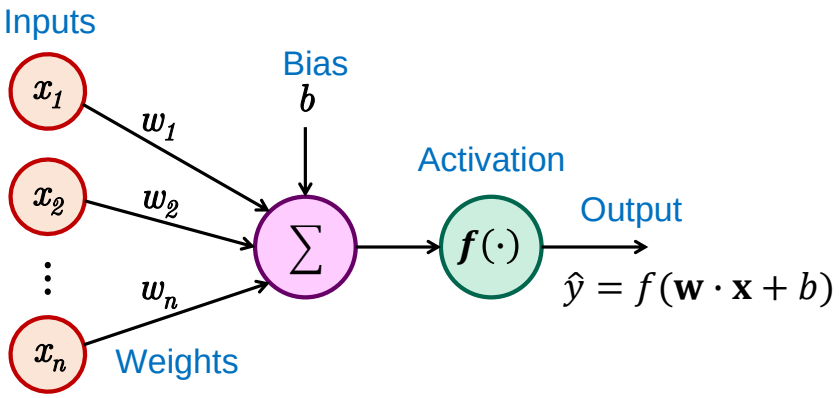


Figure 2.8: An artificial neuron.

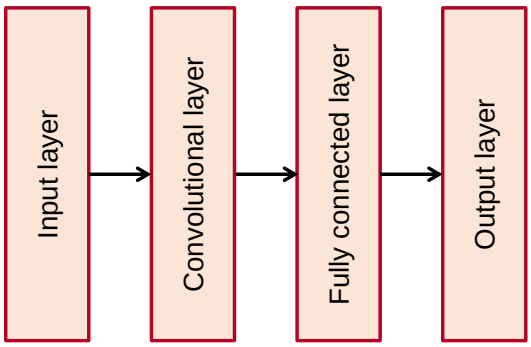


Figure 2.9: A typical CNN model.

length of output data o is given by:

$$o = \lfloor \frac{n-k}{s} \rfloor + 1 \tag{2.2}$$

Therefore, the length of y based on the example mentioned above is $o = (6-3)/1 + 1 = 4$.

Apart from the convolutional layer, another commonly used layer in convolutional neural networks is the *pooling layer*. Typically placed after the convolutional steps, the pooling layer is responsible for downsampling the dimensions of the convolution output. There are different types of pooling layers, including *max pooling* and *average pooling*. In Fig. 2.11, it can be observed that max pooling selects the

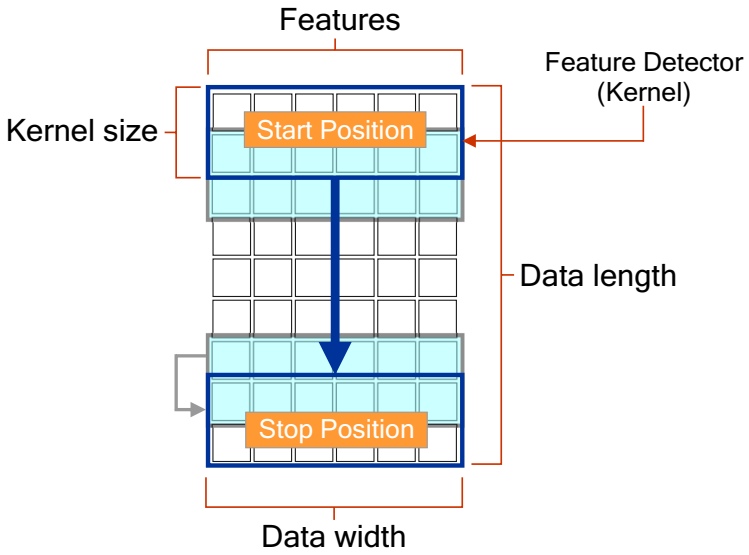


Figure 2.10: The feature detector of 1D CNN slides over time-series data.

maximum value within the pooling window, while average pooling computes the average value within the window.

The convolutional layers in a neural network can produce output with multiple dimensions. A *flattening* process is employed to prepare CNN output for fully connected layers, as shown in Fig. 2.12. This process reduces the output dimensionality and creates a flattened structure suitable for further processing in fully connected layers.

2.5.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) [145] is a structural modification of the Recurrent Neural Network (RNN) that adds memory cells in the hidden layer so that it can be implemented to control the flow of information in time-series data. Figure 2.13 shows the LSTM network cell structure.

As shown in Figure 2.13, the network inputs and outputs on the LSTM structure are described as follows:

$$F_t = \sigma(W_f \cdot [H_{t-1}, X_t] + b_f) \tag{2.3}$$

$$I_t = \sigma(W_i \cdot [H_{t-1}, X_t] + b_i) \tag{2.4}$$

$$\tilde{C}_t = \tanh(W_c \cdot [H_{t-1}, X_t] + b_c) \tag{2.5}$$

$$C_t = F_t * C_{t-1} + I_t * \tilde{C}_t \tag{2.6}$$

$$O_t = \sigma(W_o \cdot [H_{t-1}, X_t] + b_o) \tag{2.7}$$

$$H_t = O_t * \tanh(C_t) \tag{2.8}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.9}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.10}$$

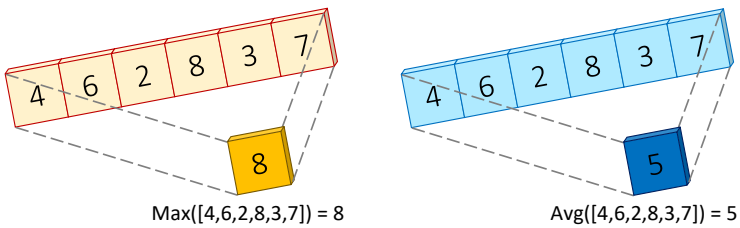


Figure 2.11: Examples of pooling layer operations.

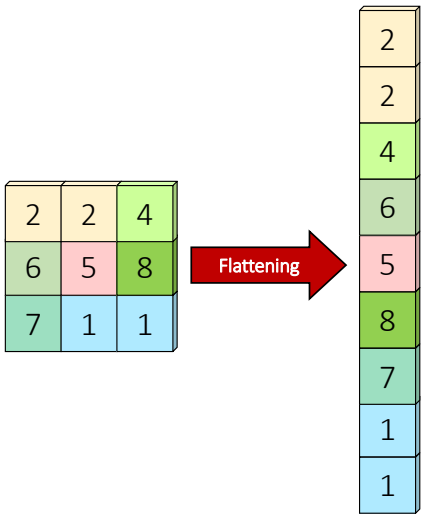


Figure 2.12: An example of flattening operation.

with W_f, W_i, W_c and W_o as input weights; b_f, b_i, b_c and b_o as biases; t the current time; $t-1$ the previous state; X the input; H the output; and C the status of the cell. The notation σ is a sigmoid function, which produces an input between 0 and 1. A value of 0 means not allowing any value to pass to the next stage, while a value of 1 means letting the output fully enter the next stage. The hyperbolic tangent function (tanh) is used to overcome the loss of gradients during the training process, which generally occurs in the RNN structure.

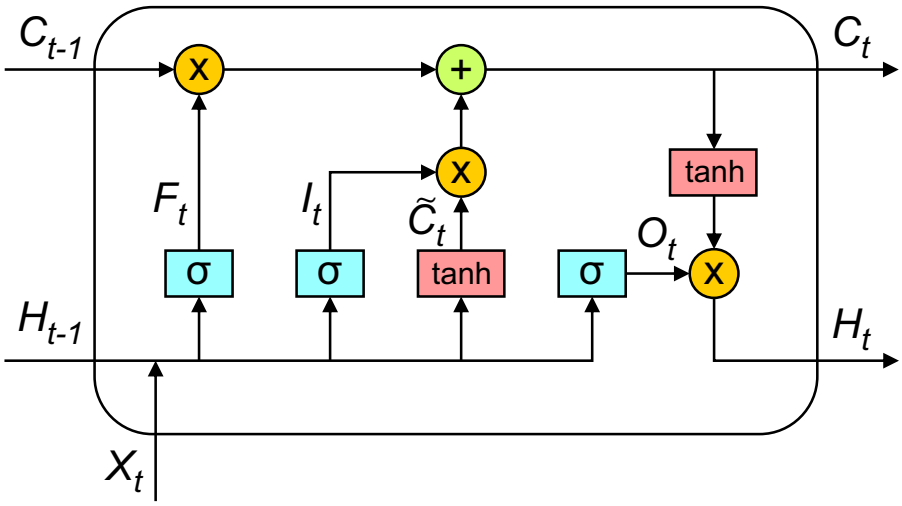


Figure 2.13: An LSTM cell structure.

2.6 Evaluation Metrics

There are several methods usually used to evaluate the model performance. This thesis uses root mean square error (RMSE) and mean absolute error (MAE). Another broadly used method to evaluate model performance in machine learning studies is the coefficient of determination (R^2 or R-squared). Also, mean absolute percentage error (MAPE) is also used as an evaluation metric, especially in Chapter 5.

Chicco *et al.*[146] suggested implementing R^2 for regression task evaluation as this method is more informative to qualify the regression results. However, the limitation of R^2 arises when the calculated score is negative. In this case, the model performance can be arbitrarily worse, but it is impossible to recognise how bad a machine learning model performed [147]. RMSE, MAE, MAPE, and R^2 can be calculated using Equations (2.11), (2.12), (2.13), and (2.14), respectively.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2} \tag{2.11}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y}_i| \tag{2.12}$$

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{2.13}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.14}$$

where n is the total number of data samples, y_i are the actual values, $\hat{y}_i$ are the predicted values, and $\bar{y}$ is the overall mean of the actual values.

Following work conducted by Ma *et al.* [148], this thesis implemented a rate of improvement on RMSE (RIR) to measure the performance of our methods in comparison with the existing imputation techniques. The RIR is calculated using the following equation:

$$RIR^{A,B} = \frac{RMSE^A - RMSE^B}{RMSE^A} \times 100\% \tag{2.15}$$

where, $RMSE^A$ denotes the RMSE value of the benchmarked method and $RMSE^B$ is the RMSE value of the proposed method.

2.7 Summary

The surge in economic activity due to urbanisation and rising living standards has led to significant environmental challenges. Expanding industrial and urban areas complicates the balance between ecological preservation and economic development. Industrial activities, particularly the excessive emission of greenhouse gases, have intensified environmental degradation and climate change, posing risks to global security and human well-being. Consequently, integrating air pollution measurement into smart city initiatives has become increasingly important. Within the smart city framework, environmental health scientists and other stakeholders correlate air pollutant levels with potential health impacts. Monitoring and evaluating pollution and air quality are essential for protecting public health, conserving the environment, and informing policy-making.

Air pollution monitoring has shifted from exclusive reliance on standardised government-operated networks to a hybrid approach incorporating reference-grade monitors and novel sensor technologies using low-cost air quality devices. The substantial volume of spatial and temporal data gathered by low-cost air quality monitors presents enhanced opportunities for applying machine learning (ML) techniques in air quality. ML, particularly deep learning (DL), has gained prominence in various domains, including air quality research. However, deep learning models demand significant computational resources and often depend on cloud computing for training and evaluation.

Recently, a trend towards edge computing in deep learning has emerged, shifting computational processes closer to data sources rather than centralising them in cloud systems. Edge computing can solve issues of latency, privacy, and scalability that cloud-based systems encounter. "Edge" in this context refers to devices located near data sources. Depending on the application, a range of devices, including software programmable platforms, Application Specific Integrated Circuits (ASICs), and Field-Programmable Gate Arrays (FPGAs), fall into this category. Machine learning at the edge is closely associated with Embedded machine learning, which involves applying ML on hardware with constrained resources. This requires developing models that function efficiently on devices with limited memory and processing capabilities. To adapt to these constraints, researchers often modify standard ML models to lower precision through quantisation, enabling these models to function with reduced precision in weights and activations compared to full-precision models. One method to achieve quantised models is post-training quantisation, where the quantisation process is applied after training the full-precision model.

Chapter 3

Deep Learning for Missing Data Imputation

This chapter is based on work published in:

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Estimation of missing air pollutant data using a spatiotemporal convolutional autoencoder," *Neural Comput & Applic*, vol. 34, no. 18, pp. 16129–16154, Sep. 2022 [2].

3.1 Introduction

Predicting air pollution by its potential exposures and health impacts can be even more challenging when there are missing values in measurement data. The presence of missing data can impact the interpretation of the data being processed and the research conclusions [149]. Moreover, missing data can ultimately affect the function of public services related to air quality [150]. Missing data is a common problem in air pollutant measurement processes and can often be experienced in other fields such as clinical, energy, traffic, etc [151, 152, 153]. The causes of data loss can vary, including power outages, sensor malfunctions, computer system failures, sensor sensitivity, routine maintenance, human error, and other reasons often experienced in the field [148, 154]. Depending on the cause, air pollution data can be lost over long-consecutive periods or short intervals [155]. While power outages and temporary routine maintenance can cause short intervals of missing data, sensor malfunctions, severe natural disasters, and other critical failures can cause long gaps in data collection.

Missing data can occur due to scenarios such as a node going down or the absence of a single reading. When a node goes down, it usually indicates a complete

loss of data over a period of time. Various factors, such as power outages, connectivity disruptions, or equipment failure, can cause these absences. As a result, all data that should have been captured during this period is lost. This absence can be important, especially if the node is critical in collecting important or unique data. In contrast, a single reading missed refers to occasional irregularities when certain data points are missing, even though most of the data was collected correctly. Reasons may include temporary interruptions, short-lived sensor problems, or environmental conditions preventing data collection. In these cases, lost data is usually an isolated incident and often does not significantly impact the integrity of the data set as a whole. The impact depends largely on how frequently and uniformly these events occur.

When dealing with air pollution measurements and assessing potential exposures and health impacts, encountering missing data can present significant challenges. The presence of missing data can potentially affect the interpretations and conclusions of studies. According to Rubin, incomplete data can be classified based on their generating mechanisms into three categories: missing completely at random (MCAR), missing at random (MAR), and missing not at random (MNAR) [156].

MCAR refers to data that is missing purely due to random events [157]. However, the assumption that missing values in MCAR are a random sample of observed values is restrictive [158]. In MAR, the probability of missingness may depend on observed data values but not on the missing values themselves. In MAR conditions, it is possible to estimate the missing values using other observed predictor variables [68, 157]. On the other hand, MNAR occurs when the probability of an observation being missing depends on unobserved values, including those of the missing observations [156, 157, 159]. MNAR represents nonignorable missingness and can lead to biased parameter estimates [160]. It is important to note that missing data in practice often do not strictly adhere to either MCAR or MNAR [159]. In the case of air quality data, the missingness can be considered at least MAR. While the reasons for missing air quality data may be unknown (i.e., MCAR), most missing values can be attributed to explainable circumstances such as routine maintenance, sensor malfunction, power outages, etc. [157, 161]. Therefore, in this thesis, MAR conditions are assumed for the air quality data.

3.2 Approaches for Dealing with Missing Data

Two common ways to handle missing data are deleting the missing parts and imputing (substituting) the missing values [162]. The deletion method can be further

specified as pairwise deletion and listwise deletion. The pairwise deletion method discards the specific missing values, whereas the listwise approach removes the entire record, even if there is only one missing value. Excluding incomplete observations with a high level of missing values may reduce the precision of the analysis [158]. In addition, since pollutant measurement generates time-series data, the deletion method may corrupt the data structure, resulting in the loss of important information.

In contrast to the deletion method, the imputation method reconstructs the missing values using existing information [163]. Reconstruction techniques inspired by machine learning have been developed to recover corrupted data. One of these techniques is the denoising autoencoder (DAE) [164]. Many fields have adopted the standard DAE and its variants, such as image denoising [165, 166, 167, 168], medical signal processing [169, 170], fault diagnosis [171, 172], etc. Some works also utilised DAE to impute missing data. Gondara *et al.* [173] attempted to address the challenge of multiple imputations by utilising an overcomplete representation of DAEs. Initial training for the proposed method does not require exhaustive observations, making it applicable to real-world scenarios. Abiri *et al.* [174] showed that DAE could recover various missing data for many datasets, demonstrating the method's robustness. Abiri *et al.* provided evidence that the proposed stacked DAE outperformed other established methods, such as K-nearest neighbour (KNN), multiple imputations by chained equations (MICE), random forest and mean imputations. Jiang *et al.* [175] utilised DAE to approximate the missing traffic flow data and compared three different architectures composing the DAE, namely standard DAE, convolutional neural network (CNN), and bi-directional long short-term memory (Bi-LSTM). Jiang *et al.* evaluated the proposed model's test sets with a general missing rate of 30%. Moreover, splitting traffic data into weekdays and weekends significantly improved the model performances. Jiang et al. noted distinct error patterns between weekdays and weekends, with the latter showing significantly higher errors. They divided the dataset into weekday and weekend categories for separate training, testing, and error calculation. This segmentation strategy proved effective for training and testing the model's predictive accuracy. Such an approach can be practically applied in traffic data prediction to achieve more precise imputation results.

Imputation techniques for missing air quality data have faced numerous challenges and have seen significant advancements in recent years. First, missing data is a recurring issue in environmental research. Although several methods have been proposed for dealing with missing data in various fields, additional research is re-

quired to predict missing data for air quality [148]. Many works were mainly focused on clinical, energy, traffic, images, etc. Second, most related studies concentrated on a small number of missing data. Ma *et al.* stated that the previous works apply to short-interval missing imputations or consecutive missing values with a missingness rate of less than 30%. This concern was also raised by Alamoodi *et al.* [176]. Only a few works investigated missing data at large percentages (i.e., more than 80%), either using deletion or imputation. Third, multiple imputation methods can improve imputation performance [149]. This chapter considers that implementing multiple imputations for air quality data is a deserving attempt. Fourth, many studies demonstrated the robustness of denoising autoencoder in recovering noisy data. However, few studies implemented the denoising autoencoder for missing air quality data imputation [177]. Finally, even though air pollutants strongly relate to spatiotemporal characteristics, these factors are rarely included in predicting the missing values of air pollution data. The air quality data collected from air monitoring stations can hold intensely stochastic spatiotemporal correlations among them [178].

3.3 Missing Data Imputation in Air Quality Research

Several works address the topic of missing data in air quality research. One of the studies conducted by Chen *et al.* introduced a method known as "First five last three logistic regression imputation (FTLRI)" [179]. This approach combines standard logistic regression with the newly proposed "first Five & last Three" model. The model is adept at describing relationships between various attributes and identifying the most relevant data points for missing values, both temporally and by attribute. FTLRI was compared with conventional imputation techniques (mean imputation, median imputation, k-nearest neighbour imputation, logistic regression imputation, and random forest imputation) and a new dynamic imputation method utilising neural networks to assess its effectiveness. This comparison was carried out at different levels of missing data, namely 5%, 10%, 20%, and 40%. In another study, Alsaber *et al.* explored the use of an iterative imputation technique named "missForest", which is based on the random forest approach, to address missing values in air quality data [180]. They collected air quality data from five monitoring stations in Kuwait. To improve the accuracy of their estimates, climatological variables such as air temperature, relative humidity, wind direction, and wind speed were included as control factors. The findings show that the MAR technique produces the lowest RMSE and MAE. MissForest showed the lowest calculation error among the various

calculation methods tested, indicating its suitability for air quality data analysis.

Belachsen and Broday (2022) introduced a new approach known as the weighted k-nearest neighbours multivariate imputation method (wkNNr) for predicting missing $PM_{2.5}$ values at 59 air quality monitoring stations across Israel [181]. Data intervals were randomly omitted to evaluate the method's performance, varying in length from 0.5 hours to 2 years. Unlike the standard kNN approach, which fills in missing values using the average from the k most similar non-missing observations, this novel method leverages correlations between station records to weigh the distances between observations. Additionally, including lagging and leading observations as model inputs provides insights into short-term temporal relationships.

Although several approaches have been proposed to handle missing data in various fields, there is a need for more research addressing air quality missing data prediction. Because there is a spatial and temporal correlation in the air quality data collected from various air monitoring stations, developing a deep learning model architecture that can include spatiotemporal factors as input for the model is necessary. In other words, imputing missing values in a target station can be realised by leveraging the neighbouring stations' data.

This chapter discusses how deep learning can be implemented to predict missing values by leveraging air quality spatiotemporal data. Inspired by the denoising autoencoder's ability to reconstruct corrupted images [164], this section proposes a novel imputation method for the air quality domain.

3.4 Contributions

The contributions in this chapter are listed as follows:

- Extracting air pollution features using a deep convolutional denoising autoencoder with spatiotemporal considerations. The suggested method utilises data from neighbouring stations to impute the missing data at the target station.
- Proposing a method suitable for short-period and long-interval consecutive missing imputations and simultaneously offering multiple imputations to obtain less biased results.
- Minimising data exchange by incorporating only the targeted pollutant data from neighbouring stations.

3.5 Air Quality Dataset

Machine learning, a subset of artificial intelligence (AI), involves training computers to learn from experience and make predictions based on input data [182]. In machine learning, the term *dataset* typically refers to the data used for developing models. Datasets are commonly divided into three subsets: *training*, *validation*, and *test sets*. In this thesis, the air quality dataset is categorised into two main types: the public dataset and the direct measurement dataset. The public datasets consist of air quality data from three cities: **Beijing**, **Delhi**, and **London**. On the other hand, the direct measurement dataset is obtained by collecting air parameter data using a low-cost air monitoring device. In this thesis, Chapters 3, 4, and 5 utilise the public datasets, while Chapter 6 focuses specifically on the direct measurement dataset.

3.5.1 Beijing Dataset

Beijing dataset is multi-station air quality data provided by Zhang *et al.* [183], which can be downloaded from the UCI Machine learning repository page [184]. The dataset captures Beijing air quality, collected from 12 different Guokong (state-controlled) monitoring sites in Beijing and its surroundings [183]. These 12 monitoring sites are Aotizhongxin, Changping, Dingling, Dongsi, Guanyuan, Gucheng, Huairou, Nongzhanguan, Shunyi, Tiantan, Wanliu and Wanshouxigong. The dataset comprises 12 columns (features) and 36,064 rows, representing data collected from 1 March 2013 to 28 February 2017. Each row represents hourly data, including pollutant measurements ($PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, and O_3) and meteorological data. The meteorological data are temperature (Temp), air pressure (Pres), dew point (Dewp), rain, wind direction, and wind speed (Wspd).

The dataset consists of both numerical and categorical data. One of the categorical attributes in the dataset is the wind direction, which can take on 16 different values: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, and NNW. These categorical features have been label-encoded. To determine the labels, the 360-degree circle is divided by 16 (the number of wind directions) and rounded down to the nearest integer. Consequently, the label for N is assigned as 360, NNE as 22, NE as 45, ENE as 67, and so on. Instead of labelling N as 0, the value 360 is assigned to this particular direction. An example of descriptive statistics of the Aotizhongxin monitoring station (excluding wind direction) is shown in Table 3.1

Table 3.1: Descriptive statistics of Aotizhongxin dataset.

	$PM_{2.5}$	PM_{10}	SO_2	NO_2	CO	O_3	Temp	Pres	Dewp	Rain	Wspd
count	35,063	35,063	35,063	35,063	35,063	35,063	35,064	35,064	35,064	35,064	35,064
mean	83.17	110.11	17.64	59.42	1271.04	55.36	13.58	1011.85	3.12	0.07	1.71
std	82.60	95.64	23.14	37.30	1255.37	57.47	11.40	10.40	13.69	0.91	1.20
min	3.00	2.00	0.29	2.00	100.00	0.21	-16.80	985.90	-35.30	0.00	0.00
25%	22.00	38.00	3.00	30.00	500.00	7.00	3.10	1003.30	-8.10	0.00	0.90
50%	59.00	87.00	9.00	54.00	900.00	41.00	14.50	1011.40	3.80	0.00	1.40
75%	115.00	154.50	22.00	82.00	1600.00	82.00	23.30	1020.10	15.60	0.00	2.20
max	898.00	984.00	341.00	290.00	10000.00	423.00	40.50	1042.00	28.50	72.50	11.20

3.5.2 Delhi Dataset

The dataset was compiled by Rohan Rao from the Central Pollution Control Board (CPCB) website and can be downloaded from Kaggle's collection [185]. The dataset consists of air quality data and Air Quality Index (AQI) at both hourly and daily levels for multiple stations across various cities in India. The dataset includes data from 26 cities, including Ahmedabad, Aizawl, Amaravati, Amritsar, Bengaluru, Bhopal, Brajrajnagar, Chandigarh, Chennai, Coimbatore, Delhi, and more. The recorded features in the dataset include $PM_{2.5}$, PM_{10}, NO, NO_2, NO_x, NH_3, CO, SO_2, O_3, Benzene, Toluene, Xylene, AQI and AQI bucket.

Missing values significantly impact the India air quality dataset. Hence, Chapter 3 of this thesis addresses the missing data imputation technique. Specifically, this thesis focuses on air quality data from the city of Delhi, covering the period from February 2018 to July 2020. Table 3.2 presents the descriptive statis-

Table 3.2: Descriptive statistics of Anand Vihar monitoring station.

	$PM_{2.5}$	PM_{10}	NO	NO_2	NO_x	NH_3	CO	SO_2	O_3
count	17,976	17,444	16,481	17,126	16,850	16,851	16,772	17,664	17,545
mean	131.50	281.29	88.24	87.69	112.26	45.55	2.21	13.85	34.62
std	120.90	188.86	99.23	61.82	97.88	29.32	1.50	11.87	31.09
min	0.2	1	0.1	0.2	0.05	0.1	0	0.1	0.1
25%	53.00	145.25	17.75	43.95	39.15	24.29	1.25	7.05	14.60
50%	90.125	235	54.92	76.39	85.33	38.6	1.85	10.72	23.24
75%	168.25	367.00	116.40	114.17	151.13	60.55	2.73	17.88	45.15
max	985.00	1000.00	500.00	490.55	500.00	139.20	19.70	193.60	197.95

Table 3.3: Descriptive statistics of Trafalgar Road monitoring station.

	NO_2	PM_{10}	Temp	Dwpt	RH	WD	Wspd	Press
count	15,452	15,452	15,452	15,452	15,452	15,452	15,452	15,452
mean	38.48	20.50	12.54	7.57	74.75	184.48	14.67	1014.10
std	20.57	13.83	6.70	4.86	17.88	96.67	8.45	10.97
min	1.18	-3.11	-5.00	-9.50	19.00	0.00	0.00	971.20
25%	22.47	12.79	7.68	4.20	64.00	100.00	9.40	1007.70
50%	35.26	17.87	11.70	7.60	79.00	210.00	13.00	1015.20
75%	51.21	25.08	17.20	11.20	89.00	260.00	18.40	1021.30
max	137.89	380.53	37.20	19.80	100.00	360.00	66.60	1047.40

tics of selected features measured at the Anand Vihar monitoring station.

3.5.3 London Dataset

The London datasets were collected using the Openair tool [186]. Openair is an *R* package developed by Carslaw and Ropkins to analyse air quality data. The package finds extensive usage in academia, as well as in the public and private sectors. Initially funded by the UK Natural Environment Research Council (NERC), the project also received additional funding from Defra [187].

The package offers a range of functions that enable users to access a comprehensive collection of UK air quality data, including the Automatic Urban and Rural Network (AURN) and Imperial College London's London Air Quality Network (LAQN). Additionally, the package includes data from other networks such as Air Quality Scotland, Air Quality Wales, Air Quality England, and Northern Ireland. Furthermore, a function is available to import data from locally managed air quality networks in England, primarily operated by Local Authorities. These air quality databases provide free public access to the data [188]. Table 3.3 presents the descriptive statistics of selected features measured at the Trafalgar Road monitoring station from January 2018 to January 2021.

3.6 Spatiotemporal Convolutional Autoencoder

3.6.1 Denoising Autoencoder

Figure 3.1 shows denoising autoencoder concept implemented in this work. The denoising autoencoder consists of three parts: *encoder*, *code* and *decoder*. The

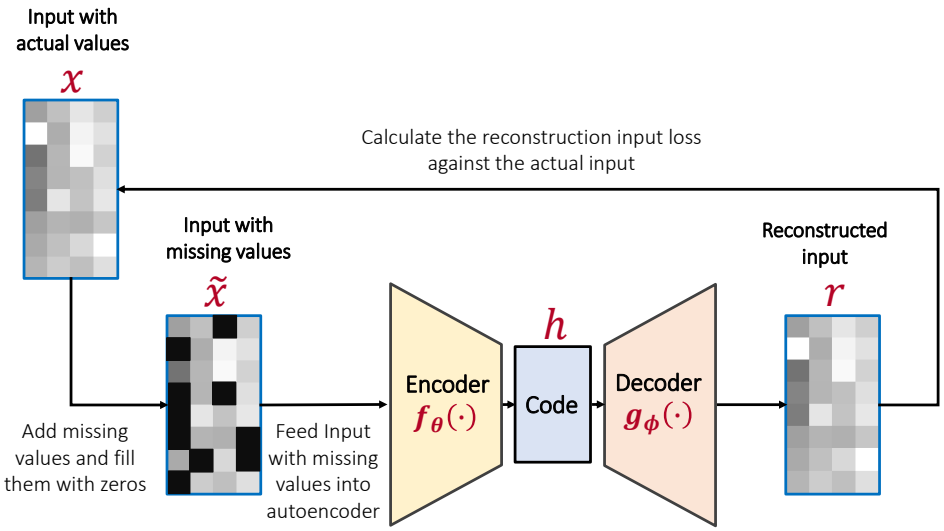


Figure 3.1: A denoising convolutional autoencoder workflow.

encoder function $f_\theta(\cdot)$ is parameterised by $\theta = \{\mathbf{W}, \mathbf{b}\}$, and decoder function $g_\phi(\cdot)$ is parameterised by $\phi = \{\mathbf{W}', \mathbf{b}'\}$, where $\mathbf{W}$, $\mathbf{W}'$, $\mathbf{b}$ and $\mathbf{b}'$ represent the weight and bias of the encoder and decoder, respectively. Then, the encoder function is written as $\mathbf{h} = f_\theta(\mathbf{x})$ and the decoder function as $\mathbf{r} = g_\phi(\mathbf{h})$, where $\mathbf{x}$ is the input, $\mathbf{h}$ is the code representation learning, and $\mathbf{r}$ is the reconstructed input. In the ideal condition, the output model can be expressed as $g_\phi(f_\theta(\mathbf{x})) = \mathbf{x}$, meaning that the model can reconstruct the noisy inputs perfectly. However, the perfect condition can not be achieved, and instead, the model tries to minimise the error between the actual input and the reconstructed input [189]. For each input set $\mathbf{x}^{(i)}$, the parameters θ and ϕ are optimised to minimise the average reconstruction error. If L is the model loss function, the optimisation can be expressed as [164]:

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} \frac{1}{n} \sum_{i=1}^{n} L(\mathbf{x^{(i)}}, g_\phi(f_\theta(\mathbf{x^{(i)}}))) \tag{3.1}$$

The loss function is typically defined as squared error, that is, $L(\mathbf{x}, \mathbf{r}) = \|\mathbf{x} - \mathbf{r}\|^2$. Instead of the original input $\mathbf{x}$, a notation of $\widetilde{\mathbf{x}}$ is mentioned as the noisy input of $\mathbf{x}$ in the case of denoising autoencoder [189]. Finally, the loss function of the denoising autoencoder is written as:

$$L(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x^{(i)}} - g_\phi(f_\theta(\widetilde{\mathbf{x}}^{(i)})))^2 \tag{3.2}$$

3.6.2 Correlation of Pollutant Data

The correlation coefficient of pollutant data under investigation is calculated by collecting the same pollutant from all monitoring stations. For instance, if the PM_{10} is decided as a target pollutant, PM_{10} values from all monitoring stations will be combined, and Pearson's correlation formula is applied to find the relation among air quality monitoring stations. Pearson's correlation captures the linear correlation of two time-series data [148]. In statistics, Pearson's correlation is utilised primarily to evaluate the linear relationship between two continuous variables. Pearson's correlation formula measures this linear relationship's strength and direction. Therefore, neighbouring stations are selected based on their degree of correlation with the target stations rather than their geographical proximity.

A temporal sequence of specific pollutant data in the target station can be written as $\boldsymbol{S}^t = [s_1^t, s_2^t, s_3^t, \ldots, s_{n-1}^t, s_n^t]$, and a temporal sequence of the same pollutant data at a neighbouring station is written as $\boldsymbol{S}^s = [s_1^s, s_2^s, s_3^s, \ldots, s_{n-1}^s, s_n^s]$. It is worth noting that this work assumes $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$ have the same time frame, ranging from sample 1 to n. Pearson's correlation coefficient of temporal data between the target station and a neighbouring station is written as follows:

$$r(\boldsymbol{S}^t, \boldsymbol{S}^s) = \frac{\sum_{i=1}^{N}((s_i^t - \mu_t)(s_i^s - \mu_s))}{\sqrt{\sum_{i=1}^{N}(s_i^t - \mu_t)^2 \sum(s_i^s - \mu_s)^2}} \tag{3.3}$$

where, $r(\boldsymbol{S}^t, \boldsymbol{S}^s)$ denotes the Pearson's correlation coefficient between temporal sequence $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$. s_i^t and s_i^s represent the i-th samples of $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$, respectively. Finally, $\mu_t = \frac{1}{n}\sum_{i=1}^{N} s_i^t$ and $\mu_s = \frac{1}{n}\sum_{i=1}^{N} s_i^s$ are the mean values of temporal sequence of $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$, respectively.

The numerator in equation 3.3 is called the covariance, a measurement about how temporal sequence $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$ vary together from their mean value. In the denominator, the equation covers the variance of $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$. Correlation is a normalised version of covariance, scaled between -1 to 1 [190]. When $r = 1$, the temporal sequence of $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$ are completely positively correlated. When $r = -1$, $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$ are completely negatively correlated. Finally, when $r = 0$, the linear correlation between $\boldsymbol{S}^t$ and $\boldsymbol{S}^s$ is not obvious [191].

3.6.3 Proposed Deep Learning Model

This study proposes a convolutional autoencoder model to learn the missing patterns from the corrupted input sets and the provided actual sets. The proposed model architecture is shown in Figure 3.2. The main idea of this approach is to

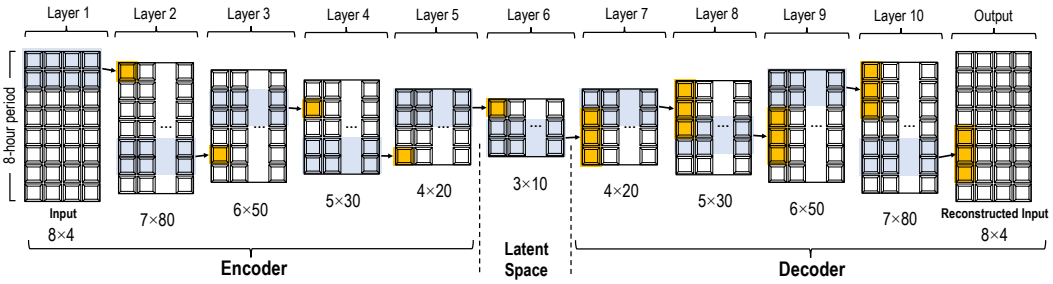


Figure 3.2: Proposed deep convolutional autoencoder model.

apply the denoising concept using a deep autoencoder. The model will learn how to reconstruct the "noised" input. Analogous to a noised image, the commonly implemented autoencoder model will learn how to rebuild that image. In this approach, the "noise" is the missing data itself. The "image" is created based on pollutant data of the target station and the neighbouring stations. No meteorological data are involved in this reconstruction technique so that the reconstruction process can be done effectively.

The autoencoder model accepts the collection of input sets as 8×4 matrices. The individual input comprises four columns of pollutant data, a group of hourly targeted pollutant concentrations from four monitoring stations, and eight rows that indicate 8-hour of observed data. The original input sets are purposely corrupted by deleting the actual values and filling them with zeros to train the model. The input columns represent spatial behaviour, and the rows capture temporal characteristics of air pollution features.

The autoencoder contains an encoder, a code and a decoder. All parts are based on one-dimensional convolution layers in this work. While the encoder consists of convolution layers, the decoder is constructed using *transposed* convolution layers. The code layer (latent space) has the smallest dimension and can be considered the encoder's last layer. As the model receives eight temporal values as the feature's length, a small kernel should be applied to extract the information from input features. In this work, the kernel and stride sizes are set to two and one, respectively. Also, no padding is implemented in each layer. These selections are applied to all layers, both in the encoder and decoder sections.

Figure 3.2 illustrates the layers' height and width size changes. The width of the next layers is governed by the size of the filter used in the previous layer. After conducting various experiments, the number of filters is determined and reported in Table 3.4. The encoder comprises different output filters, from 80 in the first layer to 10 in the fifth. From the latent space, the number of filters is expanded from 20

Table 3.4: Layer properties of proposed convolutional autoencoder model.

No.	Type	Filter	Kernel	Activation	Output
0	Input Layer	-	-	-	(8,4)
1	1D Convolution	80	2	ReLU	(7,80)
2	1D Convolution	50	2	ReLU	(6,70)
3	1D Convolution	30	2	ReLU	(5,50)
4	1D Convolution	20	2	ReLU	(4,30)
5	1D Convolution	10	2	ReLU	(3,10)
6	1D Transposed conv.	20	2	ReLU	(4,30)
7	1D Transposed conv.	30	2	ReLU	(5,50)
8	1D Transposed conv.	50	2	ReLU	(6,70)
9	1D Transposed conv.	80	2	ReLU	(7,80)
10	1D Transposed conv.	4	2	ReLU	(8,4)

in the sixth layer to 80 in the ninth layer. Finally, the final layer size is conditioned equally to reconstructed inputs (i.e., 8×4 matrices).

3.7 Processing of Spatiotemporal Data

3.7.1 Air Quality Monitoring Stations

This chapter uses air quality datasets from three cities: London, Delhi, and Beijing. Each city has ten monitoring stations that study two pollutants per station. This selection of ten stations per city is considered sufficient for implementing and evaluating the performance of the proposed algorithm. Additionally, the choice of pollutants in each city is varied to demonstrate the applicability of the proposed method to different pollutants. Several considerations are taken into account during the station selection process. Two major concerns are the availability of pollution data and the measurement period for all stations. Only stations with at least three years of data from the same period are included in the analysis. Furthermore, since the proposed method relies on the correlation coefficient between stations, stations with varying degrees of correlation are incorporated to ensure the robustness of the proposed method.

In the London city dataset, the selected pollutants are nitrogen dioxide (NO_2) and particulate matter with a diameter of less than 10 μm (PM_{10}). Data from ten monitoring stations across London are used, covering the period from January 2018

Table 3.5: Dataset and stations involved in the experiment.

London		Delhi		Beijing	
Code	**Station**	**Code**	**Station**	**Code**	**Station**
CT3	Aldgate School	DL02	Anand Vihar	AOT	Aotizhongxin
GN5	Trafalgar Road	DL03	Ashok Vihar	CHA	Changping
GR8	Woolwich Flyover	DL04	Aya Nagar	DIN	Dingling
IS2	Holloway Road	DL07	Mathura Rd.	DON	Dongsi
IS6	Arsenal	DL08	DTU Delhi	GUA	Guanyuan
LB5	Bondway Intchg.	DL10	Dwarka-Sec.8	GUC	Gucheng
LW4	Loampit Vale	DL12	IGI Airport	HUA	Huairou
SK6	Elephant & Castle	DL13	IHBAS, Delhi	NON	Nongzhanguan
TH001	Millwall Park	DL14	ITO, Delhi	SHU	Shunyi
TH002	Victoria Park	DL15	Jahangirpuri	TIA	Tiantan

to January 2021. Ten monitoring stations across Delhi are considered for the Delhi dataset, and the data spans from February 2018 to July 2020. The chosen pollutants for the Delhi dataset are hourly measurements of nitrogen dioxide (NO_2) and particulate matter with a diameter of less than 2.5 μm ($PM_{2.5}$). In the Beijing dataset, the hourly pollutant data covers the period from January 2013 to February 2017. The focus is on carbon monoxide (CO) and ozone (O_3) measurements. The selected monitoring stations for the Beijing dataset are Aotizhongxin, Changping, Dingling, Dongsi, Guanyuan, Gucheng, Huairou, Nongzhanguan, Shunyi, and Tiantan. Table 3.5 summarises the air quality monitoring stations used in this study.

3.7.2 Data Preprocessing for Spatial Correlation

The visual concept of leveraging neighbouring data is illustrated in Figure 3.3. In the event that S^3 is unable to collect pollutant data from the environment, the neighbouring stations S^2, S^5 and S^6 send their data to S^3. The participating neighbouring stations (S^2, S^5 and S^6) eligible to send data are chosen based on their coefficient correlations with the target station. A deep autoencoder model at S^3 that covers the spatiotemporal behaviour of pollutant data is applied. Based on the collected spatiotemporal data at the target and neighbouring stations, the missing data at S^3 can be estimated.

This work's final design covers spatial correlation involving three neighbour-

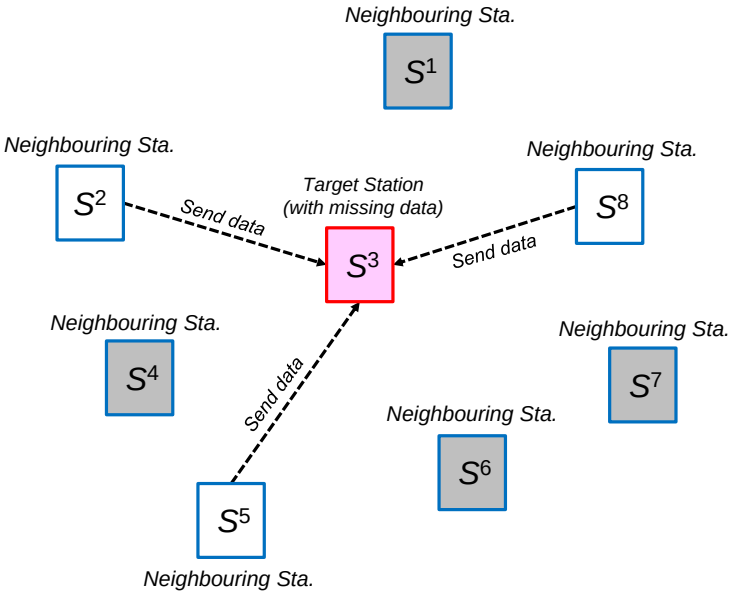


Figure 3.3: Target station leverages measurement data from neighbouring stations to impute the missing data.

ing stations (discussed further in Section 3.8.2) with the closest relationship to the station under investigation.

Let $\boldsymbol{S}^t = \begin{bmatrix} s^t_{1,1} & s^t_{1,2} & \dots & s^t_{1,n} \\ s^t_{2,1} & s^t_{2,2} & \dots & s^t_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ s^t_{m,1} & s^t_{m,2} & \dots & s^t_{m,n} \end{bmatrix} = (s^t_{i,j}) \in \mathbb{R}^{m \times n}$ be a matrix containing

m rows of measurement data and n different pollutants at target station t, where t ranges from 1 to 10 (this work uses ten different stations). Besides measurement data at target station $\boldsymbol{S}^t$, there is a collection of pollutant data from all stations of $\boldsymbol{S}^1$, $\boldsymbol{S}^2$, $\boldsymbol{S}^3$,..., $\boldsymbol{S}^{10}$. In this case, each row in matrix $\boldsymbol{S}^t$ is hourly measurement data.

A matrix $\boldsymbol{J} = \begin{bmatrix} s^1_{1,p} & s^2_{1,p} & \dots & s^{10}_{1,p} \\ s^t_{2,p} & s^2_{2,p} & \dots & s^{10}_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ s^1_{m,p} & s^2_{m,p} & \dots & s^{10}_{m,p} \end{bmatrix}$ as a collection of the same pollutant p taken

from all stations can be defined, where p is a single value chosen from 1 to n. p represents the selected column in $\boldsymbol{S}^t$. In this scenario, all monitoring station data

in the same city have the same column header. Finally, the pairwise correlation of columns in $\boldsymbol{J}$ using equation 3.3 can be calculated, excluding null/missing values.

After collecting neighbouring data, Pearson's correlation between stations can be measured using equation (3.3). The yielded correlation coefficients for each station are then sorted from the highest to the smallest. Based on this result, three neighbouring stations are selected. Finally, along with the target station, four columns of the input set reflecting spatial correlation are obtained.

3.7.3 Data Preprocessing for Temporal Correlation

Air quality temporal correlation indicates dependency among pollutants at different times [192]. The temporal characteristic in this work is determined by implementing the autocorrelation coefficient of the contaminant under investigation. Evaluating the paired series of targeted pollutants and its shifted self reflects the concept of autocorrelation. In other words, autocorrelation computes the relation between the same time series at current and lagged times, i.e., the historical air pollutant data. Assume that the temporal series of pollutant data at the target station is given as $\boldsymbol{S}^t = [s_1^t, s_2^t, s_3^t, \ldots, s_{n-1}^t, s_n^t]$, then the equation (3.3) can be restructured to express the lag-k relations, described as follows:

$$r_k = \frac{\sum_{i=k+1}^{n}((s_i^t - \mu_t)(s_{i-k}^t - \mu_t))}{\sum_{i=1}^{n}(s_i^t - \mu_t)^2} \tag{3.4}$$

where, r_k denotes the autocorrelation function, k is a lagged event, s_i^t and s_{i-k}^t represent the i-th and lag-k samples of $\boldsymbol{S}^t$, and $\mu t = \frac{1}{n}\sum_{i=1}^{N} s_i^t$ denote the mean values of time series $\boldsymbol{S}^t$. Implementing this formula, the seven lagged data are chosen (will be discussed further in Section 3.8.1).

3.7.4 Missing Period Distribution

As this study develops a model for short-term and long-interval consecutive missing imputations, the nature of missing patterns is essential to be investigated. The duration of all missing patterns in the original dataset is explored, and the findings are reported in Figure 3.4. The figure visualises the distribution of missing data durations using continuous probability density curves. The graph delivers information on how the missing data durations are distributed. The results show that most missing durations in the London dataset are less than 400 hours, whereas the Delhi and Beijing datasets exhibit shorter missing durations, approximately less than 200 hours. The peaks of the probability density curve in all datasets commonly occur

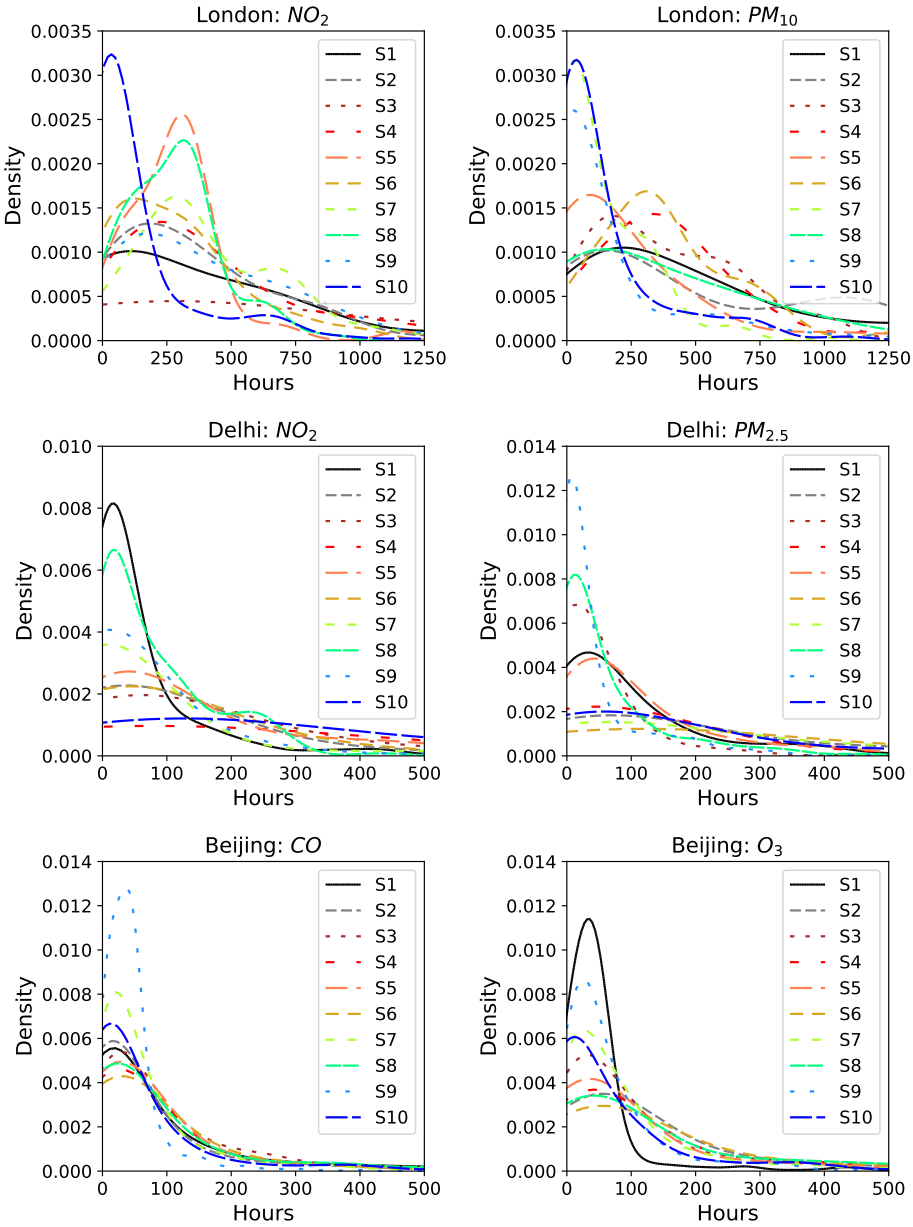


Figure 3.4: Probability density function of missing data in all stations.

within 100 hours. Thus, all datasets are occupied mainly by short-interval missing patterns with less than one-week periods.

3.7.5 Missing Data Generation and Perturbation Procedure

Another pre-processing step in this chapter is handling the initial missing data in the original datasets. Missing values can occur every time in the form of discontinuous or consecutive missing patterns. The input sets should be well-prepared to cover any possibilities during model training. As the proposed deep convolutional autoencoder in this study is trained in a supervised manner, the input-target set pairs should be engineered. Deleting missing data from the original dataset is a straightforward procedure for many cases and may give a satisfying model prediction. However, this action may break the data structure for cases involving time-series sets, and valuable information contained in the dataset may be lost. One solution offered in this study is to carefully pick the series of data with a minimum period of one week (168 hours) to minimise the defect of the original data structure. The minimum period of one week is obtained after investigating the missing pattern of the original dataset (as will be reported in Section 3.7.4). This study involves multiple air quality monitoring stations, and other station periods comply with the target station's period when selecting the input sets for model training.

This work's strategy focuses on acquiring continuous segments of training and test data. This work utilises an unsupervised learning approach, which requires the target to be known beforehand. Therefore, it is crucial to select uninterrupted data segments, both for target and neighbouring stations. Subsequently, certain data is intentionally removed to simulate missing information. This deliberate removal is critical in aligning with the unsupervised learning methodology.

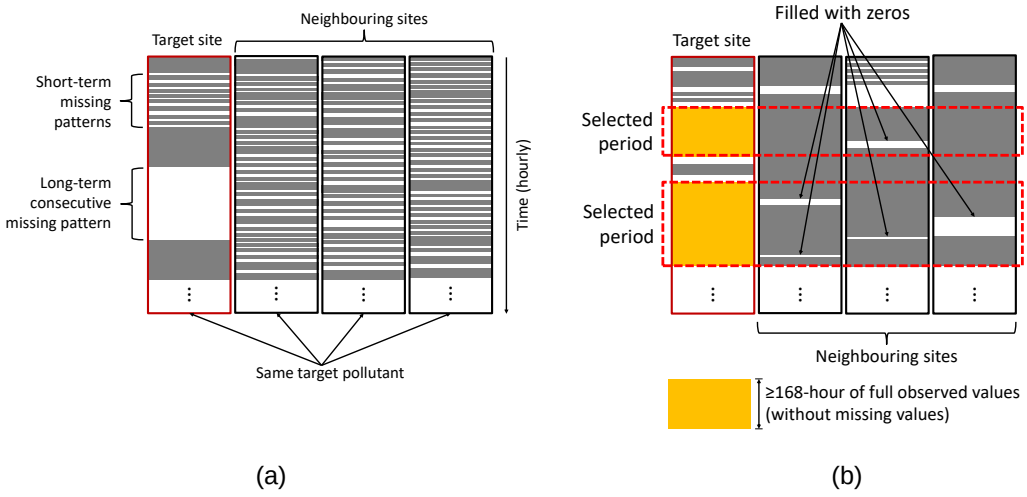


Figure 3.5: (a) Implemented method to handle the initial missing data in the original datasets, and (b) illustration of perturbation patterns applied to the dataset.

