5. EXPERIMENTS AND RESULT ANALYSIS

This chapter discusses the experimentation and the result analysis in two sub-sections. The first sub-section starts with the data analysis and visualization details of the proposed air quality monitoring system using IoT. Next, the sub-section discusses the details of the performance of the power consumption optimization scheme and the event-based transmission scheme. In the end, it discusses the quality of service and system performance under periodic transmission. The second sub-section starts with discussing the performance of the air quality parameter prediction using the proposed approach. Next, the performance under the employed regularization techniques and attention mechanism with various hyperparameter settings are discussed in the same sub-section.

5.1 RESULTS AND DISCUSSION OF PROPOSED AIR QUALITY MONITORING SYSTEM

Figure 5.1(a) represents the prototype design of the sensing smart node of the proposed system. Figures 5.1(b) and 5.1(c) depict the deployment of the sensing node at indoor(home) and outdoor(rooftop), respectively.

(a)

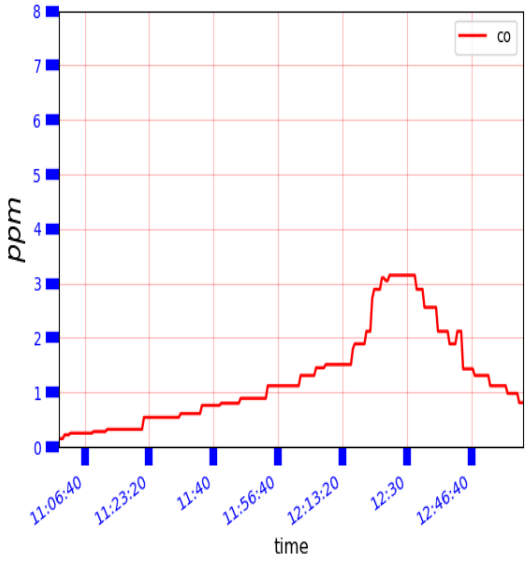
(b) (c)

Figure. 5.1. (a) prototype (b) deployment at home(indoor) (c) deployment rooftop(outdoor)

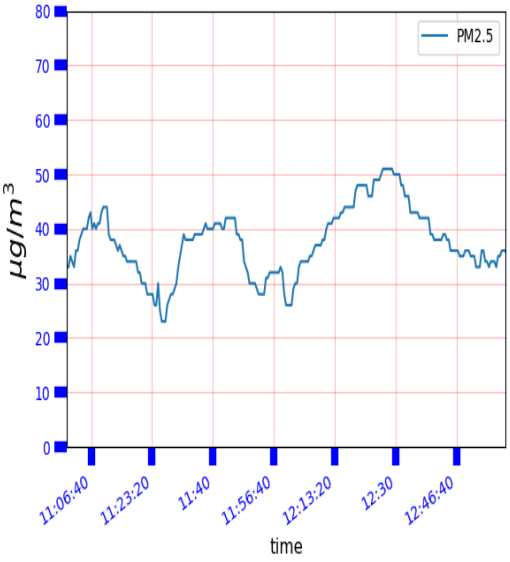
5.1.1 Data Analysis and Visualization

We have conducted the deployment and testing of the proposed air quality monitoring system at two sites (indoor and outdoor). The experiments for the indoor site were conducted within the home environment which we will refer to as site 2, and outdoor experiments were conducted at the rooftop of the building, to which we will refer as site 1.

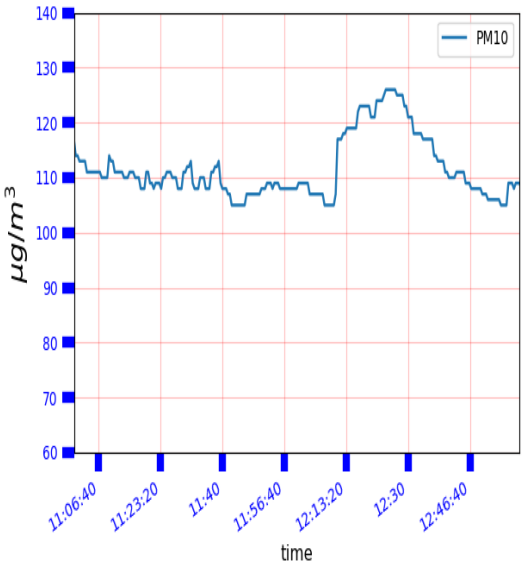
We utilized the HiveMQ broker during the experiments, and the MQTT publisher was set to work at QoS 0 and QoS 1 levels. The publisher transmits the observed data to the broker from the two sites. Figure 5.2 represents the sample real-time graph generated using python script for a visual appearance from the data collected and logged at the MQTT subscriber at the server-side. These real-time graphs of data collected at the server are implemented using the “matplotlib” library using python script. The obtained data of CO, PM2.5, and PM10 from the remote side at a particular instance is shown in the figure. The observation data received at the periodic interval at the server are displayed in the graph by updating the graph's data at every one-hour sliding window in the matplotlib library. Parallel to the rendering of the real-time graph of retrieved parameters of air pollutants available at broker from the deployment sites, MQTT subscribers also log the data in the MYSQL database, which can be used for further analysis.


co
ppm
time
11:06:40
11:23:20
11:40
11:56:40
12:13:20
12:30
12:46:40

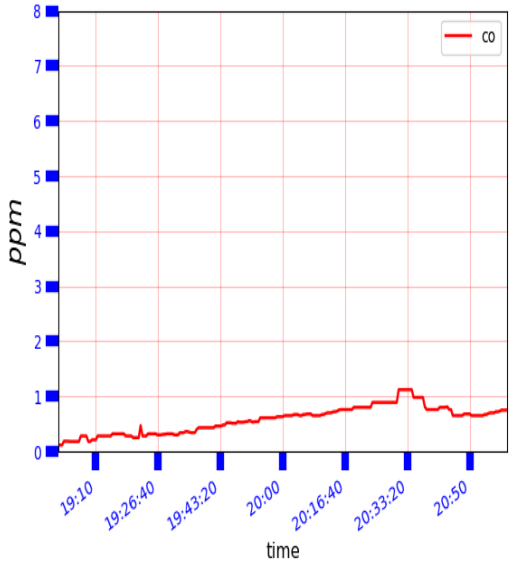
(a)


PM2.5
µg/m³
time
11:06:40
11:23:20
11:40
11:56:40
12:13:20
12:30
12:46:40

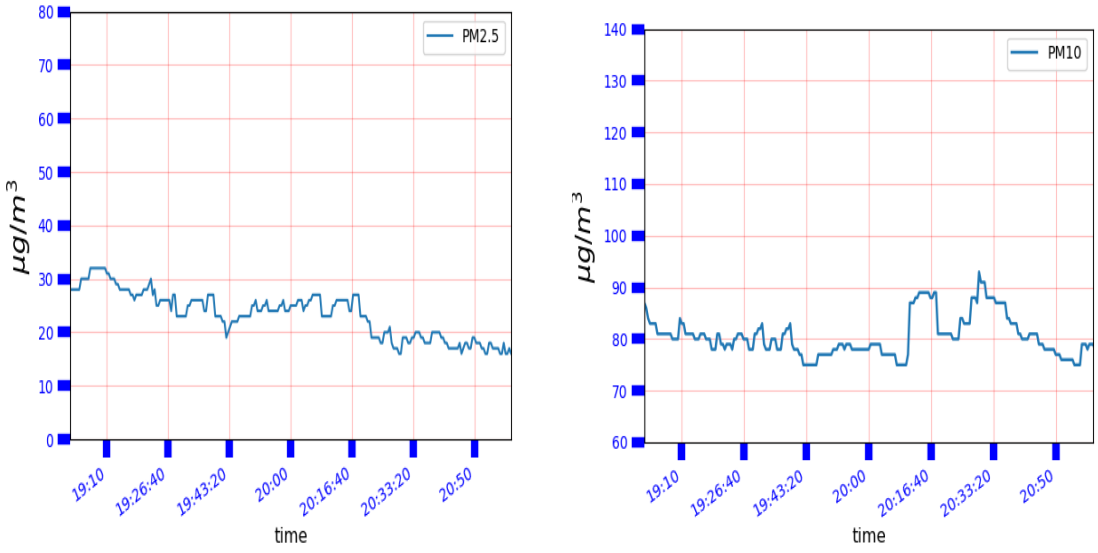
(b)


PM10
µg/m³
time
11:06:40
11:23:20
11:40
11:56:40
12:13:20
12:30
12:46:40

(c)


co
ppm
time
19:10
19:26:40
19:43:20
20:00
20:16:40
20:33:20
20:50

(d)



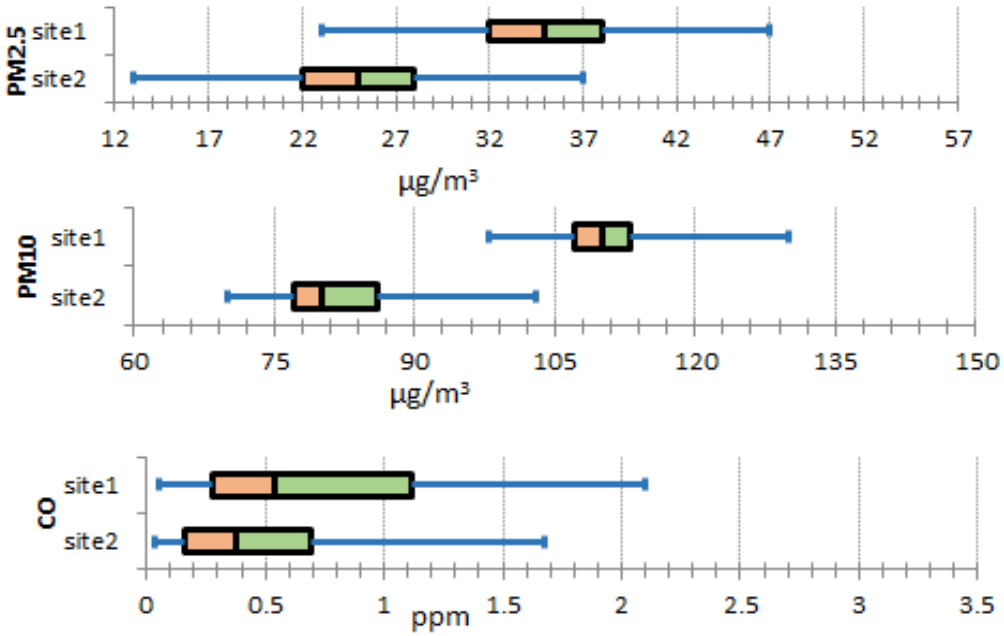
(e) (f)

Figure 5.2. Some snaps of GUI cum graphs generated from the data at server for monitoring the parameters of home rooftop (site 1: outdoor) and home (site 2: indoor) per one-hour sliding window (a) CO (at rooftop) (b) PM 2.5 (at rooftop) (c) PM 10 (at rooftop) (d) CO (at home) (e) PM 2.5 (at home) (f) PM (at home)

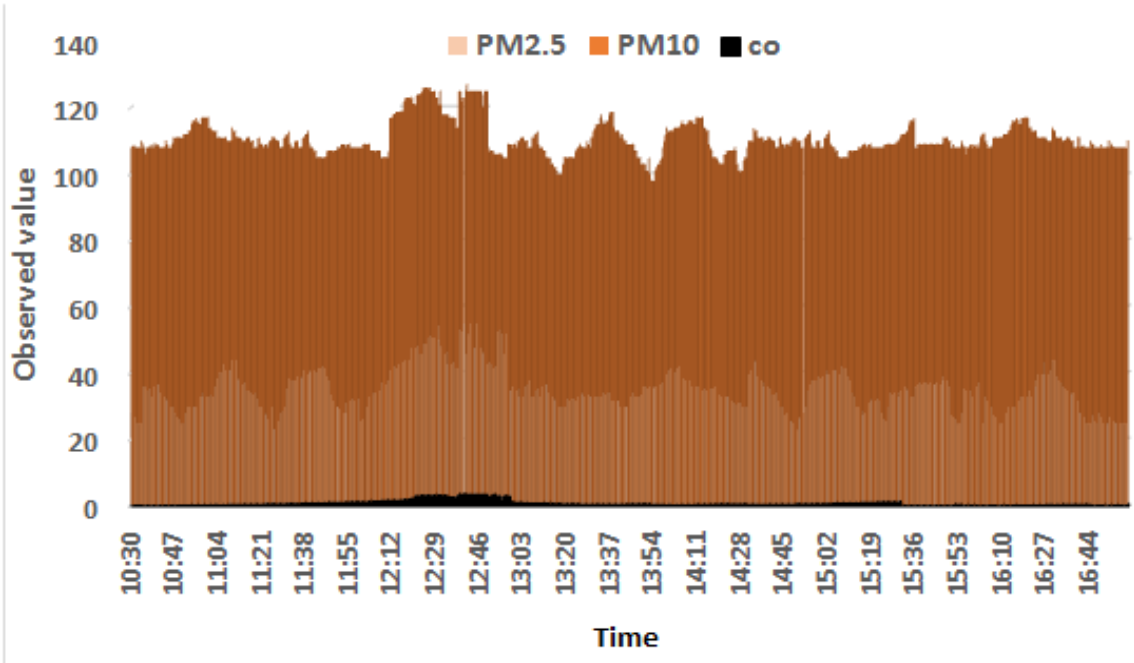
Figure 5.3(a) represents the distribution of the pollutants recorded at the server and compares the daytime distribution for indoor and outdoor sites. The readings displayed are analysed for 6 hours' duration at the specific instance for the two sites. It can be seen that the PM2.5 parameter values vary from the minimum value of 23 to the maximum value of 51 μg /$m^{3,}$ The median of the PM2.5 observed is 35 at site1. The PM10 is observed to be in the range 98 to 126 μg /$m^3$and the median is 110 at site 1. At site 1, the maximum value of carbon monoxide observed r) is 3.15 ppm. The temperature varies from the minimum value of 32.5◦C to the maximum value of 34.8◦C during the 6 hours' interval at site 1. The relative humidity is found to be fluctuating in the range of 24.1 % to 31.1 %.

