Cureus Journal of Computer Science

Open Access Original Article

An Internet of Things-Based Weather Forecasting System With Blockchain-Enabled Transmission

Tulsi Pawan Fowdur [1] , Dhaveesh Parbhoo [1]

1. Department of Electrical and Electronic Engineering, University of Mauritius, Reduit, MUS

Corresponding author: Tulsi Pawan Fowdur, p.fowdur@uom.ac.mu

Received 04/28/2025
Review began 05/06/2025
Review ended 07/26/2025
Published 07/28/2025

DOI:
https://doi.org/10.7759/s44389-025-04872-y

Abstract

The purpose of this paper is to develop an Internet of Things-based weather forecasting system with blockchain-enabled transmission, employing deep learning algorithms to perform short-term predictions. The system uses a SparkFun® Meter Kit connected to an Arduino Uno, with data transmitted to a Raspberry Pi, which relayed the data to a Java server via Wi-Fi. The blockchain transmission involves asymmetric encryption using pre-shared keys. Deep learning models such as Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM) were employed for weather forecasting, and predictions were made using data from a local MySQL database. Results obtained showed that the LSTM algorithm outperformed MLP, with a difference of 0.41% and 0.79% in their mean absolute percentage error values for the experiments conducted. The blockchain analysis revealed that larger reading sets increased the block size, while decreasing the total number of generated blocks due to longer processing times. Time complexity, latency and throughput increased with more readings being taken per block. The main novelty of the paper is the development of a real-time weather forecasting system with secure, tamper-proof data transmission that provides a framework for the performance analysis of both weather prediction accuracy and metrics to assess the blockchain performance of the application.

Categories: AI applications, IoT Applications and Use Cases, Machine Learning (ML)
Keywords: iot, ai, blockchain, 4g, lstm, mlp, java

Introduction

Climate change has inevitable effects on the lives of every individual across the globe and remains the planet's greatest existential threat caused by mankind. Its effects range from rising global temperatures and instability in sea levels to drastic changes in atmospheric conditions, causing droughts, hurricanes, and floods. If neglected, these effects will undeniably lead to detrimental outcomes [1].

Statistics demonstrate Mauritius is no exception of flash floods. Mauritius witnessed the most devastating flash flood in March 2013, claiming 11 deaths [2]. A total of 150 houses were affected by flash floods in Cottage and an additional of 50 in Fond du Sac in April 2019 [3]. Coming to the most recent flash floods event, Cyclone Belal, in January 2024, claimed at least one death, and more than 100 vehicles were abandoned by their owners [4]. The main issue, therefore, becomes better preparing the island against flash floods by means of real-time monitoring of the weather conditions as well as predicting accurately such unwanted events.

Blockchain and AI form part of the fifth wave of disruptive computing paradigms, following the Mainframe and PC paradigms, the revolutionary Internet paradigm, and the subsequent Mobile and Social Networking paradigm. Blockchain offers decentralized and reliable automation of a publicly shared ledger of records, transactions, and data to all users, and using smart contracts, it can also govern transactions of users whereby each transaction is completely hashed and verified by every mining node in the blockchain. It makes timestamped records of immutable well-organized and secured data accessible to all peers involved. AI, on the other hand, provides devices with human-level and autonomous reasoning and decision-making capacity using machine learning (ML) and deep learning (DL) algorithms [5]. There is no doubt, therefore, that the combination of blockchain technology and AI offers far-reaching applications in combating climate change by enabling the secure and reliable distribution of large datasets of weather parameters for monitoring, data analytics, ML and DL, all without the need for a centralized authority [6].

The aim of this paper is to develop a system for monitoring weather parameters, securely transmitting the collected data and performing short-term predictions on weather parameters. A data acquisition system, consisting of microcontrollers and sensors, is implemented to send data to a Java server application, where it is visualized using time series plots. Finally, short-term predictions are carried out on the collected data for next instants.

The main research questions (RQ) identified in this paper are as follows:

How to cite this article
Fowdur T, Parbhoo D (July 28, 2025) An Internet of Things-Based Weather Forecasting System With Blockchain-Enabled Transmission. Cureus J Comput Sci 2 : es44389-025-04872-y. DOI https://doi.org/10.7759/s44389-025-04872-y

RQ1. How to implement a blockchain-enabled transmission for weather data to a local server?

RQ2. How to evaluate the blockchain-enabled transmission in terms of time complexity, latency and throughput?

RQ3. What prediction accuracy can be obtained for weather data using DL algorithm?

The novelties of this work include:

• A concise implementation of an IoT-based weather monitoring system with blockchain-enabled transmission.

• An analysis of the performance metrics of the blockchain system in terms of time complexity, latency and throughput.

• An analysis of the performance of the DL algorithms used including LSTM and MLP to perform short-term predictions on weather parameters.

This paper is organized as follows. The Literature Survey section provides details about the related works on weather monitoring and forecasting systems. The Materials and Methods section describes the complete weather monitoring and forecasting system, with a brief explanation on the classes and methods involved for both the data acquisition system and the Server application and the logic behind. Mathematical expressions for the prediction algorithms used, GUI layout of the Server application and the database setup procedures are also provided. The Results and Analysis section explains the complete experimental setup, field testing setup, results obtained for the wind speed, wind direction, rainfall intensity, relative humidity, temperature and barometric pressure parameters, blockchain metrics analysis such as latency, time complexity and throughput. Finally, the Conclusions section summarizes the paper and provides insights on the blockchain metrics analyzed and the LSTM and MLP prediction results obtained in terms of their mean absolute percentage error (MAPE) values.

Literature survey

Muck and Homam [7] implemented a weather station with real-time notifications for weather monitoring, interfacing it with a cloud platform. The research was carried out in three phases: namely, the design and prototype development of the weather station using SparkFun® Weather Shield and Arduino, the transferring data to a cloud platform using Raspberry Pi 3 Model B, and the application of IoT for displaying real-time weather data and sending notifications. The result of the research included the successful assembly of a weather station that collected weather data and stored it in Google Cloud SQL, which can be accessed via a web application. Results obtained show very low accuracy due to low experiment time period. The paper is useful in displaying the expected weather parameter plots and the use of the SparkFun® Weather Station-based data acquisition system in providing the desired weather parameter readings. Other real-time monitoring system is reviewed by Ahire et al. [8].

Bella et al. [9] designed a smart weather station that gathers and sends real-time information on temperature, humidity, wind speed and other data with the use of IoT sensors. Morón et al. [10] also implemented such a station. The collected data will be sent to the IOTA network for secure and decentralized storage. Data here can be used for any desired purpose: weather forecasting, climate research, or management of disasters. The main highlights of the system include real-time monitoring, accessibility at a distance and energy efficiency. The proposed system uses a GISMO microcontroller, similar to ESP32 microcontroller, and a Wi-Fi module for efficient data gathering and transmission. Thing Speak Cloud is used to visualize collected data. This system is cost-effective and user-friendly, making it suitable for individuals and organizations. Results show an experimental percentage error of 6.6%, 2.56%, 6.06% and 0% for the temperature, humidity, pressure and rainfall parameters, respectively, when compared to national information available for Hyderabad City, India. Rao et al. [11] and Viciana and Juan [12] also provide similar methodologies and analysis in IOT-based weather monitoring systems.

De Jesús Medel Juárez et al. [13] aimed to provide access to the meteorological data concerning specific geographical areas that is provided by a mobile data acquisition system within Wi-Fi coverage. The proposed system model used the SparkFun® Weather Shield and its kit, the SparkFun® RedBoard, a microcontroller board similar to Arduino Uno and an Electric Imp® device connected to an Electric Imp® agent. The Electric Imp® device is responsible for relaying the serialized data from the RedBoard to the Electric Imp® device, which in turn injected the data into a cloud database system hosted by a Hostinger server. The cloud system allowed user to access the data from Meteorologic4 website showing graphs with weather parameters. The authors aimed at limiting the experimental uncertainty to 3% of the actual data collected by existing stations.

Sivakumar and Nanjundaswamy [14] developed a smart weather monitoring system using sensors, for

collecting data on temperature, humidity, wind speed and other weather conditions, and the Arduino Uno microcontroller. Further, the collected data is sent to a web page, and an API is used to analyze the data for making predictions. The implemented system also makes use of a GSM module, acting as a web gateway, and sends weather data via SMS. The system is designed to be energy-efficient and low-maintenance. It can be powered by solar panels and is more affordable than other similar devices on the market.

Shah et al. [15] aimed to develop an IoT-based system for real-time weather, traffic and pollution monitoring. By leveraging the Internet of Things (IoT) technology, the researchers aimed to create an integrated system that could communicate data effortlessly, thus minimizing human error and maximizing productivity. Methodology involved using sensors with a Raspberry Pi and AWS for data processing. Results showed 94-95% accuracy, enhancing urban management. Novelties include integrated monitoring and improved data accessibility for smart cities. Studies by Malik et al. [16] and Mohammad et al. [17] show how the integration of blockchain can be beneficial in terms of data security.

Rahman et al. [18] published a review about weather forecasting using ML models. The paper further focused on evaluating the accuracy and performance of these algorithms. The research was carried out using a 20-year dataset extracted from a weather station in Dhaka city, Bangladesh. Various machine learning algorithms, including Gradient Boosting, AdaBoosting, ANN, and so on, were employed. The results obtained corresponded to the algorithm efficiency in terms of accuracy of 92.11%, 92.51%, 92.15% and 91% for Gradient Boosting, Ada Boosting, ANN and Stacking ANN, respectively, with AdaBoost achieving the most accurate score and Stacking KNN the least. Other efficiency parameters such as Precision, Recall and F1 score with values of 89%, 93% and 90% respectively for Ada Boost, being the highest. The findings underscore the potential of machine learning in enhancing weather forecasting accuracy. The paper can serve as a reference point for involving predictive models using machine learning techniques for weather-related conditions.

Kulkarni and Mukhopadhyay [19] investigated the challenges of accurate weather forecasting and the potential of utilizing IoT-enabled systems and hybrid methodologies to improve weather prediction accuracy. The research was carried out by using remote sensing technology to gather and analyze weather data, implementing IoT technology, and creating a low-cost weather display system. The use of deep learning algorithms, such as MLP and LSTM to improve weather prediction accuracy can be beneficial. While the research focuses on a practical IoT-based approach, deep learning algorithms can provide advanced analytical methods for processing and predicting weather parameters through data learned from historical patterns. Similar findings and use cases of mentioned technologies are also included other studies [20-23].

Thapelo [24] introduced the programmable toolkit for creating a smart weather system using the uPyCraft SDK and MicroPython, which gather data from sensors such as DHT11 and BME280 to send to the database developed on NetBeans SDK with SQLite and Java. The RStudio SDK is used for the analysis and forecasting of weather conditions. The K-nearest neighbor algorithm is applied for the prediction of future air temperatures, where the MIMO strategy outperforms the Recursive strategy. This can serve to enhance the capabilities of a weather station to perform tasks beyond the norm and add a touch of more informed decision-making. The collected data can be used for research, education, or even practical use in smart irrigation systems. Results demonstrate on average, the MIMO strategy outperformed Recursive strategy for multi-step ahead forecasting with an average of 74% compared to the latter at 65%. The MIMO strategy has a 95% accuracy rate, while the recursive one has only 67%. Both MIMO and recursive strategies benefited from larger batch sizes, yielding lower mean absolute error (MAE) and higher correlation coefficients (r). The KNN model, especially with K=3 and 62 lags, was able to capture the trend in maximum air temperature. The performance of the two models increased significantly beyond a forecast horizon of 18 hours. Beyond 120 hours, however, performance for the two strategies showed deterioration, due to the intrinsic unpredictability at finer temporal resolutions.