Figure 3.5(a) depicts a possibility of missing patterns in the original dataset. The shadowed areas indicate measurements without missing values, whereas the white strips indicate the existing missing values. Among periods without missing values, a minimum of 168 hours of observations are carefully selected from the station's columns under investigation (target station). The same selection periods are then expanded to the entire columns to maintain the consistency of the time frame between monitoring stations. After completing these steps, the target station's column has no missing data. However, unlike the target station's column, there is a possibility that missing values still exist in the neighbouring stations' columns. The remaining data at neighbouring station columns with missing values are filled with zeros to solve this issue (please refer to Figure 3.5(b)). After completing the aforementioned steps, the collections of input sets contain no unknown values, and the actual targets for all input sets can be provided.

In the following steps, a perturbation procedure can be performed. The perturbation procedure is carried out to cover as many missing patterns as possible in the dataset, either in the form of short-interval or long-interval consecutive missing patterns. The variations of missing patterns can be achieved by intentionally removing some values in the input sets, and all deleted values are filled with zeros. Two perturbation pattern procedures can be explained as follows:

1. *Short missing interval.* Different missing levels are applied to the input sets for the short-interval perturbation procedure. Following the work conducted by Hadeed *et al.* [68], four missing rates are set for the target station. These variations are 20%, 40%, 60% and 80% of missing rates. While the missing rate varies for the target station, a fixed missing rate of 20% is applied to the neighbouring stations. The missing rate of 20% is considered an error probability for the neighbouring stations [193]. Due to the initial zero imputation illustrated in Figure 3.5(a), the neighbouring stations will have more than a 20% missing rate after the perturbation procedure.

2. *Long-consecutive missing interval.* A maximum of 500 hours of consecutive values is removed from some parts of the correct dataset for the long-interval perturbation procedure. The successive missing periods vary between 100 and 500 hours. This procedure is implemented only in the target station, and the neighbouring stations follow the short interval procedure (i.e., 20% of missing rate is applied to neighbouring stations).

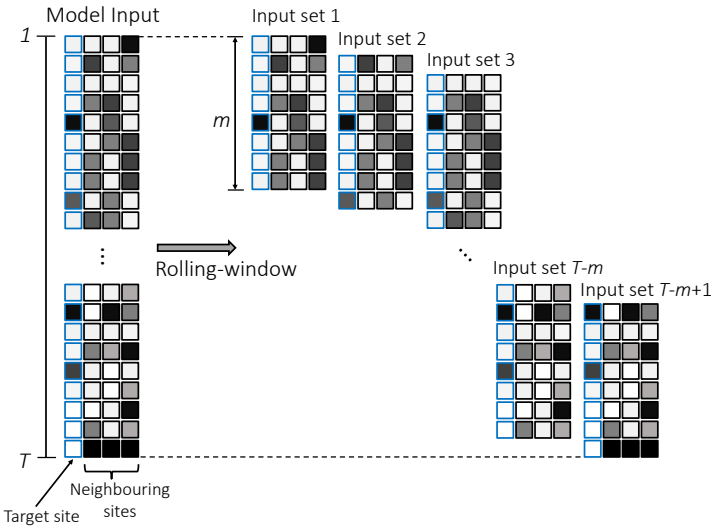


Figure 3.6: Extracting input sets from the preprocessed dataset.

3.7.6 Pre-training Model Input Construction

The normalisation of the input set takes place after completing the perturbation procedure. Following the normalisation step, the model input construction is performed. The step yields input sets ready to be fed to the model during training and testing. As illustrated in Figure 3.6, the dataset contains air pollutant data, sampled between $t = 1, \ldots, T$, with the rolling window size of m. The input sets for the model are obtained by shifting the pre-processed dataset. This study takes 8 hours of data and shifts the features by one hour to get the next input set. This process is similar to the rolling window scheme. In our case, the increment between successive rolling windows is one period.

The proposed model acts as a denoising tool, trying to recover the noisy inputs (input with missing values). Given the noisy inputs, the autoencoder model will reconstruct these inputs to achieve the given target, that is, the complete dataset itself. Constructed by convolutional layers, our proposed model can be called a denoising convolutional autoencoder.

3.7.7 Post-training Model Outputs

To evaluate the effectiveness of the proposed method, the test sets are fed to the model after model training. The model accepts input and produces output with the same size (8×4). Since min-max normalisation is employed to scale the input and target sets for effective learning, the model's predictions must be reverted to their original values. After reverting the scaled outputs to their original values,

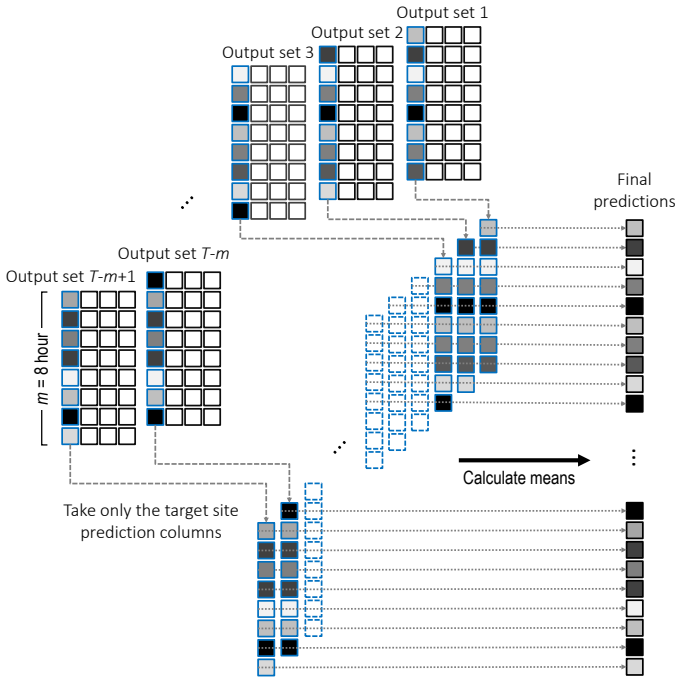


Figure 3.7: Approach to obtaining the final prediction.

each hour's single prediction must be determined. As illustrated in Figure 3.7, the autoencoder produces overlapping outputs for a certain prediction period. An approach conducted in this study is aggregating the values of overlapped output sets to give a single-point estimation. As the targeted results are located in the model outputs' first columns (target station), the means of the first columns of the output sets is calculated, as illustrated in Figure 3.7.

The post-training output interpretations are systematically presented in Algorithm 3.1. As shown in Algorithm 3.1, the test sets $\mathbf{X}$ are fed to the model (row 3), resulting in a 3-dimension prediction set $\mathbf{Y}$ with a size of $(n, 8, 4)$, where n is the number of test sets fed to the model. The prediction set $\mathbf{Y}$ must be scaled back to their original values, resulting in a matrix $\mathbf{YY}$ (row 4). Only predictions in the target station's columns are selected to minimise the computing process. The target station prediction values are obtained by extracting the first column of each output set (row 5). This process results in a 2D matrix $\mathbf{YY}$ with a size of $(n, 8)$. Next, the following row is right-shifted one step from the previous row (rows 10:12), provided that there is a sparse matrix $\mathbf{A}$ appropriate to handle this rolling scheme (rows 6:8). The sums of each column are computed to get a single row matrix $\mathbf{S}$ (row: 13). As the matrix $\mathbf{S}$ is obtained from different layers of overlapped values, the divisors of each element in $\mathbf{S}$ are varied (rows 16:23). For the first seven elements of $\mathbf{S}$, divisors

Algorithm 3.1 Interpretation of post-training model outputs.

Input: Given test sets $\mathbf{X}$
Output: Series of imputed missing values $\mathbf{S}$
$\mathbf{Y} \leftarrow \text{model\_predict}(\mathbf{X})$
$\mathbf{YY} \leftarrow \text{invers\_transform}(\mathbf{Y})$
$\mathbf{YY} \leftarrow \mathbf{YY}[:, :, 1]$
$r_y \leftarrow \text{row\_size}(\mathbf{YY})$
$\mathbf{Z} \leftarrow \text{zeros}(r_y, r_y - 1)$
$\mathbf{A} \leftarrow \text{concatenate } ((\mathbf{YY}, \mathbf{Z}), axis = 1)$
$r_a \leftarrow \text{row\_size}(\mathbf{A})$
for $i = 0, r_a$ **do**
    $\mathbf{A}[i, :] \leftarrow \text{roll}(\mathbf{A}[i, :], i)$
end for
$\mathbf{S} \leftarrow \text{sum}(\mathbf{A}, axis = 0)$
$l_s \leftarrow \text{column\_size}(\mathbf{S})$
$l_p \leftarrow 8$
for $i = 0, l_p - 1$ **do**
    $j \leftarrow i + 1$
    $\mathbf{S}[i] \leftarrow \mathbf{S}[i]/j$
    $\mathbf{S}[-j] \leftarrow \mathbf{S}[-j]/j$
end for
for $i = l_p - 1, (l_s - l_p + 1)$ **do**
    $\mathbf{S}[i] \leftarrow \mathbf{S}[i]/l_p$
end for

are increased from 1 to 7, whereas for the last seven elements, divisors are decreased from 7 to 1. All values between the seven first and seven last elements of $\mathbf{S}$ are equally divided by 8.

3.8 Spatiotemporal Evaluation

3.8.1 Temporal Evaluation

The temporal relationship is evaluated to determine the length of the input series to be fed to the model. Temporal behaviours for each monitoring station are assessed based on obtained Pearson's autocorrelation coefficient between the series. As previously explained, the correlation coefficient is computed between the actual time series data and their shifted lag-k hour data. This study sets the variable k between 1 and 11. For instance, $k = 1$ means that the actual time-series data are shifted 1 hour backwards. A maximum of $k = 11$ is considered enough to capture the temporal behaviour. Too long historical data leads to less contribution for model

training as the temporal correlation of historical data degrades over time.

Figure 3.8 reports the calculated autocorrelation coefficients for each city and pollutant. For $k \leq 11$ hours, the autocorrelation coefficients vary between 1 (i.e., at $k = 0$) and 0. For the London dataset, all monitoring stations measuring NO_2 pollutants have similar autocorrelation coefficient slopes, ranging from 1 to about 0.2. Monitoring station S^1 has the flattest slope, indicating that S^1 has the strongest relationship among the lagged hours of NO_2 pollutants compared to other stations. For PM_{10} in the same air quality dataset, the stations S^2, S^3 and S^6

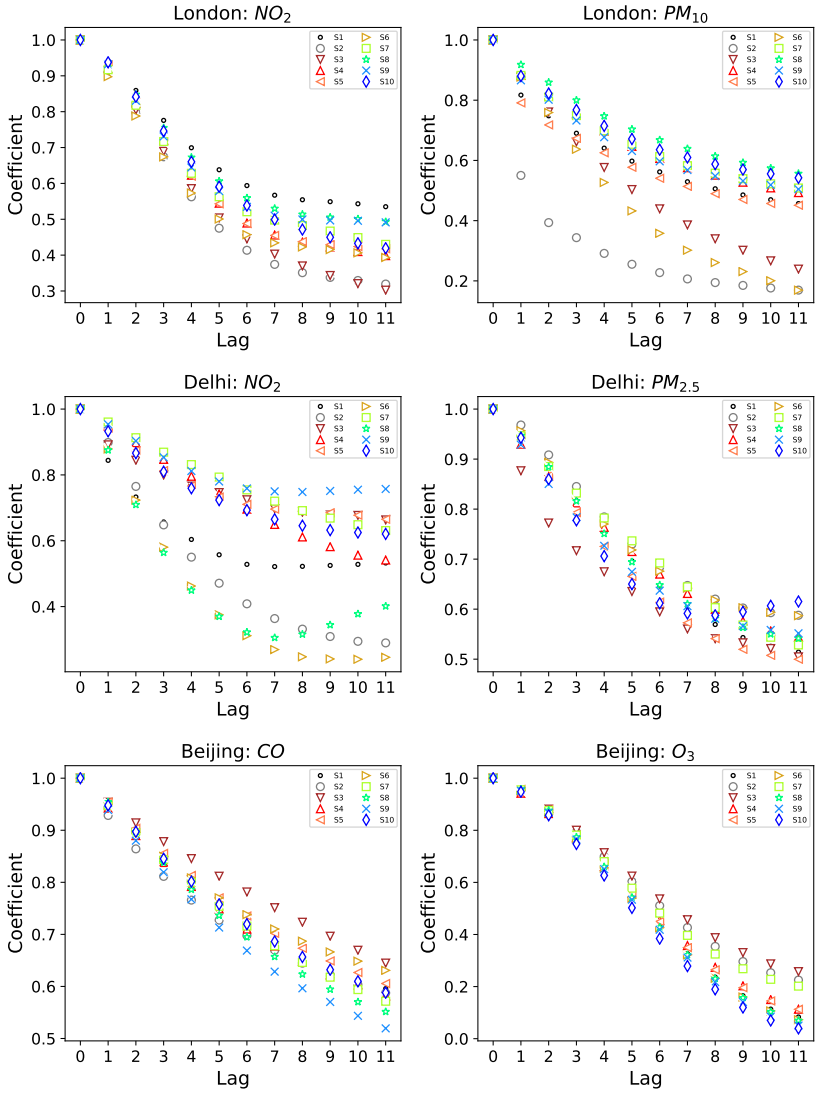


Figure 3.8: Temporal characteristics of air quality datasets based on autocorrelation coefficients.

autocorrelation coefficients plunge to about 0.2 in the first six lagged hours, whereas other stations' coefficients remained above 0.5. Among these, S^2 has the weakest temporal dependency.

For the Delhi air quality dataset, the $PM_{2.5}$ autocorrelation coefficient slopes are relatively flattered for the same pollutant, ending between 0.55 and 0.65 at $k = 11$. Autocorrelation coefficients for NO_2 between monitoring stations in Delhi degrade more diversely, especially from $k = 3$ to $k = 11$. Station S^1 and S^8 have exceptional slopes, which the coefficients tend to increase after $k = 7$. Less varied autocorrelation coefficient slopes are shown in Beijing dataset for both CO and O_3 data. However, O_3 pollutant coefficients decrease more rapidly than CO coefficients.

The number of pollutants (i.e., the length of the input set) for the autoencoder model is based on the obtained coefficients shown in Figure 3.8. A simple model is introduced as a base model to determine these properties carefully. The base model is used to evaluate the temporal and spatial dependencies. Temporal evaluation determines the number of input set rows, whereas spatial evaluation defines the number of input columns. Some other model architectures are derived from the base model until the final design is decided. The final design is shown in Figure 3.2 with properties presented in Table 3.4.

Figure 3.9 presents the base model used to determine the size of temporal and spatial properties. Compared to the final design, the base model uses a convolutional autoencoder design with shallower hidden layers. In this study, the base architecture is written as $L^1 40 - L^2 30 - L^3 20 - L^4 30 - L^5 40 - L^6 x$, where x will vary depending on the intended number of output columns. For the temporal evaluation, x equals 4, which combines the target station with three neighbouring stations. $L^1 40$ means

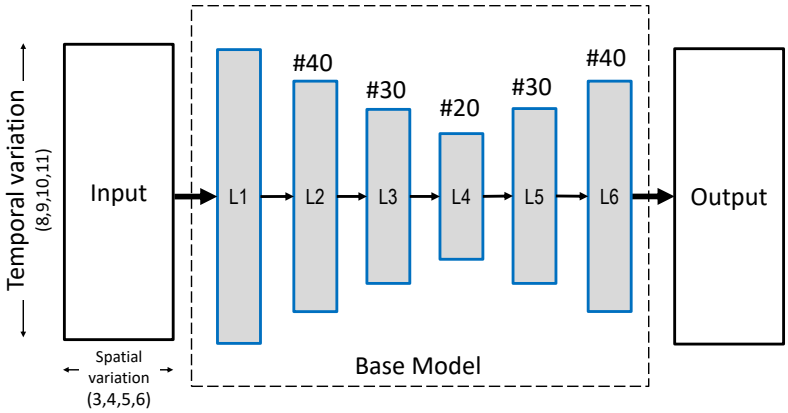


Figure 3.9: The proposed base model for temporal and spatial characteristic evaluations.

the first layer has 40 output filters and yields 40 columns placed in the second layer (please refer to Figure 3.9). The sixth layer (i.e., L^6x, where $x = 4$) has four output filters and forms $n \times 4$ output sets, where n depends on the input length, kernel and filter size. This study uses the kernel size equals 2, while the stride equals 1. Furthermore, no padding is applied to all layers.

This study uses 60% of the total observation as training sets, applied for each station and pollutant. Besides training data, test data are determined based on unbroken time-series segments with a minimum of 400 hours of consecutive observed values. The target station is corrupted with a missing rate of 40%, whereas the neighbouring data are lightly corrupted with a missing rate of 20%. A 5-fold cross-validation in the dataset is performed to obtain less biased results. An example of temporal data evaluation targeting NO_2 as the target pollutant in the city of London is presented in Table 3.6.

Indicated as emphasised texts in Table 3.6, the minimum RMSE values are dominantly obtained when $k = 7$. The lag-7 hours means the model accepts eight temporal data as the number of rows (input length). However, selecting $k = 7$ does not significantly improve the RMSE values. For example, the obtained RMSE at S^5 equals 6.28 $\mu g/m^3$ at $k = 7$, which slightly improves the RMSE to only about 4% from the highest RMSE (i.e., when $k = 10$). Since the temporal correlations of measurement values are weaker over time, the performance of a model does not generally improve as the number of temporal data increases. Weak temporal correlations contribute fewer essential features for the autoencoder model. In summary, a window size of 8-time steps is selected as the length of the input sets.

Table 3.6: Average of RMSE($\mu g/m^3$) and standard deviation values after 5-fold cross-validation targeting NO_2 for the London dataset.

	Test Period		**Lag-k**			
	Start	**End**	**7**	**8**	**9**	**10**
S^1	2020-12-02	2020-12-31	**8.56(0.35)**	8.93(0.43)	8.71(0.27)	8.87(0.28)
S^2	2021-01-12	2021-01-31	14.98(0.25)	15.07(0.34)	**14.25(0.43)**	14.26(0.64)
S^3	2020-12-07	2021-01-26	**15.86(0.6)**	16.92(0.88)	16.26(0.39)	16.43(0.88)
S^4	2021-01-12	2021-01-31	12.96(0.57)	**12.86(0.37)**	13.17(0.4)	13.02(0.35)
S^5	2021-01-12	2021-01-30	**6.28(0.19)**	6.42(0.39)	6.31(0.34)	6.56(0.11)
S^6	2020-12-07	2021-01-06	9.31(0.15)	9.58(0.17)	**9.19(0.24)**	9.39(0.19)
S^7	2021-01-11	2021-01-31	16.29(0.2)	16.32(0.72)	16.23(0.25)	**16.11(0.31)**
S^8	2021-01-08	2021-01-25	8.81(0.46)	**8.62(0.29)**	9.02(0.51)	8.85(0.24)
S^9	2020-12-30	2021-01-31	**7.43(0.19)**	7.70(0.17)	7.74(0.19)	7.45(0.23)
S^{10}	2020-10-29	2021-01-31	**7.38(0.14)**	7.54(0.23)	7.39(0.07)	7.4(0.14)

3.8.2 Spatial Evaluation

The correlation coefficient between the two stations is calculated for each target pollutant. For example, Table 3.7 and Table 3.8 report the obtained correlation coefficients for NO_2 and PM_{10} for London air quality data. The same procedure is also implemented for Delhi and Beijing datasets, and the results are reported from Table A.1 to Table A.4 of Appendix A.

The correlation coefficients reflect a linear relationship between station pairs and can be calculated using Equation (3.3). As shown in Table 3.7, the correlation coefficients among monitoring stations measuring NO_2 fall between 0.49 and 1.00. The paired stations such as $S^1 - S^9$, $S^4 - S^6$, $S^8 - S^9$, and $S^5 - S^{10}$ have strong correlations for pollutant NO_2. In contrast, the paired stations $S^3 - S^5$, $S^3 - S^7$, and $S^3 - S^{10}$ have weaker correlations. The correlation coefficients between the paired stations measuring PM_{10} as presented in Table 3.8 are more diverse, ranging between 0.27 and 1.00. Each station exhibits both strong and weak correlations with others. Station S^6 has more weak correlations with other stations. Each station exhibits both strong and weak correlations with others. Station S^6 appears to have weaker correlations with other stations, ranging from 0.27 to 0.52. This condition indicates that station S^6 has the weakest strength of a linear association with other stations. In other words, this station's air pollution data series trend differs from that of most London stations.

This thesis determines spatial correlation by the number of participating neighbouring stations allowed to send data to the target station with missing data. Varying the number of neighbouring stations affects the model input width (i.e., the number of columns). As shown in Figure 3.9, the width of the input set is evaluated

Table 3.7: Coefficient of correlation targeting NO_2 in London data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.77	1.00								
S^3	0.63	0.83	1.00							
S^4	0.73	0.78	0.82	1.00						
S^5	0.81	0.73	0.57	0.79	1.00					
S^6	0.78	0.79	0.79	0.84	0.71	1.00				
S^7	0.72	0.62	0.49	0.60	0.61	0.70	1.00			
S^8	0.84	0.67	0.50	0.68	0.84	0.74	0.75	1.00		
S^9	0.85	0.76	0.54	0.66	0.84	0.71	0.74	0.88	1.00	
S^{10}	0.80	0.68	0.49	0.69	0.88	0.66	0.66	0.85	0.85	1.00

Table 3.8: Coefficient of correlation targeting PM_{10} in London data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.47	1.00								
S^3	0.57	0.46	1.00							
S^4	0.76	0.53	0.69	1.00						
S^5	0.70	0.47	0.55	0.82	1.00					
S^6	0.35	0.27	0.52	0.45	0.31	1.00				
S^7	0.75	0.48	0.55	0.76	0.69	0.33	1.00			
S^8	0.81	0.50	0.60	0.86	0.78	0.34	0.81	1.00		
S^9	0.77	0.53	0.58	0.82	0.75	0.36	0.80	0.85	1.00	
S^{10}	0.77	0.64	0.57	0.83	0.76	0.35	0.77	0.85	0.83	1.00

from 3 to 6 monitoring stations. Based on this convention, the number of involved neighbouring stations is calculated. A 5-fold cross-validation is implemented in this step, and the 8-step time window is maintained. This thesis utilises the scikit-learn library for cross-validation, employing the default number of folds (k) set to five. Common choices for k include 3, 5, and 10. Given that this study may involve short periods of series data, a value of $k = 5$ is selected. This decision is based on experiments that revealed that $k = 5$ offers a favourable balance between computational efficiency and low bias in estimating model performance.

As an example, the result of neighbourhood selection targeting PM_{10} in the London dataset is shown in Table 3.9. The table shows that the best model performances are mostly obtained when involving three neighbouring stations. The same results are also observed from the Delhi and Beijing datasets. Thus, three neighbouring stations and a target station are kept for the rest of the model evaluation. For additional information, the WHO Global Air Quality guidelines stipulate that the daily PM_{10} concentration should not exceed $45\mu g/m^3$. Additionally, interim targets are provided to facilitate a gradual transition from high to lower concentrations. These interim targets are set at 150, 100, 75, and 50 m$\mu g/m^3$ for targets 1, 2, 3, and 4, respectively.

After evaluating spatial correlation among stations, three neighbouring stations with the strongest correlation coefficients to the target station are carefully selected. The selected neighbouring stations for NO_2 and PM_{10} in London are reported in Table 3.10, sorted from largest to smallest coefficients. Table A.5 and Table A.6 present the selected neighbouring stations for Beijing and Delhi, respectively.

Figure 3.10 depicts the locations of two target stations (S^1 and S^2), along

Table 3.9: Average of RMSE (std. deviation) after 5-fold cross-validation for selecting the number of involved neighbouring stations targeting PM_{10} in London (measured in $\mu g/m^3$).

	Test Period		**Number of neighbouring stations**			
	Start	**End**	**2**	**3**	**4**	**5**
S^1	2019-11-03	2019-11-19	59.05(1.23)	**56.67(2.52)**	59.38(1.09)	60.49(0.83)
S^2	2020-03-28	2020-04-19	11.84(0.79)	**11.45(0.24)**	12.07(1.35)	12.51(1.52)
S^3	2020-05-31	2020-06-29	15.19(0.17)	14.98(0.30)	14.87(0.39)	**14.80(0.18)**
S^4	2020-03-07	2020-04-03	18.67(0.71)	**18.50(1.01)**	21.20(0.48)	20.68(0.36)
S^5	2020-04-11	2020-04-29	19.13(0.36)	**17.82(0.43)**	18.43(0.75)	18.01(0.31)
S^6	2020-03-23	2020-05-14	21.41(1.20)	21.74(1.05)	21.25(0.83)	**21.09(0.71)**
S^7	2020-04-22	2020-05-28	21.78(0.57)	**21.47(0.53)**	21.51(0.58)	21.57(0.78)
S^8	2019-03-16	2019-04-03	45.06(1.66)	**42.80(0.82)**	48.56(3.10)	50.32(2.03)
S^9	2019-05-23	2019-06-13	25.49(2.57)	25.68(1.67)	26.23(3.07)	**25.42(1.00)**
S^{10}	2020-03-29	2020-05-06	15.35(0.49)	**15.24(0.83)**	15.27(0.63)	14.60(0.46)

Table 3.10: Strongest correlation coefficient for neighbouring stations selection in London data.

	Strongest corr. coeff. (NO_2)			**Strongest corr. coeff. (PM_{10})**		
Target station	1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}
S^1	S^9	S^8	S^5	S^8	S^9	S^{10}
S^2	S^3	S^6	S^4	S^{10}	S^9	S^4
S^3	S^2	S^4	S^6	S^4	S^8	S^9
S^4	S^6	S^3	S^5	S^8	S^{10}	S^5
S^5	S^{10}	S^9	S^8	S^4	S^8	S^{10}
S^6	S^4	S^3	S^2	S^3	S^4	S^9
S^7	S^8	S^9	S^1	S^8	S^9	S^{10}
S^8	S^9	S^{10}	S^1	S^4	S^{10}	S^9
S^9	S^8	S^1	S^{10}	S^8	S^{10}	S^4
S^{10}	S^5	S^9	S^8	S^8	S^4	S^9

with their selected neighbouring stations and respective distances in the London dataset. Based on Pearson's correlation coefficient evaluation (please refer to Table 3.10), the neighbouring stations for S^1 are S^9, S^8, and S^5 (for NO_2), and S^{10}, S^9, and S^4 (for PM_{10}). However, it is important to note that these neighbouring stations may not always be the closest to the target stations. The selection is based on Pearson's correlation, not on distance. For example, in Fig. 3.10, the neighbouring stations for S^2 include station S^4 (or IS2) for both NO_2 and PM_{10}, even though this station is geographically farther compared to some other stations.

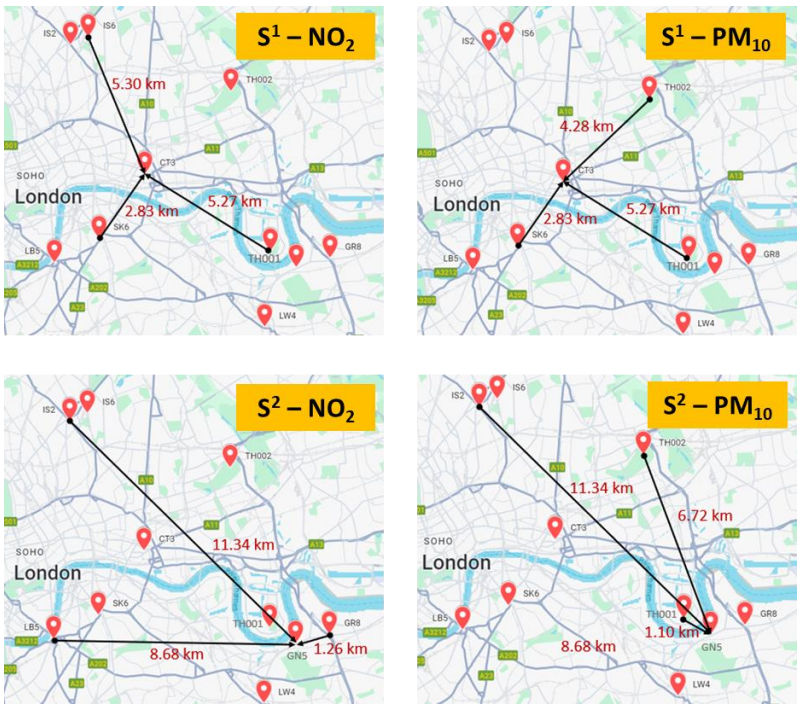


Figure 3.10: Locations of two target stations (S^1 and S^2), along with their selected neighbouring stations and respective distances in the London dataset.

3.9 Imputation Performance

3.9.1 Model Architecture Evaluation

The final model architecture proposed in this study is verified in this section. Several alternative autoencoder architectures are derived from the base model, created by expanding the base model's layers and modifying the number of output filters. There are countless possibilities for combining architecture and output filters in designing autoencoders, or neural networks in general. There are no rigid rules governing this design process. With such vast options available, it is crucial to establish limitations in the design process. In this study, the number of output filter layers will be reduced in the encoder part and expanded in the decoder part. As indicated in Table 3.11, both increases and decreases in the number of output filters, such as 10, 15, 20, or 30, are explored. Additionally, there is a slight increase in the number of layers from the base layer.

The proposed kernel and stride sizes are identical to preserve the base model's properties. Additionally, all proposed models are unpadded. As presented in Table 3.11, six different autoencoder architectures, denoted as $M1$, $M2$, ..., $M6$, are presented. All models have a kernel size of 2, a stride of 1, and no padding is applied

Table 3.11: Proposed autoencoder architectures.

	Assigned output filters
$M1$	$L^1 40 - L^2 30 - L^3 20 - L^4 30 - L^5 40 - L^6 4$
$M2$	$L^1 50 - L^2 40 - L^3 30 - L^4 20 - L^5 30 - L^6 40 - L^7 50 - L^8 4$
$M3$	$L^1 50 - L^2 40 - L^3 30 - L^4 20 - L^5 10 - L^6 20 - L^7 30 - L^8 40 - L^9 50 - L^{10} 4$
$M4$	$L^1 80 - L^2 70 - L^3 50 - L^4 30 - L^5 10 - L^6 30 - L^7 50 - L^8 70 - L^9 80 - L^{10} 4$
$M5$	$L^1 75 - L^2 60 - L^3 45 - L^4 30 - L^5 15 - L^6 30 - L^7 45 - L^8 60 - L^9 75 - L^{10} 4$
$M6$	$L^1 80 - L^2 50 - L^3 30 - L^4 20 - L^5 10 - L^6 20 - L^7 30 - L^8 50 - L^9 80 - L^{10} 4$

to any layer. The $M1$ is the base model for characterising the spatiotemporal features. As an example, this section presents the outcomes obtained from air quality data collected in Beijing, focusing on CO as the target pollutant. A 40% missing rate is applied to the target station and a 20% missing rate to the neighbouring stations for model evaluation. The experiment also involves 5-fold cross-validation. Based on the spatiotemporal evaluation, a fixed input size of 8×4 is used for model selection.

According to the final prediction results obtained from each model, $M6$ achieved the most precise imputation outcomes, as demonstrated in Table 3.12. Among the ten monitoring stations, $M6$ outperformed other models by providing the best prediction results for six stations. For instance, $M6$ predicted the missing data for S^8 with an RMSE value of 240.88 $\mu g/m^3$, which is approximately 30%

Table 3.12: Average RMSE ($\mu g/m^3$) for deep autoencoder architecture selection in Beijing, focusing on CO pollutants.

	Test Period		**Autoencoder model**					
	Start	**End**	***M*1**	***M*2**	***M*3**	***M*4**	***M*5**	***M*6**
S^1	2016-06-14	2016-07-26	207.27	210.33	189.25	191.37	189.56	**183.95**
S^2	2016-09-13	2016-10-19	432.21	431.53	435.73	**420.39**	437.41	435.33
S^3	2016-08-10	2016-09-06	204.86	205.77	**191.46**	201.56	202.59	199.66
S^4	2016-10-18	2016-11-25	395.77	388.99	**363.43**	369.30	372.26	373.27
S^5	2016-08-02	2016-08-29	312.01	305.28	311.99	309.76	298.78	**279.61**
S^6	2016-06-14	2016-07-26	207.27	210.33	189.25	191.37	189.56	**183.95**
S^7	2016-08-04	2016-08-25	216.54	213.53	204.10	199.69	203.75	**189.93**
S^8	2016-10-15	2016-11-08	346.45	329.10	248.19	246.75	241.69	**240.88**
S^9	2016-10-09	2016-10-26	349.76	338.14	**287.25**	315.04	312.94	302.92
S^{10}	2016-06-25	2016-07-27	187.51	180.91	164.21	171.75	169.32	**162.83**

more accurate than the base model's performance. This study shows that deeper model architectures yield better predictions, with ten-layered models outperforming six- and eight-layered models in most cases. However, excessively deep architectures were avoided to prevent the latent space from becoming too small. The deeper the architecture of the autoencoder from the first layer, with a certain number of output filters, the smaller the latent space will be. If the latent space is too small, then the autoencoder will not be very effective, because with such a small size, this latent space will be expanded in the decoder part. A latent space that is too small does not carry effective information when expanded in the decoder part, so the process of reconstructing input sets from the first layer to the last layer will also not be effective.

3.9.2 Short Interval Imputation

The term *short interval* refers to a missing period created by eliminating specific values from the actual data with a designated missing rate. The initial random state determines the values removed from the original data. This setting can be specified during programming. For this study, some values from the actual data are deliberately removed with four different missing rates (i.e., 20%, 40%, 60%, and 80%). An example of a test set missing pattern variation at station S^3 in the London dataset is illustrated in Figure 3.11. The figure shows 648 hourly samples of NO_2, collected from 20-Feb-2020 13:00:00 to 20-Mar-2020 00:00:00. The white stripes

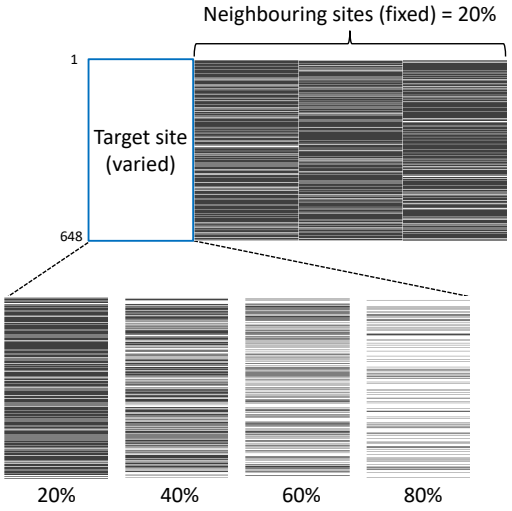


Figure 3.11: Short-interval missing patterns in the test set obtained from station S^3 of London dataset.

signify the missing values. As depicted in the figure, more missing values result in more prominent white strips. These values are then imputed with zeros. While the missing rate at the target station changes, the missing rate at neighbouring stations remains fixed at 20%.

Representative monitoring stations for short-interval imputation are shown in Table 3.13. The table lists two monitoring stations for each city, encompassing all pollutants in the respective dataset. Therefore, there are a total of 12 experiments reported. Table 3.14 demonstrates the imputation results for each experiment, which numbers detailed in Table 3.13. The imputation performances are evaluated using three different error metrics, i.e., RMSE, MAE and R^2.

The proposed method in this study is less effective in imputing missing values of NO_2 pollutants in Delhi compared to other monitoring stations. To provide more detailed information, model performance for NO_2 pollutants in Delhi will be discussed separately in Section 3.9.4. Generally, lower missingness levels result in lower RMSE/MAE values and higher R^2 scores. Due to variations in the physical nature of each pollutant, the RMSE/MAE values may differ significantly. For instance, RMSE/MAE values for some pollutants are considerably higher than PM_{10}. Hence, the R^2 score is introduced to provide a more intuitive performance measure. As shown in Table 3.14, the proposed method yields satisfactory results with an R^2 score greater than 0.8 for all target stations at a 20% missing rate, ranging between 0.80 and 0.95. At 40% and 60% missing levels, our proposed model maintains its performance, providing R^2 scores between 0.72 and 0.94. However, as the missing

Table 3.13: Properties of short-interval imputation experiment.

				Train Period		**Test Period**	
No.	**City**	**station**	**Pollutant**	**Start**	**End**	**Start**	**End**
1	London	S3	NO_2	2018-01-01	2019-10-21	2020-02-20	2020-03-20
2	London	S3	PM_{10}	2018-01-01	2019-11-18	2020-03-23	2020-04-20
3	London	S7	NO_2	2018-01-01	2019-09-29	2020-09-23	2020-10-13
4	London	S7	PM_{10}	2018-01-01	2019-11-23	2020-11-03	2020-12-01
5	Delhi	S2	NO_2	2018-02-05	2019-07-25	2020-05-31	2020-07-01
6	Delhi	S2	$PM_{2.5}$	2018-02-03	2019-07-17	2019-08-28	2019-10-27
7	Delhi	S7	NO_2	2018-02-05	2019-07-10	2019-11-21	2019-12-14
8	Delhi	S7	$PM_{2.5}$	2018-02-05	2019-07-16	2020-02-10	2020-04-22
9	Beijing	S1	CO	2013-03-03	2015-08-28	2016-11-11	2016-12-27
10	Beijing	S1	O_3	2013-03-03	2015-08-04	2016-12-10	2016-12-27
11	Beijing	S6	CO	2013-01-03	2015-09-13	2016-06-14	2016-07-26
12	Beijing	S6	O_3	2013-01-03	2015-08-16	2016-06-14	2016-07-26

Table 3.14: Performance metrics of short-interval imputation for all experiments described in Table 3.13.

		Experiment no.											
Rate	**Metrics**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
20%	RMSE	7.83	5.92	5.24	5.57	7.32	15.49	38.72	16.71	471.88	8.91	92.25	15.17
	MAE	5.99	3.97	3.87	3.90	4.07	9.14	22.94	10.81	296.36	3.93	71.97	10.78
	R^2	0.85	0.81	0.89	0.83	0.44	0.95	0.80	0.85	0.93	0.81	0.89	0.95
40%	RMSE	8.51	6.09	5.33	8.63	6.49	16.53	50.26	17.75	559.37	7.56	98.87	18.06
	MAE	6.62	4.08	4.05	4.69	4.11	9.36	28.63	11.10	354.90	3.85	77.51	13.13
	R^2	0.81	0.79	0.88	0.79	0.52	0.94	0.72	0.83	0.91	0.86	0.88	0.93
60%	RMSE	10.24	6.93	6.37	8.64	8.14	16.29	69.85	17.85	663.89	7.61	124.59	22.18
	MAE	7.74	4.67	4.99	5.17	4.68	9.55	40.04	11.42	421.72	4.19	93.15	16.66
	R^2	0.74	0.73	0.83	0.80	0.39	0.94	0.49	0.84	0.88	0.85	0.81	0.90
80%	RMSE	12.05	7.85	8.19	10.89	8.20	16.42	76.93	18.84	778.34	7.95	149.01	26.59
	MAE	9.21	5.53	6.47	5.77	4.78	9.98	45.75	12.31	514.88	4.76	111.02	20.43
	R^2	0.64	0.65	0.72	0.75	0.39	0.93	0.31	0.83	0.83	0.84	0.73	0.86

rate increases to 80%, more imputation errors occur, leading to a decline in R^2 scores. In experiment 1, for example, the R^2 value decreases from 0.85 at a missing rate of 20% to only 0.64 at 80%. Among the selected test period, the imputation of missing values of $PM_{2.5}$ at station S^2 in Delhi (i.e., experiment no. 6) delivers the most satisfactory results, achieving R^2 scores greater than 0.9 at all levels of missingness.

The proposed autoencoder utilises the data from neighbouring stations to effectively fill in missing values while accounting for prediction errors. Even in the case of severely corrupted data at the target station, our model and method have demonstrated the ability to achieve desirable results, as evidenced by satisfying performance metrics in most stations (please refer to Table 3.14). In conclusion, the proposed method can provide satisfactory accuracy for imputing missing values over a short interval.

3.9.3 Long Interval Imputation

In contrast to the short-interval method, which generates missing values based on a specific random state, the long-interval consecutive process involves removing all data at the target station for a specific period. In this regard, a minimum missing period of 400 hours is set. Figure 3.12 illustrates a long-interval missing values test set pattern applied to S^8 (Nongzhanguan station) of the Beijing dataset, consisting of 514 hourly samples from 23-Sep-2016 05:00:00 to 14-Oct-2016 14:00:00. As seen

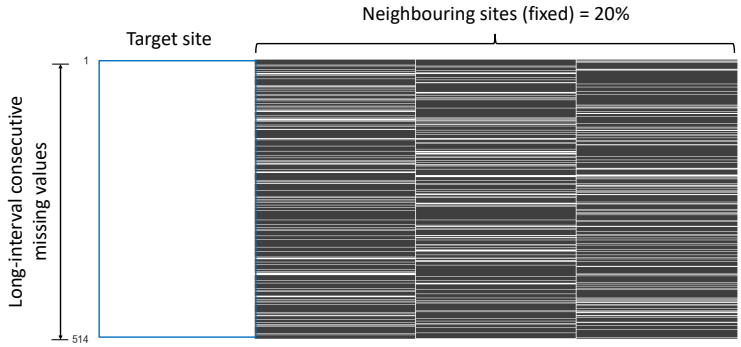


Figure 3.12: Long-interval missing patterns in the test set obtained from station S^8 of Beijing dataset.

in the figure, the values at the target set are entirely missing and replaced then will be replaced with zeros. Since no data from the target station is available, the autoencoder predicts the missing values based entirely on the adjacent available data.

Table 3.15 presents the results of six experiments conducted to represent long-interval consecutive imputation scenarios. One station is selected in each dataset, and all pollutants are considered. Station S^5, S^6, and S^8 represent London, Delhi, and Beijing datasets, respectively. The error metrics obtained from the long-interval imputation for specific missing periods are also presented in Table 3.15. The proposed model effectively imputes missing values for a minimum of 400 hours (about 17 days), with some experiments yielding R^2 scores of 0.90 and higher. However, the Delhi dataset's S^6 station measuring NO_2 produces the lowest R^2 score, consistent in the case of short-interval imputation. It is observed that stations with low correlation coefficients could affect imputation performance, and this issue will be discussed separately in section 3.9.4.

Figure 3.13 compares actual and imputed values for the experiments detailed in Table 3.15. The plots also exhibit a 95% confidence interval following the

Table 3.15: Results of long-interval consecutive imputation.

No.	**City**	**Station**	**Pollutant**	**Start**	**End**	**RMSE**	**MAE**	**R^2**
1	London	S^5	NO_2	2021-01-12	2021-01-30	5.840	4.371	0.845
2	London	S^5	PM_{10}	2020-11-10	2020-12-27	3.57	2.29	0.79
3	Delhi	S^6	NO_2	2020-05-14	2020-06-12	12.16	8.56	0.47
4	Delhi	S^6	$PM_{2.5}$	2019-09-29	2019-11-20	49.99	31.44	0.90
5	Beijing	S^8	CO	2016-03-25	2016-04-20	132.91	96.81	0.95
6	Beijing	S^8	O_3	2016-09-23	2016-10-14	13.91	8.29	0.92

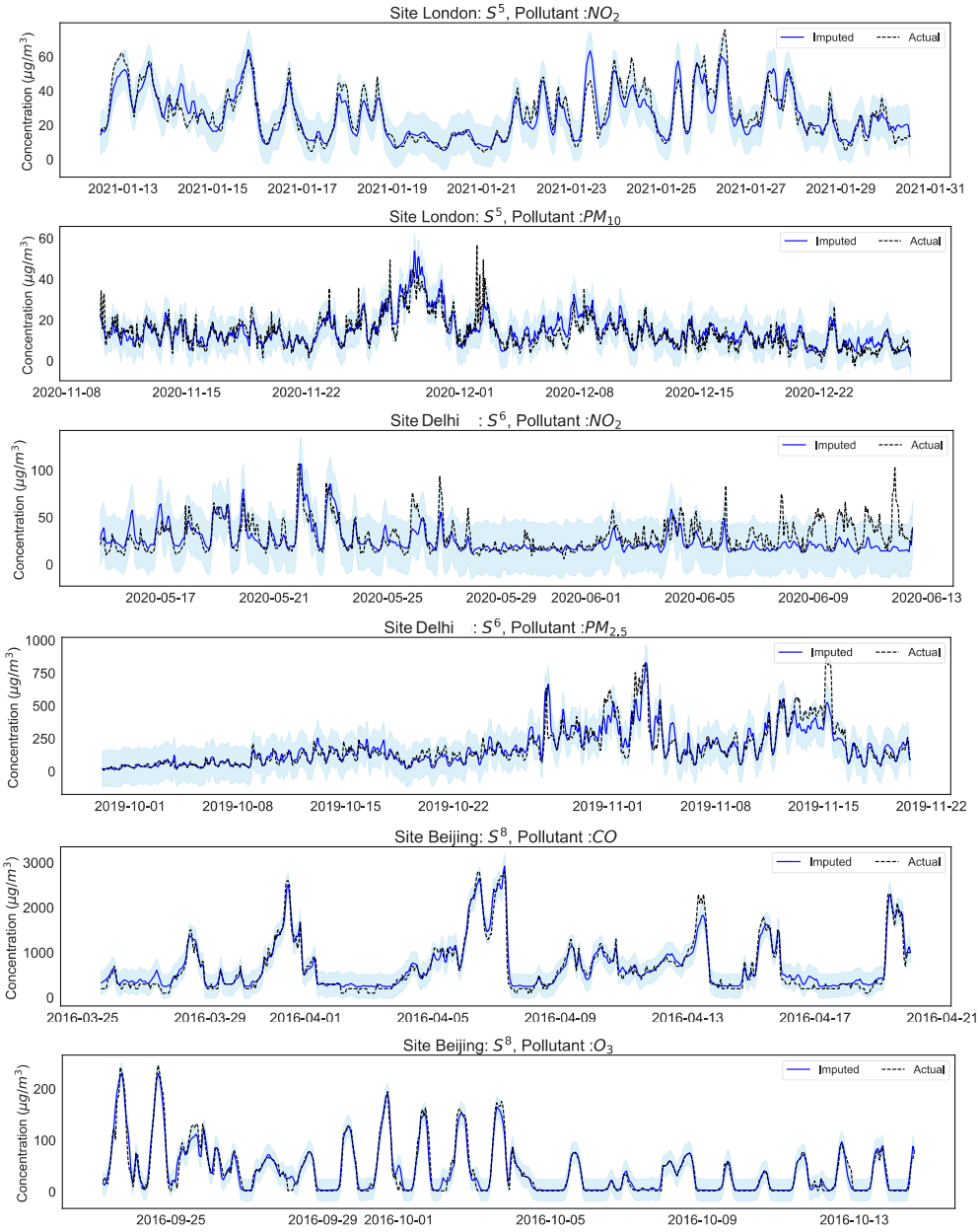


Figure 3.13: Plot of long-interval missing imputation between actual and imputed values along with 95% confidence intervals.

methodology proposed by [194, 195]. This study determines the confidence interval by adding and subtracting RMSE two times from the imputed values. Utilising the RMSE value to form the confidence interval provides a better summary than standard deviation and enables a direct assessment of the imputed values' uncer-

tainty [147]. Figure 3.13 indicates that the imputed values can capture the actual values' dynamics, and the autoencoder model can effectively recognise the missing values. Based on the shaded interval areas, only 5% of imputed values fall outside the confidence interval.

3.9.4 Effect of Correlation Levels

The correlation levels of coefficients between paired stations can affect the proposed method's performance. For example, in the case of short- and long-interval imputa-

Table 3.16: Coefficient of correlation among stations measuring NO_2 in Delhi data.

	$\boldsymbol{S^1}$	$\boldsymbol{S^2}$	$\boldsymbol{S^3}$	$\boldsymbol{S^4}$	$\boldsymbol{S^5}$	$\boldsymbol{S^6}$	$\boldsymbol{S^7}$	$\boldsymbol{S^8}$	$\boldsymbol{S^9}$	$\boldsymbol{S^{10}}$
S^1	1.00									
S^2	0.41	1.00								
S^3	0.04	0.07	1.00							
S^4	0.14	0.32	-0.03	1.00						
S^5	0.24	0.35	0.13	0.16	1.00					
S^6	0.35	0.65	0.01	0.27	0.30	1.00				
S^7	0.12	0.26	0.05	0.29	0.38	0.24	1.00			
S^8	0.50	0.58	0.06	0.33	0.37	0.55	0.26	1.00		
S^9	0.32	0.21	0.09	0.00	0.02	0.22	0.07	0.29	1.00	
S^{10}	0.33	0.49	-0.24	0.34	0.27	0.55	0.30	0.48	0.12	1.00

Table 3.17: Coefficient of correlation among stations measuring $PM_{2.5}$ in Delhi data.

	$\boldsymbol{S^1}$	$\boldsymbol{S^2}$	$\boldsymbol{S^3}$	$\boldsymbol{S^4}$	$\boldsymbol{S^5}$	$\boldsymbol{S^6}$	$\boldsymbol{S^7}$	$\boldsymbol{S^8}$	$\boldsymbol{S^9}$	$\boldsymbol{S^{10}}$
S^1	1.00									
S^2	0.90	1.00								
S^3	0.71	0.72	1.00							
S^4	0.86	0.84	0.72	1.00						
S^5	0.86	0.90	0.69	0.81	1.00					
S^6	0.81	0.84	0.71	0.81	0.84	1.00				
S^7	0.82	0.84	0.76	0.82	0.83	0.87	1.00			
S^8	0.83	0.82	0.67	0.75	0.76	0.71	0.74	1.00		
S^9	0.85	0.85	0.73	0.82	0.81	0.79	0.80	0.79	1.00	
S^{10}	0.85	0.89	0.67	0.81	0.88	0.82	0.78	0.76	0.80	1.00

tions, the NO_2 measurements in Delhi have led to poor estimations. The correlation coefficients of NO_2 and PM_{10} in the Delhi dataset are presented in Table 3.16 and Table 3.17, respectively. The pair of $S^3 - S^6$ shows the minimum coefficient of 0.01 for NO_2, and the correlation between S^3 and S^{10} is even negative. The maximum coefficient correlation for NO_2 is only 0.65, obtained from $S^2 - S^6$. In contrast, the monitoring stations measuring PM_{10} in the same city yield much stronger correlation coefficients. The minimum coefficient of PM_{10} is 0.67, calculated from the pairs of $S^3 - S^8$ and $S^3 - S^10$, while the maximum coefficient of 0.90 is observed in the $S^1 - S^2$ pair.

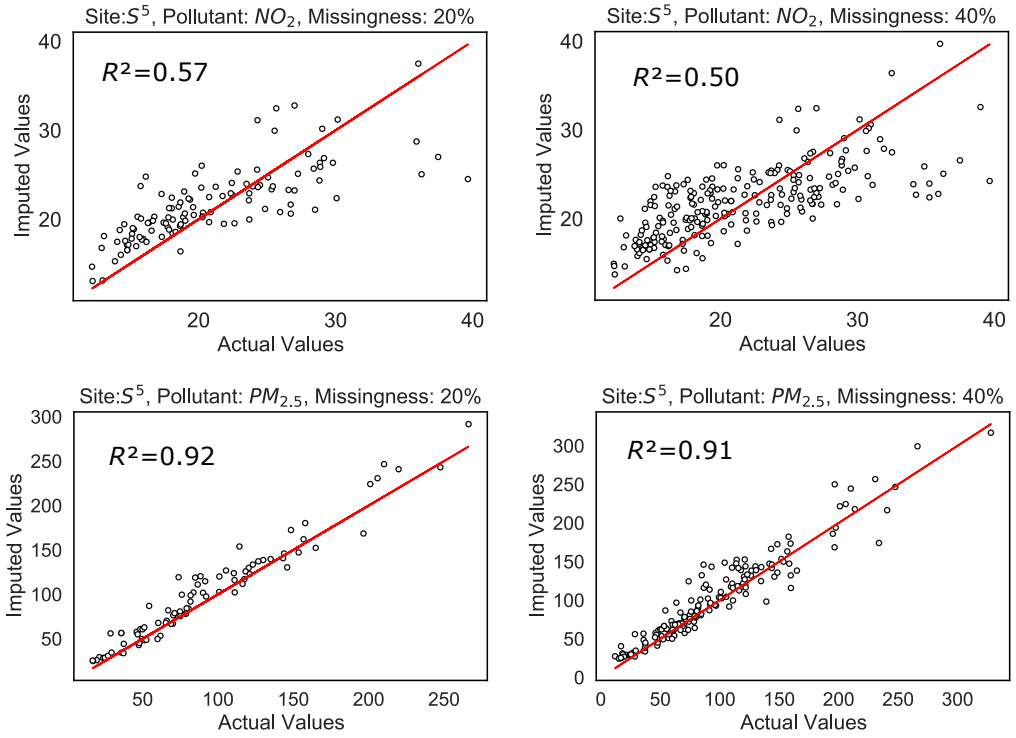


Figure 3.14: Scatter plot of short-interval imputation at Delhi station S^5, with 20% and 40% of missingness levels.

