Article

Optimising Deep Learning at the Edge for Accurate Hourly Air Quality Prediction

I Nyoman Kusuma Wardana, Julian W. Gardner, and Suhaib A. Fahmy

Abstract: Accurate air quality monitoring requires processing of multi-dimensional, multi-location sensor data, that has previously been considered in centralised machine learning models. These are often unsuitable for resource-constrained edge devices. In this article, we address this challenge by (1) designing a novel hybrid deep learning model for hourly PM$_{2.5}$ pollutant prediction, (2) optimising the obtained model for edge devices, and (3) examining model performance running on the edge devices in terms of both accuracy and latency. The hybrid deep learning model in this work is composed of 1D Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) to predict hourly PM$_{2.5}$ concentration. Results show that our proposed model outperforms other deep learning models, evaluated by calculating RMSE and MAE errors. After selecting the best model, we optimise the model for edge devices, using Raspberry Pi 3 Model B+ (RPi3B+) and Raspberry Pi 4 Model B boards (RPi4B). The lite version produced 4 times smaller file size compared to the original version. From the lite version, further size reduction can be achieved by implementing different post-training quantisations. About a 47% reduction can be achieved by dynamic range quantisation, about 45% by full integer quantisation, and about 35% by float16 quantisation. A total of 8272 hourly samples were continuously executed directly at the edge. The RPi4B executed these data two times faster compared to the RPi3B+ in all quantisation modes. Full-integer quantisation produced the most effective time execution, with latencies of 2.19 seconds and 4.73 seconds for RPi4B and RPi3B+, respectively.

Keywords: Air quality prediction; PM$_{2.5}$ prediction; deep learning; post-training quantisation; edge computing

1. Introduction

Edge computing refers to the deployment of computation closer to data sources (edge) [1], rather than more centrally as is the case with cloud computing. It can address latency, privacy, and scalability issues faced by cloud-based systems [2,3]. In terms of latency, moving computation closer to the data sources decreases end-to-end network latency. In terms of privacy, computation performed at the edge or at a local trusted edge server prevents data from leaving the device, potentially reducing the chance for cyber-attacks. In terms of scalability, edge computing can avoid network bottlenecks at central servers by enabling a hierarchical architecture of edge nodes [4]. Moreover, edge computing can address energy-aware and bandwidth saving applications [5].

For data processing and information inference, it is also possible to embed intelligence at edge devices, which can be enabled by Machine Learning (ML) algorithms [6,7]. Deep Learning [8], a subset of Machine Learning, can be implemented on edge devices, such as mobile phones, wearables, and IoT nodes. 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 its training phase.
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 [9].

Efficient deep learning design (e.g. deep neural networks) for embedded devices can be achieved by optimising both algorithmic (software) and hardware aspects [9]. At the algorithmic level, two methods can be implemented, namely model design and model compression [4]. In model design, researchers focus on designing deep learning models with a reduced number of parameters. This results in reduced memory size and latency, while trying to maintain high accuracy. In model compression, models are adapted for edge deployment by applying a number of different techniques on a trained model, such as parameter quantisation, parameter pruning, and knowledge distillation.

Parameter quantisation is a conversion technique to reduce model size with minimal degradation in model accuracy. Parameter pruning eliminates the least essential values in weight tensors. This method is related to the dropout technique [10]. Knowledge distillation [11] creates a smaller deep learning model by mimicking the behaviour of a larger model. It can be realised by training the smaller model using the outputs obtained from the larger model. At the hardware level, the training and inferencing processes of deep learning models can be accelerated by the computation power of server-class central processing units (CPUs), graphics processing unit (GPUs), tensor processing units (TPUs), neural processing units (NPUs), application-specific circuits (ASICs) and field-programmable gate arrays (FPGAs). Deep learning accelerators with diversity of layers and kernels built from custom low density FPGAs can provide high-speed computation while maintaining reconfigurability [12]. Both ASICs and FPGAs are generally more energy-efficient than conventional CPUs and GPUs [4].

Deep learning at the edge can be applied for air pollution prediction. Air pollution exposure causes negative impacts on human health [13,14] and economic activities [15]. Among many air pollutants, particulate matter (PM) harms the human respiratory system, as it may enter into the human respiratory tract or even the lungs through inhalation [16,17]. Particulate matter can be in the form of PM$_{2.5}$ (particulate matter with diameter less than 2.5 $\mu$m, or fine particles) and PM$_{10}$ (diameter less than 10 $\mu$m, or inhalable particles) [18]. It may lead to lung cancer [18], affect cardiovascular diseases [19], and even result in death [20]. Particulate matter causes premature death, and it is considered as responsible for 16% of global deaths [21]. The complex mixture of particulate matter and other gases like ozone was recorded to be associated with an all-cause death rate of up to 9 million in 2015 [22]. In this connection, building a forecasting system based on hourly air quality prediction plays an important role in health alerts [23].

Many works on PM$_{2.5}$ prediction considered only the performance evaluation by comparing predicted values to the dataset for accuracy. 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. In this work, our main contribution lies in (1) designing a novel hybrid deep learning model for PM$_{2.5}$ pollutant prediction based on the selected dataset, (2) optimising the obtained model to a lightweight version suitable for edge devices, and (3) examining the model performance when running on the edge devices. We implement post-training quantisation as a part of the algorithmic-level optimisation. This technique compresses model parameters by converting floating-point numbers to reduced precision numbers. Quantisation can improve CPU and hardware accelerator latencies and potentially reduce the original deep learning model size.

The remainder of this paper is structured as follows. Section 2 describe the related works and clarify our originality. Section 3 describes some of the basic theories related to this research. Section 4 explains the dataset and the requirement preprocessing. This section also defines our proposed deep learning model and gives a brief overview of the edge devices used in this work. Section 5 depicts the results of our proposed model in
terms of prediction accuracy. This section also explains the model optimisation results for the selected edge devices. Section 6 offers conclusions and discusses future work.

2. Related Works

Numerous works have been published in the last few years around the use of deep learning for air quality prediction. Navares and Aznarte [24] implemented Long Short-Term Memory (LSTM) to predict PM$_{10}$ and other air pollutants. They demonstrated a Recurrent Neural Network (RNN) that can map input sequences to output sequences by including the past context into its internal state, making it suitable for time-series problems. However, as the time series grows, relevant information occurs further in the past making RNNs unable to connect suitable information. Moreover, RNNs suffer from the vanishing gradient problem due to cyclic loops.

LSTMs, a variation of RNNs, are capable of learning long-term dependencies and able to deal with vanishing gradients. Li et al. [25] predicted hourly PM$_{2.5}$ concentration by using an LSTM model. The authors combined historical air pollutant data, meteorological data, and time stamp data. For one-hour predictions, the proposed LSTM model outperformed other models such as the spatiotemporal deep learning (STDL), time-delay neural network (TDNN), autoregressive moving average (ARMA), and support vector regression (SVR) models. Xayasouk et al. [26] implemented LSTM and Deep Autoencoder to predict 10-day of PM$_{2.5}$ and PM$_{10}$ concentrations. By varying the input batch size and recording the total average of the model performances, the proposed LSTM model are more accurate than the DAE model. Seng et al. [27] used LSTM model to predict air pollutant data (PM$_{2.5}$, CO, NO$_2$, O$_3$, SO$_2$) at 35 monitoring stations in Beijing. They proposed a comprehensive model called multi-output and multi-index of supervised learning (MMSL) based on spatiotemporal data of present and surrounding stations. The effectiveness of the proposed model was compared to the existing time series model (Linear Regression, SVR, Random Forest, ARMA), and baseline models (CNN-LSTM, CNN-Bidirectional RNN). Xu et al. [28] proposed a framework called HighAir. This framework used hierarchical graph neural network based on encoder-decoder architecture. Both encoder and decoder consist of LSTM network. Other works based on LSTMs are also reported in [29] and [30].