Materials And Methods

Proposed weather forecasting system

Complete Transmission System

This section presents the complete schematic diagram of the blockchain-based transmission system, as shown in Figure *1*. It starts with the data acquisition process from the SparkFun® Weather Kit to the Arduino Uno board and SparkFun® Weather Shield. The processing of serial input and setting up of the APN on the SIM7600X module is performed using AT Commands which uses a SIM card to register the network and an IP address is provided to the Raspberry Pi. The blockchain transmission is then performed followed by the visualization on the Java Server application via real-time series plots. Finally, injection of data into a local MySQL database and predictions using DL algorithms such as LSTM and MLP are performed. An external server is involved, which does not participate in the blockchain system, which receives data from the data acquisition system in unencrypted JSON format.

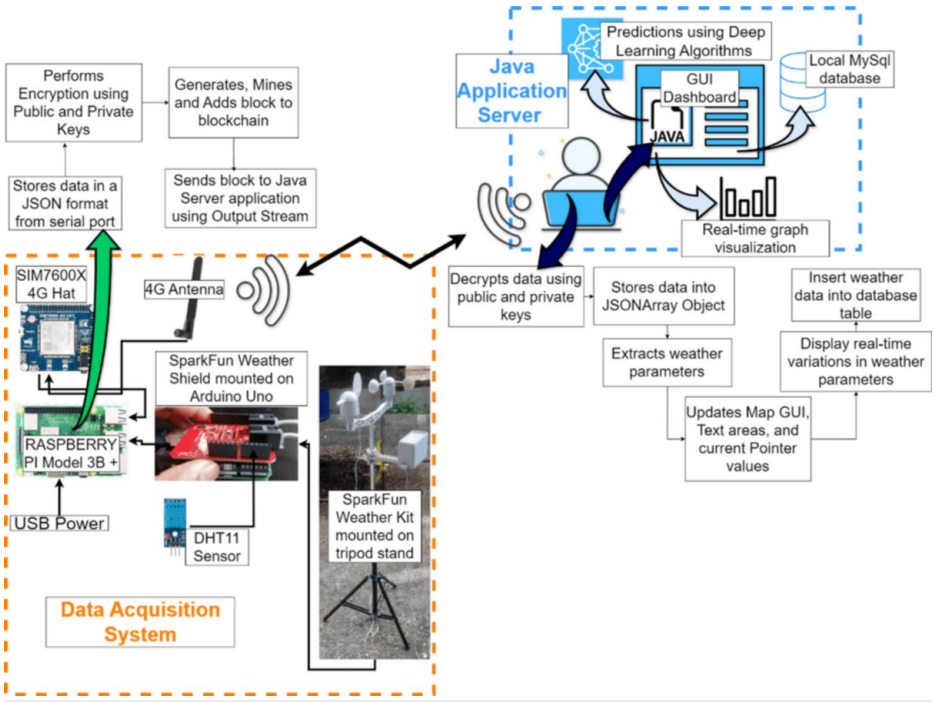


FIGURE 1: Complete schematic system

Microcontroller Setup

Figure *2* shows the connections made by the Arduino Uno board, SparkFun® weather shield, Raspberry Pi, SIM7600X Hat and the DHT-11 sensor. To start, the wind vane and the wind anemometer are connected to the RJ11 female ports on the SparkFun® weather kit shield, which is mounted onto the Arduino Uno board. The DHT-11 sensor is connected to pins 5V, GND and 10 of the SparkFun® weather kit, which links the same to the Arduino Uno board beneath it. Next, the SIM7600X hat is mounted onto the Raspberry Pi with their pins exactly on top of each other. The Arduino Uno board and the SIM7600X module both are connected to the USB 2.0 port of the Raspberry Pi. The detailed pins are given in Table *1*.

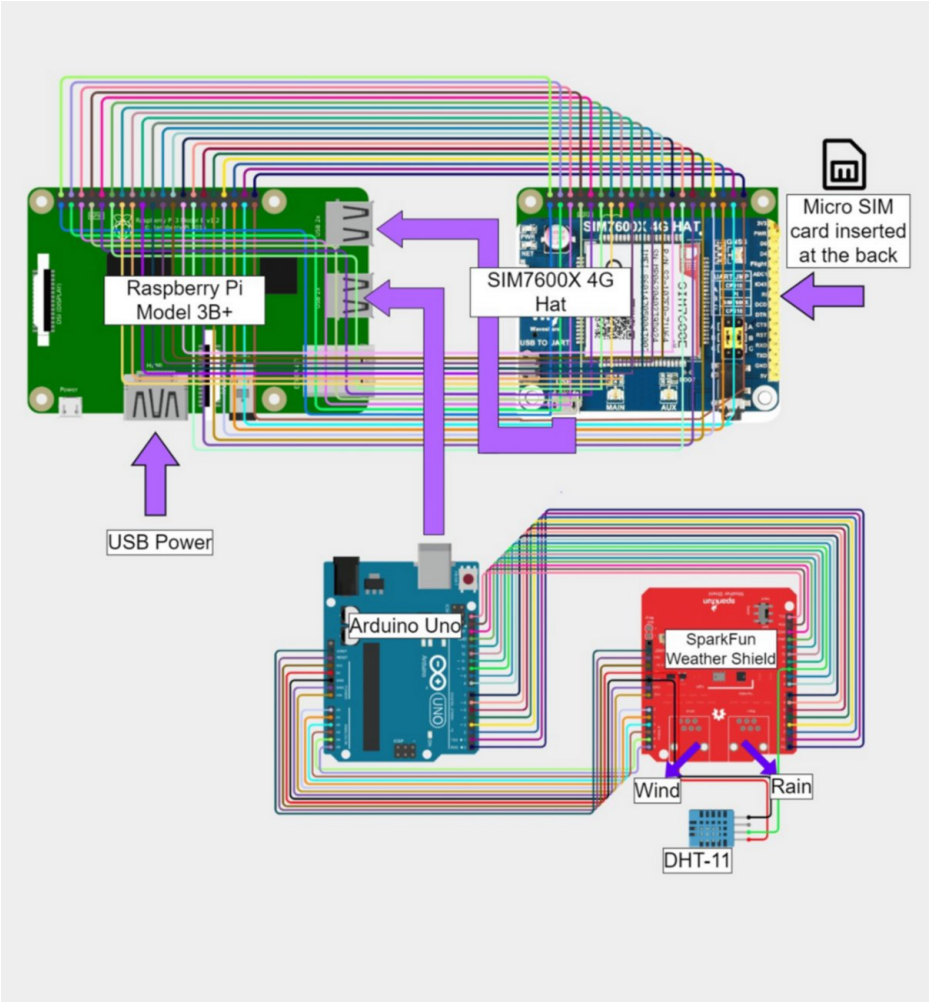


FIGURE 2: Microcontroller connections

Arduino Uno	SparkFun® Weather Shield	DHT11
5V	IOREF (Input/Output Reference Voltage)	
RESET	RST (RESET)	
3.3V	+3.3V	
5V	+5V	+
GND	GND (Ground)	-
GND	GND	
VIN	VIN (Voltage Inpur)	
A0	WDir 0 (Wind Direction)	
A1	Light 1	
A2	Batt Lvl 2 (Battery Level)	
A3	3.3V 3	
A4	SDA 4 (Serial Data Line)	
A5	SCL 5 (Serial Clock Line)	
0	RX (Receiver)	
1	TX (Transmitter)	
2	2 RAIN	
3	3 WSpd (Wind Speed)	
4	4 GTX (GPS Transmit)	
5	5 GRX (GPS Receive)	
6	6 PWR CTL (Power Control)	
7	7 LED1 (Light Emitting Diode)	
8	8 LED2	
9	9	
10	10	Out
11	11	
12	12	
13	13	
GND	GND	
AREF	AREF (Analog Reference Voltage)	
SDA	SDA	
SCL	SCL	

TABLE 1: Pin connections for Arduino, Weather Shield and DHT11

Table 2 lists all the pin connections made from the Raspberry Pi 3B+ to the SIM7600X 4G module. The waveshare SIM7600 hat is mounted on top of the RPI to form the connections listed in Table 2.

Raspberry Pi Model 3B+	Waveshare SIM7600X 4G Hat
3.3V	

5V	
GPIO2 (General-Purpose Input/Output)	
5V	5V
GPIO3	
GND (Ground)	GND
GPIO4	D4
GPIO14	PTX (Power Amplifier Transmit Enable)
GND	GND
GPIO15	PRX (Power Amplifier Receive Enable)
GPIO17	
GPIO18	
GPIO27	
GND	GND
GPIO22	
GPIO23	
3.3V	
GPIO24	
GPIO10	
GND	GND
GPIO9	
GPIO25	
GPIO11	
GPIO18	
GND	GND
GPIO7	
DNC (Do Not Connect)	
DNC	
GPIO5	
GND	GND
GPIO6	D6
GPIO12	
GPIO13	
GND	GND
GPIO19	
GPIO16	
GPIO26	
GPIO20	
GND	GND
GPIO21	

TABLE 2: Pin connections for RPI and 4G Hat

Arduino Programming

This section provides a description of the Arduino Programming for the data acquisition process, including taking readings from the SparkFun® Weather Shield and the DHT-11 Humidity and Temperature sensor and processing the collected reading by the Arduino Uno microcontroller. Figure *3* displays the functionalities of the setup() and loop() methods running in the SparkFun® Weather_Readings.ino sketch.

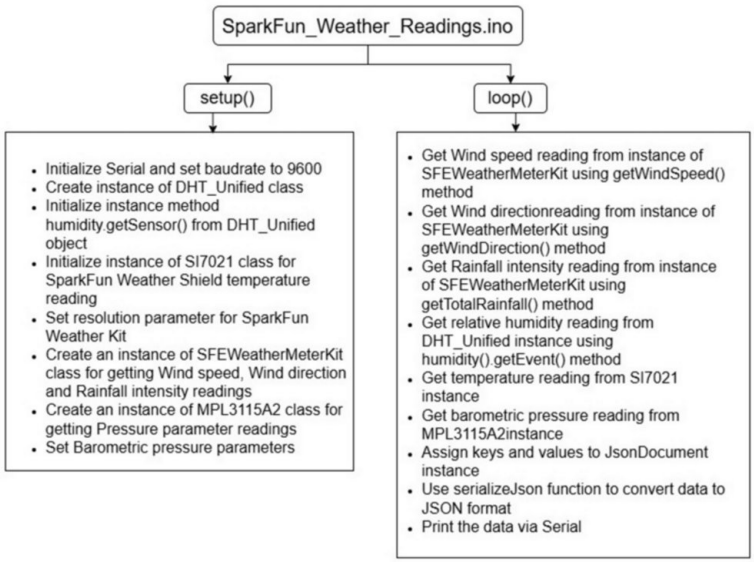


FIGURE 3: SparkFun® Weather_Readings.ino

[25]

Table *3* gives a brief description of some of the methods used in the Arduino Programming for the weather parameters.

Function	Purpose
getSensor()	This is an instance method that takes an argument of struct sensor_t, which provides information about the DHT sensor used. This method interacts with the DHT11 sensor.
getWindSpeed()	This is a static method that gives the value of the wind speed parameter reading. It takes no argument and returns a float. This method is provided by the SFEWeatherMeterKit class, which is an instance of the weather meter kit.
getWindDirection()	This is a static method that gives the value of the wind direction parameter reading. It takes no argument and returns a float. This method is provided by the SFEWeatherMeterKit class, which is an instance of the weather meter kit.
getTotalRainfall()	This is a static method that gives the value of the rainfall intensity parameter reading. It takes no argument and returns a float. This method is provided by the SFEWeatherMeterKit class, which is an instance of the weather meter kit.
getEvent()	This is an instance method that takes an argument of struct sensor_event_t, which provides a sensor event. This method interacts with the Si7021 module on the weather shield.