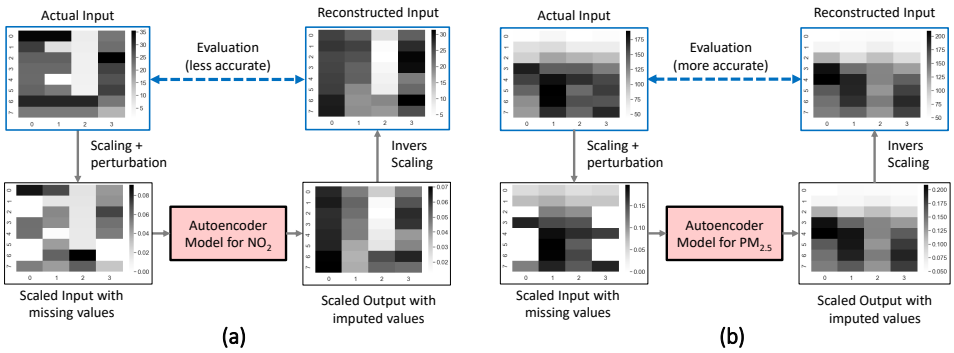


Figure 3.15: Example of input and output sets retrieval before and after denoising process in Delhi station S^5: (a) retrieval of NO_2, and (b) retrieval of $PM_{2.5}$.

When stations have a very low coefficient correlation, the imputation values tend to be highly biased. Various experiments have been conducted to study this phenomenon, and some of the results are presented in Figure 3.14. The figure shows the scatter plots between the actual and imputed NO_2 and $PM_{2.5}$ at S^5 of the Delhi dataset, where the experiments are designed for a short-interval missing scenario with 20% and 40% missing rates. For NO_2, the test period spans from 06-Apr-2020 at 04:00:00 to 29-Apr-2020 at 23:00:00, while the $PM_{2.5}$ period runs from 22-Feb-2020 at 19:00:00 to 11-Mar-2020 at 14:00:00. Despite being conducted at the same monitoring station, the imputation results for the two pollutants differ significantly. While the $PM_{2.5}$ imputation values are relatively close to the diagonal line, the missing estimations for NO_2 are more scattered.

The input set for NO_2 pollutants consists of Station S^5 and three neighbouring stations (S^7, S^8, and S^2). The correlation coefficients for $S5 - S7$, $S5 - S8$, and $S5 - S7$ pairs are 0.38, 0.37, and 0.35, respectively, indicating low correlations. In contrast, the input set for $PM_{2.5}$ pollutants, which includes joint stations S^5, S^2, S^{10}, and S^1, yields much stronger correlation coefficients. Specifically, the computed correlation coefficients for $S^5 - S^2$, $S^5 - S^{10}$, and $S^5 - S^1$ are 0.90, 0.88, and 0.86, respectively. Weak correlations can lead to input sets that appear more randomised, resulting in neighbouring station data that contributes insufficient knowledge to the model. Figure 3.15 provides a more intuitive explanation of this issue. The figure displays the first input set fed to the model with a 40% missing rate for NO2 and PM25. The figure shows that the reconstructed input for NO_2 is less accurate than the reconstructed input for $PM_{2.5}$. The weak correlation significantly affects the values in each input set column, making it challenging for the model to estimate the missing parts.

3.9.5 Comparison with Other Methods

This section aims to validate the proposed model's effectiveness compared to existing methods. The results from univariate and multivariate imputation methods are also presented. For imputing the missing values, the univariate method for imputing utilises the existing values in that feature dimension, while the multivariate method attempts to leverage the non-missing data across all feature dimensions. The univariate imputations selected for this study include the most frequent, median, and mean methods. As for the multivariate imputation, this study employs four estimators: Bayesian ridge, decision tree, extra-trees, and k-nearest neighbours.

The effectiveness of the proposed model against other methods for all monitoring stations is demonstrated through 60 experiments covering different cities, stations, and pollutants in the datasets. For the London dataset, the training data for NO_2 and PM_{10} spans from January 2018 to around October 2019, while the test sets are taken from several unbroken segments around November 2019 to January 2021. Short and long-interval perturbation procedures are combined for the training and test sets, with the perturbation step removing approximately 45% of the target training set and 50% of the test set. To obtain less biased results, 5-fold cross-validation is implemented in the dataset. This section examines the effectiveness of the proposed method compared to commonly used methods. As mentioned earlier in Section 3.7.5, the typical probability of missing data is around 20% [193]. Therefore, in this section, the missing data rate is increased to approximately 50%, which is higher than the typical rate. The autoencoder model is trained with higher levels of missing values for both the training and test sets in this section.

The training period for NO_2 and $PM_{2.5}$ pollutants in the Delhi dataset is from February 2018 to mid-July 2019, while the test period spans from July 2019 to July 2020. Similar to the London dataset, perturbation procedures are applied, resulting in missing rates of approximately 45% and 50% for the training and test sets in the target station. In Beijing monitoring stations, CO and O_3 pollutants are also treated similarly, with the training data selected from March 2013 to around September 2015 and the testing data chosen from September 2015 to February 2017. The missing values in the target station for training and test steps are kept at a rate of 45% and 50%, respectively. Figure 3.16 displays bar charts illustrating the proportion of the RMSE scores obtained from each method.

In Figure 3.16, the performance of our proposed autoencoder model and seven commonly used imputation methods are shown. The abbreviations used for other methods are as follows: $Most$ (most frequent imputation), Med (median imputation), $Mean$ (mean imputation), $DecT$ (decision tree regressor), ExT (extra-

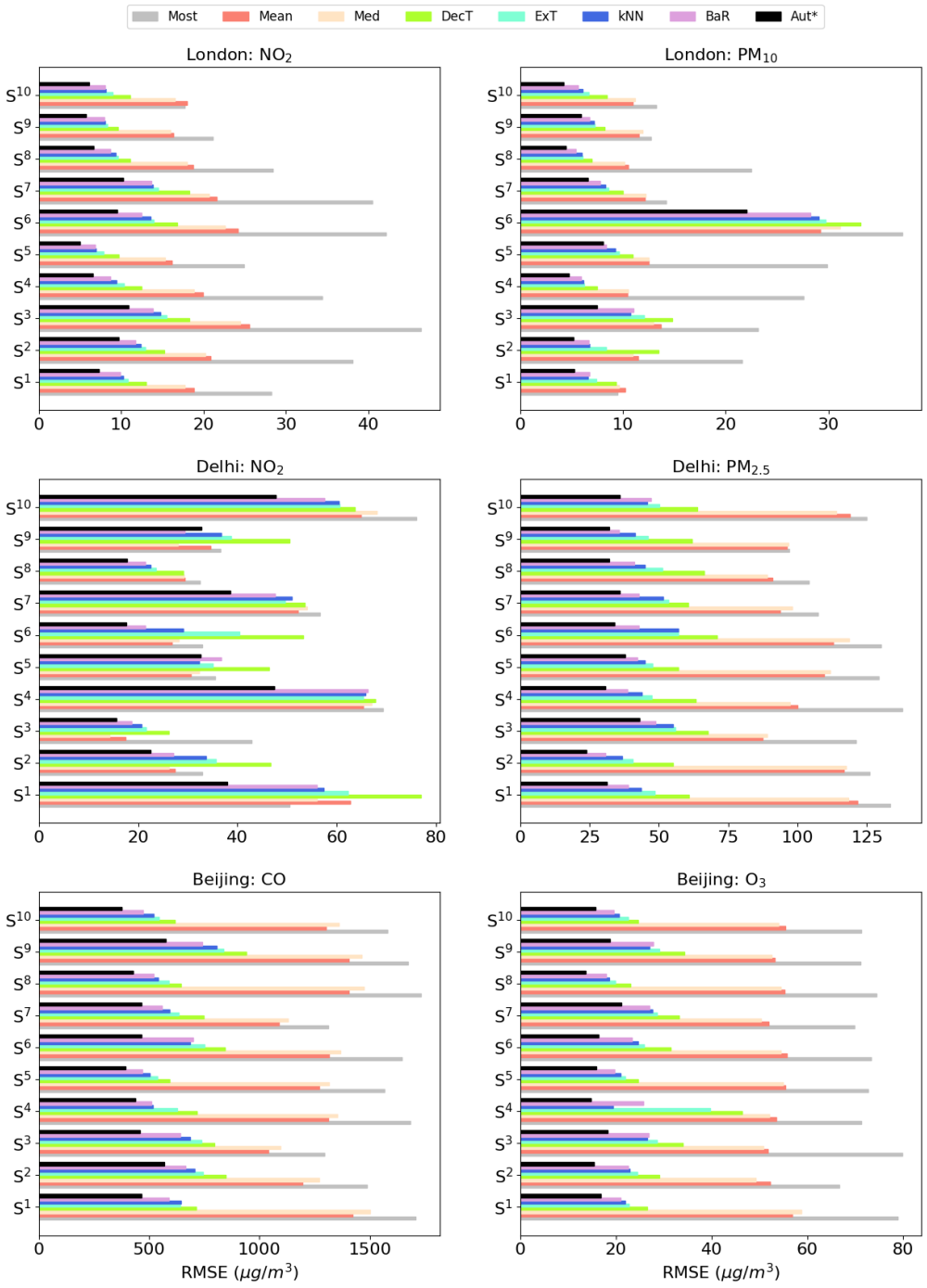


Figure 3.16: Performance comparison of the proposed model and commonly used methods.

tress regressor), *KNN* (k-nearest neighbours regressor), *BaR* (Bayesian ridge regressor) and *Aut* (proposed autoencoder). Distinct colours are used to represent each

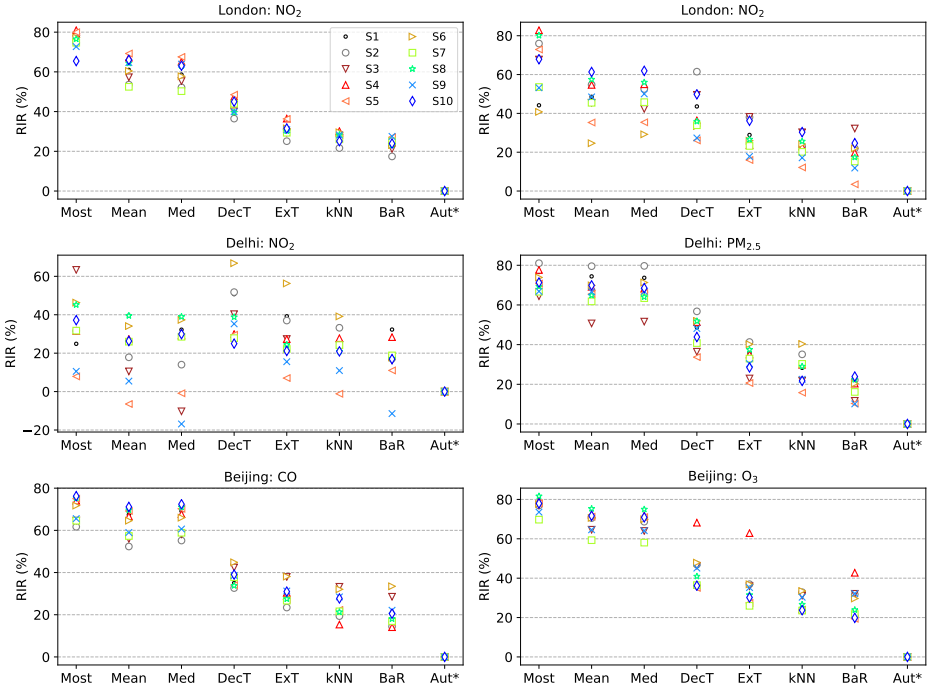


Figure 3.17: Performance comparison of the proposed model against commonly used methods.

method, and the black-filled areas in the chart represent the results of our proposed autoencoder method (*Aut*).

The figure shows that univariate imputation using statistic properties such as most frequent, median and mean leads to highly inaccurate imputation results. In contrast, multivariate imputation techniques produce significantly lower imputation errors. Except for NO_2 measurements at Delhi monitoring stations, our proposed method outperforms all other methods for all stations and pollutants. However, other methods yield marginally better results in three monitoring stations (S^3, S^5, and S^9). As discussed in the previous section, weak correlations among stations are responsible for the lower performance of the proposed method. It is worth noting that the commonly used imputation methods (decision tree regressor, extra trees regressor, k-nearest neighbours regressor, and Bayesian ridge regressor) are implemented using their default parameters provided by the Python scikit-learn library. Therefore, the comparison results depicted in this figure are based on these default settings. The training and test sets are allocated in proportions identical to those employed in the proposed method.

In Figure 3.17, the rate of improvement on RMSE (RIR) is presented. A pos-

Table 3.18: Average of RIR values calculated from all stations.

Method	**Average of RIR$^{(\text{existing,proposed})}$**
Most frequent	65.21%
Mean	55.14%
Median	54.33%
Decision tree	41.69%
Extra-trees	30.66%
k-nearest neighbours	25.45%
Bayesian ridge	20.82%
Proposed Autoencoder	0.00%

itive RIR value indicates that the proposed model outperforms other methods, while a negative RIR value implies that other models perform better than the proposed model. Our autoencoder model shows a significant improvement in RMSE values ranging from 50% to 80% compared to the most frequent, median, and mean imputations in most cases. Additionally, our proposed method produces positive RIR values compared to multivariate imputation techniques (Bayesian ridge, decision tree, extra-trees, and k-nearest neighbour imputation methods), with improvement between 10% and 50%.

However, for Delhi's NO_2 measurements, the proposed method results in six negative RIR values, with half of them occurring in station S^5. Here, mean, median, and kNN imputations perform better than the proposed model, with marginal improvements of 6.46% , 0.87% , and 1.15% in RIR values, respectively. Median imputation is responsible for half of the six negative RIR values, contributing the lowest RMSE for monitoring station S^9, which is approximately 17% better than our proposed model.

To obtain a comprehensive understanding of the performance of each imputation method across all stations and pollutants, the average RIR values are computed, as presented in Table 3.18. The proposed model outperforms univariate imputations, resulting in an average RIR improvement of around 50% to 65%. In the case of multivariate imputation, the proposed method results in an average RIR improvement ranging from about 20% to 40%.

3.10 Summary

Missing values are a common occurrence when collecting real-world data. Due to various factors, measurement systems may experience missing values, some of which could be critical. The presence of missing data can impact the interpretation of studies and affect the functioning of public services related to air quality. An imputation method must be proposed to overcome the missing data issue. Furthermore, understanding the spatiotemporal characteristics of air pollutant data can enhance the robustness of air quality missing data imputation.

This study addresses the challenges of implementing a suitable method for imputing missing air quality data. Inspired by the denoising autoencoder's ability to reconstruct corrupted data, an imputation method that utilises both temporal and spatial data to improve imputation accuracy is proposed. An optimal temporal window size of 8-time steps and a spatial combination of 3 neighbouring stations is determined, resulting in an 8×4 input set for the model. The input sets are aggregated to obtain a single prediction at a specific time. This study conducted two imputation scenarios: short-interval imputation and long-interval consecutive imputation. For short-interval imputation, various levels of missingness were introduced (i.e., 20%, 40%, 60%, and 80%). In contrast, long-interval imputation steps removed all data in a specific period.

The performance of our proposed autoencoder model is compared with seven commonly used imputation methods, including most frequent imputation, median imputation, mean imputation, decision tree regressor, extra-trees regressor, k-nearest neighbours regressor, Bayesian ridge regressor. The results demonstrate that the proposed method and model yield satisfactory imputation outcomes, with $R^2 \geq 0.6$, even when all data in the target station are missing. However, degraded imputation performance occurs when stations are weakly correlated. Low correlation coefficients result in more irregular input values, which the proposed autoencoder model cannot recover effectively. The proposed model performs significantly better than univariate imputation techniques, improving up to 65% of the average RIR and 20% - 40% compared to multivariate imputation techniques.

Currently, the study utilises Pearson's correlation coefficient to evaluate the linear correlation between pollutant data from two stations. An alternative approach could involve implementing non-linear correlation methods, such as Spearman's rank correlation coefficient or Kendall's rank correlation coefficient, to identify more robust neighbouring stations for inclusion in the analysis. Another potential step is to address missing data in the deep learning model development process. Instead of

replacing missing values with zeros, alternative strategies could be explored, such as imputing the most frequent values or employing interpolation techniques. Adopting different strategies for handling missing data has the potential to impact the patterns within the input dataset substantially.

Chapter 4

Optimising Deep Learning at the Edge

The work in this chapter has been published in:

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Optimising Deep Learning at the Edge for Accurate Hourly Air Quality Prediction," *Sensors*, vol. 21, no. 4, p. 1064, Feb. 2021 [1].

4.1 Introduction

When implemented on embedded devices, deep learning (DL) models must be optimised for efficient design. As shown in Fig. 4.1, the optimisation can be performed at *algorithmic* and *hardware* levels [103]. Two common ways to optimise models at the algorithmic level are by conducting *model design* and *model compression* [107]. Model design optimisation seeks fewer parameters while designing the model. This strategy lowers memory size and reduces latency while maintaining the model's accuracy compared to the more complex models. The designers adapted the trained models to fit edge deployment in the model compression strategy. Some common techniques implemented in model compression include *parameter quantisation*, *model pruning*, and *knowledge distillation*. Parameter quantisation converts the original model parameters into a lower precision number with minimal degradation in model accuracy. While quantisation works on reducing the number representing weights, biases and activation functions of the original model, parameter pruning eliminates the less essential units comprising the original model. This method is associated with the dropout technique [196]. Dropout is a common technique in training deep neural networks due to its effectiveness in mitigating overfitting and

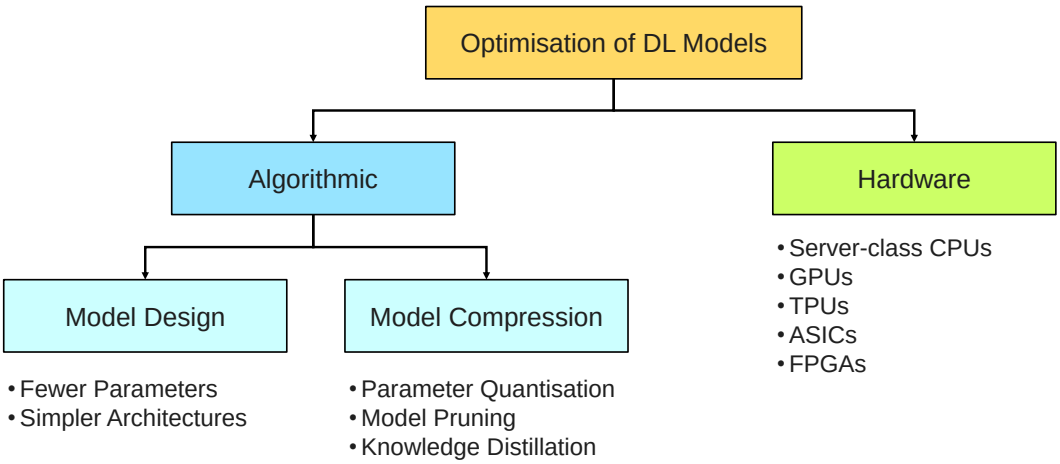


Figure 4.1: Optimisation options for deep learning models on embedded devices.

reducing the model's size. This method combats overfitting by randomly disabling a certain percentage of neurons in the network during the training process. Finally, knowledge distillation transfers knowledge from a larger model to a smaller model. The larger model can be a deep neural network or an ensemble model. The knowledge distillation strategy creates a smaller model by mimicking the behaviour of the larger model and trains the smaller model using outputs obtained from the larger one.

In addition to algorithmic level improvement, model optimisation can be performed at the hardware level. At this level, deep learning model training and inferencing phases can be accelerated by leveraging the computation power of server-class central processing units (CPUs), graphics processing units (GPUs), tensor processing units (TPUs), neural processing units (NPUs), application-specific circuits (ASICs) and field-programmable gate arrays (FPGAs). Custom low-density FPGAs can be utilised to construct deep learning accelerators with varied layers and kernels, enabling high-speed computation while preserving the reconfiguring ability [197]. Moreover, ASICs and FPGAs typically exhibit greater energy efficiency compared to traditional CPUs and GPUs [107].

The CPU is a standard component in almost all types of computers, making it an easily accessible solution for deep learning. However, CPUs are generally slower than TPUs or GPUs for tasks requiring high parallelism, such as deep learning. TPUs are specifically designed for deep learning operations, particularly those involving matrix operations [198]. However, TPUs are less flexible than CPUs because they are designed for specific tasks. GPUs are a powerful choice for deep learning optimisation, particularly for tasks requiring intensive and parallel processing [199].

However, their cost and complexity are important factors to consider. ASICs are specifically designed for particular tasks, enabling them to perform deep learning operations faster and more efficiently than general-purpose solutions. Meanwhile, FPGAs can be reconfigured to adapt to specific needs, offering both flexibility and high performance [200]. However, developing ASICs requires significant investment in time and resources and is less flexible than FPGAs. Although FPGAs are more flexible than ASICs, they may not always be as efficient as ASICs in performance and energy efficiency for certain tasks.

Many published works have shown the successful implementation of machine learning (including deep learning) in air quality research. Nonetheless, prior research on air pollution prediction has mainly focused on assessing deep learning model accuracy by comparing predicted values to the original dataset. This chapter aims to expand on this body of work by examining the deployment of deep learning models for air quality monitoring on edge devices. The post-training quantisation method, which falls under algorithmic-level optimisation, is adopted to achieve this goal. This technique compresses model parameters by converting floating points to lower precision numbers, reducing latency and model size without sacrificing accuracy. The quantisation technique can potentially enhance CPU and hardware accelerator latencies, leading to more efficient and effective deep learning models.

In recent years, various works have explored applying deep learning techniques in predicting air quality. This includes developing new data preprocessing techniques and proposing novel deep learning architectures. For example, Navares *et al.* [201] utilised Long Short-Term Memory (LSTM) to forecast PM_{10} and other air pollutants. The authors demonstrated that Recurrent Neural Networks (RNNs), which incorporate past context into their internal state, are well-suited for time-series problems. However, when dealing with longer time series, RNNs may fail to connect relevant information that occurred further in the past. Additionally, RNNs can suffer from the vanishing gradient problem due to cyclic loops. LSTMs are a type of recurrent neural network capable of learning and processing long-term dependencies in sequential data. LSTMs are commonly implemented to solve various problems related to sequence prediction. Additionally, LSTMs can effectively address the issue of vanishing gradients that happen during the training of neural networks.

Li *et al.* [202] implemented an LSTM neural network to predict hourly $PM_{2.5}$ concentration using combined historical air pollutant, meteorological, and time stamp data. The LSTM model proposed for one-hour predictions exhibited better performance compared to other models, including the spatiotemporal deep

learning (STDL), time-delay neural network (TDNN), autoregressive moving average (ARMA), and support vector regression (SVR). Another work by Xayasouk *et al.* [203] utilised LSTM and Deep Autoencoder (DAE) models to forecast $PM_{2.5}$ and PM_{10} concentrations for ten days. By varying the input batch size and measuring the overall average performance of both models, the proposed LSTM model yielded higher accuracy than the DAE model. In their study, Seng *et al.* [204] utilised an LSTM model to make predictions of air pollutant levels, including $PM_{2.5}$, CO, NO_2, O_3, and SO_2, at 35 monitoring stations located in Beijing. A new comprehensive model, Multi-Output and Multi-Index Supervised Learning (MMSL) has been proposed. This model leverages spatiotemporal data from present and surrounding stations to improve accuracy. To evaluate the effectiveness of the proposed model, a comparison was made with the existing time series model (Linear Regression, SVR, Random Forest and ARMA) and baseline models (CNN-LSTM and CNN-Bidirectional RNN). Xu *et al.* introduced a framework named HighAir in their work [205]. The framework utilised a hierarchical graph neural network based on an encoder-decoder architecture and consisted of LSTM networks.

Other researchers have proposed several hybrid deep learning models. For instance, Zhao *et al.* [206] conducted a study to compare the performance of ANN, LSTM, and LSTM-Fully Connected (LSTM-FC) models in predicting $PM_{2.5}$ levels. The authors concluded that the LSTM-FC model outperformed the other models. The proposed model has two components: an LSTM for modelling the local $PM_{2.5}$ concentrations and a fully connected network to capture the spatial dependencies between central and neighbouring stations.

Combining Convolutional Neural Network (CNN) and LSTM models has also been investigated [85, 207, 208]. According to Li *et al.* [209], utilising CNN-LSTM can improve the accuracy of $PM_{2.5}$ prediction. The authors used 1D CNN models to extract features from sequence data and LSTM units to predict future values. In real-world scenarios, input data can originate from multiple sources, creating spatiotemporal dependencies, as discussed by Qi *et al.* [192]. Besides using CNNs and LSTMs, Gated Recurrent Units (GRUs) in predicting $PM_{2.5}$ levels have also been investigated. Tao *et al.* [210] employed a bi-directional GRU with a one-dimensional CNN to forecast $PM_{2.5}$ concentration. The authors analysed the dataset attributes to determine the optimal input features for the proposed model.

Various deep learning optimisation techniques have been proposed recently in various application scenarios. Quantising weights and activation functions can reduce post-trained model size without retraining the model. This method is called the *post-training quantisation* [211]. Banner *et al.* [211] proposed 4-bit post-training

quantisation for CNNs. They designed an efficient quantisation method by minimising mean-squared quantisation error at the tensor level and avoiding retraining the model. Moreover, a mathematical background review for integer quantisation and its implementation on many existing pre-trained neural network models was presented by Wu *et al.* [212]. With 8-bit integer quantisation, the obtained accuracy matches or is within 1% of the floating-point model. Intended for mobile edge devices, Peng *et al.* [213] proposed a fully-integer-based quantisation method tested on an ARMv8 CPU. The proposed method achieved comparable accuracy to other state-of-the-art methods. Li and Alvarez [214] specifically proposed the integer-only quantisation method for the LSTM neural network. The result obtained is accurate, efficient, and fast to execute. Moreover, the proposed method has been deployed to various target hardware.

The previous works on air quality prediction have not specifically explored the optimisation of models for resource-constrained edge devices. Our work aims to extend this body of work around deep learning models for air quality monitoring by analysing the deployment of these models to edge devices. The post-training quantisation techniques are implemented to the baseline model using tools provided by TensorFlow framework [114], and the optimised model performance running on Raspberry Pi boards is evaluated.

4.2 Contributions

The chapter contributions are listed as follows:

- Designing a novel hybrid deep learning model for accurately predicting $PM_{2.5}$ pollutant level leveraging spatiotemporal aspects.
- Optimising the obtained models to lightweight versions suitable for edge devices.
- Examining model performances when running on edge devices.

4.3 Air Quality Data

4.3.1 Dataset and Preprocessing

This chapter uses the Beijing air quality dataset. There are 12 air quality monitoring stations, namely Aotizhongxin, Changping, Dingling, Dongsi, Guanyuan, Gucheng, Huairou, Nongzhanguan, Shunyi, Tiantan, Wanliu and Wanshouxigong. Regardless

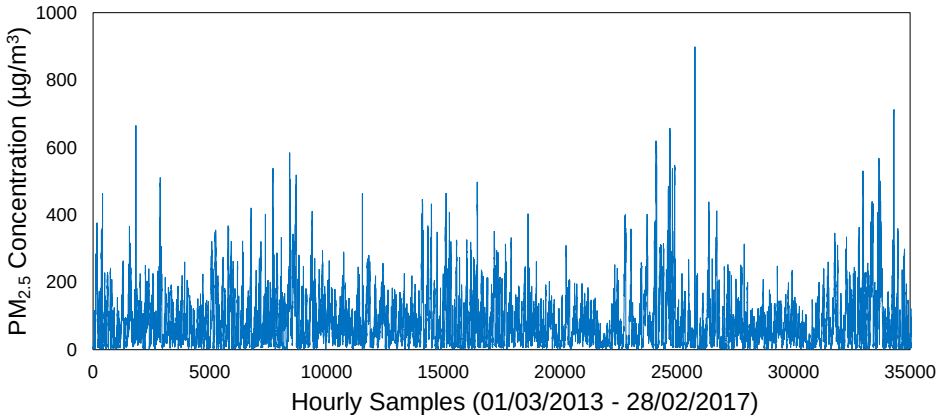


Figure 4.2: $PM_{2.5}$ concentration at Node 1 (Aotizhongxin monitoring site) from 1 March 2013 to 28 February 2017.

of the actual geographical location and each monitoring site's ability to gather pollutant and meteorological data, in this chapter, every monitoring site is considered merely as a *node*. Therefore, the complex monitoring site is modelled as a simple node. The term *node* is commonly linked to the end device where edge computing is typically performed. This chapter focuses solely on the data collected from each node and its relationship with other nodes. To provide a clear identification, node numbers are assigned to the mentioned 12 monitoring sites as follows: Aotizhongxin is Node 1, Changping is Node 2, Dingling is Node 3, Dongsi is Node 4, and so on.

To facilitate analysis, the dataset is divided into training and test sets. The training data encompasses the period from 1 March 2013 to 20 March 2016, while the test data covers 21 March 2016 to 28 February 2017. This division resulted in a total of 26,784 training samples and 8,280 test samples. This chapter focuses on predicting $PM_{2.5}$ concentrations. This chapter identifies the best model for short-term predictions of one-hour $PM_{2.5}$ concentrations. Figure 4.2 illustrates the $PM_{2.5}$ concentrations recorded at Node 1 (Aotizhongxin monitoring site).

Feature scaling is conducted to the input features during the training and testing. Feature scaling is a technique used to normalise the range of independent variables or features in data. It is commonly performed during the data preprocessing step called *data normalisation*. In this chapter, a min-max scaler is selected to normalise all input features within the range of 0 and 1. The general formula for achieving a min-max range of [0, 1] is as follows:

$$x^{'} = \frac{x - min(x)}{max(x) - min(x)} \tag{4.1}$$

Table 4.1: Correlation coefficients (r) among attributes at Node 1.

	$PM_{2.5}$	**PM_{10}**	**SO_2**	**NO_2**	**CO**	**O_3**	**Temp**	**Pres**	**Dewp**	**Rain**	**Wd**	**Wspd**
$PM_{2.5}$	1											
PM_{10}	0.87	1										
SO_2	0.49	0.47	1									
NO_2	0.67	0.65	0.44	1								
CO	0.76	0.65	0.57	0.66	1							
O_3	−0.15	−0.12	−0.22	−0.46	−0.32	1						
Temp	−0.09	−0.07	−0.36	−0.17	−0.37	0.58	1					
Pres	−0.02	−0.05	0.23	0.04	0.24	−0.42	−0.83	1				
Dewp	0.15	0.09	−0.29	0.12	−0.12	0.30	0.83	−0.78	1			
Rain	−0.01	−0.02	−0.04	−0.03	−0.01	0.03	0.04	−0.06	0.08	1		
Wd	−0.19	−0.12	−0.12	−0.24	−0.22	0.21	0.05	−0.02	−0.13	−0.01	1	
Wspd	−0.27	−0.17	−0.11	−0.48	−0.25	0.33	0.01	0.09	−0.33	0.00	0.31	1

4.3.2 Feature Selection

This chapter focuses on predicting $PM_{2.5}$ concentrations. As depicted in Table 4.1, $PM_{2.5}$ exhibits strong positive correlations with $PM_{2.5}$, NO_2, and CO (with correlation coefficients $r > 0.6$). It shows a moderate positive correlation with SO_2 (with a correlation coefficient of $r = 0.49$) and a weak negative correlation with O_3 (with a correlation coefficient of $r = -0.15$). Elevated levels of O_3 can facilitate the creation of secondary particles in conditions of intense atmospheric oxidation, leading to an increase in $PM_{2.5}$ concentrations. Conversely, a high concentration of $PM_{2.5}$ can reduce solar radiation and hinder the generation of O_3 [215].

Rain, air pressure, and temperature demonstrate the weakest correlations with $PM_{2.5}$. Then, only Rain, Pres, and Temp features are varied to determine the optimal number of input features. This resulted in four different combinations, and the recorded values of RMSE and MAE are presented in Table 4.2. It is important to note that the feature selection process was conducted specifically for Node 1. However, the results obtained from this step can be extrapolated to all other nodes.

Based on the results presented in Table 4.2, the best performance is achieved by excluding the rain attribute during training, resulting in a model with 11 input features. Consequently, the following attributes are selected as input features for our model: $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed. The same input features are used for all monitoring sites.

Table 4.2: Model performance based on different input attributes for Node 1.

Input Features	**Total Inputs**	**RMSE**	**MAE**
$PM_{2.5}$, PM_{10}, SO_2, NO_2, CO, O_3			
Temp, Pres, Dewp, Rain, Wd, Wspd	12	17.704	10.017
$PM_{2.5}$, PM_{10}, SO_2, NO_2, CO, O_3			
Temp, Pres, Dewp, Wd, Wspd	11	**17.363**	**9.807**
$PM_{2.5}$, PM_{10}, SO_2, NO_2, CO, O_3			
Temp, Dewp, Wd, Wspd	10	18.168	10.268
$PM_{2.5}$, PM_{10}, SO_2, NO_2, CO, O_3			
Dewp, Wd, Wspd	9	17.638	9.937

To calculate the RMSE and MAE values presented in Table 4.2, a simple LSTM network is initially employed as a baseline model before implementing the proposed hybrid CNN-LSTM model (refer to Section 4.4.1). The baseline model consisted of a one-layer LSTM with 15 neurons, which was selected as the predictor for our model. The autocorrelation coefficient among the lagged time series of $PM_{2.5}$ data is calculated to determine the appropriate lookback length for the input. A minimum requirement of 0.7 is set to ensure a high level of temporal correlation among the lagged data. As depicted in Figure 4.3, eight samples (including time lag = 0) are selected for the input model. At this time lag, all autocorrelation coefficients for all monitoring sites exceeded 0.7. Therefore, the current sample (time lag = 0) and the previous seven samples are utilised to predict a single sample in the future.

4.4 Deep Learning Model Architecture

4.4.1 Hybrid CNN-LSTM

In Section 4.3.2, an experiment using a simple LSTM model consisting of a single layer with 15 neurons is conducted. The purpose is to evaluate the model's performance based on input attributes and determine which ones should be included. Building upon this analysis, a hybrid model that combines one-dimensional convolutional neural networks (1D CNNs) is proposed as feature extractors, feeding the extracted features into an LSTM network. The architecture of the proposed hybrid CNN-LSTM model is illustrated in Figure 4.4. In this hybrid architecture, the CNN is employed to extract features from the input set, and the LSTM is utilised to make

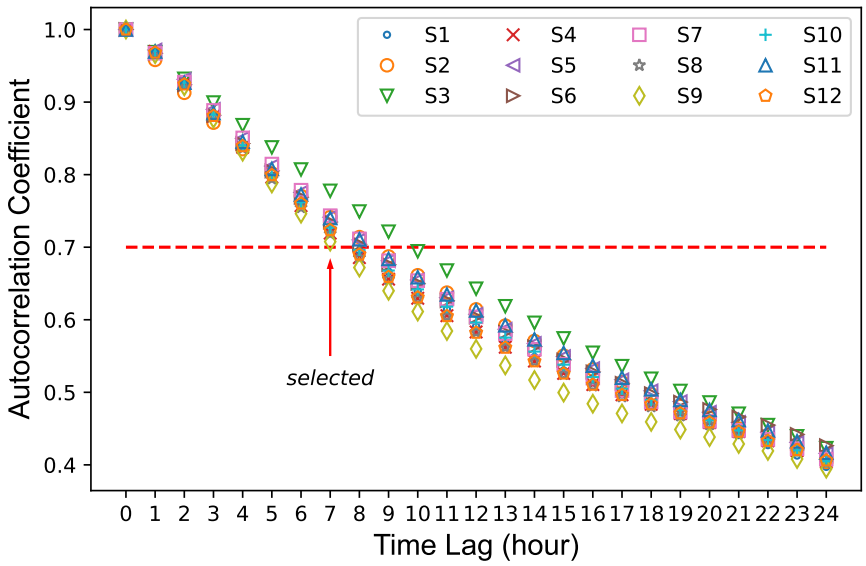


Figure 4.3: Autocorrelation coefficients for $PM_{2.5}$ concentration with different time lags.

predictions on the features extracted by the CNN. The CNN is adept at capturing the complexity of the input set, and the LSTM excels in forecasting time-series data.

The proposed model consists of two parallel inputs. The first input, denoted as INPUT-1, collects data solely from the local node for $PM_{2.5}$ prediction. The second input, denoted as INPUT-2, incorporates $PM_{2.5}$ data from both the local node and surrounding nodes. In this context, a local node refers to the node where $PM_{2.5}$ is being predicted. INPUT-1 comprises 11 features, including $PM_{2.5}$, PM_{10},

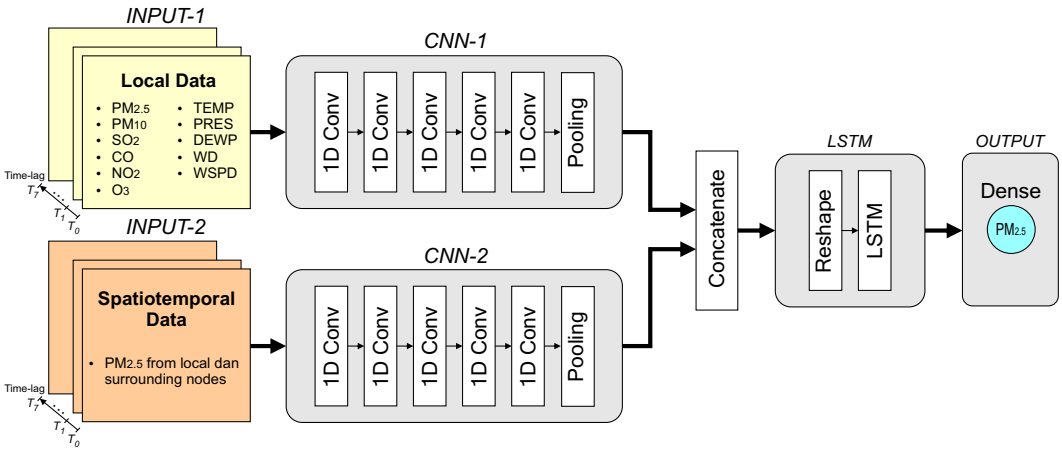


Figure 4.4: Proposed hybrid CNN-LSTM model.

Table 4.3: Hybrid CNN-LSTM network properties of the proposed model.

Layer	**Properties**
1^{st} Convolutional	filter = 50, kernel size = 3, activation = ReLU
2^{nd} Convolutional	filter = 30, kernel size = 3, activation = ReLU
3^{rd} Convolutional	filter = 15, kernel size = 2, activation = ReLU
4^{th} Convolutional	filter = 10, kernel size = 2, activation = ReLU
5^{th} Convolutional	filter = 5, kernel size = 2, activation = ReLU
Pooling	global average pooling
Reshape	reshape ((1,15))
LSTM	units = 15, activation = ReLU
Dense	units = 1

SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed. To forecast $PM_{2.5}$ for one hour into the future, eight timesteps (lookback) of these inputs are utilised. The batch of inputs is fed into the CNN network, which serves as a feature extractor before passing the information to the LSTM network.

Extensive experimentation led to the determination of the CNN network properties. Both CNN networks (block CNN-1 and CNN-2 in Figure 4.4) consist of five convolutional layers and a single average pooling layer. The reshape layer is employed to configure the outputs from the CNN layers before they are inputted into the LSTM network. The number of neurons remains consistent with the previous experiment (15 neurons) and employs the rectified linear unit (ReLU) activation function. The final prediction is achieved through a dense layer with one neuron. During the training process, the Adam optimiser is employed. A summary of each layer's properties can be found in Table 4.3.

To effectively extract features from a relatively short data length (in this case, eight samples), it is advisable to utilise smaller kernel sizes for deeper convolutional layers. The length of the subsequent convolutional layer can be calculated using Equation (2.2). Larger output sizes (o) can be obtained by employing small kernel sizes (k), which creates more opportunities for subsequent convolutional layers to operate. In this study, the kernel size of 3 is set for the first and second convolutional layers and the kernel size of 2 for the remaining three convolutional layers. These choices are determined through various experiments, and the selected kernel sizes and filters can be found in Table 4.3.

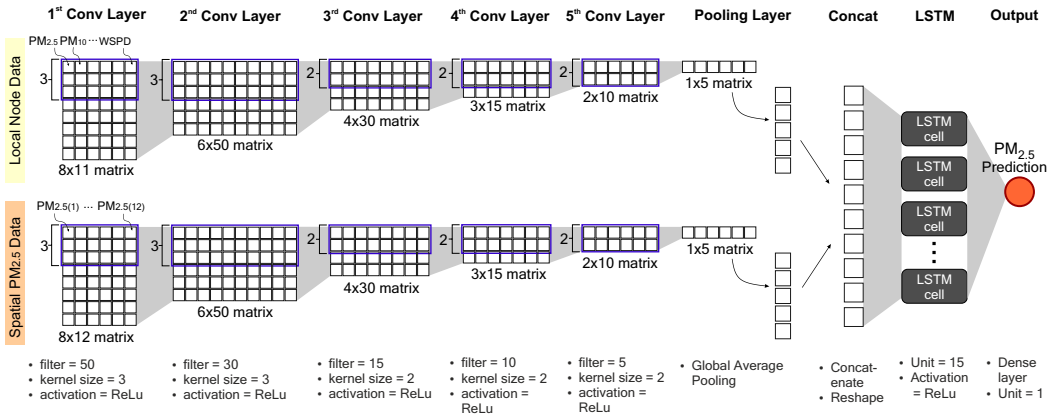


Figure 4.5: Details of data processing in the proposed deep learning model.

Opting for smaller filter sizes in each layer results in a smaller overall model size, which is beneficial for edge devices. Through experimentation, the filter sizes of 50, 30, 15, 10, and 5 for each convolutional layer are determined in our model, producing the best results. Maintaining the same properties for both CNN-1 and CNN-2 yielded optimal solutions as feature extractors, ensuring a balanced workload for each input during the training and inference stages.

In CNN-1, the eight timesteps of the 11 input features form an 8×11 matrix. These 11 features consist of pollutant and meteorological data, including $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed. In CNN-2, the eight timesteps of the 12 input features form an 8×12 matrix, with these 12 features representing the $PM_{2.5}$ concentrations at 12 different nodes.

By utilising Equation (2.2) with a kernel (or feature detector) size of 3 and a stride step of 1, the kernel slides through the input matrix in CNN-1 for a total of six steps ($(8-3)/1+1=6$). With a filter size of 50, the first convolutional layer generates a 6×50 matrix. The input for the second convolutional layer is a 6×50 matrix. Similarly, with a kernel size of 3, the kernel slides along the window for four steps ($(6-3)/1+1=4$) in the second layer, resulting in a 4×30 matrix (given the filter size of 30). The same process is applied to all subsequent convolutional layers. Consequently, the fifth convolutional layer produces a 1×5 matrix. A global average pooling layer is then employed to flatten the matrix. After concatenating the outputs of both CNN layers, the tensor is ready to enter the LSTM network. The LSTM network comprises 15 cells (or units), and a single dense layer generates the final prediction, i.e., our $PM_{2.5}$ prediction. The overall process is summarised in Figure 4.5.

Table 4.4: $PM_{2.5}$ coefficient correlation (r) for all nodes.

	Node1	**Node2**	**Node3**	**Node4**	**Node5**	**Node6**	**Node7**	**Node8**	**Node9**	**Node10**	**Node11**	**Node12**
Node1	1											
Node2	0.84	1										
Node3	0.83	0.90	1									
Node4	0.95	0.81	0.80	1								
Node5	0.96	0.83	0.83	0.97	1							
Node6	0.89	0.84	0.84	0.89	0.92	1						
Node7	0.83	0.84	0.85	0.82	0.84	0.85	1					
Node8	0.94	0.80	0.79	0.95	0.94	0.87	0.80	1				
Node9	0.88	0.80	0.81	0.88	0.88	0.85	0.89	0.87	1			
Node10	0.93	0.80	0.79	0.96	0.95	0.89	0.81	0.94	0.87	1		
Node11	0.93	0.86	0.85	0.93	0.95	0.93	0.84	0.91	0.87	0.92	1	
Node12	0.91	0.78	0.77	0.93	0.94	0.88	0.79	0.92	0.86	0.95	0.90	1

4.4.2 Spatiotemporal Model Inputs

This chapter examines both spatial and temporal aspects. The temporal factor is considered by incorporating time-lag data into the model. A time lag of zero corresponds to the current sample, while time lags less than 8 exhibit autocorrelation coefficients exceeding 0.7 for all nodes. These high autocorrelation values indicate strong temporal correlations. Therefore, eight values are utilised as the input length, involving the current measured and seven preceding values.

As discussed in Section 4.4.1, the model's first input (INPUT-1) focuses on capturing the temporal dependency of the local node data. It includes eight timesteps of attributes such as $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed. INPUT-1 primarily covers temporal data. On the other hand, the model's second input (INPUT-2) incorporates both temporal and spatial information by including data from the local node and the surrounding nodes. For INPUT-2, only eight timesteps of $PM_{2.5}$ from all nodes are considered, while other environmental and meteorological data are neglected. The spatial dependency can be assessed by analysing the $PM_{2.5}$ samples from all 12 nodes and calculating the $PM_{2.5}$ correlation coefficients between nodes.

Table 4.4 demonstrates that $PM_{2.5}$ concentrations exhibit strong correlations ($r > 0.7$) across nodes, indicating a significant spatial dependency. Therefore, the feature extraction process for the $PM_{2.5}$ concentrations from all neighbouring nodes (INPUT-2) is involved in this experiment. Feature extraction requires transforming raw data into a format optimised for the deep learning model. This not only simplifies the data by reducing the number of features to a manageable size but also

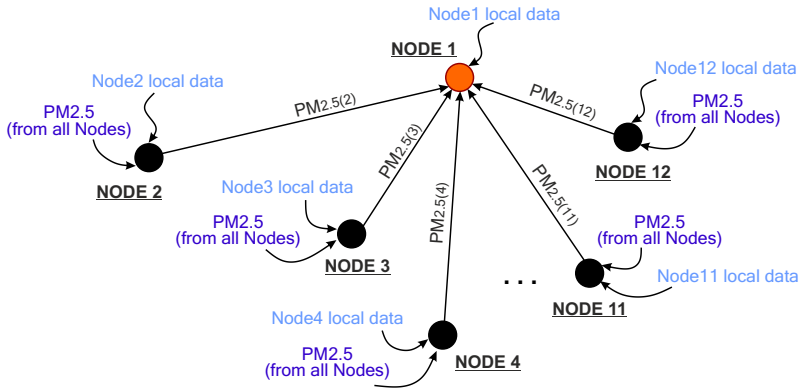


Figure 4.6: Illustration of spatiotemporal consideration for predicting the value of $PM_{2.5}$ concentration at Node 1.

preserves the most important aspects of the data. A key aspect of this process is pinpointing the data attributes that are most influential in achieving the desired results. Concentrating on these related features will improve the efficiency and precision of machine learning models, ensuring that the model is trained on data that offers the best insight into the current problem.

Figure 4.6 illustrates the types of input data required for predicting the $PM_{2.5}$ concentration at a specific node. To forecast the $PM_{2.5}$ concentration for the next 1 hour at Node 1, this works utilises the following data:

- The current pollutant and meteorological samples from Node 1.
- Seven previous pollutant and meteorological samples collected by Node 1 (forming the first input of the proposed model).
- $PM_{2.5}$ values from all other nodes (forming the second input of the proposed model).

This scenario applies not only to Node 1 but also to all other nodes, where the model's inputs are structured similarly to incorporate both temporal and spatial information for accurate $PM_{2.5}$ predictions.

4.5 Model Architecture Benchmark

The evaluation process in this section involves the following steps:

1. Twenty different deep learning models were developed and categorised into three groups.

2. The performance of all models was evaluated using RMSE and MAE values at all nodes.

3. The best-performing model was selected as the proposed model.

4. The TensorFlow file of the proposed model was converted into a TensorFlow Lite model, ensuring compatibility for edge devices.

5. TensorFlow version 2.2 was used for this work, and further optimisation was achieved by implementing post-training quantisation of the original TensorFlow model.

6. The performance of each TensorFlow Lite model was evaluated, considering factors such as model file size, execution time, and prediction accuracy.

Through these steps, the most efficient and accurate TensorFlow Lite model is identified for deployment on edge devices. The outcomes, including model file size, execution time, and prediction performance, are thoroughly reported for analysis and comparison.

Based on the pollutant and meteorological data from the current and the previous seven hours, a predictive model to forecast the short-term $PM_{2.5}$ concentration for the next one hour is developed. The models' performances are assessed using RMSE and MAE values, which are evaluated for all monitoring nodes. Table 4.5 summarises the RMSE and MAE scores obtained for all models, with Node 1 serving as a representative. The complete results for all nodes are provided in Tables B.1 and B.2.

The proposed model is compared against several other deep learning architectures, and the results demonstrated that our model outperformed the others. The comparison in Table 4.5 can be interpreted as follows:

- *Simple models with local data only* (Group I) utilise the input samples without involving any CNN layers. These models directly feed the inputs into RNN, LSTM, GRU, or Bidirectional layers, bypassing the convolutional, pooling, concatenation, and reshaping layers. The inputs used in this architecture consist of $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, as well as meteorological data including temperature, air pressure, dew point, wind direction, and wind speed.

- *Hybrid models with local data only* (Group II) incorporate CNN layers to process the input samples before passing them to the ANN, RNN, LSTM, GRU, or Bidirectional layers. These models are hybrid architectures that combine

CNN and other layer types. In this group, only INPUT-1 and CNN-1 layers are utilised, while INPUT-2 and CNN-2 layers are excluded. The properties of the CNN layers are specified in Table 4.3. For these models, the inputs consist of $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed. The neighbouring $PM_{2.5}$ samples are not considered in this configuration.

- *Hybrid models with spatiotemporal dependency* (Group III) utilise two inputs, namely INPUT-1 and INPUT-2, which are processed by CNN layers (CNN-1 and CNN-2) respectively. The first input captures the pollutant and meteorological data specific to the target node, while the second input consists of $PM_{2.5}$ samples from neighbouring nodes. Models in Group III follow the structure depicted in Figure 4.4, but the LSTM layer is varied with alternative layers such as ANN, RNN, GRU, or Bidirectional layers.

- The performance of the artificial neural network (ANN), recurrent neural network (RNN), long short-term memory (LSTM), and gated recurrent unit (GRU) models in all groups is evaluated and compared. To ensure fairness, all models in each group are configured with one hidden layer containing 15 neurons (units). The output layer consists of a dense layer with one neuron, responsible for producing the final prediction.

- *Bidirectional layers* extend conventional RNN, LSTM, and GRU models by processing the input sequence in two different directions. First, the input sequence is treated in the usual forward direction. Second, the input sequence is processed in the reverse direction. This approach provides additional context to the model and can lead to faster and more effective learning from the input sequence.

Table 4.5 presents a comparison of 20 different models, and the best models are shown in bold. It is observed that involving a deeper model with CNN layers as a feature extractor before the predictor (ANN, RNN, LSTM, or GRU) leads to slight improvements in model performance. In general, Group II outperforms Group I in performing prediction tasks. The overall accuracy can be further enhanced by including spatiotemporal considerations along with pollutant and meteorological data as inputs to the models. Notably, significant improvements can be achieved at certain nodes. At some nodes, the results can be improved significantly.

As previously stated, the training dataset spans from March 1, 2013, to March 20, 2016, whereas the testing dataset extends from March 21, 2016, to Febru-

Table 4.5: Comparison of RMSE and MAE values (in $\mu g/m^3$) for $PM_{2.5}$ prediction using different model architectures calculated for Node 1.

