4. PROPOSED SYSTEM ARCHITECTURE AND METHODOLOGY

This chapter briefly introduces the proposed work that has been carried out in two sub-sections. The first sub section discusses the proposed standardization and layered architecture, details of sensors and sensing unit utilized, and the functioning and implementation of the IoT-based air quality monitoring system. The next sub section gives the details of the proposed methodology and model for prediction using deep learning. The details of the various tools used to carry out the proposed work are given at the end.

4.1 AIR QUALITY PARAMETERS MONITORING USING IOT

In the proposed work, we have standardized the IoT system for air quality monitoring purposes, and layered architecture is proposed for the IoT-based monitoring system. The proposed system connects the smart node comprises of ESP8266 12E microcontroller, equipped with various pollution sensors such as ZE07-CO, DHT22, and SDS021 via the MQTT public broker. In the proposed research, the DHT22 digital sensor is used to calculate relative humidity and temperature, ZE07-CO (electrochemical-based sensor) is used to measure the carbon monoxide parameter, and the SDS 021 sensor module is used for PM 10 and PM 2.5 measurement purposes. Moreover, customized Arduino libraries have been developed to interconnect the proposed IoT layered design. The proposed system design is an economical alternative as compared to legacy systems. The system decreases power consumption with switching of a sensing node in 5 different modes by providing the soft solution.

Moreover, the system is implemented with a novel transmission scheme to reduce power usage further. The proposed air quality monitoring system is also evaluated for quality of service levels and accuracy focusing on the reliable message delivery under the proposed topology and communication protocol. The proposed system also provides a data logging facility, which can be utilized for further analytics**.**

4.1.1 Proposed Standardization and Layered Architecture

Following is the layered architecture of the proposed complete Air Quality Monitoring framework starting from parameter collection from the remote site to the logging and visualization at the central place.

Figure 4.1 represents the layered architecture considering the five layer standards of IoT application and figure 4.2 represents the topology of the proposed system. The layered architecture cab be categorized as per the followings:

1. Physical/Sensing Layer

This layer in the proposed architecture consists of sensor modules for measuring particulate matter 10 and 2.5, carbon monoxide, temperature, and relative humidity parameters from the surrounding environment at the deployment site. The SDS021 sensor for Particulate Matter, ZE07 CO sensor for measurement of carbon monoxide, and DTH22 for temperature and relative humidity measurement are used in the sensing layer. These sensor modules interfaced with the microcontroller forms the smart node. The TheESP8266 12E microcontroller, furnished with IEEE 802.11 and full TCP/IP stack, is utilized for data collection and processing from the connected sensor modules.

The coding is done in Arduino IDE by adding wrapper libraries for ESP8266 12E. The sensing parameter data from sensors (PM 10, PM 2.5, CO) are fetched from UART data frame format sensors. The customized Arduino libraries are developed to read the sensors' data by interpreting the datasheets from sensor manufacturers. The UART data are converted to the absolute value of measurement, and these measurement values are represented as string values.

2. Communication and Networking Layer

The communication and networking layer in the proposed architecture is accountable for providing the interfaces between the physical sensing layer and the cloud MQTT broker. For communicating the sensed data using the application layer protocol – MQTT over the internet, the Wi-Fi access point is a must in the proposed architecture.

We utilized the ESP8266WiFi library for connection establishment over Wi-Fi using the APIs provided in the library. The connection of the smart node to the internet-enabled device with a Wi-Fi interface empowers the smart node to implement the MQTT protocol. We used the PubSubClient library for the MQTT implementation using APIs provided in the library. The data at this layer is now available in the form of MQTT messages, containing the absolute string values of measurement of sensor parameters.

3. Cloud Service Layer

This layer stores the data transmitted from the smart sensor nodes. The smart node consisted of a controller, and sensors forms the smart node. The smart node publishes the MQTT messages containing the sensor parameters. The sensing node can be recognized as an MQTT publisher. The publisher (smart node) publishes the data from the sensing remote site to the cloud MQTT broker. The publisher publishes the various sensing parameters to the specific topic created for that sensing parameter. The cloud service layer is also accountable for the authentication of devices based on unique clientID. The node registers this client ID during the initialization process.

The data at the cloud MQTT server is available in the form of the site-specific topic/subtopic – value hierarchy, which the subscriber can retrieve.

4. Data Processing Layer

The data processing layer is accountable for retrieving the data from the cloud broker, processing these raw data of messages, and storing the processed data. The data is stored in a format compatible with application layer necessity and appropriate for the analysis. The MQTT subscriber is deployed at the server with the python script. The subscriber is responsible for retrieving the observed data from the remote site available at the cloud broker. Air quality parameters fetched from the cloud broker are conveyed for rendering to GUI and also logged in the database for further analytics.

5. Application Layer

The application layer provides real-time air quality parameters updates using the mobile GUI interface and graphs created through the python script.