Other researchers have also proposed hybrid deep learning models. Zhao et al. [31] attempted to compare ANN, LSTM, and LSTM-Fully Connected (LSTM-FC) models to predict PM$_{2.5}$ concentrations. They found that LSTM-FC produced better predictive performance. Their model consists of two parts. In the first, the LSTM is applied to model the local PM$_{2.5}$ concentrations. In the second, the fully connected network is used to capture the spatial dependencies between the central station and neighbour stations. The combination of CNN and LSTM models have also been actively explored [16,32–34]. CNN-LSTM may improve the accuracy for PM$_{2.5}$ prediction, as reported by Li et al. [35], where the authors implemented a 1D CNN to extract features from sequence data and used LSTM to predict future values. In many real problems, input data may come from many resources, constructing spatio-temporal dependencies as explained by Qi et al. [32]. Gated Recurrent Units (GRUs), another variant of RNNs, have also been applied to PM$_{2.5}$ prediction. Tao et al. [36] combined a 1-dimensional CNN with bi-directional GRU to forecast PM$_{2.5}$ concentration. They examined attributes in the dataset to find the best input features for the proposed model, and evaluate the model performance based on mean absolute error (MAE), root mean square error (RMSE), and symmetric mean absolute percentage error (SMAPE). Powered by AI cloud computing to interpret multimode data, a new framework based on CNN-RNN is proposed by Chen et al. [37] to predict PM$_{2.5}$ values. The framework consists of input preprocessing stages, CNN encoder, RNN-based learning network, and CNN Decoder. Input model considers the spatiotemporal factor in the form of 4D sequence data of heat maps.

A variety of deep learning optimisation techniques have been proposed recently. Even though the selected case study in these works might not be related to air quality
prediction, we review some of them as follows. The post-trained model size can be
reduced by quantising weights and activation function, without retraining the model.
This method is called the post-training quantisation [38]. Banner et al. [38] proposed 4-bit
post-training quantisation for CNN. They designed an efficient quantisation method by
minimising mean-squared quantization error at the tensor level and avoiding retraining
the model. Moreover, mathematical background review for integer quantisation and
its implementation on many existing pre-trained neural network models have been
presented by Wu et al. [39]. With 8-bit integer quantisation, the obtained accuracy
either matches or is within 1% of the floating-point model. Intended for mobile edge
device, Peng et al. [40] 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 [41] specifically proposed the integer-only quantisation
method for LSTM neural network. The obtained result is accurate, efficient and fast
to execute. Moreover, the proposed method has been deployed to a variety of target
hardware.

To the best of our knowledge, the previous explained works related to air quality
prediction do not specifically optimise their obtained models to the 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 device.
We implemented post-training quantisation techniques to the baseline model using
tools provided by TensorFlow framework [42] and evaluated the optimised model
performance on Raspberry Pi boards. Table 1 summarises the aforementioned research
related to air quality prediction. In the last rows of the table, we stated our work
contribution.

3. Related Theory

3.1. One-Dimensional Convolutional Neural Network

Many articles focus on two-dimensional Convolutional Neural Network (2D CNN)
models. These networks work best for image classifications problems. The same ap-
proach can be applied to one-dimensional (1D) sequences of data (time-series data). A
1D CNN model learns to extract features from time-series data and maps the internal
features of the sequence. This model is very efficient to gather information from raw
time-series data directly, especially from shorter (fixed-length) segments of the overall
dataset.

In our case study, we extract time-series air pollutant data such as PM$_{2.5}$, PM$_{10}$, SO$_2$,
CO, NO$_2$, and O$_3$, and meteorological data such as temperature, air pressure, dew point,
wind direction and wind speed. Figure 1 illustrates how the feature detector (or kernel)
of the 1D CNN slides across the features, by assuming that our input model is only the
pollutant data.

![Figure 1. A feature detector of 1D CNN slides over time-series data.](image)
Table 1. Summary of the works related to air quality prediction. We studied current research trend on deep learning application and extend this body of work around deep learning models by analysing the deployment of these models to edge device. In the last row of the table, we stated our work contribution

| Reference | Proposed Model                  | Target Prediction          | Model Optimisation | Note                                                                                                                                 |
|-----------|---------------------------------|-----------------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| [16]      | EWT-MAEGA-NARX                  | PM$_{2.5}$, SO$_2$, NO$_2$, and CO | No                 | Combining the EWT, MAEGA and NARX neural networks for multi-step air pollutant predictions                                     |
| [24]      | LSTM                            | CO, Average NO$_2$, Average O$_3$, Average PM$_{10}$, Average SO$_2$, Average Plantago, Average Poaceae | No                 | Combining different type of group inputs for the LSTM network. Choosing a single but comprehensive model rather than multiple individual models. |
| [25]      | LSTM Extended                   | PM$_{2.5}$                  | No                 | Adding auxiliary Inputs (meteorological data, month of year, and hour of day) for LSTM network                                    |
| [26]      | LSTM and Deep Autoencoder       | PM$_{2.5}$, PM$_{10}$       | No                 | Varying the input batch size and recording the total average of the model performances                                              |
| [27]      | LSTM-based MMSL Framework       | PM$_{2.5}$, CO, NO$_2$, O$_3$, SO$_2$ | No                 | Considering spatiotemporal factors                                                                                                 |
| [28]      | HighAir Framework               | Air Quality Index (AQI), Points of Interest (POI), weather data | No                 | Implementing graph neural network based on encoder-decoder architecture. Both encoder and decoder consist of LSTM network |
| [29]      | Geo-LSTM                        | PM$_{2.5}$                  | No                 | Implementing geo-layer to integrate the spatial-temporal correlation from other monitoring stations                                |
| [30]      | An ensemble LSTM                | PM$_{2.5}$                  | No                 | Performing 3 steps: ensemble empirical mode decomposition (EMD), LSTM, inverse EMD                                                 |
| [31]      | LSTM-FC                         | PM$_{2.5}$                  | No                 | Using LSTM-based temporal simulator and NN-based spatial combinatories                                                              |
| [32]      | Graph CNN-LSTM                  | PM$_{2.5}$                  | No                 | Implementing graph signal for spatial dependency modelling                                                                        |
| [33]      | Deep-AIR                        | PM$_{2.5}$, PM$_{10}$, NO$_2$, CO, O$_3$ | No                 | Combining CNN-LSTM networks, using ResNet to let the model learn large information                                                  |
| [34]      | CNN-LSTM                        | PM$_{2.5}$                  | No                 | Using CNN to eliminate redundancy and obtain the features, using LSTM to predict the time-series                                 |
| [35]      | CNN-LSTM                        | PM$_{2.5}$                  | No                 | Using 1D CNN for feature extraction and the LSTM layer for prediction                                                              |
| [36]      | CBGRU                           | PM$_{2.5}$                  | No                 | Using 2 layers CNN and Bidirectional GRU                                                                                                |
| [37]      | LSTM                            | PM$_{2.5}$                  | No                 | Using cloud server node with AI technology                                                                                         |
| Our work  | CNN-LSTM                        | PM$_{2.5}$                  | Yes                | Implementing post-training quantisation for model optimisation. Deploying deep learning model to edge devices                      |

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

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

For example, if we have $n = 6$, $k = 3$, and $s = 1$, then the output will be:
If it is assumed that there is no padding applied to the input data, then the length of output data $o$ is given by:

$$o = \left\lfloor \frac{n - k}{s} \right\rfloor + 1 \quad (2)$$

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

Aside from the convolutional layer, there is a pooling layer, which downsamples the dimensions of the convolution output. There are several kinds of pooling layer, such as max pooling and average pooling. Max-pooling takes the maximum of the window, whereas average pooling takes the average value of the window. The dimensions output by the convolutional layers may be greater than one. The flattening process aims to reduce the output dimension to form a flat structure suitable for fully connected layers.

### 3.2. Long Short-Term Memory Cells

Long Short-Term Memory (LSTM) [43] 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 shows the LSTM network cell structure.

![LSTM cell structure](image)

Figure 2. An LSTM cell structure.

As shown in Figure 2, the network inputs and outputs on the LSTM structure are described as follows:

$$F_t = \sigma(W_f \cdot [H_{t-1}, X_t] + b_f) \quad (3)$$
$$I_t = \sigma(W_i \cdot [H_{t-1}, X_t] + b_i) \quad (4)$$
$$\tilde{C}_t = \tanh(W_c \cdot [H_{t-1}, X_t] + b_c) \quad (5)$$
$$C_t = F_t \ast C_{t-1} + I_t \ast \tilde{C}_t \quad (6)$$
$$O_t = \sigma(W_o \cdot [H_{t-1}, X_t] + b_o) \quad (7)$$
$$H_t = O_t \ast \tanh(C_t) \quad (8)$$
$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (9)$$
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (10)$$
With $W_f$, $W_i$, $W_c$ and $W_o$ as input weights, $b_f$, $b_i$, $b_c$, and $b_o$ as biases, $t$ is the current time, $t - 1$ represents the previous state, $X$ is the input, $H$ is the output, and $C$ is the status of the cell. The notation $\sigma$ 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 to let 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.