No.	Model Type	RMSE	MAE
	Simple models with local data only (Group I)		
1	RNN	18.485	10.636
2	**LSTM**	**17.786**	**10.230**
3	GRU	18.367	10.664
4	Bidirectional RNN	19.377	12.257
5	Bidirectional LSTM	18.016	10.427
6	Bidirectional GRU	18.603	10.944
	Hybrid models with local data only (Group II)		
7	CNN-ANN	17.757	10.321
8	CNN-RNN	18.227	10.906
9	CNN-LSTM	17.652	10.203
10	**CNN-GRU**	**17.244**	**9.552**
11	CNN-Bidirectional RNN	17.334	10.001
12	CNN-Bidirectional LSTM	17.344	10.054
13	CNN-Bidirectional GRU	17.462	10.486
	Hybrid models with spatiotemporal dependency (Group III)		
14	CNN-ANN	17.160	10.307
15	CNN-RNN	15.672	9.162
16	**CNN-LSTM (Proposed Model)**	**15.268**	**8.778**
17	CNN-GRU	17.169	9.665
18	CNN-Bidirectional RNN	17.365	10.443
19	CNN-Bidirectional LSTM	15.643	8.853
20	CNN-Bidirectional GRU	16.089	9.512

ary 28, 2017. This segmentation yields 26,784 samples for training and 8,280 samples for testing. According to the model evaluations, significant improvements are observed at certain nodes. For example, at Node 1, Groups I and II achieve RMSE values ranging from 17 $\mu g/m^3$ to 19 $\mu g/m^3$, while Group III yield RMSE values between 15 $\mu g/m^3$ and 17 $\mu g/m^3$. The proposed model, identified as model number 16 in Table 4.5, achieves the best RMSE value of 15.322 $\mu g/m^3$. The RMSE value of the proposed model surpasses that of all other investigated models. For instance, the Bidirectional RNN model in Group I resulted in an RMSE value of 19.377 $\mu g/m^3$, the CNN-LSTM model in Group II produced 17.652 $\mu g/m^3$, and the

CNN-ANN model in Group III returned an RMSE value of 17.160 $\mu g/m^3$.

As shown in Table 4.5, MAE is usually smaller than RMSE. MAE calculates the average absolute difference between predicted values and actual values. In contrast, Root Mean Square Error (RMSE) calculates the square root of the average of the squared differences. In RMSE, squaring errors amplify more significant errors, resulting in higher values than MAE. RMSE is more responsive to large errors because of this squaring process. Therefore, datasets with some large errors may have increased RMSE compared to MAE. Conversely, MAE treats all errors equally, offering a more consistent measure of error magnitude.

Further examination of other nodes in Tables B.1 and B.2 reveals that the $PM_{2.5}$ concentration at Node 11 can be more accurately forecasted not only by our proposed model but also by other investigated models. In contrast, predicting the $PM_{2.5}$ concentration at Node 12 proved to be the most challenging, as indicated by the higher RMSE and MAE values. Across all nodes, the proposed model consistently demonstrated superior performance, achieving error values between 14 and 18 for RMSE and between 7 and 9 for MAE.

Figure 4.7 illustrates the boxplot of prediction deviations for all models. These deviations are calculated by subtracting the actual values from the predicted values of the models on the test data. The boxplot provides valuable information about the variability of the data and is particularly useful for comparing distributions among multiple models. In Figure 4.7, the solid line in the middle of each box

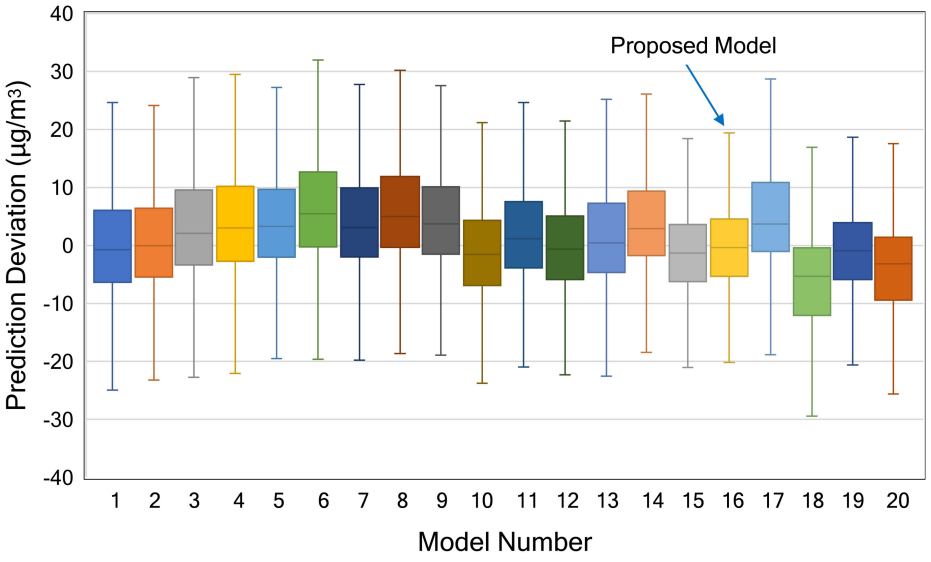


Figure 4.7: Boxplot of the prediction deviations at Node 1.

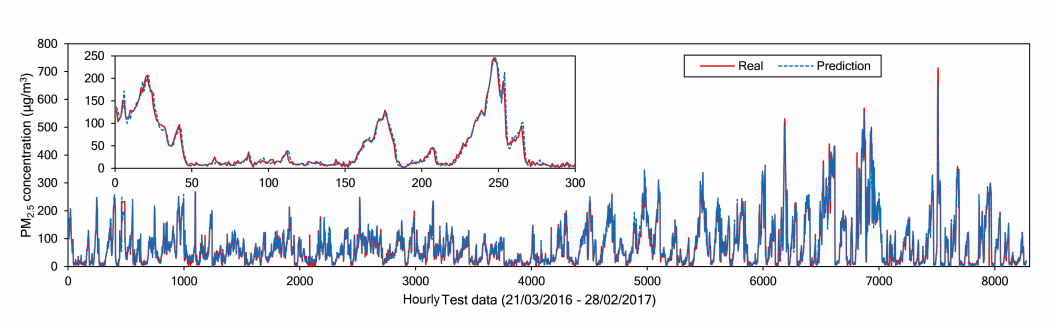


Figure 4.8: Line plot of real and predicted $PM_{2.5}$ data at Node 1.

represents the median value. Since this graph represents the prediction deviations between the predicted and actual data, a line close to zero is preferable. A shorter box and whisker plot indicate more centralised data, suggesting that the model accurately predicts the $PM_{2.5}$ data. To enhance readability, outlier values are excluded from the graph. As depicted in Figure 4.7, the proposed model outperforms the others, generating more centralised data with a median value closest to zero.

To describe model performance more intuitively, Figure 4.8 shows a line plot between the real and predicted values on the test data at Node 1. The solid and dashed lines indicate the real and predicted values, respectively. There are 8280 samples collected from 21 March 2016 to 28 February 2017. Overall, the model can capture the fluctuations of future $PM_{2.5}$ values effectively, as shown in Figure 4.8. The larger errors usually happen when there are spikes in the actual data, whereas our model forecasts successfully for smoother $PM_{2.5}$ data variations.

Figure 4.9 displays scatter plots from all nodes, illustrating the relationship between real and predicted values. The ideal scenario would align all points perfectly along the solid diagonal line. However, the points deviate from this diagonal line due to prediction errors. Points below the ideal line indicate predictions lower than the actual values, while points above the line indicate overestimations. For instance, at Node 3, a higher frequency of deviations below the ideal line is observed. The model predicted a value of 103.92 $\mu g/m^3$, while the actual value is 414 $\mu g/m^3$. Similar deviations occur at Node 7, where the model predicted 162.36 $\mu g/m^3$ instead of the actual value of 556 $\mu g/m^3$. Some of these mispredictions may be attributed to measurement errors, characterised by sudden changes in the sequence of measured samples that are not technically feasible. Figure 4.9 highlights a significant prediction error for $PM_{2.5}$ data at Node 12. The model predicted a value of 554.24 $\mu g/m^3$, whereas the actual measured value is only 3 $\mu g/m^3$ for the corresponding labeled point. Upon closer examination of the dataset for Node 12, it reveals a sharp drop

in the measured value from 621 μg/m^3 to 3 μg/m^3, followed by a sudden jump to 144 μg/m^3. The LSTM network failed to identify these abrupt changes, leading to a significant prediction error at this particular point.

A sudden spike in measurements might indicate a glitch. In practical scenarios, such as in the UK, the Department for Environment, Food and Rural Affairs (Defra) oversees the data integrity for the UK Automatic Urban and Rural Monitoring Network (AURN) through a comprehensive system of data reviews and updates [216]. This includes both automatic and manual processes. AURN's hourly mean monitoring data is uploaded every hour as provisional information. This data is initially screened to eliminate any obviously incorrect data as much as possible. The process involves two key stages: Data Validation and Data Ratification. Data Validation is an ongoing activity, essentially refining the initial provisional data. This stage includes additional manual data review to remove any results from instrument malfunctions or incorrect calibrations, including any data initially missing due to communication issues with monitoring stations, and updates to data scaling based on the latest calibration factors. Data Ratification, or verification, involves

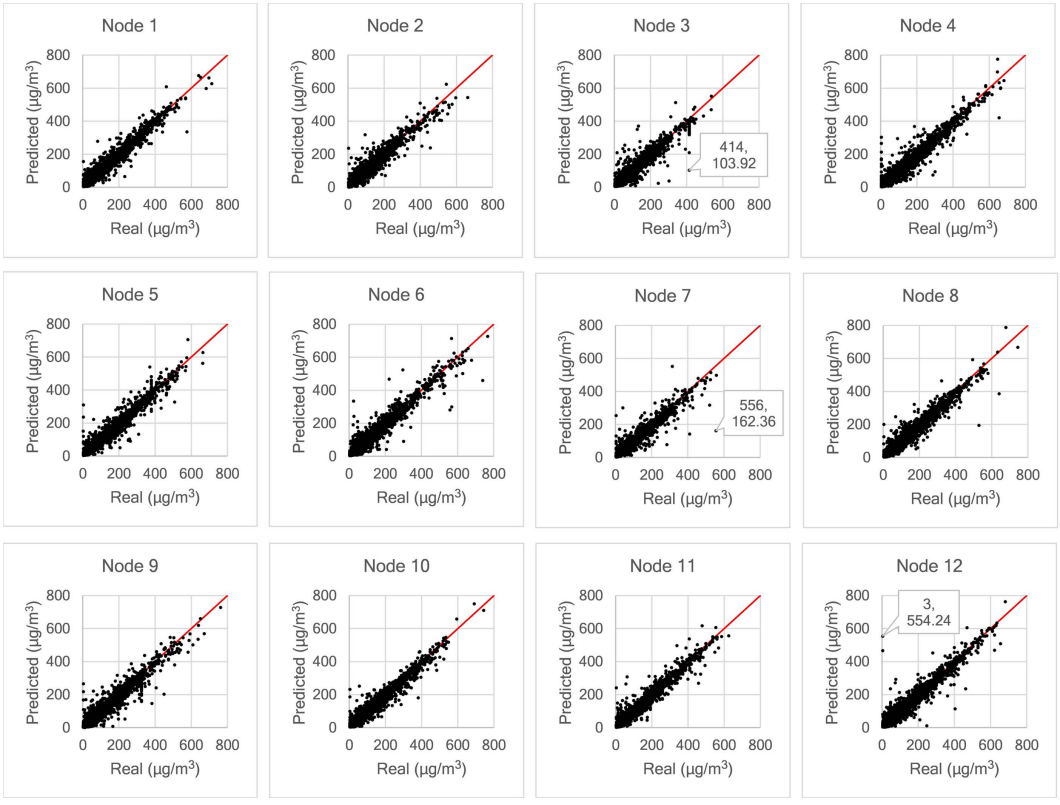


Figure 4.9: Scatter plots of real and model predicted values of $PM_{2.5}$ at all nodes.

a thorough manual examination of the dataset conducted quarterly for the AURN. This stage takes a longer-term view of the dataset, incorporating findings from independent QA/QC audits of the monitoring stations, ensuring the long-term accuracy and reliability of the data.

4.6 Model Optimisation for the Edge

4.6.1 Edge Devices

After evaluating the proposed deep learning model, the next step is to optimise and deploy it to edge devices. For this purpose, Raspberry Pi boards are selected. The Raspberry Pi board is a popular, credit card-sized single-board computer developed by the Raspberry Pi Foundation. Over the years, Raspberry Pi boards have seen numerous applications [217], making them a suitable choice for our project. Two different Raspberry Pi boards are selected for comparison: the Raspberry Pi 3 Model B+ (RPi3B+) and the Raspberry Pi 4 Model B (RPi4B), to observe variations in model performance. The RPi4B possesses higher computational capabilities compared to the RPi3B+, which adds an interesting dimension to our analysis.

The selection of Raspberry Pis is driven by their compatibility with TensorFlow and TensorFlow Lite frameworks. This allows the users to leverage a wide range of functionalities, including post-training quantisation offered by TensorFlow. Through this, the performance of both the original and quantised models is demonstrated by calculating model accuracy, file sizes, and execution times directly at the edge. Another significant advantage of using Raspberry Pi boards is their popularity within the research and hobbyist communities. This popularity has given rise to numerous online forums and communities dedicated to Raspberry Pi development, providing a wealth of resources and support for the experiments.

4.6.2 Lite Models

After the final model has been trained, the next step involves deploying it to the edge after optimisation. This optimisation process brings benefits in terms of file size and computation latency. The initial model created is the TensorFlow model (TF model). The TF model can be converted into TensorFlow Lite (TFLite) model. TFLite is a lightweight model designed specifically for edge devices. The TFLite model can be deployed with or without optimisation, allowing the users to explore and compare both possibilities.

Table 4.6 summarises the file size comparison between the TF and TFLite

Table 4.6: TensorFlow and TensorFlow Lite file size comparison.

Properties	TF Model	TFLite Model
File size (kB)	318	77

models. At this stage, the TFLite model has not yet been optimised. The original file size is 318 kilobytes, while the lite version is 77 kilobytes, making it four times smaller. This reduction in file size proves to be crucial for resource-constrained edge devices, particularly those with limited storage capabilities.

4.6.3 Post-training Optimisations

Post-training quantisation techniques are explored to achieve further reductions in size and speed. Figure 4.10 presents four optimisation techniques in the TensorFlow framework version 2.2. These techniques include dynamic range quantisation, full integer quantisation with float fallback, integer-only quantisation, and float16 quantisation. The impact of these techniques are demonstrated in Figure 4.10 and Figure 4.11.

Without any optimisation/quantisation, the TFLite model has a size of 77 kilobytes, serving as the reference. By applying dynamic range quantisation, a

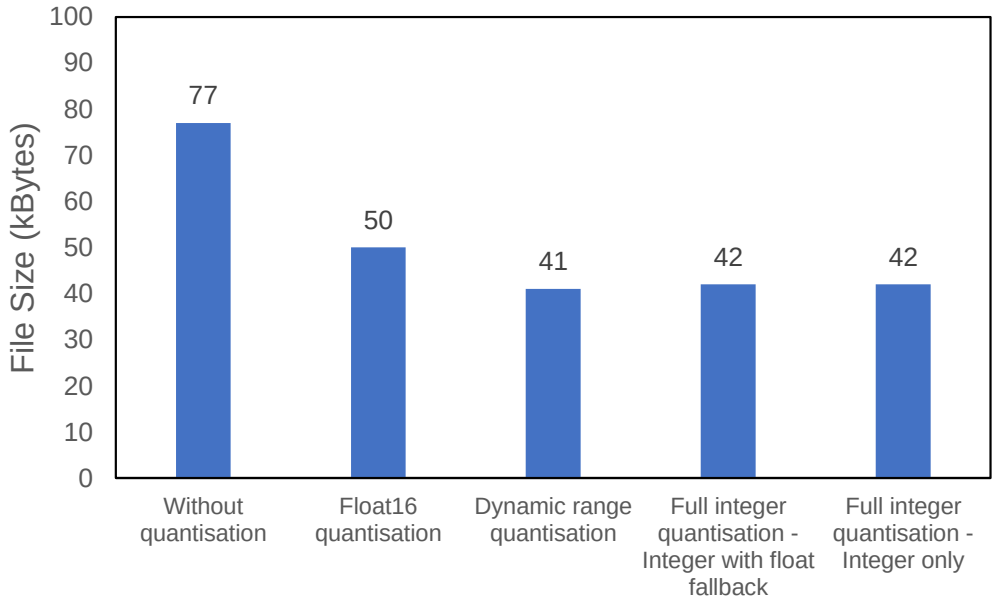


Figure 4.10: TensorFlow Lite model size comparison.

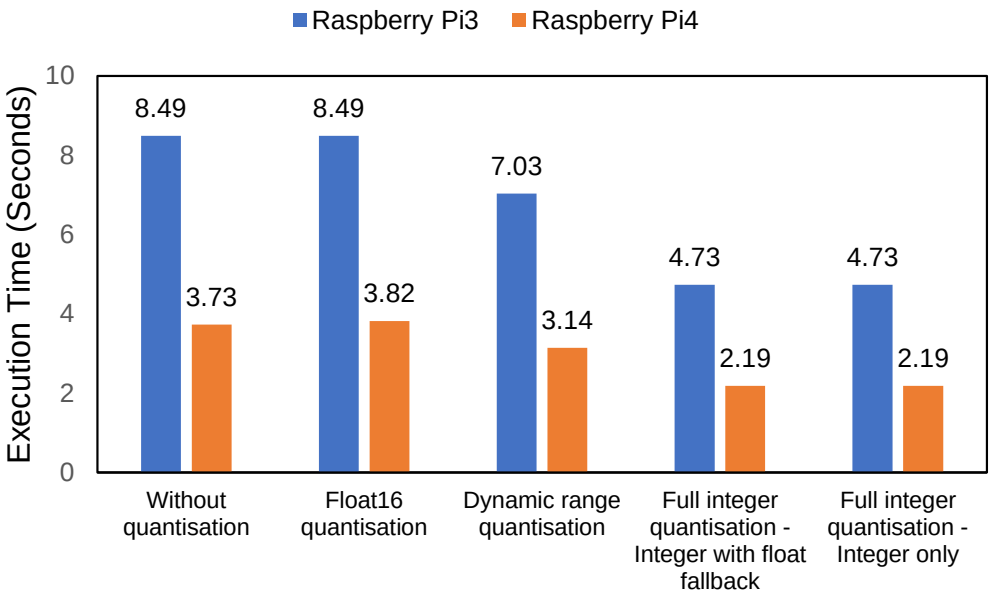


Figure 4.11: Comparison of TensorFlow Lite execution time for test data.

reduction of approximately 47% in size can be achieved. Full integer quantisation offers a reduction of about 45%, while float16 quantisation provides a reduction of around 35%. Among these techniques, dynamic range quantisation outperforms the others, although it only marginally surpasses full-integer quantisation in terms of size reduction.

The time required for edge devices to predict the available test data is measured in this study. The experiment involved the continuous execution of a total of 8272 hourly samples (data from 21 March 2016 to 28 February 2017) directly at the edge. The experiment results are presented in Figure 4.11. As depicted in the figure, the RPi4B board demonstrates a considerable advantage over the RPi3B+ board in all quantisation modes, being two times faster. This performance difference highlights the superior computational capabilities of the RPi4B, making it a more efficient choice for executing the prediction tasks at the edge.

The introduction of Float16 quantisation does not lead to an improvement in execution time as the latency remains unchanged, possibly due to the fixed 32-bit floating-point datapath on these devices. Specifically, for the RPi3B+ board, it takes 8.49 seconds to execute the complete test, while the RPi4B board exhibits a minimal difference of 0.07 seconds (3.75 and 3.82 seconds, respectively).

Although dynamic range quantisation enables a size reduction of approximately 47%, it offers minimal improvement in execution time. In this mode, the

execution time is 7.03 seconds for the RPi3B+ and 3.14 seconds for the RPi4B. On the other hand, full integer quantisation demonstrates the most effective improvement in execution time, with latencies of 4.73 seconds for the RPi3B+ and 2.19 seconds for the RPi4B.

In addition to considering the model size and execution time, it is crucial to evaluate model accuracy after applying quantisation. The details of the RMSE and MAE scores for the initial TensorFlow and TensorFlow Lite models can be found in Table C.1 in Appendix C. Despite the minimal deviation in results between the optimised models, this section presents the model performance more intuitively through a boxplot, as depicted in Figure 4.12. This figure provides insights into the prediction deviation between the TFModel and TFLite Model results.

The boxplot shows that TFLite without quantisation and TFLite with float16 quantisation exhibit similar accuracies, with very slight deviations from the original TF model. TFLite with dynamic range quantisation displays a slightly wider deviation range. On the other hand, both TFLite integer quantisations exhibit the widest box and whisker ranges, indicating that these quantisation methods are less effective in prediction accuracy compared to other post-quantisation techniques.

TFLite without quantisation proves to be a suitable technique when prioritis-

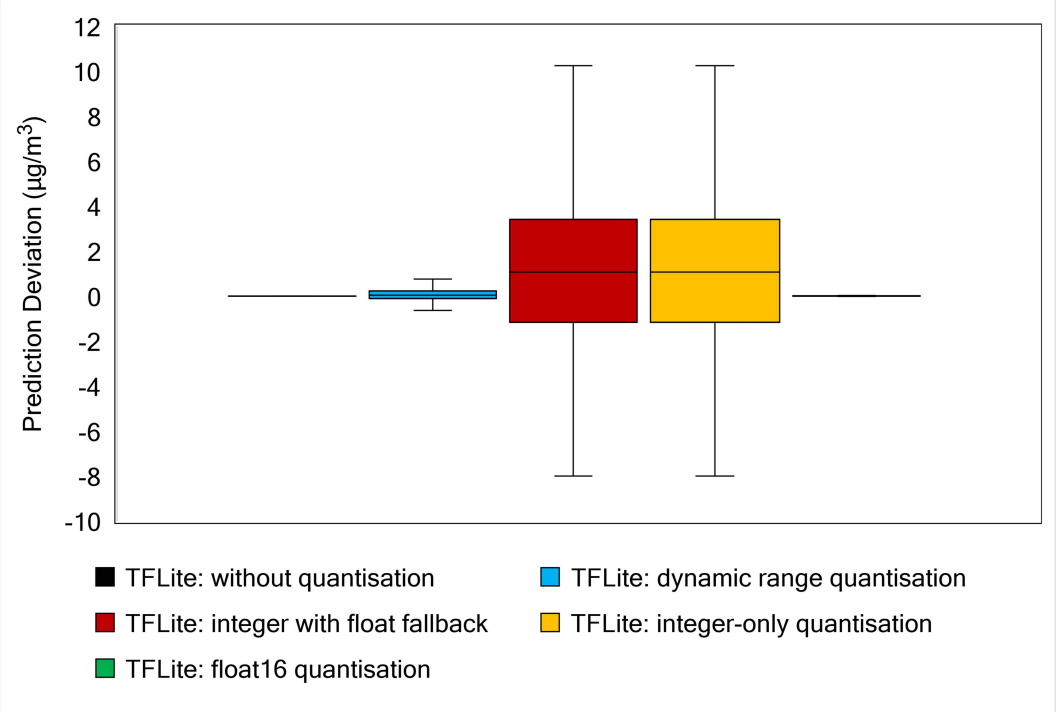


Figure 4.12: Boxplot of prediction deviation resulted from each TFLite model.

ing model accuracy. However, it may not be optimal for size reduction and execution time improvement. Both dynamic range and float16 quantisations effectively maintain model accuracy while offering different benefits. Dynamic range quantisation performs better in terms of model size reduction and execution time compared to float16 quantisation. Full integer quantisations outperform other TFLite models concerning model size and latency improvements. However, these methods do exhibit a slight reduction in model accuracy. Therefore, the choice of quantisation technique depends on the specific priorities and trade-offs the users aim to make between model accuracy, size reduction, and execution time improvement.

To visually explore the relationship between TensorFlow Lite models and their initial TensorFlow counterpart, scatter plots can be used for comparison, as illustrated in Figure 4.13. While the figure shows results for Node 1, it is important to note that the same behaviour is observed for all nodes. The scatter plots demonstrate that the results obtained by TFLite without quantisation, dynamic range quantisation, and float16 quantisation closely align with those predicted by the initial TensorFlow model, as indicated by the smooth straight-line pattern. The same effect can be observed in Figure 4.13. However, larger deviations are noticeable for integer quantisation models, both integers with fallback and full integer

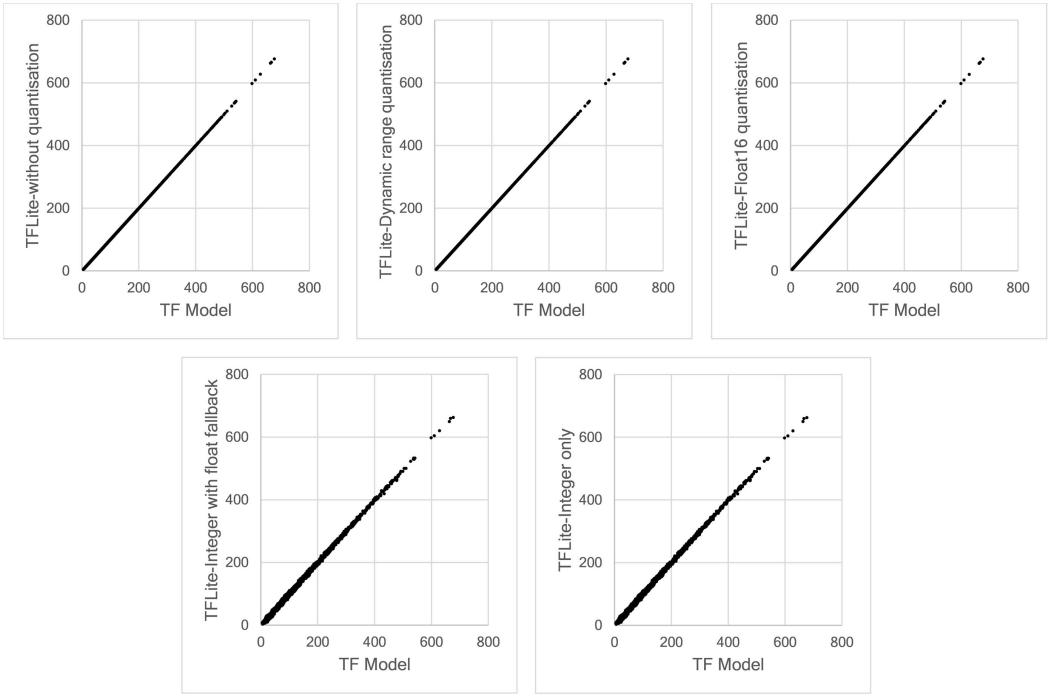


Figure 4.13: Scatter plot of the prediction data obtained by TensorFlow and TensorFlow Lite models.

quantisations. The straight-line pattern appears more scattered, concluding that full integer quantisation slightly impacts model accuracy.

4.7 Summary

Edge computing addresses latency, privacy, and scalability issues by bringing computation closer to data sources. It also enables embedding intelligence at the edge, which can be achieved by utilising Machine Learning (ML) algorithms. Specifically, Deep Learning, a subset of ML, can be effectively implemented at the edge. This study proposes a hybrid deep learning model consisting of 1D Convolutional and Long Short-Term Memory (CNN-LSTM). The model aims to predict short-term hourly $PM_{2.5}$ concentration at 12 different nodes. The results of our proposed model outperform those of other deep learning models in terms of performance, as demonstrated by the calculation of RMSE and MAE errors at each node.

Implementing a parallel structure in CNN layers effectively enhances the performance of deep learning models. One layer can focus on processing local data, while another is dedicated to incorporating spatiotemporal data from neighbouring sites. In the top layers, LSTM layers are effectively utilised to predict future time-series data. To deploy the selected model on edge devices, single-board computers are chosen. In this chapter, Raspberry Pi boards have successfully executed both the original TF models and TF models with post-training quantisation.

This chapter evaluated four different post-training quantisation techniques provided by the TensorFlow Lite framework to implement an efficient model for edge devices. These techniques include dynamic range quantisation, float16 quantisation, integer with float fallback quantisation, and full integer-only quantisation. While the dynamic range and float16 quantisation techniques maintain model accuracy, these approaches did not significantly improve latency. On the other hand, the full integer quantisation technique outperformed other TensorFlow Lite models in terms of model size and latency, even with a slight reduction in model accuracy. This chapter focuses on the Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B boards as the targeted edge devices. Technically, the Raspberry Pi 4 demonstrated lower latency due to its more capable processor.

When employing quantisation, a trade-off exists among accuracy, file size, and execution time. Based on the conducted experiments, implementing dynamic range quantisation proves to be a win-win solution. By dynamic range quantisation, the file size of the TFLite model can be reduced compared to the original model while maintaining almost the same level of accuracy. Moreover, although the execution

time is slightly reduced compared to the original model, this negligible difference is not considered significant.

Chapter 5

Collaborative Edge Learning

This chapter is based on the following submitted manuscript:

- I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Collaborative Learning at the Edge for Air Pollution Prediction," *IEEE Transactions on Instrumentation & Measurement*, vol. 34, pp.1-12, Dec. 2023 [3].

5.1 Introduction

The exponential growth of data generation in recent years has spurred significant interest in collaborative learning to address large-scale machine learning problems [218]. Numerous studies have explored this topic from diverse perspectives, contributing valuable insights to the field. For instance, Henna *et al.* [219] leveraged graph neural networks (GNN) to exploit topological dependencies among mobile edges, facilitating efficient inference. Their collaborative GNN-edge approach partitions the GNN based on latency requirements, resulting in improved prediction performance compared to cloud-based methods. The proposed techniques enhance spectrum utilisation, throughput, and response times. Published work by Song and Chai [220] presents a collaborative learning framework tailored for multi-class classification problems. The framework comprises three key components: generating the training graph, defining the learning objective, and optimizing the classifiers. Through collaborative learning of deep neural networks, their method effectively reduces generalisation errors and enhances robustness against incorrect labels or data augmentation.

The conventional approach to implementing collaborative learning involves utilising a model averaging method, commonly employed in the federated learning (FL) paradigm [221]. This method enables collaborative learning across multiple

end devices. Unlike centralised learning, FL is a distributed learning technique that allows training a global model by collaborating edge devices without sharing sensitive training data [222]. With FL, edge devices can train a shared global model locally, leveraging their own data resources. However, to overcome challenges such as poor convergence on heterogeneous data and the limited generalisation ability of the global model with respect to local distributions, personalised federated learning (PFL) has emerged as a viable solution [223]. PFL aims to tailor the training process to individual devices, ensuring improved performance and adaptation to local variations.

Gholizadeh and Musilek proposed a novel approach called federated learning with hyperparameter-based clustering [224]. The authors applied this method to predict individual and aggregate electrical loads. In their work, the central server adapts a specific neural network model based on the hyperparameters obtained for each cluster, considering factors such as accuracy, computational cost, or the trade-off between the two. The results demonstrated that the proposed clustering method can significantly reduce the convergence time of federated learning. Another approach for implementing distributed learning through model averaging was introduced by Mi *et al.* [225]. They achieved effective distributed learning by incorporating a cyclical learning rate and increasing the local training epochs. The proposed method was validated across various classification tasks involving images, text, and audio. The federated learning approach has also found applications in diverse domains such as cybersecurity for IoT [226], healthcare [227], and manufacturing [228]. In the context of air quality studies, federated learning can be leveraged to facilitate knowledge sharing among multiple monitoring stations.

While many existing studies predominantly rely on computer simulations for predicting $PM_{2.5}$, this chapter takes a different approach by directly implementing collaborative learning strategies at the edge. This involves offloading computation tasks closer to the data source. Additionally, the current body of collaborative learning research has not specifically addressed the domain of air quality prediction. This study aims to bridge this gap by introducing a collaborative learning framework tailored for air pollution prediction.

The primary advantage of employing collaborative learning lies in its ability to enhance prediction accuracy compared to individual learning methods. This approach closely mimics real-life scenarios where numerous monitoring devices are deployed across multiple locations, eliminating the dependency on cloud-assisted services. This approach considers the availability of air pollution data from multiple observation stations, the spatial and temporal correlations between these stations

that influence air quality status, and the potential for improving air quality prediction through collaborative learning. This chapter aims to achieve more accurate and robust air quality predictions by leveraging the collective knowledge from diverse monitoring stations.

5.2 Rapid Expansion of Sensing Devices

A novel approach has recently been suggested to monitor air quality status using an extensive network of low-cost sensor nodes [229, 230, 231, 232]. This innovative paradigm aims to gather spatiotemporal air pollution data by employing numerous sensing devices, complementing traditional methods that rely on more expensive and less accurate instrumentation [53]. Moreover, the data collected from these sensing nodes are often transmitted over the Internet, enabling remote air quality monitoring. Consequently, due to the rapid growth of these sensing devices, there has been a substantial increase in the volume of data generated, stored, and transmitted [55]. The concern arises as the number of Internet-connected devices increases in the future. Furthermore, it is not just air quality monitoring devices that are linked to the internet; a myriad of other devices, including personal mobile phones, gadgets, personal and office computers, and countless other IoT devices, will join the network.

Due to the extensive data collection, Machine Learning (ML) methods have gained significant positions across various domains, including the prediction of air quality tasks [56]. Deep Learning (DL), a subset of ML, presents a compelling approach to forecasting air quality status by effectively capturing spatial and temporal features. In deep learning models, many layers with many neurons collaborate synergistically to process vast input data. Each layer performs specific operations on the input data, gradually extracting increasingly abstract and meaningful patterns. This hierarchical representation allows the model to distinguish complex features and relationships in the data, ultimately enabling accurate inference and prediction. As information flows through the network, each layer refines the data representation, capturing both low-level details and high-level concepts.

The standard method of centralised learning involves gathering all data from end devices and storing them in a data centre, often implementing a cloud-centric approach to train and evaluate deep learning models. However, the rise in connected devices is causing network congestion, which forces edge devices to handle more data [233]. Optimising a resource at the edge of a network is known as edge computing [234]. Edge computing involves processing data close to its source [224].

The edge computing environment allows computing tasks to be offloaded from the centralised cloud to near-sensing devices, reducing the amount of data transferred by performing preprocessing operations at the edge [235]. For a wide range of big data applications, edge computing excels in terms of memory cost, energy consumption, and latency.

Spatial and temporal correlations exist between air quality data collected from different air monitoring stations [178]. This implies that the air pollution level at a given station is influenced by air pollution concentrations at nearby stations (spatial relation) and its own historical data (temporal relation). While air quality may appear stable during certain periods, it is subject to fluctuations. It requires continuous monitoring to assess its variability accurately. Adjacent areas also often exhibit a similar variation in spatial domains [236]. Understanding spatial and temporal dependencies is essential [237], and by comprehending these dependencies, we can make more informed decisions regarding air quality status. Understanding spatial and temporal dependencies helps us understand air pollution dynamics better. This knowledge helps us find pollution hotspots, predict trends, and create better strategies to reduce pollution and protect public health. Policymakers, planners, institutions and communities can benefit from this understanding to make decisions and take action to improve air quality. Comprehending spatiotemporal factors is critical to developing an effective air quality management system that supports environmental sustainability and community welfare. Moreover, during the development of machine learning models, collaborative learning can be used to incorporate spatiotemporal data. Collaborative learning among sensing devices using spatiotemporal data may improve deep learning model performance [236].

This chapter investigates the practical application of collaborative learning for air quality prediction on edge devices. By utilising multiple air quality monitoring stations and considering the spatiotemporal features of air pollutant data, this chapter introduces methodologies for collaborative deep learning model sharing and collaborative measurement data sharing. Additionally, a commonly used model averaging technique is incorporated into the framework. Some aspects, including training losses, method accuracies, and communication costs associated with each approach, are discussed to assess the effectiveness of the proposed collaborative learning methods.

5.3 Chapter Contributions

This chapter considers the availability of air pollution data from multiple observation stations, the spatial and temporal correlation between stations that affects air quality status, and the possibility of improving air quality prediction through collaborative learning. This work investigates three collaborative learning strategies and applies them directly to edge devices by performing on-device training and inferencing. Specifically, our contributions are as follows:

- Introducing novel approaches leveraging spatiotemporal data (called SpaTemp), deep learning model sharing (called ClustME), and the widely adopted collaborative learning method (i.e., federated learning or FedAvg) to enhance air quality prediction.
- Designing algorithms to enable collaborative learning on edge devices, utilising the MQTT communication protocol for seamless operation.
- Examining key aspects of the proposed collaborative learning strategies, including training losses, learning accuracy, learning period, and communication costs.
- Expanding the scope of this work by providing valuable insights into the potential expansion of edge device networks.

5.4 Proposed Framework

The research framework is illustrated in Fig. 5.1, presenting an overview of the key components and workflow of the study. Initially, a subset of air quality monitoring stations is selected from the available dataset, considering the availability of devices. Next, the dataset is divided into three partitions: 70% for training, 20% for validation, and 10% for testing purposes.

Before proceeding with the subsequent steps, a feature selection process is performed to identify the optimal number of input variables. The preprocessed datasets are then utilised in collaborative learning techniques implemented directly at the edge using the MQTT protocol. In this study, three collaborative learning methods are proposed: FedAvg, ClustME, and SpaTemp. These methods are applied to facilitate collaborative learning and enhance the prediction accuracy of the air quality model.

In addition to the collaborative learning approach, the performance of the locally-trained model is also evaluated. To assess the effectiveness of both approaches, several evaluation metrics, including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), coefficient of determination (R^2), and rate of improvement on RMSE (RIR) are employed. These metrics provided a comprehensive evaluation of the model's predictive accuracy.

Apart from evaluating the overall performance on the test data, this chapter specifically assesses the edge performance. This evaluation focuses on the time required for each device to complete the training session. By considering the computational efficiency of the edge devices, this chapter gains insights into the practicality and effectiveness of implementing the collaborative learning approach directly at the edge. Finally, this chapter explores the potential for scaling edge device networks.

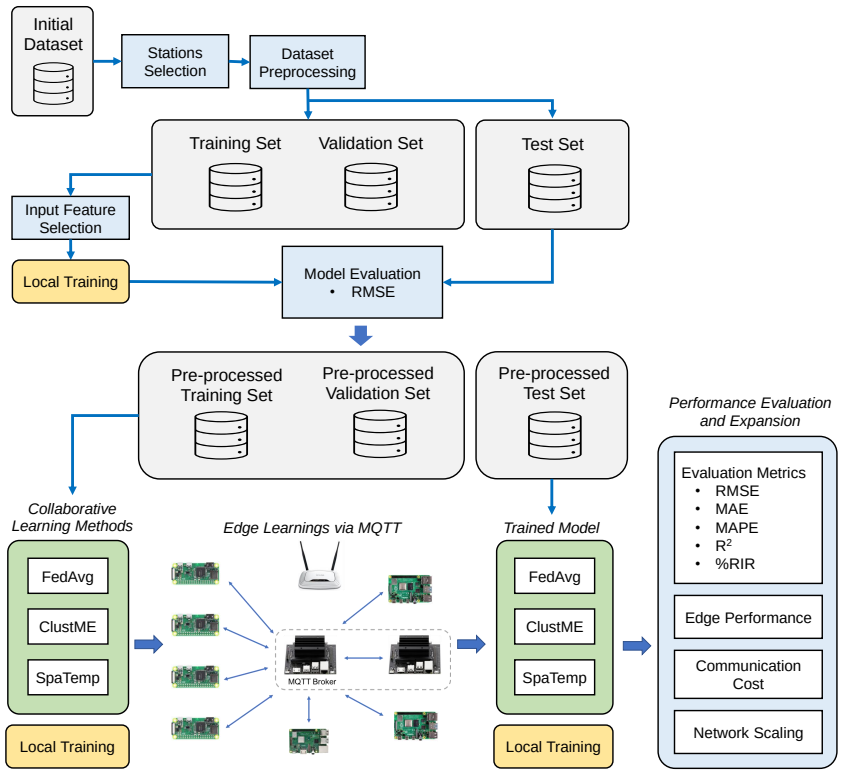


Figure 5.1: Proposed framework for collaborative learning at the edge.

5.5 Data Preprocessing

The dataset used in this chapter is similar to the one utilised in Chapter 4. However, considering the device availability during experimentation, this work specifically focuses on eight monitoring stations. Regardless of the actual geographical locations and the correlations among the stations, the stations were organised alphabetically based on their names. The first eight stations were selected for this study. The chosen stations are Aotizhongxin, Changping, Dingling, Dongsi, Guanyuan, Gucheng, Huairou, and Nongzhanguan. In this chapter, these selected stations were labelled as Station-01, Station-02, Station-03, and so forth, up to Station-08. This naming procedure for the stations aims to simplify further analysis, such as labelling table components and figures. This naming convention is unrelated to the geographical locations of the stations. Additionally, eight edge devices were employed to represent each monitoring station. The locations of all air quality monitoring sites can be shown in Fig.5.2.

Each monitoring station's data consists of 12 columns and 36,064 rows, spanning from 1 March 2013 to 28 February 2017. The recorded data is hourly, encompassing various pollutant measurements (such as $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, and O_3) as well as meteorological data (temperature, air pressure, dew point, rain, wind direction, and wind speed). Missing values in the dataset were handled by filling them with the most recently available data at the corresponding timestamp. This approach ensured that the dataset remained complete for further analysis. Additionally, to facilitate model training, all attributes were normalised using a min-max

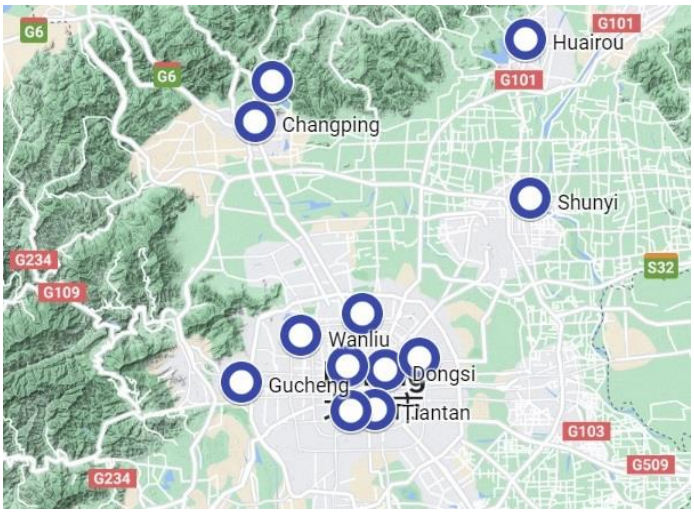


Figure 5.2: Map of air quality monitoring stations in Beijing and its surroundings.

scaler. This scaling technique transformed the attribute values to a standardised range between 0 and 1, preserving the relative relationships between the data points while accommodating different scales across attributes.

The dataset was divided into three subsets for training, validation, and testing purposes. The training data consisted of records from 1 March 2013 to 18 December 2015, accounting for approximately 70% of the dataset. The validation data spanned from 19 December 2015 to 5 October 2016, constituting around 20% of the dataset. Finally, the test data encompassed the period from 6 October 2016 to 28 February 2017, comprising approximately 10% of the dataset. In this study, the focus was specifically on predicting $PM_{2.5}$ concentrations. The primary objective was to evaluate the best model for short-term prediction, specifically forecasting the $PM_{2.5}$ concentration for the next one hour.

5.6 Collaborative Strategies

This work implements MQTT for communication protocol and federated learning (FL) for collaborative edge device learning. The terms *client* and *server* are commonly used in both MQTT protocol and federated learning (FL) algorithm. To avoid confusion, this chapter refers MQTT server as *broker* and FL server as *coordinator*. Both MQTT and FL clients are referred to as *edge devices* or *stations*.

5.6.1 Learning Overview

Three collaborative learning strategies are implemented, as depicted in Fig. 5.3. These methods are:

1. ***FedAvg***: Federated learning with a federated averaging method.
2. ***ClustME***: Learning with clustered cyclic peer-to-peer model exchanges, incorporating personalised learning through station clustering before local training and model exchanges.
3. ***SpaTemp***: Learning with spatiotemporal data exchanges, where the transferred data among edge devices includes measurement data such as pollutant data, temperature, humidity, etc.

Each strategy offers a distinct approach to performing air quality prediction. In FedAvg and ClustME, the transferred data among edge devices are the deep learning models. As a result, no measurement data, such as pollutant data,

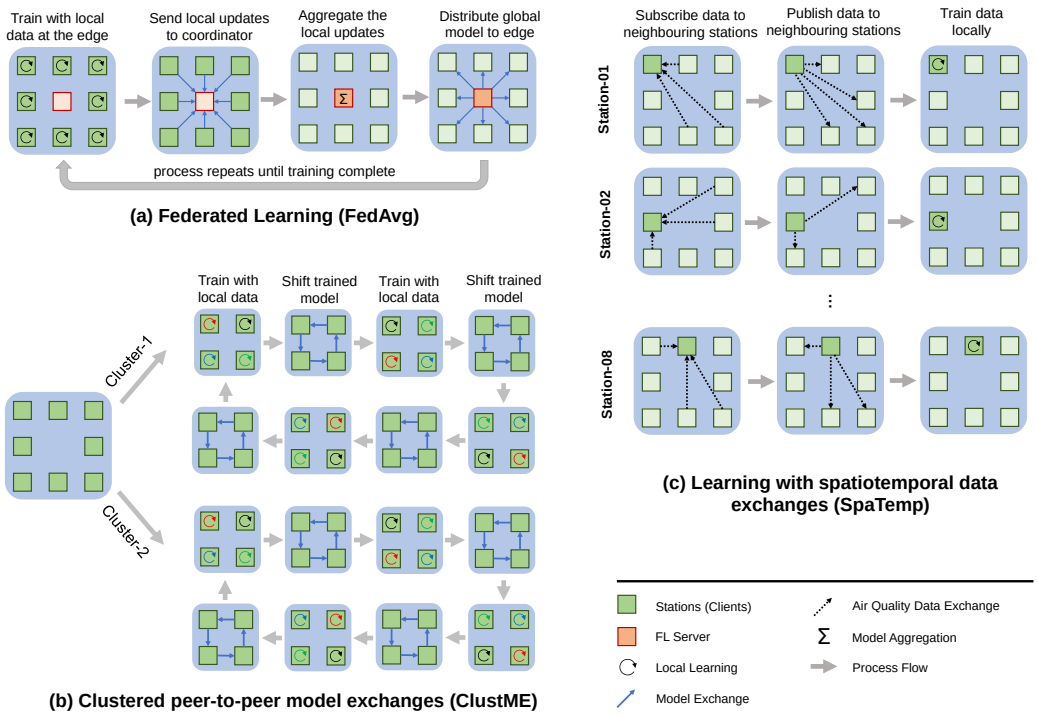


Figure 5.3: Implemented collaborated learning strategies.

temperature, humidity, etc., are exposed to other devices. These methods focus on exchanging and aggregating model parameters to improve overall learning accuracy.

In contrast, SpaTemp incorporates the exchange of measurement data. This strategy facilitates the exchange of spatiotemporal information among edge devices to enhance the learning process. Unlike FedAvg and ClustME, SpaTemp involves sharing measurement data between devices. Furthermore, ClustME incorporates personalised learning by leveraging station clustering. Before local training and model exchanges, stations are grouped into clusters. This personalised learning approach allows each cluster to tailor its training based on local characteristics and subsequently exchange models within the cluster.

The FedAvg collaborative learning strategy consists of four steps, as depicted in Fig.5.3(a). These steps are:

1. **Model initialisation**: Each edge device (station) initialises its model directly on the device.

2. **Local model updates**: Each station performs model updates by training its model using its local data. The training is conducted independently on each device. After completing the local training, each station sends its locally updated model to the coordinator.

3. **Model aggregation**: The coordinator collects all the updated models from the stations. The coordinator performs server aggregation by aggregating the received models from the stations.

4. **Global model distribution**: Once the aggregation is done, the coordinator transmits the aggregated model back to all stations.

The process described above is repeated for the desired number of training rounds, allowing the models to collectively learn from the distributed data across the stations.

The ClustME collaborative learning strategy workflow is illustrated in Fig.5.3(b). This strategy incorporates a time-series clustering step prior to local updates and model exchanges. In this chapter, the K-means clustering method is employed to create two clusters of air monitoring stations based on the similarity of their time-series data. Each formed cluster follows a similar process path. Within each cluster, the stations proceed as follows:

1. **Local model training**: Each station trains its model using its local data, taking into account the time-series characteristics specific to that station.

2. **Model sending**: After local training, each station transfers its trained model to another monitoring station within the same cluster.

3. **Model receiving**: Each station also receives a trained model from another station within the same cluster. This process creates a circular shift pattern among the stations. The exchanged models contribute to the collective learning of each station.

The process described above is repeated until all stations in the cluster obtain the trained models from other stations. This iterative exchange and shifting of models enable the stations to benefit from the collective knowledge and experiences of the other stations within the same cluster.

The SpaTemp collaborative learning strategy implemented in this chapter follows the proposed strategy described in our previous work [1]. For each station, three nearby sites that exhibited the highest correlation based on the stations' pollutant data are selected based, implementing Pearson's correlation coefficient between stations. The workflow of SpaTemp, as depicted in Fig.5.3(c), can be summarised as follows:

1. **Data request**: Each station requests data from the three most correlated nearby stations. This step allows the station to acquire additional information from stations with high correlation, enhancing the learning process.

2. **Data serving**: The station, after receiving its requested data, serves its local data to other stations that request it. This data exchange enables the sharing of information between stations, contributing to collective learning.

3. **Local training**: Once the data exchange process is completed, each station trains the shared data locally. This localised training allows stations to adapt and learn from the collective knowledge acquired through data sharing.

By incorporating the correlation-based data selection and data sharing among stations, SpaTemp promotes collaborative learning and takes advantage of the spatial and temporal relationships within the dataset to improve prediction performance.

5.6.2 Federated Learning (FedAvg)

This chapter adopts the Federated Average (FedAvg) algorithm proposed by McMahan *et al.* as the basis for our collaborative learning approach [221]. Typically, federated learning can be described as follows. First, the main model starts on a central server. This model is the same for all devices involved in federated learning. Then, the global model is sent to the participating local devices. Each device uses this model to learn from its own data. Local devices repeat this process without sharing their data with a central server to keep it confidential. Once local learning is complete, the device sends updates or learned patterns to a central server. The server combines these updates to improve the main model, using techniques such as averaging. After updating, the server checks the performance of the main model with validation data. Then, the corrected global model is saved for the next round of learning. This process continues, with local devices contributing to major model improvements while maintaining user data privacy.

FedAvg is widely used as a standard federated learning (FL) setting. Algorithm 5.1 presents the modified version of FedAvg used in this study. The total number of K edge devices representing air quality monitoring stations are used in this work, where $K = 8$. Each station possesses its local air quality data and trains its model locally, eliminating the need to send raw data to a centralised coordinator. The local dataset is divided into batches of size $\mathcal{B}$ and utilised to train the local model during the local epochs E. At round t, each station k locally trains its model. For each local epoch and batch, the station updates its model parameters as $\theta_t^k \leftarrow \theta_t^k - \eta \nabla \mathcal{L}_k(\theta_t^k)$, where θ_t^k represents the model parameters, η denotes

Algorithm 5.1 Modified FederatedAveraging implemented in this work (FedAvg).

COORDINATOR EXECUTES:
  for each global round $t = 1, 2, \ldots, R$ **do**
    for each station $k \in K$ **in parallel do**
      $\theta_{t+1}^{k} \leftarrow$ **StationUpdate**(θ_t)
    end for
    $\theta_{t+1}^{k} \leftarrow \sum_{k=1}^{K} \frac{1}{K} \theta_{t+1}^{k}$
  end for

StationUpdate(θ) :
  $\mathcal{B} \leftarrow$ (split local dataset into batches of size $\mathcal{B}$)
  for each local epoch i from 1 to E **do**
    for each batch $b \in \mathcal{B}$ **do**
      $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta; b)$
    end for
  end for
  return θ to coordinator

the learning rate, and $\mathcal{L}k(\theta_t^k)$ represents the loss function specific to each station k. Subsequently, each device transmits its parameter update, and the coordinator receives these updates from all participating stations denoted as θ_{t+1}^{k}. The coordinator aggregates the received parameters from all stations, computing the weighted average as $\theta_{t+1}^{k} \leftarrow \sum_{k=1}^{K} \frac{1}{K} \theta_{t+1}^{k}$. The aggregated results are then sent back to the stations, and the process repeats for a total of R rounds. This iterative procedure ensures that the models collectively learn from the distributed data across the stations, potentially improving the overall performance of the collaborative learning process.

It is important to highlight that in this approach, all edge devices participated in each round of federated learning. As a result, there was no need for the coordinator to perform sample fractioning in each round. The air quality datasets were uniformly trained across all stations with the same dataset size. Additionally, during the aggregation step, the updated parameters received from each station were equally weighted. To minimise the communication cost in the first round, this approach implemented a strategy where the model's weights were initialised directly at the edge devices instead of transmitting the initial weights from the coordinator to the edge devices. This approach helped reduce the amount of data that needed to be exchanged during the initial stage, improving the overall efficiency of the collaborative learning process.

Managing MQTT publish/subscribe topics enables the direct implementation

of the federated learning (FL) workflow on edge devices. This approach involves determining the necessary topics at each step of the FL process and deciding which device should publish/subscribe to a particular topic.