TABLE 3: Function and purpose for weather parameters

WeatherClient Package

The WeatherClient is responsible for managing inputs from serial ports, terminating any unused port for no further serial communication, setting up the SIM7600X hat module and, as a whole, performing all the necessary steps for the completion of blockchain transmission system for the weather parameters of wind

speed, wind direction, rainfall intensity, relative humidity, temperature and barometric pressure. The WeatherClient package contains a WeatherClient.java class and eight other classes whose functionalities are detailed further in this section. Figure 4 shows all of the classes in the WeatherClient package.

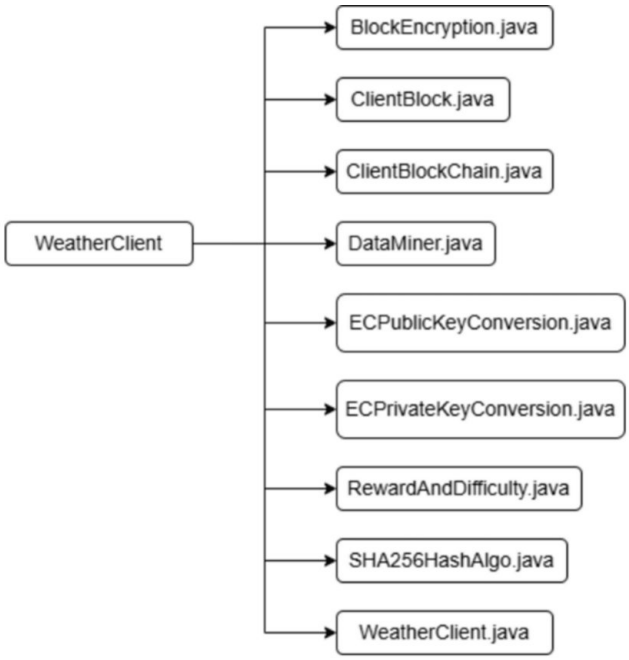


FIGURE 4: WeatherClient classes

Classes and Methods

Table 4 demonstrates the functionalities of the methods within their class for the data acquisition system.

Class	Methods	Purpose
BlockEncryption.java	encryptData(ECPrivateKey keyPrivate, ECPublicKey keyPublic, String data)	Initialises ECDH key agreements with ECPrivateKey and ECPublicKey Objects; Derives SecretKey using SHA-256 algorithm; Performs encryption on the String data; Returns the encrypted text
	getEncryptedMessage()	Returns the encrypted text
ClientBlock.java	ClientBlock(int id, String transaction, String previousHash)	Constructor to the ClientBlock class. Responsible for initializing instance variable id, nonce, timestamp, hash, previousHash and data_transaction; Executes generateHash() method
	generateHash()	Generates a hash using SHA256 hashing algorithm based on the properties provided
	incrementNonce()	Adds 1 to the nonce value
	setPreviousHash(String previousHash)	Takes a String value of the previous hash and sets it to instance variable previousHash
	getPreviousHash()	Returns the value of the instance variable previousHash
	setHash(String hash)	Takes a String value of the current hash and sets it to instance variable hash
	getHash()	Returns the value of the instance variable hash
	toString()	Returns the block of data containing id, hash, previousHash and data_transaction in a String representation
	ClientBlockChain()	Initialises a new LinkedList and assigns it to the instance variable blockChain

ClientBlockChain.java	addBlock(ClientBlock block)	Adds the provided ClientBlock object to the blockChain LinkedList
	getBlockChain()	Returns the blockChain LinkedList
	getSize()	Returns the size of the blockChain LinkedList
	Clear()	Clears the blockChain LinkedList
	toString()	Overrides the toString() method and returns a string representation containing the elements in the blockChain LinkedList
DataMiner.java	mine(ClientBlock block, ClientBlockChain blockChain)	If the ClientBlock object provided does not meet the difficulty level, it increments the nonce value and regenerate the hash
	isGoldenHash(ClientBlock block)	Checks if the ClientBlock object meets the difficulty level set.
ECPrivateKeyConversion.java	convertStringToECPrivate(String a)	Returns an object of type ECPrivateKey Uses ECDH cryptographic algorithm to convert the String provided into a ECDH private key representation
ECPublicKeyConversion.java	convertStringToECP(String a)	Returns an object of type ECPublicKey Uses ECDH cryptographic algorithm to convert the String provided into a ECDH public key representation
RewardAndDifficulty.java		Sets the difficulty level for the hashing algorithm; Sets the value of the previous hash of the first block to a String of 64 zeros
SHA256HashAlgo.java	generateHash(String data)	Generates a hash using SHA-256 hashing algorithm and returns it as a hexadecimal string
WeatherClient.java	WeatherClient(String serverAddress, int serverPort, boolean blockchainmode)	Initialises instance variables using provided arguments including
	main()	Calls CheckandKillSerialPort(), connectSIM7600(), ChoosingUSBPort() and ConnectUSBwithSerialPort() methods
	run()	Responsible for handling serial data, sending data to the server application, performing blockchain transmission
	connectSIM7600(boolean state)	Sets up the SIM7600X 4G settings if state argument is set to True Uses ATCommands to configure SIM7600 module Uses InputStream and OutputStream Objects to write/read commands to/from SIM7600 module
	sendATCommand(OutputStream out, String command)	Writes String argument via OutputStream Object
	readATCommand()	Reads from InputStream object; Checks if response from SIM7600 is “OK”
	ChoosingUSBPort()	Assigns and identifies primary and secondary port numbers to specific Arduino Uno microcontroller’s serial ports
	ConnectUSBwithSerialPort()	Handles input data from serial ports of Raspberry Pi; Sets baudrate, databit, stopbit, parity and timeout parameters on Raspberry Pi
	startSerialReader()	Calls LoggerManager(), readSerialContinuously() and writeDATALOGGER() methods to handle incoming serial data
	start()	Initialises the ExecutorService Object
	stop()	Terminates the ExecutorService Object
	readSerialContinuously(BufferedReader input1, BufferedReader input2, String DeviceID, String latitude, String longitude, String username)	Reads serial data from InputStream Object and converts data into JSON format; Concatenates a timestamp value, latitude and longitude coordinates, username and Device ID to the JSON-formatted data
	CheckandKillSerialPort(String[] portname)	Iterates over the serial ports of the Raspberry Pi and checks which ports are in use
		Terminates any unused serial ports by looking at the process ID

	killProcessUsingPort(String portName)	running on that port
	getUsedMemory()	Calculates and returns the size of memory in use
	writeStats(String[] jsonStats, File statsFile, CSVWriter statsWriter)	Writes blockchain metrics into tabulated format in a CSV file
	WriteDATALOGGER(String json, String csvFileName)	Converts incoming serial data from json String into JSON format and writes it to a CSV file
	LoggerManager(String fileName)	Writes logs into CSV file and overwrites the file after 30 days
	log(String message)	Prints argument String together with a timestamp in Runtime Output window of Netbeans IDE
	createProgramLogFile()	Creates a CSV log file to be modified

TABLE 4: Class, methods and purpose for data acquisition system

[26]

Data Acquisition System Logic Flowchart

Figure 5 illustrates the logic running behind the data acquisition system.

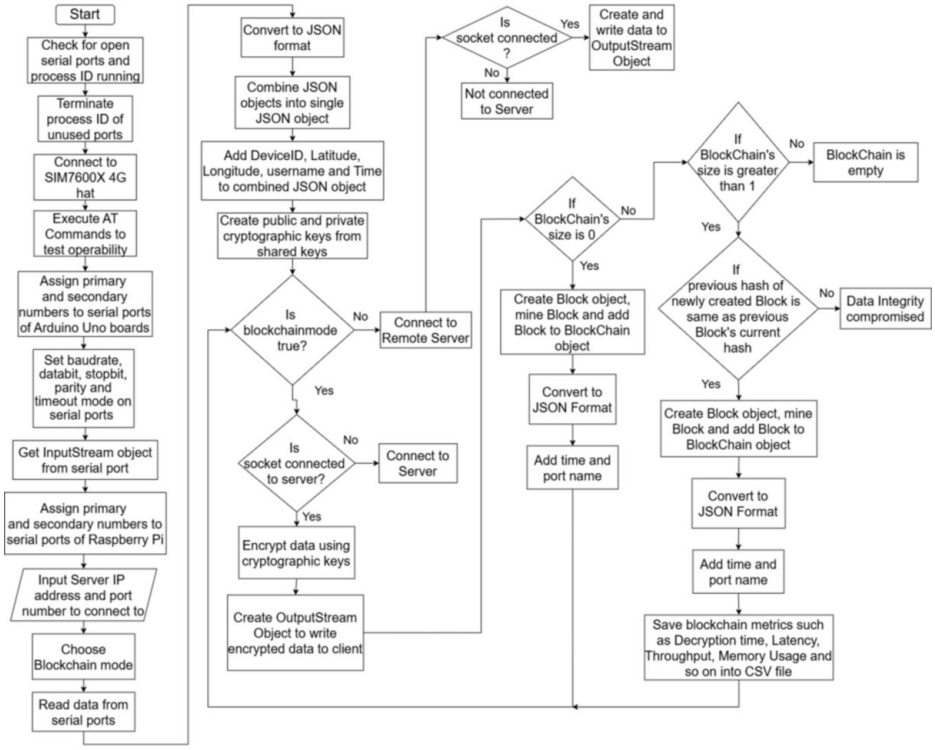


FIGURE 5: Data acquisition logic

WeatherServer Package

The WeatherServer package is responsible for handling the socket connection with the Raspberry Pi client, computation of the blockchain transmission, establishing a connection with a local MySql database, performing predictions for weather parameters and displaying a GUI to the user. Figure 6 demonstrates all the classes inside the WeatherServer package.

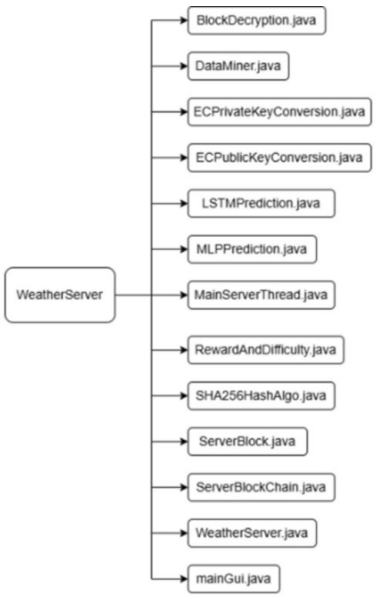


FIGURE 6: WeatherServer classes

Description of Classes and Methods

Table 5 demonstrates the functionalities of the methods for the Server application.