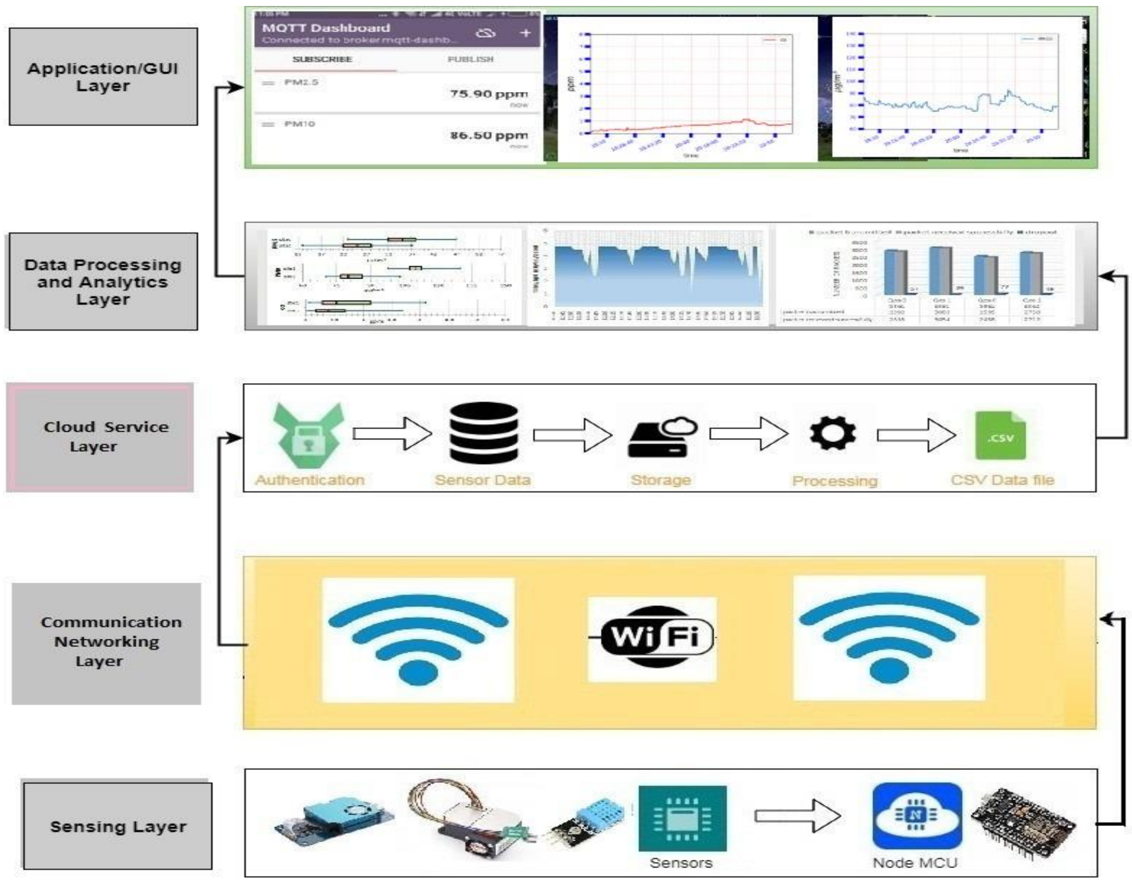


Figure 4.1. Layered Architecture of proposed air quality monitoring system

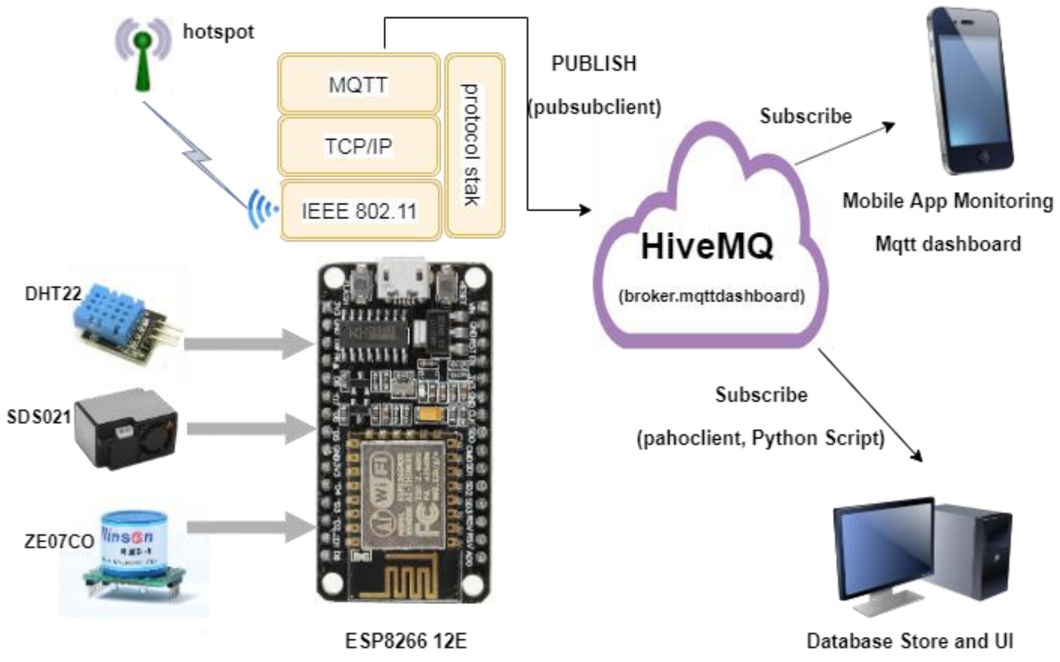


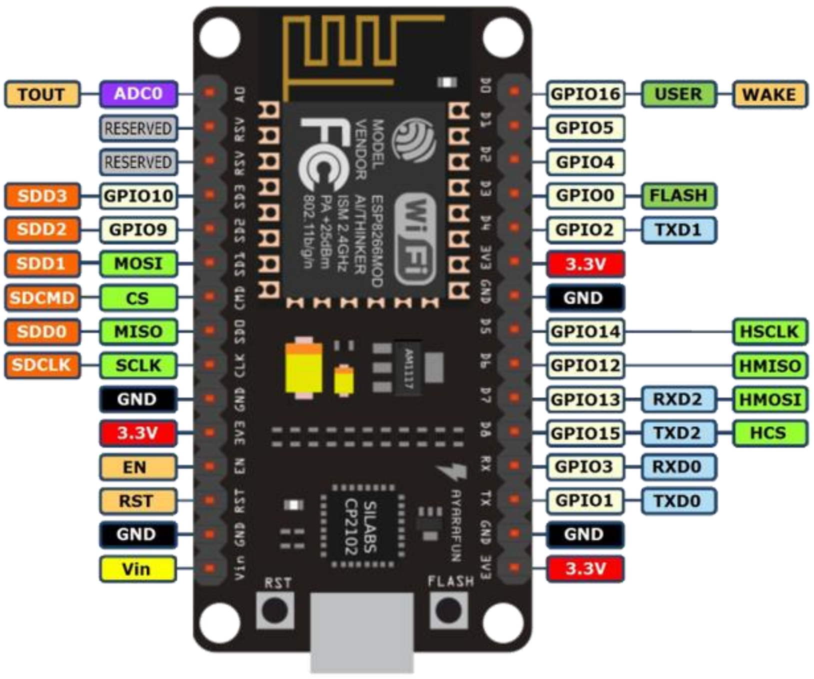
Figure. 4.2. Topology design

Advantages of the proposed layered architecture and underlying topology

- The proposed system implements a lightweight communication protocol with the employment of compatible devices to transmit sensed parameters to the remote server without any necessity of building a complex network or using relaying nodes (i.e., in the case of ZigBee-based implementation).
- Cloud broker HiveMQ is utilized for MQTT communication protocol implementation rather than any local broker. So, the system can be scaled easily.
- The proposed system is a cost-effective alternate comparing to the other legacy systems.
- The system is implemented using parameter-specific sensors(integration) for accurate measurement rather than general-purpose sensors sensitive to multiple pollutants or gases.
- HiveMQ broker supports authentication of publisher or subscriber node with the use of unique ClientID.
- The physical layer with the customized Arduino library provides a physical level integrity check for the sensed parameters while retrieving them from sensors.
- Once the sensed data is available at the broker from deployment sites, the HiveMQ MQTT broker will inform and deliver the sensed parameters to every subscriber registered for the relevant topic.
- The data (site-specific) transfer and distribution from the sensing node to the relevant subscribers are managed by the HiveMQ MQTT broker.
- The complete framework beginning from air quality parameter fetching from deployment sites to the logging and rendering of the air quality parameters on GUI with desktop and mobile platforms at the same time by providing semantic interoperability between various objects (individual parameter monitoring and logging all parameters with timestamp-topic based screening)

4.1.2 Details of the Sensing Unit (sensors and controller)

1. ESP8266 12EMicrocontroller



We utilized the ESP8266 12E(NodeMCU) microcontroller unit to build the smart node, an open-source and cost-effective economic development platform. ESP8266 12E module is using a 32-bit controller. The microcontroller is available with 4MB of memory that can store sensor reading programs and other required APIs libraries. The “ESP8266 12E” refers to the firmware rather than a device kit. The controller requires a 3.3V power backup for functioning. The development board comprises GPIOs, I2C, UART, ADC, and PWM interfaces, enabling the board to interface with a range of application-related sensors. The support is available for developing various IoT applications using the Lua scripting language using ESplorer IDE. However, the programming can also be done on Arduino IDE and using Arduino language with the wrappers added in the Arduino IDE. The ESP8266 12Eboard is embedded with the Wi-Fi capacity (IEEE 802.11 /b/g/n) and a complete stack of TCP/IP.