### 3.3. Error Measures

In this work, root mean square error (RMSE) and mean absolute error (MAE), are used as evaluation parameters. RMSE and MAE can be calculated using Equation (11) and Equation (12), respectively.

\[
RMSE = \sqrt{\frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{n}}
\]

\[
MAE = \frac{\sum_{i=1}^{n} |Y_i - \hat{Y}_i|}{n}
\]

where $n$ is the total number of data samples, $Y_i$ are the measured values, and $\hat{Y}_i$ are the predicted values.

### 3.4. Correlation Coefficient between Features

Correlation analysis can provide information about the correlation of two time-series features. In our work, we evaluate the time-series of air quality parameters. If time series data is vectored as $X = (x_1, x_2, \ldots, x_n)$ and there is another vector $Y = (y_1, y_2, \ldots, y_n)$, then the correlation coefficient $r$ of the two vectors is calculated using the following equation:

\[
r = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \sqrt{n \sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}}
\]

The value of $r$ in Equation (13) is the Pearson correlation coefficient. When $0 < r < 1$, it is said that both features have positive correlations, and when $-1 < r < 0$ they have negative correlation. A value of 0 indicates that there is no correlation between the features. When the absolute value of $r$ approaches 1, then both features higher correlation. A value of 1 indicates two series of data are identical.

### 3.5. TensorFlow Post-training Quantisation

In this work, we built deep learning models using TensorFlow 2.2 framework [42]. TensorFlow provides a lightweight version called TensorFlow Lite that offers various tools to convert and run TensorFlow models on various edge devices, including mobile, embedded and IoT devices. The deep learning models were built, trained, tested and optimised on a desktop computer. From these steps, a lightweight (optimised) deep learning model was obtained. The optimised model was then deployed to the Raspberry Pi boards. To port the model, execute it, and define the inputs/outputs on the Raspberry Pi boards, it is necessary to install the TensorFlow Lite Interpreter library.

TensorFlow provides tools for optimising deep learning model called the TensorFlow Model Optimisation Toolkit. Depending on the requirements of our applications, we can choose pre-optimised model, post-training or training-time optimisation tools. In this work, we focus on post-training quantisation. In post-training quantisation, the optimisation takes place after training process has been completed. There are three post-training quantisation methods provided 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 the 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 like EdgeTPU. In this method, the conversion fails if the model has unsupported operation. Finally, float16 quantisation converts weights to float16 (16-bit floating-point numbers). Figure 3 depicts the post-training methods graphically.

4. Materials and Methods

4.1. Dataset and Preprocessing

In this study, we used a dataset provided by Zhang et al. [45], which can be downloaded from the University of California, Irvine (UCI) Machine Learning Repository page. The dataset captures Beijing air quality, collected from 12 different Guokong (state controlled) monitoring sites in Beijing and its surroundings [45]. These 12 monitoring sites are Aotizhongxin, Changping, Dingling, Dongsi, Guanyuan, Gucheng, Huairou, Nongzhanguan, Shunyi, Tian tan, Wanliu, and Wanshouxigong.

Regardless of the real geographical location and the ability for each monitoring site to gather both pollutant and meteorological data, we consider every monitoring site merely as a Node. Therefore, we model a complex monitoring site as a simple node. The term node is closely associated with the end device, where the edge computing is usually executed. We are interested only in the data obtained by each node and its correlation with other nodes. We number the 12 monitoring sites as mentioned above, from Aotizhongxin as Node 1, Changping as Node 2, Dingling as Node 3, Dongsi as Node 4, etc.

There are a total of 12 columns (features) and 36,064 rows in the dataset, collected from 01 March 2013 to 28 February 2017. Each row in the dataset is hourly data, composed of pollutant data (PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_2$, and O$_3$) and meteorological data (temperature, air pressure, dew point, rain, wind direction and wind speed). We split data into training data and test data. Data from 01 March 2013 to 20 March 2016 were used as training data, whereas data from 21 March 2016 to 28 February 2017 were used as test data. By using this division, there are a total of 26,784 training data and 8,280 test data. In this work, we focus on predicting the PM$_{2.5}$ concentrations. We evaluated the best model for a short-term prediction, that is 1-hour particulate matter concentra-
Figure 4. PM$_{2.5}$ concentration gathered by Node 1 (Aotizhongxin monitoring site). This figure depicts the whole period of PM$_{2.5}$ concentration, starting from 01 March 2013 to 28 February 2017.

Data can be numerical or categorical. The attribute of wind direction in the dataset, which is categorical data, admits 16 values: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, and NNW. These features were label encoded. Dividing 360 degrees by 16 (number of wind directions) and applying floor rounding, we found a label for N is 360, NNE is 22, NE is 45, ENE is 67, etc. Instead of labelling N as 0, we assign it as 360. For missing values in the dataset, we filled them with the last timestamp data.

Besides labelling the categorical data and filling in missing values, we scaled the input features during the training and testing phases. Feature scaling is a method used to normalise the range of independent variables or features of data. In data processing, it is also known as data normalisation and is generally performed during the data preprocessing step. In this work, all inputs are normalised to the range of 0 and 1 (min-max scaler). The general formula for a min-max of $[0, 1]$ is given as:

$$x' = \frac{x - \text{min}(x)}{\text{max}(x) - \text{min}(x)}$$

(14)

4.2. Feature Selection

Our work aims to predict PM$_{2.5}$. As shown in Table 2, PM$_{2.5}$ are strongly correlated to PM$_{2.5}$, NO$_2$ and CO (with $r > 0.6$), moderately correlated to SO$_2$ (with $r = 0.49$), and weakly correlated to O$_3$ (with $r = -0.15$). It is also found that rain (RAIN), air pressure (PRES), and temperature (TEMP) have the weakest correlation with PM$_{2.5}$. To obtain the optimum number of input features, only RAIN, PRES and TEMP are varied. Thus, four different combinations are obtained and the values of RMSE and MAE for each combination are recorded, as shown in Table 3. Table 3 reports the feature selection process only for Node 1. The results obtained from this step can be applied to all other nodes.

As shown in Table 3, removing rain during training (11 attributes) yielded the best performance. Thus, PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_2$, and O$_3$, temperature, air pressure, dew point, wind direction and wind speed were selected as the input features for our model. We use the same input features for all monitoring sites.

To obtain the RMSE and MAE values shown in Table 3, we used a simple LSTM network as a baseline model before implementing our proposed hybrid model (see Section 4.3). A 1-layered LSTM with 15 neurons was selected as a model predictor. The lookback length of the input is determined by calculating the autocorrelation coefficient among the lagged time series of PM$_{2.5}$ data. We set 0.7 as a minimum requirement for high temporal correlation among the lagged data. As shown in Figure 5, eight samples
### Table 2. Correlation coefficients (r) among attributes at Node 1.