Class	Methods	Purpose
BlockDecryption.java	decrypt(ECPrivateKey keyPrivate, ECPublicKey keyPublic, String encryptedMessage)	Initialises ECDH key agreements with ECPrivateKey and ECPublicKey Objects; Derives SecretKey using SHA-256 algorithm; Performs decryption on the String data; Returns the decrypted text
DataMiner.java	mine(ServerBlock block, ServerBlockChain blockChain)	Same as DataMiner.java in WeatherClient (see Table 4)
ECPrivateKeyConversion.java	As given in Table 4	-
ECPublicKeyConversion.java	As given in Table 4	-
MainServerThread.java	MainServerThread(Socket clientSocket, mainGui main_G)	Initialises instance variables socket, gui_main, rpiArea, dataTable and newFile; Creates a new CachedThreadPool; Establishes a connection with the local MySQL database
	run()	Generates public and private cryptographic keys; Creates a CSV file and writes its headers; Creates an instance of the ServerBlockChain class; Checks if the size of the ServerBlockChain Object is 0 or greater than 1; Creates an instance of ServerBlock class and adds it to the ServerBlockChain object; Decrypts the data from the ServerBlock object and updates all necessary GUI components; Saves decryption time, latency and throughput values in a CSV file
	getUsedMemory()	Same as getUsedMemory() in WeatherClient (see Table 4)
	writeStats(String[] jsonStats, File statsFile, CSVWriter statsWriter)	Same as writeStats() in WeatherClient (see Table 4)
RewardAndDifficulty.java	Same as RewardAndDifficulty.java in WeatherClient (see Table 4)	-
SHA256HashAlgo.java	Same as SHA256HashAlgo.java in WeatherClient (see Table 4)	-
ServerBlock.java	Same as ClientBlock.java in WeatherClient (see Table 4)	-
ServerBlockChain.java	Same as ClientBlockChain.java in	-

	WeatherClient (see Table 4)	
SplitBlock.java	SplitBlock(String a)	Check if each component is valid characters/format; Splits the input String into id, hash, previousHash and transaction components
WeatherServer.java	main(String[] args)	Create instances of mainGui and Thread classes; Starts the Thread object
mainGui.java	mainGui()	Initialises Java Swing components; Calls initMap() to display Map Layout; Calls showChart to display time series charts; Creates a DefaultTableModel Object to display weather data
	showChart()	Creates and displays time series charts on Monitoring Window
	initMap(double lat, double longi)	Sets parameters for displaying the Map Layout; Adds Mouse Drag and Scroll events
	getData(double lat, double longi)	Sets latitude and longitude instance variables to input arguments; Stores current weather parameter readings, Plots Marker overlay on Map Layout
	initWaypooint()	Adds Marker overlay on Map Layout
	clearWaypoint()	Clear all Marker overlays on Map Layout
	getEvent()	Handles Marker-click event by displaying a Message Dialog
	monitor_selectActionPerformed()	Displays the Monitoring Window if not already displaying
	server_selectActionPerformed()	Displays the Server Window if not already displaying
	logger_selectActionPerformed ()	Displays the Data Logger Window if not already displaying
	predict_btActionPerformed()	Performs predictions using either LSTM or MLP algorithms from selected CSV file and displays predicted value with accuracy in GUI window
	getLastVal (double[] dataArray)	Returns the last value in the dataset array
	datasetArray(String csv, int window_size)	Converts data from a CSV file into an Array
	prediction_selectActionPerformed()	Displays the Prediction Window if not already displaying
	close_serverActionPerformed()	Terminates the ServerSocket, Socket and Thread instances
	defFileActionPerformed()	Sets the default path to the root directory of the project
	browseFileActionPerformed()	Uses JFileChooser Object to let user select a file
	start_serverActionPerformed()	Initialises a ServerSocket Object on port 5538; Accepts incoming client connections; Starts the main Thread

TABLE 5: Description of classes and methods for server application

CSV, Comma-Separated Values; GUI, Graphical User Interface; LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

LSTMPrediction.java

The AlgoLSTM.java class is responsible for performing predictions using LSTM algorithm.

Consider the mathematical equations typically employed in the LSTM network for an input in

$$f_{gt} = \sigma_{sig}(Z_{fg} \cdot [hid_{n-1}, i_n] + S_{fg}) \quad (1)$$

$$i_{gt} = \sigma_{sig}(Z_{ig} \cdot [hid_{(n-1)}, i_n] + S_{ig}) \quad (2)$$

$$T_c = \sigma_{tanh}(Z_c \cdot [hid_{(n-1)}, i_n] + S_c) \quad (3)$$

$$C_c = f_{gt} * C_{c-1} + i_{gt} * T_c \quad (4)$$

$$o_{gt} = \sigma_{sig}(Z_{og} \cdot [hid_{(n-1)}, i_n] + S_{og}) \quad (5)$$

$$hid_n = o_{gt} \cdot \sigma_{tanh}(C_c) \quad (6)$$

whereby f_{gt}, i_{gt} and o_{gt} denote the forget, memory and output gates, respectively; T_c and C_c denote the temporary and current cell states; σ_{sig} and σ_{tanh} denote the activation functions; Z_{fg}, Z_{ig}, Z_c and Z_{og} denote the weight matrices; S_{fg}, S_{ig}, S_c and S_{og} denote the biases; and hid_n and hid_{n-1} correspond to hidden state and previous hidden state inputs [15].

Figure 7 details the functionalities of the LSTMPrediction.java class.

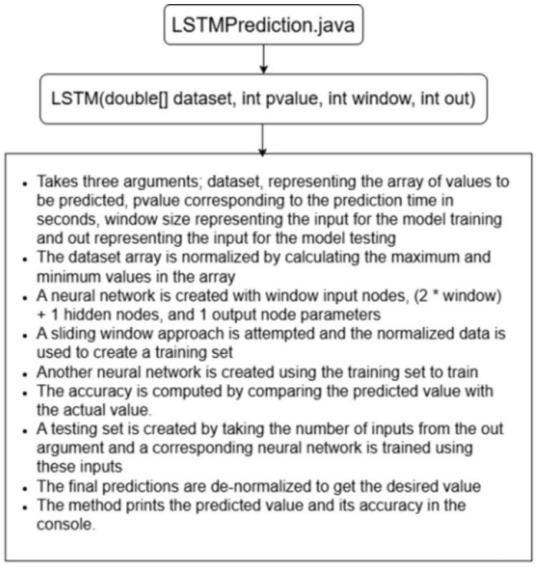


FIGURE 7: AlgoLSTM.java class and methods

LSTM, Long Short-Term Memory

MLPPrediction.java

The AlgoMLP.java class is responsible for performing predictions using MLP algorithm.

Weighted sums and irregular transformations are performed in the hidden layer of an MLP. Such weighted sum, W_{sum}, is computed for N number of inputs using

$$W_{sum} = \sum_{k=1}^{N} (w_k * i_k) + S \quad (7)$$

where W_k is the weight at position k, i_k denotes the input at position k and S denotes the bias [15].

$$O_{act} = \frac{1}{1 + e^{-W_{sum}}} \quad (8)$$

where O_{act} denotes the sigmoid activation function applied to W_{sum}.

Figure 8 demonstrates the functionalities of the MLPPrediction.java class.

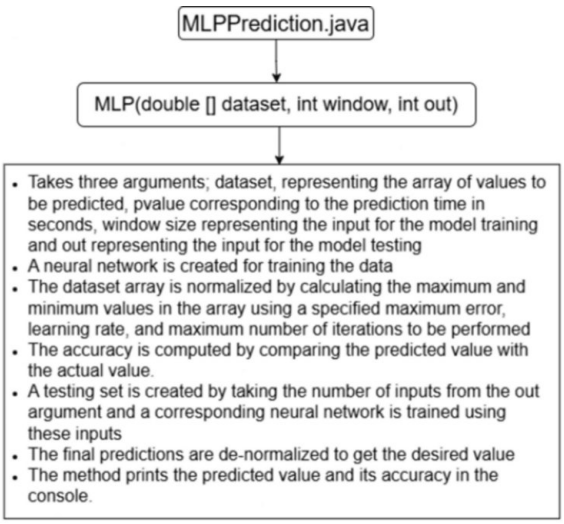


FIGURE 8: AlgoMLP.java class and methods

MLP, Multilayer Perceptron

Desktop Application Layout

Figure *9* shows the first window when the application is launched.

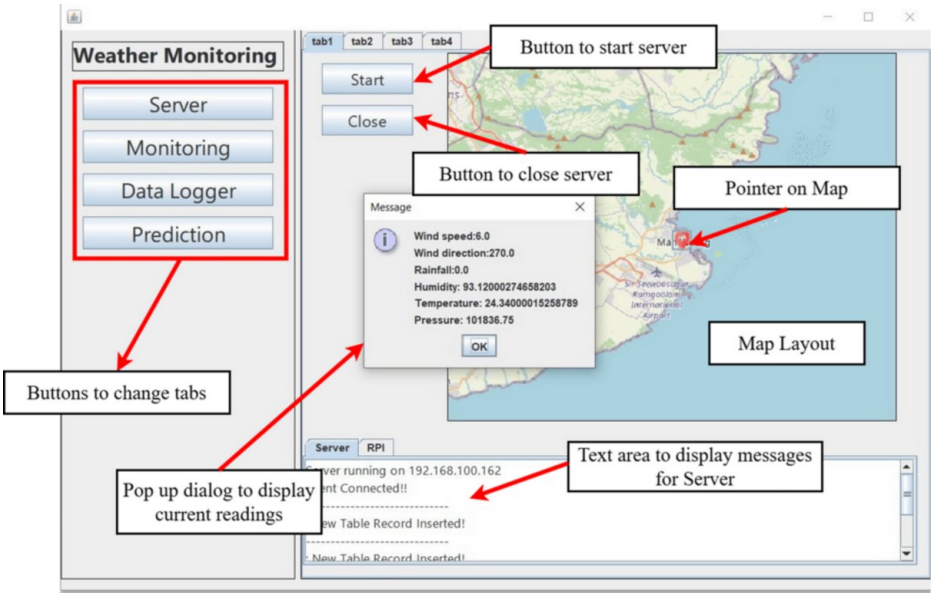


FIGURE 9: First tab of GUI layout

GUI, Graphical User Interface

Figure *10* shows the second tab responsible for displaying real-time graphs obtained from the decrypted weather parameter readings by the Server application.

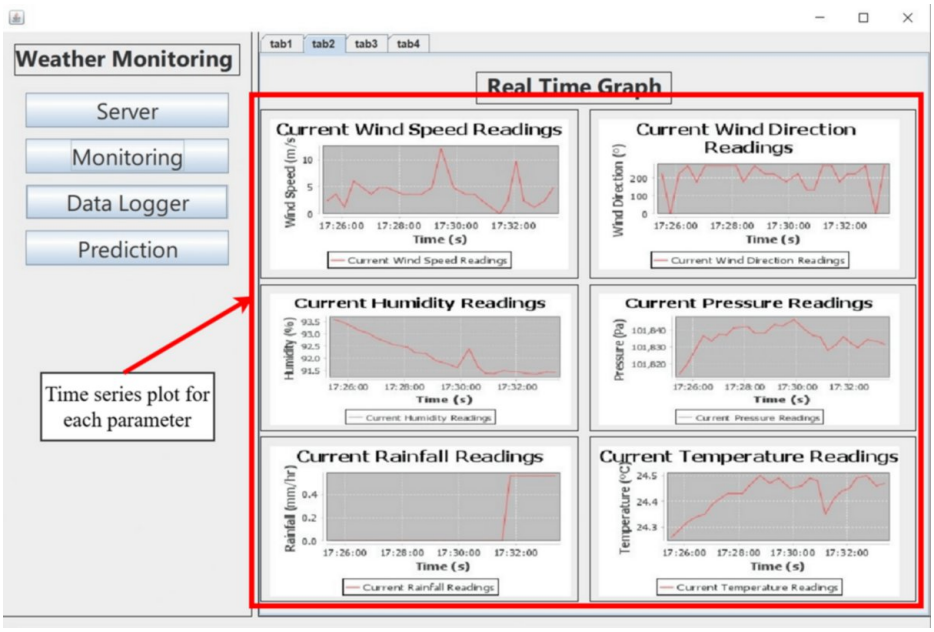
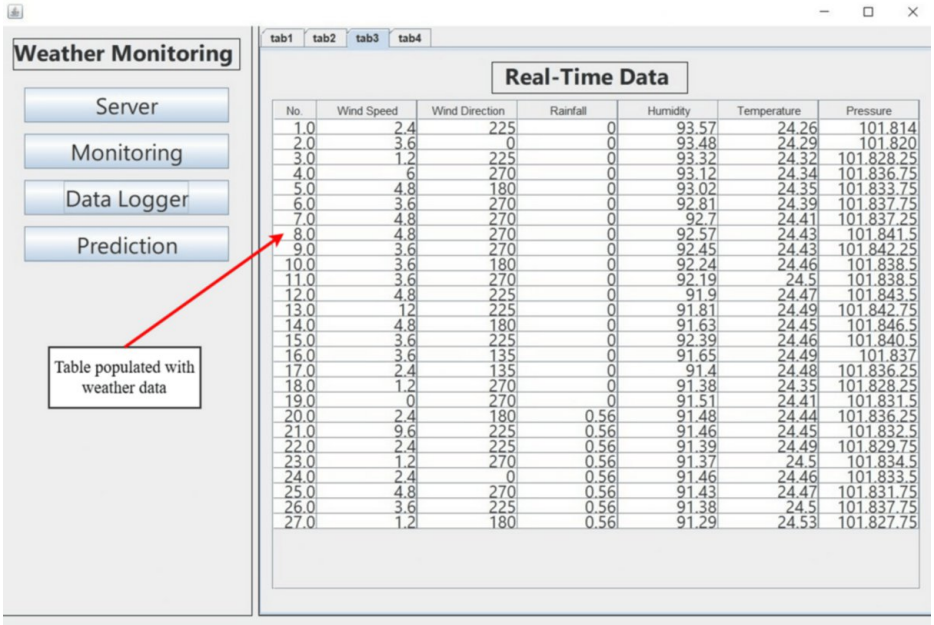


FIGURE 10: Second tab of GUI layout

GUI, Graphical User Interface

Figure *11* shows the third tab that contains a table for displaying the tabulated data that is being decrypted by the Server application.