For the observed air pollutants at site 2, carbon monoxide, Particulate matter10, and Particulate matter2.5, the median values are recorded as 0.38 ppm, 80 μg/$m^{3,}$ and 24μg /$m^{3,}$ respectively. The highest value of PM10 is recorded 102 μg/m^3 for site 2 during this time interval, which is 23% lower than the maximum value of PM10 observed at site1. The minimum value recorded for PM10 is 69 μg /m^3. Figure 5.3 (b) and Figure 5.3 (c) represent the recorded data of pollutants Particulate matter 10, 2.5, and carbon monoxide during daytime for outdoor deployment and indoor deployment. The figure represents the differences in the air pollutant levels at both sites. From the figure, it can be observed that the second site (indoor

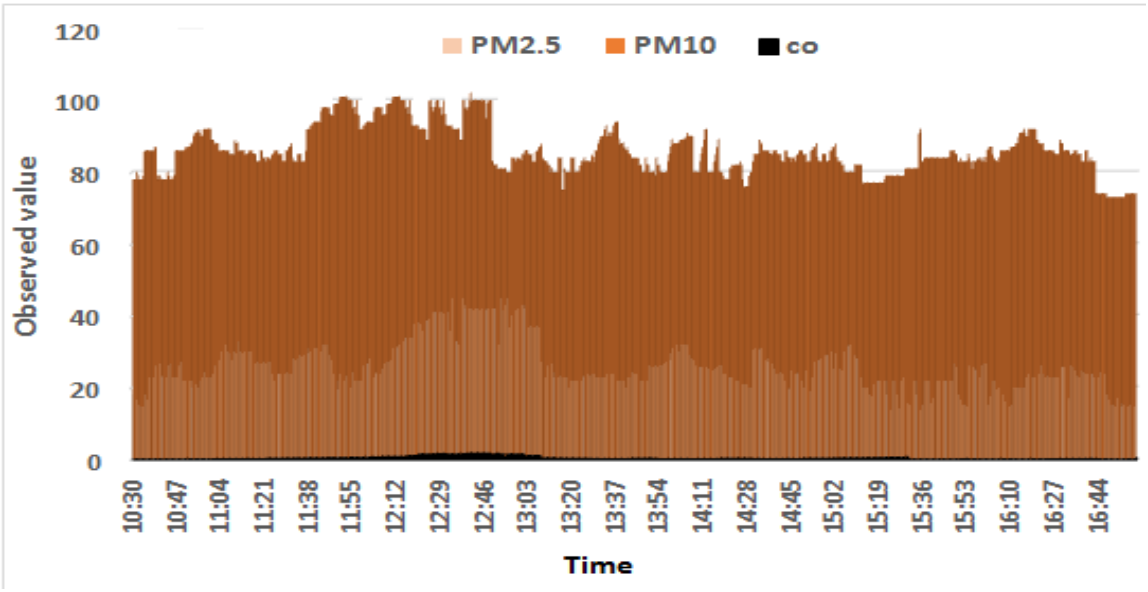
environment) is less polluted compared to the first site (outdoor site). Moreover, the collected pollutant at site 1 also represents a higher concentration of particulate matter and carbon monoxide recorded during the mid-day. The figure depicts that at site 1, the recorded value of air pollutants particulate matter 10 and 2.5 have been varying around 110 and 35 median values, respectively.



(a)



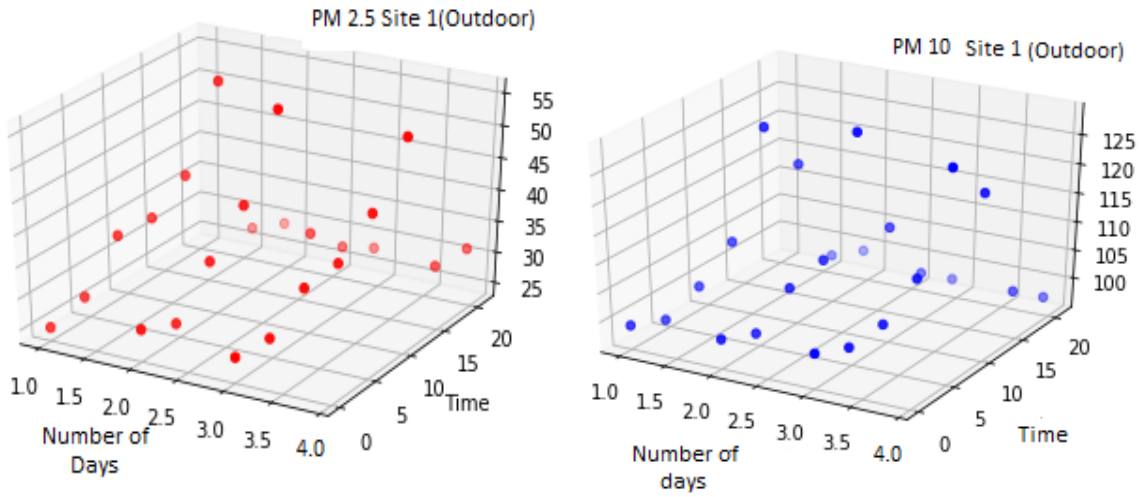
(b)



(c)

Figure. 5.3. (a) distribution comparison of observed Parameters (b) observation of PM2.5, PM10, and CO at the rooftop during day time (site 1: outdoor) (c) observation of PM2.5, PM10, and CO at home during day time (site 2: indoor)

Figures 5.4 (a) and 5.4 (b) show the aggregated air pollutant values of particulate matter 10 and 2.5 for three days. The measured periodical air pollutant values were aggregated every three hours. The figure depicts that the outdoor site has observed the higher air pollution index specifically during the time slot of 12-15 hours each of the three days.



(a) (b)

Figure. 5.4. (a) scatter plot of PM 2.5 at the rooftop (site 1: outdoor) over 3 days (b) scatter plot of PM 10 at the rooftop (site 1: outdoor)

Figure 5.5 represents the screenshots of the python shell at the server showing the received stream of air pollutants from the deployment sites. Figure 5.6 shows some of the snapshots of the mobile application developed for air pollutant parameters monitoring as one of the handy tools.