A variety of processing boards and platforms for IoT application development are available. The selection of the processing unit is affected by intrinsic restriction or condition of the application domain and available budget. Specifically, in air quality monitoring systems, the cost is a significant aspect considering the necessity of the vast deployment of sensor nodes. Table 4.1 shows the comparative analysis of various IoT boards available for IoT application development purposes. The Arduino and ESP8266 12E/NodeMCU are the most economical alternatives. However, both of the processing units comprise the microcontroller. The controllers have comparatively less memory for processing than the other options. The sensing nodes are not involved in any edge computing task (analytics at data origin). If that is the case, the sensing node requires more processing power. In the proposed system, the task of the smart node is to read sensing parameters at the decided frequency and transmit them over IEEE 802.11.

Table 4.1. Existing processing units for building sensing layer of air quality monitoring system using IoT

Product	**CPU**	**Cost (₹)**	**Weig ht**	**Memory**	**Vol.**	**Analog pin**	**Digital I/O pins**	**Operating System**	**Lang. for prog.**
Raspberry -Pi III	Broadcom 64-bit Micro Processor Quad-Core with 1.2 GHz	3200	42 g	1 GB	5 v	0	14	Raspbian,Ubuntu, ArhLinux, Free BSD, Fedora, RIC OS	Linux Angstrom
Arduino Uno	ATMEGA 328microcontroller	700	25 g	2 kB	5 v	6	14	-	Arduino
Arduino Due	Atmel SAM3X832-bit ARM cortex microcontroller	1500	30 g	96 kB	3.3 v	12	54	-	Arduino
ESP8266 12E	32-bit micro-controller with Integrated WI-Fi SOC	400	8 g	4 MB	3.3-5 v	1	17	XTOS	Lua scripting, Arduino
BeagleBone Black	AM335x,1GHz ARM Cortex A8 processor	6000	40 g	512 MB	3.3 v	6	14	Linux Angstrom	VB, VB.NET, C#,C++, Flash/Flex, Java, LabVIEW, Matlab, Action Script 3.0, Cocoa
Udoo (Quad)	Freescale MX6Quad, 4 x ARM® Cortex™-A9 core @ 1GHzprocessorAtmel SAM3X8e 32-bit ARM cortex controller	9000	150 g	1 GB	5 v	14	62+14	Ubuntu, Android, Linux, ArchLinux	Arduino, C, C++, Java
Intel Galileo Gen 2	Intel Quark ™ X1000 32-bit 400 MHz	7000	370 g	256 MB	7-15 v	6	14	Yocto 1.Poky Linux	C,C++,Python, Node.js, Linux

2. SDS021 Particulate Matter Sensor

The SDS021 sensor module has been utilized in the conducted experiments. Nova Fitness Co. Ltd manufactures it. The sensor operates on the laser scattering principle. The sensor begins operation when the particle travels through the detection area. During this process, a scrambled light equipped in a sensor module is transformed into a signal, which will be later amplified and processed to estimate the particles of size 2.5µm and 10µm. The sensor module comprises a fan that provides a continuous stream of wind over the area of detection. The SDS021 module provides the output of measurement over the UART communication protocol. The sensor works with a bit rate of 9600 bits/second. The Sensor detects particulate matter in the range from 0.0 to 999.9 µg/m^3. The sensor operates at 5V voltage, [Vin] of the microcontroller is enough for the necessity of the current. The following table shows the technical specification of the sensor module [128].

Table 4.2. Specification of SDS021

Features	**Technical Specification**
Targeted pollutant to	PM2.5 and PM10
Range of measurement	0.0 – 999.9 µg/m^3
Response time	<10 s
Voltage	5V
Current	60 mA
Current in Sleep mode	< 4 mA
Range of Temperature	-20 - + 60 Celsius
Min. Resolution	< 0.3 µm
Output data protocol	UART

3. ZE07-CO carbon monoxide sensor

ZE07-CO sensor module is manufactured by winsen electronics co. Ltd. The module operates on the electrochemical principle while calculating the carbon monoxide parameter. The module operates at 5Vworking voltage and provides data with the 9600-bit rate over the UART communication protocol. The sensor measures carbon monoxide in the range of 0 to 500 ppm. Following is the technical specification of the sensor [127].

Table 4.3. Specification of ZE07 CO

Features	Technical Specification
Targeted gas to measure	Carbon monoxide
Range of measurement	0 – 500 ppm
Response time	<= 60 s
Warm up time	<= 3 minutes
Voltage	5V
Temperature	-10 to +55 Celsius
Resolution	0.1 ppm
Output data protocol	UART and DAC

4. DHT22 sensor

DHT22 is one of the economic digital temperature and relative humidity sensors. DHT22 utilizes the thermistor and capacitive humidity approach for monitoring the ambient air. The sensor provides the calculated parameter value in the form of the calibrated digital signal as the output on the data pin. The sensor requires a power backup of 3 to 5V.

4.1.3 Detailed Functioning and Implementation

Figure 4.3 represents the detailed working flow of the proposed IoT-based Air Quality Monitoring system. Furthermore, in the publish-subscribe communication pattern, the publisher sends messages to the broker, and the subscriber retrieves messages from the broker.

In the conducted experiments, the HIVEMQ MQTT broker has been utilized as a public cloud broker in the proposed system design and development. The publisher is deployed on the controller ESP8266 12E, which transmits the sensing parameters in the form of MQTT messages to an MQTT broker. The libraries developed for reading data from sensors provide the APIs to the MQTT publisher program for collecting the sensing parameters. The MQTT publisher program uses APIs from the PubSubClient library to implement the MQTT publish task and the ESP2266Wifi library for connecting to the Wi-Fi access point. The programs are developed with Arduino IDE by setting up libraries for ESP8266 12E. Data acquisition at a server-side (MQTT subscriber) has been implemented using python language by using the package “paho-mqtt” and “paho.mqtt.client” libraries. The MQTT subscriber fetches the site-specific data from the HiveMQTT broker, and data are logged in to the excel file or MYSQL server for further analysis. The python script was also implemented to generate the graph from the logged data to visualize individual parameters. The logging at the server data is also fetched from the MQTT broker in the mobile application for display purposes. The mobile application can register to subscribe and monitor the particular air pollution parameter for the specific monitoring site of choice.

