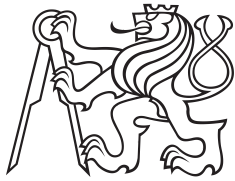


Master's Thesis



Czech  
Technical  
University  
in Prague

**F3**

Faculty of Electrical Engineering  
Department of Microelectronics

## Distributed network of meteostations with LoRa

**Bc. Vladyslav Larionov**

Supervisor: Ing. Vladimír Janíček, Ph.D.  
May 2021



## I. Personal and study details

Student's name: **Larionov Vladyslav** Personal ID number: **434954**  
Faculty / Institute: **Faculty of Electrical Engineering**  
Department / Institute: **Department of Control Engineering**  
Study program: **Cybernetics and Robotics**  
Branch of study: **Cybernetics and Robotics**

## II. Master's thesis details

Master's thesis title in English:

**Distributed network of meteostations with LoRa**

Master's thesis title in Czech:

**Distribuovaná síť meteostanic s LoRa**

Guidelines:

- 1) Analyze available solutions for monitoring environmental parameters and their centralized collection from a distributed network.
- 2) Design a device that will consist of wireless units powered by solar energy that will collect environmental data, sending it through the LoRa public gateways to the cloud storage.
- 3) Implement the designed device, verify its functions.
- 4) Monitor temperature, humidity, pressure and other key values of necessity.
- 5) Create an algorithm that will be able to tentatively predict the local meteorological situation from the measured data.
- 6) Share the collected data to community weather monitoring projects.
- 7) Compare with commercial product

Bibliography / sources:

- [1] Beginning LoRa Rasio Networks with Arduino, ISBN 978-1-4842-4357-2
- [2] IoT Networks with LoRaWAN, <https://leanpub.com/iot-networks-with-lorawan>
- [3] Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, ISBN-13: 978-1491962299

Name and workplace of master's thesis supervisor:

**Ing. Vladimír Janíček, Ph.D., Department of Microelectronics, FEE**

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **05.02.2020** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until:

**by the end of summer semester 2021/2022**

\_\_\_\_\_  
Ing. Vladimír Janíček, Ph.D.  
Supervisor's signature

\_\_\_\_\_  
prof. Ing. Michael Šebek, DrSc.  
Head of department's signature

\_\_\_\_\_  
prof. Mgr. Petr Páta, Ph.D.  
Dean's signature

## III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

\_\_\_\_\_  
Date of assignment receipt

\_\_\_\_\_  
Student's signature

## Acknowledgements

I wish to offer my sincerest gratitude to my supervisor, Ing. Vladimír Janíček, Ph.D., for his support, enthusiasm and insightful comments of the present thesis.

Also, I would like to thank my family and friends for supporting me throughout all my studies.

## Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 21, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 21. května 2021



## Abstract

The present work analyzes the possibilities of deploying a distributed network of meteorostations in an urban environment. The aim of this work is the design and implementation of a device with an emphasis on the simplest possible installation, minimum power consumption, wireless data transmission and the use of alternative power source. Also, within the framework of this work, an algorithm based on the LSTM neural network architecture has been implemented, capable of generating a forecast of the measured parameters.

In addition, an infrastructure was deployed on Amazon hosting, combining both centralized data collection from all devices, predicting measured parameters, sharing data with community weather monitoring projects, and, moreover, the web interface was implemented displaying both device data along with measured and predicted parameters. The developed system has been successfully tested in real climatic conditions. Finally, a comparative analysis of the developed device and commercial counterparts from the same and premium price segments was carried out. The result of the present work is a system with commercial potential and the ability to compete with popular existing solutions.

**Keywords:** Meteorostation, LoRa, WSN, LPWAN, LoRaWAN, Neural Networks, LSTM, AWS

**Supervisor:** Ing. Vladimír Janíček, Ph.D.

## Abstrakt

Současná práce analyzuje možnosti nasazení distribuované sítě meteorostanic v městském prostředí. Cílem této práce je návrh a implementace zařízení s důrazem na jednoduchost instalace, minimální spotřebu, bezdrátový přenos dat a využití alternativního zdroje energie. V rámci této práce byl také implementován algoritmus založený na architektuře neuronové sítě LSTM, schopný generovat předpověď měřených parametrů.

Kromě toho na hostingu Amazon byla nasazena infrastruktura, která kombinuje centralizovaný sběr dat ze všech zařízení, předpovídání měřených parametrů, sdílení dat s komunitními projekty monitorování počasí a navíc bylo poskytnuto webové rozhraní pro zobrazování měřených a předpovězených dat. Vyvinutý systém byl úspěšně otestován v reálných klimatických podmínkách. Nakonec byla provedena srovnávací analýza vyvinutého zařízení a komerčních analogů ze stejné a vyšší cenové kategorie. Výsledkem této práce je systém, který má komerční potenciál a je schopen konkurovat populárním stávajícím řešením.

**Klíčová slova:** Meteorostanice, LoRa, WSN, LPWAN, LoRaWAN, Neuronové Sítě, LSTM, AWS

**Překlad názvu:** Distribuovaná síť meteorostanic s LoRa

# Contents

<b>1 Introduction</b>	<b>1</b>	3.3.1 TTN side	69
1.1 Background	1	3.3.2 AWS side	70
1.2 Implementation analysis	2	3.4 Deployment	71
1.2.1 Operating environment and installation	2	3.4.1 Backend	73
1.2.2 Measured values and monitoring parameters	4	3.4.2 Frontend	77
1.3 Related works	6	<b>4 Data sharing</b>	<b>81</b>
1.3.1 Urban applications	6	4.1 Sharing data with Weather Observations Website	82
1.3.2 Industrial applications	6	4.2 Sharing data with OpenWeatherMap	82
1.3.3 Precision agriculture	7	<b>5 System evaluation</b>	<b>85</b>
1.3.4 Habitat monitoring	8	5.1 Results	85
1.3.5 Volcano monitoring	9	5.1.1 Device lifetime	85
1.3.6 Environmental monitoring	9	5.1.2 Operating environment	87
<b>2 Theoretical framework</b>	<b>11</b>	5.1.3 Weather forecast	89
2.1 Wireless Sensor Networks	11	5.1.4 Web page	91
2.1.1 WSN topology	11	5.2 Comparison with commercial products	92
2.1.2 End-node structure	13	5.2.1 Operation	93
2.2 Low-Power Wide-Area Networks	14	5.2.2 Price	96
2.2.1 LoRa	14	<b>6 Conclusion</b>	<b>99</b>
2.2.2 LoRaWAN	20	<b>A List of Acronyms</b>	<b>101</b>
2.2.3 The Things Network	25	<b>B Example of TTN uplink message</b>	<b>103</b>
2.3 Artificial Neural Networks	26	<b>C CD Content</b>	<b>105</b>
2.3.1 Basic concepts	26	<b>D Photos</b>	<b>107</b>
2.3.2 Recurrent Neural Networks	32	<b>E Bibliography</b>	<b>109</b>
2.3.3 LSTM Networks	33		
2.4 Amazon Web Services	37		
2.4.1 Global Infrastructure	37		
2.4.2 Services	38		
2.5 Solar Radiation	39		
2.5.1 Irradiance of a plane	40		
2.5.2 Extraterrestrial radiation	40		
2.5.3 Atmospheric optical depth	41		
2.5.4 Sun elevation angle	42		
<b>3 Implementation</b>	<b>45</b>		
3.1 Device Assembly	45		
3.1.1 Device Hardware	45		
3.1.2 Device Case	56		
3.1.3 Device Software	59		
3.2 Prediction Model	63		
3.2.1 Training dataset	63		
3.2.2 Data preprocessing	63		
3.2.3 Model architecture	65		
3.2.4 Compilation and training	66		
3.2.5 Model validation	66		
3.3 TTN-AWS Connection	69		

## Figures

1.1 Thesis structure . . . . .	1	3.6 Li-Po battery charge-discharge curve . . . . .	51
1.2 Private meteorostation mounting examples . . . . .	3	3.7 Li-Po battery protection board . . . . .	52
1.3 The Things Network coverage in Prague . . . . .	4	3.8 Battery protection circuit . . . . .	52
1.4 Examples of WSN in different applications . . . . .	8	3.9 Voltage divider circuit . . . . .	53
2.1 Basic Wireless Sensor Network topologies . . . . .	11	3.10 TP4056 Mini Lithium Battery Charging Control Board . . . . .	54
2.2 The structure of a typical wireless sensor node . . . . .	13	3.11 TPL5111 Low-Power timer . . . . .	55
2.3 LoRaWAN technology stack . . . . .	14	3.12 3D model of a meteorostation case . . . . .	56
2.4 LoRa upchirp and downchirp . . . . .	16	3.13 Device appearance . . . . .	58
2.5 LoRa Spreading Factor . . . . .	17	3.14 Assembled devices . . . . .	59
2.6 LoRa frequency bands . . . . .	18	3.15 TTN infrastructure comparison . . . . .	60
2.7 LoRa Frame structure . . . . .	19	3.16 Device program structure . . . . .	61
2.8 LoRa PHY Tx and Rx chains . . . . .	20	3.17 Machine Learning model implementation . . . . .	63
2.9 LoRaWAN communication model . . . . .	21	3.18 Dataset splitting . . . . .	64
2.10 LoRaWAN device classes . . . . .	23	3.19 Data normalization . . . . .	64
2.11 Security in LoRaWAN applications . . . . .	24	3.20 Data distribution over time intervals . . . . .	65
2.12 Device overview in the TTN Console . . . . .	26	3.21 Model structure . . . . .	65
2.13 Neural Network basic structure . . . . .	27	3.22 The training and validation loss while training . . . . .	66
2.14 Structure of the Human neuron . . . . .	27	3.23 Model performance in the test dataset . . . . .	68
2.15 Gradient Descent Algorithm . . . . .	28	3.24 Received uplink packets showed in the TTN Console . . . . .	69
2.16 Underfitting, overfitting and appropriate-fitting in machine learning . . . . .	29	3.25 TTN HTTP Integration . . . . .	70
2.17 Sigmoid and Tanh activation functions . . . . .	29	3.26 Database entity relationship diagram . . . . .	71
2.18 An unrolled Recurrent Neural Network . . . . .	32	3.27 Proxy server working principle . . . . .	71
2.19 LSTM cell with its internal structure . . . . .	34	3.28 Application deployment structure on the server . . . . .	72
2.20 AWS Global Infrastructure . . . . .	37	3.29 Forecast application workflow . . . . .	73
2.21 AWS Regions and Availability Zones . . . . .	38	3.30 Device states on the map . . . . .	78
2.22 Solar geometry . . . . .	42	3.31 Device info . . . . .	78
3.1 Scheme of the assembled device . . . . .	46	3.32 Generated forecasts . . . . .	79
3.2 BME280 digital sensor module . . . . .	48	4.1 Shared data showed on the WOW . . . . .	82
3.3 BME280 measurement cycle . . . . .	48	4.2 Workflow of sharing the data with OWM . . . . .	83
3.4 The principle of trilateration . . . . .	49	5.1 Deployed sensor nodes . . . . .	88
3.5 Ublox NEO-6M GPS module . . . . .	50	5.2 Maximum data transmission range . . . . .	88
		5.3 Comparison of forecast model performance . . . . .	89
		5.4 Main web page appearance . . . . .	91

5.5 Main page appearance on mobile device .....	92
5.6 Commercial products selected for comparison .....	92
D.1 Device assembly process and nodes deployment .....	107

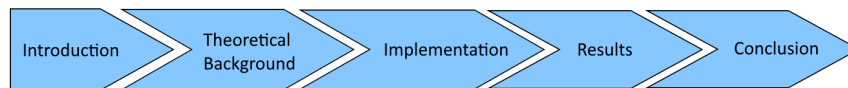
## Tables

3.1 BME280 technical data .....	49
3.2 TP4056 Technical data .....	54
3.3 ESP32 Low-Power Modes .....	55
3.4 Mean errors in the test dataset for 1-hour forecast .....	67
3.5 Improvement in prediction accuracy .....	67
3.6 API description .....	75
5.1 Measured parameters comparison between developed station and the commercial ones .....	93
5.2 Calculation of the final price of one meteostation unit .....	96
5.3 Comparison of the final price of the assembled device with commercial products .....	97

# Chapter 1

## Introduction

The present thesis can be divided into five logical parts. The first part is introductory, describing the research question itself and containing both a preliminary analysis of the solution and the study of the problem as a whole based on existing solutions. The next part deals with a detailed analysis of the methodology used. Then, it is followed by the main part describing the particular implementation of each component of the whole solution. Finally, the results of the work and the goals achieved are summarised.



**Figure 1.1:** Thesis structure

## 1.1 Background

For centuries, people have been interested in the surrounding climatic processes. This interest carries both exploratory and industrial components, since changes and development of climatic parameters over time serve not only as a source of scientific research, but also as a criterion by which planning decisions can be made in both the urban and agro-industrial sectors, depending on the measured local environmental parameters.

Usually, regardless of the ultimate goal of the study, the concept of collecting and then analyzing data is the same. Firstly, the data is measured with sensors and then analyzed in different ways. In a number of applications, a forecast of future values is built on the basis of the collected data. At the moment, most applications for the prediction of meteorological parameters are based on the so-called *Numerical Weather Prediction (NWP)* which is based on a mathematical model of the ocean and atmosphere and uses new measured values to generate predictions of future values. But now, with the development of neural networks, deep neural networks are gaining more and more popularity in meteorological applications [1].

But still, the process of measuring and collecting data, regardless of the type of downstream analysis, starts with sensors, represented as devices (i.e. end nodes). Data collection from such devices most commonly occurs in

a centralized manner. Depending on the project goals and environment of operation, each end node has corresponding specifications.

From the moment the first wireless data collection systems appeared until today, developers have faced the same tasks: data transmission to the maximum range while reducing energy and total device price, useful signal isolation against the background noise, and an increase of flexibility of application in general. The development and year on year increasing utilization of the *Internet of Things (IoT)* technology has put forward such requirements for systems as maximum autonomy, interconnection of devices in a network for centralized collection of information combined with the compactness of the devices themselves. When building *IoT* systems, devices with radio frequency transmitters are often used. These devices are capable of transmitting the measured data to an intermediate station (i.e. gateway), which, in turn, transmits the data further to the server and to the global network, where the data is then analyzed and evaluated. There is a wide variety of radio modules and protocols, each suitable for a particular application. Among the most famous and most used are such standards as *WiFi*, *Bluetooth*, *LoRa*, *SigFox* and *ZigBee* [2].

## ■ 1.2 Implementation analysis

The development of any device starts with planning and design, with drawing up requirements and analyzing the future solution. Implementation of a weather station with subsequent data processing is a complex task, especially when it comes to maximum autonomous operation and minimization of the requirements for maintenance and installation of the device. The task of planning the device implementation is to find a balance between the requirements for measuring environmental parameters, the maximum accuracy of measurements, as well as the operational capabilities that impose their own limitations [3], such as the reach of public gateways and the mounting of meteorostations in unprepared urban conditions.

### ■ 1.2.1 Operating environment and installation

It is assumed that the devices will be located in an urban environment, which is very challenging [4]. Unlike rural areas, where it is possible to install the device either in an open area in the backyard or on the roof of your own summer cottage, in a city the number of potential places for mounting a station is very limited. Ideally, the device should be located on a rooftop where there are good conditions for measuring wind direction and wind speed. But there is not always access to the roof. But even if there is access to the roof, many buildings in Prague still have a ceramic roof covering and a triangular shape, where the possible area for mounting our future device is minimal. In addition, not everyone has balconies at home - another potential installation location for the device.