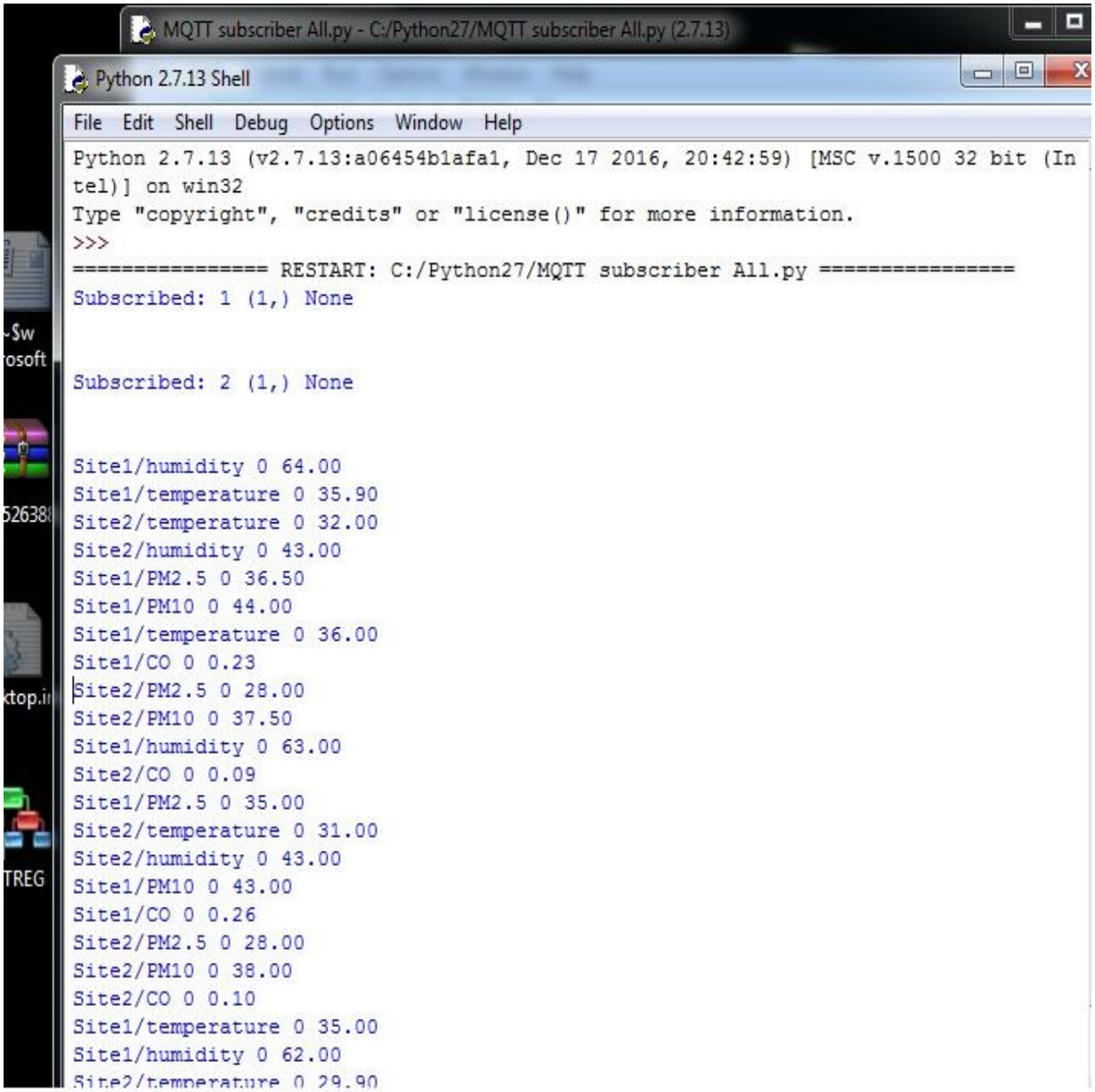


Figure. 5.5. Screenshots of the python shell at sever

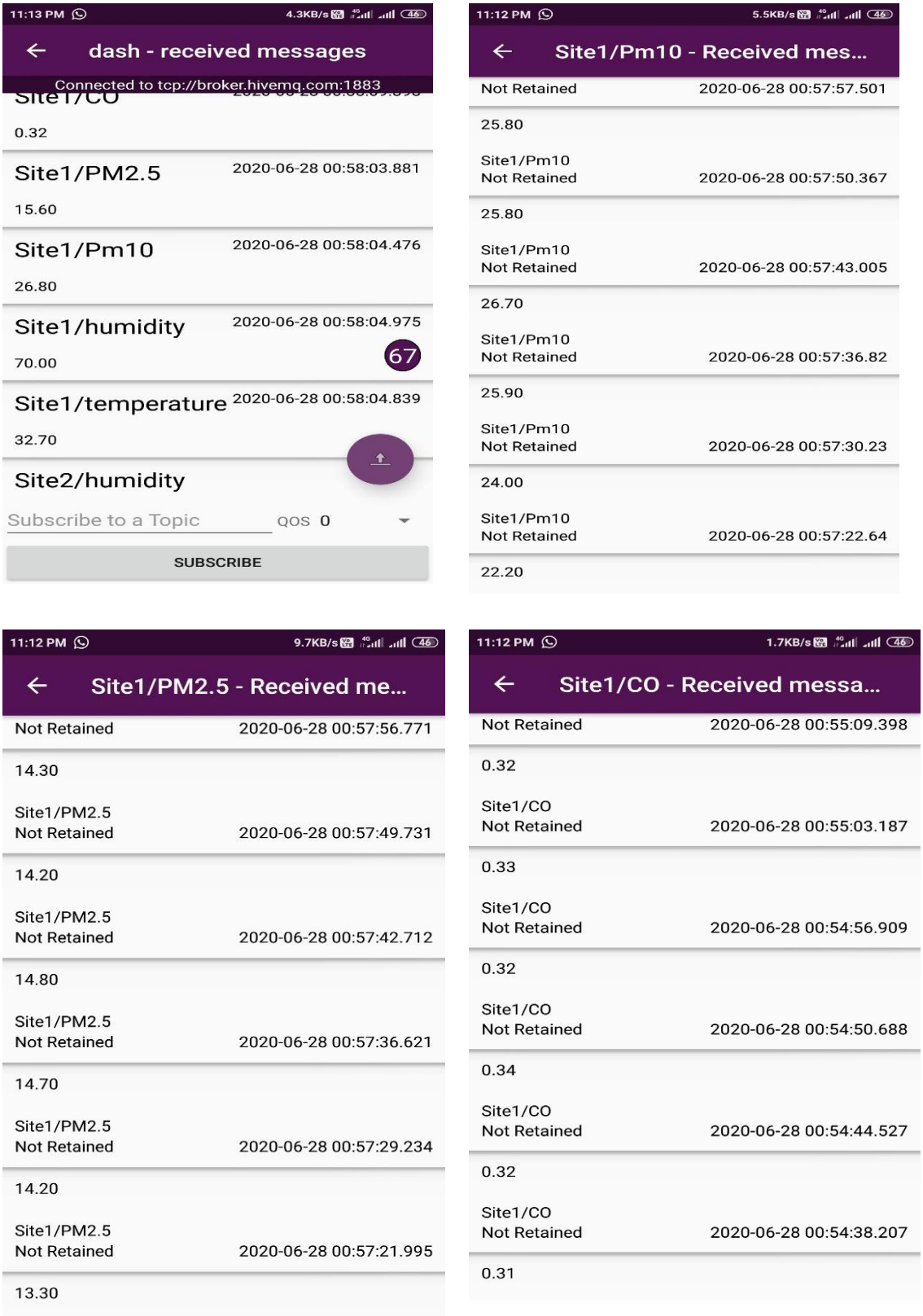


Figure. 5.6. Screenshots of the mobile application

5.1.2 Power Consumption Optimization Scheme and Performance

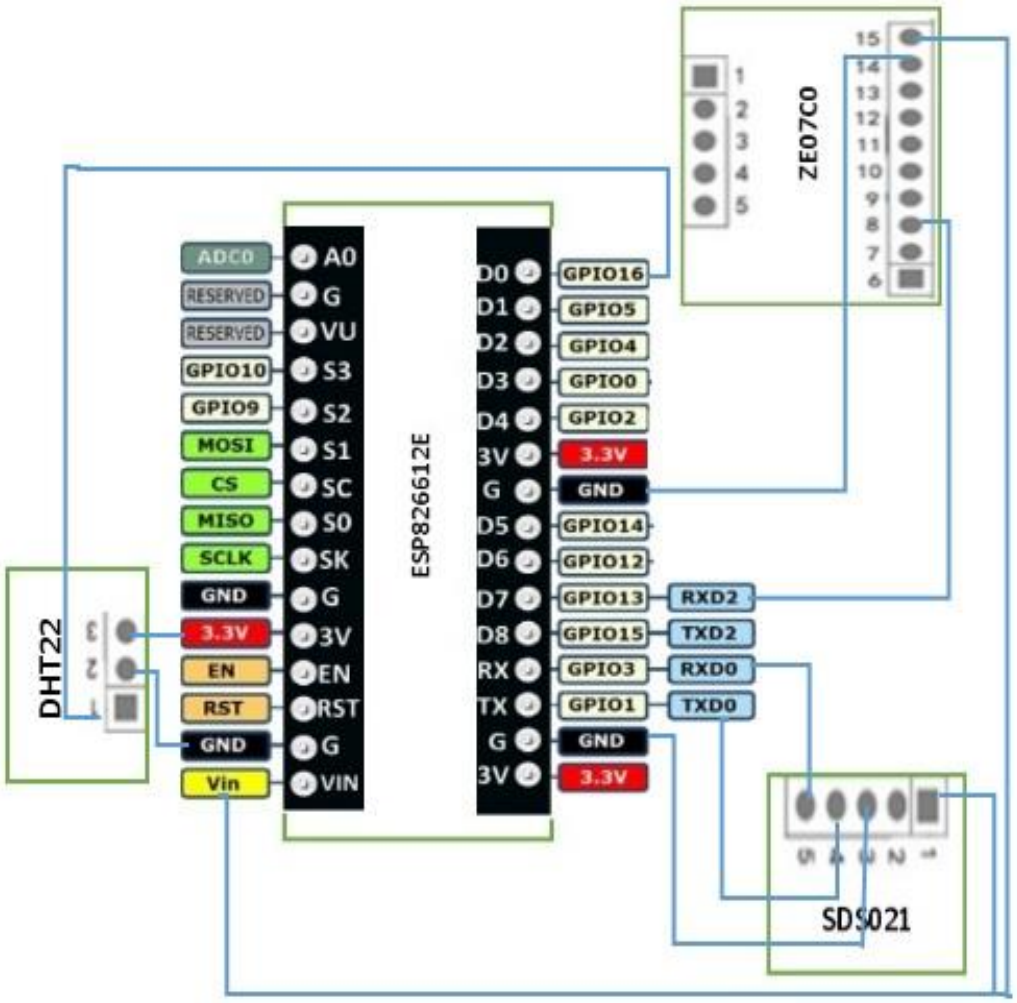


Figure. 5.7. The detailed circuit design of sensor interfacing

The power consumption of the sensing node is a big issue when the deployment is done at remote places. We address the issue and attempt genuine effort to reduce the power consumption of the sensing node by switching the smart node to five different modes. Power consumption of the smart node has been reduced by switching the smart node to sleep mode at suitable intervals, thus the soft solution instead of hardware optimization. The controller in sleep mode draws very little power comparing to regular consumption during the active mode. The ESP8266 12E controller supports deep-sleep, light-sleep, and modem-sleep, three types of sleep modes [126]. The controller draws a 0.9 mA current during light sleep mode and 120 mA in an active mode. The controller has been kept to a light sleep mode when the node is not fetching(reading) the air quality parameters data to reduce the power consumption. The RF

transceiver remains in an idle state while it is reading data from sensors. The transceiver is switched on only when the transmission of the data starts to reduce the power consumption.

Figure 5.7 shows the interfacing of the sensors with the sensor board. As shown in the figure, PM sensor SDS021 and CO sensor ZE07C0 are connected to [vin] directly instead of the power out pin of the controller. The [Vin] here is the direct output from the power backup supply, which supplies the controller's power. So, even if the controller is kept in a light sleep mode, the [vin] will not idle and can supply the necessary power to the two sensors. The sensors can also be put to a power-down state while readings are not taken. It is also possible and beneficial to power down SDS021 by switching the module to a sleep mode. It is necessary to maintain the ZE07CO sensor in the active mode all the time because the sensor needs a warm-up time of 3-minute before getting stable whenever it is switched on and off [127]. So it is not suitable to switch the sensor ZE07CO sensor off during the communication period.

Table 5.1. Sensing cycle phases for various components of sensing unit during parameter reading

Phases	**controller**	**DHT22**	**SDS021**	**ZE07CO**	**Wi-Fi**
P1	light sleep	Power down	sleep	active	OFF
P2	active	active	sleep	Active	OFF
P3	active	active	active	Active	OFF
P4	active	active	sleep	Active	OFF
P5	active	active	sleep	Active	ON

The operation of the sensing node can be separated into five different phases as shown in Table 5.1. Initially, the system remains in the P1 phase. During this phase, the controller stays in a light sleep mode for 60 seconds. During this phase, the PM sensor module stays in hibernation mode, and DHT22 remains in sleep mode. The controller will switch to an active mode from a light sleep mode when the timer expires, and the P2 phase starts. On entering the p2 phase, DHT22 takes 10 seconds considering the stabilization period of 8 seconds (delay) and records the temperature and humidity parameter. During the p3 phase, the ESP8266 12E will activate the SDS021. It will read the particulate matter data for around 10 seconds and switch into sleep mode again. After reading the Carbon monoxide data, the proposed system will enter into the P5 phase. Eventually, a publisher turns

on the transceiver and transmits the gathered data using MQTT messages. The average power usage of the node can be calculated as:

$$P = \left[\frac{I_{lsm} * T_{sleep} + I_{active} * T_{active}}{T_{si}} + I_{CO}\right] * V_{in} \quad (5.1)$$

where I_{lsm} is the current usage by sensing node in the light sleep mode, T_{sleep} is period spent in sleep mode by the node, I_{active} is usual current is drawn when the node is in active mode, T_{active} is the total active time comprised of data collection and stabilization time of the sensors, T_{si} is the total time per sampling duration which comprises of node operation (active period) and sleep period, I_{co} is the current usage of carbon monoxide sensor which is never at rest. Vin is the input voltage power supply. The smart node remains in a sleep mode for 60 seconds duration out of around 90 seconds sampling interval. The ESP8266 12E draws 0.9 mA current during the light sleep mode and 120 mA in an active mode as per the manufacturers' datasheet. The average power consumption of a smart node for 90 seconds is 316 mW plus the consumption of CO sensor (CO sensor is never at rest) by applying a power optimization scheme around 900 mW plus the consumption of CO sensor without power optimization. The battery's lasting time is extended, as mentioned in the table below during the experiments.

	Without the Power Reduction scheme	**Under Power Reduction Scheme**
Battery Life Time (with 2500 mAh)	7 hrs 15 minutes	12 hrs 50 minutes

5.1.3 Event-based Transmission for Power Consumption Optimization and Performance

The average power consumption of the sensing node depends on two criteria, the number of readings taken periodically (sampling frequency) and the number of transmissions that occur of the sensed parameters. It can be seen from Table 5.1 that the transceiver is activated only when the transmission takes place. The power consumption of the system can be reduced further if the number of transmissions can be decreased, in addition to the power optimization method represented above.

$$|\overline{X_t} - \overline{X_{t-1}}| > \delta\, \overline{X_{t-1}} \quad (5.2)$$

Where, $\overline{X_t} = \frac{\sum_{k=t-(N-1)}^{k=t} X_k}{N}$ and $\overline{X_{t-1}} = \frac{\sum_{k=t-(N-1)}^{k=t-1} X_k}{N-1}$

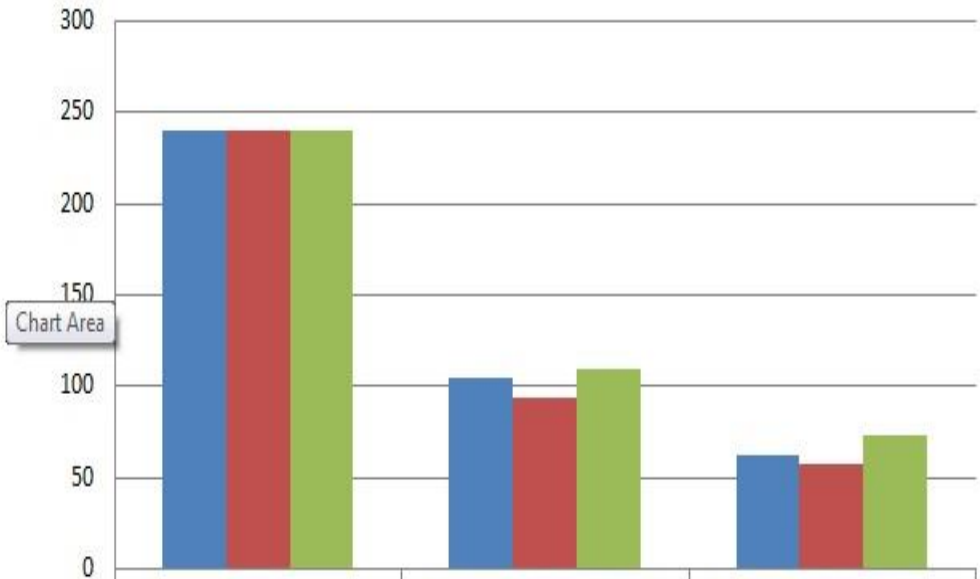
The proposed air quality monitoring system was also investigated and implemented with the event-based transmission with the goal of overall transmission reduction and eventually the power consumption reduction. The proposed event-based transmission uses equation 5.2 for the decision of transmission. On fulfilling the condition of the equation, the transmission happens otherwise not. The $\overline{X_t}$ is the average of the last N measurement, including the latest measurement at time t and $\overline{X_{t-1}}$ is the average of previous N-1 measurement till time step t-1 (excluding the recent measurement). If the change in the average due to the contribution of the current or recent value is greater than δ percent, then the update or change is said to be significant for reporting, and thus the transmission happens. If any recent value of the carbon monoxide, PM 2.5, or PM 10 makes the condition true of the equation, then the MQTT message generated by the publisher is sent; otherwise, the message transmission will be skipped.

Algorithm 4. MQTT publisher with event-based transmission

Step 1:	The smart node gets registered with the MQTT broker using a unique ClientID.
Step 2 :	The ESP8266 12E (publisher) turns on the Particulate matter sensor, fetches the observations from three sensors, and again turns the SDS021 sensor into the hibernation mode. Store and swap the values of parameters for keeping the previous two values that can be used in the calculation of equation 5.2 in step 3. (for N=3)
Step 3 :	If (the equation is true for any of the sensed parameters) or (skip counter=9) Reset the skip counter to zero and go to step 4 Else increase the skip counter and go to step 7.
Step 4:	The publisher creates the MQTT message by allocating the value fetched in step2 to the relevant sub-topic.
Step 5:	The publisher gets connected to the HIVEMQ broker, authenticated using the unique ClientID.
Step 6:	The publisher's MQTT message created in step 4 is published using the topic set for an individual site.
Step 7:	The controller again switches to a light sleep mode.
Step 8:	After the timer gets expired, the controller auto awakes from sleep mode and then goes to step 2.

The above algorithm displays the MQTT publisher with the event-based transmission scheme. The gradual and steady small change can make it happen that the condition of the equation never (or for a longer period) becomes true under the employed scheme. Due to that, the air quality parameters cannot be reported at the server for a very long period. To stop such kind of scenario, one more condition is added to the scheme. If continuous nine transmissions are skipped in the algorithm (no transmission of observed data), then forceful tenth transmission occurs. The event-based transmission scheme is implemented using a skip counter that is incremented for each transmission skipping in the MQTT publisher.

Figure 5.8 represents the total number of transmissions in the applied scheme experimented for 6 hours a day. The number of transmissions is calculated by considering the number of MQTT messages subscribed and logged at the subscriber. The figure shows very few transmissions in the event-based transmission scheme with N=3. Reduced number of transmissions results in power consumption reduction.