| Attribute | PM$_{2.5}$ | PM$_{10}$ | SO$_2$ | NO$_2$ | CO | O$_3$ | TEMP | PRES | DEWP | RAIN | WD | WSPD |
|-----------|------------|-----------|--------|--------|----|------|------|------|------|------|----|------|
| PM$_{2.5}$ | 1          | 0.87      | 0.49   | 0.67   | 0.76| -0.15| -0.09| -0.02| 0.15 | -0.01| -0.19| -0.27|
| PM$_{10}$  | 0.87       | 1         | 0.47   | 0.65   | 0.65| -0.12| -0.07| -0.05| 0.09 | -0.02| -0.12| -0.17|
| SO$_2$     | 0.49       | 0.47      | 1      | 0.44   | 0.57| -0.22| -0.36| 0.23 | -0.29| -0.04| -0.12| -0.11|
| NO$_2$     | 0.67       | 0.65      | 0.44   | 1      | 0.66| -0.46| -0.17| 0.04 | 0.12 | -0.03| -0.24| -0.48|
| CO         | 0.76       | 0.65      | 0.57   | 0.66   | 1   | -0.32| -0.37| 0.24 | 0.12 | -0.01| -0.22| -0.25|
| O$_3$      | -0.15      | -0.12     | -0.22  | -0.46  | -0.32| 1    | 0.58 | -0.42| 0.30 | 0.03 | 0.21 | 0.33|
| TEMP       | -0.09      | -0.07     | -0.36  | -0.17  | -0.37| 0.58 | 1    | -0.83| 0.83 | 0.04 | 0.05 | 0.01|
| PRES       | -0.02      | -0.05     | 0.23   | 0.04   | 0.24| -0.42| -0.83| 1    | -0.78| -0.06| -0.02| 0.09|
| DEWP       | 0.15       | 0.09      | -0.29  | 0.12   | -0.12| 0.30| 0.83 | -0.78| 1    | 0.08 | -0.13| -0.33|
| RAIN       | -0.01      | -0.02     | -0.04  | -0.03  | -0.01| 0.03 | 0.04 | 0.06 | 0.08 | 1    | -0.01| 0.00|
| WD         | -0.19      | -0.12     | -0.12  | -0.24  | -0.22| 0.21 | 0.05 | -0.02| -0.13| -0.01| 1    | 0.31|
| WSPD       | -0.27      | -0.17     | -0.11  | -0.48  | -0.25| 0.33 | 0.01 | 0.09 | -0.33| 0.00 | 0.31 | 1    |

### Table 3. Model performance based on different input attributes for Node 1.

| Input features                  | Number of inputs | RMSE  | MAE  |
|---------------------------------|------------------|-------|------|
| All                             | 12               | 17.704| 10.017|
| Without rain                    | 11               | 17.363| 9.807 |
| Without rain and pressure       | 10               | 18.168| 10.268|
| Without rain, pressure and temp | 9                | 17.638| 9.937 |

In this section, we propose a hybrid model by combining one-dimensional convolutional neural networks (1D CNN) as feature extractors, and feeding the output of these CNNs to an LSTM network, as shown in Figure 6.

The proposed model is composed of two inputs, both are formed in a parallel structure. In the first input (INPUT-1), only local (present) node data are collected, whereas in the second input (INPUT-2), all PM$_{2.5}$ data obtained from local and surrounding nodes (including time lag = 0) were selected as the length of the input model. At this time lag, all autocorrelation coefficients have values higher than 0.7 for all monitoring sites. Thus, we used the current sample (time lag = 0) and the previous 7 samples to predict 1 sample in the future.

**Figure 5.** Autocorrelation coefficients for PM$_{2.5}$ concentration with different time lags.

#### 4.3. Proposed Model

In Section 4.2, we implemented a simple, single-layered LSTM model composed of 15 neurons to evaluate model performance based on different input attributes. From this experiment, we can determine which attributes should be fed to the model. In this section, we propose a hybrid model by combining one-dimensional convolutional neural networks (1D CNN) as feature extractors, and feeding the output of these CNNs to an LSTM network, as shown in Figure 6.

The proposed model is composed of two inputs, both are formed in a parallel structure. In the first input (INPUT-1), only local (present) node data are collected, whereas in the second input (INPUT-2), all PM$_{2.5}$ data obtained from local and surrounding nodes...
are fed. Local node refers to the node where PM$_{2.5}$ is being predicted. Data for INPUT-1 are PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_x$, and O$_3$, temperature, air pressure, dew point, wind direction and wind speed (11 features in total). Eight timesteps (lookback) of these inputs are used to predict one hour of PM$_{2.5}$ in the future. Each batch of inputs is fed to the CNN network, which acts as a feature extractor before entering the LSTM network. After various experiments, we determined the properties of the CNN networks. Both CNN networks (block CNN-1 and CNN-2 in Figure 6) are composed of five convolutional layers and a single average pooling layer. The reshape layer configures the outputs produced by the CNN layers before entering the LSTM network. The same number of neurons are maintained from the previous experiment (15 neurons) with the rectified linear unit (ReLU) activation function. A dense layer with one neuron yields the final prediction. During the training process, the Adam optimiser was used. The properties of each layer are summarised in Table 4.

**Table 4.** Hybrid CNN-LSTM network properties of the proposed model. CNN-1 and CNN-2 have the same convolutional layer properties

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

As explained in Section 4.2, we use eight samples to predict one future sample. To implement deeper convolutional layers as feature extractors to a relatively short data length (in our case eight samples), we should set small kernel sizes. The length of the next convolutional layer can be calculated using Equation 2. By setting a small value of kernel size $k$, we can get higher output size $o$. Thus, a small kernel size will give more possibilities to operate another convolutional layer in the next step. In our work, we use kernel size equal to 3 for the first and second convolutional layers, and kernel size equal to 2 for remaining three convolutional layers. The selected kernel sizes and filters shown in Table 4 are obtained based on our various experiments. Choosing a smaller filter size for each layer will produce a smaller final model size. Thus, it will benefit to our edge
device. We found that the filter sizes of 50, 30, 15, 10 and 5 for each convolutional layer in our model produce the best result. We also discovered that the same properties of CNN-1 and CNN-2 yield optimum solutions as feature extractors while maintaining work balance for each input during training and inferencing stages.

4.4. Spatiotemporal Dependencies

In this study, both spatial and temporal qualities are studied. The temporal factor is taken into account by selecting time-lag data (lookbacks) as a model input, as discussed in Section 4.2. A time lag equal to zero indicates the current sample. When the time-lag is less than 8, the autocorrelation coefficient is higher than 0.7 for all nodes. This autocorrelation value indicates a high temporal correlation. Therefore, we use 8 values for the input length (current measured value plus 7 past values).

As mentioned previously in Section 4.3, in the first input of the model (INPUT-1), temporal dependency of the local node data is covered. The attributes involved for the first input are 8 timesteps of PM$_{2.5}$, PM$_{10}$, CO, NO$_2$, and O$_3$, temperature, air pressure, dew point, wind direction and wind speed. We can consider that in INPUT-1, only temporal data are covered. However, in the second input of the model (INPUT-2), both temporal and spatial data of the local and pairing nodes are included. In the second input, we collect only 8 timesteps of all PM$_{2.5}$ data (from local and surrounding nodes) and neglect all other environmental and meteorological data. All PM$_{2.5}$ samples from 12 nodes are analysed and the PM$_{2.5}$ correlation coefficients between nodes are calculated. Evaluating the correlation coefficient can indicate the effect of spatial dependency. As shown in Table 5, PM$_{2.5}$ concentrations have a strong correlation ($r > 0.7$) among nodes. A strong correlation implies that there is a high spatial dependency for PM$_{2.5}$ among nodes. Therefore, in this experiment, we include a feature extraction process for the PM$_{2.5}$ concentrations at all neighbouring nodes (data INPUT-2).

Table 5. PM$_{2.5}$ coefficient correlation for all nodes. A strong correlation implies that there is a spatial dependency for particulate matter data. Therefore, when predicting PM$_{2.5}$ data at a certain node, we should consider PM$_{2.5}$ values at other nodes.

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

Figure 7 depicts the kinds of input data required to forecast the value of PM$_{2.5}$ at a certain node. If we want to forecast the next 1-hour value of PM$_{2.5}$ concentration at Node 1, we need to use current pollutant and meteorological samples plus 7 previous samples collected by that node (the first input of the proposed model) and collect all PM$_{2.5}$ values from all other nodes (the second input of the proposed model). This scenario also applies to all other nodes.