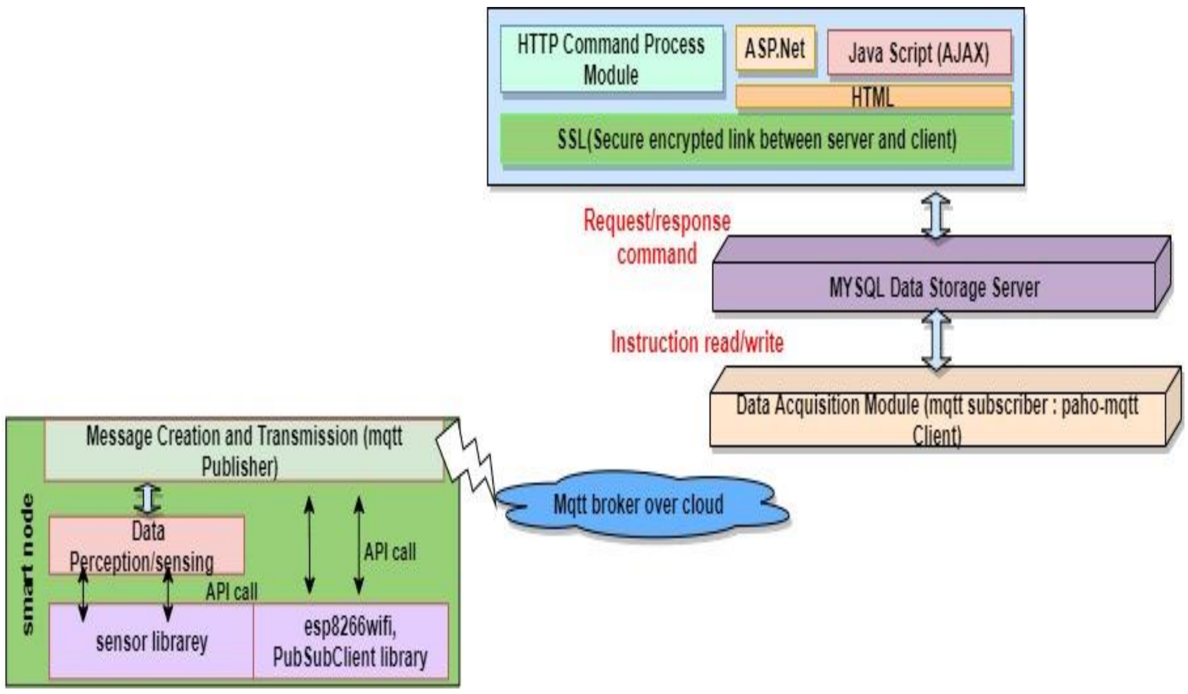

HTTP Command Process Module
ASP.Net
Java Script (AJAX)
HTML
SSL(Secure encrypted link between server and client)
Request/response command
MYSQL Data Storage Server
Instruction read/write
Data Acquisition Module (mqtt subscriber : paho-mqtt Client)
Mqtt broker over cloud
Message Creation and Transmission (mqtt Publisher)
smart node
Data Perception/sensing
API call
API call
sensor librarey
esp8266wifi, PubSubClient library

Figure 4.3 Detailed working flow of the proposed IoT-based air quality monitoring

SDS021 module gives the measured value in the form of a data frame available on the UART port of the ESP8266 12E. The ZE07CO sensor can give the measured value on UART as well DAC. We utilized the UART data frame for the measurement of carbon monoxide from the sensor. Table 4.4 and Table 4.5 indicate the significance of individual bytes received on the UART port for two sensor modules. Algorithm 1 gives the steps for reading the particulate matter 2.5 and 10. Algorithm 2 gives the steps implemented for reading carbon monoxide values from the sensor. The proposed algorithm also implements the integrity checking of the UART data frame received from the sensor to prevent any error in sampling at the physical level.

Table 4.4. Significance of bytes in UART Data Frame for SDS021

Bytes	**Significance**
Byte 0	Starting Byte (0xFF)
Byte 1	Type of Gas (0x04)
Byte 2	Unit - ppm (0x03)
Byte 3	Number of Decimal
Byte 4	CO Concentration high byte
Byte 5	CO Concentration low byte
Byte 6	Full Range High Byte
Byte 7	Full Range Low Byte
Byte 8	Byte for Checksum

Table 4.5. Significance of byte in UART data frame for ZE07 CO

Bytes	**Significance**
Byte 0	(AA) Message Header
Byte 1	(CO) Command ID
Byte 2	DATA1-PM2.5 low byte
Byte 3	DATA2-PM2.5 high byte
Byte 4	DATA3-PM10 low byte
Byte 5	DATA4-PM10 high byte
Byte 6	DATA5-IDbyte1
Byte 7	DATA6-IDbyte2
Byte 8	Checksum
Byte 9	Message Tail(AB)

Algorithm 1. PM2.5 and PM10 parameter measurement process (with integrity check)

Step 1:	Reading serial port of a sensor, on the accumulation of continuous ten bytes clearing of a local buffer and searching for a message header 0xAA
Step 2:	Once the required header of frame is found in step1, check for the second byte of the data frame, which must be 0xCO to get the reading in μg/m^3 unit.
Step 3:	Read next second byte in μ, third byte in α, fourth byte in β, fifth byte in ϒ.
Step 4:	Read the eighth byte and perform an integrity check as per equation 4.3. If integrity is not maintained, go to step 6; otherwise, go to step 5.
Step 5:	Calculate PM2.5 and PM10 values using equations 4.1 and 4.2.
Step 6:	If integrity is lost, then go to step 1.

$$PM2.5\left(\frac{\mu g}{m^3}\right) = \frac{((\alpha \times 256 + \mu))}{10} \tag{4.1}$$

$$PM10\left(\frac{\mu g}{m^3}\right) = \frac{((\gamma \times 256 + \beta))}{10} \tag{4.2}$$

$$Checksum = \sum_{i=1}^{6} DATA\_BYTE_i \tag{4.3}$$

Algorithm 2.CO parameter measurement process (with integrity check)

Step 1:	Read the serial port of a CO sensor and on availability of start byte 0xFFgo to step 2.
Step 2:	Read the second byte and the third byte if bytes are matched with 0x04(gas type CO) and 0x03(unit is in PPM), then go to step 3; otherwise, go to step 1.
Step 3:	Read next coming the fourth byte in ψ and fifth byte in ω.
Step 4:	Read the eighth byte and check for integrity using equation 4.4. If the match is successful, then go to step 5; otherwise, go to step 6.
Step 5:	Calculate CO values using equation 4.5.
Step 6:	If integrity is lost, then go to step 1 again.

$$Checksum = \left(\sum_{i=1}^{7} DATA\_BYTE_i\right)' + 1 \tag{4.4}$$

$$CO(PPM) = \frac{((\psi \times 256 + \omega))}{10} \tag{4.5}$$

- **Message Queue Telemetry Transport(MQTT) Implementation**

MQTT protocol is the open-source and ISO standard publishes-subscribe kind of lightweight communication protocol introduced by IBM. The protocol works on the underlying TCP/IP transportation protocol. The protocol is designed for connecting with a remote location for constrained devices with high latency where network bandwidth is limited. The design principles are to reduce the bandwidth of the network along with the attempt to ensure reliability.