Usually, for the installation of both professional and amateur weather stations, several types of steel or aluminum mounts are used. Such mounting solutions firmly fix the station during wind and precipitation. For example, tripod mounts that are mounted on a horizontal surface.

Alternatively, side or chimney mounts that are installed on the side of the building facade and on the chimney, respectively. The so-called masts are also used, representing the pipes on which the meteorostation is mounted [5].

All of the above types of mounts require additional calculations and installation, making these stations static, not allowing potential customer to quickly redeploy it to a new location. In addition, we must not forget that many people do not have access to the roof or the use of separate mounts is impossible in the urban environment of a particular person's residence place, for example, it is difficult to imagine an installation of a side mount for a meteorostation on the facade of a panel house, moreover, this will entail additional costs.

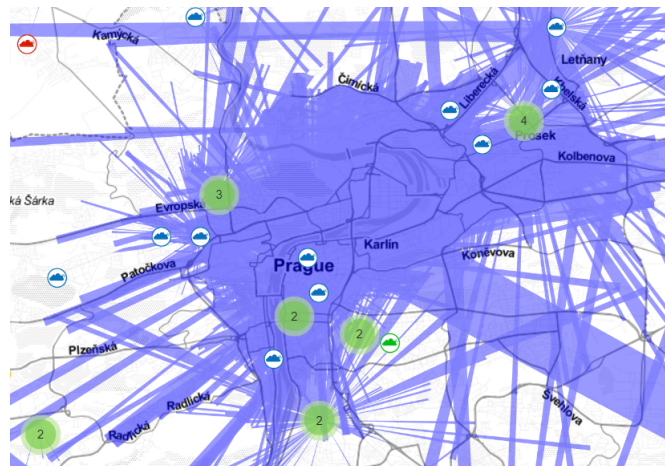


**Figure 1.2:** Private meteorostation mounting examples [5]

Besides the lack of mobility, this solution has the disadvantage of maintenance complexity, since in case of any malfunctions the owner will have to get to the station for servicing, which is not literally the *"open my window and immediately took out the station"* process. It is necessary at least to disassemble the lock holding the station on the mount.

Based on all of the above, it is necessary to develop a device that is as easy as possible to install, is very compact and does not require additional investment. This device should be suitable for limited urban conditions, where not every station owner has access to the roof or other suitable place to install the station. In addition, communication with the station should be independent of the location, and the power supply should be autonomous and energy efficient. Most modern commercial stations use *Wi-Fi* protocol for data transmission. The two main drawbacks of this protocol are its attachment to a specific location with the need to reprogram the device during relocation, and its power consumption, which is not optimal for *IoT* devices. Therefore, the *LoRa* protocol was chosen for communication.





**Figure 1.3:** The Things Network coverage in Prague [6]

Despite the fact that here in Prague there are not so many public gateways from *LoRaWAN* providers, as for example in Berlin or Zürich, but nevertheless this number is enough to deploy a network of devices that will use the public infrastructure of gateways.

### 1.2.2 Measured values and monitoring parameters

There are many commercial meteorostations available on the market, differing both in price, size, level of autonomy, as well as in measured parameters. One can choose both a meteorological station that measures only the air temperature, and meteorological station that in addition to temperature measures relative humidity, atmospheric pressure, wind speed and wind direction, as well as the amount of precipitation.

The key point when choosing the measured values by the station is finding a balance between two limiting factors: the possibility of station mounting in an urban environment, which was described above, and the minimum requirements for the location of sensors in a personal meteorostation in order to ensure the correct measurement of the selected environmental values. The second factor is no less important than the first, since it is assumed that the weather station will transmit correct data to the server. Otherwise, incorrect data will affect all downstream data processing.

According to the recommendations from the *National Oceanic and Atmospheric Administration (NOAA)*, the correct outdoor meteorological values of personal weather stations can be obtained if the following requirements for installing individual sensors are met [7]:

- *Temperature/Humidity sensor* - location height from 1.2 to 2.0 meters above ground level. The temperature sensor should not be exposed to direct sunlight. The moisture sensor should not be placed near plants or other moisture-containing objects that could falsify measurements. Both sensors must have access to open air for measurements.



- *Anemometer* - location height 10 meters above ground level. Although, multipliers' table for locations below this recommended height exists to compensate for lower height anemometer [8] , but the measured data will still be approximate. In addition, it is necessary to maintain a horizontal distance from the nearest interference objects that can distort measurements, which is rather difficult in urban conditions. For example, placing an anemometer in an alley results in a wind tunnel effect. The perfect location for this sensor is still on a building roof.
- *Rain Gauge sensor* - location height from 1.2 to 1.8 meters above ground level. Also, an important requirement is to locate the sensor away from any horizontal surfaces that could introduce distortion caused by rain-splashing or surrounding snow buildup. In addition, the neighborhood with any buildings leads to the so-called "shadows". It means that when the sensor is placed in the shadow of the building and in case of wind, the building can partially or completely block the sensor from the rain, thereby the sensor will miss a lot of falling rainfall.

Also, in addition to the above points, it is necessary to take into account the scenario of using the meteostation, or, more precisely, the network of meteostations. Since the goal is not to measure all possible meteorological parameters, but to measure a *sufficient* number of parameters for the subsequent local meteorological situation forecast creation. In addition, the last, but therefore no less important point is the profitability and autonomy of the entire system, since each additional sensor leads to an increase in the total price of the entire device (in our case, a network of devices) as a whole and an increase in power consumption.

So, proceeding from all the points described above, it was decided to choose air temperature, atmospheric pressure and relative humidity as measured values. These parameters are key and sufficient to describe the local meteorological situation and to create a model capable of predicting future values. In addition, meteorological values such as dew point, frost point and saturation deficit can also be calculated from these measured parameters. The anemometer readings are not necessary for the purpose of predicting the local meteorological situation within the framework of this project, since to correctly account for all possible parameters related to the direction and speed of wind in meteorology, complex models are used that perform calculations on high-performance computer clusters, which would greatly increase the entire project cost. Also, the anemometer, as well as the rain gauge sensors, must send correctly measured data due to the correct installation of the meteostation, according to the points described above. But it is very difficult to achieve in a city. In conclusion, these sensors will increase the total cost of the system and the energy consumption, which, together with the points described above, makes the integration of these sensors unnecessary.

Therefore, the purpose of this work is to develop a device that allows for the simplest possible installation in a city, maximum autonomy and protection from environmental influences. The potential owner of the developed

meteorological station should obtain a system where its devices correctly measure selected parameters, measured parameters forecast is generated and displayed together with all relevant information on a user-friendly interface. The time spent on servicing the device and the money spent by a potential customer should be minimized.

In addition to meteorological parameters, to ensure the use of meteorostations in different locations without additional preparation, it was decided to add a *GPS* module to determine the current location of the device. Also, it is necessary to monitor the battery voltage level in order to replace/recharge the battery in time. Finally, special attention should be paid to the device case, since it should include both a mount for a weather station, protection of boards from external factors during outdoor use, as well as a solar panel for charging the battery.

## 1.3 Related works

Thanks to their versatility, wireless sensor networks are becoming more popular every year and find new areas of application. The unambiguous advantages of such networks are autonomy and freedom in choosing the deployment location, since there are no restrictions caused by wires for communication with the server and for power supply. Among the most interesting and well-established use cases of centralized data collection by using such networks are the following projects.

### 1.3.1 Urban applications

In urban environments, sensor networks are used in dozens of applications ranging from gas and smoke measurements to corrosion detection. Built-in systems called intelligent lighting are gaining more and more popularity. Sensors in these systems measure the presence of traffic or people and the level of environmental lighting. Based on the collected data, sensors control the lighting level of specific locations, thereby significantly optimizing energy costs. As an example of use in a city is a system deployed in Hungary in Budapest for environmental monitoring [9]. This sensor network was implemented using star topology. Each end device measures relative humidity, atmospheric pressure, particulate matter and other parameters that are sent to the data collection server for storage in the database. The very process of data analysis takes place in the cloud. The product of the analysis is a short-term forecast of particulate matter levels, the result of which are displayed on a particulate matter map.

### 1.3.2 Industrial applications

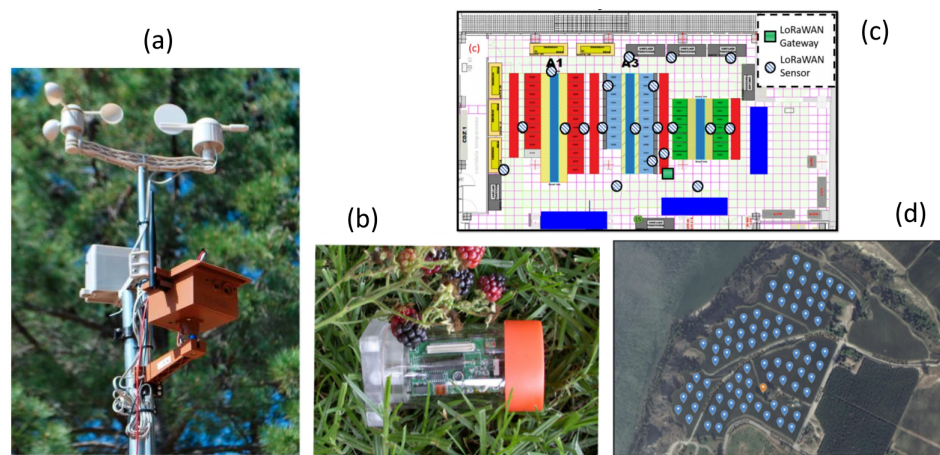
This area includes both robotics, where a terrain map is built using sensors from the sensor network on the basis of which a robot then builds a path using internal algorithms, and various models that evaluate the state and

performance of technical equipment, and also monitor the state of the selected parameters - system voltage or how much the device heats up during operation. For example, a *Wireless sensor network (WSN)* system based on *LoRaWAN* architecture has been proposed for monitoring temperature in a data center [10]. Large data centers consist of hundreds of high performance nodes that are used in scientific and industrial applications to calculate models of varying complexity. In addition to the increased power consumption compared to conventional computers, these data centers must also solve the problem of overheating of the system, which can lead to all sorts of malfunctions. The power consumption of the computing units themselves is about half of the total power consumption of the data center. The second half of the consumption comes from cooling systems such as *Computer Room Air Conditioning* or *Computer Room Air Handlers* units. The authors of this work designed a *WSN* capable of monitoring the temperature and humidity of the air inside the data center in a distributed manner. The collected data was then analyzed to optimize the performance of the cooling mechanisms. Initially, the authors planned to use the *Wi-Fi HaLow* or *Zigbee* protocol, but in the end the choice fell on the *LoRa* protocol, since it is the best option for transferring small data packets in terms of energy efficiency and data transmission distance. Star topology was chosen as the network topology. The competing topology was the mesh topology, but here in order to reduce the likelihood of data loss, some end devices must be programmed as routers transmitting foreign packets, leading to the increase in total energy consumption and, accordingly, to the decrease in system autonomous operation lifespan. The authors calculated that their solution based on the star topology and the *LoRa* protocol consumes almost half less current than a similar solution using the mesh topology and *Zigbee* protocol. The number of *LoRaWAN* nodes deployed at key locations of the data center was twenty. The device itself was based on the *STM32L4* microcontroller from *STMicroelectronics*. The operation mode consisted in measuring temperature and humidity and sending data to the server with a period of every 30 seconds. Data from all devices was collected centrally in the *InfluxDB* database and then separately displayed in the *Grafana* dashboard. In addition to the measured data, information about the state of the end device was also sent, such as the battery level. The consumption of the end device in sleep mode was  $4 \mu A$ . Each node was powered by a  $1000 mAh$  battery that, according to the authors of the study, extended the node operational time to seven months, in contrast to the alternative system based on the *Zigbee* protocol that could work autonomously for only seven days. This example confirms that *LoRa* wireless sensor networks are an inexpensive and effective solution for monitoring environmental parameters such as temperature and humidity.

### ■ 1.3.3 Precision agriculture

A relatively new trend in the use of wireless sensor networks is the so-called precision agriculture, which is an approach in agriculture that uses collected information from a sensor network to provide optimal conditions for the

maturation and growth of crops. In [11], a working prototype of such a network was created for use in *The Okanagan Valley* in Canada. The network consisted of many end devices based on the *Feather M0 LoRa* module, each of which sent temperature, humidity and pressure data from the *BME280* digital sensor, as well as its geographical coordinates and battery voltage level to the gateway. The central single-channel gateway was implemented using the *Raspberry Pi 3 B+*. Also, due to the high density of nodes and the presence of only one gateway, the *Transmission Queue* was developed, which assigned windows for data transmission for each device. All data was collected centrally in the *MySQL* database. A local dashboard for data visualization was also implemented, in which it was possible to select the values measured by specific devices, for each of which separate graphs of the last measured data were built. In subsequent work, authors plan to use the collected data for disease modeling.



**Figure 1.4: Examples of WSN in different applications:** Illustration of deployed node in the Doñana National Park (a)[14]. Acrylic enclosure used for deploying the node on Great Duck Island (b)[12]. Sensor nodes deployment in Data Center (c)[10] and in a grape vineyard in The Okanagan Valley (d)[11].

### 1.3.4 Habitat monitoring

As part of the following study [12], a distribution network for habitat and environmental monitoring was developed on *Great Duck Island*. Researchers have studied the effect of environmental parameters on seabirds nesting during the breeding season. It was necessary to come up with the most autonomous and non-invasive solution, since seabirds differ from other bird species in their sensitivity to human disturbance. For example, it was found that even a 15 minute human visit increases the mortality of eggs and chicks by 20%. The network itself consisted of 32 end devices intended for localized measurements, which were distributed into separate clusters. Each cluster had its own gateway, forwarding data packets to the central base-station, which in turn was connected to the Internet via a two-way satellite connection. The gateways and base-station were powered by a solar panel. The end device itself

was based on the *Mica* weather board with *Atmel Atmega 103* microcontroller running at 4 MHz and integrated 916 MHz radio. The measured values were light intensity, temperature, barometric pressure and humidity. Each device was powered by the pair of conventional AA batteries and collected environmental data about its immediate surroundings. It was calculated that after all the optimizations made, the average consumption of one device is *6.9 mAh* per day. According to the researchers, it should be enough for more than 300 days of operation, fully meeting the requirements of the research project. Data storage was performed on the base-station side in the *PostgreSQL* database. The data from the sensor nodes were sent to the base station, from where they were synchronized via satellite connection with the base in the laboratory in *Berkeley*, where they were then analyzed in Matlab. A weatherproof case for the device was also developed allowing the end device to work in outdoor conditions. The case itself was implemented as an acrylic tube in which the control module is placed along with the sensors.

### ■ 1.3.5 Volcano monitoring

Within the framework of this study [13], the *WSN* was deployed on an active volcano. The purpose of this study was to collect and analyze longterm trends on the *Volcán Reventador* volcano located in northern Ecuador. The network consisted of 16 *MSP430* microcontroller based end devices, each equipped with a seismometer, microphone and *Chipcon's CC2420* 2.4 GHz RF transceiver. The end devices transmitted the measured data through the multihop network to the central gateway, which in turn transmitted the data using the *Free Wave* radio modem utilizing frequency hopping spread spectrum (FHSS) technology for long-distance communication to the base station. Each device was powered by a pair of alkaline D cell batteries, which resulted in two battery replacements in end devices over three weeks of research.