4.5. Deep Learning Data Processing

The properties of our proposed deep learning model have been summarised in Table 4. In this section, we discuss the internal inference process in our deep learning
model. In CNN-1, the 8 timesteps of 11 input features forms an $8 \times 11$ matrix. These 11 features are composed of pollutant and meteorological data (PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_2$, and O$_3$, temperature, air pressure, dew point, wind direction and wind speed). In CNN-2, the 8 timesteps of 12 input features forms an $8 \times 12$ matrix. These 12 features consist of PM$_{2.5}$ concentrations at 12 nodes. According to equation 2, with a kernel (or feature detector) size of 3 and a stride step of 1, the kernel slides through the input matrix for 6 steps $((8 - 3)/1 + 1 = 6)$. With a filter size of 50, the first convolutional layer yields a $6 \times 50$ matrix. In the second convolutional layer, the input is now a $6 \times 50$ matrix. With a size of 3, the kernel slides along the window for 4 steps $((6 - 3)/1 + 1 = 4)$ and produces a $4 \times 30$ matrix (since the filter size is 30). The same process applies to all convolutional layers. Thus, the fifth convolutional layer yields a $1 \times 5$ matrix. A global average pooling layer behaves as a flattening process. By concatenating both CNN layer outputs, the tensor is ready to enter the LSTM network. The LSTM network consists of 15 cells (or units). Details of the data processing inside an LSTM cell have already been discussed in Section 3.2. Finally, a single dense layer produces the final result, that is our PM$_{2.5}$ prediction. Figure 8 summarises this process.
4.6. The Selected Edge Devices

Having evaluated the proposed deep learning model, we now optimise and deploy that model to edge devices. In this work, we utilised the Raspberry Pi, a popular, credit card-sized yet powerful single-board computer (SBC) developed by the Raspberry Pi Foundation. In recent years, there have been considerable varieties of applications developed using Raspberry Pi boards Sajjad et al. [46]. We have chosen two different Raspberry Pi boards: Raspberry Pi 3 Model B+ (RPi3B+) and Raspberry Pi 4 Model B (RPi4B) to show the variation in model performance. The RPi4B is more computationally capable than the RPi3B+. Table 6 shows a feature comparisons between the two boards.

Table 6. Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B feature comparisons

| Feature          | RPi3B+                               | RPi4B                               |
|------------------|--------------------------------------|-------------------------------------|
| SoC              | Broadcom BCM2837B0                    | Broadcom BCM2711                    |
| CPU              | Quad-core Cortex-A53 (ARMv8) 64-bit @1.4 GHz | Quad-core Cortex-A72 (ARMv8) 64-bit @1.5 GHz |
| GPU              | Broadcom VideoCore IV @250 MHz        | Broadcom VideoCore VI @500 MHz      |
| FPU              | VFPv4 + NEON                          | VFPv4 + NEON                        |
| RAM              | 1 GB LPDDR2 SDRAM                      | 2 GB LPDDR4 SDRAM (used in this experiment) |
| Storage          | microSD card                          | microSD card                        |
| Power ratings    | Idle: 459 mA (2.295 Watt), Maximum: 1.13 A (5.661 Watt) | Idle: 600 mA (3 Watt), Maximum: 1.25 A (6.25 Watt) |

We selected Raspberry Pis since these boards support both TensorFlow and TensorFlow Lite frameworks. Therefore, we can explore wide-range functionalities related to post-training quantisation provided by the TensorFlow, and demonstrate the performance of both original and quantised models by calculating the model accuracy, the obtained model file sizes and the execution time directly at the edge. Moreover, the Raspberry Pi’s rapid use for research and hobbyist purposes gave rise to many online forums and communities.

5. Results and Discussion

5.1. Evaluation Scenario

The evaluation process in this section can be generally described as follows. We provide 20 different deep learning models and divide these models into three groups. The performance of all models is evaluated based on the attained RMSE and MAE values at all nodes. The best model becomes our proposed model. The TensorFlow file of the proposed model will then be converted into TensorFlow Lite model. In this work, we used TensorFlow version 2.2. Further optimisation is conducted by implementing post-training quantisation of the original TensorFlow model. Finally, the performance of each TensorFlow Lite model is evaluated. The resulted model file size, the execution time, and the prediction performance of each TensorFlow Lite model are reported. Figure 9 illustrates this process.

Figure 9. The Evaluation scenario diagram conducted in this work.

5.2. Model Performance

Based on pollutant and meteorological data from the current and the previous 7 hours, we predict the short-term PM$_{2.5}$ concentration for 1 hour in the future. Model
performance was measured based on the obtained RMSE and MAE values evaluated at all nodes. The values of RMSE and MAE were calculated using equation 9 and Equation 10, respectively. Table 7 summarises the obtained RMSE and MAE of all models, with Node 1 as a representative. The complete result of all nodes is presented in Table A1 and Table A2. We compared our proposed model against several deep learning architectures and proved that our proposed model outperforms other models. Model comparison in Table 7 can be explained as follows:

- Simple models with local data only (Group I) take input samples directly without passing them through CNN layers. In these models, the convolutional, pooling, concatenation and reshaping layers are omitted. Thus, the inputs are directly supplied to the RNN, LSTM, GRU or Bidirectional layers. The kinds of inputs used in this architecture are PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_2$, and O$_3$, and meteorological data such as temperature, air pressure, dew point, wind direction and wind speed.

- Hybrid models with local data only (Group II) filter input samples through the CNN layers before entering the ANN, RNN, LSTM, GRU or Bidirectional layers. These architectures are hybrid models. In this group, only INPUT-1 and CNN-1 layers are considered, whereas INPUT-2 and CNN-2 layers are neglected. The properties of the CNN layers have been described in Table 4. In this case, only PM$_{2.5}$, PM$_{10}$, SO$_2$, CO, NO$_2$, and O$_3$, and meteorological data such as temperature, air pressure, dew point, wind direction and wind speed are used as the model inputs, without considering neighbouring PM$_{2.5}$ samples.

- Hybrid models with spatiotemporal dependency (Group III) use two inputs (INPUT-1 and INPUT-2), and each input is filtered by CNN layers (CNN-1 and CNN-2). The first input covers the pollutant and meteorological data for the node under consideration, while the second input comprises PM$_{2.5}$ samples from neighbouring nodes. Models in Group III comply with Figure 6, but we vary the LSTM layer with ANN, RNN, GRU or Bidirectional layers.

- Artificial neural network (ANN), recurrent neural network (RNN), long short-term memory (LSTM) and gated recurrent unit (GRU) models in all groups were evaluated and their performance were compared. For fairness, all models in all groups are composed of one hidden layer with 15 neurons (units) inside the layer. At the output layer, there is a dense layer with one neuron to produce the final prediction.

- Bidirectional layers are an extension of conventional RNN, LSTM and GRU with two different input directions. First, the input sequence is treated in the usual direction. Second, the input sequence is handled in reverse direction. This scenario can offer additional context to the model and may result in faster and even deeper learning on the input sequence.

As shown in Table 7, we compared 20 different models. We can see that by adding a deeper model (CNN layers) as feature extractor before the predictor (ANN, RNN, LSTM or GRU) will slightly improve models performances. Generally, Group II has better performance than Group I. Adding spatiotemporal considerations along with pollutant and meteorological data as inputs of the model can increase the accuracy. At some nodes, the results can be improved significantly. For example, at Node 1, Group I and Group II produce RMSE values between 17 and 19, whereas Group III produces RMSE values between 15 and 17. The best RMSE value was obtained by our proposed model (model no 16 in Table 7), which is 15.322. This RMSE value is better than all other investigated models. For instance, the Bidirectional RNN model in Group I yielded an RMSE value of 19.377, the CNN-LSTM model in Group II produced 17.652, and the CNN-ANN model in Group III returned the RMSE value of 17.160. If we continue to look in more detail to other nodes in Table A1 and Table A2, the PM$_{2.5}$ concentration at Node 11 can be better forecast, not only by our proposed model but also by other investigated models. In contrast, the PM$_{2.5}$ concentration at Node 12 was the hardest to predict as indicated by the higher RMSE and MAE values. For all nodes, our proposed model produced
Table 7. Comparison of RMSE and MAE values for PM$_{2.5}$ prediction using different model architectures calculated for Node 1. Twelve different model architectures are proposed and categorised into three groups. The best model for each group is indicated in bold. The proposed model in this work belongs to Group III (model number 16). In Group III, both spatial and temporal dependencies are considered. Our proposed model yields the lowest RMSE and MAE values. Detail results for all nodes are shown in Table A1 and Table A2.

| No. | Model Type                     | RMSE  | MAE  |
|-----|--------------------------------|-------|------|
| 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|

|          |                              |       |      |
|----------|------------------------------|-------|------|
| Simple models with local data only (Group I) |       |      |
| 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 local data only (Group II) |       |      |
| Hybrid models with spatiotemporal dependency (Group III) |       |      |
| CNN-ANN | 17.160 | 10.307 |
| CNN-RNN | 15.672 | 9.162  |
| CNN-LSTM | **15.268** | **8.778** |
| CNN-GRU | 17.169 | 9.162  |
| CNN-Bidirectional RNN | 17.365 | 10.443 |
| CNN-Bidirectional LSTM | 15.643 | 8.853  |
| CNN-Bidirectional GRU | 16.089 | 9.512  |

Figure 10 shows the boxplot of prediction deviation of all model. The prediction deviation is obtained by subtracting the real value of data test from the predicted values of the model. From the boxplot, we can find information about the variability of the data. The box plot is also useful when we want to compare the distribution between many models. In Figure 10, the solid line in the middle of each box represents the median value. Since the graph represents the prediction deviation between predicted and real data, we prefer this line close to zero. The shorter box and whisker, the more centralised the data is. The more centralised data means the more accurate our model in predicting the PM$_{2.5}$ data. We also removed the outliers values in order to make the graph more readable. As shown in Figure 10, our proposed model gives the best result as it produces more centralised data and the median value closest to zero.