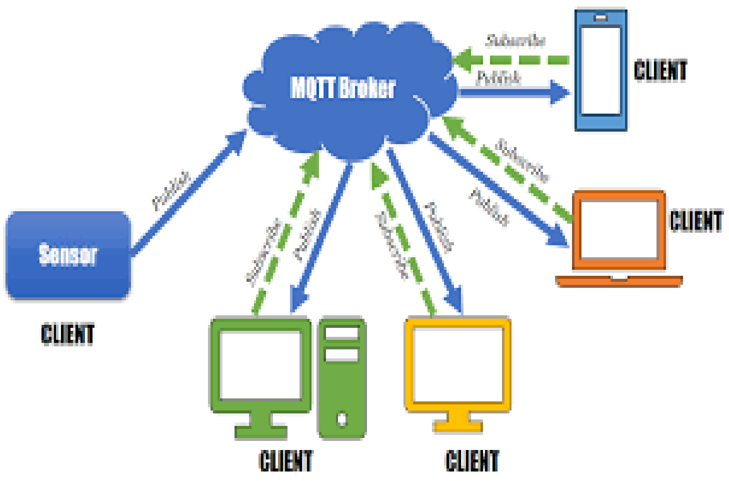
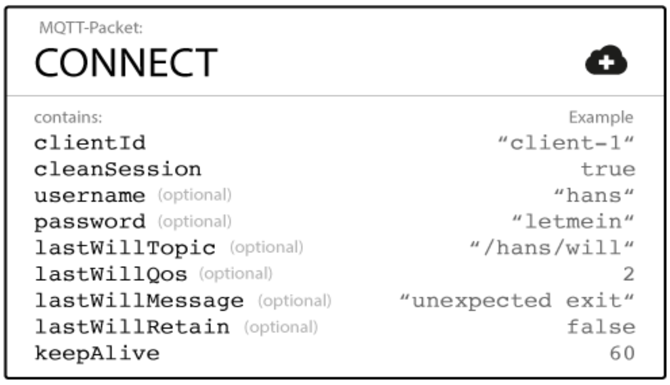


Figure 4.4. MQTT architecture

There are three entities involved: publisher, subscriber, and a broker, in the architecture of the MQTT protocol, as shown in figure 4.4. The publisher is the entity that sends the messages, and the subscriber/client is the entity that receives the messages. The exchange of messages takes place between publisher and subscriber through a third entity known as a broker. Broker is accountable for management of subscription and authentication. The publisher or sender is registered with the broker and transmits the messages associated with a specific topic. Client or subscriber subscribes for such topics at the broker. Broker notifies the client on the availability of the message associated with the topic subscribed. One topic can be subscribed to by multiple subscribers also. Multiple MQTT control packets are exchanged between broker-subscriber and broker-publisher.



MQTT-Packet:

CONNECT

contains:	Example
clientId	“client-1“
cleanSession	true
username (optional)	“hans“
password (optional)	“letmein“
lastWillTopic (optional)	“/hans/will“
lastWillQos (optional)	2
lastWillMessage (optional)	“unexpected exit“
lastWillRetain (optional)	false
keepAlive	60

Figure. 4.5. MQTT connect packet format

MQTT client initiate connection with the broker with connect message. The underlying fields of connecting messages are shown in figure 4.5. The ClientId is an identifier of each MQTT client connecting to the MQTT broker, which is unique per client. IN OUR IMPLEMENTATION, the MQTT client class (MQTTClient_createor MQTTAsync_create inC) sets the client identifier for individual ESP8266 12E. The field username and password in the CONNECT message are utilized for authentication and QoS field for setting the QoS level in the proposed framework. Algorithm 3 is showing the detailed step wise operation of the MQTT publisher with a power consumption optimization scheme implemented. The power consumption scheme that puts the smart node and sensor in sleep mode and its performance are discussed later in the chapter 5. The sensing node registers itself with the MQTT broker during device initialization with a unique client id. The client id is stored at the broker and used to identify the individual node for future communication.

Algorithm 3. MQTT publisher process flow

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.
Step 3:	The publisher creates the MQTT message by allocating the value fetched in step2 to the relevant sub-topic.
Step 4:	The publisher gets connected to the HIVEMQ broker, authenticated using the unique ClientID.
Step 5:	The publisher's MQTT message created in step 3 is published using the topic set for an individual site.

Step 6:	The controller again switches to a light sleep mode.
Step 7:	After the timer gets expired, the controller auto awakes from sleep mode and then goes to step 2.

4.2 AIR QUALITY PARAMETERS PREDICTION USING DEEP LEARNING

We proposed the LSTM model with bidirectional input at the bottom to predict the time series sequences of air pollutants. LSTM based deep learning network model that takes advantage of both forward and backward direction time step observation in learning during training is applied in the proposed work. The observed data are modelled into sequences, and a sliding window-based approach for transforming training data into supervised data is used. The model can predict the next time step value for air quality parameters from the given test sequence. In the field of air quality prediction, one such approach of using bidirectional LSTM [101] is available. Still, the work is done for label classification where the label is various severity categories rather than actual sequence to sequence (moving window based) value prediction. In another approach, authors [103] used bidirectional stacked LSTM during transfer learning from existing station data to the new station for such time series prediction. In our approach, the input layer is only bidirectional, while the stacking is done with unidirectional LSTM layers. The performance of such bi-directional LSTM is heavily influenced by the way forward, and backward layer outputs are merged. The model is critically evaluated with various merging functions(options) to optimize the performance in the proposed work.

Moreover, the stacking model is tested with an incremental approach to decide the best stacking approach for optimized performance. The performance of the deep learning model suffers from an overfitting issue. In the proposed model, the issue is resolved by analysing and implementing two regularization techniques. A proper hyperparameter setting also fine-tunes the model to improve the performance. The proposed work also implements a self-attention mechanism which shows better performance by minimizing the loss function further. The applied self-attention mechanism is also one of the first attempts to the best of our knowledge in the field of sequence to sequence air quality prediction.