	Periodic	Event based transmission (with N=3, δ=0.05)	Event based transmission (with N=3, δ=0.10)
Number of transmissions (day one)	240	104	62
Number of transmissions (day Two)	240	93	57
Number of transmissions (day Three)	240	109	73

Figure. 5.8. Message transmission under event-based transmission

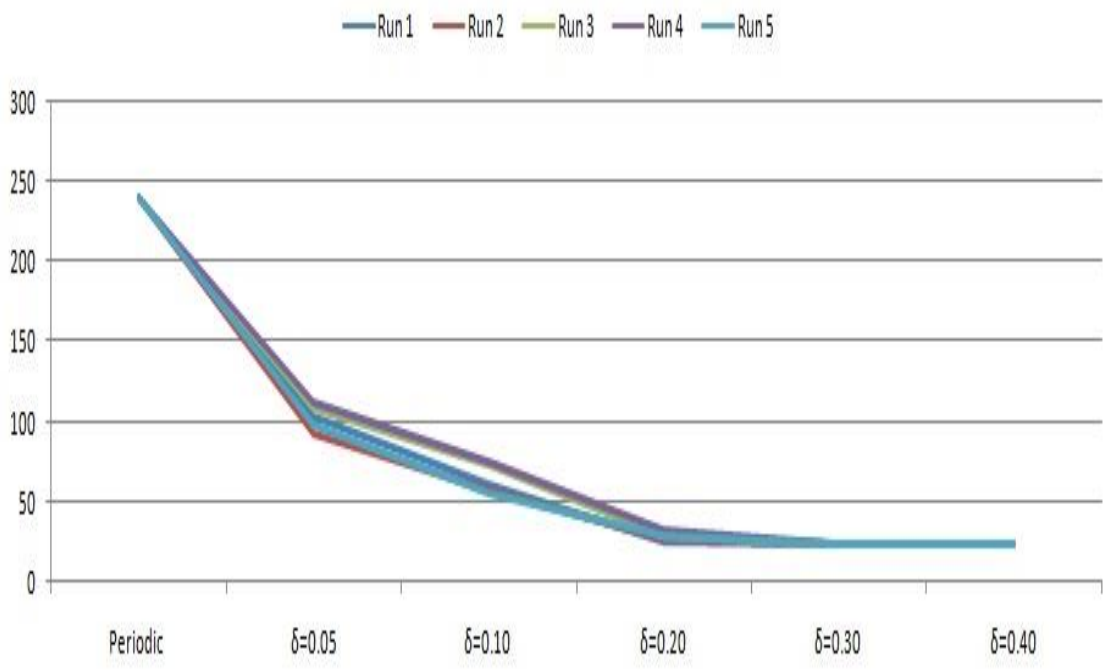


Figure. 5.9. Message transmission for N=3 and various values of delta

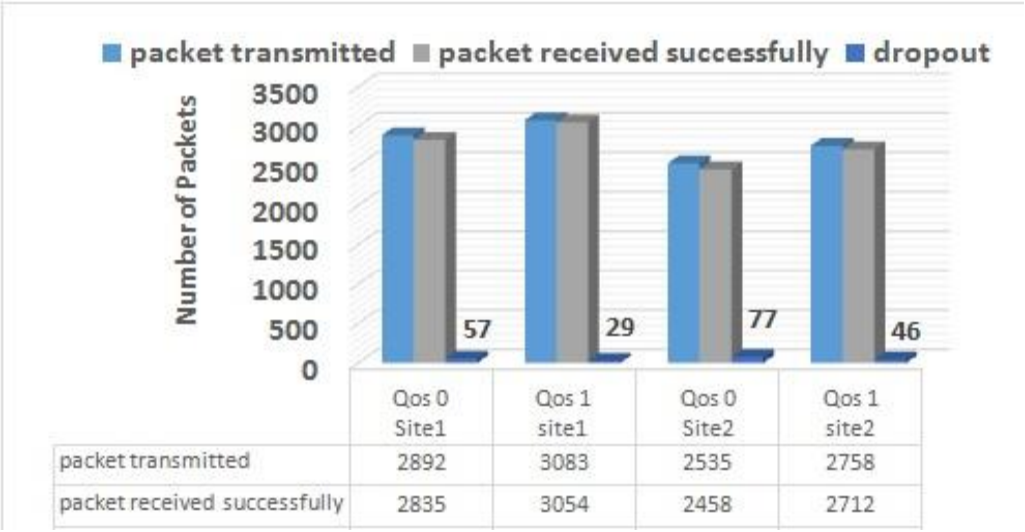
The employability of such a scheme depends on the trade-off between tolerance requirements in micro-level changes (to be reported at the server) versus power consumption. Figure 5.9 shows the effect of the delta value on the number of transmissions over five different runs. The number of transmissions becomes periodic transmission under the larger delta value, as shown in figure 5.9.

5.1.4 Quality of Service(QoS) and System Performance under Periodic Transmission

IoT-based ecosystems such as air quality monitoring systems are transporting real-time parameters and updates to the remote server. Such a system can also provide threshold-based notifications based on received data. In such a real-time system, delivering the messages (in terms of messages transmitted Vs. messages received) to the subscribers is a very important parameter. Reliable delivery or accuracy is one of the metrics representing the Quality of Service provided by the system. Thus implementation of QoS adds value to such a diverse system by providing performance, visibility, and usability of the services offered. Very few efforts have been attempted to implement and assess the performance of the implemented system under complex architecture design.

There are three levels of Quality of Services supported by the MQTT publisher. The least reliable level is Level 0 in these three levels. Level 0 is fire and forgets the type of delivery

where the MQTT publisher transmits the MQTT message and does not bother about the actual delivery at the destination happen or not. The second is QoS level 1, where the publisher retains the message until the acknowledgment - the PUBACK message received from the broker. The message will be published again if the PUBACK acknowledgment is not available. Here the multiple deliveries at the destination are possible. The QoS level 2 guarantees precisely one message delivery to the subscriber. Sender and receive use various message identification for the synchronization of the message delivery. A publisher sends the message again with a duplicate flag if PUBREC is not acknowledged.



	Qos 0 Site1	Qos 1 site1	Qos 0 Site2	Qos 1 site2
packet transmitted	2892	3083	2535	2758
packet received successfully	2835	3054	2458	2712

Figure 5.10. System performance over an MQTT protocol: rooftop (site 1: Outdoor) and home (site 2: Indoor)

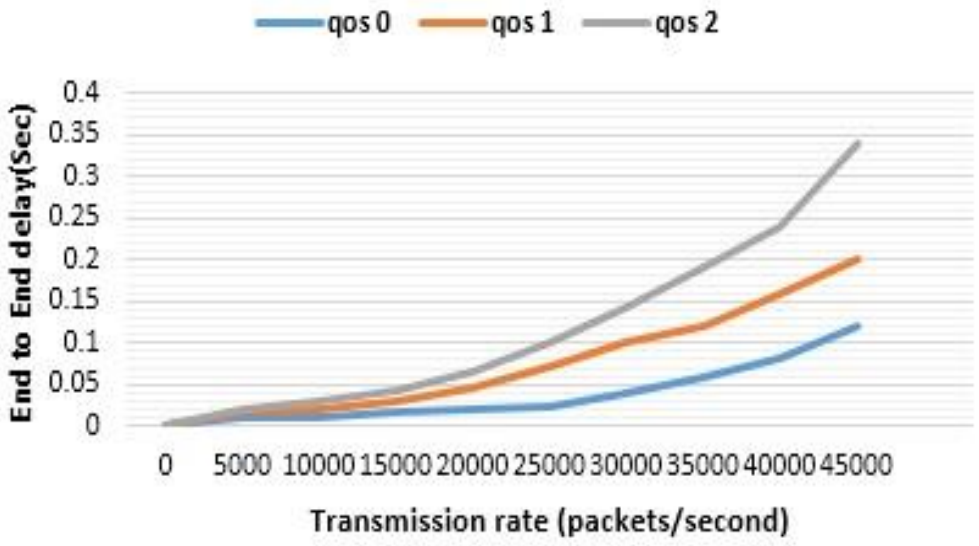


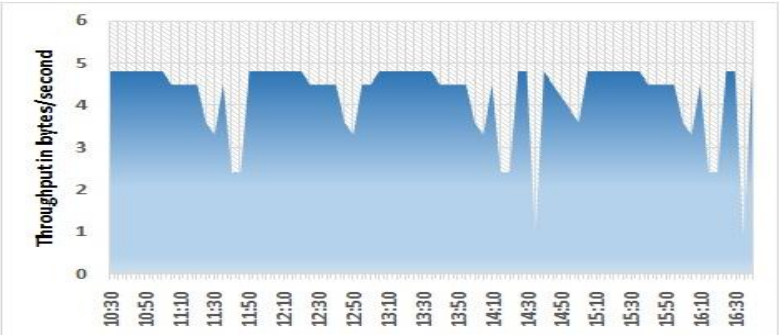
Figure. 5.11. End to End delay against QoS level in simulation

Figure 5.10 depicts the system performance at level 0 and level 1 of QoS for 12 hours. With the QoS level 0, the packet dropout ratio observed is 1.97%, and for level 1, the dropout ratio is around 0.94 % at site1. For understanding the effect of Quality of Service level on delay (end to end), the publisher environment is simulated using MQTT-JMeter (apache tool). The JMeter is configured with the MQTT plugin, which can serve to accomplish testing in which simulated clients register to the broker.

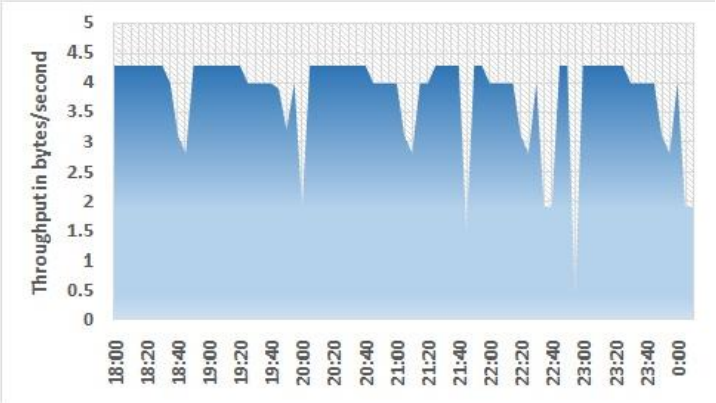
Figure 5.11 represents the correlation between the transmission rate and end-to-end delay under three QoS levels during the simulation. It can be seen from the figure that the end-to-end delay starts increasing when a packet is transmitted is shifted from a lower level to a higher level of QoS. The effect is observed in the account of the retransmission and acknowledgment overhead. The choice of the QoS level is a very important criterion for mitigating end-to-end delay and packet loss ratio.

The retransmission and acknowledgment overhead also affect the energy consumption of the system. We have not analysed the power consumption performance with the selection of higher QoS level during our experiments, but the energy consumption is expected to be more with the higher QoS level selection [145]. The QoS level selection is clear trade-off between the allowable message loss and the system performance (higher end to end delay, higher power consumption). Also the optimization strategy for QoS overhead (other than OoS 0) can be addressed in future work.

The accuracy values observed at site 1(outdoor) and site 2(indoor) are 98% and 96%, respectively, as shown in figure 5.10. The accuracy values are calculated by considering the total number of packets transmitted and received at the two sites. The dropout and accuracy show the system's performance in terms of reliable delivery of MQTT messages, including sensing parameters. Figure 5.12 shows the average throughput of the system. It can be seen that the observed average throughput of the smart node is around 4.28 for site 1 and 4.6 for site 2 in bytes per second for 6 hours under periodic transmission.



(a)



(b)

Figure. 5.12. Throughput of the Sensing Unit (a) home (site 2: indoor) (b) rooftop (site 1: Outdoor)

5.2 RESULTS AND DISCUSSION OF PROPOSED AIR QUALITY PARAMETERS PREDICTION USING FBLSTM

Experimentations of the proposed prediction system or model(FBLSTM) is conducted using Keras 2.1.6; Keras, in turn, uses Tensorflow as the back end. The proposed model utilizes 60 units in each layer of LSTM. The data are scaled as discussed in the data preparation section, and each sample's input window or sequence size is kept to be sixty. In the Keras package, the long short-term memory layer is shaped with a 3-dimensional vector. The vector is to be initialized with fields (sample space size, timestep observations in a sample, and feature). The training of the model is achieved by utilizing the stochastic gradient descent(SGD) optimization algorithm. The SGD algorithm equates the prediction to original observation, and the difference is used to approximate the error gradient. The error gradient is then utilized to modify or update the weights and biases in the neural network. The SGD [142] algorithms are facing the problem of determining the optimal step size. The issue is resolved by developing the new optimization algorithm, i.e., ADAM [142]. Adaptive moment estimation(ADAM) is one of the best stochastic optimization algorithms for deep neural network learning, and it realizes the advantages of two broadly used algorithms AdaGrad and RMSProp. We used the ADAM algorithm in the proposed model for optimization. The ADAM algorithm adjusts the rate of learning based on the average of first and second moments of the gradients. The ADAM delivers quick convergence with less memory requirement than the other two stochastic algorithms [142].

The batch size in a recurrent neural network can be defined as the total number of individual training samples used (after processing that many samples, the error gradient calculated) to estimate the error gradient. The batch size is one of the hyperparameters for the ADAM optimization algorithm. We keep the batch size of 32 during the experiments. The number of unidirectional layers in stacking to gain minimum loss for predicting air pollutant time series data is decided through experiments. The return_sequence attribute is set to true in Keras, while the output of one LSTM layer is given as input to the subsequent layer. So, instead of giving one output, the LSTM layer gives output for each timestep. The backward pass layer is implemented by setting go_backwards to be true in Keras. We use the functional API of Keras for building the proposed training model. Following is the stepwise algorithm of the proposed model implementation for prediction using Keras.