5.6.3 Clustered peer-to-peer model exchanges (ClustME)

This learning strategy shares the locally trained models among the stations through a cyclic peer-to-peer model-shifting process. Each station trains its own model and then passes it on to another station, creating a circular pattern of model transfers. To optimise the efficiency of this process, this approach employs a clustering technique.

Given that this chapter's objective is to predict the value of $PM_{2.5}$, this approach utilised K-means clustering algorithm based on the series of $PM_{2.5}$ data. This clustering method becomes particularly beneficial when numerous edge devices are involved in collaborative learning. With many participating devices, it becomes challenging to circulate the trained models among all participants. By applying clusters, the participating stations can be grouped, and the model transfers within each cluster can be limited, reducing the overall number of model transfers required. Moreover, the clustering method helps in optimising the total learning time. By limiting the model transfers to within clusters, the communication overhead can be minimised, and the process of exchanging the trained models among the stations can be streamlined.

ClustME uses the dynamic time warping metric for assignment and barycentre computations to cluster stations based on the history of $PM_{2.5}$ data. The tslearn Python package was employed for this purpose [238]. The stations are divided into two clusters using the clustering method, and the clustering results are summarised in Table 5.1. It is observed that Station-01, Station-02, Station-03, and Station-07 belonged to the same cluster, while the remaining stations formed the other cluster. Each cluster consisted of four stations. Considering that each cluster contains four stations, a total of three model shifts were required after the local updates to complete one learning workflow, as depicted in Fig. 5.3(b). This cyclic peer-to-peer model-shifting process facilitated the dissemination of locally trained models

Table 5.1: Cluster of stations based on time-series of $PM_{2.5}$ data.

Cluster-1	**Cluster-2**
Station-01,-02,-03, and -07	Station-04,-05,-06, and -08

across the stations within each cluster, enabling collaboration and knowledge sharing among the devices.

Algorithm 5.2 presents the general workflow of the ClustME method within a station cluster. Instead of sending local updates to a central coordinator, the locally trained model is transmitted to a specific target station within the cluster. To illustrate this process, let's consider Cluster-1. Station-01 sends its updated model to Station-02, Station-02 sends its updated model to Station-03, Station-03 sends its updated model to Station-07, and Station-07 sends its updated model back to Station-01. This cyclic peer-to-peer model exchange is established, enabling the circulation of trained models among the stations within the cluster. It is important to note that this method has no aggregation step. Each station receives a model

Algorithm 5.2 Clustered peer-to-peer model exchanges (ClustME).

for each cluster $c = 1, 2, \ldots$ **do**
  for each global round $t = 1, 2, \ldots, R$ **do**
    for each station $k = 1, 2 \ldots, K$ **in parallel do**
      $\theta_t^k \leftarrow$ **StationUpdate**(θ_t)
    end for
    for each station $k = 1, 2 \ldots, K$ **do**
      $\theta_{t+1}^k \leftarrow$ **ModelSharing**(θ_t^k)
    end for
  end for
end for

StationUpdate(θ) :
  $\mathcal{B} \leftarrow$ (split local dataset into batches of size $\mathcal{B}$)
  for each local epoch i from 1 to E **do**
    for each batch $b \in \mathcal{B}$ **do**
      $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta; b)$
    end for
  end for
  return θ

ModelSharing(θ^k) :
  // if $k = K$, then send to $k = 1$
  Send θ^k to $k + 1$
  // if $k = 1$, then receive from $k = K$
  Receive θ^{k-1} from $k - 1$
  // update model
  $\theta^k \leftarrow \theta^{k-1}$
  return θ^k

from a neighbouring station, updates the received model with its local data, and subsequently sends the updated model to the designated target station. This process ensures that the knowledge and insights gained from local training are shared across the cluster, fostering collaborative learning among the edge devices.

In this work, a coordinator device is assigned to facilitate and manage the collaborative learning process among the stations. The stations involved in the learning process may have varying speeds in completing the training rounds. Therefore, the coordinator device plays a crucial role in coordinating data transmission between the stations, ensuring a smooth and synchronised workflow throughout the training process.

5.6.4 Spatiotemporal data exchanges (SpaTemp)

In the SpaTemp approach, instead of transmitting deep learning models, the stations exchange pollutant data with each other. Each station combines its local data with the pollutant data received from nearby stations to train its local model. Unlike FedAvg and ClustME, in SpaTemp, the updated model is not transmitted to other stations. Instead, each station collects data from the three most correlated nearby stations based on their $PM_{2.5}$ values. By incorporating this additional data, each station can train its model locally and improve its prediction capabilities.

In addition to collecting data from the three most correlated nearby stations, each station shares its pollutant data with other stations upon request. In the context of the MQTT protocol, these request and transfer processes are referred to as *subscribing* and *publishing*, respectively. Subscribing and publishing are the fundamental actions driving MQTT's communication flow. As an illustration, if Station-01 wants to collect data from Station-04, Station-01 needs to subscribe to a specific topic related to Station-04, and Station-04 needs to publish data on that topic to Station-01. This way, the data will be transmitted from Station-04 to Station-01. The workflow of the SpaTemp process is presented in Algorithm 5.3.

After calculating the Pearson correlation coefficient for all stations, the subscribing and publishing pairs were obtained, as illustrated in Fig. 5.4(a). As shown in the figure, each station subscribes to the three most correlated nearby stations and publishes its data to one or more stations. For example, Station-01 subscribes to data from Station-05, Station-04, and Station-08. Additionally, Station-01 publishes its data to Station-04, Station-05, Station-06, and Station-08, as shown in Fig. 5.4(b). Similarly, Station-02 subscribes to data from Station-03, Station-06, and Station-07, and it publishes its local data to Station-03 and Station-07. When implementing this method directly on edge devices, one of the main challenges is

Algorithm 5.3 Learning with spatiotemporal data exchanges (SpaTemp).

STATION DATA COLLECTION:
  $\mathcal{M} \leftarrow$ (Collect $PM_{2.5}$ data from participating stations)
  $\mathcal{C} \leftarrow$ (Perform Pearson's correlation on $\mathcal{M}$)
  for station $k \in K$ **do**
    // create a list of nearby stations
    $\mathcal{S} \leftarrow$ (Select three significant $\mathcal{C}$ relative to k)
    $\mathcal{D} \leftarrow$ (Collect $PM_{2.5}$ from all members of $\mathcal{S}$)
    $\mathcal{H} \leftarrow$ (Combine $\mathcal{D}$ with station's local dataset)
  end for

for each global round $t = 1, 2, \ldots, R$ **do**
  for each station $k = 1, 2 \ldots, K$ **in parallel do**
    $\theta_{t+1}^{k} \leftarrow$ **StationUpdate**(θ_t)
  end for
end for

StationUpdate(k, θ) :
  $\mathcal{B} \leftarrow$ (split local dataset into batches of size $\mathcal{B}$)
  for each local epoch i from 1 to E **do**
    for each batch $b \in \mathcal{B}$ **do**
      $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta; b)$
    end for
  end for
  return θ

orchestrating the transfer of pollutant data among stations.

5.7 Deep Learning Models

Fig. 5.5 illustrates the deep learning models proposed in this chapter, which were implemented using the TensorFlow 2.4 framework [114]. These models primarily consist of three different types of layers: one-dimensional convolutional (Conv1D) layers, long short-term memory (LSTM) layers, and fully connected (Dense) layers. The key properties of each layer, such as the number of filters, kernel size, and unit size, are indicated in Fig. 5.5. Except for the last layer, each layer utilises the rectified linear unit (ReLU) activation function. The output (Dense) layer does not apply any activation function, and the other layer parameters follow the default settings provided by the framework.

FedAvg and ClustME employ the same model architectures, as shown in Fig. 5.5(a). On the other hand, SpaTemp utilises a slightly different design, as depicted

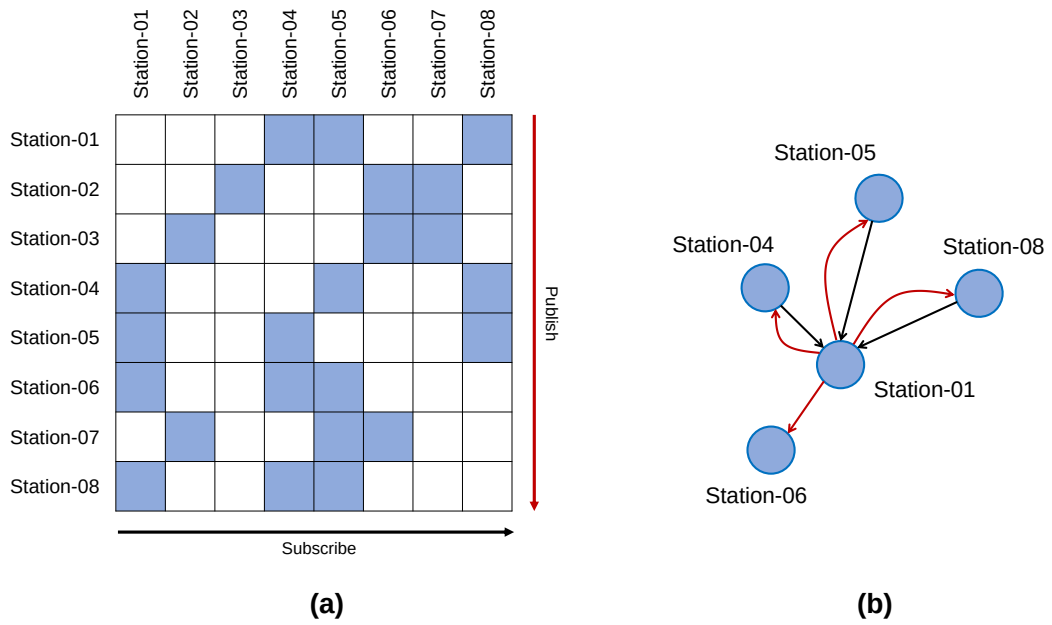
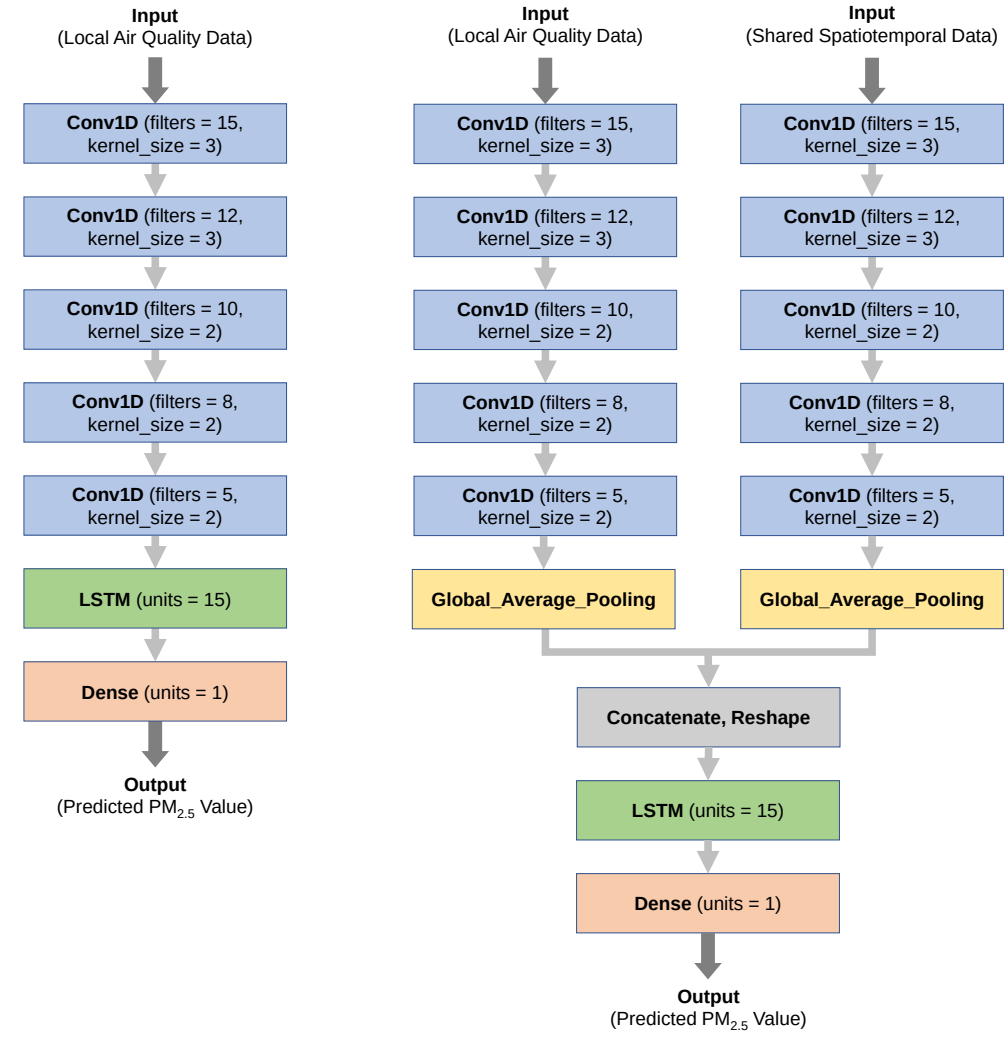


Figure 5.4: (a) Subscribing and publishing data pairs performed in SpaTemp, and (b) An example of publishing and subscribing implemented at Station-01.

in Fig. 5.5(b). SpaTemp features a parallel structure of convolutional layers, with both paths sharing the same layer properties. In this architecture, the first path is responsible for extracting features from the local data, while the second path captures the characteristics of the shared spatiotemporal data. These spatiotemporal data are obtained from three nearby air quality stations with the highest Pearson's correlation coefficients to the target station. Since our task involves time-series data for regression purposes, it is crucial to preserve the sequential order of the air quality data to extract accurate information. Furthermore, SpaTemp collects data from multiple stations, necessitating a parallel structure to fulfil these requirements.

All models in this work are designed to process current and past seven hours of data, creating input sets of eight hours in length. Given that the dataset consists of 11 columns ($PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed), the size of the input sets becomes 8×11. These input sets are then utilised for training the models for predicting hourly values of $PM_{2.5}$. In SpaTemp, the deep learning model not only accepts the local air quality data (representing the first path of the input model) but also incorporates $PM_{2.5}$ data collected from itself and the other three stations (representing the second path of the input model). Consequently, the second path of the input model takes a matrix with the size of 8×4 to account for the additional data from neighbouring



(a) Deep Learning Model for FedAvg and ClustME

(b) Deep Learning Model for SpaTemp

Figure 5.5: Proposed deep learning architectures.

stations.

5.8 Collaborative Learning Evaluation

The evaluation of the proposed collaborative learning approach involves several aspects. First, the selection of input features was conducted to determine the optimal performance, considering different numbers of features from the original dataset. The losses during training were visualised to assess the learning capability of the model with respect to the training data. Subsequently, the model's performance

was evaluated on unseen test data for each round of training. Furthermore, the time required by each learning technique to complete the training task was measured, and the performance of each edge device was assessed. The communication costs associated with the execution of the learning strategies were also evaluated. Additionally, the expansion of edge device networks was discussed, considering the scalability and potential growth of the system.

For evaluating the model's performance on the test data, several metrics were employed, including root mean squared error (RMSE), mean average error (MAE), mean absolute percentage error (MAPE), and coefficient of determination (R^2). These metrics provide insights into the models' accuracy, precision, and predictive capability.

5.9 Application Scenario

The experimental setup involves eight air quality monitoring stations, each represented by a different edge device. Station-01 to Station-07 are equipped with three Raspberry Pi (RPi) board variants, while Station-08 utilises an NVIDIA Jetson Nano 2GB Developer Kit. Notably, Station-08 serves a dual role as both the eighth station and the server. This configuration is made possible by the MQTT configuration, which allows a device to function as both a client and a server within a single network. The application scenario of the edge devices is depicted in Figure 5.6.

The communication network between the edge devices operates using the MQTT (Message Queuing Telemetry Transport) protocol. MQTT is a client-server protocol that facilitates message transmission through topic subscribing and publishing mechanisms. It operates on top of the TCP/IP protocol, ensuring reliable communication [239]. The publish/subscribe architecture of MQTT enables scalable and efficient connectivity between resource-constrained edge devices, whether over the Internet or within a local area network (LAN). MQTT is known for its simplicity and lightweight nature, making it ideal for transmitting information over networks with limited bandwidth, high latency, or unreliability [240].

Each station in the network can function as both a message publisher and receiver. The messages exchanged can include air quality measurement data and deep learning models relevant to this study. To receive messages from other stations, each station subscribes to specific topics of interest to the MQTT broker(server). The communication between the devices and the MQTT broker occurs within a local area network established using a wireless router. The MQTT protocol facilitates the seamless transmission of messages within this network, enabling efficient and reliable

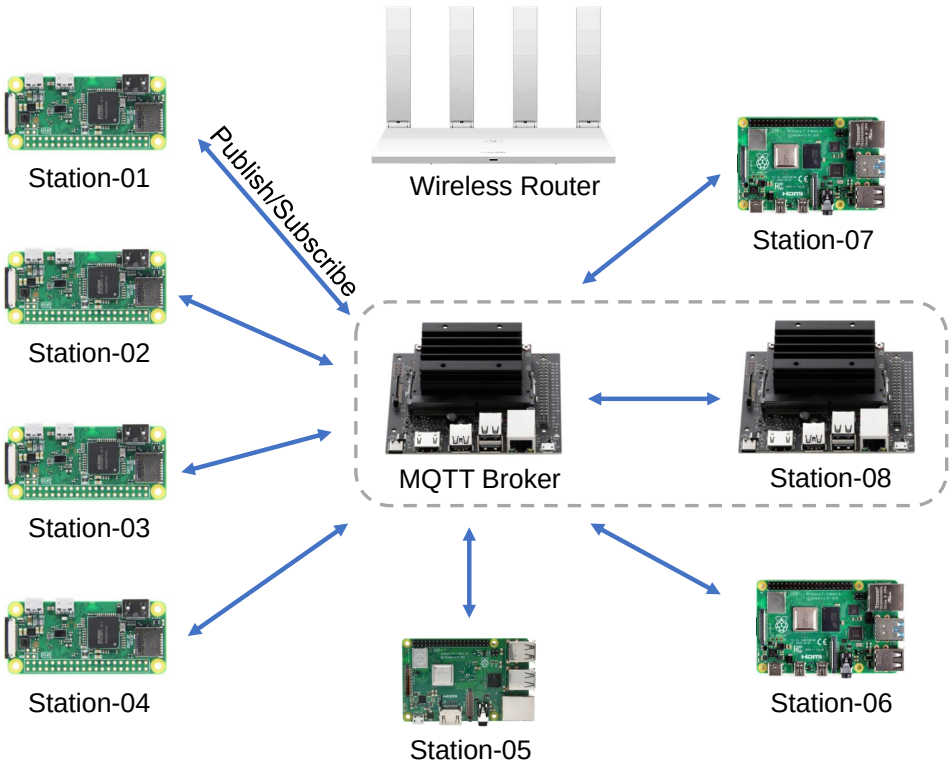


Figure 5.6: Edge devices application scenario.

station communication.

Based on the official websites, the Raspberry Pi (RPi) boards require a 5-volt power supply. The recommended power supply unit (PSU) current capacities for the RPi 4B, RPi 3B+, and RPi Zero W are 3.0A, 2.5A, and 1.2A, respectively. The Jetson Nano 2GB Developer Kit also requires a 5V/3A, the same as the RPi 4B board. Additionally, each board is equipped with a specific operating system (OS). The RPi boards utilise Raspberry Pi OS Lite version 10 (Debian Buster), while the Jetson Nano uses NVIDIA JetPack 4.6 with Ubuntu 18.04. In this work, all boards operate in headless mode, meaning all boards are operated without a display attached.

5.10 Results and Discussion

5.10.1 Feature Selection

To determine the correlation among the features, Pearson's correlation coefficients were calculated. The overall correlation was obtained by averaging the coefficients across all monitoring stations. The averaged correlation coefficients are presented

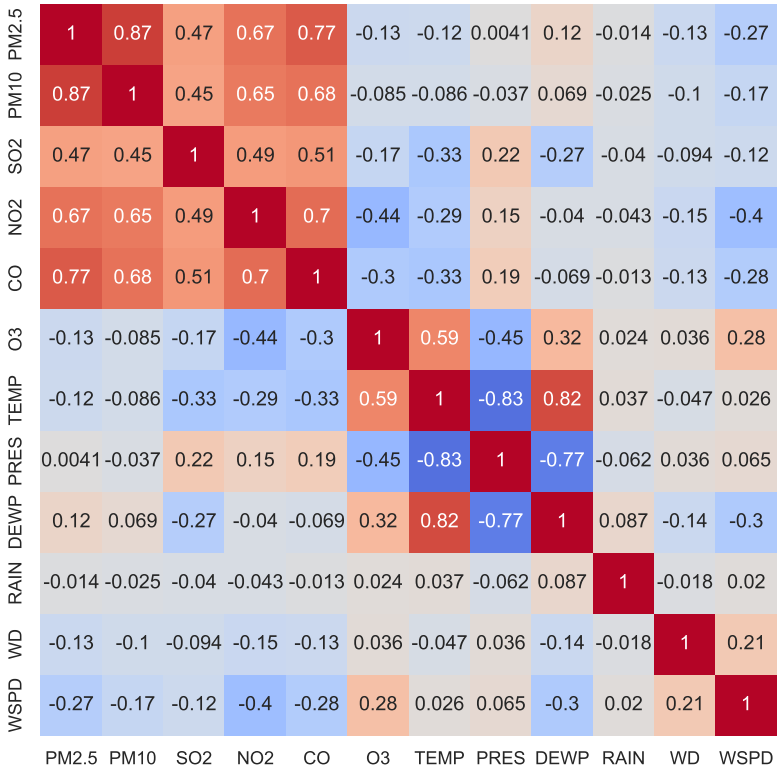


Figure 5.7: The average correlation coefficients among features.

in Fig. 5.7. As depicted in Fig. 5.7, $PM_{2.5}$ levels exhibit strong positive correlations with PM_{10}, NO_2, and CO ($r > 0.6$). There is a moderate positive correlation with SO_2 ($r = 0.47$), and a weak negative correlation with O_3 ($r = -0.13$). Additionally, rainfall (RAIN), dew point (DEWP), air pressure (PRES), and air temperature (TEMP) display the weakest correlations with $PM_{2.5}$. Only these four features were varied to determine the optimal input set for the deep learning model. The feature selection process involved training the model locally without utilising collaborative learning. The model architecture used for training is illustrated in Fig. 5.5(a).

Table 5.2 displays the outcomes of the input feature selection process. The models were locally trained for 40 epochs, and their performance was evaluated for each combination. The average root mean square error (RMSE) values across all stations were computed and reported in the table. The experimental results indicate that the best performance is obtained by excluding the rain feature during training, resulting in the selection of 11 attributes: $PM_{2.5}$, PM_{10}, SO_2, CO, NO_2, O_3, temperature, air pressure, dew point, wind direction, and wind speed as the input features for the subsequent evaluation stages

Table 5.2: Feature selection results.

Input Feature	Number of Inputs	Avg. RMSE
All	12	23.126
Without PRES	11	23.370
Without RAIN	11	**23.018**
Without PRESS and RAIN	10	23.316
Without PRESS, RAIN, and DEWP	9	23.424
Without PRESS, RAIN, and TEMP	9	23.270
Without PRESS, RAIN, DEWP, and TEMP	8	23.526

5.10.2 Losses During Training

Losses measure how accurately a predictor can capture the relationship between inputs and the provided targets. A lower loss indicates a better-performing predictor. During the training process, training losses are computed to assess the model's fit to the training data. On the other hand, validation losses are evaluated at the end of each epoch using validation data. Training loss reflects the model's performance on the training data, while validation loss evaluates unseen validation data. In this work, mean squared error (MSE) is employed as the metric for both training and validation losses. Figure 5.8 illustrates examples of training and validation losses for Station-06, as depicted in Figure 5.8(a) and 5.8(b), respectively.

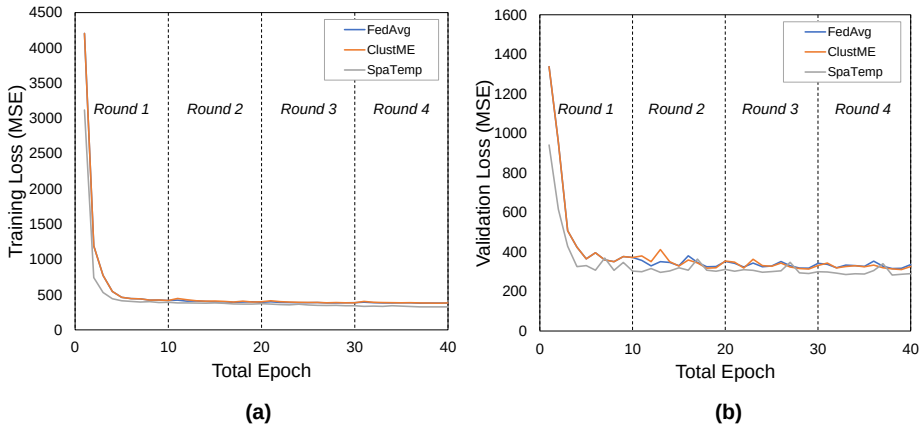


Figure 5.8: Examples of losses for Station-06: (a) Training loss and (b) validation loss.

To ensure reproducibility and reduce bias in the results, the proposed deep learning models were initialised with the same random seeds for all methods. In programming, a random seed is the starting point for generating random numbers. This initialises the pseudorandom number generator algorithm, determining the sequence of generated random values. By providing an initial value, the resulting sequence of random numbers (weights and biases) can be reproduced. The training performance was then evaluated, and it was observed that FedAvg and ClustME produced identical training and validation losses during the first round. This outcome is expected since both methods perform local training and utilise the same model architecture and training/validation data in the initial round.

However, after the first round, variations in losses between FedAvg and ClustME became apparent due to their distinct learning approaches. All collaborative learning methods demonstrated a rapid reduction in training losses, particularly within the first three epochs, as depicted in Fig. 5.8(a). Though these dynamic changes in training losses are challenging to observe from Fig. 5.8(a), a more explicit illustration of how training losses vary after the first round is provided in Fig. 5.9 to enhance clarity.

Fig. 5.9 clearly demonstrates that the SpaTemp method surpasses other approaches in minimising loss functions during the training process at all participating stations. The utilisation of spatiotemporal data and the collaborative sharing of pollutant data among stations effectively improve the performance of the models. It is worth noting that SpaTemp achieves this improvement without relying on model sharing or aggregation processes. Instead, it incorporates raw measurement data as

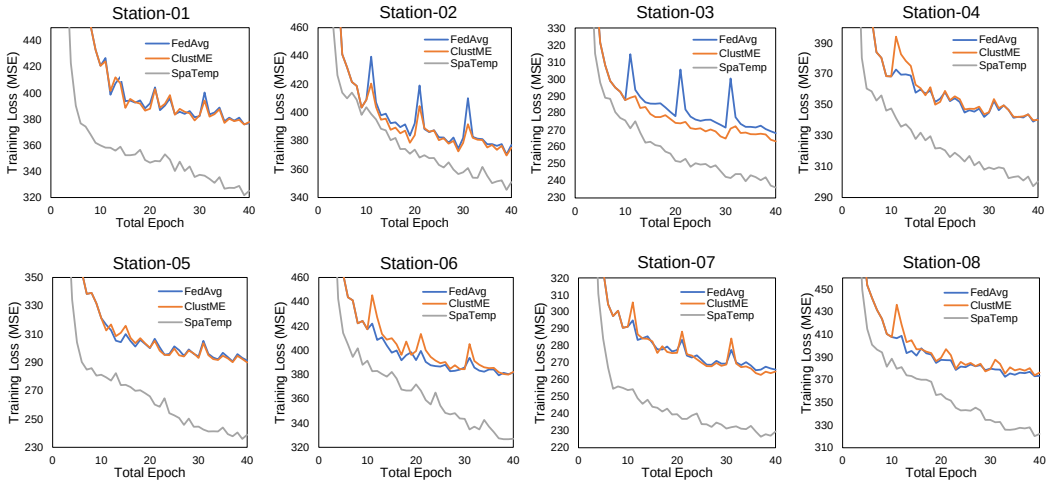


Figure 5.9: Better presentation of training losses at all stations after the first round.

additional inputs to gain knowledge.

5.10.3 Model performance on testing data.

In this study, approximately 10% of the data is reserved for testing purposes. The model performances on the test data are evaluated using various metrics, including root mean square error (RMSE), mean absolute error (MAE), mean absolute percentage error (MAPE), coefficient of determination (R^2), and rate of improvement (RIR). Table 5.3 presents these performance metrics for the proposed collaborative learning methods and the locally learned models. It is worth noting that the locally learned models do not involve any collaborative strategies, such as aggregation or model-sharing procedures, as seen in FedAvg and ClustME. Additionally, the locally learned models gain knowledge solely from their local data without collecting pollutant data from other stations, as implemented in SpaTemp. Therefore, it can be concluded that no data is leaving the station during the training process.

The effectiveness of SpaTemp during training is evident in its superior performance on the test data. SpaTemp consistently outperforms other methods in predicting unseen data for all monitoring stations, as indicated by the values highlighted in bold in Table 5.3. Furthermore, by considering the RMSE group row in Table 5.3, the Rate of Improvement (RIR) can be calculated, and the results are reported in the RIR group row. A positive RIR value signifies an improvement of the proposed collaborative learning method compared to the locally learned method used as the benchmark.

SpaTemp performs better than other collaborative learning methods, with RIR values ranging from 0.525% to 8.934%. The most notable improvement is observed at Station-01. The RIR is derived from the RMSE. Since the SpaTemp method exhibits superior performance compared to other methods, the RIR values are positive. At Station-06, the RMSE for SpaTemp is only marginally lower than that of other methods, with values of 24.166 $\mu g/m^3$ and 24.293 $\mu g/m^3$, respectively. This yields an RIR value of 0.524%, as per Equation 2.15. In contrast, SpaTemp significantly outperformed other methods at other stations, evidenced by lower RMSE values. For instance, at Station-01, the RMSE values for SpaTemp and the local method are 19.917 $\mu g/m^3$ and 21.871 $\mu g/m^3$, respectively, resulting in an RIR value of 8.934%, notably higher than those obtained at Station-06. However, it is important to note that not all collaborative learning strategies outperform the locally-learned method. ClustME shows slight degradations in model performance at Station-01, Station-05, and Station-08, as indicated by the negative RIR values. These degradations range from 0.092% to 0.488%. Similarly, the federated learning

Table 5.3: Model performance in predicting $PM_{2.5}$ on test data.

	Sta-01	Sta-02	Sta-03	Sta-04	Sta-05	Sta-06	Sta-07	Sta-08
RMSE($\mu g/m^3$)								
FedAvg	21.743	23.135	21.097	25.118	24.344	24.258	21.483	23.988
ClustME	21.978	23.345	20.772	25.021	24.213	24.249	21.619	24.169
SpaTemp	**19.917**	**22.140**	**20.738**	**24.299**	**22.470**	**24.166**	**20.177**	**23.104**
Local	21.871	23.744	20.890	25.170	24.191	24.293	21.648	24.058
MAE($\mu g/m^3$)								
FedAvg	12.008	13.182	10.981	12.916	13.651	12.538	9.867	13.594
ClustME	12.028	13.263	10.910	12.863	13.411	12.554	9.816	13.877
SpaTemp	**10.729**	**12.173**	**10.741**	**12.395**	**12.031**	**12.394**	**9.255**	**13.328**
Local	11.967	13.708	10.967	12.965	13.485	12.691	9.937	13.702
MAPE(%)								
FedAvg	26.881	28.441	30.090	37.727	37.913	24.286	32.865	37.239
ClustME	27.351	28.411	31.288	37.299	37.848	24.360	31.715	38.160
SpaTemp	**26.239**	**27.594**	**28.645**	**38.230**	**36.177**	**22.603**	**29.144**	**31.391**
Local	26.347	28.747	30.768	37.996	37.107	24.156	32.150	38.753
R^2								
FedAvg	0.958	0.935	0.950	0.953	0.949	0.954	0.938	0.954
ClustME	0.957	0.934	0.951	0.953	0.950	0.954	0.937	0.953
SpaTemp	**0.964**	**0.941**	**0.952**	**0.956**	**0.957**	**0.955**	**0.945**	**0.957**
Local	0.957	0.932	0.951	0.952	0.950	0.954	0.937	0.954
RIR(%)								
FedAvg	0.588	2.564	-0.995	0.208	-0.636	0.145	0.762	0.292
ClustME	-0.488	1.681	0.565	0.593	-0.092	0.181	0.134	-0.463
SpaTemp	**8.934**	**6.756**	**0.725**	**3.461**	**7.115**	**0.524**	**6.794**	**3.967**
Local (baseline)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

approach exhibits degradations of 0.995% and 0.636% at Station-03 and Station-05, respectively.

Figure 5.10 provides a visual representation of the model performances, specifically at Station-02, offering a more intuitive understanding. The upper plots present the predicted and observed values plotted together. Moreover, the lower plots facilitate a quick assessment of the proximity between the predicted and observed values using diagonal lines as references. Among the methods evaluated, the SpaTemp model demonstrates superior performance in capturing extreme $PM_{2.5}$

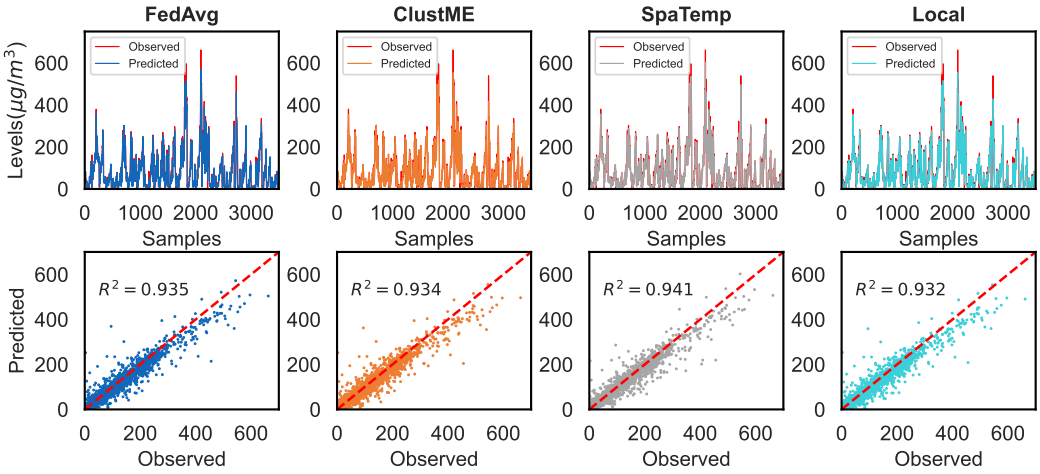


Figure 5.10: Comparison between the observed and predicted values at Station-02.

levels, resulting in a higher R^2 score of 0.941. The figure clearly illustrates the close alignment between the predicted values (depicted by the grey line) and the observed values (depicted by the red line), particularly in accurately capturing very high concentrations of $PM_{2.5}$ compared to the other methods.

As shown in Figure 5.10, the models tend to underestimate certain spikes in $PM_{2.5}$ values. Several factors can contribute to inaccurate predictions of pollutant data spikes by models. If sudden spikes are rare or infrequent in the training data, the model may not have learned to capture and generalise these patterns effectively. Additionally, deep learning models, particularly those based on recurrent neural networks (RNNs) or long short-term memory (LSTM) networks, can be sensitive to noise in the data. Sudden spikes may introduce noise that the model struggles to filter out. Furthermore, deep learning models may overfit the training data, meaning they memorise specific patterns in the training set that do not generalise well to unseen data. As a result, sudden spikes may be perceived as outliers or anomalies by the model, leading to inaccurate predictions.

The estimation of longer time periods can be conducted using the same methodology employed for acquiring subsequent 1-hour predictions and adjusting the target sets accordingly. In this section, the prediction results of $PM_{2.5}$ levels for the upcoming 3-hour, 6-hour, 9-hour, and 12-hour periods are presented. Figure 5.11 presents the prediction outcomes obtained from Station-05. The figure reveals a decline in model performance as the prediction period extends. Evaluation based on R^2 scores indicates that the model's accuracy ranges from approximately 0.8 for 3-hour predictions to about 0.4 for 12-hour predictions. Furthermore, the SpaTemp

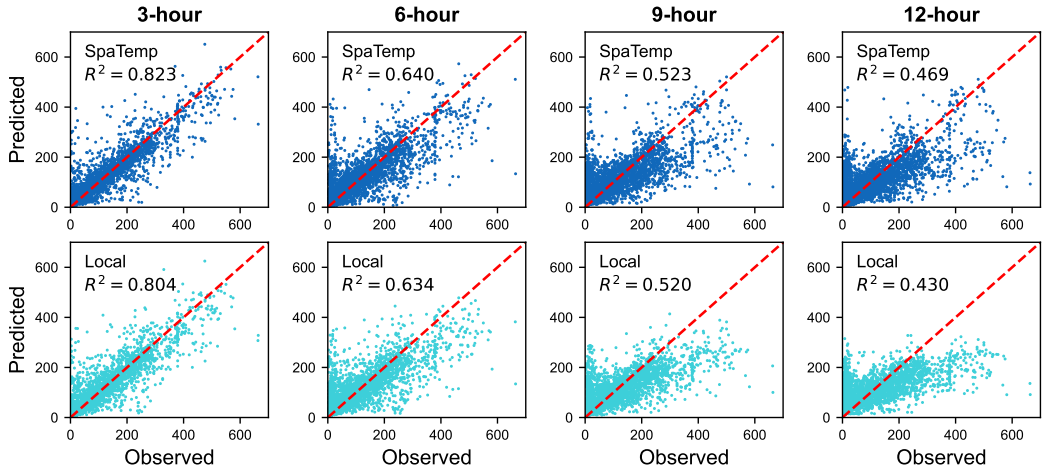


Figure 5.11: Longer prediction hours evaluated at Station-05.

approach demonstrates better performance compared to the locally-trained model.

5.10.4 Learning Execution Period

In this section, the average time required to complete the training process for the collaborative learning strategies is reported. The collaborative learning strategies were executed four times, and the time to complete the training was averaged. Among the methods considered, Federated Learning (FedAvg) exhibited the shortest training time (approximately 49 minutes). On the other hand, as expected, SpaTemp had the longest training sessions, taking approximately 61 minutes to complete. This is because SpaTemp utilised a larger deep learning model and input data, resulting in slower epoch completion than other methods. The values reported in Table 5.4 are based on the training time of the slowest edge device.

In the ClustME approach, two different clusters were formed for training. The first cluster (consisting of Station-01, -02, -03, and -07) comprised slower edge devices, specifically the Raspberry Pi Zeros. The second cluster (consisting of Station-04, -05, -06, and -08) primarily consisted of faster devices, including Raspberry Pi 3B+, Raspberry Pi 4B, and Jetson Nano. The completion times for the second cluster were approximately 240 seconds faster than the first. This difference

Table 5.4: Average time to complete the collaborative training.

	FedAvg	ClustME (Cluster-1)	ClustME (Cluster-2)	SpaTemp
Time(Sec)	2954.40	3220.62	2979.62	3682.09

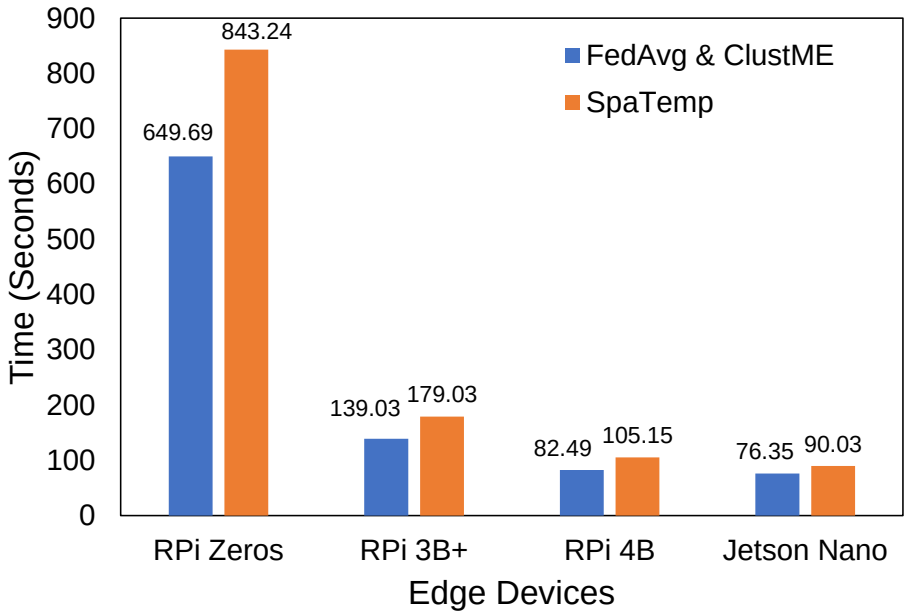


Figure 5.12: The average time of edge devices to complete the training sessions.

in completion times can be attributed to the varying processing capabilities of the devices within each cluster.

The average time required for each edge device to complete the training steps is also reported in this study. The average completion time was calculated after performing 40 rounds using different learning strategies. The results are presented in Fig 5.12. Since multiple Raspberry Pi (RPi) 4 and Zero devices were used, the execution times of the same device types were averaged. The figure demonstrates that the Jetson Nano 2GB developer kit outperforms other devices, with completion times up to approximately nine times faster than Raspberry Pi Zeros when executing the SpaTemp method. The slower devices, such as Raspberry Pi Zeros, contribute to longer completion times in collaborative learning strategies. The faster processing capabilities of the Jetson Nano contribute to its superior performance in terms of training speed.

5.10.5 Communication Cost Estimations

In this study, communication costs are measured based on the transferred payload in specific MQTT topics, considering both incoming and outgoing payloads. The payloads can either be deep learning models or pollutant data. The MQTT payloads for FedAvg and ClustME during the learning process consist of deep learning models.

However, for SpaTemp, the payloads are pollutant data (specifically $PM_{2.5}$ data) collected during data collection and not during the learning phase.

Since collaborative learning involves multiple devices operating at different speeds, certain governing MQTT topics were utilised in addition to those containing the actual payloads. These governing topics serve various purposes, such as initiating local updates or instructing a participating device to send its local model to another device. For these governing topics, zero-length payloads are published, meaning they do not contain any actual data. Although these governing topics require the transmission of bytes of data, they are excluded from the communication cost measurement since they do not contain payload data.

In this study, the initial model size for FedAvg and ClustME is approximately 44 kB, while for SpaTemp is around 77 kB. As previously mentioned in Section 5.7, FedAvg and ClustME employ the same network architectures. All models are compiled with the Adam optimiser before the training processes begin. As one round of training is completed, the model file sizes increase since the Adam optimiser maintains the gradient states during training. Consequently, the trained model sizes become approximately 91 kB for FedAvg and ClustME, and 149 kB for SpaTemp. In collaborative learning strategies, maintaining the gradient states after each round is crucial to ensure that the model progressively learns throughout the collaborative training rounds.

The deep learning models for FedAvg and ClustME are transmitted as bytearray objects, while SpaTemp demonstrates a simpler payload transfer method. In SpaTemp, pollutant data is collaboratively sent among stations primarily as text strings. In this work, the size of a single pollutant datum payload is approximately 5 bytes when encoded as UTF-8 strings. The UTF-8 encoding represents characters of pollutant data in one-byte (8-bit) units [241]. Table 5.5 provides an overview of the data transferred for both incoming and outgoing payloads at all participating stations, including the coordinator's side for FedAvg. In FedAvg, the coordinator received models from participating stations, aggregated these models, and transferred them back to the participating stations. In contrast, the coordinators in ClustME and SpaTemp functioned solely as governing devices. Consequently, ClustME reduced communication costs by half compared to FedAvg, despite having the same number of participants.

The file size of the transmitted models was 91 kB for both FedAvg and ClustME. On the other hand, SpaTemp transferred a significantly smaller amount of data, with only about 5 bytes being transmitted in each round. However, SpaTemp performed data transfer much more frequently, with up to 35,064 rounds during data

Table 5.5: Collaborative learnings communication costs.

	FedAvg	**ClustME**	**SpaTemp**
Comm. cost (MB)	11.648	5.824	8.415

collection. Based on our experimental results, the communication costs for FedAvg, ClustME, and SpaTemp are 11.648 MBytes, 5.824 MBytes, and 8.415 MBytes, respectively.

5.10.6 Network Scaling

This section explores insights into the scalability of edge device networks, which is crucial considering the potential expansion of devices used in collaborative learning. As the number of devices increases, the amount of data transmitted during the learning process also grows. For FedAvg and ClustME, the data transmission volume increases as the number of rounds progresses. In contrast, SpaTemp experiences an increase in data transmission size due to the hourly pollutant data, regardless of the number of learning rounds.

In the case of federated learning (FedAvg), the coordinator initially sends the θ_0 (initialisation weights or model) to all participating stations. Each station k trains its local data and sends the result of its training to the coordinator. The coordinator then aggregates all the collected θ_t^k and returns the updated global parameters θ_{t+1}^k to all stations. The number of data exchanges can be expressed as follows:

$$D_{sta} = N\theta_{(0)} + \sum_{i=1}^{T} C_i K \theta_{i(tx)} + \sum_{i=1}^{T} C_i K \theta_{i(rx)} \tag{5.1}$$

where D_{sta} is the number of data exchanges on the station's side, K is the total number of stations, N is the number of participating stations, C is the fraction of stations participated in each round, T is the number of FL rounds and θ_i is the amount of information exchanged during FL (also called payload in MQTT).

The amount of data transmitted to the coordinator ($\theta_{i(tx)}$) is the same as the amount of received data by the station ($\theta_{i(rx)}$). In our work, all stations must be involved in FL rounds ($N = K$ and $C = 1$). Therefore, the number of data exchanges becomes:

$$D_{sta} = K \left(\theta_0 + \sum_{i=1}^{T} 2\theta_i \right) \tag{5.2}$$

On the coordinator's side, the amount of data exchanged will be the same as the amount of data exchanged on the station's side. Also, in this work, the initial model is generated on the station side ($\theta_0 = 0$), and every round consists of the same number of epochs (i.e., the model size is the same at every round). Thus, the total communication cost between stations and the coordinator in this work can be expressed as follows:

$$D_{FedAvg} = 4KT\theta \tag{5.3}$$

ClustME does not require the stations to transmit data to the coordinator, as the coordinator is only responsible for orchestrating the process workflow. Thus, the payloads are transferred among stations, as shown in Fig. 5.13.

A station receives and transmits the same amount of data. Despite having the same number of participants, the communication cost in ClustME is half that of FedAvg. In ClustME, there is no coordinator to facilitate communication. Equation 5.4 expresses the communication cost for ClustME. The equation focuses on the total number of participating stations without explicitly mentioning the communication cost in each cluster.

$$D_{ClustME} = 2KT\theta \tag{5.4}$$

In the learning approach using spatiotemporal data (SpaTemp), the information transferred to a target station is measurement data, specifically pollutant data such as $PM_{2.5}$. The communication in SpaTemp involves sending hourly data from nearby stations to the target station. The communication cost in SpaTemp depends on the number of nearby stations sending their data to the target station and the total amount of hourly data required.

In this work, three nearby stations were selected to exchange data with the

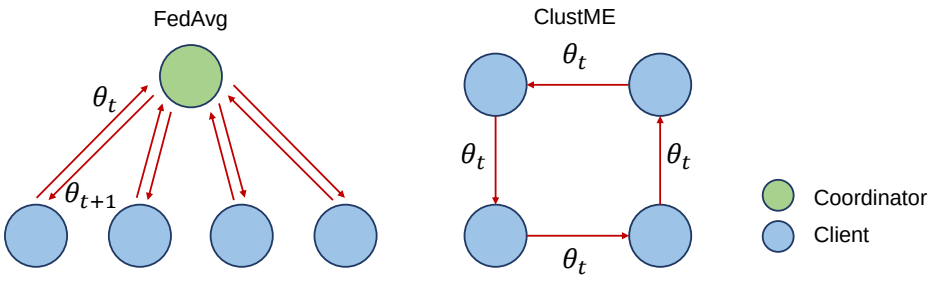


Figure 5.13: The comparison of model exchanges between FedAvg and ClustME with the same number of participating stations.

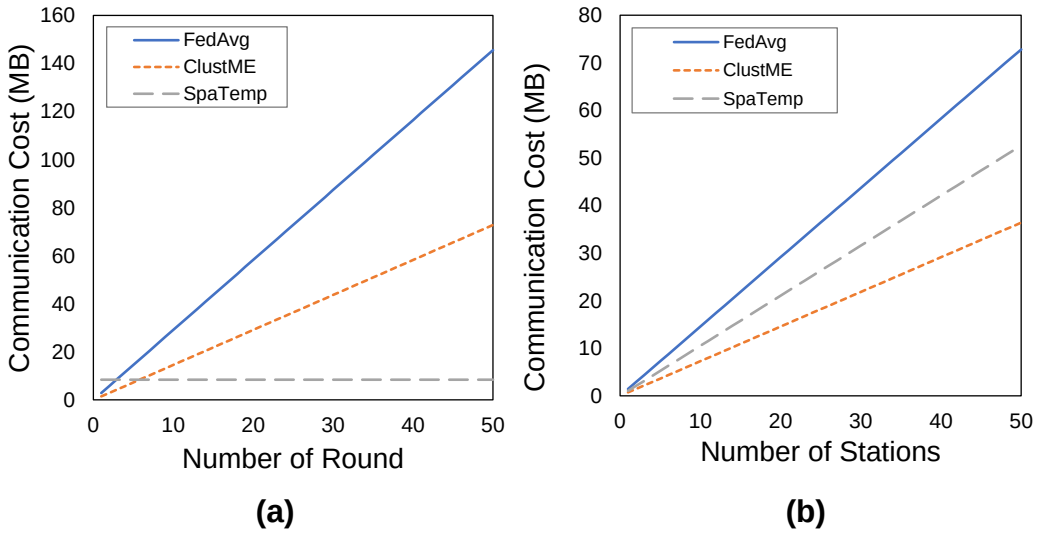


Figure 5.14: The amount of communication cost calculated by changing (a) the number of rounds, and (b) the number of stations.

target station. The data exchange between stations can be visualised as the coloured squares in Figure 5.3(a), representing the pairs of subscribing and publishing stations. Based on this setup, the communication cost in SpaTemp can be expressed as follows:

$$D_{SpaTemp} = 2KN_{neigh}H\varphi \tag{5.5}$$

where K The total number of stations, N_{neigh} the number of participating nearby stations, H is the number of hourly data, and φ the amount of information exchanged during learning (also called payload in MQTT).

As shown in Fig. 5.14, the number of learning rounds and stations are increased to 50 and the other parameters are kept unchanged. As a result, the communication costs for FedAvg and ClustME increase linearly (Fig. 5.14(a)), following Equations 5.3 and 5.4, respectively. However, the communication cost for SpaTemp remains constant, regardless of the number of rounds. In addition, increasing the number of participating stations affects the communication cost for all methods, as shown in Fig. 5.14(b).

5.11 Summary

This work explores three collaborative learning methods: federated learning (FedAvg), learning with model sharing (ClustME), and learning with spatiotemporal

data exchanges (SpaTemp). Among these methods, SpaTemp outperforms others in minimising loss functions during training at all participating stations. This effectiveness during training leads to better performance on test data.

Regarding the time required to complete the training period, FedAvg and ClustME exhibit similar performance. However, due to its larger model size, SpaTemp exhibits slower training times than FedAvg and ClustME. The communication cost in collaborative learning can vary based on factors such as the number of participating stations, learning rounds, and the dataset size. This work investigates the impact of these factors and demonstrates the potential for expanding the number of edge devices.

This chapter demonstrates that collaborative learning can enhance air pollution prediction compared to relying solely on local learning methods. However, there is a trade-off involved when implementing collaborative learning approaches. Based on the experiments conducted, collaborative learning with spatiotemporal data (SpaTemp) outperforms local learning and other collaborative learning methods, as measured by metrics such as RMSE, MAE, MAPE, R^2, and RIR. One significant advantage of SpaTemp is that its performance is not affected by the number of learning rounds. Additionally, if the number of participating stations expands, the total communication cost is lower than federated learning. However, it is important to note that SpaTemp requires transmitting measurement data to other monitoring stations, unlike FedAvg and ClustME, where only model exchanges are involved. Finally, regarding edge device performance, the Jetson Nano development kit demonstrates the fastest completion time for on-device training.

Chapter 6

Tiny Machine Learning for Microcontroller Applications

This chapter is based on the following works:

- **Subchapter 6.3**:
  I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "TinyML Models for a Low-cost Air Quality Monitoring Device," *IEEE Sensors Letters*, vol. 7, no. 11, pp. 1-4, Sep. 2023 [4].