In order to describe model performance more intuitively, Figure 11 shows a line plot between the real and predicted values on the test data at Node 1. The solid line and dashed line indicate the real and predicted values, respectively. There are a total of 8,280 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 11. The larger errors usually happen when there are spikes in the actual data, whereas for smoother PM$_{2.5}$ data variations, our model forecasts successfully.
Figure 10. The boxplot of the prediction deviation of all investigated models at Node 1.

Figure 12 shows scatter plots obtained from all nodes, showing the relationship between real and predicted values; the perfect predictions coincide with the diagonal solid line. Due to the presence of prediction errors the points diverge from this diagonal. All dots mapped below the diagonal solid (ideal) line indicate predictions that are lower than the correct values, while the opposite occurs for points above the ideal line. For example, in the case of Node 3, we observe that more deviations occur below the ideal line. The model predicted 103.92 µg/m³, whereas the actual value is 414 µg/m³. The same case occurs for Node 7. The model predicted 162.36 µg/m³, whereas the real sample is 556 µg/m³. Some mispredictions may be due to measurement error, which can be recognised from sudden changes in a sequence of measured samples that are not technically feasible. As shown for Node 12 in Figure 12, there is a significant error in predicting PM$_{2.5}$ data. The model predicted 554.24 µg/m³ whereas the measured sensor value is only 3 µg/m³ for the labelled point. If we check the dataset at Node 12 in more detail, there had been a sharp drop in the measured value from 621 µg/m³ to only 3 µg/m³. From 3 µg/m³, the measured value then jumps sharply to 144 µg/m³. The LSTM network could not recognise these changes. Therefore, there is a significant prediction error at this point.

Figure 11. Line plot of real and predicted PM$_{2.5}$ data at Node 1.
5.3. Model Optimisation for the Edge

After the final model has been trained, the next process is to deploy that model to the edge after optimising it. This optimisation benefits filesize and computation latency. The initially created model is the TensorFlow (TF model). From the TF model, we convert our model to a TensorFlow Lite (TFLite model), a lightweight model suitable for edge devices. This TF model can be deployed with or without optimisation, as explained in Section 4.6. We evaluated all possibilities, both with and without optimisation applied. Table 8 summarises the file size comparison between the TF Model and TFLite model. In this case, TFLite has not yet been optimised. The original file size is 318 kilobytes whereas the lite version is 77 kilobytes or 4 times smaller. File size reduction is an essential step for resource-constrained edges, especially devices with minimal storage available.

Table 8. TensorFlow and TensorFlow Lite file size comparison.

| Properties | TF model | TFLite model |
|------------|----------|--------------|
| File size (kB) | 318 | 77 |

Further size reduction can be achieved by implementing post-training quantisation. As shown in Figure 13, four different optimisation techniques available in the TensorFlow framework have been evaluated for our proposed deep learning model. These techniques are dynamic range quantisation, full integer quantisation with float fallback, integer only quantisation, and float16 quantisation. As shown in Table 8 and Figure 13, TFLite model without optimisation/quantisation has a size of 77 kilobytes. Using this model as a reference, about a 47% reduction can be achieved by dynamic range quantisation, about 45% by full integer quantisation, and about 35% by float16 quantisation. Based on these results, dynamic range quantisation outperforms other techniques, even though it is just slightly better than full-integer quantisation.
The time needed for edge devices to predict the available test data was measured. In this study, a total of 8272 hourly samples (data from 21 March 2016 to 28 February 2017) were continuously executed directly at the edge. The experiment results are summarised in Figure 14. As depicted in the figure, the RPi4B board is two times faster than the RPi3B+ board in all quantisation modes. The model with Float16 quantisation does not improve execution time as the latency remains the same with or without quantisation, likely due to the fixed 32-bit floating point datapath on these devices. In this case, the RPi3B+ board needs 8.49 seconds to execute the complete test whereas the RPi4B board produces a 0.07 second difference (3.75 seconds and 3.82 seconds). Even though about 47% size reduction can be achieved by dynamic range quantisation, this mode has minimal execution time improvement. The execution time for this mode is 7.03 seconds and 3.14 seconds for RPi3B+ and RPi4B, respectively. Full integer quantisation produced the most effective execution time improvement, with latencies of 4.73 seconds and 2.19 seconds for RPi3B+ and RPi4B, respectively.
Besides model size and execution time, we must also evaluate model accuracy after applying quantisation. Table A3 in Appendix A shows the details of the RMSE and MAE values for the initial TensorFlow and TensorFlow Lite models. Since the result deviation between the optimized models is very small, we provide a boxplot to present the model performance more intuitively, as shown in Figure 15. This figure provides information about prediction deviation between the result obtained by TFModel and TFLite Model. It is clearly observed that TFLite without quantisation and TFLite with float16 quantisation accuracies are very similar (or produced a very small deviation) to the original TFModel. Slightly a longer deviation range is given by TFLite with dynamic range quantisation. Both TFLite integer quantisations give the longest box and whisker range, indicating that these quantisations inferior to other post-quantisation methods in terms of prediction accuracy.

![Figure 15. Boxplot of prediction deviation resulted from each TFLite model. The deviation is calculated by substracting the real values from the predicted values of each TFLite model.](image)

If we are primarily concerned with model accuracy, TFLite without quantisation is a suitable technique. However, it is not the best choice for size reduction and execution time improvement. Dynamic range and float16 quantisations also maintain model accuracy. Dynamic range quantisation produced better model size reduction and execution time than float16 quantisation. Full integer quantisations outperformed other TFLite models in terms of model size and latency but slightly reduced model accuracy.

To examine the correspondence between our TensorFlow Lite models and the initial TensorFlow model intuitively, we can compare these models using scatter plots, as shown in Figure 16. This figure presents the result at only Node 1. However, the same behaviour occurred for all nodes. The results obtained by the TFLite without quantisation, dynamic range quantisation, and float16 quantisation match the results predicted by the initial TensorFlow model, as demonstrated by the a smooth straight-line pattern. We have can also observed the same effect from Figure 15. A larger deviation is produced by full integer quantisation models, both integer with fallback and full integer quantisations. The straight-line pattern is more scattered and lead to a conclusion that that full integer quantisation impacts model accuracy, even with a very small deviation.

6. Conclusions

Edge computing brings computation closer to data sources (edge) and can be a solution for latency, privacy and scalability issues faced by a cloud-based system. It
is also possible to embed intelligence at the edge, which can be enabled by Machine Learning algorithms. Deep Learning, a subset of ML, can be implemented at the edge. In this work, we proposed a hybrid deep learning model composed of 1D Convolutional Neural Network and Long-short Term Memory (CNN-LSTM) networks to predict a short-term hourly PM$_{2.5}$ concentration at 12 different nodes. The results show that our proposed model outperformed other possible deep learning models, evaluated by calculating RMSE and MAE errors at each node. In order to implement an efficient model for edge devices, we applied four different post-quantisation techniques provided by TensorFlow Lite framework: dynamic range quantisation, float16 quantisation, integer with float fallback quantisation, and full integer-only quantisation. Dynamic range and float16 quantisations maintain model accuracy but do not improve latency significantly. Meanwhile, full integer quantisation outperforms other TFLite models in terms of model size and latency but slightly reduces model accuracy. The targeted edge devices in our work are the Raspberry Pi 3 Model B+ and Raspberry Pi 4 Model B boards. Technically, the Raspberry Pi 4 demonstrated lower latency due to the more capable processor.