Stepwise process of model creation using Keras APIs

Tools used in experiments:

- Anaconda distribution with conda virtual environment manager
- Spyder open-source IDE
- Keras - with TensorFlow backend

Step 1: Read CSV file in panda DataFrame

Step 2: Create NumPy array from panda Data Frame

Step 3: Scale down data in [0-1] using scaler - Min-Max Scaler

Step 4: Create and fill train and test data structure as discussed in data preparation

- according to the time steps parameter

Step 5: Reshape data structure matching to Tensorflow

- Parameters: batch size, time step, and features

Step 6: Create a sequential model from Keras

Step 7: Create forward and backward LSTM layer

- set return_sequence=true for both layer and go_backwards = true for backward layer

Step 8: Add both layers using bidirectional and shape them for Tensorflow

- set merge option to the Alternatives available in Keras and of the choice

Step 9: Add hidden layers(LSTM) as per requirement

- units, activation function, return_sequences = true

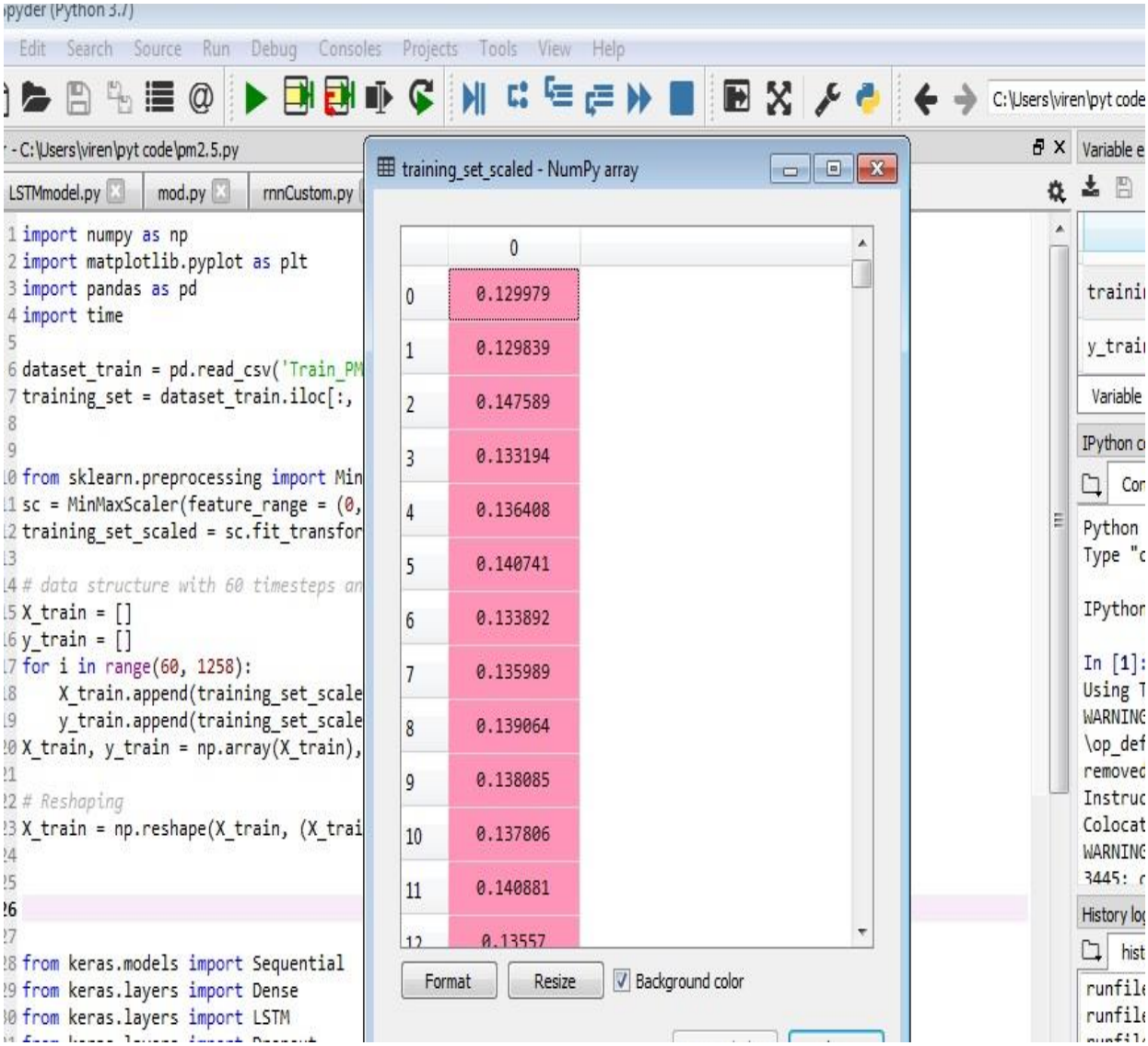
Step 10: Add Dense output layer

Step 11: Compile the model with gradient optimization algorithm and loss function

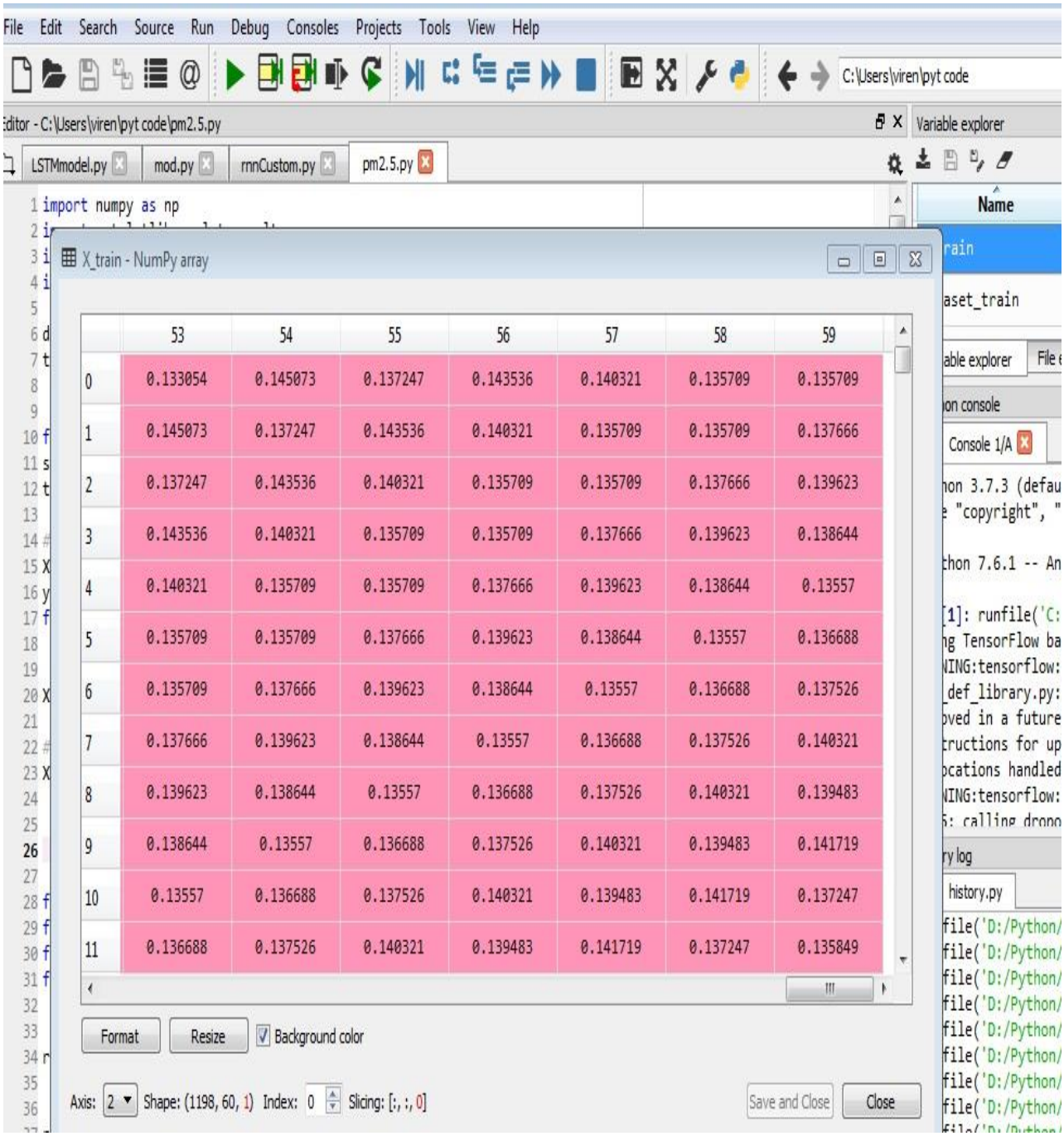
Step 12: Train with the fit function of the model created

- parameters: no. of epochs, batch size for gradient update

Sample screenshot showing the scaled training time-series sequence of PM2.5

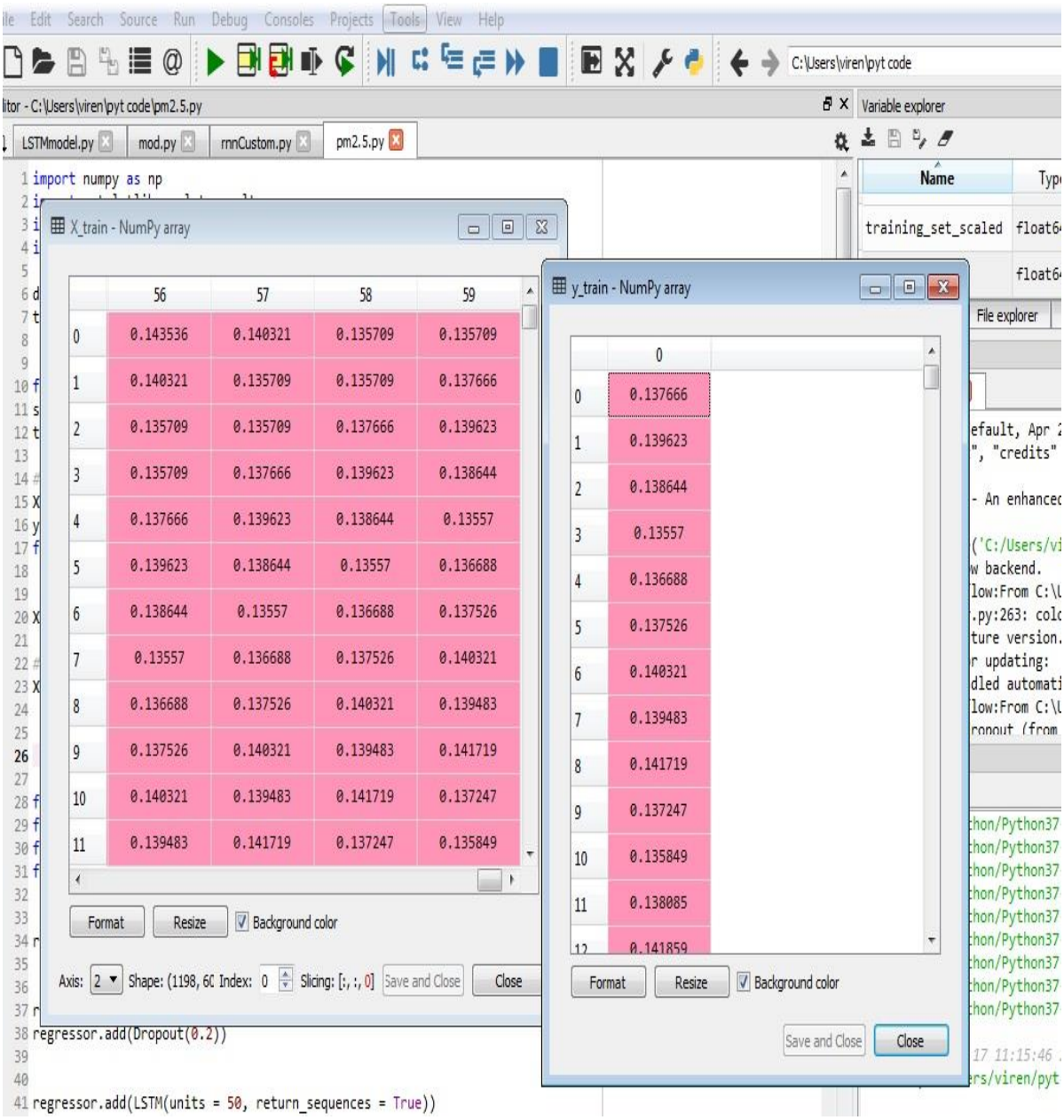


Sample screenshot of supervised learning data structure: Input window with 60-time steps for PM2.5

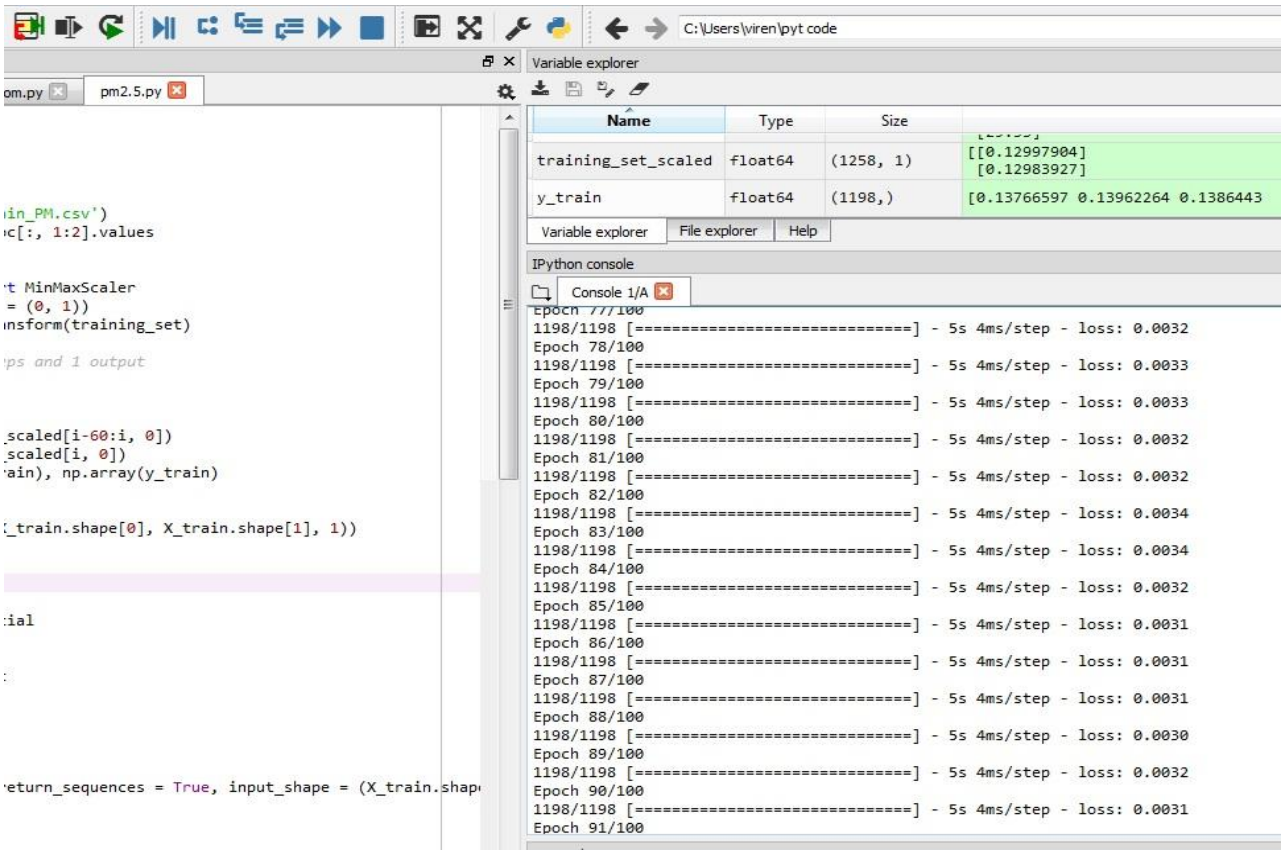


	53	54	55	56	57	58	59
0	0.133054	0.145073	0.137247	0.143536	0.140321	0.135709	0.135709
1	0.145073	0.137247	0.143536	0.140321	0.135709	0.135709	0.137666
2	0.137247	0.143536	0.140321	0.135709	0.135709	0.137666	0.139623
3	0.143536	0.140321	0.135709	0.135709	0.137666	0.139623	0.138644
4	0.140321	0.135709	0.135709	0.137666	0.139623	0.138644	0.13557
5	0.135709	0.135709	0.137666	0.139623	0.138644	0.13557	0.136688
6	0.135709	0.137666	0.139623	0.138644	0.13557	0.136688	0.137526
7	0.137666	0.139623	0.138644	0.13557	0.136688	0.137526	0.140321
8	0.139623	0.138644	0.13557	0.136688	0.137526	0.140321	0.139483
9	0.138644	0.13557	0.136688	0.137526	0.140321	0.139483	0.141719
10	0.13557	0.136688	0.137526	0.140321	0.139483	0.141719	0.137247
11	0.136688	0.137526	0.140321	0.139483	0.141719	0.137247	0.135849