No.	Wind Speed	Wind Direction	Rainfall	Humidity	Temperature	Pressure
1.0	2.4	225	0	93.57	24.26	101.814
2.0	3.6	0	0	93.48	24.29	101.820
3.0	1.2	225	0	93.32	24.32	101.828.25
4.0	6	270	0	93.12	24.34	101.836.75
5.0	4.8	180	0	93.02	24.35	101.833.75
6.0	3.6	270	0	92.81	24.39	101.837.75
7.0	4.8	270	0	92.7	24.41	101.837.25
8.0	4.8	270	0	92.57	24.43	101.841.5
9.0	3.6	270	0	92.45	24.43	101.842.25
10.0	3.6	180	0	92.24	24.46	101.838.5
11.0	3.6	270	0	92.19	24.5	101.838.5
12.0	4.8	225	0	91.9	24.47	101.843.5
13.0	12	225	0	91.81	24.49	101.842.75
14.0	4.8	180	0	91.63	24.45	101.846.5
15.0	3.6	225	0	92.39	24.46	101.840.5
16.0	3.6	135	0	91.65	24.49	101.837
17.0	2.4	135	0	91.4	24.48	101.836.25
18.0	1.2	270	0	91.38	24.35	101.828.25
19.0	0	270	0	91.51	24.41	101.831.5
20.0	2.4	180	0.56	91.48	24.44	101.836.25
21.0	9.6	225	0.56	91.46	24.45	101.832.5
22.0	2.4	225	0.56	91.39	24.49	101.829.75
23.0	1.2	270	0.56	91.37	24.5	101.834.5
24.0	2.4	0	0.56	91.46	24.46	101.833.5
25.0	4.8	270	0.56	91.43	24.47	101.831.75
26.0	3.6	225	0.56	91.38	24.5	101.837.75
27.0	1.2	180	0.56	91.29	24.53	101.827.75

FIGURE 11: Third tab of GUI layout

GUI, Graphical User Interface

Figure *12* shows the prediction window for displaying predicted output and its accuracy.

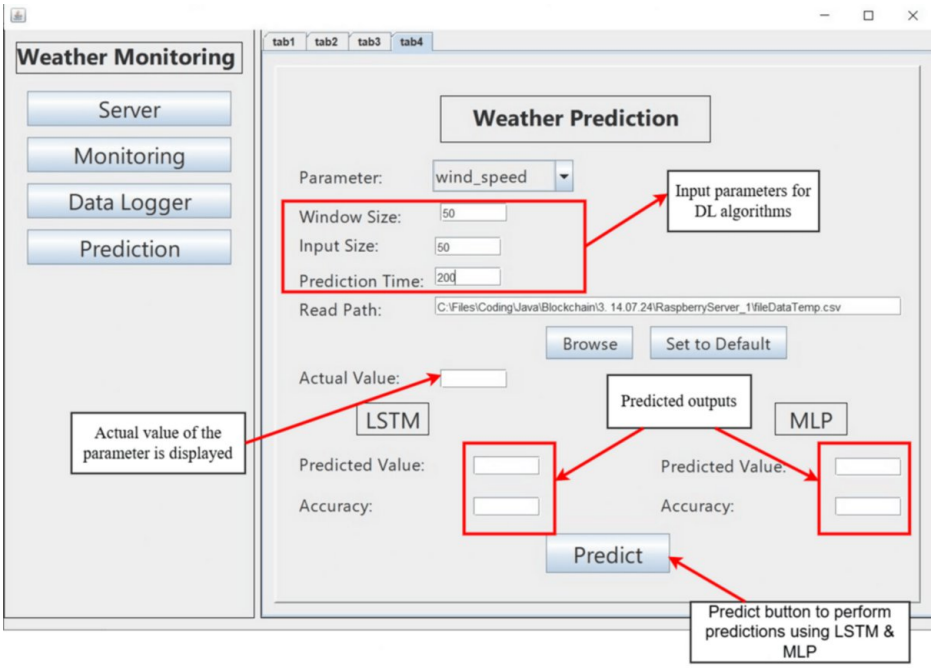


FIGURE 12: Fourth tab of GUI layout

GUI, Graphical User Interface; LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

Server Application Logic Flowchart

Figure *13* shows the flowchart for the logic behind the Server Application running.

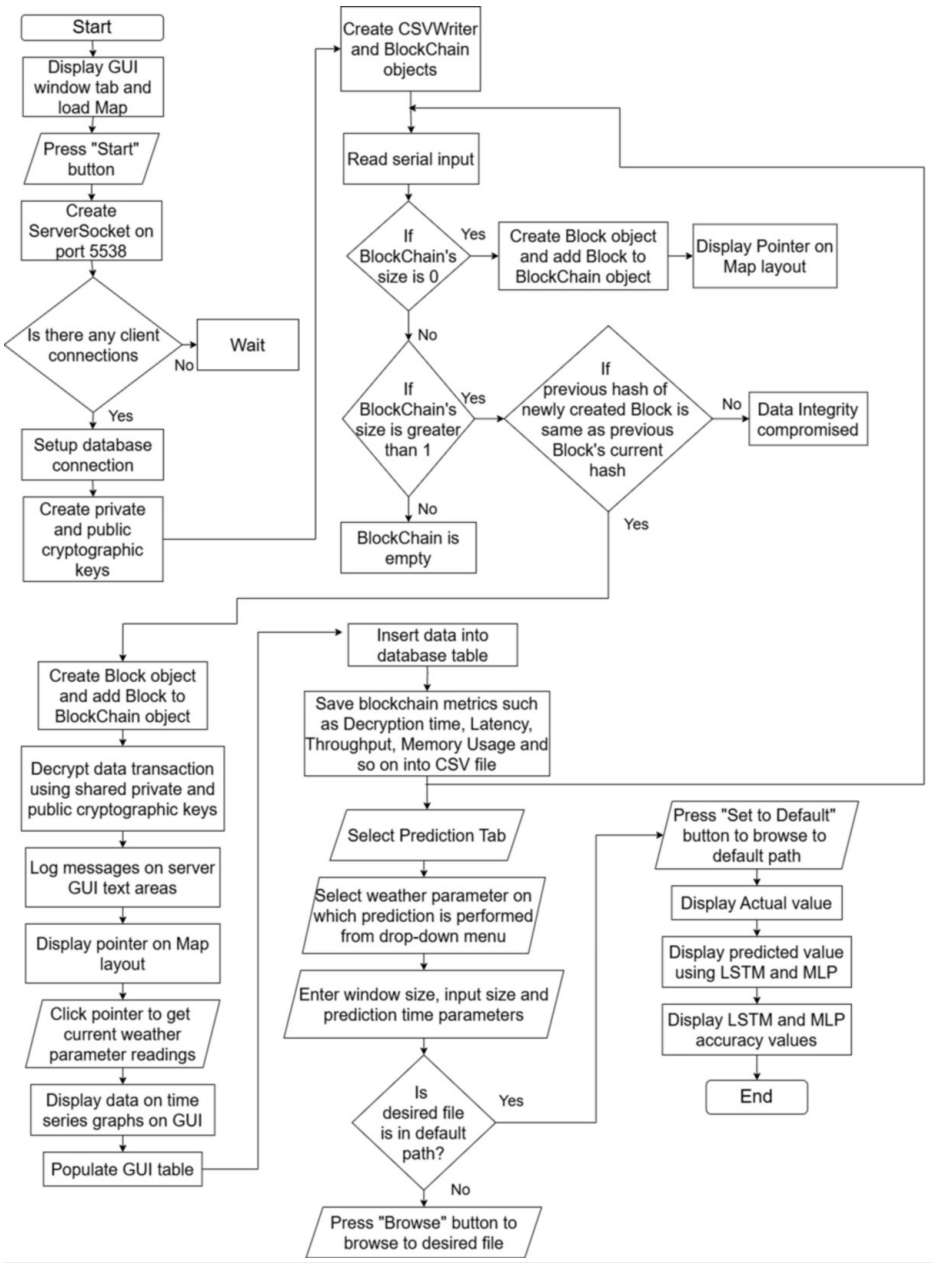


FIGURE 13: Server application logic

GUI, Graphical User Interface; LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

MySQL Database Configuration

When the Java application decrypts the data from the block, it sends the latter to a local MySQL database. This local database is responsible for keeping records of data collected. The MySQL database is set up using the XAMPP server, as shown in Figure *14*. Users can verify if MySQL is up and running and can click the Admin button to set up the database. The database name and table name are shown in Figure *15*.

FIGURE 14: XAMPP control panel

[27]

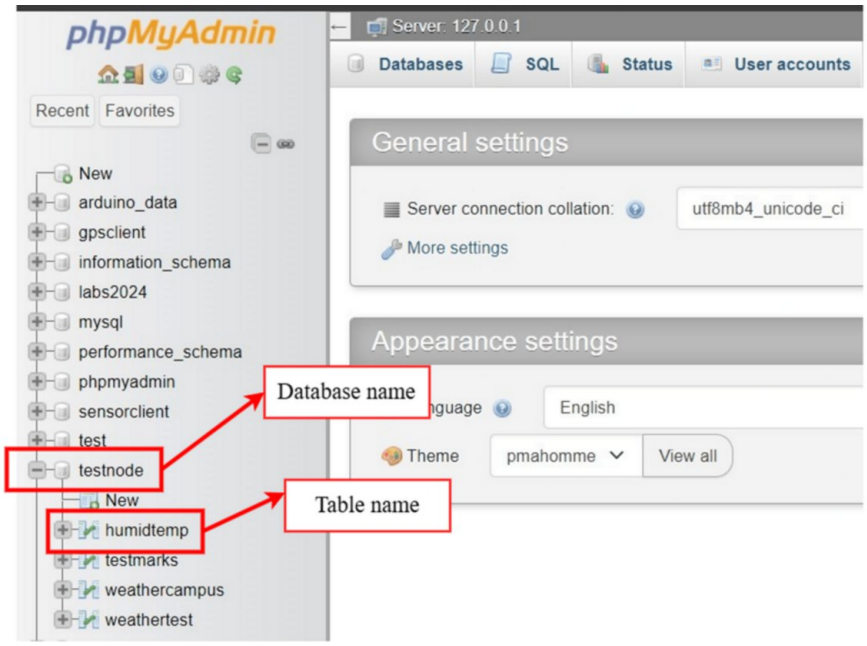


FIGURE 15: Database and table

Figure *16* shows the configured table and its variables. The variable “id” is automatically incremented pertaining to the index of the row of values being added. “datetime” denotes the time instant the data row is added to the table. The variables “wind_speed”, “wind_direction”, “rainfall”, “humidity”, “temperature” and “pressure” represent the weather data to be collected.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	datetime	datetime			No	current_timestamp()			Change Drop More
3	wind_speed	float			No	None			Change Drop More
4	wind_direction	float			No	None			Change Drop More
5	rainfall	float			No	None			Change Drop More
6	humidity	float			No	None			Change Drop More
7	temperature	float			No	None			Change Drop More
8	pressure	float			No	None			Change Drop More

Check all With selected: Browse Change Drop Primary Unique Index Spatial

FIGURE 16: Database table

Results

Experimental setup

Figure *17* shows the finalised model setup consisting of the microcontrollers connection together with the IoT apparatus. The Windows laptop is used to power the Raspberry PI via USB cable. The SIM7600 Hat is mounted onto the Raspberry PI, providing a Hotspot for the Raspberry PI to connect to. The Arduino Uno is connected via USB cable to the Raspberry PI, and the SparkFun® Weather Shield is mounted on the Arduino Uno. The SparkFun® Weather Kit is connected to the SparkFun® Weather Shield via RJ11 cables, for wind speed and wind direction parameter monitoring. The DHT11 is connected to the Arduino Uno for humidity reading via jumper cables. The whole forms the data acquisition system model that can operate remotely, while getting the WiFi connection from the SIM7600 hat and powering up via an addition to a battery (not used here).