### ■ 1.3.6 Environmental monitoring

Another study [14] suggested a way in which large messages (i.e. images) can be sent using *LoRa*. A so-called *Wireless Multimedia Sensor Network (WMSN)* has been implemented at *Doñana National Park* in Spain. In total, the network consisted of 10 nodes. Each device, in addition to sensors for measuring environmental parameters, had a microphone and two cameras. Another feature of the solution was the presence of two independent processors in each device: a low power microcontroller (based on *STM32L162*) and a high-end multimedia processor (based on *Raspberry Pi*), capable of local processing of multimedia data. The *Raspberry Pi* in this circuit works as an *STM32L162* peripheral allowing to use *Raspberry Pi* according to needs, or completely disconnect it from the power supply. Radio communication was based on *LoRa* technology using the *SX1276* module. Thanks to *LoRa* high radio range, each end device could send data directly to the base station. But communication between individual nodes was also implemented to distribute the data preprocessing load between end devices. The power supply was

implemented in the form of solar panel. The case was designed and printed on a 3D printer. Not a single device failed during the year of operation.

## Chapter 2

### Theoretical framework

In this chapter we will take a closer look at the theory used both during the creation and deployment of end nodes and machine learning model implementation.

#### 2.1 Wireless Sensor Networks

We can define a *Wireless Sensor Network (WSN)* as a wireless network of many sensors that do not require a prepared infrastructure and that use a radio channel to receive or send data [15]. Sensors in such networks usually measure physical parameters, e.g. movement, or environmental parameters such as temperature, humidity, sound level or pressure. Measured values from each device are sent to a server or cloud storage for subsequent processing. Typically, *WSNs* consist of tens or hundreds of end devices.

##### 2.1.1 WSN topology

Once a device is deployed to work, it self-identifies within a defined topology, thereby creating an infrastructure from scratch. There are several basic types of topologies used in wireless sensor networks [16].

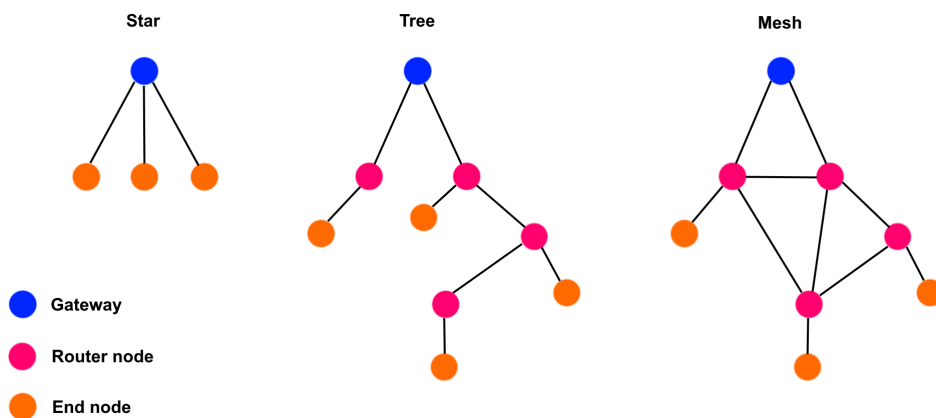


Figure 2.1: Basic Wireless Sensor Network topologies



### ■ Star topology

This topology is characterized by the fact that one base station can send or receive data from several end devices. In addition, within this topology, end devices are not allowed to exchange messages with each other. The advantage of networks of this topology is the simplicity and low latency communication between end devices and the base station. Furthermore, each device is independent from the others, which makes this network sustainable to the failure of one or more end devices, because the data will still be sent by remaining nodes. Also, the energy consumption of the end device in this type of network is reduced to zero, since the device is only responsible for sending its own data packet, without the need to transmit packets from other devices. The disadvantage of this topology is that at least one base station must be within the range of the transmitted signal of the end device.

### ■ Tree topology

It is a hierarchical architecture where all sensors form a logical tree. In this topology, each end device is either a leaf node or a parent node. Measured data is passed up the hierarchy from children to their parents all the way to the gateway. The advantage of this topology is lower power consumption than other architectures (ignoring the star topology). Also, fault detection is fast. Among the disadvantages of this topology, one can single out a longer packet delivery time, since the data must pass through all the end devices of the same group until it reaches the goal. Also, if the network is built in such a way that the communication of several nodes passes through only one node, which is within the radius of the signal of these nodes and resends their packets further, then in the event of a breakdown of this connecting node, several end devices will be cut off from the network at once.

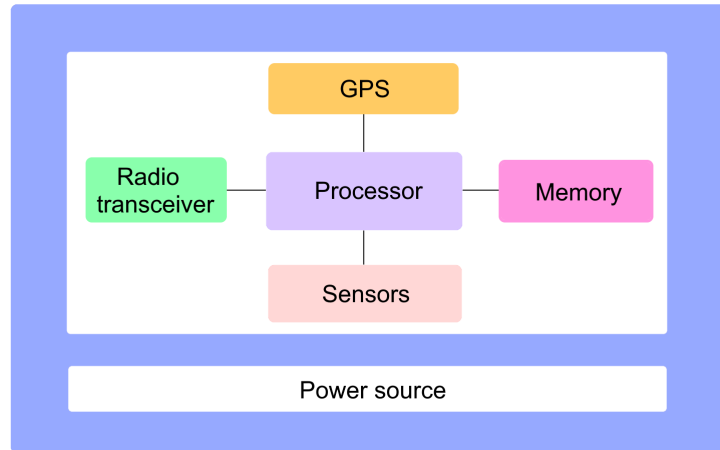
### ■ Mesh topology

It implements the so-called multi-hop communication, in which data packets can be transmitted from one end device to another, located within the radius of the transmitted signal. The advantage of this topology is deployment, which does not depend on the fact that at least one base station is within the signal radius of the end device. A node can transmit a packet with measured parameters through an intermediate node, which, in turn, will redirect the packet to the base station. It also follows that this topology is highly fault tolerant. The disadvantage of this topology is the higher power consumption of end devices implementing multi-hop communication, which can be critical from the point of view of low-power applications. In addition, the more intermediate nodes transmit the packet to the base station, the greater the delay in delivering the packet to the central server. Overall, mesh topology is difficult to design and maintain.



### ■ 2.1.2 End-node structure

The structure of the sensor node can be roughly represented as a control unit (i.e. processor), which is responsible for measuring data by sensors and handling the data transfer. The entire device module is powered by an external power supply [16].



**Figure 2.2:** The structure of a typical wireless sensor node

End devices are limited in both the data transfer rate and the bandwidth of the communication channel. Moreover, such devices usually have very limited memory resources.

There are many requirements and constraints that must be taken into account during the design of the wireless sensor network in general and a specific end device in particular [15]:

- *Energy supply* - to maximize the operating time of the device, we need to solve the problem of power consumption. In networks of this kind, the emphasis is put on renewable power supplies, since the operating conditions of the network often do not allow manual interventions and adjustments. The power supply must take into account the geographical features of the deployment of the device, as well as the requirement to provide sufficient power for the continuous operation of the entire device, including sensors and radio transceiver.
- *Design and hardware* - each end device contains at least a sensor, a power supply, a control unit and a module for data transmission. In addition, the nodes can contain additional sensors or *GPS* modules. Each additional function of the end device means increased energy costs, increased price and increased size of the device itself. Also, the design of the device must be suitable for the site of use, taking into account the way the device is mounted. The task of the developer is to find the right balance between the device requirements for a particular project and the constraints that are imposed when working with devices of the category of *Internet of Things*.

- *Price* - is an important criterion, since many use-cases of using *WSN* assume that end devices will be deployed only once and work until they fail, for example, networks for detecting forest fires. In this case, the price of one device should be as affordable as possible to compete with traditional data collection models.
- *Scalability* - wireless sensor networks can consist of a pair of end devices, or hundreds or even thousands. Therefore, the architecture of such networks should take into account the possible increase in the number of nodes by several times.

## 2.2 Low-Power Wide-Area Networks

A *Low-Power Wide-Area Network (LPWAN)* is a type of wireless telecommunication wide area network that was designed specifically for long-range communication of things (i.e. sensor nodes powered with a battery) at a low bit rate. Sending data from devices usually occurs a couple of times per hour, and the devices themselves are designed in such a way that the battery life is as long as possible (from 2 years or more) [17].

*LoRa* operates on the license-free radio spectrum and is classified as a non-cellular *LPWAN* protocol. *LoRa* allows one to transfer data over very long distances while consuming a very little amount of power. *LoRa* wireless technology itself consists of two parts, each of which implements a certain layer according to the *OSI* model [18]:

1. Implementation of the *Physical layer (PHY)*.
2. Implementation of the *Medium Access Control (MAC)* layer within Data Link layer together with organization of system architecture of the network.

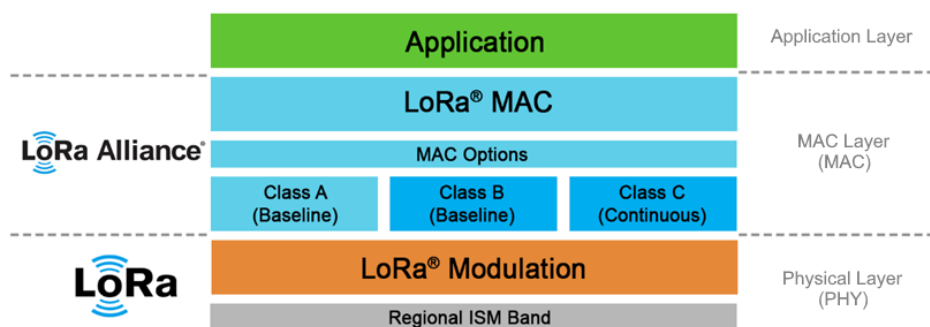


Figure 2.3: LoRaWAN technology stack [18]

### 2.2.1 LoRa

*LoRa* (i.e. *Long Range*) is a propriariety modulation method based on the *Chirp Spread Spectrum (CSS)* technique, in which data symbols are modulated

into chirps [18]. This technology was originally developed in the 1940's for use in military and marine radars.

A symbol means one or more data bits:

$$Symbol = 11100101$$

Different types of modulation encode different numbers of bits into one symbol. For example, *Quadrature Phase Shift Keying (QPSK)* modulation encodes two bits into one symbol, while *Phase-shift keying (PSK)* modulation encodes only one bit into one symbol. Bit rate  $R_b$  defines the number of bits per second, while symbol rate  $R_s$  (i.e. baud rate or modulation rate) defines the number of symbols per second.

It is also worth defining two concepts separately: chirp and chip. Despite the fact that these terms are consonant, they have separate definitions. The chirp (*Compressed High Intensity Radar Pulse*) is represented as a signal of continuously increasing or decreasing frequency, which is used in *CSS* modulation to represent symbols (i.e. chirp = symbol), as opposed to, for example, the aforementioned *QPSK*, where the symbol is represented by sinusoidal wave. Chip is a bit in a data sequence. Chip rate  $R_c$  is represented by the number of chips per second. By definition, the value of the chip rate is greater than the symbol rate, which means that one symbol contains many chips. Another parameter is the spreading factor that defines the ratio of the chip rate to the symbol rate.

Traditional *Direct Sequence Spread Spectrum (DSSS)* modulation works in such a way that the original signal containing the data changes the phase of its carrier signal when multiplied with the chip sequence (code sequence). Chip sequence's rate is much faster than the original signal rate, thereby achieving of spreading the bandwidth of the transmitted signal beyond the bandwidth of the original signal. An increase in the radio link budget is also achieved, allowing data to be transmitted over longer distances. Demodulation on the receiver side is performed in a similar way: the received signal is multiplied with a locally generated chip sequence, which should be similar to the sequence used for modulation on the transmitter side. By multiplying the spread signal with the spread bandwidth, the received signal is compressed into the original signal with the original bandwidth. This solution has a big drawback in the high requirements for the accuracy of the reference clock, which is reflected in the price [18].

In LoRa modulation, spreading the data through a bandwidth occurs by chipping the data signal at a higher data rate and then modulating onto the chirp signal, which constantly changes its frequency. *Log10* ratio of "*chips per bits*" defines the amount of spreading and is called processing gain  $G_p$ . This gain is expressed in *dB* and can be described using the equation [21]:

$$G_p = 10 \cdot \log_{10} \left( \frac{R_c}{R_b} \right) (dB) \quad (2.1)$$

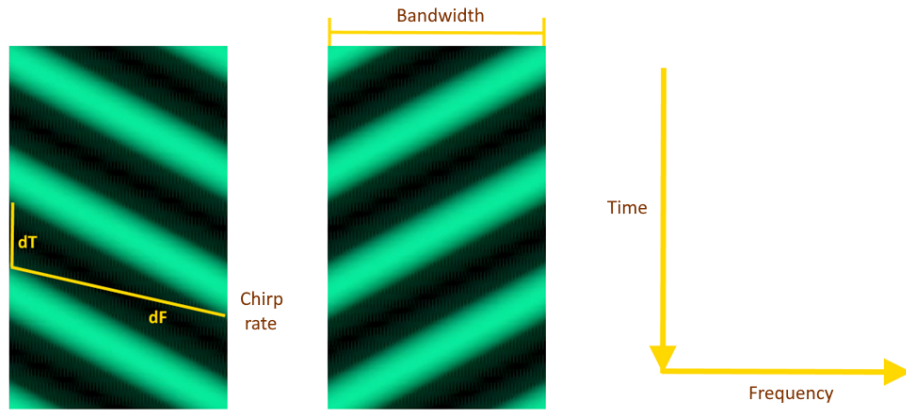
where:

- $R_c$  = chip rate (chips/second)

- $R_b =$  bit rate (bits/second)

This gain determines the receiver's ability to extract the original data signal in conditions of large negative *Signal-to-Noise Ratio (SNR)* values on the channel. *LoRa* has a high processing gain, allowing the transmitter to transmit a signal with less power while maintaining the same radio link budget and signal data rate.

Thanks to this solution, *LoRa* devices have high noise immunity: signal demodulation is possible at a level of  $20\text{ dB}$  below the noise level. The signal frequency during transmission changes linearly (the frequency can change both up-chirp up and down-chirp down) from its initial value  $f_0$  to the final  $f_1$ . This technique allows one to greatly reduce the requirements for the receiver due to the same timing and frequency offsets on both sides of the transmitter and receiver. The frequency offset without distorting data extraction from the received signal can reach as much as 20% of the bandwidth [22]. This makes *LoRa CSS* technology an excellent price, robust and low-power alternative to *DSSS*.



**Figure 2.4:** Upchirp (on the left) and downchirp (on the right) captured on an oscilloscope

Among the advantages of *LoRa* modulation are: resilience to interference, low-cost, low transmission power requirements, resistance to multi-path (making it especially good in urban applications), resistance to fading and Doppler effect. The disadvantage is low data rate (between  $27\text{ kbps}$  and  $50\text{ kbps}$ ).