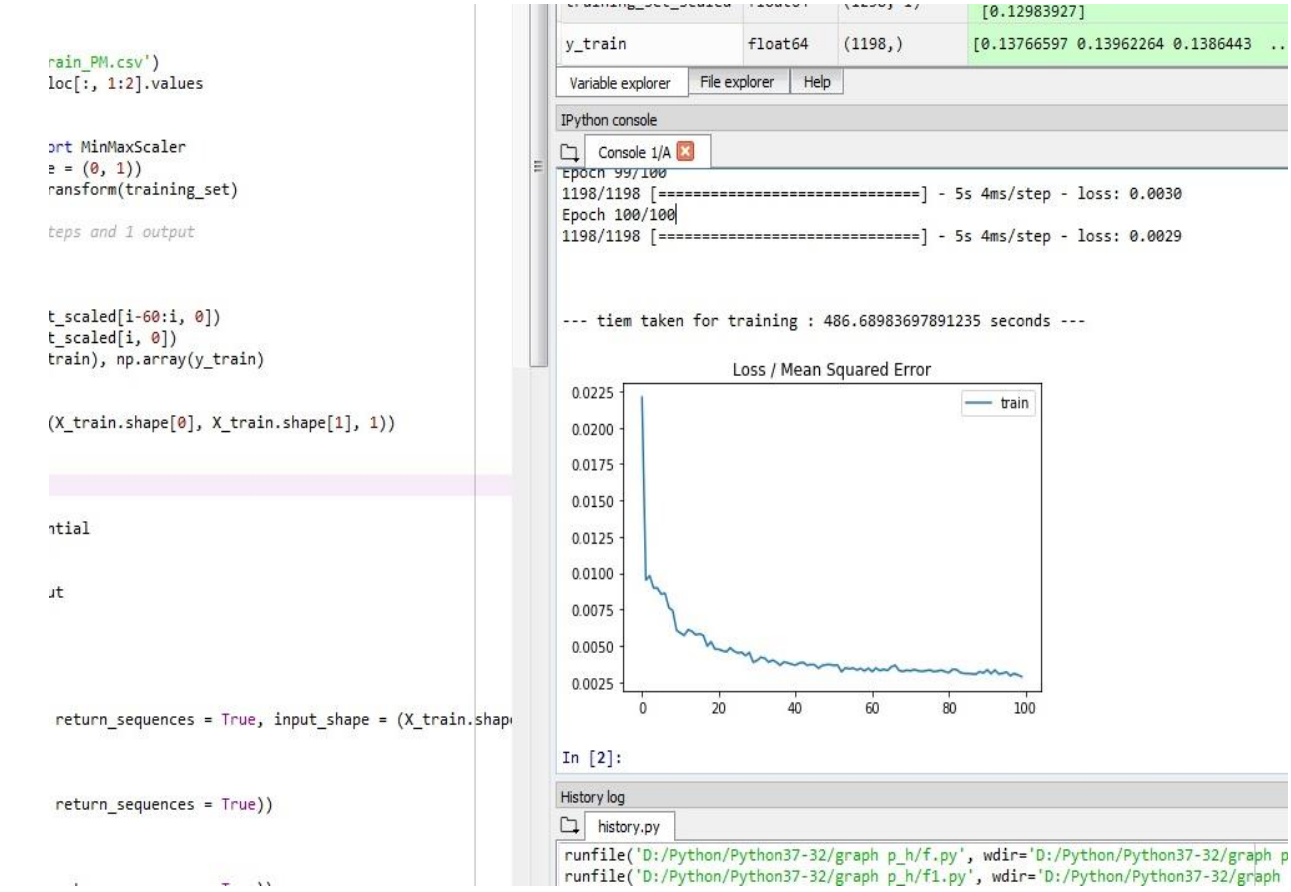
Sample screenshot Input window with 60-time steps with corresponding output(target) window for PM2.5



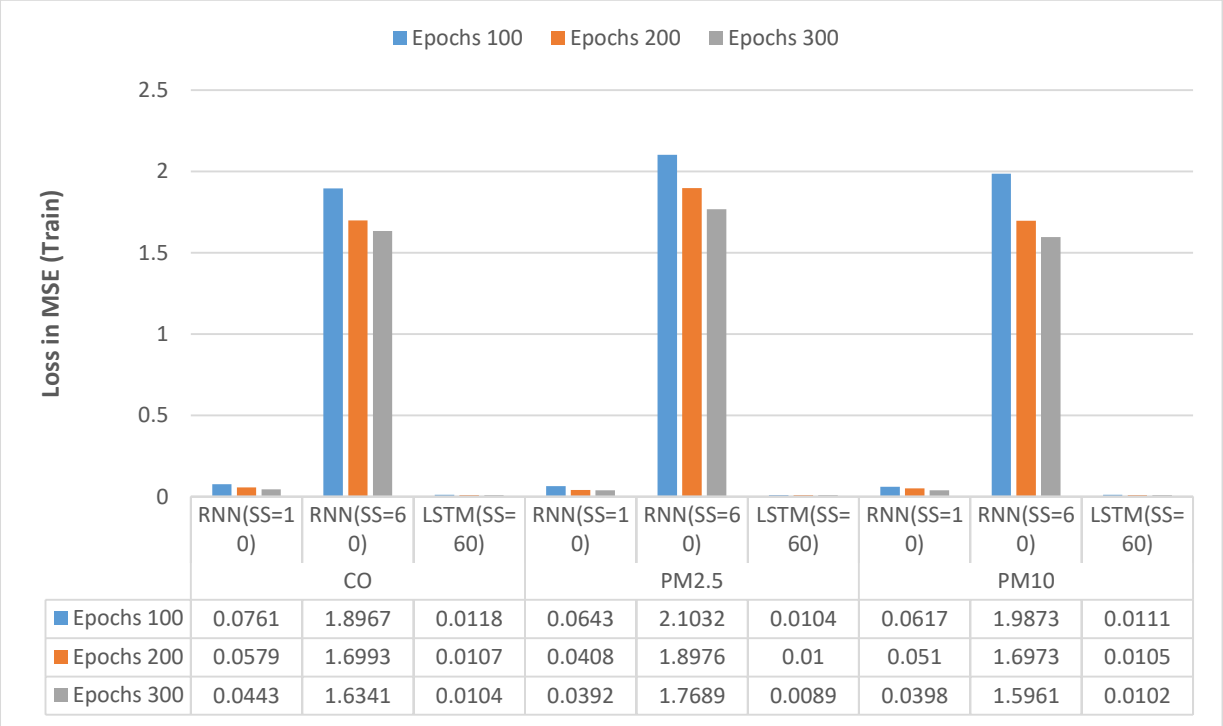
Sample screenshot showing performance after each epoch during training



Sample screenshot: plotting of MSE per epoch (up to 100) during training

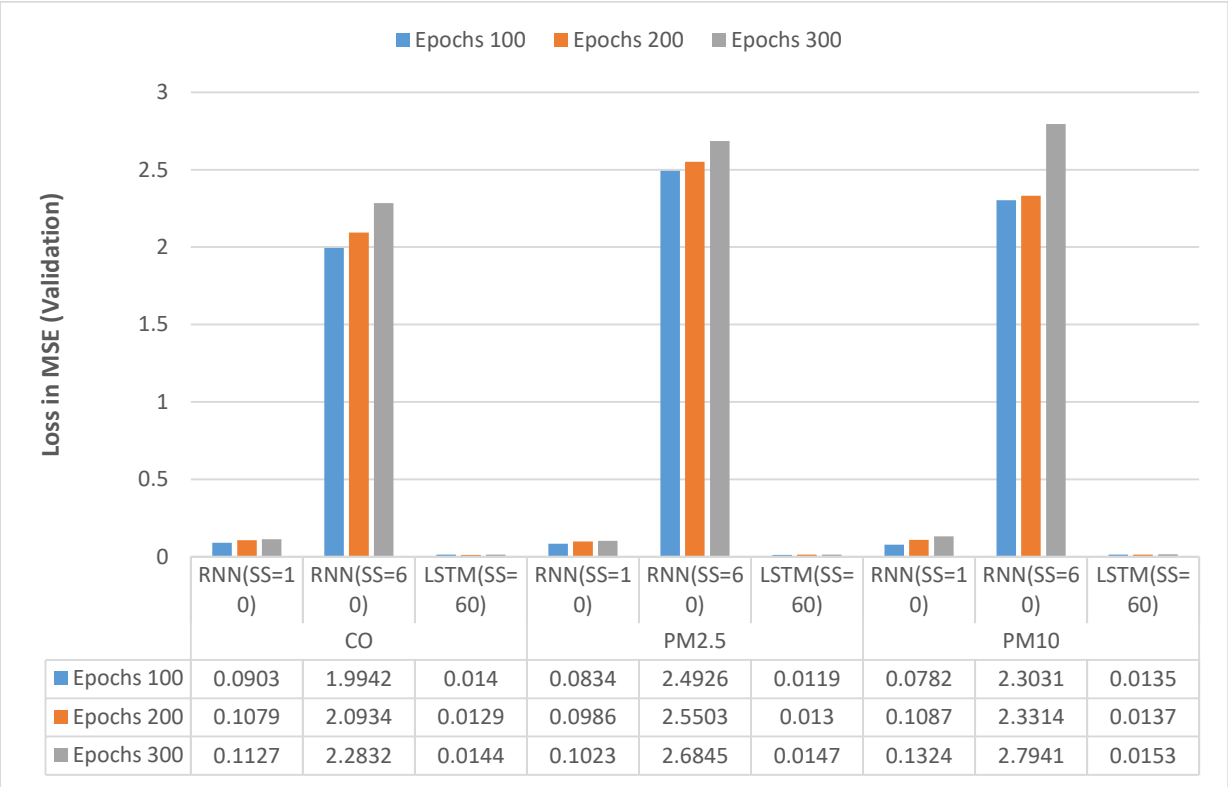


5.2.1 Performance Comparison of the Proposed Model:



	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)
	CO			PM2.5			PM10		
Epochs 100	0.0761	1.8967	0.0118	0.0643	2.1032	0.0104	0.0617	1.9873	0.0111
Epochs 200	0.0579	1.6993	0.0107	0.0408	1.8976	0.01	0.051	1.6973	0.0105
Epochs 300	0.0443	1.6341	0.0104	0.0392	1.7689	0.0089	0.0398	1.5961	0.0102

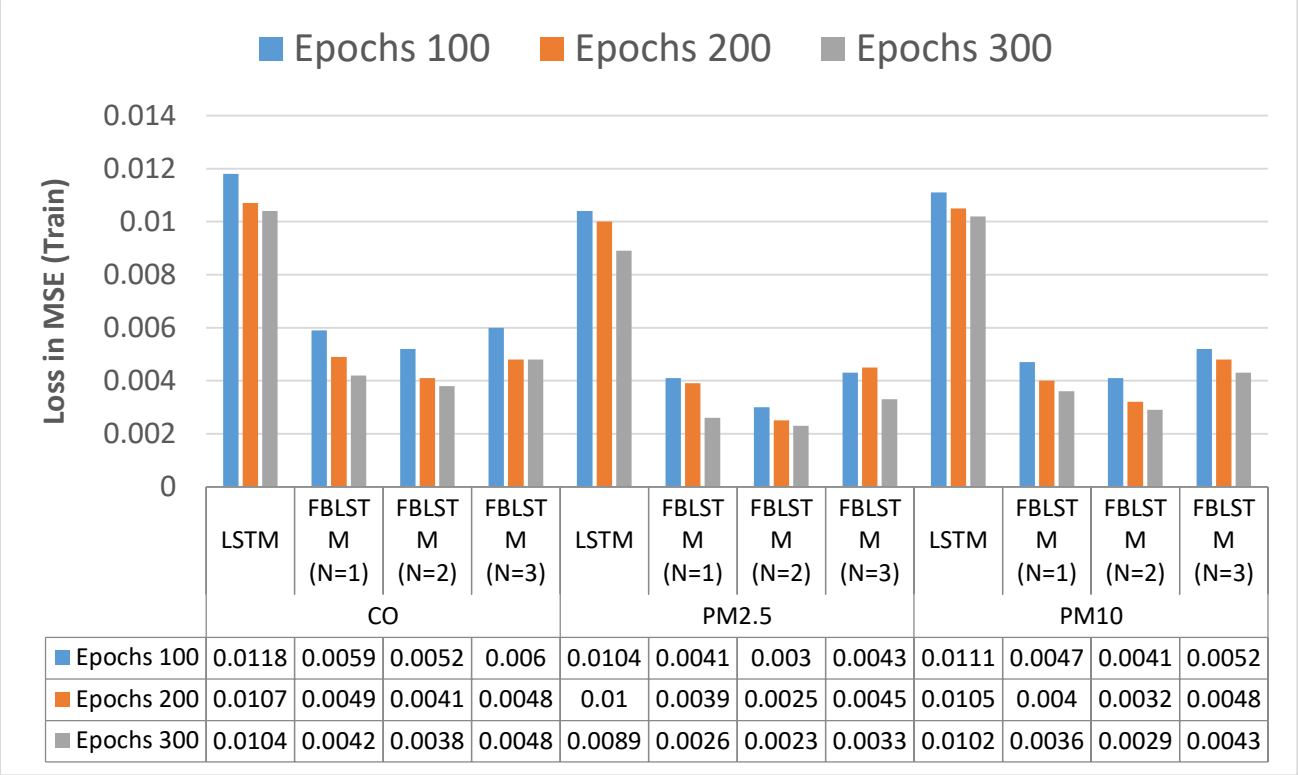
(a)