4.2.1 Proposed LSTM based Neural Network (FBLSTM: bidirectional and stacking) Model

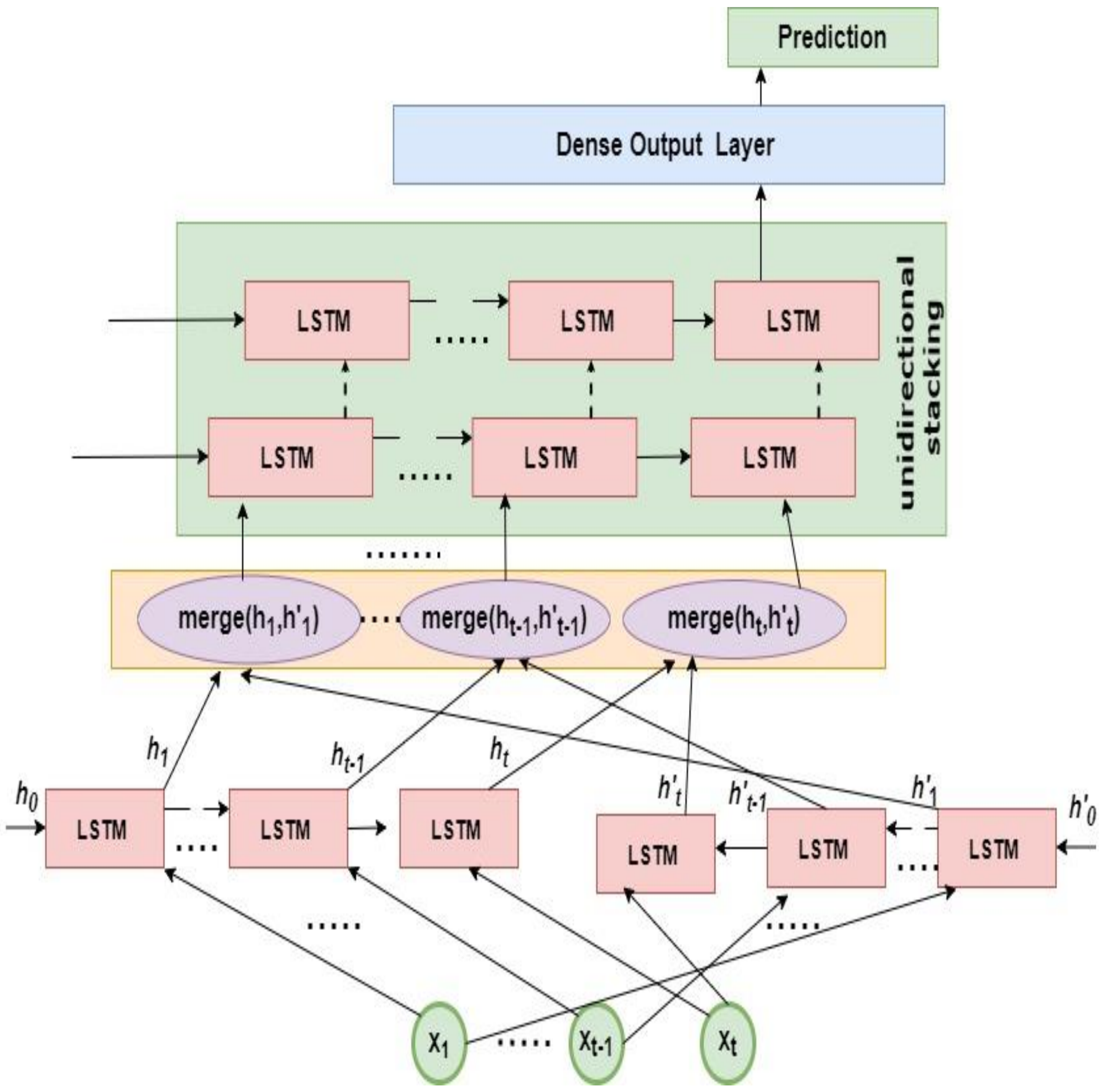


Figure 4.6. Proposed LSTM based model

The bidirectional recurrent neural network was introduced by Schuster in 1997[132] first time. The bidirectional RNN works on the time series sequences input in two directions: forward and backward. Two distinct layers of RNN process the two-pass in two directions. Graves introduced the Bidirectional LSTM neural network in the year 2005. This work using Bidirectional LSTM was proposed in the field of signal processing. The work proposed two distinct and separate forward as well backward processing layers [133,134]. We proposed the model in which sequences are processed in two directions, as shown in figure 4.6. The LSTM layer for forward pass processes the sequences forwarding direction T-1 to T+1(timesteps) and calculates the output h iteratively. The backward LSTM pass layer calculates the h' vector iteratively by processing the same sequence in inverse direction T+1 to T-1 (timesteps). The output of both layers is calculated by using equations for the standard LSTM cell discussed earlier in the section. The performance of bi-directional LSTM is majorly affected by the parameter of merging two layers (how the output of two layers is merged). The proposed model is critically experimented with and tested with different merging functions(alternatives) to select the best alternative providing the optimized performance.

Authors [135-137] have shown that constructing a deep neural network by stacking recurrent neural network layers on top of the other provides improved results. The stacking approach allows the individual hidden state at each level to work at different timescales, and each layer delivers its abstraction level. In the proposed methodology, the input layer of the model is only bidirectional LSTM based processing while further stacking is applied with unidirectional layers of LSTM. Several stacked unidirectional LSTM layers are added on top of each other. The output generated from one such layer is provided as input to the upcoming layer. The bottom layer uses forward and backward pass for learning the features in the input time series sequences. The layers on the top of the bidirectional layer of the model employ the learned features from the bottom layer for further learning. One of the hyperparameters is the number of stacking unidirectional layers on the top of the bidirectional layer that can provide the optimum performance. This hyperparameter can be decided through experiments only. The proposed training model with forward and backward layer as the first layer and with further unidirectional stacking is shown in figure 4.6. We will refer to the proposed model as the FBLSTM model now onward in the discussion. The stacking model is evaluated with the incremental method to select the best stacking option (number of layers to be stacked).

4.2.2 Attention Mechanism Employed

The attention mechanism in the deep neural network allows the underlying training model to provide more significance to the features that are having more effect on the output. The self-attention mechanism is applied on the input timestep vector of the recurrent neural network layer to focus more on significant timestep values [138,139]. The self-attention mechanism allocates specific weight to each input sample as per the importance or influence of timestep on output. The self-attention mechanism can increase the performance of the deep neural network, as shown by various authors in their time-series data experiments [138-140]. We applied the self-attention to the model shown in figure 4.6. The experiments are performed and evaluated to assess the influence on learning with different dimensions, elaborated later in the results section. In our experiments, We set return_sequence to be true in Keras, so every hidden layer provides one output for individual time steps instead of one output for the whole sequence. The output vector v ={V_1, V_2, V_3, …, Vt } of dimension size 60 (timesteps during experiments) of the very last unidirectional layer in LSTM based stacking is provided as input to the attention mechanism or attention layer. The self-attention mechanism provides the scored weight α_i to each of the v_i depending on the influence on the output. The scored weight αi is calculated as per equations 4.6 and 4.7.

$$\alpha_i = \frac{\exp(e_i)}{\sum_{i=1}^{t} \exp(e_i)} \tag{4.6}$$

$$e_i = fun(W_{i,} v_i\,) \tag{4.7}$$

$$v'_i = \alpha_i * \; v_i \tag{4.8}$$

Here W_i is the weight assigned during the training for each timestep Vi in LSTM learning via backpropagation with time, and fun is the function used for calculating $e_{i,}$ which is tanh function in experiments. The softmax function in Keras is utilized for self-attention that computes the normalized weight αi as per Equation (4.6). The softmax function looks after that the summation of all the weights(αi) is one. The concluding context vector output from the attention layer is v'={v'_1, v'_2, v'_3, …, v'_t} and v'_i can be calculated as per equation 4.8.