### ■ Modulation parameters

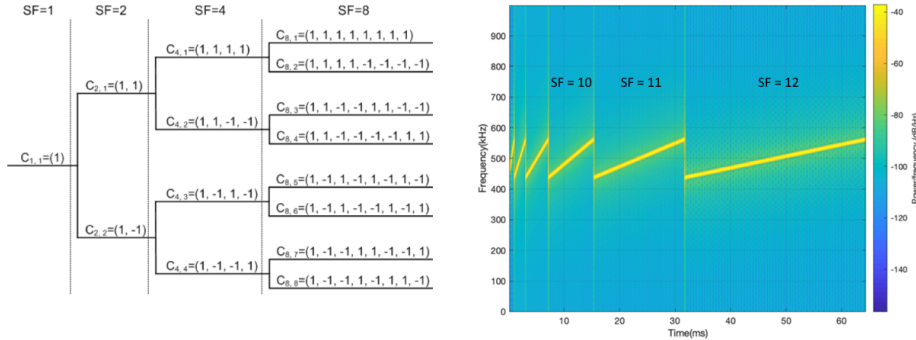
The key parameters of *LoRa* modulation are *Bandwidth (BW)*, *Chirp Rate (CR)* and *Spreading Factor (SF)*.

Bandwidth is the width of the spectrum occupied by the chirp. The channel bandwidth is determined by the difference between the frequencies  $f_0$  and  $f_1$ , which is fixed and can acquire values of  $25\text{ kHz}$ ,  $250\text{ kHz}$  and  $500\text{ kHz}$ .

Spreading Factor (can be from 7 to 12) is the number of raw bits (i.e. chips) encoded per symbol [21]:

$$SF = \log_2 \left( \frac{R_c}{R_s} \right) \quad (2.2)$$

Based on the above definition, we can say that one symbol holds  $2^{SF}$  chips. That is, using the example at the beginning of the chapter, we can say that the number of encoded bits is 8 meaning that a symbol consists of 8 bits. This fact can be written as Spreading Factor is equal to 8.



**Figure 2.5:** Orthogonal variable spreading factor (on the left)[19]. Chirp modulated with different spreading factors (on the right)[20]

LoRa uses orthogonal spreading factors to prolong battery life of end devices, as well as to prevent collisions and increase channel throughput. Packets using different  $SF$ s are invisible to each other, which allows the receiver to demodulate packets received by the same channel, but modulated using different  $SF$ s. Orthogonal spreading factors also help to optimize the data rate and power level of a particular connected device. This happens in such a way that devices located closer to the gateway must send data using a lower spreading factor (higher data rate), since a high spreading factor is not needed due to the close distance to the gateway. Conversely, devices further away must use a higher spreading factor (lower data rate), which increases processing gain. From which we can conclude that sending the same amount of data with a higher spreading factor will take more transmission time, which is designated as *air time*, but at the same time will increase the transmission range. The air time value affects both the device and the reception of information by the gateway:

- For a device, a higher air time means a higher energy consumption due to a longer up time.
- However, for a gateway, a higher air time means a better chance for a gateway to sample the signal power. This, in turn, means an increase in sensitivity and, consequently, an increase in coverage distance between transmitter and receiver.

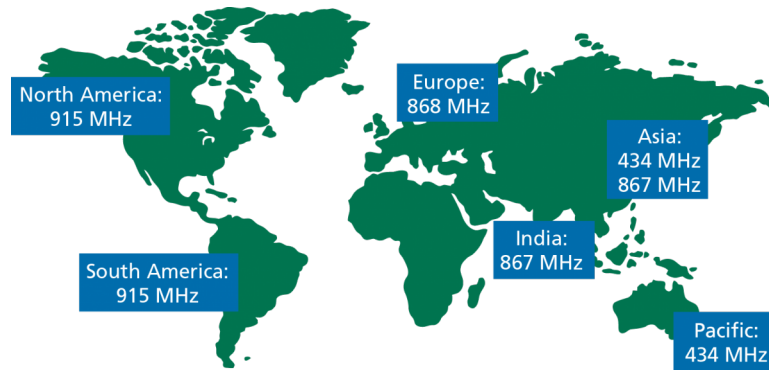
All the main relationships between the parameters used in *LoRa* modulation can be described using these three equations [21]:

$$R_b = SF \cdot \frac{BW}{2^{SF}} \quad R_s = \frac{BW}{2^{SF}} \quad R_c = R_s \cdot 2^{SF} \quad (2.3)$$

From which it follows that the chip rate depends only on the bandwidth. Also, the receiver sensitivity for *LoRa* modulation is defined as:

$$Sensitivity = -174 + 10\log(BW) + NF + SNR \quad (2.4)$$

where  $NF$  is a constant of the *Noise Floor*, which is defined by a particular receiver.

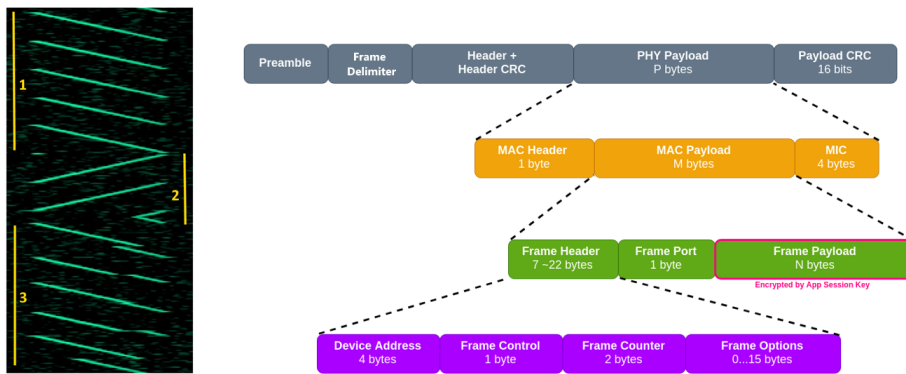


**Figure 2.6:** LoRa frequency bands [21]

*LoRa* is defined as an implementation of the physical layer according to the *OSI* communication model. That is, in fact, *LoRa* utilises air instead of physical transmission medium, through which radio signals are sent between the transmitter and receiver using *LoRa*. Transmitter in this case is represented by the end device, and the receiver is represented by the gateway. *LoRa* as a modulation technique is patented by Semtech company. Semtech transceiver microcircuits (i.e. *SX1276*) provide operation at the physical layer (*PHY*) of *LoRa*. In different geographic areas of the Earth *LoRa* works in different frequency ranges.

## ■ Frame format

Modulated data is transmitted in frames. *LoRa* defines the frame structure at both the *Physical Layer*, *MAC Layer* along with *Application Layer* [22]:



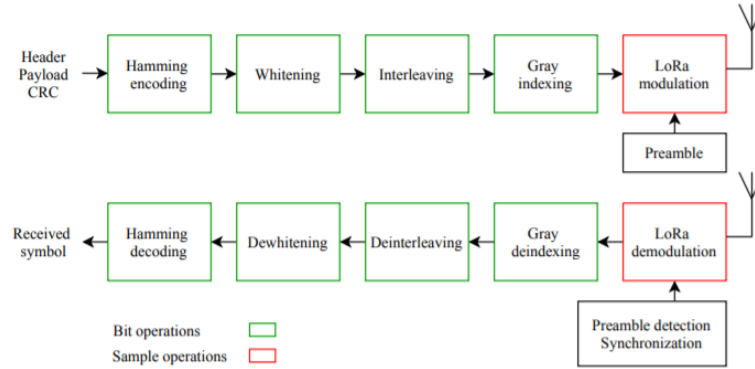
**Figure 2.7:** LoRa Frame structure captured on an oscilloscope (on the left) and represented by a block diagram (on the right)[24]

- *Physical Layer Frame* starts with a preamble. The preamble helps the receiver to identify *LoRa* signal, it says "attention, data transmission will begin now". The preamble performs the function of synchronization and is represented by several repetitive upchirps. In addition, the preamble defines the modulation scheme of the packet, since it is modulated with the same spreading factor as the entire data packet as a whole. Then, after the preamble, a *frame delimiter* starts, lasting roughly 2.25 downchirps. The meaning of frame delimiter can be described as "the preamble is over, the actual data will begin now". Finally, the last piece of the frame is the data provided by the choppy upchirps of varying length. The data can be divided into *PHY Header* and a *Header CRC*, which contain information about the payload length and the presence of a *Payload CRC* in frame. After that comes the payload itself and the *Payload CRC*.
- *MAC Layer Frame* consists of three components: *MAC Header* determines the type of message (uplink or downlink) and the protocol version, *MAC Payload* (in case of *Over The Air Activation (OTAA)* connection it contains *Join Request* or *Join Accept* messages), *MIC (Message Integrity Code)* value serves as an analogue of a check and prevents message spoofing.
- *Application Layer Frame* consists of *Frame Header*, *Frame Port*, which is defined depending on the type of application, and *Frame Payload*, which is encrypted using the *Application Session Key*. The *Frame Header* can also be divided into four parts [22]:
  1. *Device Address* consists of two parts - *Network ID* (8 bits) and *Device ID* (24 bits).
  2. *Frame Control* contains network control information.
  3. *Frame Counter* contains message sequence number.
  4. *Frame Options* allows one to add commands to change data rate, transmission power etc.



## ■ Encoding

According to the Semtech European patent application, the data is encoded in four distinct operations before being sent [25]:



**Figure 2.8:** LoRa PHY Tx and Rx chains

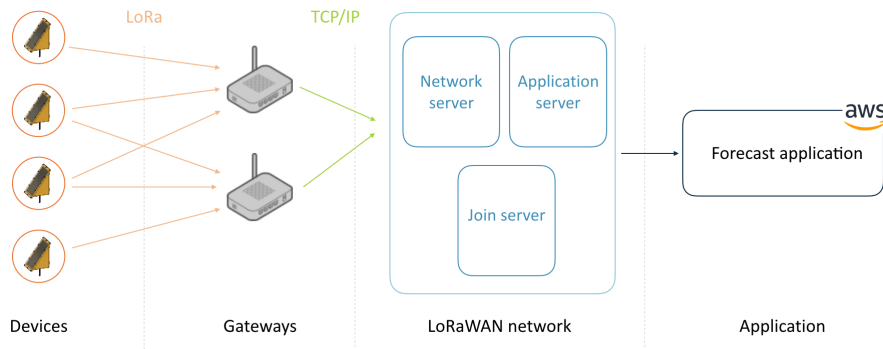
1. *Hamming(4,N) Forward Error Correction (FEC)* that adds  $N=\{5,6,7,8\}$  correcting parity bits, improving the robustness of the transmitted signal.
2. *Data whitening* that induces randomness into the symbols to provide more features for clock recovery.
3. *Interleaving* that distributes bits within the frame. This is done in order to make the transmitted data more resistant to interference.
4. Symbol *gray indexing* before sending over the air in order to add error tolerance preventing off-by-one errors when resolving symbols.

### ■ 2.2.2 LoRaWAN

*LoRaWAN (Long Range Wide Area Network)* is an open source *MAC* layer protocol that is intended to allow long-range data transfer at a low bit rate between connected sensors that monitor and send measured data back to the base station [26]. This radio frequency protocol is supported and developed by the *Lora Alliance community*, which consists of more than 500 companies (e.g. *IBM, Cisco, HP*) and whose goal is to standardize *LPWAN* for large-scale *IoT* deployments.

*LoRaWAN* uses a star-of-stars topology where gateways form a transparent bridge between devices and *LoRaWAN* network, converting radio frequency data into *Internet Protocol* data. Communication between end devices and the *LoRaWAN* network occurs in such a way that data is transmitted through the *LoRa* physical layer to the gateway, while data from the gateway is sent through the *IP*-based network to *LoRaWAN*.





**Figure 2.9:** LoRaWAN communication model

The communication model itself containing end devices and the *LoRaWAN* network consists of several parts [22]:

- *End devices (i.e nodes)*, which can be either sensors or actuators. They send packets at low frequencies to all gateways within the signal radius.
- *Base stations (i.e gateways)* that receive signals from end devices over a radio channel and then broadcast them to the central network via *TCP/IP*. The gateway together with the end devices is built in a star topology. The base station operates in half-duplex mode - the base station listens to the broadcasts in the operating frequency range (the frequency band is divided into eight 125 kHz channels) and, in case of request arrival, responds to it on the same frequency. The gateway is always connected to a stable power supply.
- *LoRaWAN network* consists of key components that meet standards of this type of networking.
- *Application* represents our target application, where the analysis and data processing takes place.

End devices send packets encoded as binary data via the *LoRa* radio protocol. Messages are received by gateways redirecting these messages to the back-end along with metadata, *Received Signal Strength Indicator (RSSI)* and *Signal-to-Noise Ratio (SNR)* values. In addition, each gateway periodically sends general information about its status, *GPS* coordinates and the number of processed messages. Each gateway is connected to one *Router*. The function of the *router* is to schedule dispatches and monitor the status of the gateway. Each *router* is connected to one or more *brokers*. *Brokers* play here a key role: they determine which application belongs to a particular device and which *Handler* to send the message to, they forward the uplink messages to the associated application and downlink messages to the corresponding router, which then forwards it to the correct gateway. The *Handler* serves as a place for data encryption and decryption. It also contains application data and is responsible for forwarding messages in the application.

The *LoRaWAN* network itself consists of several separate components where the main ones are [18]:

- *Network server* manages all base stations and defines the gateway through which the end device will communicate with the server. This server does not process information from devices. *Network server* contains *Handler*, *Broker* and *Router*. The server's task is to receive packets, deduplicate them, in case of receiving a packet by several gateways, and then send these packets to a specific application on the *Application server* in case of an uplink message and to a specific gateway in case of a downlink message. The *Network server* also manages the use of gateways, since one gateway can make one data transfer at a time. Therefore, the server distributes the load among neighboring gateways. The server also monitors security keys and frame counters.
- *Application server* is the destination server where all the data from end devices are addressed to. *Application server* contains the *LoRaWAN* application layer. This server links to a network server for receiving outgoing and sending downstream traffic. Also, on the server side, uplink packets are decrypted, and downlink packets are scheduled along with their encryption. In addition, this server supports many integrations that allow one to redirect data further to other applications for additional processing.
- *Join server* is responsible for fully-secured end-device authentication and network joining.

In addition to the above components, there is also a *Gateway server* managing *LoRaWAN* gateways and *Identity server* which is responsible for registering users along with creating new applications and devices.

Among the appropriate use-cases for *LoRaWAN* networks there are [18]:

- Long range (up to several kilometers)
- Low power (up to years of operation)
- Low cost (tens of euros)
- Low bandwidth (250 bit/s - 11 kbit/s)
- Security (128-bit encryption)

*LoraWAN* is not suitable for transferring large chunks of data or for real-time applications.

### ■ Device types

All devices within the *LoRaWAN* network are divided into three classes [17]. *Class A* is the base class that all devices must implement. *Classes B* and *Class C* are extensions to the specification of the main *Class A*.