	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)	RNN(SS=10)	RNN(SS=60)	LSTM(SS=60)
	CO			PM2.5			PM10		
Epochs 100	0.0903	1.9942	0.014	0.0834	2.4926	0.0119	0.0782	2.3031	0.0135
Epochs 200	0.1079	2.0934	0.0129	0.0986	2.5503	0.013	0.1087	2.3314	0.0137
Epochs 300	0.1127	2.2832	0.0144	0.1023	2.6845	0.0147	0.1324	2.7941	0.0153

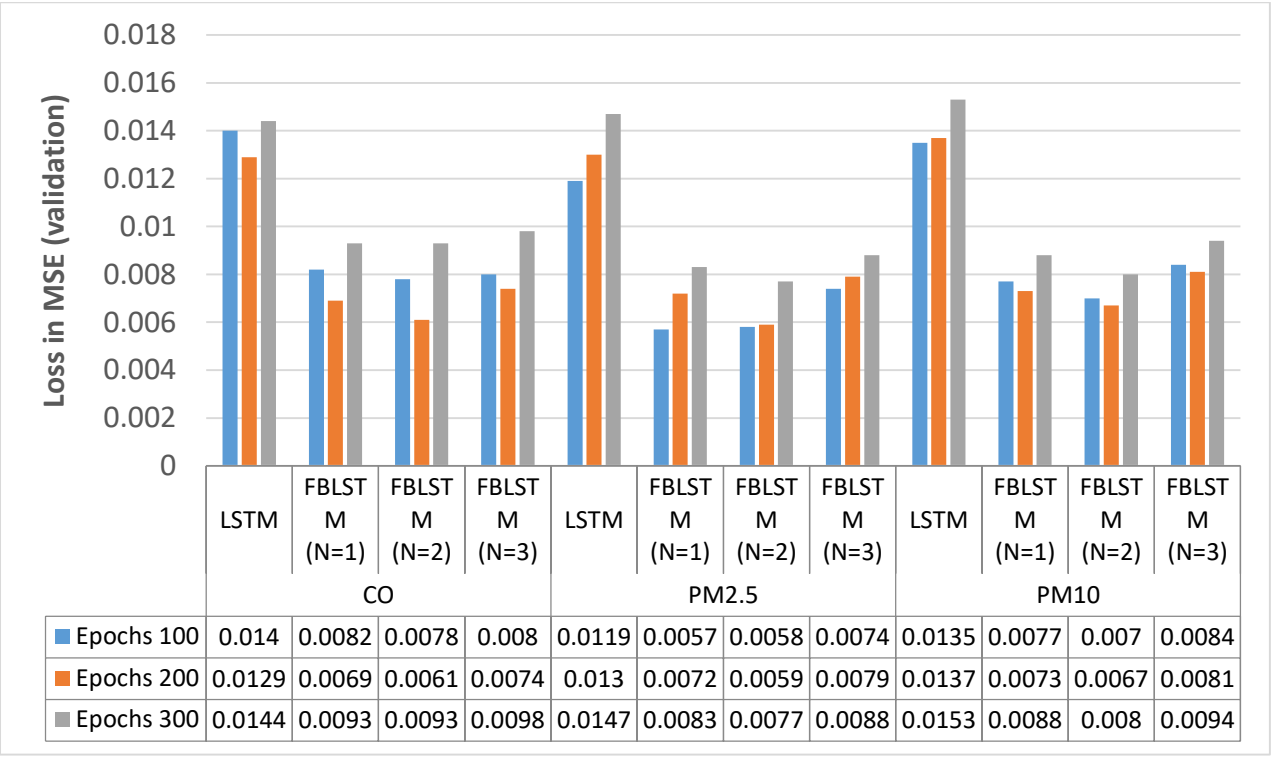
(b)

Figure. 5.13 Comparison of MSE of LSTM and RNN for (a) training and (b) validation



	CO				PM2.5				PM10			
	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)
Epochs 100	0.0118	0.0059	0.0052	0.006	0.0104	0.0041	0.003	0.0043	0.0111	0.0047	0.0041	0.0052
Epochs 200	0.0107	0.0049	0.0041	0.0048	0.01	0.0039	0.0025	0.0045	0.0105	0.004	0.0032	0.0048
Epochs 300	0.0104	0.0042	0.0038	0.0048	0.0089	0.0026	0.0023	0.0033	0.0102	0.0036	0.0029	0.0043

(a)



	CO				PM2.5				PM10			
	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)	LSTM	FBLSTM (N=1)	FBLSTM (N=2)	FBLSTM (N=3)
Epochs 100	0.014	0.0082	0.0078	0.008	0.0119	0.0057	0.0058	0.0074	0.0135	0.0077	0.007	0.0084
Epochs 200	0.0129	0.0069	0.0061	0.0074	0.013	0.0072	0.0059	0.0079	0.0137	0.0073	0.0067	0.0081
Epochs 300	0.0144	0.0093	0.0093	0.0098	0.0147	0.0083	0.0077	0.0088	0.0153	0.0088	0.008	0.0094

(b)

Figure. 5.14 Comparison of MSE of proposed model(FBLSTM) with LSTM for (a) training and (b) validation

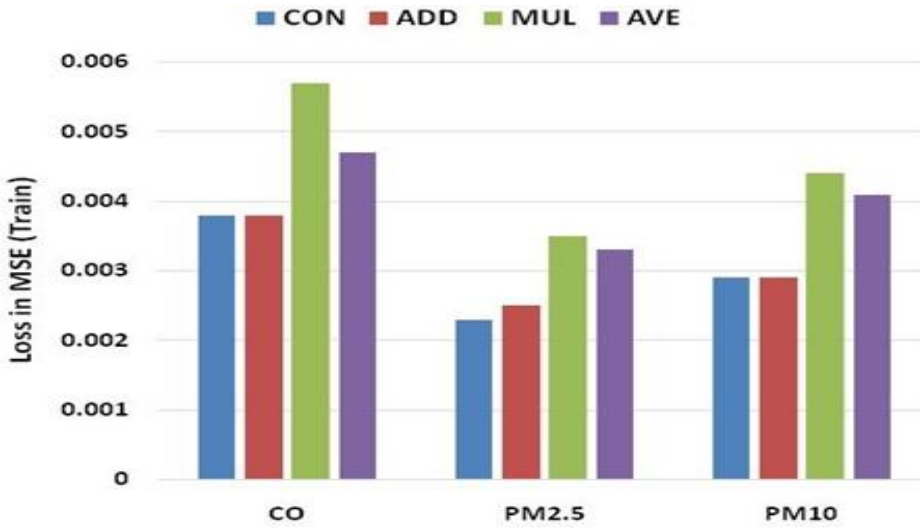
Table 5.2.: Comparison of MSE of the proposed model(FBLSTM) with LSTM and RNN for training and validation

		Epochs 100		Epochs 200		Epochs 300	
		Train	**Val.**	**Train**	**Val.**	**Train**	**Val.**
FBLSTM (1 hidden layer, CONCAT merge fun., sequence size =60)	**CO**	0.0059	0.0082	0.0049	0.0069	0.0042	**0.0093**
	PM2.5	0.0041	**0.0057**	0.0039	0.0072	0.0026	0.0083
	PM10	0.0047	0.0077	0.0040	0.0073	0.0036	0.0088
FBLSTM (2 hidden layer, CONCAT merge fun., sequence size =60)	**CO**	**0.0052**	**0.0078**	**0.0041**	**0.0061**	**0.0038**	**0.0093**
	PM2.5	**0.0030**	0.0058	**0.0025**	**0.0059**	**0.0023**	**0.0077**
	PM10	**0.0041**	**0.0070**	**0.0032**	**0.0067**	**0.0029**	**0.0080**
FBLSTM (3 hidden layer, CONCAT merge fun., sequence size =60)	**CO**	0.0060	0.0080	0.0048	0.0074	0.0048	0.0098
	PM2.5	0.0043	0.0074	0.0045	0.0079	0.0033	0.0088
	PM10	0.0052	0.0084	0.0048	0.0081	0.0043	0.0094

Figure 5.13 compares the loss in MSE (mean squared error) for simple LSTM and RNN based networks. Figure 5.14 shows the loss in MSE when applied with different stacking options (number of stacking layers N=1,2 and 3) for the FBLSTM model and compares it with the simple LSTM model. The MSE values consider for plotting the graph are the averaged MSE values calculated over six repeated runs. The experimentations are executed till 300 epochs and the MSE is highlighted at the end of 100, 200, and 300 epochs for both train and test(validation) data. The FBLSTM performance shown in figure 5.14 is implemented with the "CONCAT" (Con) function for merging the two layers (forward and backward), which is the default one in Keras.

RNN is facing the vanishing gradient problem as the sequence sample size grows, which can also be seen from figure 5.13 in the performance evaluation. On the increase of sequence size from 10 to 60, the MSE(loss) value also increases. Figure 5.14 shows the comparison of LSTM and FBLSTM with one, two, and three hidden layers. The FBLSTM outperforms the simple LSTM model. The Mean Squared Error values for the FBLSTM with

different stacking layers are shown in table 5.2 is the reproduction of figure 5.14, just to realize the comparison at a glance. The MSE values in the table highlighted in bold represent the minimum observed loss. The minimum value of the loss, in turn, indicates the best accuracy for time series prediction. It can be seen from the table that in the FBLSTM approach, along with the mentioned hyperparameter, the minimum MSE can be realized with the stacking of two layers. Going further by adding more layers to the existing unidirectional stacking, i.e., 3 layers, the performance begins degrading. The FBLSTM model performs better than the RNN and the simple LSTM layer.



(a)

(b)

Figure 5.15: MSE for merge function alternatives over: (a) training data (b) validation data

The performance of the bidirectional LSTM is influenced by the way we merge the two layers of two directions. There are four different alternatives for merging that exist in Keras. The Keras implements the Concat(Con) merging option by default, in which the outputs of the respective cell state from the two layers are concatenated together. Mul and Add are other two merge modes or functions in which these outputs from two layers are multiplied or added, respectively. Ave is the fourth alternative for which the average of the corresponding outputs from the two layers' cell state is considered. We applied the optimum FBLSTM architecture as shown in the above results, which have two hidden layers, and investigated the architecture by applying all possible four merge alternatives. As shown in figure 5.15, the optimum performance can be observed with the Con function over train and test data(minimum loss function value). Add merge mode also achieve near equal performance to the Con merge mode. The architecture with the Mul merge function has observed the highest loss amongst all four.

5.2.2 Performance with Regularization Techniques Employed:

Table 5.2 represents the mean square value at 100, 200, and 300 epochs. The results show that the MSE value decreases with the increase of the number of epochs for the train data. The epoch denotes the total number of scans throughout the whole sample space. It is anticipated and obvious that the MSE (loss function) decreases with the increase of epochs, and it becomes stable at a specific point. The same performance and behaviour were also demonstrated for the train data, but the same is not observed for validation data (test data). The in-depth performance of the model is denoted in figure 5.16 by gathering and plotting MSE values after each epoch for a particular sample space for better understanding for train and test data of PM 2.5 time-series data. The figure shows that initially, with fast convergence and after obtaining the lowest value of loss function, the performance starts degrading with the increase of epochs for validation. The behaviour depicted is due to the issue of overfitting.