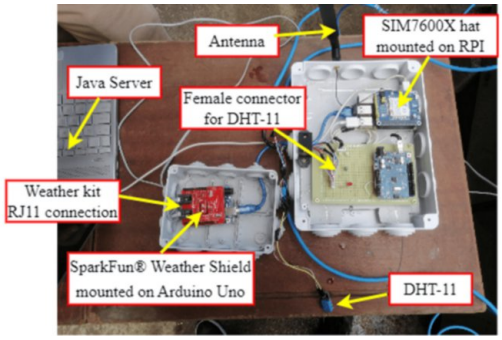


FIGURE 17: Microcontrollers setup

The SparkFun® Weather Kit consists of a wind vane for wind direction readings, an anemometer for wind speed readings and a self-emptying bucket for measuring rainfall intensity, as shown in Figure *18*.

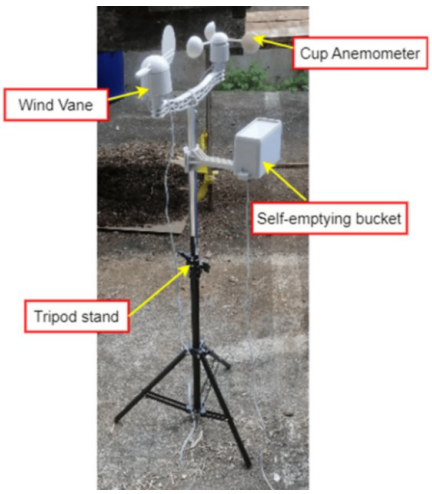


FIGURE 18: SparkFun® weather kit

Field testing setup

The system was tested in a moderately windy and rainy weather on 19 August 2024 from 16:03 to 17:44 (Figure *19*).

FIGURE 19: Setup on a moderately windy and rainy day

The events of the experiment included some little rain drops to moderate rainfall. The experiment was conducted at Beau Vallon with coordinates 20.42°S, 57.70°E. Five experiments were conducted, as described in Table *6*.

Experiment	Date and Time	Place
1	19 August 2024, from 16:03 to 17:43	Beau Vallon, Mauritius (20.42°S, 57.70°E)
2	20 August 2024, from 07:03 to 08:04	Beau Vallon, Mauritius (20.42°S, 57.70°E)
3	02 September 2024, from 17:05 to 19:00	Beau Vallon, Mauritius (20.42°S, 57.70°E)
4	05 September 2024, from 12:49 to 14:17	Beau Vallon, Mauritius (20.42°S, 57.70°E)
5	05 September 2024, from 15:51 to 16:31	Beau Vallon, Mauritius (20.42°S, 57.70°E)

TABLE 6: Experiment and description

Field deployment results

One of the tests is done on the 20th of August 2024 as from 7 a.m. to 8. a.m.

Figure *20* shows the readings obtained for the wind speed parameter.

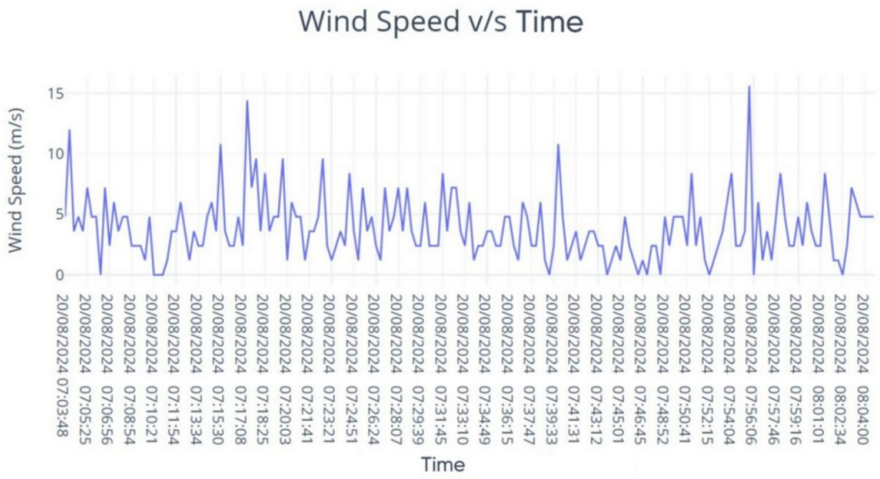


FIGURE 20: Wind speed readings

The wind speed readings obtained show variations between 0 m/s and 15.6 m/s, whereby the highest value is obtained at 07:56 a.m.

Figure *21* shows the readings obtained for the wind direction parameter.

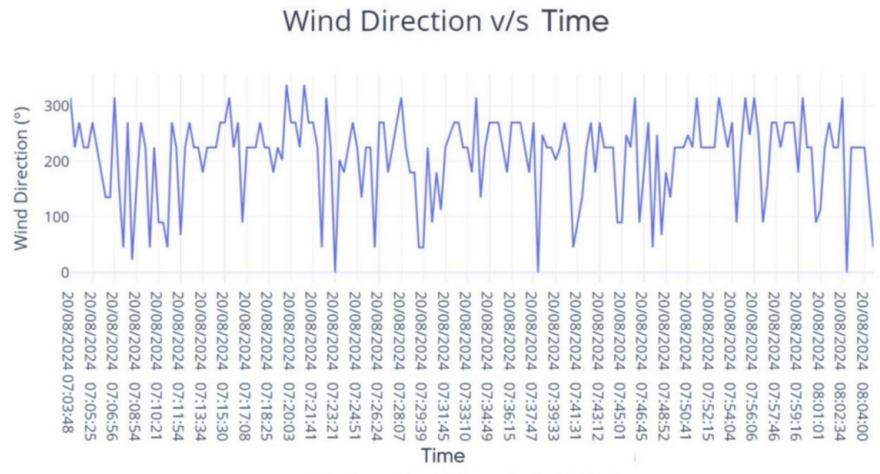


FIGURE 21: Wind direction readings

Variations obtained for the wind direction values are expected signifying wind blowing in various directions.

Figure *22* shows the readings obtained for the rainfall intensity parameter.

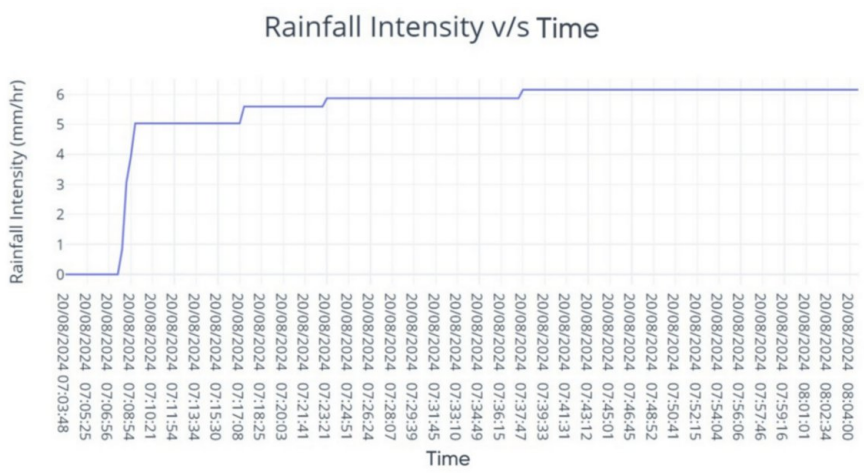


FIGURE 22: Rainfall intensity readings

The rainfall started 8 minutes after the start of the test with a rapid increase in the amplitude. This can denote an increase in the rain intensity falling for a short amount of time. A staircase relationship is observed for the rest of the experiment, which signifies different rate of rain falling. From the values obtained, it can be said that slow increase in rainfall intensity occurred during the test, meaning that the test ended in a less rainy atmosphere.

Figure *23* shows the readings obtained for the relative humidity parameter.

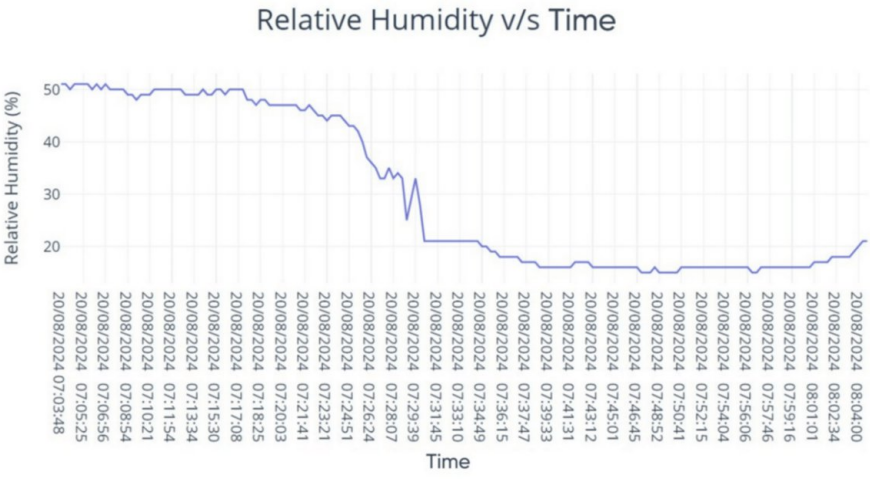


FIGURE 23: Relative humidity readings

Relative humidity readings demonstrate fluctuations at the start from 51% to 47%, and the plot is seen to gradually decrease, signifying a warmer atmosphere was established after the rainfall as expected.

Figure *24* shows the readings obtained for the temperature parameter.

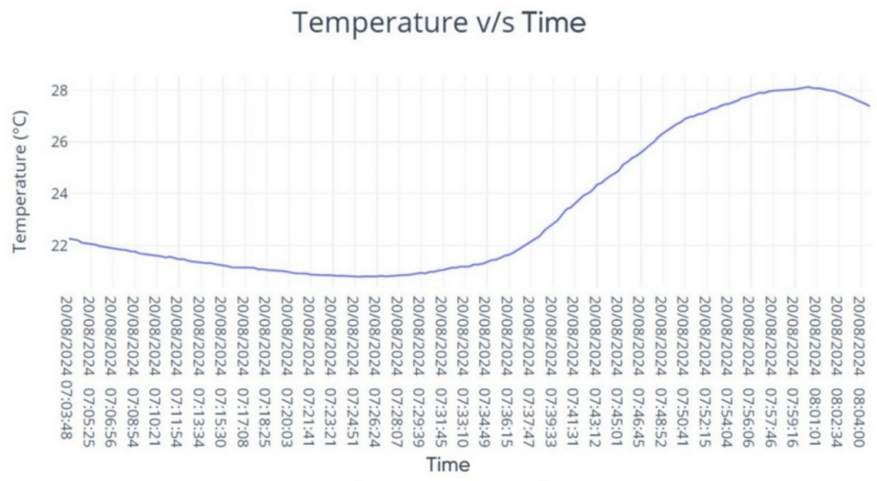


FIGURE 24: Temperature readings

The trend obtained from the temperature reading signifies that the temperature flowed from low to high during the testing done. The expected trend observed shows the weather started to normalize at the end of the test.