- **Subchapter 6.4**:
  I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising Tiny Machine Learning with Binary Weight Network for a Low-cost Air Quality Monitoring Device," *The 3rd Imperial Workshop on Intelligent Communications*, Imperial College London, 19 - 20 June 2023 [6].

  I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising TinyML Using Binary Weight Network and Meta-Learning for a Low-cost Air Quality Monitoring Device," *Warwick Secure and Intelligent Communications (WSIC) Workshop*, University of Warwick, 31 July 2023 [7].

- **Subchapter 6.5**:
  I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "TinyML with Meta-Learning on Microcontrollers for Air Pollution Prediction," *Eurosensors 2023 Conference*, Lecce, Italy, Sep. 2023 [5].

6.1 Introduction

Recent research has demonstrated the feasibility of low-cost sensor nodes for air quality monitoring systems [76, 81]. This emerging sensor-based air quality monitoring field can provide high-density spatiotemporal pollution data, supplementing the established methodology with more precise and expensive devices [53]. The immense volume of collected spatiotemporal data has provided a better opportunity to apply machine learning (ML) techniques in air quality areas, such as air contaminant prediction [1, 97], missing data imputation [2, 98], and classification tasks [99].

Numerous studies have been conducted on low-cost air quality sensor nodes, focusing on the calibration procedure against reference instruments [81, 242, 243]. Hashmy et al. extended this research line by introducing calibration and forecasting functionalities [244]. The trained machine learning calibration agent is stored in the microcontroller's EEPROM, while the forecasting function is executed on the cloud integration server. Becnel et al. studied a distributed low-cost pollution monitoring platform [245]. The study involved deploying 50 microcontroller-powered nodes across a metropolitan area, which measured various air quality parameters such as particulate matter, CO, NO, air temperature, air humidity, and light density. The results demonstrated that the proposed platform aligned well with the readings from standard instruments ($R^2 = 0.88$). However, the study does not include any air quality prediction capabilities.

The use of mobile sensor networks for measuring air quality parameters has gained popularity, particularly in the context of smart city implementation. Castello et al. developed a data concentrator platform that collected data from low-cost sensors attached to city buses [246]. Their approach involved processing all the collected measurements and leveraging information from reliable fixed stations to enhance the accuracy of the low-cost mobile sensors. DeSouza et al. designed low-cost mobile sensors mounted on trash trucks to study the spatial and temporal characteristics of pollutant sources [247].

This chapter addresses a significant gap in using low-cost air quality nodes, specifically the absence of missing data imputation and forecasting capabilities directly at the node, without relying on cloud services. The presence of missing data poses a common challenge in air pollutant measurement, impacting the interpretation and conclusions of studies and the functionality of air quality-related public services. The concept of edge computing, which involves processing data close to its source, offers potential benefits. By leveraging node computing, tasks can be offloaded from the centralized cloud to the nearby sensing devices, improving security

and reducing latency issues cloud-based systems face.

This chapter presents an implementation of the tiny machine learning (tinyML) paradigm, enabling machine learning models to be executed directly at the node. TinyML is a cutting-edge field of artificial intelligence. It brings machine learning (ML) models to resource-constrained devices, such as microcontrollers [125]. A microcontroller is typically limited to its memory and computational capabilities. Thus, effective deployment of tinyML models requires a thorough understanding of hardware, software, algorithms, and applications. Regardless of their limited performance, microcontrollers can gather physical environment data through sensors and perform decisions based on ML algorithms.

The key motivation of this chapter is to empower a low-cost air quality device with intelligence. Two different tinyML models were deployed to a single microcontroller. One model is used to predict air status and electrical power parameters, whereas another is employed to impute missing air pollution data. To the best of our knowledge, previous works on air quality prediction using tinyML have not specifically explored prediction and imputation tasks on a single microcontroller. Furthermore, this chapter explores model size reduction by implementing binary weights and suggests enhancing performance by applying meta-learning techniques.

The deep learning models discussed in Chapters 3, 4, and 5 are designed for execution on laptop/desktop computers or single board computers (SBCs). However, deploying these models to smaller devices, such as microcontrollers, requires additional steps to be conducted. This chapter explores the additional steps required to deploy deep learning models on smaller devices, such as microcontrollers, and delves into tiny machine learning (tinyML) implementations.

Figure 6.1 illustrates the tinyML development workflow, which includes creating the TensorFlow model and its final deployment on a microcontroller. Step 1 involves building the deep learning model using a laptop/desktop computer. During this process, it is crucial to consider the model's complexity and ensure it is compatible with the microcontroller's supported operations. For instance, at the time of writing, TensorFlow does not support 1D CNN operations on microcontrollers. To overcome this limitation, it is necessary to use 2D CNN operations instead. Therefore, in Step 1, 1D CNN should be avoided when designing the deep learning model. Once the model is trained and tested, it can be directly deployed to Single-Board Computers (SBCs).

In Step 2, the model obtained from Step 1 can undergo quantisation. This step, known as post-training quantisation, has been discussed in detail in Section 2.3.3. During Step 2, the quantised (optimised) model(s) is evaluated against

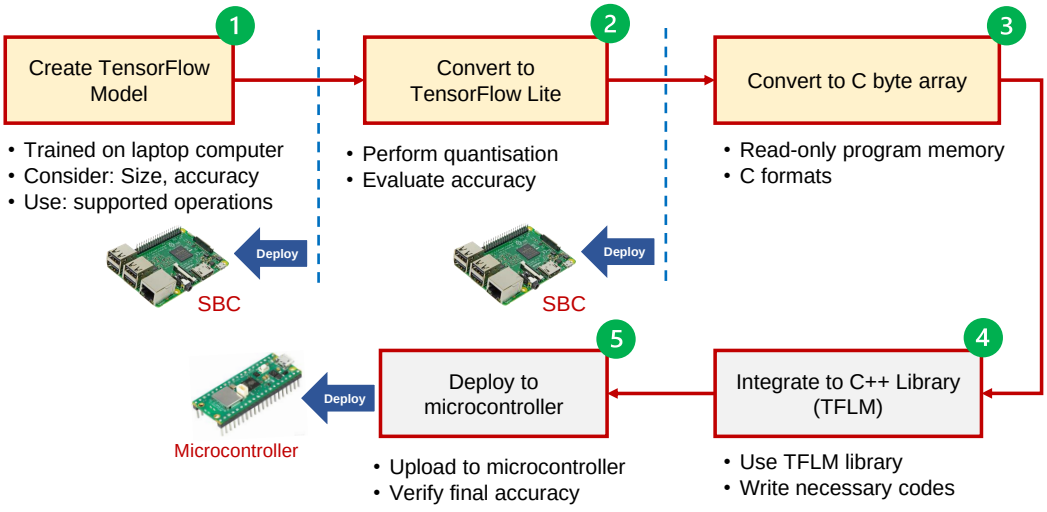


Figure 6.1: TensorFlow lite development workflow.

the original (unoptimised) model, considering the trade-off between model size, speed, and accuracy. The output of Step 2 is the TensorFlow Lite version of the model. These lightweight versions can also be deployed to SBCs.

Step 3 marks the beginning of preparation to deploy the selected model to the microcontroller. In this step, the model is converted to C-byte arrays. The process involves using the `xxd` command, which facilitates the creation of a hex dump from a file. The resulting C-byte array from Step 3 can be integrated into the microcontroller using the TensorFlow Lite for Microcontrollers (TFLM) library. In Step 4, additional codes are necessary to tailor the application to meet specific requirements. Finally, Step 5 involves verifying and uploading the codes to the microcontroller. It is crucial to perform an evaluation to compare the results obtained from the simulation with those obtained by inferring deep learning on the device.

6.2 Contributions

This chapter presents three applications of tinyML on microcontrollers. The first implementation focuses on predicting future values and imputing missing data using direct measurement data (Subchapter 6.3). The second application involves using a binary weight network to reduce the size of the TensorFlow Lite model (Subchapter 6.4). A low-cost air quality monitoring device is also developed to support these two implementations, utilising inexpensive sensors to measure air quality. Lastly, the third implementation explores an approach to enhance model predictions through meta-learning (Subchapter 6.5). The contributions of this chapter are as follows:

1. Creation of a low-cost air quality monitoring device capable of directly collecting air quality status. This device is powered by a solar panel and can gather electrical-related parameters.

2. Development of a dataset comprising air and electrical parameters, utilised for training and evaluating tinyML models, specifically for Subchapter 6.3 and Subchapter 6.4 implementations.

3. Introduction of novel tinyML models that run on microcontrollers, enabling prediction of future and missing values (Subchapter 6.3), reduction of model size using binary weights (Subchapter 6.4), and performance improvement through meta-learning techniques (Subchapter 6.5).

6.3 TinyML Low-cost Air Quality Monitoring Device

6.3.1 Motivation

The primary motivation of this study is to enhance a low-cost air quality device by incorporating intelligent capabilities. To achieve this, two distinct tinyML models were implemented on a single microcontroller. One model was designed to predict air quality and electrical power parameters, while the other focused on imputing missing air pollution data. Previous research on air quality prediction using tinyML has not explicitly investigated both prediction and imputation tasks on a single microcontroller, making this work a novel contribution to the field.

6.3.2 Data Collection and Preprocessing

This subchapter used a dataset derived from direct measurements to train and evaluate the tinyML models. The air quality monitoring device gathered data over approximately three months, from 21 July 2022 to 20 October 2022. The device was installed in a suburban area of Coventry, CV4 7BZ, UK, situated in front of the author's residence. During measurements, eight features are recorded, namely CO_2, air temperature (T_{air}), air humidity (RH_{air}), solar panel output current (I_{solar}), solar panel output voltage (V_{solar}), battery voltage (V_{batt}), battery temperature (T_{batt}), and battery capacity (C_{batt}). Table 6.1 shows the descriptive statistic of the features collected by the device from 21 July 2022 to 20 October 2022

The device collected air quality and power parameter data at 10-minute intervals, resulting in 13,080 rows of data by the end of the measurement period. To facilitate air quality and power parameter estimations, hourly averages were

Table 6.1: Descriptive statistics of direct measurement dataset.

	CO_2	T_{air}	RH_{air}	I_{solar}	V_{solar}	V_{batt}	T_{batt}	C_{batt}
count	4,402	4,402	4,402	4,402	4,402	4,402	4,402	4,402
mean	814.77	24.35	50.67	26.36	11.01	4.09	88.79	22.44
std	59.45	6.26	16.31	46.76	8.85	0.10	10.01	5.75
min	642.00	10.41	14.45	-0.60	0.42	3.67	5.70	9.30
25%	775.00	19.67	37.50	-0.10	0.46	4.03	83.30	18.00
50%	818.00	23.29	51.32	1.90	10.18	4.10	91.30	21.50
75%	854.00	28.20	64.90	39.30	21.37	4.16	97.30	26.40
max	1017.00	45.25	80.20	440.50	23.93	4.60	99.90	39.70

computed by aggregating every six measurements, generating a new dataset of 2,180 rows. However, for the purpose of missing data imputation, data gathered at 10-minute intervals were used.

Data is divided by allocating 70% of the dataset to the training set and the remaining 30% to the test set. To facilitate future parameter estimations, the features are standardised, resulting in a mean of zero and a standard deviation of one. On the other hand, for missing data imputation, the features are scaled to a range of [0, 1].

6.3.3 Device Design

Fig.6.2 illustrates the hardware interfaces of the low-cost air quality device. In this project, a Raspberry Pi (RPi) Pico W is employed as the primary controller board. The board utilises the RP2040 chip as its microcontroller. With a dual-core Cortex-M0+ processor, the board has 264kB of SRAM and 2MB of flash memory. The RP2040 chip provides versatile I/O, I2C, SPI, UART, and GPIO options. Additionally, the board is equipped with a single-band 2.4GHz wireless interface (802.11n) integrated onboard.

The air quality monitoring device is powered by two sources: a solar panel and a 18650 Li-Ion rechargeable battery. The solar panel, manufactured by Hisunage in China, has a nominal voltage of 12V and a power output of 20 Watts. It is used to recharge the 18650 Li-Ion battery, which has a nominal voltage of 3.6V and a capacity of 3500mAh. The Li-Ion battery utilised in this device is sourced from Samsung SDI.

A solar manager product from Waveshare Electronics in China is employed

to manage solar power. This solar manager is compatible with solar panels ranging from 6V to 24V and can recharge the 18650 Li-Ion battery. It also provides a regulated output of 5V/3A, suitable for supplying power to the RPi Pico board.

This study uses three sensor modules: INA219, LC709203F, and SCD41. A DS3231 real-time clock (RTC) module is also incorporated to ensure accurate timekeeping. The INA219 sensor module measures the solar output current and voltage. It allows for monitoring the current generated by the solar panel, which directly influences the battery's state of charge. The LC709203F module is employed to determine the battery cell capacity and cell voltage. It is also combined with a 10kΩ thermistor to measure the battery pack temperature. The SCD41 sensor module is responsible for detecting the carbon dioxide (CO_2) concentration in parts per million (ppm). Additionally, it measured temperature and relative humidity. The SCD4x series comprises miniature CO_2 sensors that leverage the photoacoustic NDIR sensing principle and Sensirion's proprietary technology. The following accuracy specifications characterise this sensor [248]: 400-1,000 ppm is ±(50 ppm + 2.5% of reading), 1,001-2,000 ppm is ±(50 ppm + 3% of reading), and 1,001-2,000 ppm is ±(50 ppm + 3% of reading). All the collected data are stored on a microSD memory card for further analysis and processing.

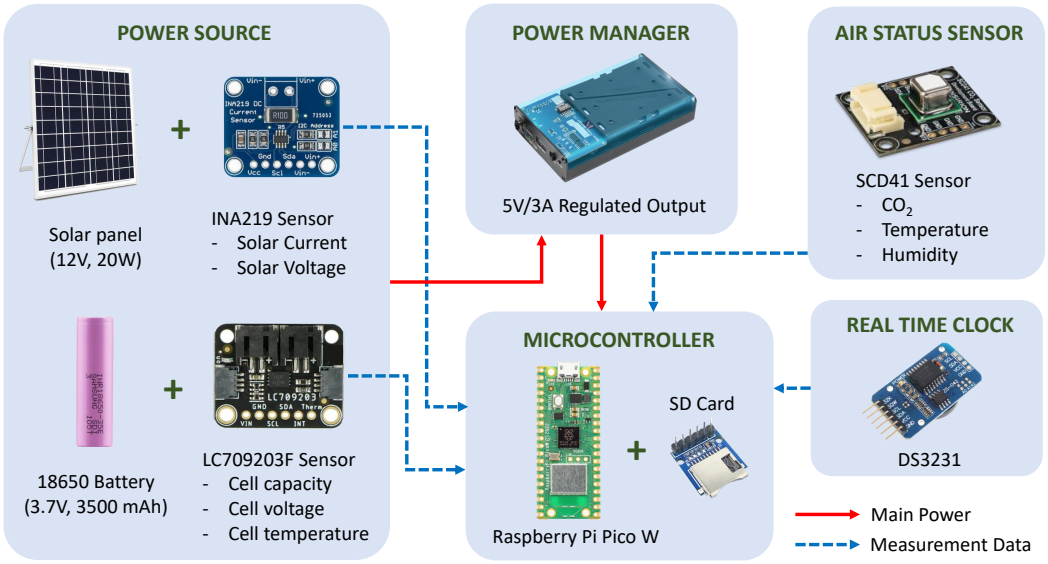


Figure 6.2: Module interfaces of the proposed device.

6.3.4 TinyML Framework

Two deep learning models, namely the *model predictor* and *model imputer*, are deployed on a single microcontroller. The model predictor performs the prediction task, while the model imputer is utilised for imputing missing sensor data. When a sensor fails to collect data in real-life applications, the microcontroller identifies this event as missing and initiates the data imputation process before executing the prediction task. During the data collection phase, no missing values are encountered, allowing the model predictor to be trained using the complete dataset. On the other hand, for the model imputer, measurement values with deliberately introduced missing values are removed at different levels. Finally, the prediction results obtained from the complete dataset and the dataset with missing values are compared to evaluate the effectiveness of the imputation process and its impact on the prediction performance.

The deep learning models in this work are constructed using TensorFlow (TF) 2.4.0, an open-source framework developed by Google specifically designed for deep learning applications [114]. The models are developed, trained, and evaluated using TF CPU, which runs on a desktop computer. Once the TF models are trained, they are converted to the TensorFlow Lite (TFLite) format using the TF Lite converter. Post-training quantisation techniques are applied to quantise the TFLite models to optimise the deployed model without compromising their accuracies. The quantised TFLite models are then converted into C-byte arrays and stored in the read-only program memory on the microcontroller. Inference on the microcontroller is performed using the TFLite for microcontrollers (TFLM) libraries. This process ensures that the deep learning models, originally built using TensorFlow, are efficiently deployed on the microcontroller for inference using the optimised TFLite format and TFLM libraries.

Inference on the testing data is conducted directly on the device. Since the testing data comprises numerous rows and multiple features, a microSD card is employed to store this data. The tinyML models are fed with input sets by reading the contents of the SD card, processing the data row-by-row, and performing on-device inference. This approach allows efficient and accurate evaluation of the tinyML models using the testing data.

6.3.5 Model Predictor and Model Imputer

The model predictor is designed to handle input sets consisting of eight features: CO_2, air temperature, air humidity, solar panel output current, solar panel output

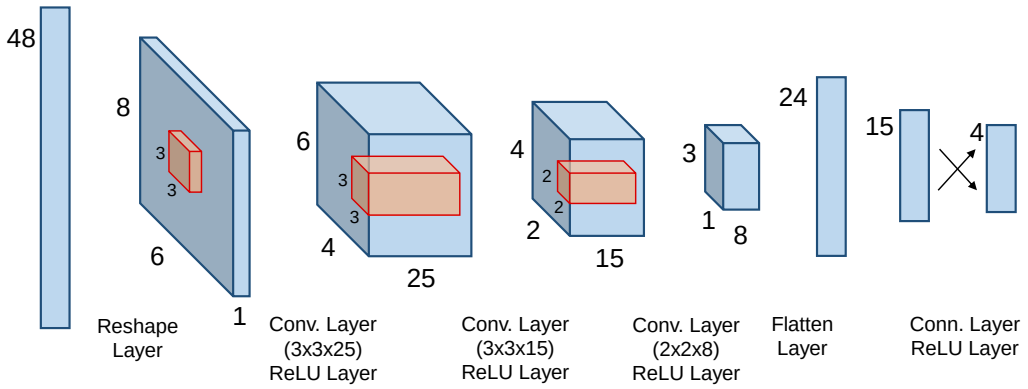


Figure 6.3: Model predictor architecture.

voltage, battery voltage, battery temperature, and battery capacity. It is specifically built to predict three air status features (CO_2, air temperature, and air humidity) and one electrical power feature (battery capacity). The prediction task focuses on short-term forecasting, specifically one hour into the future. The architecture of the deployed model for the prediction task can be seen in Figure 6.3.

The model predictor operates on input sets comprising eight features and six hours of historical measurements. To process this input, the data is flattened, resulting in an input size of 48. The input sets are then reshaped into three-dimensional data using 2D convolution layers for feature extraction. A fully connected layer with 15 units is employed as the prediction layer. This study uses rectified linear unit (ReLU) layers as the activation function throughout the model. However, for the last layer, no activation function is applied.

The model imputer utilised in this study is based on an autoencoder architecture, drawing inspiration from image denoising techniques [164]. The input sets with missing values are considered noisy inputs, and the autoencoder framework is employed to address this issue. This concept is also applied in Chapter 3.

In Chapter 3, a more complex implementation of this concept involved utilising spatiotemporal data from neighbouring air quality monitoring stations to predict missing values in the target station. However, this work develops a simpler lightweight model specifically tailored for a resource-constrained device. Local data is used to train the model, avoiding the need for incorporating spatiotemporal data from other air quality monitoring stations. The denoising autoencoder concept, which forms the basis of the model imputer, is illustrated in Fig. 6.4.

In this work, the proposed denoising autoencoder consists of several dense layers, as shown in Fig 6.5. All layers are fully connected, and rectified linear unit (ReLu) layers are used as the activation functions. Similar to the model predictor,

the model imputer also accepts input sets consisting of 8 features and 6 hours of measurement. Flattening this input, we get 48 as the input size.

6.3.6 Perturbation Method

To train and test the missing data estimation, certain measurement values are deliberately removed from the input sets, with each deleted value being replaced with zero. Following the approach of Hadeed *et al.* [68], four different missing rates (20%, 40%, 60%, and 80%) are selected for this study. As mentioned earlier, data was collected by the device every 10 minutes, but the model predictor operated on hourly average data. Therefore, it is assumed that missing data could occur at the 10-minute measurement level.

A single sensor can measure multiple parameters. For example, the LC709203F sensor measures three parameters: battery capacity, battery voltage, and battery temperature. It is assumed that if this sensor fails to perform a measurement, all these parameters will be unavailable. Consequently, during model training and testing, all parameters measured by the same sensor exhibit the same missing patterns. This assumption helps ensure consistency in handling missing data across multiple parameters measured by a single sensor.

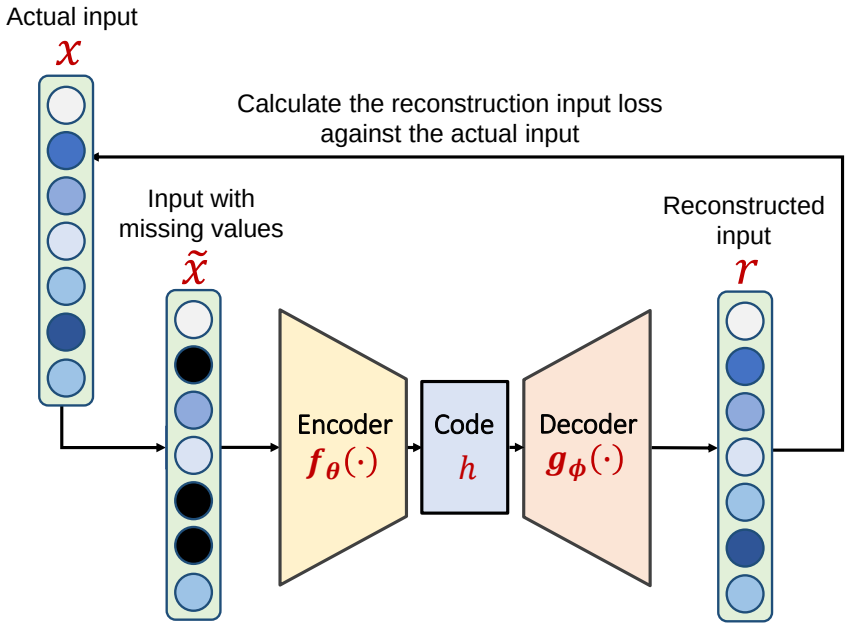


Figure 6.4: A denoising convolutional autoencoder workflow.

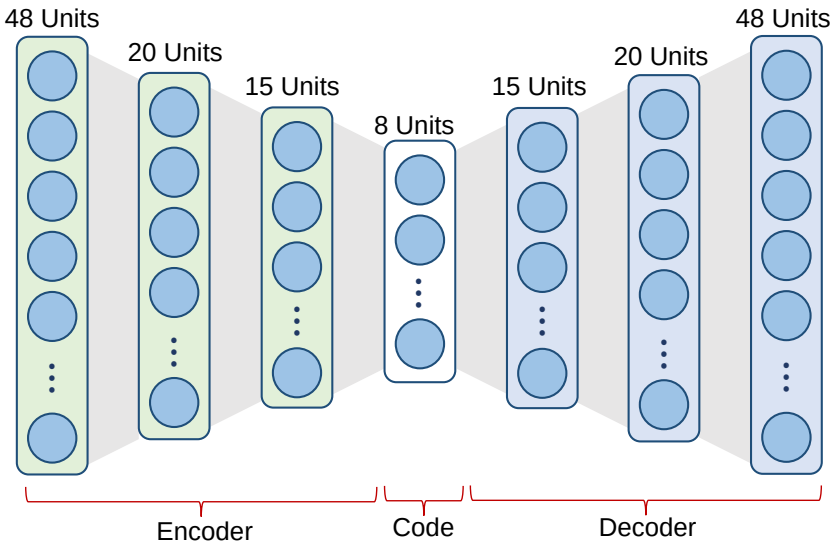


Figure 6.5: Model imputer architecture.

6.3.7 Device Realisation

The prototype of a low-cost air quality monitoring device, as depicted in Figure 6.6, was developed for this study. The figure illustrates the device along with the sensors and electronic modules used. The device was installed in a suburban area of Coventry city, UK, in front of the author's house.

6.3.8 Model Performance

A comprehensive evaluation was conducted on 648 test sets to predict the average 1-hour values of four features: CO_2, air temperature, air humidity, and battery capacity. The performance of the model predictor on testing data is presented in

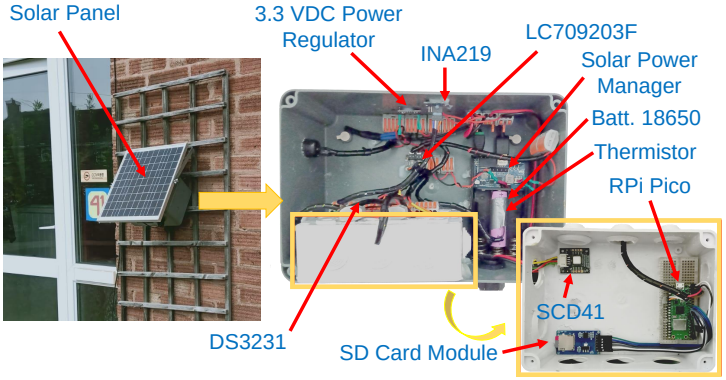


Figure 6.6: The low-cost air quality monitoring device.

Figure 6.7. It is important to note that this evaluation was performed using testing data containing no missing values.

The results indicate that the proposed model predictor can estimate each feature in the dataset with a coefficient of determination (R^2) above 0.70. Among the four features, the model exhibits the highest accuracy in predicting air humidity, achieving an impressive R^2 score of nearly 0.9. However, the model appears less sensitive in accurately predicting sharp declines in battery capacity. This observation suggests that further improvements could be made to enhance the model's performance in capturing abrupt changes in battery capacity. The model predictor operates at the hourly level, while the model imputer operates at the 10-minute measurement level. The air quality monitoring device collects data every 10 minutes, and the model imputer fills in any missing values at this level. By averaging every six measurements, hourly data without missing values is obtained.

To evaluate the effectiveness of the proposed imputation method, hourly-based assessments were conducted. Figure 6.8 illustrates the R^2 scores, which indicate how closely the recovered hourly data aligns with the hourly clean data. The model imputer was trained using training data containing 60% missing values, while the missing rates in the testing data ranged from 20% to 80%. Ten experiments were performed using different random seeds to account for various missing patterns.

The results depicted in Figure 6.8 demonstrate that the model effectively

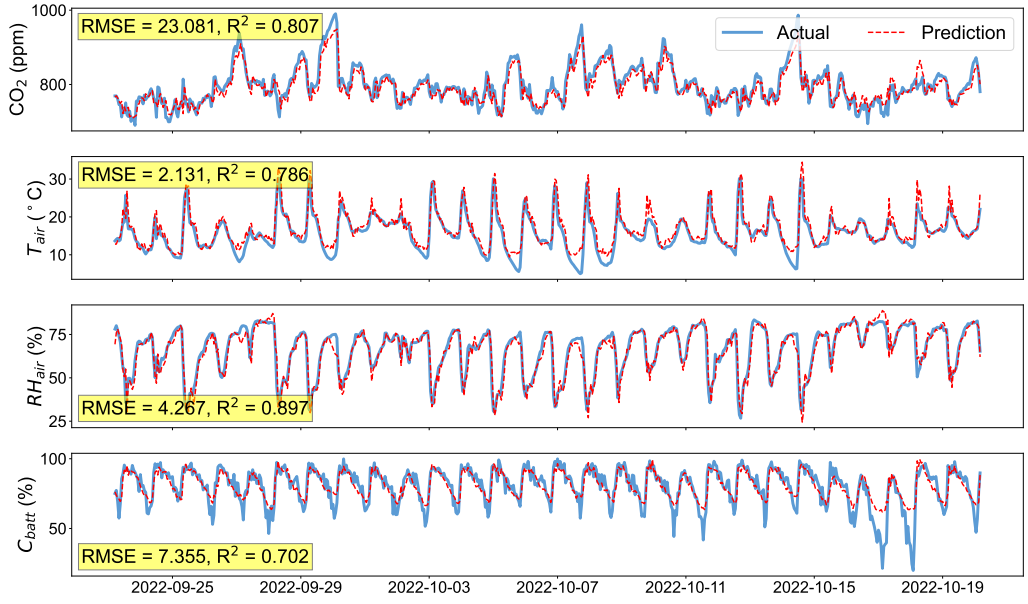


Figure 6.7: Performance of model predictor on testing data.

estimates the testing data with missing rates below 80%. Notably, air temperature, humidity, solar panel voltage, and battery temperature achieve exceptional accuracy, with R^2 scores surpassing 0.9, particularly at a missing rate of 40%. These findings highlight the model's proficiency in accurately estimating the target features, even in the presence of missing data.

6.3.9 Post-training Quantisation

The models' size is significantly reduced by converting the trained TensorFlow (TF) models to the TF Lite format. In this work, the model predictor size is reduced by 85.4 kilobytes, while the model imputer size is reduced by 102.6 kilobytes. These reductions are crucial for optimising the models for deployment on a tiny, resource-constrained device.

To achieve these size reductions, this work takes advantage of the post-training options offered by the TensorFlow framework. Considering factors such as the deep learning architecture, framework version, and microcontroller type, only the integer with float fallback quantisation technique is successfully implemented. This technique attempts to quantise the model into integers totally. However, float operators are still used when the model does not support integer applications. This quantisation technique further reduces the model size by 10.6 kilobytes for the model predictor and 13.6 kilobytes for the model imputer. This size reduction is crucial for efficiently utilising the device's limited resources. Table 6.2 comprehensively compares the model sizes before and after the conversion and quantisation processes.

Table 6.3 presents the Root Mean Square Error (RMSE) values obtained from different TF model formats. When the TF models are converted to TF Lite models (TFL), the model accuracies are preserved. However, some degradation in accuracy is observed after applying quantisation (TFL Q.). Among the features, the

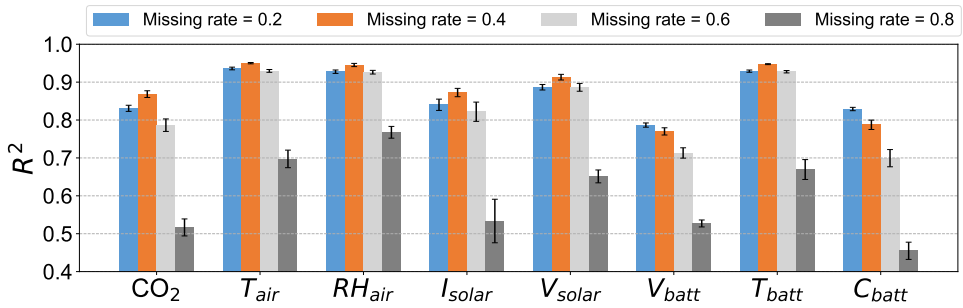


Figure 6.8: R^2 scores yielded from different missing rates.

Table 6.2: Comparison of different tinyML model sizes.

TinyML Model	**Model Predictor (bytes)**	**Model Imputer (bytes)**
TF	107,708	126,536
TF Lite	22,280	23,948
TF Lite Quantised	11,648	10,384

Table 6.3: RMSE values of different TF model formats.

	Model Predictor			**Model Imputer**		
Feature	**TF**	**TFL**	**TFL Q.**	**TF**	**TFL**	**TFL Q.**
CO_2(ppm)	23.081	23.081	23.392	26.591	26.591	33.743
T_{air}(°C)	2.131	2.131	2.458	1.176	1.176	1.338
RH_{air}(%)	4.267	4.267	4.927	3.447	3.447	4.524
I_{solar}(mA)	-	-	-	29.349	29.349	30.167
V_{solar}(V)	-	-	-	3.203	3.203	3.361
V_{batt}(V)	-	-	-	0.099	0.099	0.101
T_{batt}(°C)	-	-	-	1.02	1.02	1.237
C_{batt}(%)	7.355	7.355	7.909	12.393	12.393	12.803

CO_2 imputation shows the most significant degradation, with the RMSE increasing from 26.591 ppm to 33.743 ppm. The RMSE degradations for the other features are relatively less significant.

6.3.10 Summary

This work focused on developing a low-cost air quality monitoring device with tiny machine learning (tinyML) models to enhance its capabilities. The device utilised multiple tinyML models deployed on a single microcontroller. These models employed 2D CNN layers and a denoising autoencoder architecture to facilitate parameter prediction and missing feature imputation tasks. The proposed model predictor exhibited a coefficient of determination above 0.70 for estimating testing data, while the model imputer performed well when missing rates below 80%. The quantised versions of the models showed a decrease in size compared to their original lite models, with reductions of 47.7% and 56.6% for the model predictor and model imputer, respectively, while maintaining relatively high accuracies.

6.4 Optimising TinyML with Binary Weight Network

6.4.1 Introduction

In Subchapter 6.3, the workflow for implementing tinyML is depicted in Fig. 6.1. However, an extra step is introduced in this section. This section generates an additional binary version of the TF Model's weights, denoted as **Step 1b** in Fig. 6.9. This binary version is called *Binary Weight Network* (BWN).

In Step 1a, a deep learning model is developed using standard TensorFlow procedures. This model is trained and tested to achieve optimal performance in its full-precision formats. In Step 1b, the quantised version of the model is created using the Larq library and TensorFlow framework. Both the model from Step 1a and the quantised model from Step 1b undergo the same subsequent processes.

6.4.2 Objectives

The primary objectives of this study encompass the following:

- To enhance the capabilities of a low-cost air quality monitoring device by integrating a tiny machine learning model.
- To optimise the proposed tiny machine learning model implemented on a microcontroller, employing techniques such as binary weight network (BWN).

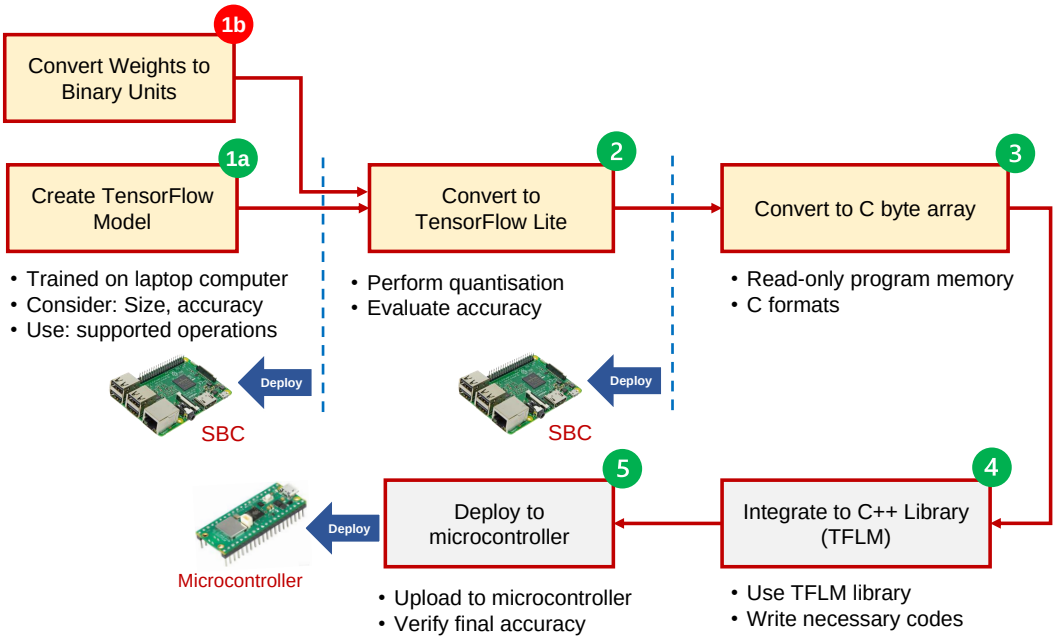


Figure 6.9: BWN development workflow.

- To compare the performance of different versions of tiny machine learning models.

6.4.3 Binary Neural Network

Courbariaux *et al.* [249] developed the Binary Neural Network (BNN) methodology, serving as a foundational framework for developing numerous subsequent network binarisation techniques [250]. BNNs are a particular type of Quantised Neural Networks (QNNs) in which the quantisation output is binary. The quantisation output x_q is binary:

$$x_q = q(x), \qquad \text{where } x_q \in \{-1, +1\}, x \in \mathbb{R} \tag{6.1}$$

The forward pass uses the `sign` quantisation function to convert the activations and latent full-precision weights into binary values. However, this process results in nearly zero gradients across most regions, making it challenging for the model to learn effectively. Consequently, the model faces difficulties updating its weights during training, limiting its overall learning capabilities.

$$q(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ -1, & \text{otherwise.} \end{cases} \tag{6.2}$$

The Straight-Through Estimator (STE) technique is utilised during model training to estimate the gradient [251]. This technique replaces the binarisation process with a clipped identity operation during the backward propagation. This approach enables efficient gradient computation with binarised weights, ensuring smooth model training.

$$\frac{dq(x)}{dx} = \begin{cases} 1, & \text{if } |x| \leq 1, \\ 0, & \text{if } |x| > 1. \end{cases} \tag{6.3}$$

6.4.4 Layer Quantisation

Figure 6.10 depicts the computational graph of a quantised layer, where the kernels and inputs can be independently quantised. The output y can be written as [252]:

$$y = f(\sigma(q_{input}(x), q_{kernel}(w)) + b) \tag{6.4}$$

As kernels and inputs can be independently quantised, three terms can be described as follows [252]:

- **Binary Weight Network (BWN)**: in the case where only the kernels are quantised.
- **Binary Activation Network (BAN)**: only the inputs are quantised.
- **Binary Neural Network (BNN)**: both inputs and kernels are binarised in a network.

Following extensive experimentation, it was observed that both Binary Neural Networks (BNNs) and Binary Activation Networks (BANs) yielded inaccurate prediction results. Consequently, for the specific case of air pollution prediction in this study, only the Binary Weight Network (BWN) was applicable. Hence, BWN has been chosen for this work.

6.4.5 Proposed Model

This study binarises the weights while the bias and activation are maintained in full precision. The first and last layers are retained in their original full precisions, as shown in Fig.6.11. TensorFlow 2.12 was used to construct the deep learning model, and the Larq library [251] was employed to train neural networks with exceptionally low precision weights.

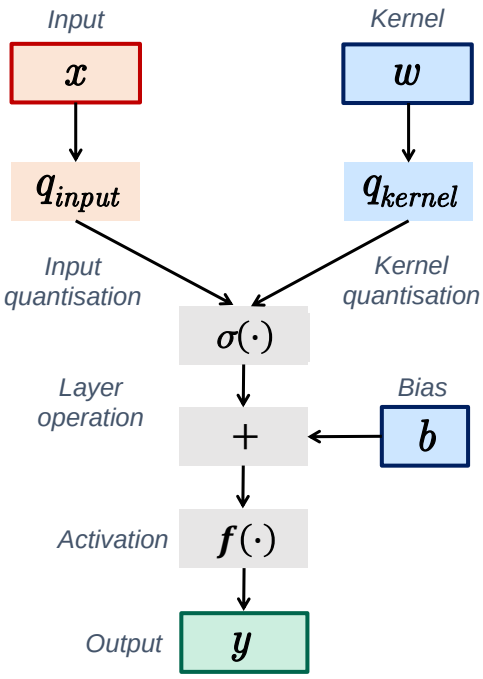


Figure 6.10: Computational graph of layer quantisation.

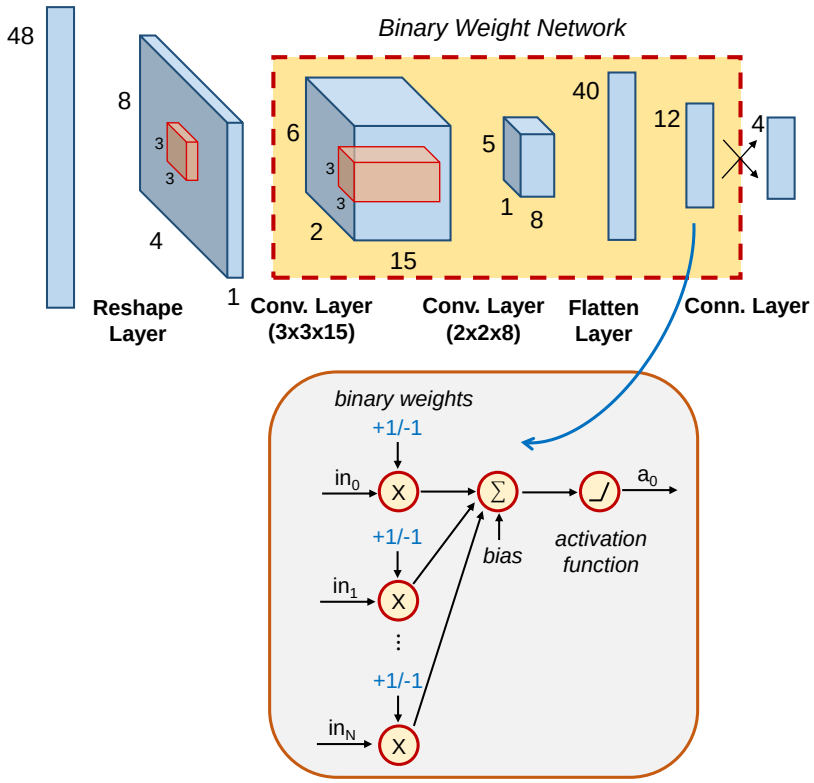


Figure 6.11: Proposed model with binary weight section.

6.4.6 Research Workflow

The research framework is shown in Fig. 6.12. Data from the air quality monitoring device were divided into training and testing sets. The target labels included CO_2 levels, air temperature, humidity, and battery capacity. Two models were developed: a full precision model and a BWN model. Both models underwent the same steps in the process. Once trained, the models were evaluated and optimised using TF Lite. Additionally, the models can be quantised to 8-bit (with fallback) precision. Finally, the chosen model was deployed to the microcontroller.

6.4.7 Data Collection

This section employs the same device used in Subchapter 6.3. However, in this section, the training and testing sizes are larger. The training data consists of information collected between 21 July 2022 and 20 October 2022, while the testing data was obtained from 15 March 2023 to 5 June 2023. The measurements recorded eight features, just like in the previous section. These features include CO_2 levels, air temperature, air humidity, solar panel output current, solar panel output voltage,

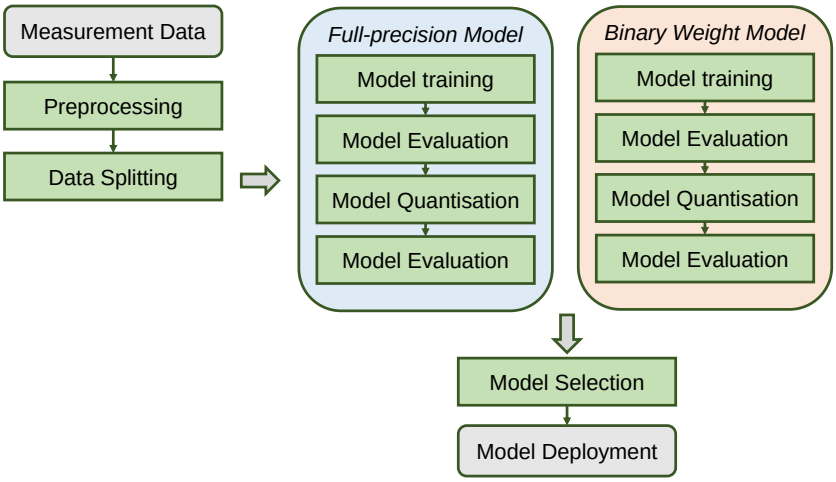


Figure 6.12: BWN implementation workflow.

battery voltage, battery temperature, and battery capacity. Moreover, the sampling period remained the same, i.e., every 10 minutes. The data were then averaged every six measurements to derive hourly data. All features were standardised by removing the mean and scaling to unit variance.

6.4.8 Quantisation Results

Converting a trained TensorFlow model to the TensorFlow Lite format offers the benefit of reducing the model size. This conversion applies to both the full precision and BWN models, as shown in Fig. 6.12. In this work, performed the integer with float fallback quantisation is conducted.

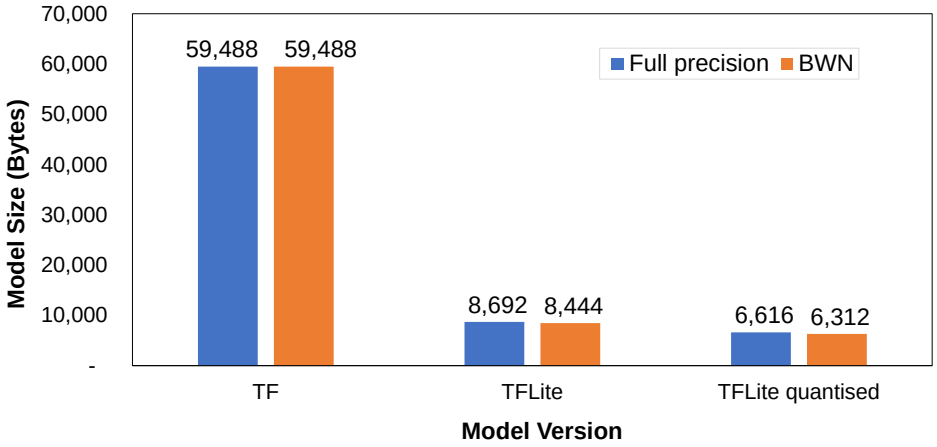


Figure 6.13: Model size comparisons.

Table 6.4: RMSE predictions obtained from different model versions.

	Full Precision			BWN		
Feature	TF	TFL	TFL Quant'd.	TF	TFL	TFL Quant'd.
CO_2(ppm)	30.113	30.113	29.911	28.851	28.851	28.980
T_{air}(°C)	1.665	1.665	2.213	2.637	2.637	2.922
%H	4.302	4.302	5.514	4.471	4.471	5.430
%Batt	10.352	10.352	12.631	8.755	8.755	10.486

The conversion of TF models to TF Lite models preserves the accuracy of the models. As depicted in Figure 6.13, both model formats initially have the same sizes since they utilise 32-bit floating-point precision. However, some memory space can be saved by converting the binary weights to C-byte arrays. Specifically, in the TFLite version, converting the original model to Binary Weight Networks (BWN) can result in memory savings of approximately 248 Bytes. Similarly, for TFLite quantised models, Binary Weight Masks (BWM) can save around 304 bytes. Although these reductions may seem small, they hold significant value for devices with limited resources.

Table 6.4 presents the RMSE values obtained from various TensorFlow model formats. The trained model was designed to predict multiple targets. It is important to note that certain targets in the Binary Weight Networks (BWN) format may exhibit slightly lower accuracy compared to the original version. However, in the case of predicting CO_2 levels and battery capacity, the BWN format performs better than the original version for both TFLite and TF Lite quantised models.

Fig. 6.14 illustrates the performance of the selected tinyML model, which is a Binary Weight Network (BWN) model without any applied quantisation. Subsequently, the selected model was deployed to the microcontroller. For this study, we used the Raspberry Pi Pico W as the target device for tinyML deployment. Fig. 6.14 demonstrates that larger distortions are observed in the predictions of battery capacity. Specifically, the estimated values are higher than the observed values, particularly for low battery capacities.

6.4.9 Summary

This work proposes an optimisation technique for deep learning models in conjunction with the standard TensorFlow Lite method. By preserving the first and last layers and applying binary quantisation to the weights of all intermediate layers, an additional reduction in size can be achieved compared to the TFLite model alone.

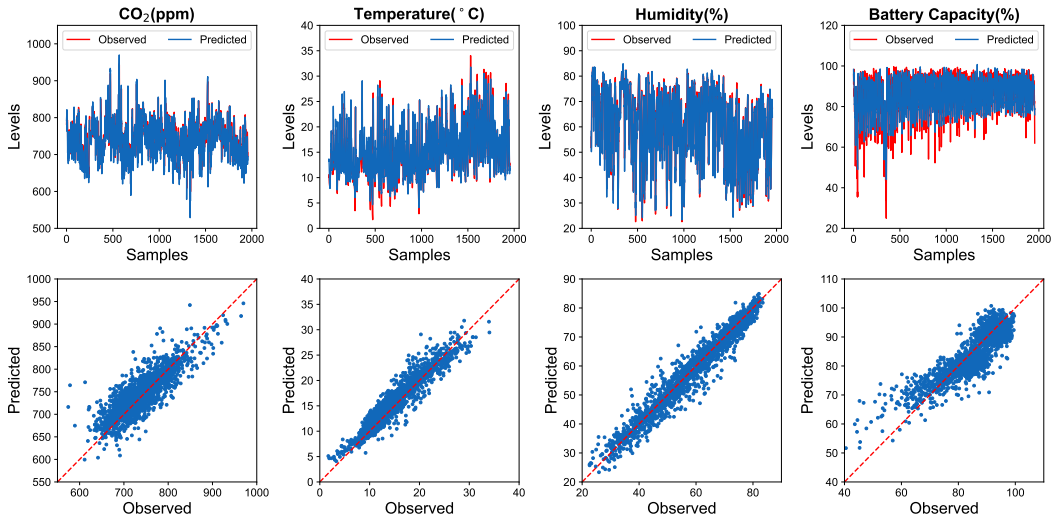


Figure 6.14: Deployed BWN performance.

This approach, known as Binary Weight Network (BWN), potentially reduces the file size of the final model intended for deployment on a microcontroller. The impact on model accuracy depends on the architecture, as BWN has the potential to improve accuracy. However, it is important to note that there may be a slight degradation in performance for certain target features after the binarisation process.

6.5 TinyML with Meta-Learning

6.5.1 Introduction

Meta-learning involves consolidating knowledge gained from multiple learning episodes and leveraging this knowledge to enhance future learning performance [253]. This thesis adopts the approach depicted in Fig. 6.15 to implement tinyML using a meta-learning approach. The meta-learning concept introduced in this section offers an alternative approach to enhancing conventional model performance with minimal effort and requiring only a small amount of device memory. In this chapter, we introduce a simple linear regression. By incorporating additional constant values after the model output, it is possible to potentially enhance performance. The process consists of several steps. In Step 1a, multiple individual deep learning models are created and assessed. These individual models serve as the base models, which can vary in their architectural designs. Although it is possible to employ multiple base models, this thesis focuses on using only two base models to minimise the memory requirements of the final device, i.e., the microcontroller.

In Step 1b, the meta-learner is introduced, and its performance is evaluated. This thesis uses a straightforward meta-learner known as Ordinary Least Squares (OLS) Linear Regression (LR). It is important to note that the LR is distinct from the TensorFlow process path. Consequently, the LR's coefficients obtained through the simulation process are subsequently embedded in the microcontroller programming stage, denoted as Step 5a in Fig. 6.15.

6.5.2 Objectives

This research aims to enhance tinyML models deployed on microcontrollers by applying a meta-learning approach for predicting hourly air pollutants. Using a stacking ensemble architecture, the meta-learner assimilates knowledge from individual base models to enhance the final prediction. This study demonstrates that a simple step can be implemented to enhance the performance of individual TensorFlow models. Performance improvements can be achieved by appending linear regression coefficients at the end of each base model.