In future, we plan to develop this work further by offloading model computation for multiple nodes to a gateway device, thereby allowing the sensor nodes to be extremely lightweight. We would also like to explore methods for efficient sharing of a gateway deep learning model by multiple nodes. Finally, we would like to explore how models can be evolved on these edge devices.

Author Contributions: Conceptualization, K.W. and S.F.; methodology, K.W., J.G. and S.F.; software, K.W.; data collection, K.W.; data analysis K.W. and S.F.; writing—original draft preparation, K.W.; writing—review and editing, K.W., J.G. and S.F.; supervision, J.G. and S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Indonesia Endowment Fund for Education (LPDP), Ministry of Finance, Republic of Indonesia under grant number Ref: S-1027/LPDP.4/2019.

Acknowledgments: INK Wardana was supported with a studentship from Indonesia Endowment Fund for Education (LPDP).

Conflicts of Interest: The authors declare no conflict of interest.
Abbreviations

The following abbreviations are used in this manuscript:

1D CNN One-Dimensional Convolutional Neural Network
ANN Artificial Neural Network
DL Deep Learning
GRU Gated Recurrent Units
IoT Internet of Things
LSTM Long Short-Term Memory
MAE Mean Absolute Error
ML Machine Learning
PM Particulate Matter
RMSE Root Mean Square Error
RNN Recurrent Neural Network
RPi3B+ Raspberry Pi Model 3B+
RPi4B Raspberry Pi Model 4B

Appendix A

In this section, we report the RMSE and MAE of the predicted values of all nodes. Table A1 and Table A2 show the RMSE and MAE values, respectively. Table A3 shows the effect of quantisation techniques on the RMSE and MAE values.

Table A1. 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  |
|-----|--------------------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 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  |
| 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  | 16.766  | 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  |

| No. | Architectures            | Node1   | Node2   | Node3   | Node4   | Node5   | Node6   | Node7   | Node8   | Node9   | Node10  | Node11  | Node12  |
|-----|--------------------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 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  | 16.618  | 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  | 16.345  | 18.134  | 15.874  | 15.785  | 19.267  |
### Table A2. 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 |
|-----|------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1   | RNN                    | 10.636 | 10.123 | 10.206 | 10.403 | 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 | 10.513 | 11.183 | 11.947 |
| 4   | Bidirectional RNN      | 12.257 | 9.898  | 9.349  | 11.871 | 10.746 | 9.040  | 10.571 | 11.154 | 10.587 | 11.144 | 11.560 |
| 5   | Bidirectional LSTM     | 10.427 | 9.622  | 10.335 | 10.710 | 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 | 11.635 | 11.57  | 8.338  | 10.171 | 10.124 | 10.404 | 9.658  | 11.348 |

### Table A3. Effect of quantisation techniques on RMSE and MAE values.

| No. | Architectures                      | Node1  | Node2  | Node3  | Node4  | Node5  | Node6  | Node7  | Node8  | Node9  | Node10 | Node11 | Node12 |
|-----|------------------------------------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 1   | Initial TensorFlow 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 |
|     | RMSE                               | 8.778  | 8.873  | 8.803  | 9.653  | 9.228  | 9.590  | 7.607  | 9.116  | 9.588  | 8.935  | 8.486  | 9.641  |
|     | MAE                                 | 9.162  | 9.602  | 8.998  | 10.106 | 9.510  | 10.069 | 8.326  | 11.068 | 12.817 | 10.055 | 9.543  | 10.295 |
| 2   | TFLite without quantisation         | 15.268 | 15.710 | 17.082 | 17.706 | 16.557 | 17.743 | 15.493 | 16.172 | 17.920 | 14.894 | 14.951 | 18.962 |
|     | RMSE                               | 8.778  | 8.873  | 8.803  | 9.653  | 9.228  | 9.590  | 7.607  | 9.116  | 9.588  | 8.935  | 8.486  | 9.641  |
|     | MAE                                 | 9.162  | 9.602  | 8.998  | 10.106 | 9.510  | 10.069 | 8.326  | 11.068 | 12.817 | 10.055 | 9.543  | 10.295 |
| 3   | TFLite - dynamic range quantisation | 15.270 | 15.710 | 17.116 | 17.679 | 16.580 | 17.741 | 15.490 | 16.189 | 17.906 | 14.905 | 14.994 | 18.974 |
|     | RMSE                               | 8.784  | 8.872  | 8.844  | 9.639  | 9.236  | 9.592  | 7.611  | 9.129  | 9.604  | 8.943  | 8.550  | 9.678  |
|     | MAE                                 | 9.418  | 9.420  | 9.562  | 10.425 | 10.004 | 10.394 | 8.081  | 9.674  | 10.792 | 9.048  | 10.048 | 10.785 |
| 4   | TFLite - integer with float fallback| 15.704 | 16.102 | 17.506 | 18.108 | 17.032 | 18.167 | 15.694 | 16.536 | 18.758 | 15.467 | 15.216 | 19.690 |
|     | RMSE                               | 9.418  | 9.420  | 9.562  | 10.425 | 10.004 | 10.394 | 8.081  | 9.674  | 10.792 | 9.048  | 10.048 | 10.785 |
|     | MAE                                 | 15.268 | 15.708 | 17.082 | 17.707 | 16.557 | 17.742 | 15.494 | 16.171 | 17.920 | 14.896 | 14.952 | 18.962 |
| 5   | TFLite - integer-only quantisation  | 8.777  | 8.871  | 8.803  | 9.654  | 9.228  | 9.589  | 7.607  | 9.115  | 9.588  | 8.936  | 8.486  | 9.642  |

**References**

1. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* 2018, 6, 6900–6919. doi:10.1109/access.2017.2778504.
2. Ngu, A.H.H.; Gutierrez, M.; Metsis, V.; Nepal, S.; Sheng, M.Z. IoT Middleware: A Survey on Issues and Enabling technologies. *IEEE Internet of Things Journal* 2016, pp. 1–1. doi:10.1109/jiot.2016.2615180.
3. Frustaci, M.; Pace, P.; Aloï, G.; Fortino, G. Evaluating Critical Security Issues of the IoT World: Present and Future Challenges. *IEEE Internet of Things Journal* 2018, 5, 2483–2495. doi:10.1109/jiot.2017.2767291.
4. Chen, J.; Ran, X. Deep Learning With Edge Computing: A Review. *Proceedings of the Institute of Electrical and Electronics Engineers* **IEEE 2019**, *107*, 1655–1674. doi:10.1109/jproc.2019.2921977.

5. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* **2016**, *3*, 637–646. doi:10.1109/jiot.2016.2579198.

6. Samie, F.; Bauer, L.; Henkel, J. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet of Things Journal* **2019**, *6*, 4921–4934. doi:10.1109/jiot.2019.2893866.

7. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* **2020**, *22*, 869–904. doi:10.1109/comst.2020.2970550.

8. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. doi:10.1038/nature14539.

9. Verhelst, M.; Moons, B. Embedded Deep Neural Network Processing: Algorithmic and Processor Techniques Bring Deep Learning to IoT and Edge Devices. *IEEE Solid-State Circuits Magazine* **2017**, *9*, 55–65. doi:10.1109/msc.2017.2745818.

10. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.

11. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network, 2015, [arXiv:stat.ML/1503.02531](https://arxiv.org/pdf/1503.02531.pdf).

12. Véstias, M.P.; Duarte, R.P.; de Sousa, J.T.; Neto, H.C. A fast and scalable architecture to run convolutional neural networks in low density FPGAs. *Microprocessors and Microsystems* **2020**, *77*, 103136. doi:10.1016/j.micpro.2020.103136.

13. Yang, T.; Liu, W. Does air pollution affect public health and health inequality? Empirical evidence from China. *Journal of Cleaner Economics* **2018**, *17*, 43–52. doi:10.1016/j.jcleocem.2018.08.242.

14. Sun, C.; Kahn, M.E.; Zheng, S. Self-protection investment exacerbates air pollution exposure inequality in urban China. *Economic and Political Studies* **2017**, *131*, 468–474. doi:10.1007/s40813-016-0030-2.

15. Ma, J.; Li, Z.; Cheng, J.C.; Ding, Y.; Lin, C.; Xu, Z. Air quality prediction at new stations using spatially transferred bi-directional long short-term memory network. *Science of The Total Environment* **2020**, *705*, 135771. doi:10.1016/j.scitotenv.2019.135771.