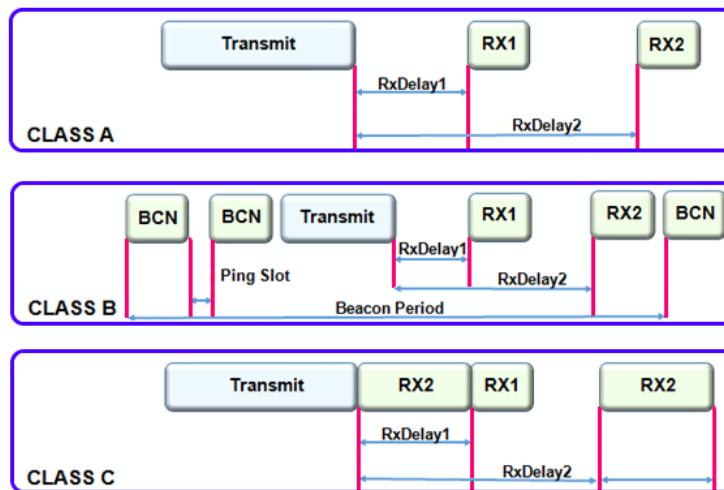


Figure 2.10: LoRaWAN device classes

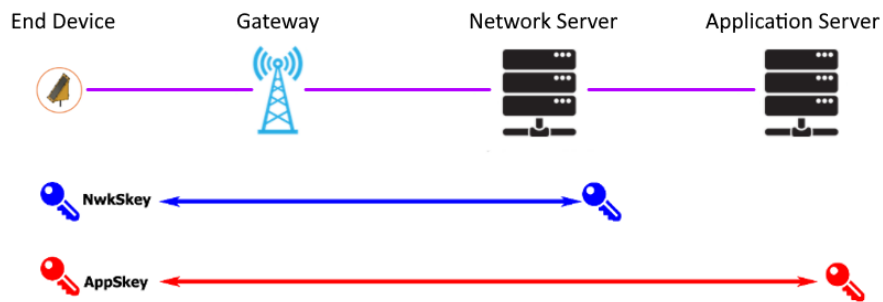
- *Class A* can be described as “battery powered sensors”. This is the base class that should be supported by all devices. It is the most energy efficient, since the device is in sleeping mode most of the time. The principle of operation is that as soon as a programmed trigger appears, a device wakes up, initiates an uplink and transmits data to the network. After the uplink message has been transmitted, the device listens for a response from the server for two time slots. In other words, the downlink is available only after the uplink is sent. That is, devices of *Class A* cannot be woken up by downlink from the application on the server making devices of this class not suitable for actuators. If the device does not receive a downlink message during the first time slot, it resumes sleeping for a short time, after which it wakes up and again listens to a response from the server during the second time slot. If during the second time slot the device does not receive a response from the server again, then it goes back to sleep until the next uplink message is sent.
- *Class B* can be described as “battery powered actuators”. A distinctive feature of devices of this class from *Class A* devices is that in addition to two time slots after uplink messages, devices also have separate scheduled time slots of a fixed duration for receiving downlink messages, which is possible thanks to a process called *beaconing*. During the *beaconing process*, the network broadcasts a temporarily synchronized beacon via gateways that must be periodically received by the end device in order to synchronize its internal clock with the network. After synchronizing the internal clock, end devices can open time slots (the so-called *Ping Slots*), during which the network can initiate downlink sending.
- *Class C* can be described as “mains powered actuators”. Devices of this class are constantly in the on mode and always listen to downlink messages (except for the time when they themselves send an uplink message). So, we can conclude that devices of this class do not depend

on power supply. Also, devices of this class are characterized by the lowest latency for downlink among all mentioned classes of devices. The principle of communication is similar to *Class A* devices, with the difference that *RX2* does not close until the device sends a new uplink message to the server.

## Security

Security plays a critical role in *LPWAN* networks. In order to make communication within the network as secure as possible, *LoRaWAN* uses two types of keys [18]:

1. Unique 128-bit *Network Session Key* using which the communication between the end device and the network server establishes.
2. Unique 128-bit *Application Session Key*, which uniquely maps the end device with the application where the particular device is registered.



**Figure 2.11:** Security in LoRaWAN applications

The process of encryption and subsequent decryption of data has four stages:

1. The data measured by the end device is encrypted using the *Application Session Key*.
2. Then the data is again encrypted by the *LoRaWAN* protocol using the *Network Session Key*, after which the packet is sent to the gateway.
3. Gateway receives the packet and forwards it to the *Network server*, which decrypts the *LoRaWAN* data. After that, the *Network server* sends the data further to the *Application server*.
4. *Application server* receives data and decrypts it using *Application Session Key*. The decrypted data is the data actually measured by the sensors.

## Join procedure

Each device must go through the registration procedure before starting to send data to the network.

In order to identify the device within the network, each device receives a globally unique identifier (i.e. *EUI-64-based DevEUI*). There are two ways to add a *LoRaWAN* device to the network [18]:

1. *Activation by Personalization (ABP)* method consists in preprogramming the *Network Session Key (NwkSKey)* and *Application Session Key (AppSKey)* in the device application code, thus the device is pre-registered on the network. Device registration does not require *DevEUI*, *AppKey* or *AppEUI*. In order to send the data, the device does not need to go through the *Join Procedure*, but can immediately send data to the application server.

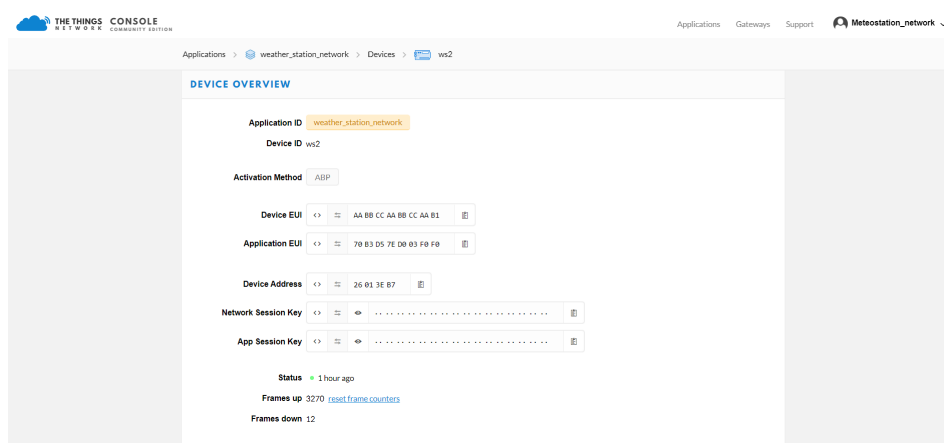
During the development and testing of the present system, this method was used. But when building a commercial network, deployable devices is preferred to be activated by *OTAA*.

2. *Over-The-Air-Activation (OTAA)* method consists of several steps. First, the end device sends a *Join Request* to the *Join server*. *Join Request* consists of: *DevEUI*, *AppEUI*, *AppKey* (encoded with hash), *DevNonce* (a unique 16-bit random number generated by device) and a *Message Integrity Code (MIC)* (that is generated in an analogue way as a checksum). After the server has received the *Join Request*, it validates the request using the *AppEUI* and, if successful, sends the *Join Accept* message to the device. The *Join Accept* message consists of: *NetID*, *DevAddr*, *AppNonce* (data generated by the *Application server*) and additional network settings. After receiving the *Join Accept* message, the end device saves the network settings along with the *NetID* and *DevAddr*. Using the received *AppNonce*, the device makes the last step - it generates session keys (*AppSKey* and *NwkSKey*). Only after completing the authentication process on the network, the data can be encrypted and sent to the *Application server*.

After adding the device to the network, data transfer becomes possible. Depending on *LoRaWAN* providers, the interface provided and the available functionality for working with the device differs. Within the framework of this project we will use *The Things Network* provider.

### ■ 2.2.3 The Things Network

*The Things Network (TTN)* is the most widespread and popular operator of *LoRaWAN* networks. It provides open and decentralized infrastructure for the Internet of Things. The main aim of the *TTN* as a member of *LoRa Alliance* is to deploy *LoRaWAN* networks around the world [29]. *The Things Network* sits between the gateway and the custom application, handling packet routing and processing. In addition, *TTN* allows one to create a completely private network.



**Figure 2.12:** Device overview in the TTN Console

This operator provides the ability to register both end devices and gateways. Each device is tied to a specific application within the account. In console, one can both create and configure a device, and also track the incoming and outgoing traffic.

The structure of *The Things Network* itself consists of communities. *Community* in this case means a city or locality where gateways and end devices are deployed.

## 2.3 Artificial Neural Networks

The term "*neural networks*" means a set of various algorithms capable of recognizing patterns in the input dataset. Input data can be any real data starting from text, sounds, pictures and ending with the results of sensor measurements. Neural networks are used both in classification and optimization, as well as in data analysis and predicting future trends.

With the development of new neural networks architectures, the overall interest has increased in the use of deep neural networks for predicting meteorological parameters. Every year, the accuracy of forecasts generated by deep neural networks increases, thereby creating a direct alternative to the existing *Numerical Weather Prediction* based solutions [1].

### 2.3.1 Basic concepts

The architecture of neural networks is designed in such a way that based on the principle of their work they can be compared to the brain work. Neural networks consist of many interconnected operating units (i.e. neurons), which are capable of solving many different problems by exchanging information. The use of one or another neural network architecture depends on the task at hand.

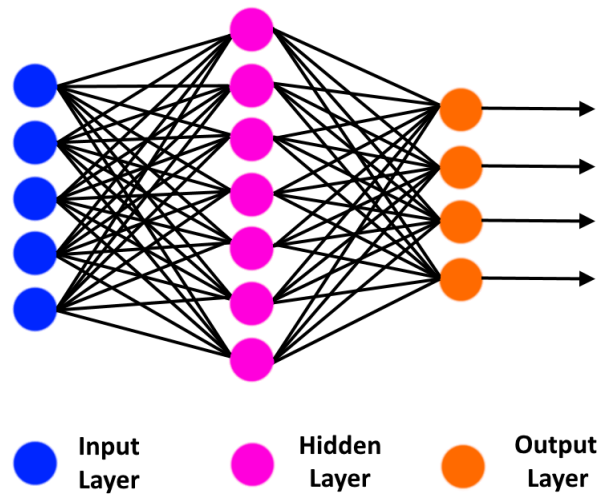


Figure 2.13: Neural Network basic structure

Globally, the structure of each neural network can be divided into separate layers: the first layer serves as a door through which the prepared data is fed into the neural network; then each subsequent layer contributes to the processing of the input data before getting the final answer to the task assigned to the neural network; the last layer forms the result of the entire neural network [30].

### ■ Neuron

Both biological and artificial neural networks consist of many computational elements (i.e. neurons) that receive and transmit signals to each other. A *neuron* consists of a cell body (i.e. soma), which is a summation function of inputs and connecting neurons. The input data for an artificial neuron (i.e. *perceptron*) is represented as input vector of numerical values while for a biological neuron the input data are electrical signals. These data are fed into the neuron through *dendrites*. *Nucleus* processes the data (i.e. calculates a weighted sum on its inputs and adds bias) and in case of a neuron is activated it can transmit the result through a connection called an *axon* to outputs called *synapses* [31].

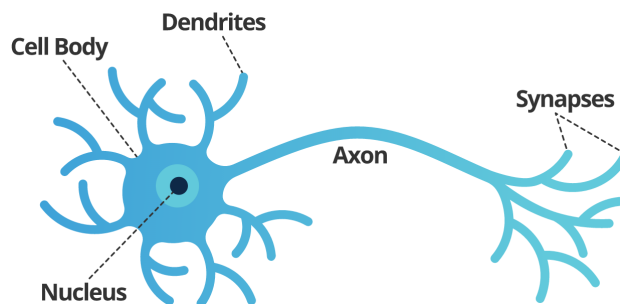


Figure 2.14: Structure of the Human neuron [32]

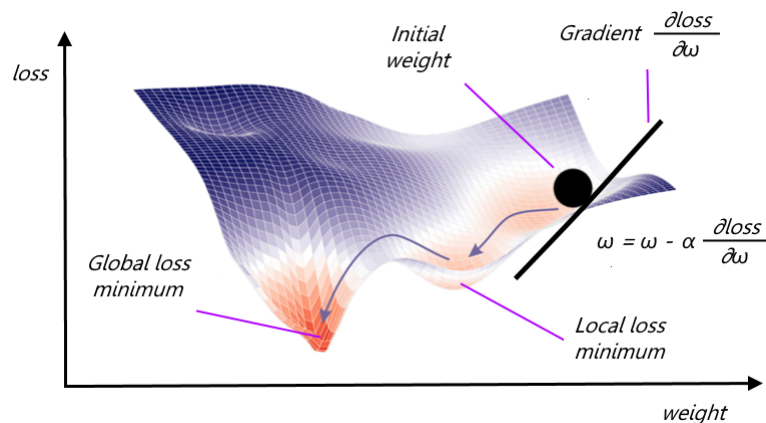
*Synapses* have 1 parameter called *weight*. Thanks to it, the input information changes when being transmitted from one neuron to another. That information will be dominant in the next neuron for the neuron with the greater weight. During the initialization of the neural network, the weights are randomly assigned. It is also important to remember that neurons operate with numbers in the range  $[0,1]$  or  $[-1,1]$ .

### ■ Learning process

In order the neural network to be able to solve the tasks assigned to it, it is necessary to train this network first. During the process of training a neural network, complex dependencies between input and output data are identified and information is generalized to obtain the correct result based on new data. The learning process consists in the sequential transmission of information from input neurons to output ones. This stage is called *forward propagation*. After that, we calculate the error and based on it we make a postback, which consists in sequentially changing the weights of the neural network, starting with the weights of the output neuron. That is, the so-called *backpropagation* occurs. The weights will change in the direction that gives us the best result [33].

When training a neural network, we also need to select appropriate *hyperparameters*, that is, parameters that are selected manually during the training of a neural network. The hyperparameters are: *learning rate*, *optimizer*, *number of layers* used, *loss function*, *batch size* and the *number of epochs* [33].

The *backpropagation* method, which we will talk about below, is based on the *gradient descent algorithm*, which is a way to find the global minimum of a function by moving along a gradient. The minimum in this case should be interpreted as the smallest possible error leading to an increase in the accuracy of the model [33].

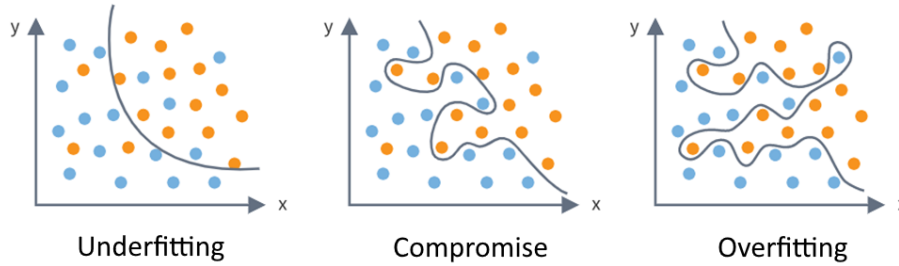


**Figure 2.15:** Gradient Descent Algorithm

In practice, however, training is continued not until the network is fine-tuned to a minimum of the error function, but until a sufficiently accurate



approximation is achieved. This will allow, on the one hand, to reduce the number of training iterations, and on the other hand, to avoid *overfitting*.



**Figure 2.16:** Underfitting, overfitting and appropriate-fitting in machine learning [27]

Also, there is a possibility of getting stuck in a local minimum, never reaching the global minimum during gradient descent. To prevent such a scenario, a so-called *momentum* is used, which can be described as the necessary acceleration to overcome the local minimum.

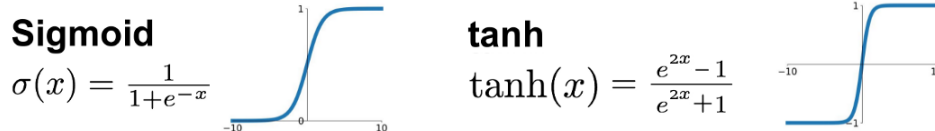
### ■ Hyperparameters

As mentioned above, when training a neural network, the most important task is to optimize *hyperparameters*. Each *hyperparameter* makes its own adjustments to the final result of the machine learning algorithm, therefore, for the optimal solution of each specific problem, suitable hyperparameters are manually selected [33].