Figure *25* shows the readings obtained for the barometric pressure parameter.

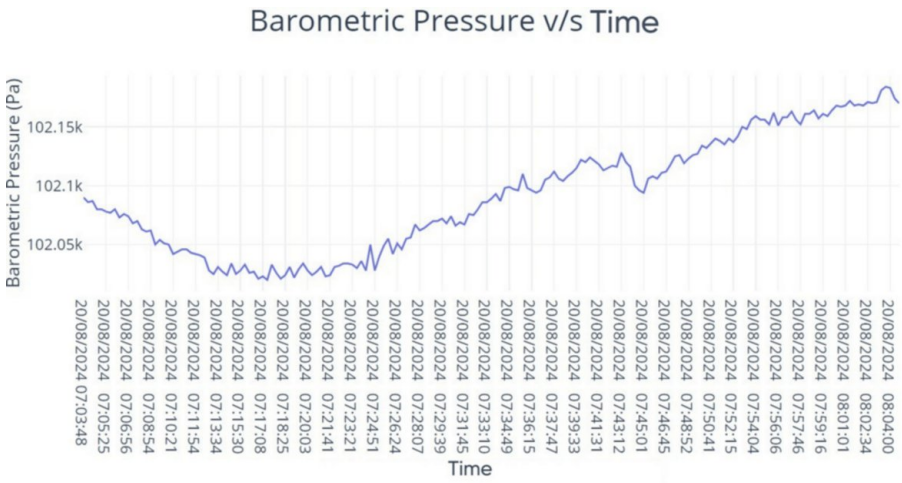


FIGURE 25: Barometric pressure

Similarly, a gradually increasing trend is observed for the barometric pressure reading, signifying the weather was cleared at the end of the test.

Performance analysis of blockchain transmission

The performance analysis of the blockchain system is done by running a number of tests for a fixed duration of 45 minutes and varying the number of readings inside the block. The number of readings was varied by 1, 5, 10, 15, 20 and 25. The set of readings to be used contained four constant values (username, DeviceID, Latitude and Longitude) and six sensor values (Wind Speed, Wind Direction, Rainfall Intensity, Relative Humidity, Temperature and Barometric Pressure).

Table *7* summarizes the values that have been constant for all comparison done ahead. It can be seen that as the number of sets of readings is increased, the block size also increases, meaning that more memory is being used by the block as more readings are being taken into account. In addition, it can be seen that as the number of readings is increased, the total number of blocks being generated decreases. This is because as more sets of readings are taken into account, more time is taken to process those readings, resulting in less blocks being generated.

Test No.	No. of Sets of Readings	Block Size in Bytes	Total Blocks Generated	Colour
1.	1	368.841	63	Red
2.	5	1648.830508	59	Blue
3.	10	3100.827586	57	Orange
4.	15	4444.811321	53	Green
5.	20	6024.705882	34	Yellow
6.	25	7496.62963	27	Violet

TABLE 7: Blockchain analysis output values

Time complexity against number of blocks

The Time Complexity of a block denotes the time taken by the block to be sent to the Server application via the Java socket by the application running on the Raspberry Pi, which depends solely on the number of sets of readings being taken in account.

Figure *26* shows the time complexity, in milliseconds, against the number of readings per block.

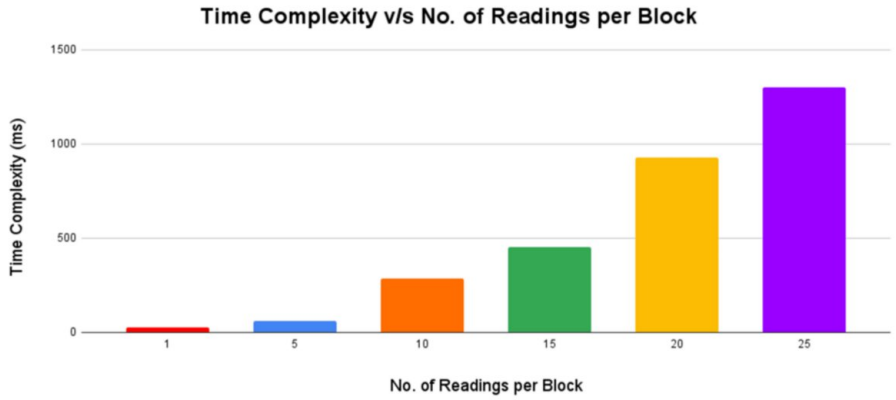


FIGURE 26: Time complexity against number of readings

The bar charts obtained show an increasing trend in the time complexity values. It can be said that as the number of readings in a block increases, the time complexity also increases. Consequently, as the number of readings is increased, the block size also increases as more values are being taken into account. The smallest time complexity is 28.71428571 milliseconds, with only one reading being taken into account per block, whereas the highest time complexity is 1301.481481 milliseconds, with 25 readings being considered.

Latency against number of blocks

Latency of the system is computed using an equation, which denotes the time taken for the block to travel from the Raspberry Pi client to the Java Server application. It can be computed using the following equation:

$$Latency(ms) = t_e - t_s \qquad (9)$$

where t_s is the start time at the Raspberry Pi client, and t_e is the end time at the Java server application.

Figure *27* shows the effect of the number of readings per block generated on the latency.

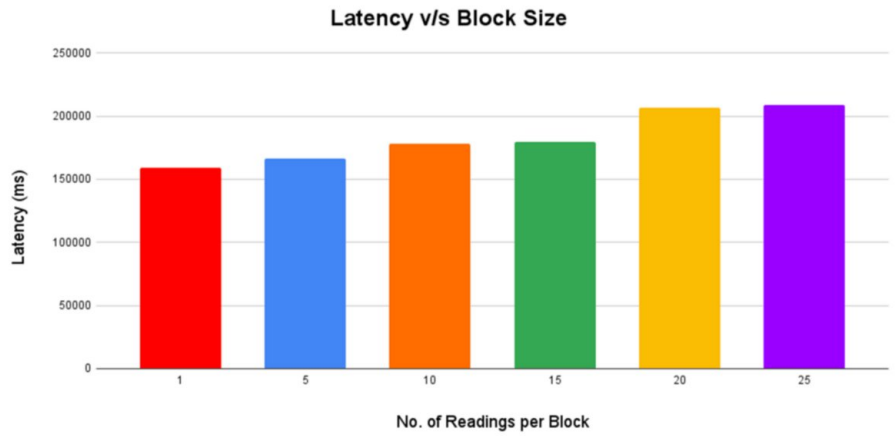


FIGURE 27: Latency against number of readings

The bar charts obtained show that the latency increases as the number of readings increases. These latency values are obtained from the Java server application, whereby the lowest value is 158,969.32 milliseconds for one set of readings and the highest is 208,669.56 for 25 reading sets.

Throughput against number of blocks

Throughput is denoted as the number blocks of data that can be sent for a period of time. This measure is computed using the block size, and the latency as shown in the equation below.

$$Throughput(bps) = \frac{B_s \times 8}{Latency} \qquad (10)$$

where B_s is the block size in bytes.

Figure *28* shows the effect of the number of readings per block generated on the throughput.

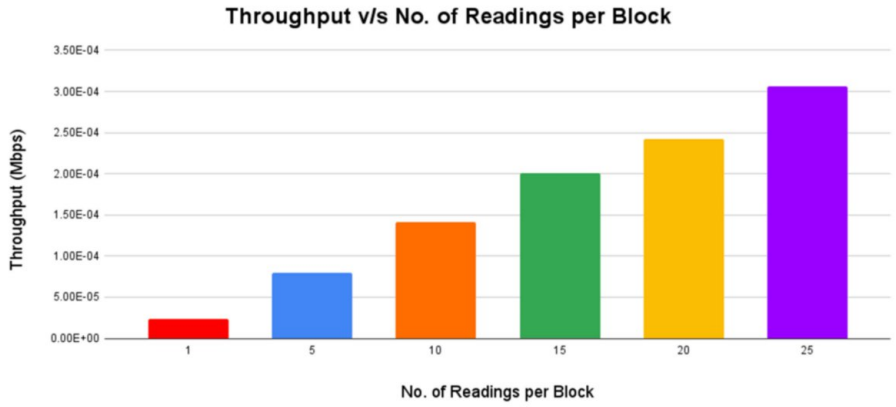


FIGURE 28: Throughput against number of readings

The bar charts obtained show an increasing trend in the throughput values because more readings are being taken per block. As the number of readings in the block increases, the block size also grows, hence impacting the throughput values. As discussed earlier, the latency also increases with the increase in the number of readings per block. The latency has an inversely proportional relationship with the throughput. From early analysis, the latency values increased gradually, which had very little effect on the throughput values. The throughput values are mostly affected by the block size values, as shown in Table *7*, where it increases along with the number of reading sets. The highest observed throughput is 306.48 bps at 25 reading sets per block, and the lowest is 24 bps at one reading set per block.

Performance analysis of machine learning algorithms

The data collected on 19 and 20 August 2024 are used for performing predictions of the weather parameters. LSTM and MLP prediction algorithms are performed for a prediction time of duration 900 seconds for each weather parameters: Wind speed, Wind direction, Rainfall intensity, Relative humidity, Temperature and Barometric Pressure.

The %Error is then computed using the following equation:

$$\text{Error}(\%) = \frac{|\text{Actual}(k) - \text{Predicted}(k)|}{\text{Actual}(k)} \times 100\% \qquad (11)$$

where Actual(*k*) is the actual value at the position *k*, Predicted(*k*) is the predicted value at position *k* and *N* is the number of total positions.

The MAPE is then computed from the %Error using the following equation:

$$MAPE(\%) = \frac{1}{N}\sum_{k=1}^{N} \%Error(k) \qquad (12)$$

where *%Error(k)* is the value of the error percentage at position *k*, and *N* is the number of instances.

Table *8* demonstrates the parameter set used for the LSTM and MLP prediction algorithms.

Parameter	LSTM	MLP
Window Size	50	50
Input Time	50	50
Prediction Time (seconds)	900	900
Max Epochs	1,000	1,000
Max Error	0.001	0.001
Learning Rate	0.1	0.1

TABLE 8: Deep learning parameter

LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

Findings

Table *9* lists the actual and computational values.

Experiment No.	Weather Parameters	Actual	Predicted	
			LSTM	MLP
1	Wind Speed	1.2	1.26	1.12
	Wind Direction	90	83.58	86.7
	Rainfall	17.6	16.1	16.82
	Humidity	18	18.1	18.28
	Temperature	24.25	24.31	24.87
	Pressure	98,717.5	98,687.66	98,692.19
2	Wind Speed	9.6	9.55	10.52
	Wind Direction	337.5	320.59	322.29
	Rainfall	5.59	5.35	5.45
	Humidity	47	49.29	48.79
	Temperature	21.01	21.62	21.46
	Pressure	102,034	102,087.92	102,021.44

TABLE 9: Actual and predicted values

LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

Table *9* shows the all data obtained from the respective weather parameters for the predictions. The tabulated data shown includes the actual reading after the prediction time together with their predicted values obtained from LSTM algorithm, followed by MLP algorithm. From these cells, the average error percentage can be computed.

Table *10* shows the percentage error computed for each parameters for both LSTM and MLP algorithms. The same is repeated for the two experimental datasets obtained on the 19th August 2024 and 20th August 2024.