4.2.3 Data Preparation

Our IoT-based air quality monitoring system collects Real-time data of air quality parameters for the prediction purpose developed and discusses earlier. The observed sensing parameters of the remote site are logged in at the server periodically, which works as the timestep (90 seconds) of the input time series sequences of air quality parameters in the proposed forecasting model. The IoT-based system provides the air quality parameters sequences of carbon monoxide, and particulate matter 2.5 and 10 are utilized in the proposed prediction model.

To develop the model for forecasting the time series data of pollutants gathered, we must convert these time series data into the appropriate data structure suitable for supervised learning. The methodology used by authors [141] depends on the prediction method that represents the many to many mapping of given inputs to the outputs with preservation of the stochastic dependencies of the time series events. We applied the same methodology in our model for time series experiments. The approach comprises of predicting the upcoming k values using equation 4.9.

$$P(x_t, \ldots\ldots x_{t-l+1}) = (x_{t+k}, \ldots\ldots, x_{t+1}) \tag{4.9}$$

Where *l* represents the total number of previous observations used for prediction of next *k* events and *t* ∈ *{l, ... n − k}*, the right-hand side portion of the equation indicates the observations involved in the target or output window. Thus *k* is indicating the size of the output window. The left-hand side portion of the underlying equation indicates the observations that are part of the input window. The size of the input window is *l*. The moving window or sliding window approach is applied by dividing the time series sequences of air quality pollutants of size *n* into the sample sequences of length *[input window (l) + output window (k)]*. So in total, there are *[n-(input window size +output window size) +1]* such samples in the employed sample space. The employed sample space termed as *X* after segmentation over the time-series with n timesteps can be represented as the following matrix:

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_l & x_{l+1} & \cdots & x_{l+k} \\ x_2 & x_3 & \cdots & x_{l+1} & x_{l+2} & \cdots & x_{l+k+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ x_{n-(l+k)+1} & x_{n-(l+k)+2} & \cdots & x_{n-k} & x_{n-k+1} & \cdots & x_n \end{bmatrix} \tag{4.10}$$

The output window is set to be one size, which means only the next time step is forecasted during experiments. The input window size *l* is set to be 60. The training samples are acquired with the moving input window, as shown in figure 4.7. During the training of the LSTM based networks. The input window of size *l* keeps sliding over the air pollutant time-series sequence until the last window is reached. The output window observation is utilized as a target value for the given input window observation used for learning further with the use of backpropagation through time and for calculation of the error gradient over the employed batch size.

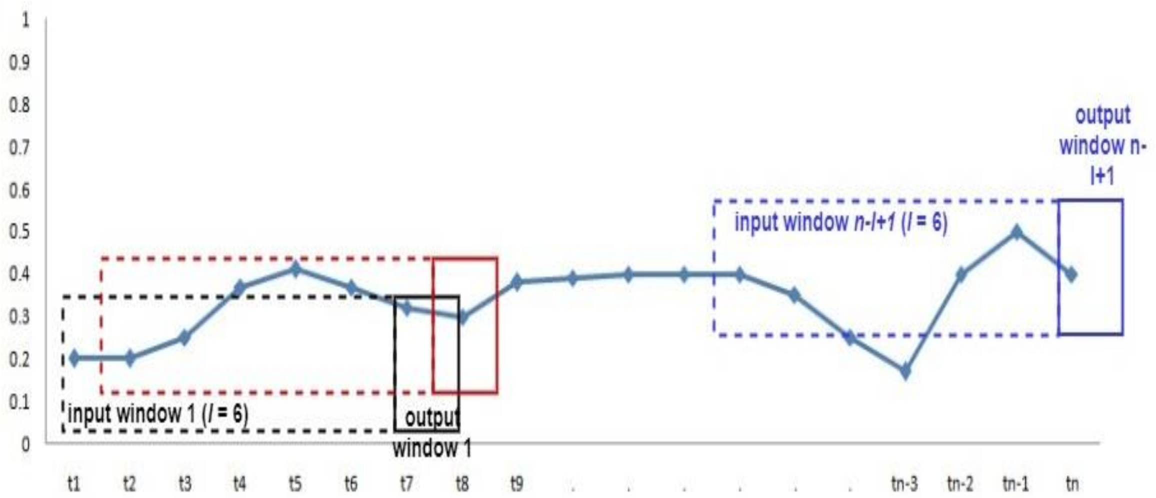


Figure 4.7. Moving window over timesteps in time-series

4.2.4 Preprocessing and Performance Metrics

Figure 4.8 represents the whole framework of the proposed air quality prediction system. During the pre-processing of the data, normalization of the time series data is achieved using linear scaling. The linear scaling can be performed as per equation 4.11. The linear scaling approach converts the time series observations into the new data interval represented by the lower bound *(lb)* and upper bound *(ub).* The newly transformed air pollutant time series data now falls in the range [0,1].

$$X_{i\_new} = lb + \frac{X_i - MIN(X)}{MAX(X) - MIN(X)} * (ub - lb) \tag{4.11}$$

The data preparation subsection shows that the newly transformed training data is modeled into supervised learning data. There are various evaluation metrics MAPE, MSE, and RMSE available that can be used in the performance evaluation of the prediction model. Mean Square Error (MSE) is used as an evaluation metric (MSE loss function in Keras) in the experiments conducted that can be defined as per equation 4.12, where X_{pred_i} is the predicted value X_{actual_i} is the actual value of the ith air quality parameter observation.

$$MSE = \frac{1}{n}\sum_{n=1}^{n} (\mathrm{X}_{\mathrm{pred\_i}} - \mathrm{X}_{\mathrm{actual\_i}})^2 \tag{4.12}$$