**Activation function.** The activation function determines the output value of the neuron depending on the result of the weighted sum of the inputs and the threshold value. Or in other words, an activation function is a way to normalize input data. In case of having a large number at the input, then after passing it through the activation function we will get an output in the range we need.

$$Y = \sum (weight * input) + bias$$

The activation function checks the  $Y$ -value produced by the neuron to see if external connections should treat this neuron as activated, or if it can be ignored.



**Figure 2.17:** Sigmoid and Tanh activation functions

In recurrent neural networks, which are described in next section, the following nonlinear activation functions are used:

- *Sigmoid* translates input data in the range  $[-\infty; +\infty]$  to the range in  $(0; 1)$ . In particular, large negative numbers turn into zero, and large positive numbers turn into one. Historically, the sigmoid function has found wide application, since its output is well interpreted as the level of neuron activation: from no activation (0) to fully saturated activation (1).
- *Hyperbolic tangent* with a range of values  $[-1,1]$ . It is used when values can be either negative or positive.

**Number of Epochs and Early Stopping.** *Epoch* represents the number of complete passes through the training dataset during the training process. But during the training of the neural network, there is a danger that too long training (i.e. a large number of epochs) will lead to overfitting on the training data resulting in the increasing of generalization error that in its turn will lead to a poor performance of making predictions on a new data. One of the most effective regularization forms to prevent this issue is called *early stopping*. The principle of this method is that we evaluate the model every  $n$  epochs on both training and validation data and if the model loss on the validation data begins to increase then we stop the training procedure.

In our model, we used the *patience* parameter, which determines the number of epochs without improvement, after which the training will be stopped. Thus, we can immediately set a fairly large number of epochs, since the algorithm itself will stop training and save the best parameters of the model.

**Learning rate.** It is a hyperparameter that determines the step size at each iteration when moving towards the minimum of the loss function. The lower the value, the slower we move down the incline. Although using a low learning rate factor we can get a positive effect in the sense of not missing a single local minimum, it can also mean that we will have to spend a lot of time for convergence, especially if we hit a plateau area.

**Batch size.** It is a hyperparameter that defines the number of samples (i.e. rows of data) used during one iteration (i.e. one epoch) to update the internal parameters of the model. In case when the batch size is greater than one but less than the size of the training data set, then we are talking about the concept of *mini-batch*. The choice of batch size is always a compromise, since a large batch size reduces the computational cost, but leads to an increase in the generalization error and the overall performance decreases.

**Loss function.** The *Loss function* is a measure of how well our prediction model predicts the expected value. In the training process, we try to minimize the loss function by updating connections' weights to improve accuracy. In our model, we used *Mean Squared Error (MSE)* loss function to calculate the accuracy of the prediction model:

$$MSE = \frac{1}{n} \sum_{n=1}^n \left( Y_i - \hat{Y}_i \right)^2 \quad (2.5)$$

where:

- $Y_i$  is the  $i^{th}$  observed/actual value and  $\hat{Y}_i$  is the  $i^{th}$  predicted value

**Optimizer.** An optimizer is an algorithm that helps update model parameters to minimize the loss function. The *Adaptive Moment Estimation (Adam)* optimizer [28] will be used within the scope of the present work. *Adam* is a widely used optimization algorithm that combines the characteristics of the adaptive parameters update in the *AdaGrad* algorithm and the *stochastic gradient descent with momentum*. *Adam* uses the estimates of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network [28]:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.6)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.7)$$

where:

- $m_t$  is the first moment estimation (the mean)
- $v_t$  is the second moment estimation (the uncentered variance)
- $g_t$  is the gradient at time step  $t$

However, in this formulation, there is a problem of long accumulation of  $m_t$  and  $v_t$  at the beginning of the algorithm's work, especially when the decay rates  $\beta_1$  and  $\beta_2$  are close to 1. In order to get rid of this problem and not introduce new hyperparameters, the estimates of the first and second moments are modified [28]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (2.8)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.9)$$

Then the equation for updating the parameters is [28]:

$$\theta_{t+1} = \theta_t + \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.10)$$

where:

- $\beta_1, \beta_2, \epsilon$  - coefficients
- $\theta_{t+1}$  - updated set of parameters for the next optimization step
- $\theta_t$  - set of parameters at the current step
- $\eta$  - learning rate

The authors of the algorithm recommend using the following coefficient values:  $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e - 8$

### 2.3.2 Recurrent Neural Networks

*Recurrent neural networks (RNNs)* are a kind of feedforward neural networks that operate in time using internal memory allowing to calculate the output of each cell based not only on the value of the input data sequence, but also on the output of the previous cell [35]. Traditional neural networks can process and classify data, carry out correlations but at a specific point in time. The architecture of such networks does not allow remembering the data context in the past. Which is their main disadvantage when dealing with sequential data.

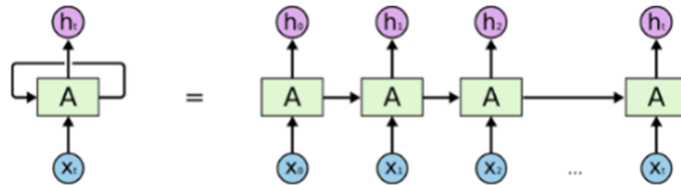
#### Sequential Data

*Sequential data* can be defined as any kind of data where consistency is important, meaning that the data is interdependent. In this kind of data, each observed state at time  $t$  is dependent on the value of the state at time  $t-1$ . Among the types of sequential data, one can distinguish audio or video records, text streams and time-series data. Meteorological data is an example of time-series data.

#### RNN Structure

Unlike traditional neural networks, where all data inputs are independent from each other, in *RNN* the inputs are interdependent [36]. *RNNs* implement a semblance of memory, with the help of which they are able to determine patterns and correlations in sequential data, that is, neurons in this type of networks receive and process sequential information not only from the neurons of the previous layer, but also from themselves of the previous iteration, interacting with themselves, making the order of data submission one of the key parameters of the operation of the network.

*RNN* elements are depicted as neurons with a circular arrow. In addition to the input signal  $x_t$ , the neuron also uses its current state, that is, its own output from the previous iteration. If we carry out an abstract decomposition of the structure of one *RNN* cell, then we get a chain of multiple copies of the same cell unit, each of which receives its own sequence element ( $x_0...x_t$ ) as input, generates a prediction ( $h_0...h_t$ ) and passes it further along the chain as a kind of cell memory.



**Figure 2.18:** An unrolled Recurrent Neural Network [34]

Let's assume that a layer is formed from neurons of this type, then the

output of the layer can be described by the following equation [37]:

$$\mathbf{y}_t = \phi(\mathbf{W}_x^T \cdot \mathbf{x}_t + \mathbf{W}_y^T \cdot \mathbf{y}_{t-1} + \mathbf{b}) \quad (2.11)$$

where:

- $\mathbf{W}_x$  - weight matrix at the time of observation
- $\mathbf{W}_y$  - matrix of outputs at previous time points
- $\mathbf{x}_t$  - input vector at the time of observation
- $\mathbf{y}_{t-1}$  - output vector at previous time point
- $\mathbf{b}$  - vector of bias parameters
- $\phi$  - activation function

The network is trained by unrolling layers through time and the backpropagation's technique called the *chain rule* with time dependency is applied. The general name for training these networks is called the *Backpropagation through time (BPTT)*. The process of training and updating weights continues until the global minimum in the gradient descent is reached.

However, the main disadvantage of this type of neural networks is the vanishing gradient problem, in which, during the *backpropagation*, the error signal used to train the network exponentially decreases with each layer towards the first layer. As a result, layers closer to the network input layer remain untrained. Thus, there is a rapid loss of information over time, which makes this type of neural network unsuitable for working with long data sequences.

### ■ 2.3.3 LSTM Networks

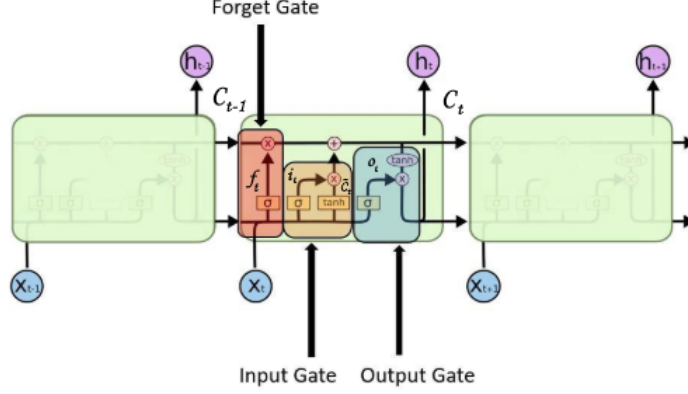
*LSTM* stands for *Long Short Term Memory* and was first proposed by Hochreiter and Schmidhuber [38]. *LSTM* is a type of *RNN* networks that was designed specifically to solve the vanishing gradient problem, in which the stored value in memory has blurred over time. *LSTM* enables memorizing values for both short and long periods of time, making them a suitable tool for time series prediction.

This type of neural networks is widely used in all kinds of projects, for example, Google translator is currently based on machine learning methods and uses the operation principle of *LSTM* networks. The system performs calculations to understand the meaning of a word or phrase based on previous meanings in the sequence (i.e. context). Such an algorithm helps the system to understand the context of the sentence and correctly select the translation among various options.

There are also many projects that have successfully developed models for meteorological parameters prediction, including an hourly forecast of solar radiation levels for the next 24 hours [39], as well as a project in which the

temperature values are predicted using *LSTM* while restoring missing values [40].

The main structural unit of *LSTM* is the cell state, represented by the so-called *conveyer line*, where  $C_{t-1}$  is the cell state (i.e. memory) of the previous cell, and  $C_t$  is the cell state of the current cell.



**Figure 2.19:** LSTM cell with its internal structure [41]

Each neuron contains three components (i.e. gates), which are able to add or remove information from the cell state [42]:

1. *Forget Gate* determines what information will be passed or excluded in the cell state. At the input, this gate gets predicted values from the previous cell  $h_{t-1}$  and the input sequence  $x_t$ . The sigmoid function takes a value of 0 (exclude) or 1 (do not exclude) for each memory state of the previous  $C_{t-1}$  cell. In the end, a pointwise vector product of the previous cell's state  $C_{t-1}$  with the output of the sigmoid function is calculated. The result of the calculation will be either the skipping of the state further or its blocking. This operation can be described using the equation:

$$f_t = \sigma(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f) \quad (2.12)$$

2. *Input Gate* determines which value of input  $x_t$  to use to change the memory. Conventionally, this *Input gate* can be divided into two sub-gates: *Input Gate* that uses the sigmoid function, and *Candidate Gate* that uses the hyperbolic tangent function. First, the sigmoid function takes the values 0 (do not skip anything) or 1 (skip everything) deciding which values will be updated. Then, the tangent function creates a vector of  $\tilde{C}_t$  values that can be added to the cell state by assigning weights (-1 to 1) to the values, allowing the importance of the individual parts of the data vector to be determined:

$$i_t = \sigma(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i) \quad (2.13)$$

$$\tilde{C}_t = \tanh(\sigma(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c)) \quad (2.14)$$

At the final step, the cell state is updated from the  $C_{t-1}$  value to the  $C_t$  value:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (2.15)$$

Where the first part of the product "forgets" irrelevant information, and the second part adds information to remember.

3. *Output Gate* determines the output value of the cell. The sigmoid function decides which constituents of the cell state  $C_t$  will be fed to the cell output. The tangent function assigns a weight (-1 to 1) to the passed values. Both functions are then multiplied:

$$o_t = \sigma(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o) \quad (2.16)$$

$$h_t = o_t * \tanh(C_t) \quad (2.17)$$

The list of designations used is as follows:

- $x_t \in \mathbb{R}^d$ : input vector to the LSTM unit
- $\mathbf{W} \in \mathbb{R}^{h \times d}$ : input weights matrix
- $\mathbf{U} \in \mathbb{R}^{h \times d}$ : state weights matrix
- $b \in \mathbb{R}^h$ : vector of bias parameters
- $f_t \in \mathbb{R}^h$ : activation vector of the forget gate
- $i_t \in \mathbb{R}^h$ : input/update gate's activation vector
- $\hat{C}_t \in \mathbb{R}^h$ : activation vector of the cell input
- $C_t \in \mathbb{R}^h$ : cell state vector
- $o_t \in \mathbb{R}^h$ : output gate's activation vector
- $h_t \in \mathbb{R}^h$ : hidden state vector (i.e. LSTM unit output vector)
- $\sigma$ : sigmoid activation function
- $\tanh$ : hyperbolic tangent activation function
- $d$ : number of input features
- $h$ : number of hidden units

### ■ Forward propagation

During the forward propagation process, we calculate the temporal gradients of the neural network (i.e. the values of the output layer), which are then compared with true values using the loss function to subsequently change each network weight according to the ratio between an error derivative to the corresponding weight derivative. Forward propagation in *LSTM* network is already described with the help of equations [2.12 - 2.17].

## ■ Backpropagation through time

The feedforward neural networks learning process itself is performed using an algorithm called *backpropagation*. The goal of the algorithm is to change and optimize the weights of the neural network connections, so that the predicted values are as close as possible to the true values. As described earlier, unlike classical neural networks that are trained at a specific time  $t$ , *LSTM* neural networks in addition to training at time  $t$  are also trained using the data from time before  $t$  (i.e.  $t-1, t-2 \dots t-n$ ). This variation of the backpropagation algorithm used in *LSTM* networks is called *Backpropagation Through Time (BPTT)*. Vanishing gradient problem in *RNN* networks is solved in *LSTM* using the architecture of the cell itself: the error remains in the *LSTM* cell of the particular unit as long as the error backpropagates from the output layer to each unit's gate until gates learn to eliminate this error.

Derivative value of each internal gate during the *BPTT* process can be described as follows [36]:

$$\delta h_t = \Delta_t + \Delta h_t \quad (2.18)$$

$$\delta C_t = \delta h_t + \Delta h_t \odot o_t \odot (1 - \tanh^2(C_t)) + \delta C_{t+1} \odot f_{t+1} \quad (2.19)$$

$$\delta a_t = \delta C_t \odot i_t \odot (1 - a_t^2) \quad (2.20)$$

$$\delta i_t = \delta C_t \odot a_t \odot i_t \odot (1 - i_t) \quad (2.21)$$

$$\delta f_t = \delta C_t \odot C_{t-1} \odot f_t \odot (1 - f_t) \quad (2.22)$$

$$\delta o_t = \delta h_t \odot \tanh(C_t) \odot o_t \odot (1 - o_t) \quad (2.23)$$

$$\delta x_t = \mathbf{W}^T \cdot \delta gates_t \quad (2.24)$$

$$\delta h_{t-1} = \mathbf{U}^T \cdot \delta gates_t \quad (2.25)$$

The final update of internal parameters can be described as follows:

$$\delta \mathbf{W} = \sum_{t=0}^T \delta gates_t \otimes x_t \quad (2.26)$$

$$\delta \mathbf{U} = \sum_{t=0}^{T-1} \delta gates_{t+1} \otimes h_t \quad (2.27)$$

$$\delta b = \sum_{t=0}^T \delta gates_{t+1} \quad (2.28)$$

where:

- $a_t$  - Candidate Gate value at time  $t$  during forward pass
- $gates_t$  - gate state at time  $t$
- $\odot$  is the Hadamard product
- $\otimes$  is the outer product
- $\cdot$  is the inner product



## 2.4 Amazon Web Services

*Amazon Web Services (AWS)* is a commercial public cloud platform owned and developed by Amazon. This platform provides cloud serverless computing services, cloud databases, infrastructure solutions such as storage resources, virtual servers and much more.