6.5.3 Air Quality Dataset

In contrast to Subchapter 6.3 and Subchapter 6.4, the data used in this section is sourced from a publicly available dataset. Air quality data from five monitoring sites in the Greater London area between 1 July 2019 and 13 December 2021

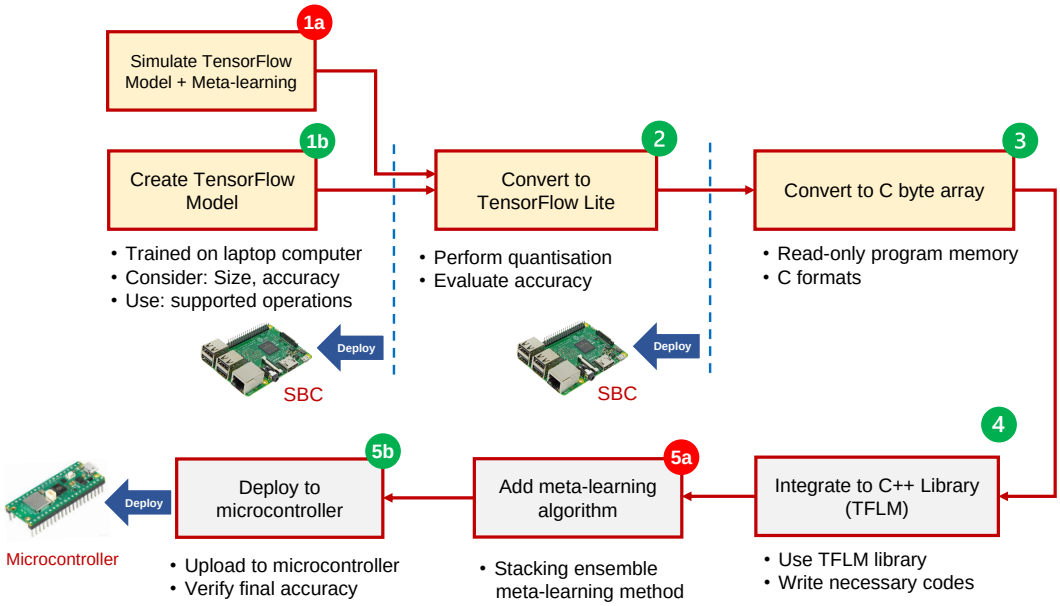


Figure 6.15: Meta development workflow.

were collected using the Openair framework [186]. These sites are London Bexley (BEX), London Westminster (HORS), London N. Kensington (KC1), London Eltham (LON6), and London Marylebone Road (MY1). Seven features were selected as inputs (NO_x, NO_2, NO, $PM_{2.5}$, modelled wind speed, wind direction, and air temperature) to predict two hourly pollutants (NO_2 and $PM_{2.5}$).

This work uses 80% of the data for the training set and 20% for the test set. All features in the dataset are normalised to the range of [0,1], and missing values are filled using a multivariate imputer function provided by scikit-learn. In this strategy, the library uses a method to fill in missing values that involves modelling each feature that has missing data based on other features in a round-robin manner [254].

6.5.4 Stacking Ensemble Process

Figure 6.16 shows the stacking ensemble concept. The stacking architecture consists of two base models in Level-0 and a least squares linear regression as the meta-learner in Level-1. A detailed process outlining the steps to develop a stacking ensemble model is illustrated in Fig. 6.17

The individual base model was trained separately to obtain the best performance, and all trained models were then saved. Subsequently, the layers of each selected base model were frozen, preventing further training. A meta-learner was then added on top of the base models. During the stacked model's training, the base models' weights and biases remained unchanged, ensuring that the learned features were preserved. The only parameter adjusted was the linear regression coefficients, optimised to minimise the residual sum of squares between the input and target sets, resulting in a linear approximation.

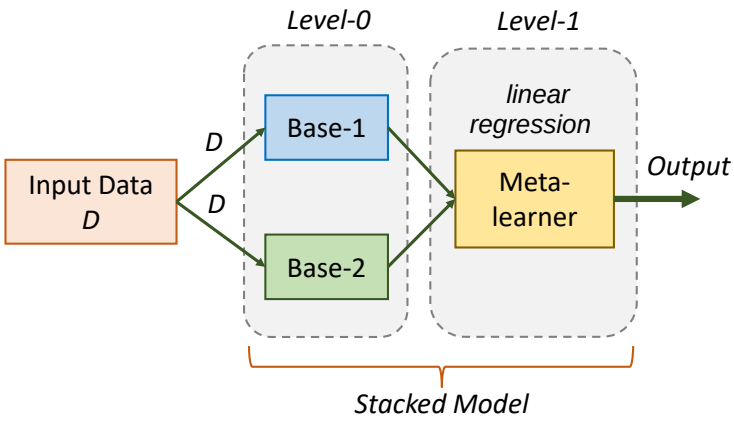


Figure 6.16: Stacking ensemble architecture.

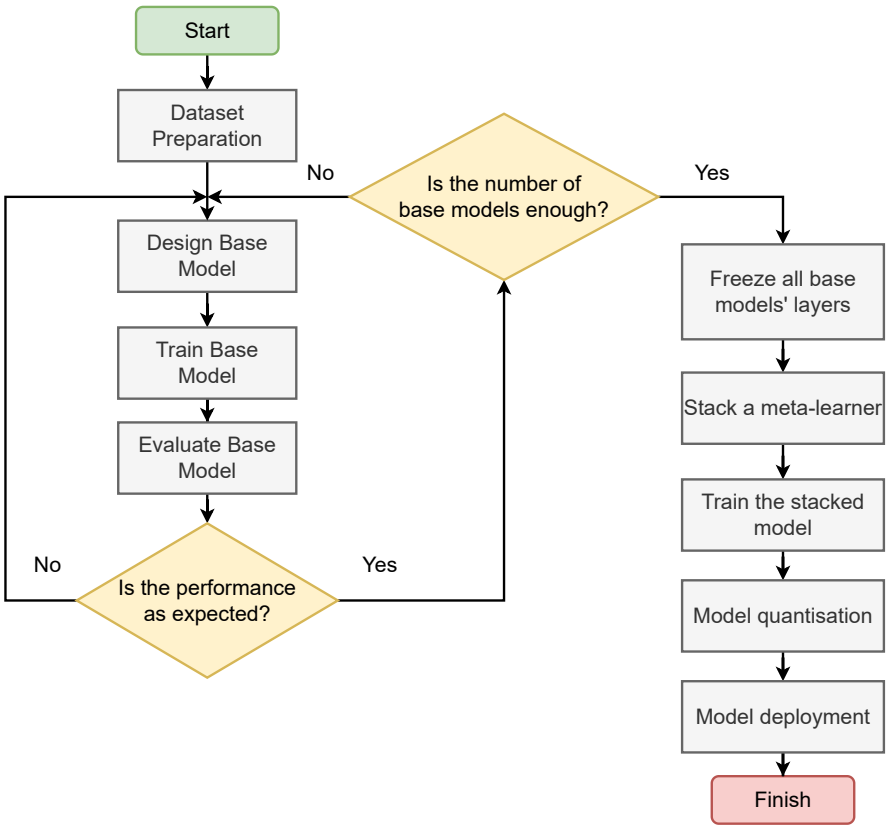


Figure 6.17: Flowchart of deploying stacking ensemble meta-learning model.

Once the meta-learner is trained, the base models undergo quantisation using standard TensorFlow procedures. Subsequently, the final LR coefficients are manually incorporated during the programming of the microcontroller.

6.5.5 Proposed Stacking Ensemble Model

Figure 6.18 illustrates the proposed base models, which include dense layers as Base-1 and a combination of 2-D CNN and dense layers as Base-2. These distinct architectures were intentionally chosen to examine the capabilities of each as a base model. The rectified linear activation function (ReLU) is applied to all layers except for the output layers. This study uses the TensorFlow framework version 2.12 to develop the ML models and deploy the lite versions on a Raspberry Pi Pico microcontroller.

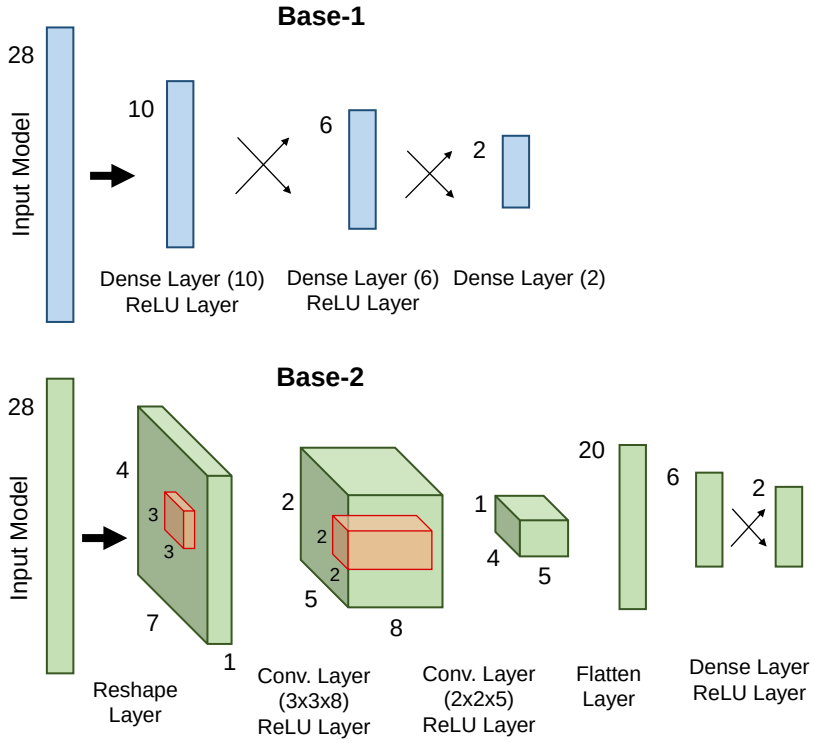


Figure 6.18: Proposed base models.

6.5.6 Results and Discussion

The meta-learner receives inputs from the base models, which consist of two base models. Each base model generates two predictions (NO_2 and $PM_{2.5}$). As a result, the meta-learner receives four values from the base models. The linear model has the following form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n \tag{6.5}$$

where y is the meta-learner output, β_0 is intercept, $\beta_1, \beta_2, \beta_3 \ldots \beta_n$ are the linear regression coefficients, and $x_1, x_2, x_3 \ldots x_n$ is the base model outputs. The values of $\beta_0, \beta_1, \beta_2, \ldots \beta_n$ vary between NO_2 and $PM_{2.5}$. Fig. 6.19 depicts the process of acquiring the meta-learner output.

As depicted in Fig. 6.19, each base model generates two outputs: one for predicting NO_2 (x_1 and x_3) and another for $PM_{2.5}$ (x_2 and x_4). During training, the meta-learner calculates intercepts and linear regression coefficients for each pollutant. Since the total input for the meta-learner is four, there are also four linear regression coefficients ($\beta_1, \beta_2, \beta_2$, and β_4), in addition to one intercept β_0. Further-

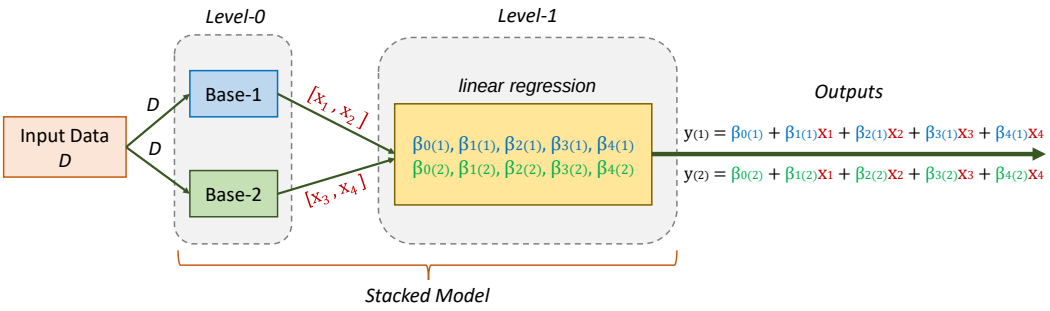


Figure 6.19: Process of acquiring the meta-learner output.

more, since there are two different target pollutants, the values of $\beta_0, \beta_1, \beta_2, \beta_3$, and β_4 also differ for each pollutant. As shown in Fig. 6.19, the coefficients $\beta_{0(1)}$, $\beta_{1(1)}$, $\beta_{2(1)}$, $\beta_{3(1)}$, and $\beta_{4(1)}$ are specific to NO_2, while coefficients $\beta_{0(2)}$, $\beta_{1(2)}$, $\beta_{2(2)}$, $\beta_{3(2)}$, and $\beta_{4(2)}$ are dedicated to $PM_{2.5}$.

Based on the training results, the meta-learner outputs can be represented by the following equations:

$$NO_2 = 0.00470543 + 0.18501297x_1 + 0.08231181x_2 + 0.80033445x_3 - 0.08300869x_4 \tag{6.6}$$

$$PM_{2.5} = 0.11539364 - 0.06124004x_1 + 0.67543256x_2 + 0.06094074x_3 + 0.31981403x_4 \tag{6.7}$$

The values $\beta_{0(1)} \ldots \beta_{4(1)}$ and $\beta_{0(2)} \ldots \beta_{4(2)}$ are manually incorporated during the microcontroller programming. This process is straightforward and requires a small amount of memory space.

Table 6.5 presents the model performance. The table shows that the stacked model can reduce RMSE values obtained by individual base models. The linear approximation by the meta-learner finds the best way to combine the Level-0 members' outputs by minimising the residual sum of squares between the input and target sets.

In this work, converting ML models to lite versions can reduce the model size by about 83% and 77% for Base-1 and Base-2, respectively. The deployed tinyML model sizes are 3,012 bytes for Base-1 and 5,076 bytes for Base-2, considered light enough even without performing any quantisation techniques. The lite version's accuracy is not degraded compared to the original model's.

Table 6.5: RMSE values of base and stacked models.

	$\mathbf{NO_2}$ ($\mu g/m^3$)			$\mathbf{PM_{2.5}}$ ($\mu g/m^3$)		
Station	**Base-1**	**Base-2**	**Stacked**	**Base-1**	**Base-2**	**Stacked**
BEX	5.460	5.451	**5.374**	2.702	2.847	**2.597**
HORS	5.435	5.416	**5.401**	3.431	3.679	**3.234**
KC1	5.136	5.073	**4.962**	1.656	1.662	**1.609**
LON6	4.530	4.447	**4.308**	2.147	2.185	**1.873**
MY1	8.826	8.979	**8.759**	2.556	2.583	**2.526**

6.5.7 Summary

In this implementation, a stacked ensemble model architecture based on meta-learning is proposed to enhance the accuracy of tinyML predictions. The meta-learning approach, on average, improves the predictions of the base models across all stations and pollutants by approximately 4.4%. Implementing meta-learning is straightforward and requires minimal memory usage by employing linear regression on top of the base models. However, it is important to note that memory consumption is influenced by the number of base models involved in the meta-learning process. Therefore, keeping the number of base models small is recommended to minimise memory usage.

Chapter 7

Conclusions and Further Work

7.1 Overview

Chapter 1 discussed that air pollution has emerged as a significant global threat to public health. In the pursuit of environmental sustainability, numerous stakeholders have developed air pollution monitoring systems to measure, analyse, and predict the concentration levels of air pollutants. Recent research has demonstrated the feasibility of employing low-cost sensor nodes in air quality monitoring systems. These sensor-based monitoring systems provide high-density spatiotemporal pollution data. The rapid deployment of these sensors has resulted in a substantial increase in data volume. Machine learning techniques have great potential in leveraging this wealth of data in air quality research. The future of machine learning is shifting towards edge computing, which addresses challenges related to latency, privacy, and scalability commonly encountered in cloud-based systems. In the field of air quality research, there is a growing demand for denser spatiotemporal data on pollutant levels from communities. Traditional industrial-grade instruments, while reliable, are often expensive and challenging to install due to their size. As a viable alternative, low-cost sensor instruments are gaining traction. Furthermore, machine learning is becoming increasingly ubiquitous, even at the edge. We anticipate that more communities will adopt intelligent sensing and prediction techniques using low-cost air quality devices. Finally, this chapter encompassed the research objectives and the organisation of the thesis.

Chapter 2 delved into the research background by highlighting the evolution of air pollution monitoring systems. Many stakeholders have started developing these systems to measure air pollutants effectively and enhance environmental sustainability. This evolution has seen a shift away from the traditional reliance on

standard, government-managed networks towards the incorporation of reference-level monitors and emerging sensor technologies. Later, this chapter introduced the concept of edge computing, which emphasised applying computational capabilities near the data source, including the possibility of running machine learning at the edge to improve efficiency and real-time processing. In addition, this chapter extensively explored machine learning platforms and edge devices used in this thesis.

One area where machine learning finds application in air quality research is missing data imputation, which was discussed in **Chapter 3** of the thesis. Using a denoising autoencoder model, a novel imputation method is proposed to enhance temporal and spatial data accuracy. The model has the capability to predict missing air quality data for both short and long-consecutive time intervals. The proposed method has been tested on air quality data from Delhi, London, and Beijing. The model demonstrates satisfactory performance across these diverse datasets. Specific values were intentionally omitted from the data to evaluate short interval prediction, encompassing four distinct missing rates (20%, 40%, 60%, and 80%). For instance, at a 20% missing rate, the R^2 scores exceed 0.8 for all target stations. Generally, a decrease in missingness levels correlates with lower RMSE/MAE values and higher R^2 scores. The procedure involves the removal of all data at the target station for a designated timeframe to predict missing values over longer consecutive intervals. The results indicate that the imputed values successfully capture the underlying dynamics of the actual values. The proposed autoencoder model exhibits adeptness in recognising and filling the gaps left by missing data. However, it is important to note that the level of correlation coefficients between paired stations can influence the performance of the proposed method. The imputed values are notably biased when stations exhibit extremely low correlation coefficients. Currently, the study employs Pearson's correlation coefficient, which assesses the linear correlation between pollutant data from two stations. An alternative approach could involve implementing a non-linear correlation method to identify more robust neighbouring stations for inclusion in the analysis, such as Spearman's rank correlation coefficient and Kendall's rank correlation coefficient. Another potential step is to address missing data by using values other than zero when developing the deep learning model. Instead of replacing missing values with zeros, alternative strategies could be explored, such as using the most frequent values or employing interpolation techniques. Adopting different strategies for filling in missing data could significantly alter the patterns within the input dataset. Moreover, the proposed autoencoder model outperforms commonly used univariate imputations in handling missing data, resulting in root mean square error improvement rates of approximately 50% to 65%, and about 20%

to 40% for multivariate imputation.

Optimising efficient design becomes crucial when deploying deep learning models on edge devices. This topic is specifically discussed in **Chapter 4**. This chapter is devoted to the design of a novel hybrid CNN-LSTM model for accurately predicting $PM_{2.5}$ pollutant levels using spatiotemporal features. The chapter discussed prior works in deep learning for air quality predictions. Subsequently, the chapter delved into the dataset and preprocessing techniques employed in this study. The feature selection was explained, and the proposed model was discussed. This model comprises two parallel inputs: the first aggregates data exclusively from the local node, while the second collects $PM_{2.5}$ data from the local and its neighbouring nodes. The evaluation phase encompasses a comprehensive exploration of 20 distinct deep learning models. Notably, integrating a deeper model with CNN layers as a feature extractor preceding the predictor (ANN, RNN, LSTM, or GRU) yields marginal enhancements in model performance. The subsequent endeavour involves optimising and deploying the proposed deep learning model onto edge devices. To this end, two Raspberry Pi boards are selected: the RPi4B and RPi3B+. Raspberry Pi boards enjoy widespread popularity, but they are not the exclusive option for low-cost air quality monitoring stations. A plethora of other single-board computers (SBCs) exist in the market. Additionally, many edge devices opt for microcontrollers as an alternative to SBCs. Despite their comparatively modest computing power, microcontrollers provide robust sensor interfacing capabilities, boasting features like built-in analog-to-digital converters (ADCs) within their chips. Moreover, modern microcontrollers support various communication protocols and offer additional storage peripherals, all at generally more budget-friendly prices compared to SBCs. In this study, the RPi4B showcases significant advantages, proving to be twice as fast in all experiments compared to the RPi3B+. Post-training quantisation techniques are explored in pursuit of further reductions in both size and speed. Four distinct post-training optimisation approaches are examined: dynamic range quantisation, float16 quantisation, integer quantisation with float fallback, and full integer-only quantisation. While dynamic range quantisation leads to an approximate size reduction of 47%, its impact on execution time improvement is minimal. The selection of the quantisation technique should be based on the specific priorities and trade-offs users intend to strike between model accuracy, size reduction, and execution time improvement.

Chapter 5 investigated the practical implementation of collaborative learning techniques for air quality prediction on edge devices. The fundamental premise of this chapter is to take advantage of the spatial and temporal correlations embedded

in air quality data collected at various monitoring stations. By leveraging these spatiotemporal dynamics through collaborative learning between sensing devices, the performance of machine learning models can be significantly improved. Contributions to this chapter include an introduction to innovative methodologies that leverage spatiotemporal data (SpaTemp), shared deep learning models (ClustME), and widely used collaborative learning techniques (FedAvg). The experimental framework is conducted by a comprehensive setup involving eight air quality monitoring stations, each represented by three different Raspberry Pi (RPi) board variants, along with a 2GB NVIDIA Jetson Nano Developer Kit. The results show that the SpaTemp method surpasses other approaches in minimising the loss function during the training process at all participating stations and performs better than other collaborative learning methods, with RIR values ranging from 0.525% to 8.934%. Regarding the Learning execution timeframe, the Jetson Nano 2GB developer kit outperforms other devices, with turnaround times up to approximately nine times faster than Raspberry Pi Zeros when executing the SpaTemp method. ClustME reduces communication costs by up to half compared to FedAvg. Finally, this chapter broadens the scope of the research by providing valuable insights into the potential of extending edge device networks for a wider range of applications.

Chapter 6 was dedicated to tinyML experiments. Deploying models into resource-constrained devices, such as microcontrollers, is challenging. This necessitates the development of tiny machine learning (tinyML) models that are both compact and efficient. This chapter explored the implementation of a low-cost air quality monitoring device designed to acquire air quality information directly. This device uses a solar panel to gather air quality and electrical parameters. Furthermore, the chapter delved into creating a dedicated dataset involving direct measurements. This dataset is a valuable resource for training and assessing tinyML models. In this chapter, three experiments around the implementation of tinyML were discussed. The first experiment leverages tinyML models for predicting the future and imputing the current missing values. The second experiment was about the reduction of model size using binary weights. Finally. the last experiment was about performance improvement through meta-learning techniques.

7.2 Objectives and Achievements

Chapter 1.2 outlined four main objectives of this thesis. In this section, these objectives are presented along with the corresponding achievements accomplished in this thesis.

- **Objective 1**: To develop a method for imputing missing values on measurement data, considering spatiotemporal behaviour of air quality status.

  Achievement: A convolutional denoising autoencoder architecture was employed to develop a method for imputing missing data in air quality datasets. Three distinct air quality datasets were used, with varying pollutants targeted to assess the performance of the proposed method. The method utilised spatiotemporal data from neighbouring stations to assist in filling in the missing data at the target stations. Both short-term and long-term consecutive missing data were selected to evaluate the effectiveness of the proposed model. The results demonstrate that the proposed method outperforms commonly used imputation methods, including univariate and multivariate approaches.

- **Objective 2**: To develop a deep learning model to predict air pollution levels accurately with model optimisations for edge devices.

  Achievement: To predict air pollutant data, a hybrid deep learning model called the 1D Convolutional and Long Short-Term Memory (CNN-LSTM) was developed. This model uses a parallel structure of CNN layers, enabling it to effectively capture local and neighbouring spatiotemporal data. The performance of the proposed model was compared to various other deep learning architectures. Additionally, different post-training model quantisation methods were evaluated directly on Raspberry Pi boards to optimise the model for edge devices. When implementing quantisation, there is a trade-off between accuracy, file size, and execution time, which should be carefully considered. Based on the conducted experiments, dynamic range quantisation was found to be a beneficial solution. This approach significantly reduces the file size of the TFLite model compared to the original model while maintaining a similar accuracy level.

- **Objective 3**: To develop collaborative learning strategies among edge devices and evaluate the proposed strategies regarding model accuracy, device performance, and communication cost.

  Achievement: Collaborative learning methods were directly implemented on various edge devices using the MQTT protocol. The study tested several devices, including Raspberry Pi boards and a Jetson Nano board. Three collaborative strategies were explored: FedAvg, ClustME, and SpaTemp. Each strategy employs a different approach to predict pollutant data. Among

the collaborative learning methods, SpaTemp demonstrated superior performance compared to others, exhibiting a range of improvement rates on RMSE (RIR) values from 0.525% to 8.934%. However, it should be noted that this method generally requires more time for model training compared to FedAvg, ClustME, and local approaches. Additionally, the work was extended by modelling network expansion, providing insights into scaling up the collaborative learning system.

- **Objective 4**: To deploy tiny machine learning models on resource-constrained microcontrollers as target devices to address air quality issues.

  Achievement: Three distinct implementations of tinyML on microcontrollers were conducted and discussed in this study. The first implementation involved deploying tinyML using the standard TensorFlow procedure. Air pollutant data was directly measured using a low-cost air quality monitoring device. The proposed method encompassed both prediction and missing data functionalities, and the quantised versions of the models exhibited a size reduction compared to their original lite models. The model predictor and model imputer experienced size reductions of 47.7% and 56.6%, respectively, while maintaining relatively high accuracies. In the second implementation, a binary weight network was employed to reduce further the size of the standard tinyML model obtained from the TensorFlow framework. By preserving the first and last layers and applying binary quantisation to the weights of the intermediate layers, additional size reductions were achieved compared to the standard lite model. Finally, a meta-learning approach was employed to enhance the performance of the tinyML model. The meta-learner utilised ordinary least squares linear regression on top of the proposed base models. The conversion of ML models to lite versions resulted in size reductions of approximately 83% for Base-1 and 77% for Base-2. The deployed tinyML models for Base-1 and Base-2 had sizes of 3,012 bytes and 5,076 bytes, respectively, which were considered light enough even without applying additional quantisation techniques.

7.3 Conclusions

A paradigm shift has occurred within the air pollution monitoring field, moving from reliance on standard, government-operated networks to a hybrid approach that combines reference-level monitors and new sensor technologies that employ low-cost sensing devices. The emergence of new sensor technologies that utilise low-cost

sensing devices does not mean a complete replacement of reference-level monitors. Rather, this technology complements existing monitoring frameworks, offering an additional layer of data collection and analysis. While traditional reference level monitors remain essential to ensure accuracy and reliability in air quality assessments, integrating low-cost sensing devices introduces a new dimension, expanding the scope and granularity of data acquisition. Therefore, both reference level monitors and low-cost sensing devices coexist, synergistically contributing to a more comprehensive understanding of air quality dynamics.

This transition has led to a major surge in the volume of data generated through these sensing devices, paving the way for applying machine learning techniques in air quality research. Machine learning is heading towards the edge, and a recent study also demonstrates how developers significantly contribute to tiny machine learning. This thesis explores various aspects of air quality research, linking machine learning advances to practical environmental challenges.

The presented results demonstrate the effectiveness of our proposed approach, showing R^2 scores exceeding 0.8 for all target stations when confronted with a 20% missing data rate in short-term imputation. The hybrid CNN-LSTM model, utilising spatiotemporal inputs, outperforms the other 19 deep learning models, yielding an RMSE of 15.286 $\mu g/m^3$. Employing dynamic range quantisation proves advantageous for edge optimisation, as it significantly reduces file sizes while preserving near-original accuracy levels. In collaborative learning experiments, the SpaTemp method surpasses FedAvg and ClustME, exhibiting RIR scores ranging from 0.525% to 8.934%. It achieves this while transmitting notably smaller data quantities during the learning process. Nevertheless, it is worth noting that SpaTemp transmits actual measurement data and employs a larger model size. A low-cost sensing device has been developed for tinyML experiments. For tinyML with BWN applications, maintaining the first and last layers while applying binary quantisation to intermediate layer weights achieves further size reduction compared to full-precision models. Incorporating meta-learning with a stacked model presents another viable approach. Through this experiment, the utilisation of meta-learning effectively decreases the RMSE values achieved by individual base models. Additionally, converting machine learning models into lightweight versions reduces model size by around 83% and 77% for Base-1 and Base-2, respectively.

Edge computing represents an evolving frontier, with machine learning increasingly migrating to the edge, encompassing even low-cost air quality monitoring devices. This study introduces the notion of collaborative learning among edge devices to address challenges in air quality research, such as missing data and air

pollutant predictions. While collaborative learning among air quality stations is currently in its nascent stages, the potential for expansion exists, albeit at a laboratory scale for now. Traditionally, air quality data have been sourced solely from fixed monitoring sites and utilised accordingly. However, the practical implementation of federated learning, a form of collaborative learning, by Google demonstrates its viability [255], paving the way for its potential adaptation to air pollution research in the future.

7.4 Further Work

7.4.1 Broader Perspectives of AI-based Smart Sensing and Approaches to Driving Change

Research efforts could explore implementing AI-based smart networks designed specifically for devices with limited resources [256]. These networks would be highly valuable tools for local governments, community organisations, and environmental agencies tasked with monitoring air quality. By focusing on this demographic, researchers can address critical needs in regions where resources may be limited, but air quality issues remain pressing. Potential avenues for research include developing frameworks that offer detailed insights into the deployment and management of AI-based smart networks. This requires elucidating best practices for data collection, analysis, and interpretation in the context of limited resources, including the use of edge devices. Additionally, investigating strategies to optimise network performance and scalability in resource-constrained devices presents an intriguing area for exploration [257]. Furthermore, research efforts could explore the development of user-friendly interfaces and decision support systems tailored to the needs of stakeholders in resource-constrained environments. These tools would empower local authorities and community organisations to make informed decisions regarding air quality management and intervention strategies.

The potential of machine learning extends to addressing further challenges associated with deploying low-cost air quality monitors. Machine learning offers a promising avenue for enhancing sensor and sensor network quality control, thereby ensuring the reliability and accuracy of the data collected. Through sophisticated machine learning models, anomalies and inconsistencies in sensor readings can be swiftly identified and rectified, thereby ensuring the integrity of air quality measurements. Additionally, machine learning techniques can streamline the calibration process by automating adjustments based on real-time data feedback. This automation reduces reliance on manual intervention, minimises human error, and

enhances the overall efficiency of air quality monitoring systems. These systems can dynamically adapt to changing environmental conditions by integrating AI-based calibration mechanisms, ensuring accuracy and consistency over time.

It is crucial to ensure that a diverse range of stakeholders can utilise the information provided to make decisions and drive change across various layers of society. For example, at the individual level, residents can benefit from accessing real-time air quality information, enabling them to take proactive measures to safeguard their health. This could involve actions such as adjusting outdoor activities or using personal protective equipment when air quality levels are poor. On a broader scale, local authorities can utilise the data to pinpoint pollution hotspots and implement targeted interventions. These interventions might include implementing traffic management strategies, establishing green spaces or vegetation buffers, or introducing emission reduction measures for industrial facilities located in the affected areas. At the community level, collaborative efforts can be developed to address broader environmental challenges, leveraging insights gained from AI-driven data analysis and machine learning approaches. This could involve partnerships between local governments, non-profit organisations, businesses, and community groups to develop and implement comprehensive air quality improvement initiatives. Furthermore, at the council level, policymakers can utilise the information to formulate evidence-based policies and regulations to enhance overall air quality [258]. These policies might encompass measures to reduce emissions from transportation and industry, promote the use of clean energy technologies, or incentivise sustainable urban planning practices.

7.4.2 Collaborative Learning and Air Quality Monitoring Network

Collaborative learning at the edge involves two main aspects: local learning on devices and data (or model) transmission between the server and participating devices. One viable exploration approach involves leveraging the LoRa (Long-Range) protocol to facilitate communication between the server and the involved devices. LoRa is a wireless platform known for its long-range capabilities and low-power consumption, making it an ideal choice for Internet of Things (IoT) applications. It allows for long-range communication between devices, making it suitable for deployments in remote areas without an internet connection. However, the protocol is limited by its small payload capability, which poses challenges when transferring data or models during device-to-server or device-to-device communication. To overcome these limitations, designing lightweight models and exploring collaborative learning among microcontrollers are still worthwhile areas of investigation. This can ensure

efficient communication and collaboration while considering the constraints of the LoRa protocol. Giménez *et al.* [259] have led the way in this field by pioneering the implementation of federated learning over LoRa, utilising an open dataset comprising 480 samples across three distinct spoken keywords. Their study involved the integration of two microcontrollers: an Arduino Portenta H7 for model training and a TTGO LORA32 board for LoRa communication. The Arduino Portenta H7 is a high-end, dual-core microcontroller designed for industrial applications. Its default configuration includes 8MB of SDRAM and 16MB of flash memory. Future endeavours could be directed towards air quality research. There is potential for simplifying the approach by utilising a single microcontroller with lower specifications, thereby reducing costs.

The exploration of implementing approaches like SpaTemp and ClustME remains open, along with the prospect of designing a lightweight yet highly accurate machine learning model. Realising a smart, low-cost air quality network requires a combination of theoretical knowledge, methodological approaches, programming skills, prototyping, and hardware expertise. Integrating intelligence into a single device is challenging, but extending that intelligence to a network of interconnected devices poses even greater difficulties. However, a smart network of low-cost air quality monitors can be achieved with individual devices equipped with intelligence. In such a network, devices facing issues like missing data can be compensated by neighbouring devices, ensuring comprehensive coverage. Additionally, devices unable to participate directly in collaborative learning can still benefit from neighbouring devices by leveraging their updated data and knowledge. Despite the advancements made, there are still numerous challenges to address in realising smart, low-cost networks for air quality monitoring. These challenges continue to drive further research and development in the field.

This thesis focuses on outdoor air quality research. Collaborative learning between outdoor sensor nodes requires consideration of the communication protocols used to establish connections between nodes, especially when large distances separate them from other nodes. It should be noted that collaborative learning can also include indoor air monitoring. For example, sensor nodes in multiple rooms in a building can collaboratively learn using local area networks or short-range communication protocols such as Bluetooth, BLE, ZigBee, Wi-Fi, SigFox, LoRa, Ingenu, NB-IoT, and Wi-Fi HaLow. In indoor air quality, machine learning models can be enhanced by leveraging air-related parameters and other factors, commonly referred to as *data fusion*. Multisensor data fusion involves leveraging insights from multiple sources to increase the comprehensive understanding of a phenomenon and to

consolidate evidence or decisions. For example, combining data from motion, sound levels, light intensity sensors, and air quality parameters can potentially improve the performance of deep learning models.

References

[1] I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Optimising Deep Learning at the Edge for Accurate Hourly Air Quality Prediction," *Sensors*, vol. 21, p. 1064, Feb. 2021.

[2] I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Estimation of Missing Air Pollutant Data Using a Spatiotemporal Convolutional Autoencoder," *Neural Computing and Applications*, vol. 34, pp. 16129–16154, May 2022.

[3] I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "Collaborative Learning at the Edge for Air Pollution Prediction," *IEEE Transactions on Instrumentation and Measurement*, vol. 73, pp. 1–12, 2024.

[4] I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Tinyml models for a low-cost air quality monitoring device," *IEEE Sensors Letters*, vol. 7, no. 11, pp. 1–4, 2023.

[5] I. N. K. Wardana, J. W. Gardner, and S. A. Fahmy, "TinyML with Meta-Learning on Microcontrollers for Air Pollution Prediction." XXXV Eurosensors Conference, Sep. 2023.

[6] I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising Tiny Machine Learning With Binary Weight Network for a Low-Cost Air Quality Monitoring Device." The 3rd Imperial Workshop on Intelligent Communications, June 2023.

[7] I. N. K. Wardana, S. A. Fahmy, and J. W. Gardner, "Optimising TinyML Using Binary Weight Network and Meta-Learning for a Low-Cost Air Quality Monitoring Device." Warwick Secure and Intelligent Communications (WSIC) Workshop, July 2023.

[8] H. E. Institute, "State of Global Air 2019," tech. rep., Health Effects Institute, Boston, MA, 2019.

[9] A. Elessa Etuman and I. Coll, "Integrated Air Quality Modeling for Urban Policy: A Novel Approach with Olympus-chimere," *Atmospheric Environment*, vol. 315, p. 120134, Dec. 2023.

[10] W. H. Organization, "Ambient (Outdoor) Air Pollution." `https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health`. Accessed: 2023-12-25.

[11] F. Perera, "Pollution from Fossil-Fuel Combustion is the Leading Environmental Threat to Global Pediatric Health and Equity: Solutions Exist," *International Journal of Environmental Research and Public Health*, vol. 15, p. 16, Dec. 2017.

[12] S. Ameer, M. A. Shah, A. Khan, H. Song, C. Maple, S. U. Islam, and M. N. Asghar, "Comparative Analysis of Machine Learning Techniques for Predicting Air Quality in Smart Cities," *IEEE Access*, vol. 7, pp. 128325–128338, 2019.

[13] United Nations, *World Population Prospects 2022: Summary of Results*. Statistical Papers - United Nations (Ser. A), Population and Vital Statistics Report, United Nations, Aug. 2022.

[14] W. H. Organization *et al.*, *WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide*. World Health Organization, 2021.

[15] Y. Guo, H. Zeng, R. Zheng, S. Li, A. G. Barnett, S. Zhang, X. Zou, R. Huxley, W. Chen, and G. Williams, "The Association Between Lung Cancer Incidence and Ambient Air Pollution in China: A Spatiotemporal Analysis," *Environmental Research*, vol. 144, pp. 60–65, 2016.

[16] G. B. Hamra, N. Guha, A. Cohen, F. Laden, O. Raaschou-Nielsen, J. M. Samet, P. Vineis, F. Forastiere, P. Saldiva, T. Yorifuji, and D. Loomis, "Outdoor Particulate Matter Exposure and Lung Cancer: A Systematic Review and Meta-Analysis," *Environmental Health Perspectives*, vol. 122, no. 9, pp. 906–911, 2014.

[17] Q. Chen, Q. Wang, B. Xu, Y. Xu, Z. Ding, and H. Sun, "Air Pollution and Cardiovascular Mortality in Nanjing, China: Evidence Highlighting the Roles of Cumulative Exposure and Mortality Displacement," *Chemosphere*, vol. 265, p. 129035, 2021.

[18] H. Saygin, Y. Mercan, and F. Yorulmaz, "The Association Between Air Pollution Parameters and Emergency Department Visits and Hospitalizations Due to Cardiovascular and Respiratory Diseases: A Time-Series Analysis," *International Archives of Occupational and Environmental Health*, Oct 2021.

[19] Y. Ma, H. Zhang, Y. Zhao, J. Zhou, S. Yang, X. Zheng, and S. Wang, "Short-Term Effects of Air Pollution on Daily Hospital Admissions for Cardiovascular Diseases in Western China," *Environmental Science and Pollution Research*, vol. 24, pp. 14071–14079, Jun 2017.

[20] J. M. Delgado-Saborit, V. Guercio, A. M. Gowers, G. Shaddick, N. C. Fox, and S. Love, "A Critical Review of the Epidemiological Evidence of Effects of Air Pollution on Dementia, Cognitive Function and Cognitive Decline in Adult Population," *Science of The Total Environment*, vol. 757, p. 143734, 2021.

[21] C. Li and S. Managi, "Spatial Variability of the Relationship Between Air Pollution and Well-Being," *Sustainable Cities and Society*, vol. 76, p. 103447, 2022.

[22] J. Ma, Z. Li, J. C. Cheng, Y. Ding, C. Lin, and Z. Xu, "Air Quality Prediction at New Stations Using Spatially Transferred Bi-directional Long Short-Term Memory Network," *Science of The Total Environment*, vol. 705, p. 135771, Feb. 2020.

[23] Z. Zhang, G. Zhang, and B. Su, "The Spatial Impacts of Air Pollution and Socio-Economic Status on Public Health: Empirical Evidence From China," *Socio-Economic Planning Sciences*, p. 101167, 2021.

[24] M. Tainio, Z. Jovanovic Andersen, M. J. Nieuwenhuijsen, L. Hu, A. de Nazelle, R. An, L. M. Garcia, S. Goenka, B. Zapata-Diomedi, F. Bull, and T. H. d. Sá, "Air Pollution, Physical Activity and Health: A Mapping Review of the Evidence," *Environment International*, vol. 147, p. 105954, Feb. 2021.

[25] R. Sivarethinamohan, S. Sujatha, S. Priya, Sankaran, A. Gafoor, and Z. Rahman, "Impact of Air Pollution in Health and Socio-Economic Aspects: Review on Future Approach," *Materials Today: Proceedings*, vol. 37, pp. 2725–2729, 2021. International Conference on Newer Trends and Innovation in Mechanical Engineering: Materials Science.

[26] J. Rentschler and N. Leonova, "Global Air Pollution Exposure and Poverty," *Nature Communications*, vol. 14, July 2023.

[27] G. Shaddick, M. L. Thomas, P. Mudu, G. Ruggeri, and S. Gumy, "Half the World's Population are Exposed to Increasing Air Pollution," *npj Climate and Atmospheric Science*, vol. 3, June 2020.

[28] S. Abdul Jabbar, L. Tul Qadar, S. Ghafoor, L. Rasheed, Z. Sarfraz, A. Sarfraz, M. Sarfraz, M. Felix, and I. Cherrez-Ojeda, "Air Quality, Pollution and Sustainability Trends in South Asia: A Population-Based Study," *International Journal of Environmental Research and Public Health*, vol. 19, p. 7534, June 2022.

[29] M. J. Moya, "These countries have the most polluted air in the world, new report says." `https://phys.org/news/2022-03-countries-polluted-air-world.html`. Accessed: 2023-12-25.

[30] W. Liu, Z. Xu, and T. Yang, "Health Effects of Air Pollution in China," *International Journal of Environmental Research and Public Health*, vol. 15, p. 1471, July 2018.

[31] X. Lu, S. Zhang, J. Xing, Y. Wang, W. Chen, D. Ding, Y. Wu, S. Wang, L. Duan, and J. Hao, "Progress of Air Pollution Control in China and Its Challenges and Opportunities in the Ecological Civilization Era," *Engineering*, vol. 6, p. 1423–1431, Dec. 2020.

[32] T. Hassan Bhat, G. Jiawen, and H. Farzaneh, "Air Pollution Health Risk Assessment (AP-HRA), Principles and Applications," *International Journal of Environmental Research and Public Health*, vol. 18, p. 1935, Feb. 2021.

[33] J. Burns, H. Boogaard, S. Polus, L. M. Pfadenhauer, A. C. Rohwer, A. M. van Erp, R. Turley, and E. Rehfuess, "Interventions to Reduce Ambient Particulate Matter Air Pollution and Their Effect on Health," *Cochrane Database of Systematic Reviews*, vol. 2019, May 2019.

[34] H. Orru, K. L. Ebi, and B. Forsberg, "The Interplay of Climate Change and Air Pollution on Health," *Current Environmental Health Reports*, vol. 4, p. 504–513, Oct. 2017.

[35] A.-C. Pinho-Gomes, E. Roaf, G. Fuller, D. Fowler, A. Lewis, H. ApSimon, C. Noakes, P. Johnstone, and S. Holgate, "Air Pollution and Climate Change," *The Lancet Planetary Health*, vol. 7, p. e727–e728, Sept. 2023.

[36] R. M. Doherty, M. R. Heal, and F. M. O'Connor, "Climate Change Impacts on Human Health Over Europe Through Its Effect on Air Quality," *Environmental Health*, vol. 16, Nov. 2017.

[37] A. Jonidi Jafari, E. Charkhloo, and H. Pasalari, "Urban Air Pollution Control Policies and Strategies: a Systematic Review," *Journal of Environmental Health Science and Engineering*, vol. 19, p. 1911–1940, Oct. 2021.

[38] K.-J. Liao, X. Hou, and M. J. Strickland, "Resource Allocation for Mitigating Regional Air Pollution-Related Mortality: A Summertime Case Study for Five Cities in the United States," *Journal of the Air & Waste Management Association*, vol. 66, p. 748–757, July 2016.

[39] L. Fu, J. Li, and Y. Chen, "An Innovative Decision Making Method for Air Quality Monitoring Based on Big Data-Assisted Artificial Intelligence Technique," *Journal of Innovation & Knowledge*, vol. 8, p. 100294, Apr. 2023.

[40] F. Chen, M. Wang, and Z. Pu, "The Impact of Technological Innovation on Air Pollution: Firm-level Evidence from China," *Technological Forecasting and Social Change*, vol. 177, p. 121521, Apr. 2022.

[41] R. D. Brook, D. E. Newby, and S. Rajagopalan, "The Global Threat of Outdoor Ambient Air Pollution to Cardiovascular Health: Time for Intervention," *JAMA Cardiology*, vol. 2, pp. 353–354, 04 2017.

[42] Y. Zhang, J. J. West, R. Mathur, J. Xing, C. Hogrefe, S. J. Roselle, J. O. Bash, J. E. Pleim, C.-M. Gan, and D. C. Wong, "Long-term Trends in the Ambient $PM_{2.5}$- and O_3-related Mortality Burdens in the United States Under Emission Reductions from 1990 to 2010," *Atmospheric Chemistry and Physics*, vol. 18, no. 20, pp. 15003–15016, 2018.

[43] UNDP, "What are the Sustainable Development Goals?." `https://www.undp.org/sustainable-development-goals`. Accessed: 2023-12-29.

[44] Editorial, "Clean air for a sustainable world," *Nature Communications*, vol. 12, p. 5824, Oct. 2021.

[45] W. H. Organization, "What are the WHO Air Quality Guidelines?." `https://www.who.int/news-room/feature-stories/detail/what-are-the-who-air-quality-guidelines`. Accessed: 2023-12-28.

[46] UNCC, "COP28 Agreement Signals "Beginning of the End" of the Fossil Fuel Era." https://unfccc.int/news/cop28-agreement-signals-beginning-of-the-end-of-the-fossil-fuel-era. Accessed: 2023-12-29.

[47] Y. Zhao and B. Kim, "Environmental regulation and chronic conditions: Evidence from china's air pollution prevention and control action plan," *International Journal of Environmental Research and Public Health*, vol. 19, p. 12584, Oct. 2022.

[48] C. Chen, J.-L. Fang, W.-Y. Shi, T.-T. Li, and X.-M. Shi, "Clean Air Actions and Health Plans in China," *Chinese Medical Journal*, vol. 133, p. 1609–1611, June 2020.

[49] P. Wang, "China's Air Pollution Policies: Progress and Challenges," *Current Opinion in Environmental Science & Health*, vol. 19, p. 100227, Feb. 2021.

[50] T. Ganguly, K. L. Selvaraj, and S. K. Guttikunda, "National Clean Air Programme (NCAP) for Indian Cities: Review and Outlook of Clean Air Action Plans," *Atmospheric Environment: X*, vol. 8, p. 100096, Dec. 2020.

[51] A. Kansal, S. P. Subuddhi, P. Pandey, D. Gupta, T. Rawat, A. S. Gautam, and S. Gautam, "Investigating the impression of national clean air programme in enhancement of air quality characteristics for non-attainment cities of uttarakhand," *Aerosol Science and Engineering*, vol. 7, p. 415–425, Apr. 2023.

[52] M. A. Fekih, W. Bechkit, H. Rivano, M. Dahan, F. Renard, L. Alonso, and F. Pineau, "Participatory Air Quality and Urban Heat Islands Monitoring System," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–14, 2021.

[53] A. C. Rai, P. Kumar, F. Pilla, A. N. Skouloudis, S. D. Sabatino, C. Ratti, A. Yasar, and D. Rickerby, "End-User Perspective of Low-Cost Sensors for Outdoor Air Pollution Monitoring," *Science of The Total Environment*, vol. 607-608, pp. 691–705, Dec 2017.

[54] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," *IEEE Internet of Things Journal*, vol. 6, pp. 8182–8201, Oct. 2019.

[55] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated Machine Learning: Survey, Multi-Level Classification, Desirable Criteria and Future Directions in Communication and Networking Systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1342–1397, 2021.

[56] X. Dai, B. Zhang, X. Jiang, L. Liu, D. Fang, and Z. Long, "Has the Three-Year Action Plan Improved the Air Quality in the Fenwei Plain of China? Assessment Based on a Machine Learning Technique," *Atmospheric Environment*, vol. 286, p. 119204, Oct 2022.

[57] Arm, "The Future of ML Shifts to the Edge." `https://www.arm.com/markets/artificial-intelligence`, 2023. Accessed: 2023-10-11.

[58] J. Bier, "AI and Vision at the Edge." `https://www.eetimes.com/ai-and-vision-at-the-edge/`, 2023. Accessed: 2023-10-11.

[59] E. Impulse, "What Is Embedded ML, Anyway?." `https://docs.edgeimpulse.com/docs/what-is-embedded-machine-learning-anyway`, 2023. Accessed: 2023-10-11.

[60] M. Buchecker and J. Frick, "The Implications of Urbanization for Inhabitants' Relationship to Their Residential Environment," *Sustainability*, vol. 12, p. 1624, Feb. 2020.

[61] E. X. Neo, K. Hasikin, K. W. Lai, M. I. Mokhtar, M. M. Azizan, H. F. Hizaddin, S. A. Razak, and Yanto, "Artificial Intelligence-Assisted Air Quality Monitoring for Smart City Management," *PeerJ Computer Science*, vol. 9, p. e1306, May 2023.

[62] United Nations, Department of Economic and Social Affairs, Population Division, *World Urbanization Prospects: The 2018 Revision (ST/ESA/SER.A/420)*. New York: United Nations, 2019. `https://population.un.org/wup/publications/Files/WUP2018-Report.pdf`.

[63] United Nations, Department of Economic and Social Affairs, "68% of the World Population Projected to Live in Urban Areas by 2050, Says UN." `https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html`. Accessed: 2023-12-31.

[64] H. Dong, M. Xue, Y. Xiao, and Y. Liu, "Do Carbon Emissions Impact the Health of Residents? Considering China's Industrialization and Urbanization," *Science of The Total Environment*, vol. 758, p. 143688, Mar. 2021.

[65] T.-B. Jiang, Z.-W. Deng, Y.-P. Zhi, H. Cheng, and Q. Gao, "The Effect of Urbanization on Population Health: Evidence From China," *Frontiers in Public Health*, vol. 9, June 2021.

[66] A. Bekkar, B. Hssina, S. Douzi, and K. Douzi, "Air-Pollution Prediction in Smart City, Deep Learning Approach," *Journal of Big Data*, vol. 8, Dec. 2021.

[67] S. R. Garzon, S. Walther, S. Pang, B. Deva, and A. Küpper, "Urban Air Pollution Alert Service for Smart Cities," in *Proceedings of the 8th International Conference on the Internet of Things*, IOT '18, ACM, Oct. 2018.

[68] S. J. Hadeed, M. K. O'Rourke, J. L. Burgess, R. B. Harris, and R. A. Canales, "Imputation Methods for Addressing Missing Data in Short-Term Monitoring of Air Pollutants," *Science of The Total Environment*, vol. 730, p. 139140, Aug. 2020.

[69] N. Shahid, M. A. Shah, A. Khan, C. Maple, and G. Jeon, "Towards Greener Smart Cities and Road Traffic Forecasting Using Air Pollution Data," *Sustainable Cities and Society*, vol. 72, p. 103062, 2021.

[70] H. Zhang, J. Li, B. Wen, Y. Xun, and J. Liu, "Connecting Intelligent Things in Smart Hospitals Using NB-IoT," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1550–1560, 2018.

[71] O. Postolache, J. Pereira, and P. Girao, "Smart Sensors Network for Air Quality Monitoring Applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, p. 3253–3262, Sept. 2009.

[72] H. Chojer, P. Branco, F. Martins, M. Alvim-Ferraz, and S. Sousa, "Source Identification and Mitigation of Indoor Air Pollution Using Monitoring Data – Current Trends," *Environmental Technology & Innovation*, vol. 33, p. 103534, Feb. 2024.

[73] J. Wang, W. Du, Y. Lei, Y. Chen, Z. Wang, K. Mao, S. Tao, and B. Pan, "Quantifying the Dynamic Characteristics of Indoor Air Pollution Using Real-time Sensors: Current Status and Future Implication," *Environment International*, vol. 175, p. 107934, May 2023.

[74] S. Taheri and A. Razban, "Learning-based CO_2 Concentration Prediction: Application to Indoor Air Quality Control Using Demand-controlled Ventilation," *Building and Environment*, vol. 205, p. 108164, Nov. 2021.

[75] Z. Chu, M. Cheng, and N. N. Yu, "A Smart City Is a Less Polluted City," *Technological Forecasting and Social Change*, vol. 172, p. 121037, 2021.

[76] N. Castell, F. R. Dauge, P. Schneider, M. Vogt, U. Lerner, B. Fishbain, D. Broday, and A. Bartonova, "Can Commercial Low-Cost Sensor Platforms Contribute to Air Quality Monitoring and Exposure Estimates?," *Environment International*, vol. 99, pp. 293–302, Feb. 2017.

[77] L. Morawska, P. K. Thai, X. Liu, A. Asumadu-Sakyi, G. Ayoko, A. Bartonova, A. Bedini, F. Chai, B. Christensen, M. Dunbabin, J. Gao, G. S. Hagler, R. Jayaratne, P. Kumar, A. K. Lau, P. K. Louie, M. Mazaheri, Z. Ning, N. Motta, B. Mullins, M. M. Rahman, Z. Ristovski, M. Shafiei, D. Tjondronegoro, D. Westerdahl, and R. Williams, "Applications of Low-Cost Sensing Technologies for Air Quality Monitoring and Exposure Assessment: How Far Have They Gone?," *Environment International*, vol. 116, pp. 286–299, July 2018.

[78] E. G. Snyder, T. H. Watkins, P. A. Solomon, E. D. Thoma, R. W. Williams, G. S. W. Hagler, D. Shelow, D. A. Hindin, V. J. Kilaru, and P. W. Preuss, "The Changing Paradigm of Air Pollution Monitoring," *Environmental Science & Technology*, vol. 47, pp. 11369–11377, Oct. 2013.