16. Liu, H.; Wu, H.; Lv, X.; Ren, Z.; Liu, M.; Li, Y.; Shi, H. An intelligent hybrid model for air pollutant concentrations forecasting: Case of Beijing in China. *Sustainable Cities and Society* **2019**, *47*, 101471. doi:10.1016/j.scs.2019.101471.

17. Zhang, Y.; Li, Z. Remote sensing of atmospheric fine particulate matter (PM2.5) mass concentration near the ground from satellite observation. *Remote Sensing of Environment* **2015**, *160*, 252–262. doi:10.1016/j.rse.2015.02.005.

18. Hamra, G.B.; Guha, N.; Cohen, A.; Laden, F.; Raaschou-Nielsen, O.; Samet, J.M.; Vineis, P.; Forastiere, F.; Saldíva, P.; Yorifuji, T.; Loomis, D. Outdoor Particulate Matter Exposure and Lung Cancer: A Systematic Review and Meta-Analysis. *Environmental Health Perspectives* **2014**, doi:10.1289/ehp.1408092.

19. Dominiic, F.; Peng, R.D.; Bell, M.L.; Pham, L.; McDermott, A.; Zeger, S.L.; Samet, J.M. Fine Particulate Air Pollution and Hospital Admission for Cardiovascular and Respiratory Diseases. *JAMA* **2006**, *295*, 1127. doi:10.1001/jama.295.10.1127.

20. Langrish, J.P.; Li, X.; Wang, S.; Lee, M.M.; Barnes, G.D.; Miller, M.R.; Cassee, F.R.; Boon, N.A.; Donaldson, K.; Li, J.; Li, L.; Mills, N.L.; Newby, D.E.; Jiang, L. Reducing Personal Exposure to Particulate Air Pollution Improves Cardiovascular Health in Patients with Coronary Heart Disease. *Environmental Health Perspectives* **2012**, *120*, 367–372. doi:10.1289/ehp.1103898.

21. Islam, N.; Saikia, B.K. Atmospheric particulate matter and potentially hazardous compounds around residential/road side soil in an urban area. *Chemosphere* **2020**, *259*, 124753. doi:10.1016/j.chemosphere.2020.124753.

22. Daiber, A.; Kuntic, M.; Hahad, O.; Delogu, L.G.; Rohrbach, S.; Lisa, F.D.; Schulz, R.; Münzel, T. Effects of air pollution particles (ultrafine and fine particulate matter) on mitochondrial function and oxidative stress – Implications for cardiovascular and neurodegenerative diseases. *Archives of Biochemistry and Biophysics* **2020**, *696*, 108662. doi:10.1016/j.abb.2020.108662.

23. Castelli, M.; Clemente, F.M.; Popović, A.; Silva, S.; Vanneschi, L. A Machine Learning Approach to Predict Air Quality in California. *Complexity* **2020**, 2020, 1–23. doi:10.1155/2020/8049504.

24. Navares, R.; Aznarte, J.L. Predicting air quality with deep learning LSTM: Towards comprehensive models. *Ecological Informatics* **2020**, *55*, 101019. doi:10.1016/j.ecoinf.2019.101019.

25. Li, X.; Peng, L.; Yao, X.; Cui, S.; Hu, Y.; You, C.; Chi, T. Long short-term memory neural network for air pollutant concentration predictions: Method development and evaluation. *Environmental Pollution* **2017**, *231*, 997–1004. doi:10.1016/j.envpol.2017.08.114.

26. Xayasouk, T.; Lee, H.; Lee, G. Air Pollution Prediction Using Long Short-Term Memory (LSTM) and Deep Autoencoder (DAE) Models. *Sustainability* **2020**, *12*, 2570. doi:10.3390/su12062570.

27. Seng, D.; Zhang, Q.; Zhang, X.; Chen, G.; Chen, X. Spatiotemporal prediction of air quality based on LSTM neural network. *Alexandria Engineering Journal* **2021**, *60*, 2021–2032. doi:10.1016/j.aej.2020.12.009.

28. Xu, J.; Chen, L.; Lv, M.; Zhan, C.; Chen, S.; Chang, J. HighAir: A Hierarchical Graph Neural Network-Based Air Quality Forecasting Model. *arXiv:2101.04264*.

29. Ma, J.; Ding, Y.; Cheng, J.C.; Jiang, F.; Wan, Z. A temporal-spatial interpolation and extrapolation method based on geographic Long Short-Term Memory neural network for PM2.5. *Journal of Cleaner Production* **2019**, *237*, 117729. doi:10.1016/j.jclepro.2019.117729.

30. Bai, Y.; Zeng, B.; Li, C.; Zhang, J. An ensemble long short-term memory neural network for hourly PM2.5 concentration forecasting. *Chemosphere* **2019**, *222*, 286–294. doi:10.1016/j.chemosphere.2019.01.121.

31. Zhao, J.; Deng, F.; Cai, Y.; Chen, J. Long short-term memory - Fully connected (LSTM-FC) neural network for PM2.5 concentration prediction. *Chemosphere* **2019**, *220*, 486–492. doi:10.1016/j.chemosphere.2018.12.128.

32. Qi, Y.; Li, Q.; Karimian, H.; Liu, D. A hybrid model for spatiotemporal forecasting of PM2.5 based on graph convolutional neural network and long short-term memory. *Science of The Total Environment* **2019**, *664*, 1–10. doi:10.1016/j.scitotenv.2019.01.333.
33. Zhang, Q.; Lam, J.C.; Li, V.O.; Han, Y. Deep-AIR: A Hybrid CNN-LSTM Framework for Fine-Grained Air Pollution Forecast, 2020, [arXiv:cs.SP/2001.11957].

34. Qin, D.; Yu, J.; Zou, G.; Yong, R.; Zhao, Q.; Zhang, B. A Novel Combined Prediction Scheme Based on CNN and LSTM for Urban PM2.5 Concentration. *IEEE Access* 2019, 7, 20050–20059. doi:10.1109/access.2019.2897028.

35. Li, T.; Hua, M.; Wu, X. A Hybrid CNN-LSTM Model for Forecasting Particulate Matter (PM2.5). *IEEE Access* 2020, 8, 26933–26940. doi:10.1109/access.2020.2971348.

36. Tao, Q.; Liu, F.; Li, Y.; Sidorov, D. Air Pollution Forecasting Using a Deep Learning Model Based on 1D Convnets and Bidirectional GRU. *IEEE Access* 2019, 7, 76690–76698. doi:10.1109/access.2019.2921578.

37. Chen, H.C.; Putra, K.T.; Chun-WeiLin, J. A Novel Prediction Approach for Exploring PM2.5 Spatiotemporal Propagation Based on Convolutional Recursive Neural Networks 2021. [arXiv:2101.06213].

38. Banner, R.; Nahshan, Y.; Soudry, D. Post training 4-bit quantization of convolutional networks for rapid-deployment. Advances in Neural Information Processing Systems. Curran Associates, Inc., 2019, Vol. 32, pp. 7950–7958.

39. Wu, H.; Judd, P.; Zhang, X.; Isaev, M.; Micikevicius, P. Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation 2020. [arXiv:2004.09602].

40. Peng, P.; You, M.; Xu, W.; Li, J. Fully integer-based quantization for mobile convolutional neural network inference. *Neurocomputing* 2021, 432, 194–205. doi:10.1016/j.neucom.2020.12.035.

41. Li, J.; Alvarez, R. On the quantization of recurrent neural networks 2021. [arXiv:2101.05453].

42. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mane, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viegas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2016, [arXivcs.DC/1603.04467].

43. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Computation* 1997, 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.

44. Post-training quantization. Available online: https://www.tensorflow.org/lite/performance/post_training_quantization. (accessed on 22-Jan-2021).

45. Zhang, S.; Guo, B.; Dong, A.; He, J.; Xu, Z.; Chen, S.X. Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2017, 473, 20170457. doi:10.1098/rspa.2017.0457.

46. Sajjad, M.; Nasir, M.; Muhammad, K.; Khan, S.; Jan, Z.; Sangahia, A.K.; Elhoseny, M.; Baik, S.W. Raspberry Pi assisted face recognition framework for enhanced law-enforcement services in smart cities. *Future Generation Computer Systems* 2017, 108, 995–1007. doi:10.1016/j.future.2017.11.013.