### 2.4.1 Global Infrastructure

The *AWS* infrastructure network is located around the world and is designed to maximize performance, provide high availability, and optimize costs while using services.

#### Regions

The structure of this platform begins with the concept of "*regions*", which represent the geographical location of data centers in different parts of the world [43]. Each region is completely independent and all popular cloud services (e.g. *AWS EC2*, *AWS RDS*, *AWS Elastic Beanstalk* etc.) are supported by each region. The exchange of data between regions is carried out via the public Internet, from which it follows that when exchanging data between regions an additional encryption should be used to protect the data. At the moment there are 25 regions available around the world.



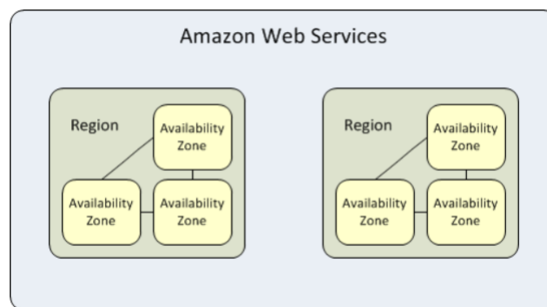
**Figure 2.20:** AWS Global Infrastructure [44]

Distribution by region is implemented in order to achieve the highest possible fault tolerance and stability. When choosing a region, one should be guided by the following concepts:

1. *Availability of the required services*, since not all regions provide the full range of existing services and solutions.
2. *Latency*, since it is better to choose regions located as close as possible to the user's position in order to reduce data latency.

## ■ Availability Zones

Each region is divided into *Availability Zones* [43]. An *Availability Zone* is a specific physical location within a region, designed to provide maximum isolation and service resiliency. Each *Availability Zone* has its own independent infrastructure that is isolated from potential problems in other *Availability Zones* (e.g. fire, earthquake or equipment failure) within the same region while maintaining high-speed connectivity between the zones. Data transfer between *AZs* is encrypted.



**Figure 2.21:** AWS Regions and Availability Zones [45]

This infrastructure design allows the high availability and fault tolerance. Amazon guarantees that any service in any geographic region will have high availability of *four nines* (99.99% of uptime) during a year meaning that downtime per month will be no more than 4.38 minutes.

### ■ 2.4.2 Services

*AWS* offers a wide range of services: starting from server rental, database creation and maintenance, ending with load balancers and application health monitoring [46]. The following are the services used within the scope of this project.

#### ■ Amazon EC2

*Amazon EC2* is an analogue of a virtual dedicated server (*VDS*). *VDS* uses its own copy of the operating system, accessing which the user has a super-user access level to the instance of this operating system. *EC2 instances* are used to run custom applications. There are different types of instances depending on the goals and required resources of the application. Each type of instance has its own number of CPUs, memory and storage.

#### ■ Amazon RDS

*Amazon RDS* is a cloud-based service that provides users with relational databases for later use in their applications. The benefits of using *Amazon RDS* are easy maintenance, fast deployment, and easy scalability. Also,

the database administration (including software updates, backups etc.) is done automatically. Currently, *Amazon RDS* supports *MySQL*, *MariaDB*, *PostgreSQL*, *Oracle*, and *Microsoft SQL Server* database engines.

## ■ AWS Elastic Beanstalk

*AWS Elastic Beanstalk* is an orchestration service meaning that it is a service that provides tools to automate background tasks such as automated configuration, coordination and management of the system. *Elastic Beanstalk* is an additional abstraction layer over *EC2* and is used for deploying and scaling web applications and services. With *Elastic Beanstalk* one can set up an environment containing one or more *EC2* instances, a database, as well as additional useful components such as auto scaling (i.e. dynamically adjusting the amount of computing resources) and *Elastic Load Balancers* (i.e. distributing a set of tasks across a set of resources). Thanks to *Elastic Beanstalk*, the developer can devote more time to the application development itself, rather than solving tens of background nuances of the infrastructure.

## ■ 2.5 Solar Radiation

In this work, for a more accurate prediction of the local meteorological situation, numerical calculations are used introducing an additional training parameter for the developed neural network. This parameter is *solar radiation* representing the electromagnetic radiation emitting by the Sun and falling on a surface.

There are three main climate-forming factors that determine the climate at any point on the Earth: *solar radiation*, *terrain* and *atmospheric circulation* [47]. Since parameters related to solar radiation will be used within the framework of this work, we will focus on this factor.

*Solar radiation* determines the flow of solar energy and heat to different parts of the earth's surface. The amount of heat is determined by latitude (i.e. angle between the Equator plane and observer current position measured at the Earth center) [48]. Both all life processes and other indicators of climate - pressure, precipitation and atmospheric circulation - directly depend on the amount of heat. The circulation of the atmosphere, in its turn, determines the movement of air masses, which redistribute pressure, temperature and humidity.

Solar radiation strongly affects the Earth only during the daytime, when the Sun is above the horizon. Radiation can be divided into the amount of radiation that enters the outer boundary of the atmosphere, and radiation directly received by the earth's surface [48]. These parameters are not identical, since the solar energy flux decreases while passing to the earth's surface through the atmosphere containing dust, various gases and aerosols. In addition, the amount of radiation received by the Earth depends on the distance from the Sun. Even small changes in the distance between the Earth

and the Sun, depending on the orbit, lead to a significant change in the amount of radiation received by the Earth surface.

The rotation of the Earth around its axis determines the day-night rhythm, and the rotation around the Sun determines the change of seasons, on which the amount of incoming solar radiation depends much more strongly. In summer, the flow of solar radiation is much greater than in winter.

### ■ 2.5.1 Irradiance of a plane

Since there is a cross-correlation between the solar radiation and other meteorological parameters, we can calculate the value of the solar irradiance for a specific location (without taking into account clouds) in order to use it as an additional parameter that helps us to more accurately characterize the local meteorological situation.

Solar radiation, entering the Earth's atmosphere, loses its energy and attenuates before reaching the ground. Attenuation of solar direct radiation while passing through the medium changes in proportion to the distance traveled and local radiation flux. This relationship is expressed by the Bouguer-Lambert law [49]:

$$H_d = S \cdot G_{SC} \cdot e^{-\mu m} \cdot \sin(h) = G_0 \cdot e^{-\mu m} \quad (2.29)$$

where:

- $H_d$  - direct irradiance on a surface
- $S$  - Sun-Earth distance correction factor
- $G_{SC}$  - solar constant (i.e. intensity of the solar radiation hitting one  $m^2$  of the Earth)
- $\mu$  - total attenuation coefficient
- $m$  - optical air mass
- $h$  - solar altitude angle

The  $S \cdot G_{SC} \cdot \sin(\theta)$  product represents the radiation at the top of the Earth's atmosphere (i.e. extraterrestrial radiation  $G_0$ ). Attenuation coefficient is also called *atmospheric optical depth*.

### ■ 2.5.2 Extraterrestrial radiation

The following parameter represents the Sun's intensity at the top of the Earth's atmosphere. It changes during the year and can be described by the equation [51]:

$$G_0 = G_{SC} \left( 1 + 0.033 \cdot \cos \frac{360 \cdot n}{365} \right) \quad (2.30)$$

where:

- $n$  - particular day of the year

### ■ 2.5.3 Atmospheric optical depth

The attenuation coefficient arising when solar radiation passes the atmosphere is called the *atmospheric optical depth*. A certain amount of solar radiation is scattered or absorbed by aerosols and molecules as solar radiation passes through the earth's atmosphere. The total atmospheric optical depth can be divided into two parts - the first part is related to Rayleigh scattering, associated with the fact of scattering of sunlight out of a direct beam while passing the atmosphere (i.e. molecular optical depth); the second part deals with scattering of sunlight by aerosols (i.e. aerosol optical depth) [55].

#### ■ Molecular optical depth

The molecular optical depth (MOD) parameter can be calculated using the equation [48]:

$$MOD = \frac{1}{9.4 + 0.9 \cdot m} \quad (2.31)$$

where:

- $m$  - optical air mass

**Optical air mass.** This parameter directly depends on the path length of sunlight through the atmosphere, that is, on the solar altitude angle. Optical air mass is a parameter describing the ratio of the path length of solar radiation through the atmosphere in a certain sun angle ( $h$ ) to a coordinate point relative to the path length from the sea level to the zenith angle. If the Sun is 90 degrees above the horizon, air mass is equal to 1. This parameter can be calculated using the formula [52]:

$$m = \frac{2.0016}{\sqrt{\sin^2 h + 0.003147 + \sin h}} \quad (2.32)$$

where:

- $h$  - solar altitude angle

#### ■ Aerosol optical depth

Represents the attenuation coefficient of solar radiation from the top of the atmosphere down to the surface caused by aerosols. Atmospheric aerosol is created by the infiltration of miniature solid and liquid particles into the atmosphere. This parameter is measured using a sunphotometer, which measures spectral solar irradiance at particular wavelengths. The average value of this parameter can be taken from open sources, for example, from the data that NASA project *AERONET* (*Aerosol Robotic Network*) provides on its website [53].

### 2.5.4 Sun elevation angle

The location of the Sun in the sky is defined by the term *solar altitude* (i.e. elevation), which is the angle between rays emitted by the Sun and the horizon [51]:

$$h = \arcsin(\sin\delta \cdot \sin\varphi + \cos\delta \cdot \cos\varphi \cdot \cos\omega) \quad (2.33)$$

where:

- $\delta$  - solar declination
- $\varphi$  - local latitude
- $\omega$  - hour angle

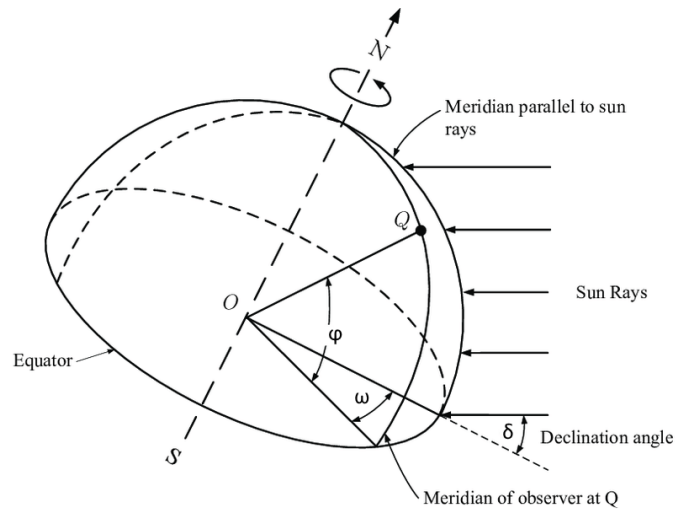


Figure 2.22: Solar geometry [50]

### Declination

It is a function of the time of year and is represented as the angle between the line connecting the centers of the Earth and the Sun and the plane of the Equator. Declination angle is the same for the whole Earth on any specific day and can be represented as a Fourier series using Spencer's formula [54]:

$$\begin{aligned} \delta = & 0.006918 - 0.399912\cos(\Gamma) + 0.070257\sin(\Gamma) - 0.006758\cos(2\Gamma) + \\ & + 0.000907\sin(2\Gamma) - 0.002697\cos(3\Gamma) + 0.00148\sin(3\Gamma) \end{aligned} \quad (2.34)$$

where:

- $\Gamma$  - day angle

The *day angle* can be calculated in radians (*rd*) as:

$$\Gamma = \frac{2\pi(n-1)}{365} \quad (2.35)$$

where:

- $n$  - particular day of the year

### ■ Hour angle

This parameter is a virtual translation of the Sun over the local meridians due to Earth rotation. Hour angle can be calculated from the *Solar Time* (ST) using the equation [51]:

$$\omega = 15^\circ \cdot (ST - 12) \quad (2.36)$$

### ■ Solar time

Solar time (ST) is defined as time based on the position of the Sun in the sky in relation to the local meridian and on the rotation of the Earth. It can be calculated as [55]:

$$ST = \text{Local Standard Time} + \frac{\lambda}{15} + E \quad (2.37)$$

where:

- $\lambda$  - longitude (i.e. angle between an observer current position and Greenwich meridian measured at Earth center)
- $E$  - Equation of Time

Longitude  $\lambda$  is expressed in hours since the Earth rotates 15 degrees per hour. *Local Standard Time* indicates fixed time for all locations that are geographically located on the same meridian meaning that places with the same longitude have the same *Local Standard Time*.

### ■ Equation of Time

*Equation of Time* represents a periodic (i.e. one year period) function of the difference between Mean Solar Time and True Solar Time depending on the day of year [55]:

$$E_t(rd) = 0.000075 + 0.001868\cos(\Gamma) - 0.032077\sin(\Gamma) - 0.014615\cos(2\Gamma) - 0.04089\sin(2\Gamma) \quad (2.38)$$

where:

- $\Gamma$  - day angle





## Chapter 3

### Implementation

This chapter describes the process of developing an entire system. The chapter itself is divided into several component parts. The first part pays attention to the design of a device hardware along with the device software. Device part is then followed by a description of the developed machine learning model. Lastly, the description of the implemented cloud infrastructure for processing and displaying measured and predicted parameters with an integrated data sharing process finalizes the present chapter.

#### 3.1 Device Assembly

The purpose of this section is to design a device that must meet the following *IoT* criteria:

- The device should measure meteorological parameters sufficient to subsequently predict the local meteorological situation.
- Maximum energy autonomy of the device.
- Ability of the device to transmit measured data to the server using the *LoRa* protocol.
- Compactness of the device, low price and design allowing the usage in urban environment.

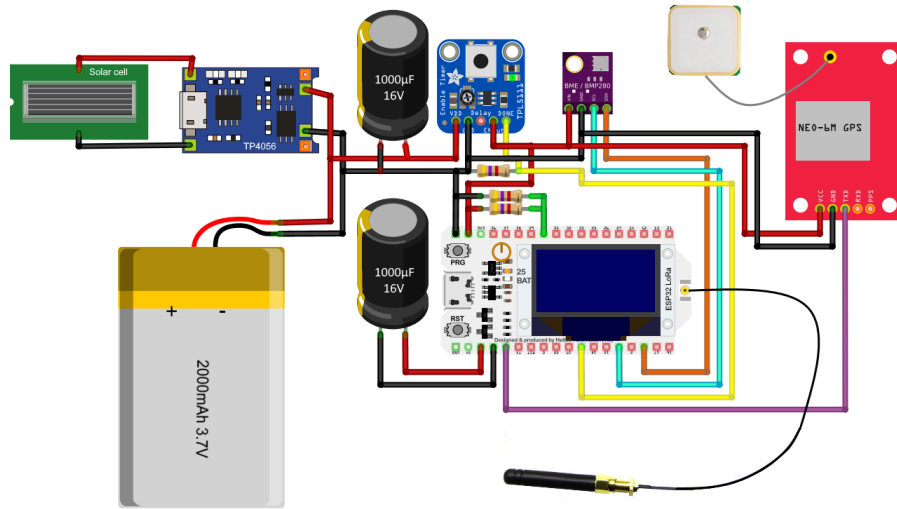
To fulfill all of the above points, the work was carried out on the selection of individual components and designing the weatherproof cover along with the mounting mechanism.