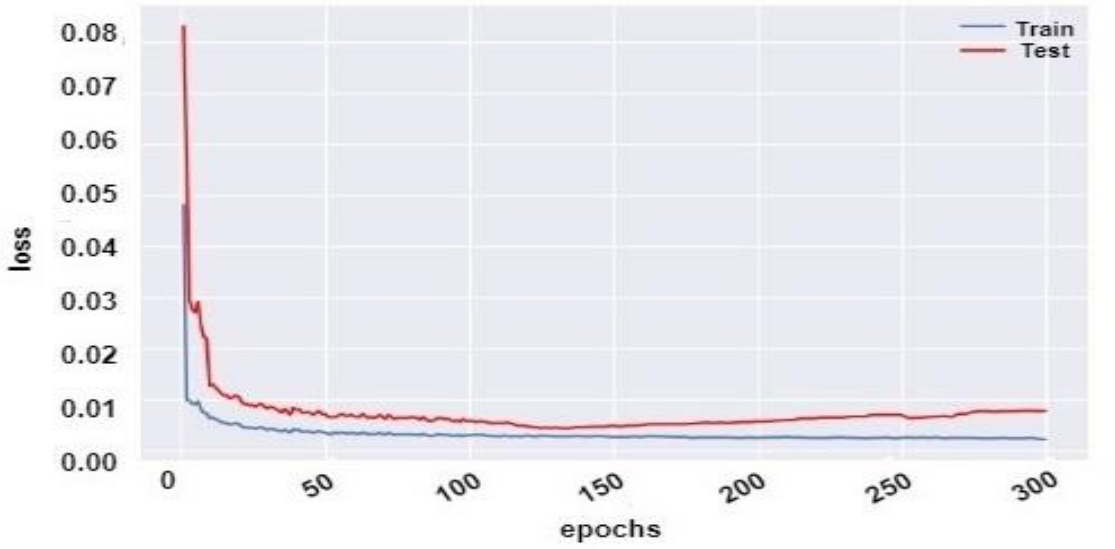
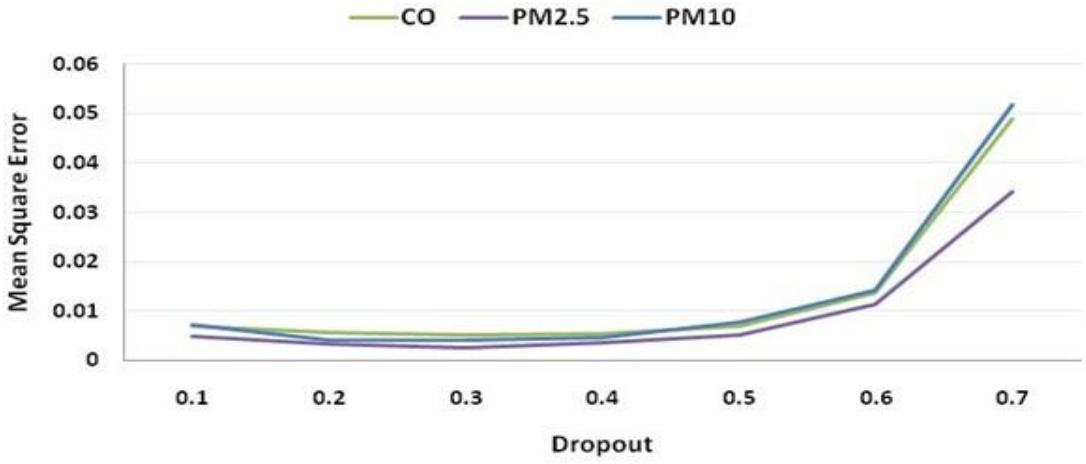
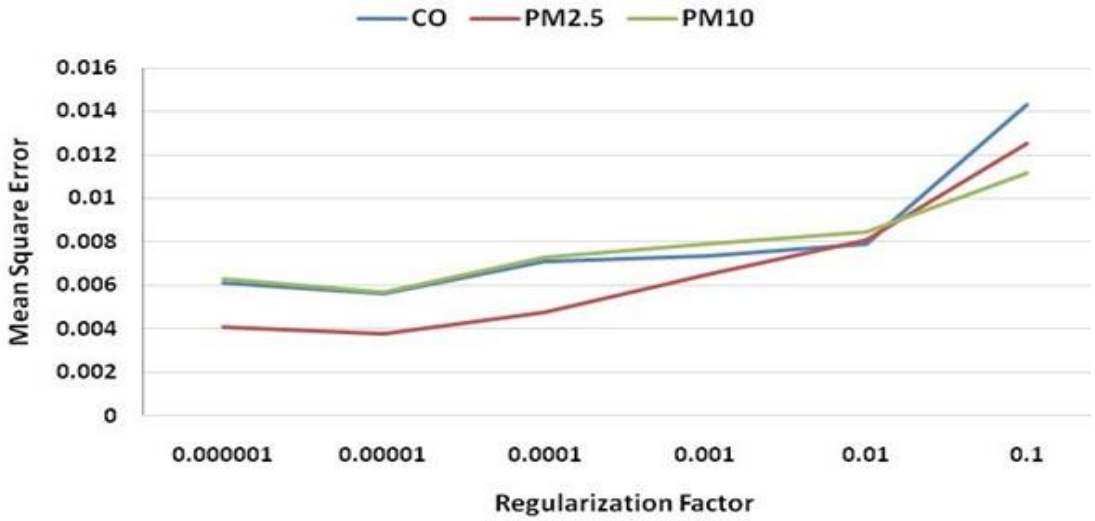


Figure 5.16: Plotting of MSE per every epoch for training and validation

In the domain of deep neural networks, the researchers present various regularization methods to overcome the issue of overfitting. The dropout approach is one of the regularization techniques utilized to prevent the model performance from overfitting. The dropout regularization method is implemented by keeping the leaving edges of hidden units in the hidden layer to zero at every update of the training phase [143]. Keras employs the dropout technique with the use of dropout layers. Dropout layers are added in between hidden layers. Input and recurrent edges (connections to LSTM units) are omitted from activation with the provided probability. The addition of dropout layers provides the environment with many networks having a very dynamic structure in parallel. Also, due to dropout, a neural network can never rely on any input node because every node has the probability of being removed. The overall effect is that; the neural network will not allocate any high weight to a particular feature. The probability setting for getting optimum performance is again hyperparameter which is required to be set by experiments. Figure 5.17 (a) represents all values of dropout applied during experiments, which are plotted against the MSE observed for specific dropout values. The employed dropout value varies in the range of 0 to 1. The lowest MSE is obtained at 0.3 dropout value during experiments, and also it can be seen from figure 5.17 (a) that after 0.5 dropout value, there is a speedy increase in the MSE function. A dropout value of 0.3 indicates the 30 percent of probability of node removal during training in Keras.



(a)



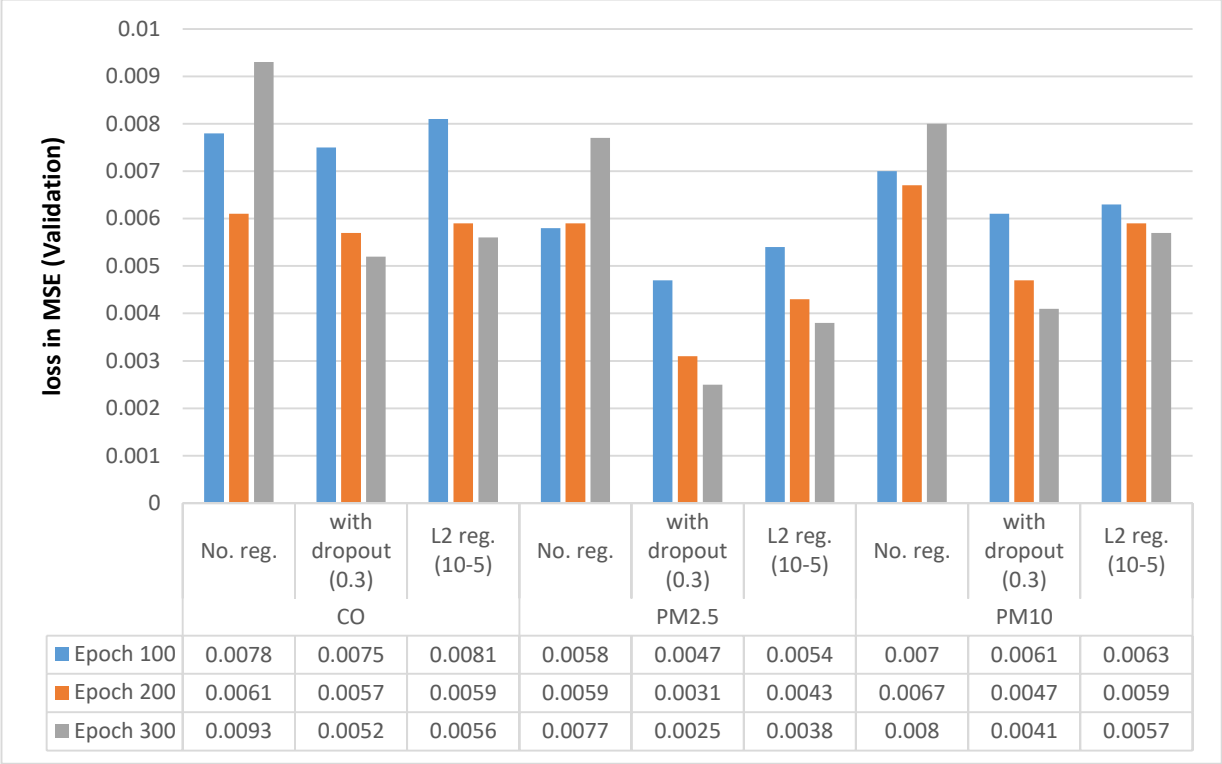
(b)

Figure 5.17: Performance of the model for (a) various values of dropout parameter under dropout technique (b) various values of lambda or regularization factor under L2 regularization

Another approach for regularization is utilizing weight decay, also termed L2 regularization [144]. The neural network always attempts to decrease the cost function by modification of weights and biases. For the L2 regularization method, a factor is added that penalizes the large weights. The factor or component is added to the cost function. The addition of the factor leads the overall weight matric values down, which, in turn, decreases the activation function effect. As an overall effect, the relatively less complex activation function may fit the observations, which assists in overfitting reduction. The component of the factor added can be given using the following equation.

$$New\ Minimization\ Goal\ = L(W, B) + \lambda \|W\|^2 \quad (5.3)$$

Lambda(λ), in the added component here, is the regularization or tuning parameter that balances the trade-off between a low value of weights and low training loss. Lambda is also the hyperparameter that is required to be optimized by experiments. We applied the initial value of lambda provided as an argument to the L2 regularization in Keras, beginning from 10^{-1} to 10^{-6}. Figure 5.17 (b) represents all the employed values of lambda plotted against the MSE value observe for that specific lambda value (regularization factor). The figure shows that the minimum MSE observed for the lambda value of 10^{-5}.



	CO			PM2.5			PM10		
	No. reg.	with dropout (0.3)	L2 reg. (10-5)	No. reg.	with dropout (0.3)	L2 reg. (10-5)	No. reg.	with dropout (0.3)	L2 reg. (10-5)
Epoch 100	0.0078	0.0075	0.0081	0.0058	0.0047	0.0054	0.007	0.0061	0.0063
Epoch 200	0.0061	0.0057	0.0059	0.0059	0.0031	0.0043	0.0067	0.0047	0.0059
Epoch 300	0.0093	0.0052	0.0056	0.0077	0.0025	0.0038	0.008	0.0041	0.0057

Figure. 5.18. MSE comparison of the proposed model(FBLSTM) under regularization techniques for validation data

Figure 5.18 represents the MSE value observed for the three time-series air pollutants data at the end 100, 200, and 300 epochs for validation or test data. It shows the comparison of the MSE values for the dropout value of 0.3 and the lambda value of $10^{-5,}$ which is found to be achieving minimum loss during the respective regularization method application. The figure shows that the dropout-based regularization method performs better than the weight decay(L2) regularization method and is more appropriate for our model. The dropout regularization method gains the stable converse and loss in MSE with values 0.0052, 0.0025, and 0.0041 for CO, PM 2.5, and PM 10, respectively.

5.2.3 Performance with the Self-Attention Mechanism:

Table 5.3: MSE comparison of FBLSTM with attention and without attention after 300 epochs for various time horizons

	Simple FB-LSTM	With self-atten.	Simple FB-LSTM	With self–atten.	Simple FB-LSTM	With self-atten.	Simple FB-LSTM	With self-atten.
	(Tx)		(4Tx)		(8Tx)		(12Tx)	
CO	0.0051	0.0047	0.0055	0.0049	0.0073	0.0053	0.011	0.006
Rate	-	-	7.84	4.26	32.73	8.16	50.68	13.21
PM2.5	0.0025	0.0023	0.0028	0.0025	0.0044	0.0029	0.0083	0.0034
Rate	-	-	12.00	8.70	57.14	16.00	88.64	17.24
PM10	0.0041	0.0036	0.0045	0.0039	0.0066	0.0044	0.0118	0.0052
Rate	-	-	9.76	8.33	46.67	12.82	78.79	18.18

Table 5.4: MSE comparison of FBLSTM with attention and without attention after 300 epochs for various input windows

	Input Window size	CO	Rate	PM2.5	Rate	PM10	Rate
FBLSTM	60	0.0055	-	0.0035	-	0.0046	-
FBLSTM + Attention		0.0036	-	0.0021	-	0.0034	-
FBLSTM	80	0.0084	52.73	0.0057	62.86	0.0074	60.87
FBLSTM + Attention		0.0045	25.00	0.0029	38.10	0.0045	32.35
FBLSTM	120	0.0223	165.48	0.0191	235.09	0.0208	181.08
FBLSTM + Attention		0.0086	91.11	0.0061	110.34	0.0091	102.22

As discussed in subsection 4.2.2, the self-attention mechanism was also applied and tested during the experiments. The Self-attention layer is kept as the last layer in the model(FBLSTM) shown in figure 4.6. The output of the self-attention layer is given as input to the final dense layer for prediction. To understand the effect on loss function and improvement to the existing model, we analyse the self-attention mechanism with two dimensions; time horizon and input window size or input lag. While increasing the time

horizon, the sequence size is kept of the same length. By keeping the same sequence size and increase in time horizon, employed recorded parameter samples in training realize more fluctuations than the small-time horizon. Table 5.3 compares the loss function value (mean squared error) obtained for the FBLSTM (with two hidden layers), without attention, and with attention mechanism for the three air quality parameters. The table shows the effect on MSE value with the increase in the time horizon. The first two rows in the table depict the MSE value for T_x (the basic time step in the input sequence) 90 seconds. The time horizon increment further is obtained by aggregating the recorded value for the basic time step, i.e., $4T_x$ horizon is the aggregated value over 360 seconds, and so on. The rate column in the table shows the percentage of increase in MSE value with the increase of time horizon from the previous one. It can be seen from the table that with the extension of the time horizon, the rate of increase in MSE (compared to the previous horizon) remains small for the model with an attention mechanism. The high rate of increase in MSE represents the rapid reduction in prediction performance with the extension in the horizon. It can be seen that initially, there is not much difference between the performance of the two models. Still, with a higher time horizon, the attention mechanism model performs substantially better than the one without attention. Table 5.4 shows the performance of the two models with the increase in input window size over recorded air quality parameters observations of a single day. The MSE value and rate of increase in MSE are listed for an input window size of 60, 80, and 120. The table indicates that the model with the attention mechanism realizes lower MSE and a slow rate of increase in MSE for larger input window size. Thus the table depicts self-attention mechanism provides better performance for longer sequences.