Weather Parameters	Experiment 1		Experiment 2	
	LSTM	MLP	LSTM	MLP
Wind Speed	5	6.66	0.52	9.58
Wind Direction	7.13	3.66	5.27	4.5
Rainfall	8.52	4.43	4.48	2.5
Humidity	0.55	1.55	4.64	3.8
Temperature	0.24	2.55	2.82	2.14
Pressure	0.03	0.02	0.05	0.01

TABLE 10: %Error values

LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

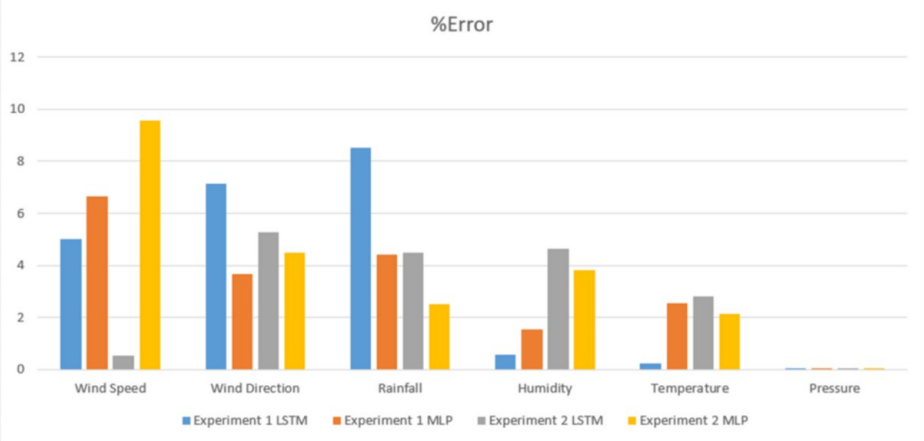


FIGURE 29: %Error graph for two experiments

LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron

Figure *29* shows the error percentage for each weather parameters during the two experiments conducted. The percentage error for the pressure parameter can be seen to be negligible when compared to the other weather parameters for both experiments. The percentage error for the wind speed parameter is seen to be moderately high throughout each experiment but has values less than 15%. Wind direction, humidity and temperature parameters' error percentages are seen to be relatively low as compared to the wind speed and rainfall intensity parameters. The LSTM algorithms can be seen to provide more accurate predictions than the MLP algorithm for weather parameters such as wind speed, humidity and temperature. On the other hand, the MLP algorithm can be seen as more accurate than LSTM algorithm for the wind direction, rainfall and pressure parameters.

Table *11* shows the MAPE values computed from readings in Table *10* for the weather parameters.

Experiment No.	LSTM	MLP
1	2.74	3.15
2	2.96	3.75

TABLE 11: MAPE values for the two experiments

LSTM, Long Short-Term Memory; MLP, Multilayer Perceptron; MAPE, Mean Absolute Percentage Error

Discussion

The aim of this paper was to develop an IoT-based weather monitoring system with blockchain-enabled transmission and to perform weather forecasting on weather parameters using DL algorithms. The data acquisition system consisted of a SparkFun® Meter Kit connected to a SparkFun® Weather Shield, which was mounted on an Arduino Uno microcontroller. The data were then sent to an interconnected Raspberry Pi via a serial port. The system used a SIM7600X 4G HAT to provide the Raspberry Pi with a Wi-Fi connection by generating a hotspot for it to connect to. The data were subsequently relayed to a Java Server application that provided a GUI window for visualization of incoming data and insertion into a local MySQL database. Blockchain transmission was provided by the Raspberry Pi and the Java server, forming the required peer connection. The results were obtained from a series of experiments conducted over four days, totaling nine hours. Since the testing was done near the summer season, moderate rainfall was observed, which affected the weather parameter readings. The implemented DL algorithms included MLP and LSTM. These algorithms were evaluated using the sliding window technique, where predictions were made after querying 50 rows of data from the MySQL database. Once trained, the models were used to generate forecasts. The LSTM algorithm outperformed the MLP algorithm, showing a difference of 0.41% in their MAPE values for the first experiment and 0.79% for the second.

For the blockchain system, the results showed that as the number of reading sets increased, the block size also grew, indicating that more memory is being utilized by each block since more readings are considered. Additionally, with more reading sets, the total number of generated blocks decreased because it took

longer to generate each block, resulting in fewer blocks being produced. The current block's previous hash value is compared with the previous block's current hash value to ensure that the blocks are being added in a sequential manner. If these hashes do not match, it indicates that the data may have been tampered with. The mining process computes these hashes and adds difficulty to the block. Moreover, it can be seen that the time complexity of the system increases as the number of reading sets is taken into account. This is because as more inputs are being considered, more time it is going to take to send the data. The latency values exhibited similar insights, hence validating the claim that the number of reading sets brings delays to the system.

Conclusions

This paper has successfully proposed and developed an IoT-based weather forecasting system with blockchain-enabled transmission, employing deep learning algorithms to perform short-term predictions. The system uses a SparkFun® Meter Kit connected to an Arduino Uno, with data transmitted to a Raspberry Pi, which relayed the data to a Java server via Wi-Fi. The blockchain transmission involves asymmetric encryption using pre-shared keys. Deep learning models such as MLP and LSTM were employed for weather forecasting, and predictions were made using data from a local MySQL database. Results obtained showed that the LSTM algorithm outperformed MLP, with a difference of 0.41% and 0.79% in their mean absolute percentage error values for the experiments conducted. The blockchain analysis revealed that larger reading sets increased the block size while decreasing the total number of generated blocks due to longer processing times. Time complexity, latency and throughput increased with more readings being taken per block. The main novelty of the paper is the development of a real-time weather forecasting system with secure, tamper-proof data transmission that provides a framework for the performance analysis of both weather prediction accuracy and metrics to assess the blockchain performance of the application. This IoT-based blockchain-secured weather forecasting system has broad practical implications, enabling precision agriculture through real-time crop management, enhancing disaster preparedness with tamper-proof early warnings, and optimizing smart city infrastructure such as traffic and energy grids. It can also be used in logistics and transportation routing, supporting renewable energy forecasting for grid stability, and providing policymakers with scalable IoT-security benchmarks. By combining high-accuracy LSTM predictions with blockchain's data integrity, the system can potentially address critical needs across agriculture, urban resilience, transportation, and energy sectors while offering a framework for secure, decentralized environmental monitoring in both developed and under developed regions.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Tulsi Pawan Fowdur, Dhaveesh Parbhoo

Critical review of the manuscript for important intellectual content: Tulsi Pawan Fowdur

Supervision: Tulsi Pawan Fowdur

Acquisition, analysis, or interpretation of data: Dhaveesh Parbhoo

Drafting of the manuscript: Dhaveesh Parbhoo

Disclosures

Human subjects: All authors have confirmed that this study did not involve human participants or tissue. **Animal subjects:** All authors have confirmed that this study did not involve animal subjects or tissue. **Conflicts of interest:** In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Lindwall C: What Are the Effects of Climate Change?. (2022). https://www.nrdc.org/stories/what-are-effects-climate-change#weather..
2. UNEP. Reducing climate change and disaster risk in Mauritius. (2019). https://www.unep.org/news-and-stories/story/reducing-climate-change-and-disaster-risk-mauritius.
3. Le Defi Media Group. Torrential rain: horrendous experiences of flooded places. (2019).

https://defimedia.info/torrential-rain-horrendous-experiences-flooded-places.
4. Mohabuth Y: Cyclone Belal: Mauritius assesses damage after flash flooding. (2024). https://www.bbc.com/news/world-africa-67990946.
5. Bachmann N, Tripathi S, Brunner M, Jodlbauer H: The contribution of data-driven technologies in achieving the sustainable development goals. Sustainability. 2022, 14:2497. 10.3390/su14052497
6. Taherdoost H: Blockchain technology and artificial intelligence together: a critical review on applications. Applied Sciences. 2022, 12:12948. 10.3390/app122412948
7. Muck PY, Homam MJ: Iot based weather station using Raspberry Pi 3. International Journal of Engineering & Technology. 2018, 7:145. 10.14419/ijet.v7i4.30.22085
8. Ahire DB, Gond VJ, Ahire NL: IoT based real-time monitoring of meteorological data: a review. SSRN Electronic Journal. 2022, 10.2139/ssrn.4043518
9. Bella HK, Khan M, Naidu MS, Digumarti SJ, Khan Y: Developing a sustainable IoT-based smart weather station for real time weather monitoring and forecasting. E3S Web of Conferences. 2023, 430:01092. 10.1051/e3sconf/202343001092
10. Morón C, Diaz JP, Ferrández D, Saiz P: Design, development and implementation of a weather station prototype for renewable energy systems. Energies. 2018, 11:2234. 10.3390/en11092234
11. Rao BS, Rao KS, Ome N: Internet of Things (IOT) based weather monitoring system. International Journal of Advanced Research in Computer and Communication Engineering. 2016, 5:312-319.
12. Viciana G, Juan A: A Modular Internet-connected Weather Monitoring Station at Haiti. Master's Thesis. Universidad Pontificia Comillas, Madrid; 2018.
13. de Jesús Medel Juárez J, Urbieta Parrazales R, Garduño Mendieta V: Meteorological portable system consulted via Wi-Fi. Computación y Sistemas. 2019, 23:1487-1497. 10.13053/cys-23-4-3261
14. Sivakumar B, Nanjundaswamy C: Weather monitoring and forecasting system using IoT. Global Journal of Engineering and Technology Advances. 2021, 8:008-016. 10.30574/gjeta.2021.8.2.0109
15. Shah S, Deshpande K, Jaiswal RC: IOT based smart city: Weather, traffic and pollution monitoring system. International Research Journal of Engineering and Technology (IRJET). 2017, 4:572-576.
16. Malik HAM, Shah AA, Muhammad A, Kananah A, Aslam A: Resolving security issues in the IoT using blockchain. Electronics. 2022, 11:3950. 10.3390/electronics11233950
17. Mohammad AS, Brohi MN, Khan IA: Integration of IoT and blockchain. Technium Romanian Journal of Applied Sciences and Technology. 2021, 3:32-41. 10.47577/technium.v3i8.4692
18. Rahman MS, Tumpa FA, Islam MS, Arabi AA, Hossain MSB, Haque MSU: Comparative evaluation of weather forecasting using machine learning models. 2024, 10.48550/arXiv.2402.01206
19. Kulkarni A, Mukhopadhyay D: Internet of things based weather forecast monitoring system. Indonesian Journal of Electrical Engineering and Computer Science. 2018, 9:555-557. 10.11591/ijeecs.v9.i3.pp555-557
20. Dedeoglu V, Jurdak R, Dorri A, Lunardi RC, Michelin RA, Zorzo AF, Kanhere SS: Blockchain technologies for IoT. Advanced Applications of Blockchain Technology. Kim S, Deka G (ed): Springer, Singapore; 2020. 60:55-89. 10.1007/978-981-13-8775-3_3
21. Abdul-Niby M, Farhat M, Abdullah M, Nazzal A: A low cost automated weather station for real time local measurements. Engineering, Technology & Applied Science Research. 2017, 7:1615-1618. 10.48084/etasr.1187
22. Fowdur TP, Sanghan A: Performance analysis of a blockchain enabled domestic power consumption monitoring and forecasting system. Sensor Review. 2024, 44:369-387. 10.1108/sr-01-2024-0045
23. Heston TF: A blockchain AI solution to climate change. International Journal of Science and Research Archive. 2024, 11:450-454. 10.30574/ijsra.2024.11.2.0471
24. Thapelo TS: A Programmable Tool-kit for a Smart Weather System Using Predictive Analytics at the Botswana. Master's Thesis. International University of Science and Technology, Botswana; 2020.
25. Tutorials Point - Arduino. https://www.tutorialspoint.com/arduino/index.htm.
26. Kanetkar Y: Let Us Java: Strong Foundation for JAVA Programming, 6th Edition. BPB Publications, Kolkata, India; 2022.
27. Apache Friends. XAMPP Installers and Downloads for Apache Friends. (2022). https://www.apachefriends.org/..