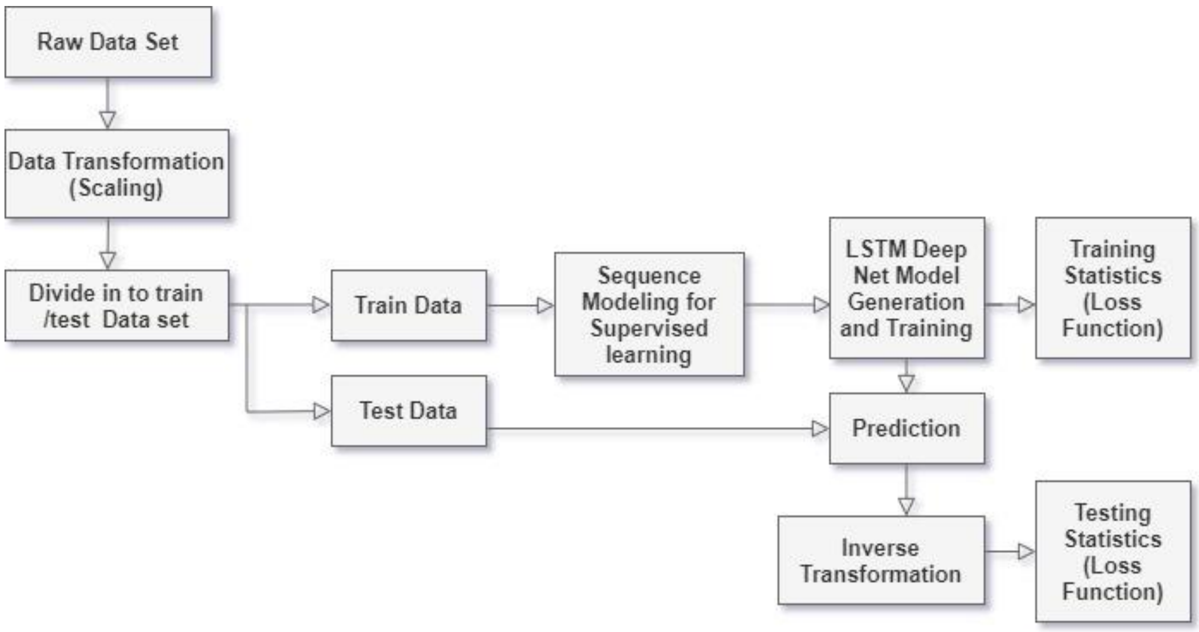

Raw Data Set
Data Transformation (Scaling)
Divide in to train /test Data set
Train Data
Test Data
Sequence Modeling for Supervised learning
LSTM Deep Net Model Generation and Training
Training Statistics (Loss Function)
Prediction
Inverse Transformation
Testing Statistics (Loss Function)

Figure 4.8. Detailed training and testing framework of proposed forecasting task

4.3 TOOLS USED FOR IMPLEMENTATION

IDE used for development with ESP826612E

There are various alternatives available for programming the ESP8266 12E like Lua, MicroPython, or Arduino language. For Lua-based development, the ESPlorer is the IDE to be utilized. Lua scripting language is built over the top of C language. Some general-purpose IDEs like the Pymakr Atom package provide the environment for micro-python-based development on ESP8266 12E. The ESP8266 12E development board could be added to the Arduino IDE by installing necessary libraries in the IDE, which we used for ESP826612E programming purposes. Arduino IDE provides the environment for programming, which lets the developers develop programs and load the programs into the Arduino family of the microcontroller. The language used for programming is also known as Arduino. Once the program or code has been written, the IDE compiles the converts the program to the assembly language. After the translation, the IDE uploads the code to the controller connected to the computer via USB. The IDE is coming with the code parser for checking the code.

The Arduino language is inspired and developed based on the C++ language. The language is fundamentally just a collection or set of c/c++ functions. The setup and loop are the two necessary and compulsory functions to have existed in the valid program of Arduino. The setup function is called first when the controller is powered on at the beginning of the program. The one-time initialization is done in the setup function. The program then keeps running continuously forever inside the loop function.

Python IDE: IDLE

Python language is an object-oriented, interactive, and high-level programming language. It is also a *Programming language for everyone*! It was developed in the late 90s and has recently gained momentum due to its applicability and versatility for machine learning applications.

We used a simple IDLE environment for the IoT server-side implementation of MQTT subscribers using python scripts. The APIs from the Paho MQTT Client are utilized in python script.

Spyder

Python is also used for the implementation of the prediction system based on deep learning. We used Anaconda distribution with the conda virtual environment manager and the Spyder open-source IDE to develop the deep learning model. We used the Keras package and libraries available for deep learning; Keras uses TensorFlow in the backend.

Keras

Keras is a deep learning library based on python language. Keras is the wrapper, and it executes over the top of the other backend open-source libraries such as Theano, TensorFlow, and CNTK - Cognitive Toolkit. TensorFlow is one of the universal and well-accepted math libraries utilized for setting up deep learning and neural networks-based models.

TensorFlow is an extremely dominant library but is hard to understand for developing neural network models. Keras is one of the wrappers dependent on the minimal structure, which avails less complex methods to develop deep learning solutions based on TensorFlow. Keras is developed for the fast implementation of deep learning-based models. So, Keras is one of the optimum options for applications based on deep learning. It uses many optimization methods to avail of easy high-level neural network API with better performance. Keras supports many features as follows:

- Easy, Consistent, and extensible APIs.
- Minimal structure
- Supports various platforms and backends.

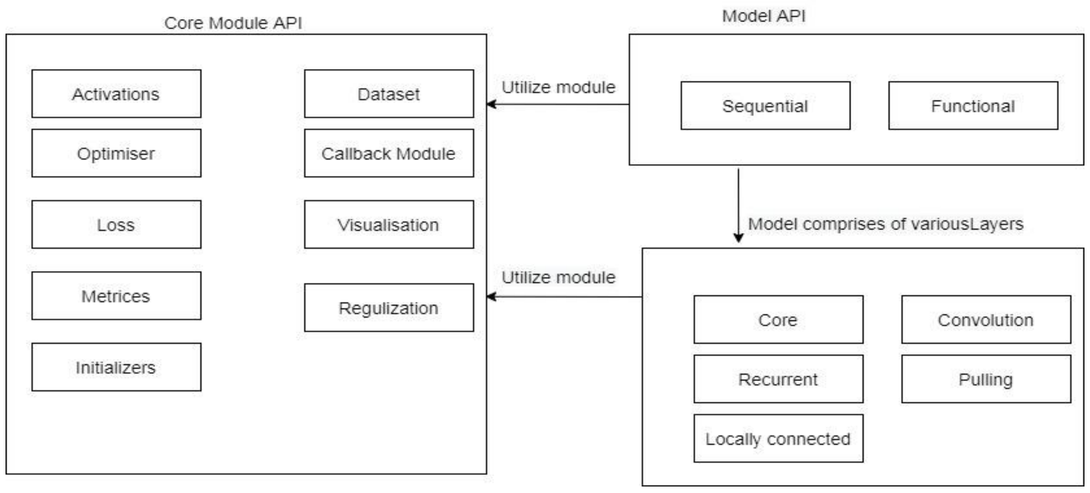
- The framework that executes on CPU and GPU both.
- Extremely scalability of computation.

Keras depends on the following python libraries.

- Numpy
- Pandas
- Scikit-learn
- Matplotlib
- Scipy

Keras APIs can be classified into the following major categories −

- Model
- Layer
- Core Modules



In Keras, each neural network is denoted by Models. The Model is comprised of layers and performs as various artificial neural network layers such as input, hidden, convolution, and output layer. The layer uses various modules for activation, loss calculation, and regularization, etc. With models, Layers, and modules, we can effectively create any deep learning-based architecture. The above figure represents the interdependency and relation between these components.