[79] K. Kelly, J. Whitaker, A. Petty, C. Widmer, A. Dybwad, D. Sleeth, R. Martin, and A. Butterfield, "Ambient and Laboratory Evaluation of a Low-Cost Particulate Matter Sensor," *Environmental Pollution*, vol. 221, pp. 491–500, Feb. 2017.

[80] S. Esfahani, P. Rollins, J. P. Specht, M. Cole, and J. W. Gardner, "Smart City Battery Operated IoT Based Indoor Air Quality Monitoring System," in *2020 IEEE SENSORS*, IEEE, Oct. 2020.

[81] K. Yadav, V. Arora, M. Kumar, S. N. Tripathi, V. M. Motghare, and K. A. Rajput, "Few-Shot Calibration of Low-Cost Air Pollution ($PM_{2.5}$) Sensors Using Meta Learning," *IEEE Sensors Letters*, vol. 6, pp. 1–4, May 2022.

[82] E. Instruments, "AQMesh: The Proven Small Sensor Air Quality Monitoring System." `https://www.aqmesh.com`. Accessed: 2023-07-04.

[83] G. Zhou, J. Xu, Y. Xie, L. Chang, W. Gao, Y. Gu, and J. Zhou, "Numerical Air Quality Forecasting Over Eastern China: An Operational Application of WRF-Chem," *Atmospheric Environment*, vol. 153, pp. 94–108, Mar. 2017.

[84] J. Liu, K. Luo, Z. Zhou, and X. Chen, "ERP: Edge Resource Pooling for Data Stream Mobile Computing," *IEEE Internet of Things Journal*, vol. 6, pp. 4355–4368, June 2019.

[85] H. Liu, H. Wu, X. Lv, Z. Ren, M. Liu, Y. Li, and H. Shi, "An Intelligent Hybrid Model for Air Pollutant Concentrations Forecasting: Case of Beijing in China," *Sustainable Cities and Society*, vol. 47, p. 101471, 2019.

[86] R. Zhao, X. Gu, B. Xue, J. Zhang, and W. Ren, "Short Period $PM_{2.5}$ Prediction Based on Multivariate Linear Regression Model," *PLOS ONE*, vol. 13, pp. 1–15, July 2018.

[87] P. Gupta and S. A. Christopher, "Particulate Matter Air Quality Assessment Using Integrated Surface, Satellite, and Meteorological Products: Multiple Regression Approach," *Journal of Geophysical Research*, vol. 114, July 2009.

[88] C. Li, N. C. Hsu, and S.-C. Tsay, "A Study on the Potential Applications of Satellite Data in Air Quality Monitoring and Forecasting," *Atmospheric Environment*, vol. 45, pp. 3663–3675, July 2011.

[89] U. Kumar and V. K. Jain, "ARIMA Forecasting of Ambient Air Pollutants (O_3, NO, NO_2 and CO)," *Stochastic Environmental Research and Risk Assessment*, vol. 24, pp. 751–760, Dec. 2009.

[90] P. G. Nieto, F. S. Lasheras, E. García-Gonzalo, and F. de Cos Juez, "PM_{10} Concentration Forecasting in the Metropolitan Area of Oviedo (Northern Spain) Using Models Based on SVM, MLP, VARMA and ARIMA: A Case Study," *Science of The Total Environment*, vol. 621, pp. 753–761, Apr. 2018.

[91] E. Chianese, F. Camastra, A. Ciaramella, T. Landi, A. Staiano, and A. Riccio, "Spatio-Temporal Learning in Predicting Ambient Particulate Matter Concentration by Multi-Layer Perceptron," *Ecological Informatics*, vol. 49, pp. 54–61, Jan. 2019.

[92] Y. Huang, Y. Xiang, R. Zhao, and Z. Cheng, "Air Quality Prediction Using Improved PSO-BP Neural Network," *IEEE Access*, vol. 8, pp. 99346–99353, 2020.

[93] V. Yadav and S. Nath, "Daily Prediction of PM10 Using Radial Basis Function and Generalized Regression Neural Network," in *2018 Recent Advances on Engineering, Technology and Computational Sciences (RAETCS)*, IEEE, Feb. 2018.

[94] P. G. Nieto, E. Combarro, J. del Coz Díaz, and E. Montañés, "A SVM-Based Regression Model to Study the Air Quality at Local Scale in Oviedo Urban Area (Northern Spain): A Case Study," *Applied Mathematics and Computation*, vol. 219, pp. 8923–8937, May 2013.

[95] J. C. Patni and H. K. Sharma, "Air Quality Prediction Using Artificial Neural Networks," in *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, IEEE, Apr. 2019.

[96] D. Mishra and P. Goyal, "Neuro-Fuzzy Approach to Forecast NO2 Pollutants Addressed to Air Quality Dispersion Model over Delhi, India," *Aerosol and Air Quality Research*, vol. 16, no. 1, pp. 166–174, 2017.

[97] Y. Yang, G. Mei, and S. Izzo, "Revealing Influence of Meteorological Conditions on Air Quality Prediction Using Explainable Deep Learning," *IEEE Access*, vol. 10, pp. 50755–50773, 2022.

[98] J. Ma, Z. Li, J. C. Cheng, Y. Ding, C. Lin, and Z. Xu, "Air Quality Prediction at New Stations Using Spatially Transferred Bi-directional Long Short-Term Memory Network," *Science of The Total Environment*, vol. 705, p. 135771, 2020.

[99] M. M. Samsami, N. Shojaee, S. Savar, and M. Yazdi, "Classification of the Air Quality Level based on Analysis of the Sky Images," in *2019 27th Iranian Conference on Electrical Engineering (ICEE)*, IEEE, Apr. 2019.

[100] H.-J. Oh and J. Kim, "Monitoring Air Quality and Estimation of Personal Exposure to Particulate Matter Using an Indoor Model and Artificial Neural Network," *Sustainability*, vol. 12, p. 3794, May 2020.

[101] J. A. Stingone, O. P. Pandey, L. Claudio, and G. Pandey, "Using Machine Learning to Identify Air Pollution Exposure Profiles Associated with Early Cognitive Skills Among U.S. Children," *Environmental Pollution*, vol. 230, p. 730–740, Nov. 2017.

[102] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015.

[103] M. Verhelst and B. Moons, "Embedded Deep Neural Network Processing: Algorithmic and Processor Techniques Bring Deep Learning to IoT and Edge Devices," *IEEE Solid-State Circuits Magazine*, vol. 9, no. 4, pp. 55–65, 2017.

[104] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A Survey on the Edge Computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[105] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling technologies," *IEEE Internet of Things Journal*, pp. 1–20, 2016.

[106] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2483–2495, 2018.

[107] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the Institute of Electrical and Electronics Engineers IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[108] 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.

[109] F. Samie, L. Bauer, and J. Henkel, "From Cloud Down to Things: An Overview of Machine Learning in Internet of Things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[110] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[111] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[112] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An Overview of Machine Learning within Embedded and Mobile Devices–Optimizations and Applications," *Sensors*, vol. 21, p. 4412, June 2021.

[113] A. Biglari and W. Tang, "A Review of Embedded Machine Learning Based on Hardware, Application, and Sensing Scheme," *Sensors*, vol. 23, p. 2131, Feb. 2023.

[114] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," *arXiv*, 2016. eprint 1605.08695.

[115] TensorFlow, "TensorFlow." `https://www.tensorflow.org/`. Accessed: 2023-07-08.

[116] T. L. Foundation, "PyTorch." `https://pytorch.org/`. Accessed: 2023-08-23.

[117] Keras, "Keras." `https://keras.io/`. Accessed: 2023-08-23.

[118] BAIR, "Caffe." `https://caffe.berkeleyvision.org/`. Accessed: 2023-08-23.

[119] MathWorks, "MATLAB." `https://mathworks.com/`. Accessed: 2023-08-23.

[120] V. Kurama, "PyTorch vs. TensorFlow: Key Differences to Know for Deep Learning." `https://builtin.com/data-science/pytorch-vs-tensorflow`. Accessed: 2024-01-31.

[121] J. Terra, "Keras vs Tensorflow vs Pytorch: Key Differences Among Deep Learning." `https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article`. Accessed: 2024-01-31.

[122] Tensorflow, "Deploy Machine Learning Models on Mobile and Edge Devices." `https://www.tensorflow.org/lite`. Accessed: 2024-01-31.

[123] Y. S. Lee, Y.-H. Gong, and S. W. Chung, "Scale-CIM: Precision-Scalable Computing-in-Memory for Energy-Efficient Quantized Neural Networks," *Journal of Systems Architecture*, vol. 134, p. 102787, Jan. 2023.

[124] "Post-training Quantization." Available online: `https://www.tensorflow.org/lite/performance/post_training_quantization`. (Last accessed on 30-Mar-2023).

[125] M. Z. M. Shamim, "TinyML Model for Classifying Hazardous Volatile Organic Compounds Using Low-Power Embedded Edge Sensors: Perfecting Factory 5.0 Using Edge AI," *IEEE Sensors Letters*, vol. 6, pp. 1–4, Sept. 2022.

[126] G. M. Iodice, *TinyML Cookbook: Combine Artificial Intelligence and Ultra-Low-Power Embedded Devices to Make the World Smarter*. Birmingham:

Packt, 2022. https://www.packtpub.com/product/tinyml-cookbook/9781801814973.

[127] J. Botero-Valencia, C. Barrantes-Toro, D. Marquez-Viloria, and J. M. Pearce, "Low-cost Air, Noise, and Light Pollution Measuring Station with Wireless Communication and TinyML," *HardwareX*, vol. 16, p. e00477, Dec. 2023.

[128] F. Sakr, F. Bellotti, R. Berta, and A. De Gloria, "Machine Learning on Mainstream Microcontrollers," *Sensors*, vol. 20, p. 2638, May 2020.

[129] L. Ioannou, *Exploring the Capabilities of FPGA DSP Blocks in Neural Network Accelerators*. PhD thesis, University of Warwick, 2021.

[130] U. A. Shah, S. Yousaf, I. Ahmad, S. U. Rehman, and M. O. Ahmad, "Accelerating Revised Simplex Method Using GPU-Based Basis Update," *IEEE Access*, vol. 8, pp. 52121–52138, 2020.

[131] Arm, "What Is ASIC?." https://www.arm.com/glossary/asic, 2023. Accessed: 2023-08-24.

[132] J. Xu, Y. Huan, B. Huang, H. Chu, Y. Jin, L.-R. Zheng, and Z. Zou, "A Memory-Efficient CNN Accelerator Using Segmented Logarithmic Quantization and Multi-Cluster Architecture," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, pp. 2142–2146, June 2021.

[133] Intel, "FPGA Architecture Overview." https://www.intel.com/content/www/us/en/docs/programmable/683152/23-2/fpga-architecture-overview.html, 2023. Accessed: 2023-08-25.

[134] R. Kastner, J. Matai, and S. Neuendorffer, "Parallel programming for fpgas," *arXiv*, 2018. eprint 1805.03648.

[135] J. A. Ramírez-Montañez, J. d. J. Rangel-Magdaleno, M. A. Aceves-Fernández, and J. M. Ramos-Arreguín, "Modeling of Particulate Pollutants Using a Memory-Based Recurrent Neural Network Implemented on an FPGA," *Micromachines*, vol. 14, p. 1804, Sept. 2023.

[136] S. Abbasi, S. Gignac, and H. Koc, "An FPGA Based Multi-Sensor Atmospheric Testing Device for Confined Spaces," in *2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, IEEE, Dec. 2021.

[137] Arm, "Arm Ethos-U NPU Application development overview Version 5.0." https://developer.arm.com/documentation/101888/0500/?lang=en, 2023. Accessed: 2023-08-24.

[138] Nvidia, "Jetson Nano Developer Kit." https://developer.nvidia.com/embedded/jetson-nano-developer-kit. Accessed: 2023-07-03.

[139] R. Pi, "Raspberry Pi." https://www.raspberrypi.com/. Accessed: 2023-07-03.

[140] P. Warden and D. Situnayake, *TinyML: Machine learning with tensoflow lite on Arduino and ultra-low-power Microcontrollers*. Sebastopol: O'Reilly Media, Inc., 2020.

[141] S. F. Barrett and D. J. Pack, "Introduction to Microcontroller Technology," in *Synthesis Lectures on Digital Circuits & Systems*, pp. 1–19, Springer International Publishing, 2019.

[142] E. Upton, "A New Path From Hobbyist to Embedded and IoT Development: Raspberry Pi Pico W." https://www.arm.com/blogs/blueprint/raspberry-pi-pico-w. Accessed: 2023-07-04.

[143] STMicroelectronics, "Arm®Cortex®-M in a Nutshell." https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers.html. Accessed: 2023-07-04.

[144] R. Pi, "Raspberry Pi Pico." https://www.raspberrypi.com/products/raspberry-pi-pico/. Accessed: 2023-07-04.

[145] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[146] D. Chicco, M. J. Warrens, and G. Jurman, "The Coefficient of Determination R-squared Is More Informative Than SMAPE, MAE, MAPE, MSE and RMSE in Regression Analysis Evaluation," *PeerJ Computer Science*, vol. 7, p. e623, July 2021.

[147] N. R. Council, *Improving Information for Social Policy Decisions – The Uses of Microsimulation Modeling*. Washington, D.C.: National Academies Press, Jan. 1991.

[148] J. Ma, J. C. Cheng, Y. Ding, C. Lin, F. Jiang, M. Wang, and C. Zhai, "Transfer Learning for Long-Interval Consecutive Missing Values Imputation Without External Features in Air Pollution Time Series," *Advanced Engineering Informatics*, vol. 44, p. 101092, Apr. 2020.

[149] X.-H. Zhou, "Challenges and Strategies in Analysis of Missing Data," *Biostatistics & Epidemiology*, vol. 4, no. 1, pp. 15–23, 2020.

[150] Y. Yu, J. J. Q. Yu, V. O. K. Li, and J. C. K. Lam, "A Novel Interpolation-SVT Approach for Recovering Missing Low-Rank Air Quality Data," *IEEE Access*, vol. 8, pp. 74291–74305, 2020.

[151] P. C. Austin, I. R. White, D. S. Lee, and S. van Buuren, "Missing Data in Clinical Research: A Tutorial on Multiple Imputation," *Canadian Journal of Cardiology*, vol. 37, pp. 1322–1331, Sept. 2021.

[152] J. Ma, J. C. Cheng, F. Jiang, W. Chen, M. Wang, and C. Zhai, "A Bidirectional Missing Data Imputation Scheme Based on LSTM and Transfer Learning for Building Energy Data," *Energy and Buildings*, vol. 216, p. 109941, June 2020.

[153] I. Laña, I. I. Olabarrieta, M. Vélez, and J. D. Ser, "On the Imputation of Missing Data for Road Traffic Forecasting: New Insights and Novel Techniques," *Transportation Research Part C: Emerging Technologies*, vol. 90, pp. 18–33, May 2018.

[154] M. Pena, P. Ortega, and M. Orellana, "A Novel Imputation Method for Missing Values in Air Pollutant Time Series Data," Nov. 2019.

[155] S. Moshenberg, U. Lerner, and B. Fishbain, "Spectral Methods for Imputation of Missing Air Quality Data," *Environmental Systems Research*, vol. 4, p. 26, Dec 2015.

[156] D. B. Rubin, "Inference and Missing Data," *Biometrika*, vol. 63, pp. 581–592, Dec 1976.

[157] M. Gómez-Carracedo, J. Andrade, P. López-Mahía, S. Muniategui, and D. Prada, "A Practical Comparison of Single and Multiple Imputation Methods to Handle Complex Missing Data in Air Quality Datasets," *Chemometrics and Intelligent Laboratory Systems*, vol. 134, pp. 23–33, 2014.

[158] W. Junger and A. Ponce de Leon, "Imputation of Missing Data in Time Series for Air Pollutants," *Atmospheric Environment*, vol. 102, pp. 96–104, 2015.

[159] A. R. T. Donders, G. J. van der Heijden, T. Stijnen, and K. G. Moons, "Review: A Gentle Introduction to Imputation of Missing Values," *Journal of Clinical Epidemiology*, vol. 59, pp. 1087–1091, Oct 2006.

[160] J. W. Graham, "Missing Data Analysis: Making It Work in the Real World," *Annual Review of Psychology*, vol. 60, no. 1, pp. 549–576, 2009.

[161] A. Plaia and A. Bondì, "Single Imputation Method of Missing Values in Environmental Pollution Data Sets," *Atmospheric Environment*, vol. 40, no. 38, pp. 7316–7330, 2006.

[162] X. Zhou, X. Liu, G. Lan, and J. Wu, "Federated Conditional Generative Adversarial Nets Imputation Method for Air Quality Missing Data," *Knowledge-Based Systems*, vol. 228, p. 107261, Sept. 2021.

[163] Y.-F. Zhang, P. J. Thorburn, W. Xiang, and P. Fitch, "SSIM—A Deep Learning Approach for Recovering Missing Time Series Sensor Data," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6618–6628, 2019.

[164] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and Composing Robust Features With Denoising Autoencoders," in *Proceedings of the 25th international conference on Machine learning - ICML '08*, ACM Press, 2008.

[165] A. Saleh Ahmed, W. H. El-Behaidy, and A. A. Youssif, "Medical Image Denoising System Based on Stacked Convolutional Autoencoder for Enhancing 2-Dimensional Gel Electrophoresis Noise Reduction," *Biomedical Signal Processing and Control*, vol. 69, p. 102842, 2021.

[166] M. Juneja, S. Kaur Saini, S. Kaul, R. Acharjee, N. Thakur, and P. Jindal, "Denoising of Magnetic Resonance Imaging Using Bayes Shrinkage Based Fused Wavelet Transform and Autoencoder Based Deep Learning Approach," *Biomedical Signal Processing and Control*, vol. 69, p. 102844, 2021.

[167] Z. Fang, T. Jia, Q. Chen, M. Xu, X. Yuan, and C. Wu, "Laser Stripe Image Denoising Using Convolutional Autoencoder," *Results in Physics*, vol. 11, pp. 96–104, 2018.

[168] K. Bajaj, D. K. Singh, and M. A. Ansari, "Autoencoders Based Deep Learner for Image Denoising," *Procedia Computer Science*, vol. 171, pp. 1535–1541, 2020. Third International Conference on Computing and Network Communications (CoCoNet'19).

[169] E. Dasan and I. Panneerselvam, "A Novel Dimensionality Reduction Approach for ECG Signal via Convolutional Denoising Autoencoder With LSTM," *Biomedical Signal Processing and Control*, vol. 63, p. 102225, 2021.

[170] S. Nagar, A. Kumar, and M. Swamy, "Orthogonal Features-Based EEG Signal Denoising Using Fractionally Compressed Autoencoder," *Signal Processing*, vol. 188, p. 108225, 2021.

[171] H. Zhu, J. Cheng, C. Zhang, J. Wu, and X. Shao, "Stacked Pruning Sparse Denoising Autoencoder Based Intelligent Fault Diagnosis of Rolling Bearings," *Applied Soft Computing*, vol. 88, p. 106060, 2020.

[172] Z. Meng, X. Zhan, J. Li, and Z. Pan, "An Enhancement Denoising Autoencoder for Rolling Bearing Fault Diagnosis," *Measurement*, vol. 130, pp. 448–454, 2018.

[173] L. Gondara and K. Wang, *Advances in Knowledge Discovery and Data Mining*, ch. MIDA: Multiple Imputation Using Denoising Autoencoders, pp. 260–272. Cham: Springer International Publishing, 2018.

[174] N. Abiri, B. Linse, P. Edén, and M. Ohlsson, "Establishing Strong Imputation Performance of a Denoising Autoencoder in a Wide Range of Missing Data Problems," *Neurocomputing*, vol. 365, pp. 137–146, 2019.

[175] B. Jiang, M. D. Siddiqi, R. Asadi, and A. Regan, "Imputation of Missing Traffic Flow Data Using Denoising Autoencoders," *Procedia Computer Science*, vol. 184, pp. 84–91, 2021.

[176] A. Alamoodi, B. Zaidan, A. Zaidan, O. Albahri, J. Chen, M. Chyad, S. Garfan, and A. Aleesa, "Machine Learning-Based Imputation Soft Computing Approach for Large Missing Scale and Non-reference Data Imputation," *Chaos, Solitons & Fractals*, vol. 151, p. 111236, 2021.

[177] K. K. R. Samal, K. S. Babu, and S. K. Das, "Temporal Convolutional Denoising Autoencoder Network for Air Pollution Prediction with Missing Values," *Urban Climate*, vol. 38, p. 100872, July 2021.

[178] S. Abirami and P. Chitra, "Regional Air Quality Forecasting Using Spatiotemporal Deep Learning," *Journal of Cleaner Production*, vol. 283, p. 125341, Feb 2021.

[179] M. Chen, H. Zhu, Y. Chen, and Y. Wang, "A Novel Missing Data Imputation Approach for Time Series Air Quality Data Based on Logistic Regression," *Atmosphere*, vol. 13, p. 1044, June 2022.

[180] A. R. Alsaber, J. Pan, and A. Al-Hurban, "Handling Complex Missing Data Using Random Forest Approach for an Air Quality Monitoring Dataset: A Case Study of Kuwait Environmental Data (2012 to 2018)," *International Journal of Environmental Research and Public Health*, vol. 18, p. 1333, Feb. 2021.

[181] I. Belachsen and D. M. Broday, "Imputation of Missing $PM_{2.5}$ Observations in a Network of Air Quality Monitoring Stations by a New kNN Method," *Atmosphere*, vol. 13, p. 1934, Nov. 2022.

[182] S. V. Mahadevkar, B. Khemani, S. Patil, K. Kotecha, D. R. Vora, A. Abraham, and L. A. Gabralla, "A Review on Machine Learning Styles in Computer Vision—Techniques and Future Directions," *IEEE Access*, vol. 10, pp. 107293–107329, 2022.

[183] S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, and S. X. Chen, "Cautionary Tales on Air-Quality Improvement in Beijing," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2205, p. 20170457, 2017.

[184] S. Chen, "Beijing Multi-Site Air-Quality Data." UCI Machine Learning Repository, 2019. DOI: https://doi.org/10.24432/C5RK5G.

[185] R. Rao, "Air Quality Data in India (2015 - 2020)." `https://www.kaggle.com/rohanrao/air-quality-data-in-india`, 2021. Accessed: 2022-02-04.

[186] D. C. Carslaw and K. Ropkins, "Openair — An R Package for Air Quality Data Analysis," *Environmental Modelling & Software*, vol. 27-28, pp. 52–61, Jan. 2012.

[187] D. Carslaw, "Openair: Open Source Tools for Air Quality Data Analysis." `https://davidcarslaw.github.io/openair/index.html`, 2023. Accessed: 2023-10-07.

[188] D. Carslaw, "The Openair Book." `https://bookdown.org/david_carslaw/openair/`, 2023. Accessed: 2023-10-07.

[189] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. `http://www.deeplearningbook.org`.

[190] N. Carter, ed., *Data Science for Mathematicians*. Boca Raton, FL: Chapman and Hall/CRC, Sept. 2020.

[191] I. Jebli, F.-Z. Belouadha, M. I. Kabbaj, and A. Tilioua, "Prediction of Solar Energy Guided by Pearson Correlation Using Machine Learning," *Energy*, vol. 224, p. 120109, June 2021.

[192] Y. Qi, Q. Li, H. Karimian, and D. Liu, "A Hybrid Model for Spatiotemporal Forecasting of $PM_{2.5}$ Based on Graph Convolutional Neural Network and Long Short-Term Memory," *Science of The Total Environment*, vol. 664, pp. 1–10, May 2019.

[193] E.-L. Silva-Ramírez and J.-F. Cabrera-Sánchez, "Co-active Neuro-Fuzzy Inference System Model as Single Imputation Approach for Non-monotone Pattern of Missing Data," *Neural Computing and Applications*, vol. 33, pp. 8981–9004, Feb. 2021.

[194] R. Noori, G. Hoshyaripour, K. Ashrafi, and B. N. Araabi, "Uncertainty Analysis of Developed ANN and ANFIS Models in Prediction of Carbon Monoxide Daily Concentration," *Atmospheric Environment*, vol. 44, pp. 476–482, Feb. 2010.

[195] S. Moazami, R. Noori, B. J. Amiri, B. Yeganeh, S. Partani, and S. Safavi, "Reliable Prediction of Carbon Monoxide Using Developed Support Vector Machine," *Atmospheric Pollution Research*, vol. 7, pp. 412–418, May 2016.

[196] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, 2014.

[197] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "A Fast and Scalable Architecture to Run Convolutional Neural Networks in Low Density FPGAs," *Microprocessors and Microsystems*, vol. 77, p. 103136, 2020.

[198] A. A. Asyraaf Jainuddin, Y. C. Hou, M. Z. Baharuddin, and S. Yussof, "Performance Analysis of Deep Neural Networks for Object Classification with Edge TPU," in *2020 8th International Conference on Information Technology and Multimedia (ICIMU)*, IEEE, Aug. 2020.

[199] E. BUBER and B. DIRI, "Performance Analysis and CPU vs GPU Comparison for Deep Learning," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, IEEE, Oct. 2018.

[200] Z. Li, F. Ge, F. Zhou, and N. Wu, "An a3c deep reinforcement learning fpga accelerator based on heterogeneous compute units," in *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*, IEEE, Nov. 2022.

[201] R. Navares and J. L. Aznarte, "Predicting Air Quality With Deep Learning LSTM: Towards Comprehensive Models," *Ecological Informatics*, vol. 55, p. 101019, 2020.

[202] X. Li, L. Peng, X. Yao, S. Cui, Y. Hu, C. You, and T. Chi, "Long Short-Term Memory Neural Network for Air Pollutant Concentration Predictions: Method Development and Evaluation," *Environmental Pollution*, vol. 231, pp. 997–1004, 2017.

[203] T. Xayasouk, H. Lee, and G. Lee, "Air Pollution Prediction Using Long Short-Term Memory (LSTM) and Deep Autoencoder (DAE) Models," *Sustainability*, vol. 12, no. 6, p. 2570, 2020.

[204] D. Seng, Q. Zhang, X. Zhang, G. Chen, and X. Chen, "Spatiotemporal Prediction of Air Quality Based on LSTM Neural Network," *Alexandria Engineering Journal*, vol. 60, no. 2, pp. 2021–2032, 2021.

[205] J. Xu, L. Chen, M. Lv, C. Zhan, S. Chen, and J. Chang, "HighAir: A Hierarchical Graph Neural Network-Based Air Quality Forecasting Method," *arXiv*, 2021. eprint 2101.04264.

[206] J. Zhao, F. Deng, Y. Cai, and J. Chen, "Long Short-Term Memory – Fully Connected (LSTM-FC) Neural Network for $PM_{2.5}$ Concentration Prediction," *Chemosphere*, vol. 220, pp. 486–492, 2019.

[207] Q. Zhang, J. C. Lam, V. O. Li, and Y. Han, "Deep-AIR: A Hybrid CNN-LSTM Framework for Fine-Grained Air Pollution Forecast," *arXiv*, 2020. eprint 2001.11957.

[208] D. Qin, J. Yu, G. Zou, R. Yong, Q. Zhao, and B. Zhang, "A Novel Combined Prediction Scheme Based on CNN and LSTM for Urban $PM_{2.5}$ Concentration," *IEEE Access*, vol. 7, pp. 20050–20059, 2019.

[209] T. Li, M. Hua, and X. Wu, "A Hybrid CNN-LSTM Model for Forecasting Particulate Matter ($PM_{2.5}$," *IEEE Access*, vol. 8, pp. 26933–26940, 2020.

[210] Q. Tao, F. Liu, Y. Li, and D. Sidorov, "Air Pollution Forecasting Using a Deep Learning Model Based on 1D Convnets and Bidirectional GRU," *IEEE Access*, vol. 7, pp. 76690–76698, 2019.

[211] R. Banner, Y. Nahshan, and D. Soudry, "Post Training 4-Bit Quantization of Convolutional Networks for Rapid-Deployment," in *Advances in Neural Information Processing Systems*, vol. 32, pp. 7950–7958, Curran Associates, Inc., 2019.

[212] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation," *arXiv*, 2020. eprint 2004.09602.

[213] P. Peng, M. You, W. Xu, and J. Li, "Fully Integer-Based Quantization for Mobile Convolutional Neural Network Inference," *Neurocomputing*, vol. 432, pp. 194–205, 2021.

[214] J. Li and R. Alvarez, "On the Quantization of Recurrent Neural Networks," *arXiv*, 2021. eprint 2101.05453.

[215] H. Fu, Y. Zhang, C. Liao, L. Mao, Z. Wang, and N. Hong, "Investigating $PM_{2.5}$ Responses to Other Air Pollutants and Meteorological Factors Across Multiple Temporal Scales," *Scientific Reports*, vol. 10, Sept. 2020.

[216] Defra, "The Air Quality Data Validation and Ratification Process." `https://uk-air.defra.gov.uk/assets/documents/Data_Validation_and_Ratification_Process_Apr_2017.pdf`. Accessed: 2024-02-11.

[217] M. Sajjad, M. Nasir, K. Muhammad, S. Khan, Z. Jan, A. K. Sangaiah, M. Elhoseny, and S. W. Baik, "Raspberry Pi Assisted Face Recognition Framework for Enhanced Law-Enforcement Services in Smart Cities," *Future Generation Computer Systems*, vol. 108, pp. 995–1007, 2017.

[218] X. Deng, T. Sun, F. Liu, and D. Li, "SignGD With Error Feedback Meets Lazily Aggregated Technique: Communication-Efficient Algorithms for Distributed Learning," *Tsinghua Science and Technology*, vol. 27, pp. 174–185, Feb. 2022.

[219] S. Henna and A. Davy, "Distributed and Collaborative High-Speed Inference Deep Learning for Mobile Edge with Topological Dependencies," *IEEE Transactions on Cloud Computing*, vol. 10, pp. 821–834, Apr. 2022.

[220] G. Song and W. Chai, "Collaborative Learning for Deep Neural Networks," *arXiv*, 2018. eprint 1805.11761.

[221] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv*, 2016. eprint 1602.05629.

[222] J. Mills, J. Hu, and G. Min, "Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, pp. 630–641, Mar. 2022.

[223] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, "Towards Personalized Federated Learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2022.

[224] P. Velentzas, M. Vassilakopoulos, and A. Corral, "GPU-Aided Edge Computing for Processing the K Nearest-Neighbor Query on SSD-Resident Data," *Internet of Things*, vol. 15, p. 100428, Sep 2021.

[225] H. Mi, K. Xu, D. Feng, H. Wang, Y. Zhang, Z. Zheng, C. Chen, and X. Lan, "Collaborative Deep Learning Across Multiple Data Centers," *Science China Information Sciences*, vol. 63, July 2020.

[226] B. Ghimire and D. B. Rawat, "Recent Advances on Federated Learning for Cybersecurity and Cybersecurity for Federated Learning for Internet of Things," *IEEE Internet of Things Journal*, vol. 9, pp. 8229–8249, June 2022.

[227] R. Kumar, A. A. Khan, J. Kumar, Zakria, N. A. Golilarz, S. Zhang, Y. Ting, C. Zheng, and W. Wang, "Blockchain-Federated-Learning and Deep Learning Models for COVID-19 Detection Using CT Imaging," *IEEE Sensors Journal*, vol. 21, pp. 16301–16314, July 2021.

[228] K. I.-K. Wang, X. Zhou, W. Liang, Z. Yan, and J. She, "Federated Transfer Learning Based Cross-Domain Prediction for Smart Manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 18, pp. 4088–4096, June 2022.

[229] D. Bousiotis, G. Allison, D. C. S. Beddows, R. M. Harrison, and F. D. Pope, "Towards Comprehensive Air Quality Management Using Low-cost Sensors for Pollution Source Apportionment," *npj Climate and Atmospheric Science*, vol. 6, Aug. 2023.

[230] H. Khreis, J. Johnson, K. Jack, B. Dadashova, and E. S. Park, "Evaluating the Performance of Low-Cost Air Quality Monitors in Dallas, Texas," *International Journal of Environmental Research and Public Health*, vol. 19, p. 1647, Jan. 2022.

[231] Z. Nieckarz, K. Pawlak, and J. A. Zoladz, "Health Risks for Children Exercising in an Air-polluted Environment can be Reduced by Monitoring Air Quality with Low-cost Particle Sensors," *Scientific Reports*, vol. 13, Oct. 2023.

[232] P. F. C. de Marinho, G. M. Santana, M. L. Felix, R. de Medeiros Morais, A. A. Santos, and R. M. de Jesus, "Intelligent, Low-cost, High-performance System for Environmental Air Quality Monitoring through Integrated Gas, Temperature, and Humidity Analysis," *International Journal of Environmental Science and Technology*, vol. 21, p. 4881–4898, Dec. 2023.

[233] S. Douch, M. R. Abid, K. Zine-Dine, D. Bouzidi, and D. Benhaddou, "Edge Computing Technology Enablers: A Systematic Lecture Study," *IEEE Access*, vol. 10, pp. 69264–69302, 2022.

[234] M. Ashouri, P. Davidsson, and R. Spalazzese, "Quality Attributes in Edge Computing for the Internet of Things: A Systematic Mapping Study," *Internet of Things*, vol. 13, p. 100346, Mar 2021.

[235] H. Li, K. Ota, and M. Dong, "Learning IoT in Edge: Deep Learning for the Internet of Things With Edge Computing," *IEEE Network*, vol. 32, pp. 96–101, Jan 2018.

[236] G. Zhao, G. Huang, H. He, H. He, and J. Ren, "Regional Spatiotemporal Collaborative Prediction Model for Air Quality," *IEEE Access*, vol. 7, pp. 134903–134919, 2019.

[237] F. Amato, F. Guignard, S. Robert, and M. Kanevski, "A Novel Framework for Spatio-Temporal Prediction of Environmental Data Using Deep Learning," *Scientific Reports*, vol. 10, Dec 2020.

[238] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. RuÃŸwurm, K. Kolar, and E. Woods, "Tslearn, A Machine Learning Toolkit for Time Series Data," *Journal of Machine Learning Research*, vol. 21, no. 118, pp. 1–6, 2020.

[239] X. Liu, T. Zhang, N. Hu, P. Zhang, and Y. Zhang, "The Method of Internet of Things Access and Network Communication Based on MQTT," *Computer Communications*, vol. 153, pp. 169–176, Mar. 2020.

[240] A. Velinov, A. Mileva, S. Wendzel, and W. Mazurczyk, "Covert Channels in the MQTT-Based Internet of Things," *IEEE Access*, vol. 7, pp. 161899–161915, 2019.

[241] T. A. Hilal and H. A. Hilal, "Turkish Text Compression via Characters Encoding," *Procedia Computer Science*, vol. 175, pp. 286–291, 2020.

[242] H. Yu, Q. Li, R. Wang, Z. Chen, Y. Zhang, Y.-a. Geng, L. Zhang, H. Cui, and K. Zhang, "A Deep Calibration Method for Low-Cost Air Monitoring Sensors With Multilevel Sequence Modeling," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, p. 7167–7179, Sept. 2020.

[243] M. A. Zaidan, N. H. Motlagh, P. L. Fung, A. S. Khalaf, Y. Matsumi, A. Ding, S. Tarkoma, T. Petaja, M. Kulmala, and T. Hussein, "Intelligent Air Pollution Sensors Calibration for Extreme Events and Drifts Monitoring," *IEEE Transactions on Industrial Informatics*, vol. 19, p. 1366–1379, Feb. 2023.

[244] Y. Hashmy, Z. U. Khan, F. Ilyas, R. Hafiz, U. Younis, and T. Tauqeer, "Modular Air Quality Calibration and Forecasting Method for Low-Cost Sensor Nodes," *IEEE Sensors Journal*, vol. 23, p. 4193–4203, Feb. 2023.

[245] T. Becnel, K. Tingey, J. Whitaker, T. Sayahi, K. Le, P. Goffin, A. Butterfield, K. Kelly, and P.-E. Gaillardon, "A Distributed Low-Cost Pollution Monitoring Platform," *IEEE Internet of Things Journal*, vol. 6, p. 10738–10748, Dec. 2019.

[246] P. Castello, C. Muscas, P. A. Pegoraro, and S. Sulis, "Improved Fine Particles Monitoring in Smart Cities by Means of Advanced Data Concentrator," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, p. 1–9, 2021.

[247] P. deSouza, A. Anjomshoaa, F. Duarte, R. Kahn, P. Kumar, and C. Ratti, "Air Quality Monitoring Using Mobile Low-Cost Sensors Mounted on Trash-Trucks: Methods Development and Lessons Learned," *Sustainable Cities and Society*, vol. 60, p. 102239, Sept. 2020.

[248] Sensirion, "SCD41." `https://sensirion.com/products/catalog/SCD41/`. Accessed: 2024-02-20.

[249] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks With Weights and Activations Constrained to +1 or -1," *arXiv*, 2016. eprint 1602.02830.

[250] T. Simons and D.-J. Lee, "A Review of Binarized Neural Networks," *Electronics*, vol. 8, p. 661, June 2019.

[251] L. Geiger and P. Team, "Larq: An Open-Source Library for Training Binarized Neural Networks," *Journal of Open Source Software*, vol. 5, p. 1746, Jan. 2020.

[252] Larq, "Key Concepts." `https://docs.larq.dev/larq/guides/key-concepts/`. Accessed: 2023-07-05.

[253] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey, "Meta-Learning in Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[254] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[255] B. McMahan and D. Ramage, "Federated Learning: Collaborative Machine Learning without Centralized Training Data." `https://blog.research.google/2017/04/federated-learning-collaborative.html`. Accessed: 2024-02-24.

[256] A. Garcia-Perez, R. Miñón, A. I. Torre-Bastida, and E. Zulueta-Guerrero, "Analysing edge computing devices for the deployment of embedded ai," *Sensors*, vol. 23, p. 9495, Nov. 2023.

[257] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for ai-enabled iot devices: A review," *Sensors*, vol. 20, p. 2533, Apr. 2020.

[258] T. Feng, Y. Sun, Y. Shi, J. Ma, C. Feng, and Z. Chen, "Air Pollution Control Policies and Impacts: A Review," *Renewable and Sustainable Energy Reviews*, vol. 191, p. 114071, 2024.

[259] N. L. Giménez, J. M. Solé, and F. Freitag, "Embedded Federated Learning Over a LoRa Mesh Network," *Pervasive and Mobile Computing*, vol. 93, p. 101819, June 2023.

Appendix A

Additional Evaluation of Correlation Coefficients

This section reports the Pearson's correlation coefficients obtained from Beijing and Delhi datasets, supplementing the explanation of Section 3.8.2.

A.1 Pearson's Correlation for Beijing Air Quality Data

Table A.1: Coefficient of correlation targeting CO in Beijing air quality data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.78	1.00								
S^3	0.78	0.82	1.00							
S^4	0.89	0.75	0.73	1.00						
S^5	0.93	0.78	0.77	0.92	1.00					
S^6	0.86	0.80	0.77	0.83	0.88	1.00				
S^7	0.78	0.76	0.79	0.75	0.78	0.78	1.00			
S^8	0.92	0.75	0.73	0.92	0.92	0.84	0.76	1.00		
S^9	0.82	0.72	0.74	0.79	0.80	0.78	0.84	0.81	1.00	
S^{10}	0.89	0.75	0.73	0.92	0.93	0.85	0.76	0.93	0.80	1.00

Table A.2: Coefficient of correlation targeting O_3 in Beijing air quality data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.87	1.00								
S^3	0.85	0.93	1.00							
S^4	0.87	0.82	0.79	1.00						
S^5	0.95	0.89	0.86	0.88	1.00					
S^6	0.92	0.88	0.85	0.84	0.92	1.00				
S^7	0.84	0.89	0.88	0.78	0.85	0.85	1.00			
S^8	0.96	0.89	0.86	0.89	0.96	0.92	0.86	1.00		
S^9	0.89	0.88	0.85	0.83	0.89	0.88	0.89	0.91	1.00	
S^{10}	0.94	0.88	0.85	0.89	0.95	0.92	0.85	0.96	0.90	1.00

A.2 Pearson's Correlation for Delhi Air Quality Data

Table A.3: Coefficient of correlation targeting $PM_{2.5}$ in Delhi air quality data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.90	1.00								
S^3	0.71	0.72	1.00							
S^4	0.86	0.84	0.72	1.00						
S^5	0.86	0.90	0.69	0.81	1.00					
S^6	0.81	0.84	0.71	0.81	0.84	1.00				
S^7	0.82	0.84	0.76	0.82	0.83	0.87	1.00			
S^8	0.83	0.81	0.67	0.75	0.76	0.71	0.74	1.00		
S^9	0.85	0.85	0.73	0.82	0.81	0.79	0.80	0.79	1.00	
S^{10}	0.85	0.89	0.67	0.81	0.88	0.82	0.78	0.75	0.80	1.00

Table A.4: Coefficient of correlation targeting NO_2 in Delhi air quality data.

	S^1	S^2	S^3	S^4	S^5	S^6	S^7	S^8	S^9	S^{10}
S^1	1.00									
S^2	0.41	1.00								
S^3	0.04	0.07	1.00							
S^4	0.13	0.32	-0.03	1.00						
S^5	0.24	0.35	0.13	0.16	1.00					
S^6	0.35	0.65	0.01	0.27	0.30	1.00				
S^7	0.12	0.25	0.05	0.29	0.38	0.23	1.00			
S^8	0.50	0.58	0.05	0.33	0.37	0.55	0.26	1.00		
S^9	0.32	0.21	0.09	0.00	0.02	0.22	0.07	0.29	1.00	
S^{10}	0.33	0.49	-0.24	0.34	0.27	0.54	0.30	0.48	0.12	1.00

A.3 Neighbouring Stations Selection for Beijing Air Quality Data

Table A.5: Strongest correlation coefficient for neighbouring stations selection in Beijing air quality data.

	Strongest corr. coeff. (O_3)			**Strongest corr. coeff. (CO)**		
Target station	1st	2nd	3rd	1st	2nd	3rd
S^1	S^8	S^5	S^{10}	S^5	S^8	S^{10}
S^2	S^3	S^8	S^7	S^3	S^6	S^5
S^3	S^2	S^7	S^8	S^2	S^7	S^1
S^4	S^8	S^{10}	S^5	S^8	S^5	S^{10}
S^5	S^8	S^1	S^{10}	S^{10}	S^1	S^8
S^6	S^8	S^5	S^1	S^5	S^1	S^{10}
S^7	S^2	S^9	S^3	S^9	S^3	S^5
S^8	S^{10}	S^1	S^5	S^{10}	S^1	S^5
S^9	S^8	S^{10}	S^1	S^7	S^1	S^8
S^{10}	S^8	S^5	S^1	S^5	S^8	S^4

A.4 Neighbouring Stations Selection for Delhi Air Quality Data

Table A.6: Strongest correlation coefficient for neighbouring stations selection in Delhi air quality data.

	Strongest corr. coeff. (NO_2)			**Strongest corr. coeff. ($PM_{2.5}$)**		
Target station	1^{st}	2^{nd}	3^{rd}	1^{st}	2^{nd}	3^{rd}
S^1	S^8	S^2	S^6	S^2	S^4	S^5
S^2	S^6	S^8	S^{10}	S^1	S^5	S^{10}
S^3	S^5	S^9	S^2	S^7	S^9	S^4
S^4	S^{10}	S^8	S^2	S^1	S^2	S^9
S^5	S^7	S^8	S^2	S^2	S^{10}	S^1
S^6	S^2	S^8	S^{10}	S^7	S^2	S^5
S^7	S^5	S^{10}	S^4	S^6	S^2	S^7
S^8	S^2	S^6	S^1	S^1	S^2	S^9
S^9	S^1	S^8	S^6	S^1	S^2	S^4
S^{10}	S^6	S^2	S^8	S^2	S^5	S^1

Appendix B

Additional Model Evaluation Metrics

This section reports the RMSE and MAE for all nodes as discussed in Subchapter 4.5. Tables B.1 and B.2 show the RMSE and MAE values, respectively. Table B.1 shows the effect of quantisation techniques on the RMSE and MAE values.

Table B.1: Comparison of RMSE values for $PM_{2.5}$ prediction using different model architectures for all nodes.

No.	Architectures	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8	Node9	Node10	Node11	Node12
	Simple Models Group I												
1	RNN	18.485	17.961	19.356	20.655	19.332	21.088	17.186	19.560	21.046	17.892	20.920	23.149
2	LSTM	17.786	18.203	19.802	19.775	19.186	19.434	17.620	18.208	21.143	17.586	17.946	21.180
3	GRU	18.367	18.353	18.475	22.143	20.578	20.459	17.449	19.471	20.914	18.435	19.267	22.257
4	Bidirectional RNN	19.377	17.383	17.799	20.703	18.864	20.737	17.522	18.740	20.125	17.450	18.442	21.996
5	Bidirectional LSTM	18.016	17.084	18.967	20.806	18.829	19.563	17.547	19.041	19.299	17.144	17.335	22.520
6	Bidirectional GRU	18.603	17.606	17.650	21.290	19.275	19.339	16.899	18.443	18.764	17.052	17.138	21.489
	Hybrid Models Group II												
7	CNN-ANN	17.757	16.838	17.841	19.752	19.207	19.793	17.174	18.777	20.364	17.635	17.041	21.537
8	CNN-RNN	18.227	16.813	17.445	20.021	18.420	19.303	16.952	18.418	18.686	16.713	17.018	21.492
9	CNN-LSTM	17.652	16.801	18.387	19.743	19.184	19.228	17.261	18.242	18.663	17.040	17.209	21.160
10	CNN-GRU	17.244	16.742	17.733	19.667	19.759	20.897	17.001	18.391	19.652	16.611	17.081	21.818
11	CNN-Bidirectional RNN	17.334	16.804	17.571	19.514	18.520	19.083	18.062	18.109	20.697	16.908	16.962	21.421
12	CNN-Bidirectional LSTM	17.344	17.981	20.118	20.267	19.640	19.071	16.862	18.058	18.427	16.676	17.071	21.667
13	CNN-Bidirectional GRU	17.462	17.518	20.038	21.214	18.642	19.098	16.800	19.131	18.497	17.230	16.944	21.268
	Hybrid Models Group III												
14	CNN-ANN	17.160	17.661	18.493	18.074	17.282	18.598	19.235	17.969	18.196	17.018	17.705	19.134
15	CNN-RNN	15.672	17.159	18.377	18.135	16.933	18.262	16.135	18.596	20.215	16.053	15.981	19.494
16	**CNN-LSTM (proposed model)**	**15.268**	**15.710**	**17.082**	**17.706**	**16.557**	**17.743**	**15.493**	**16.172**	**17.920**	**14.894**	**14.951**	**18.962**
17	CNN-GRU	17.169	16.136	20.252	18.900	17.034	20.692	18.166	18.327	18.897	16.250	17.031	19.098
18	CNN-Bidirectional RNN	17.365	18.182	18.229	19.658	17.591	18.301	16.045	16.818	18.068	15.513	15.154	19.426
19	CNN-Bidirectional LSTM	15.643	16.647	18.085	17.941	17.574	18.544	15.845	16.584	19.187	15.530	16.423	19.038
20	CNN-Bidirectional GRU	16.089	16.855	19.121	18.193	17.269	19.350	15.789	16.545	18.134	15.874	15.785	19.267

Table B.2: Comparison of MAE values for $PM_{2.5}$ prediction using different model architectures for all nodes.

No.	Architectures	Node1	Node2	Node3	Node4	Node5	Node6	Node7	Node8	Node9	Node10	Node11	Node12
	Simple Models Group I												
1	RNN	10.636	10.123	10.206	11.412	10.904	12.043	8.636	11.080	11.571	10.686	12.403	12.139
2	LSTM	10.230	10.710	10.376	10.670	10.906	11.174	8.786	10.499	11.829	10.880	10.523	11.078
3	GRU	10.664	10.769	9.881	12.505	11.635	11.57	8.880	11.548	11.977	11.053	11.183	11.947
4	Bidirectional RNN	12.257	9.898	9.349	11.871	10.743	11.887	9.040	10.571	11.154	10.587	11.144	11.560
5	Bidirectional LSTM	10.427	9.462	10.335	11.710	10.584	11.098	8.764	10.826	10.461	10.357	9.765	11.710
6	Bidirectional GRU	10.944	10.345	9.429	13.051	10.924	10.665	8.338	10.171	10.124	10.404	9.658	11.348
	Hybrid Models Group II												
7	CNN-ANN	10.321	9.387	9.671	10.679	11.351	11.333	8.714	11.267	11.466	10.757	9.385	11.558
8	CNN-RNN	10.906	9.959	8.888	11.311	10.303	10.746	8.306	10.419	9.961	10.043	9.615	11.123
9	CNN-LSTM	10.203	9.365	9.727	10.582	11.176	10.571	8.914	10.027	10.407	10.084	9.612	11.216
10	CNN-GRU	9.552	9.423	9.213	10.622	11.771	12.006	8.420	10.410	11.313	10.077	9.492	11.52
11	CNN-Bidirectional RNN	10.001	9.443	9.383	10.274	10.562	10.312	9.750	10.179	12.417	10.072	9.683	10.833
12	CNN-Bidirectional LSTM	10.054	10.439	11.576	11.778	12.542	10.670	8.313	10.245	9.858	9.895	9.700	11.282
13	CNN-Bidirectional GRU	10.486	9.923	11.638	12.638	10.590	10.440	8.273	11.383	10.290	10.684	9.466	11.210
	Hybrid Models Group III												
14	CNN-ANN	10.307	10.351	9.221	9.872	10.168	10.446	12.906	10.637	10.245	10.564	10.191	9.985
15	CNN-RNN	9.162	9.602	8.998	10.106	9.510	10.069	8.326	11.068	12.817	10.055	9.543	10.295
16	**CNN-LSTM (proposed model)**	**8.778**	**8.873**	**8.803**	**9.653**	**9.228**	**9.590**	**7.607**	**9.116**	**9.588**	**8.993**	**8.486**	**9.641**
17	CNN-GRU	9.665	8.967	11.431	11.118	9.981	11.852	11.023	10.252	10.838	10.068	10.274	9.878
18	CNN-Bidirectional RNN	10.443	10.265	9.054	11.354	9.978	10.115	8.286	9.676	9.709	9.213	8.527	10.269
19	CNN-Bidirectional LSTM	8.853	9.297	9.657	9.763	10.177	9.746	7.980	10.095	10.167	9.203	10.149	9.815
20	CNN-Bidirectional GRU	9.512	9.644	9.838	10.037	9.331	10.863	7.977	9.665	9.780	9.786	9.143	10.111

Appendix C

Post-Training Quantisations

Table C.1 shows the effect of quantisation techniques on the RMSE and MAE values as discussed in Subchapter 4.5.

Table C.1: Effect of post-training quantisation techniques on RMSE and MAE values.

	Node1	**Node2**	**Node3**	**Node4**	**Node5**	**Node6**	**Node7**	**Node8**	**Node9**	**Node10**	**Node11**	**Node12**
Initial TensorFLow model												
RMSE	15.268	15.710	17.082	17.706	16.557	17.743	15.493	16.172	17.920	14.894	14.951	18.962
MAE	8.778	8.873	8.803	9.653	9.228	9.590	7.607	9.116	9.588	8.935	8.486	9.641
TFLite—without quantisation												
RMSE	15.268	15.710	17.082	17.706	16.557	17.743	15.493	16.172	17.920	14.894	14.951	18.962
MAE	8.778	8.873	8.803	9.653	9.228	9.590	7.607	9.116	9.588	8.935	8.486	9.641
TFLite—dynamic range quantisation												
RMSE	15.270	15.710	17.116	17.679	16.580	17.741	15.490	16.189	17.906	14.905	14.994	18.974
MAE	8.784	8.872	8.844	9.639	9.236	9.592	7.611	9.129	9.604	8.943	8.550	9.678
TFLite—integer with float fallback												
RMSE	15.704	16.102	17.506	18.108	17.032	18.167	15.694	16.536	18.758	15.467	15.216	19.690
MAE	9.418	9.420	9.562	10.425	10.004	10.394	8.081	9.674	10.792	9.694	9.048	10.785
TFLite—integer-only quantisation												
RMSE	15.704	16.102	17.506	18.108	17.032	18.167	15.694	16.536	18.758	15.467	15.216	19.690
MAE	9.418	9.420	9.562	10.425	10.004	10.394	8.081	9.674	10.792	9.694	9.048	10.785
TFLite—float16 quantisation												
RMSE	15.268	15.708	17.082	17.707	16.557	17.742	15.494	16.171	17.920	14.896	14.952	18.962
MAE	8.777	8.871	8.803	9.654	9.228	9.589	7.607	9.115	9.588	8.936	8.486	9.642