In addition, a software must be developed for the device, capable of serving both measurements of environmental parameters along with their subsequent sending.

##### 3.1.1 Device Hardware

This paragraph describes the process of assembling the device hardware. After a preliminary analysis of the project requirements, a list of required components was prepared. The present scheme consists of a main control

module and peripherals responsible for measuring and sending data. There are also components responsible for powering and shutting down the entire circuit to conserve the maximum amount of energy.



**Figure 3.1:** Scheme of the assembled device

All modules are connected together within one scheme and placed in a designed weatherproof case.

### ■ ESP32 LoRa module

The *ESP32 LoRa* microcontroller module based on the *ESP32-D0WDQ6* [56] chip was chosen as the control unit for our device. In addition to the control chip, this module also contains peripherals for connecting sensors, imbedded *LED* display and *LoRa SX1276* chip. Moreover, this module has a radio module, *Wi-Fi* module and *Bluetooth* (both classic and *Bluetooth Low Energy*).

The present module is based on a chip developed by *Espressif Systems*. This chip has been specially designed for the use in mobile and *IoT* applications. Among the state-of-the-art *IoT* characteristics of this chip, multiple power modes and low duty cycle can be emphasized.

*ESP32* runs on  $3.3\text{ V}$  but it can also be powered by a  $5\text{ V USB}$  due to the existence of a built-in voltage regulator that transforms  $5\text{ V}$  to the desired  $3.3\text{ V}$ .

**CPU and Memory.** The module has a dual-core low-power *Tensilica Xtensa 32-bit LX6* microprocessor running on 160 or 240 MHz frequency. The first core (*Protocol CPU*) handles work with peripherals and the second (*Application CPU*) handles the application code. For booting and core functions *ROM* memory of the capacity of  $448\text{ KB}$  is used. For data and instructions the on-chip *SRAM* memory of  $520\text{ KB}$  is used. Also, *ESP32*

supports multiple external *QSPI* flash and *SRAM* chips that can be accessed through high-speed caches.

**Timers and Watchdogs.** *ESP32* chip has four integrated 64-bit general-purpose timers. Also, there are three watchdog timers available. Despite the high stability of modern computer systems, they are still susceptible to freezes and errors. Watchdog timer allows us to timely track a malfunction and force it to reset.

**Peripherals.** There are 34 *General Purpose Input/Output (GPIO)* pins available within this chip. Also, 12-bit *Successive Approximation Register (SAR) Analog-to-Digital Converter (ADC)* is available. In order to convert digital signals into analog voltage signals two *8-bit Digital-to-Analog Converter (DAC)* channels can be used.

For communication with external modules, among others, the following interfaces can be used: *I<sup>2</sup>C* (Inter-Integrated Circuit), *UART* (Universal Asynchronous Receiver/Transmitter), *CAN 2.0* (Controller Area Network), *SPI* (Serial Peripheral Interface), *I<sup>2</sup>S* (Integrated Interchip Sound), *RMII* (Reduced Media-Independent Interface).

*ESP32* integrates an *ULP (Ultra-Low-Power)* co-processor, which is a simple *FSM (Finite State Machine)* that allows measurements using internal sensors or sensors connected via an *I<sup>2</sup>C* interface while the main processor is in a deep sleep mode. The measured values can serve as a trigger to wake up the main processor.

Also, *ESP32* has integrated *Pulse Width Modulation (PWM)* controller and hardware accelerators.

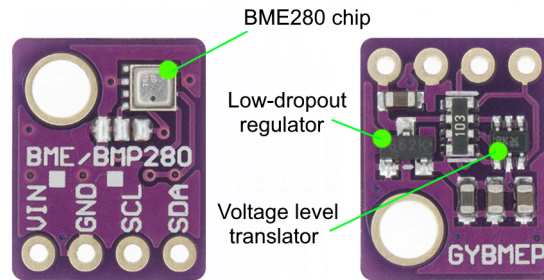
**LoRa chip.** The *ESP32 LoRa* module uses the *SX1276* chip [57]. This chip is a member of the *Semtech* family of transceivers and provides ultra-long range spread spectrum communication and high interference immunity. Moreover, this chip boasts small current consumption making it a perfect choice for use in low-power applications.

## ■ BME280 sensor

The *BME280* module [58] was chosen as the main sensor of our weather station. The following measurement principles are used within given sensor:

- *Temperature* is measured using the diode voltage. This principle is based on the fact that at constant current, the voltage at the *PN* junction of a diode will decrease by about  $2\text{ mV}$  per  $1^\circ\text{C}$ . Then, using calculations or a table, the temperature for a particular diode is determined.
- *Pressure* is measured by using piezoresistive strain gauges which changes its resistance when stretched. The strain gauge is attached to the diaphragm. Resistance of the diaphragm changes when it is compressed or elongated. Resistance change is directly proportional to the changes in pressure.

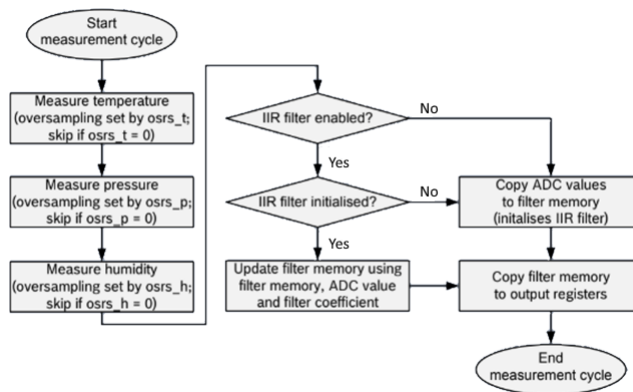
- *Humidity* is measured using a capacitor. It means that the sensing element is represented by a capacitor composed from two metal electrodes separated by a hygroscopic dielectric material. The total capacitance is affected by the changes of atmosphere's relative humidity represented as the process of an absorption of water vapor by the sensor (i.e. capacitance is increasing).



**Figure 3.2:** BME280 digital sensor module [58]

Communication with the microcontroller is performed via the  $I^2C$  interface. Measurements can be carried out either by command from the microcontroller, or at regular intervals.

The measurement flow itself consists of several parts. First, temperature, pressure and humidity are measured with selectable oversampling in order to reduce the noise.



**Figure 3.3:** BME280 measurement cycle [58]

After that, the measured temperature and pressure (it is not necessary to apply such a filter for humidity) can be passed through an optional *IIR filter*, which reduces the bandwidth of the temperature and pressure output signals and increases the resolution of both parameters' output data to 20 bit.

While measuring pressure, temperature and humidity parameters this sensor consumes  $3.6 \mu A$  (using  $1 Hz$  data refresh rate).

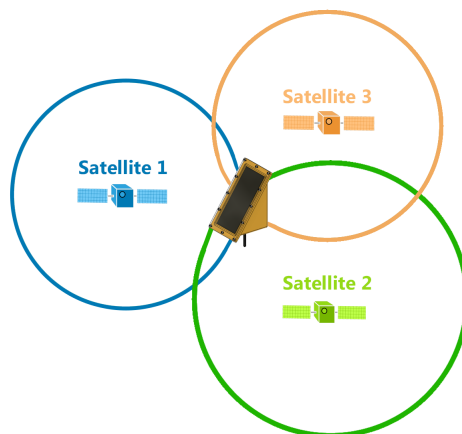
Supply Voltage	1.8 - 5V DC
Interface	$I^2C$ (up to 3.4 MHz), SPI (up to 10 MHz)
Operational Range	Temperature: -40 to +85°C Humidity: 0 - 100% Pressure: 300 - 1100 hPa
Resolution	Temperature: 0.01°C Humidity: 0.008% Pressure: 0.18 Pa
Accuracy	Temperature: $\pm 1^\circ\text{C}$ Humidity: $\pm 3\%$ Pressure: $\pm 1$ Pa

**Table 3.1:** BME280 technical data

Due to its size, power consumption, resolution and price this sensor is a perfect choice for the *Internet of Things*. The module is connected to the *ESP32* microcontroller via the  $I^2C$  interface.

## ■ GPS

The *NEO-6M GPS* [59] module is used to accurately determine the location of the designed device. The algorithm for determining the location by this module is based on the *trilateration method*, which measures distance (as opposed to the consonant term triangulation, which measures angle). To accurately determine the location of the device in *2D* coordinates (longitude and latitude, without calculating the altitude), the GPS module must receive a signal from at least three satellites.

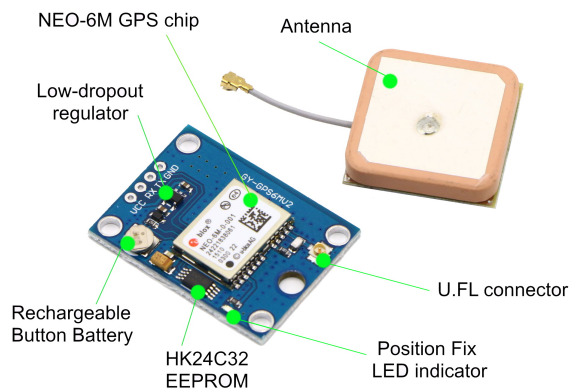


**Figure 3.4:** The principle of trilateration

Each satellite broadcasts information about its location and current time. The receiver in the GPS module catches this signal and calculates the distance (the radius of the circle in the center of which the satellite is located). Having

a signal from only one satellite, the device can be located at any point on a given circle. With a signal from two satellites, the number of possible positions is reduced to two (two points of intersection of two circles). By adding the signal from the third satellite, we get the exact location of the device, as the only intersection point of three circles.

The main component of the used module is the *NEO-6M GPS* chip from the Swiss company *u-blox*. The receiver of this chip has 50 channels that can track up to 22 satellites. Communication with the microcontroller is performed via the *UART* interface for serial communication. The default *UART (TTL)* baud rate is 9600.



**Figure 3.5:** Ublox NEO-6M GPS module

Position fix LED indicator displays the status of the GPS receiver. If the LED is blinking, it means that the position has been determined. Otherwise, when the LED is not blinking, it means that a search for a sufficient number of satellites takes place at the moment in order to determine the position.

The input supply voltage range of this module is  $3.3\text{ V} - 6\text{ V}$ . Low-dropout voltage regulator maintains  $3.3\text{ V}$  voltage. Also, there is a *MS621FE Lithium Rechargeable Battery* by which the calculated GPS data can be stored in the *HK24C32* two wire serial *EEPROM 4 KB* memory.

A patch antenna is used to receive GPS signals from satellites. The antenna is connected to the module through the *U.FL* connector. Antenna capture time is 1 second (*hot start*) if the GPS remembers its last calculated position, or 27 seconds (*cold start*) otherwise. The antenna itself has a flat appearance, which differs from the type of antenna for *LoRa* communication. This is because GPS wave has *Right Hand Circular Polarization (RHCP)*, in contrast to radio waves which is said to be linearly polarized. Tracking and navigation sensitivity of the antenna is  $-165\text{ dBm}$ . The position accuracy of this module is 2 meters, but it can be more accurate if there are multiple good signals from satellites. The supply current of the entire module is  $45\text{ mA}$ . But it is also possible to enable power save mode with a current consumption equal to  $11\text{ mA}$ .

## Power supply

A 3.7V single-cell *Li-Po* (*Lithium-Polymer*) battery [60] with a capacity of 2000 mAh was chosen as the power supply. The capacity of this battery is enough to power our device, it is also very compact, meeting the requirements of our device. The battery of this type is currently the most efficient while has a high energy intensity, much lighter weight, lack of a parasitic memory effect and is used in both mobile phones and electric vehicles.

In order to calculate the maximum current that will charge our battery we will use the  $0.5C$  rate as:

$$I_{max} = \frac{C_{battery}}{0.5} = \frac{2000 \text{ mAh}}{0.5} = 1000 \text{ mA} \quad (3.1)$$

$0.5C$  rate means that it will take us 2 hours to fully charge our battery with this current. This current value is recommended for charging, but not the maximum possible current. We can charge the battery with a higher current reducing the overall charge time, but firstly, at higher currents rates the battery will start to overheat increasing the internal resistance of the cell faster, resulting in a shorter battery lifespan, and secondly, the battery charger used in the system will not allow currents above 1000 mA.

Generally, an important point in using such batteries is a more sophisticated operation process, since in case of inaccurate charging and discharging, the battery may become unusable or explode.

The battery is charged using the *Constant Current/Constant Voltage* principle. On the practical side, this means that as soon as the battery reaches its peak voltage during charging, the current will slowly decrease, while the voltage level is maintained.

The battery discharge process is non-linear. A fully charged battery has a voltage of 4.2 V. Then, during operation, the voltage drops to the nominal voltage of 3.7 V - 3.8 V. This voltage provides the batteries with the most stable state, and when one buy a new *Li-Po* battery, it will be charged to this stable state. Then, after reaching 3.5 V - 3.6 V, the voltage begins to drop sharply, up to the discharge cut-off voltage at 3 V, when the battery is considered completely discharged.

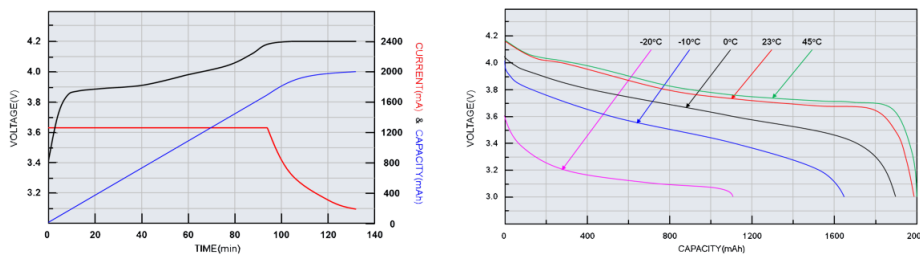


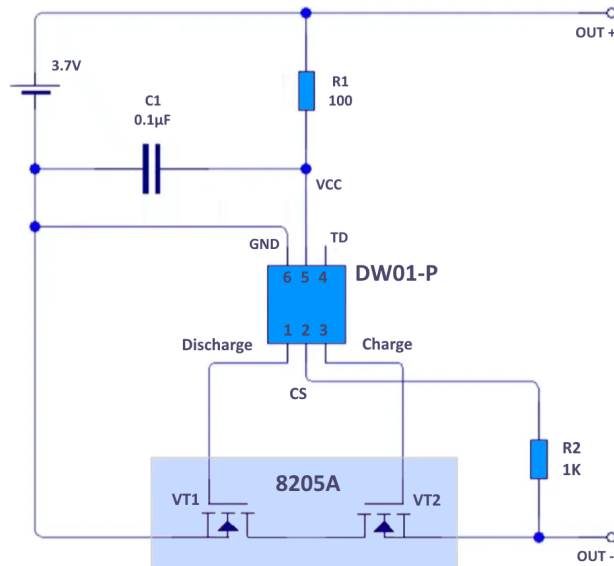
Figure 3.6: Li-Po battery charge-discharge curve [61]

To prevent different dangerous limit states during an operation, this battery contains an integrated battery protection circuit: *over-charge protection*, *over-discharge protection* and *over-current protection*.



**Figure 3.7:** Li-Po battery protection board [62]

The battery protection board imbedded in present battery contains two microcircuits - *8205A* and *DW01-P* series. The *DW01-P* controller integrated circuit is specially designed for lithium-ion/polymer batteries and protects them from damage or deterioration in service life due to overcharge, over-discharge and/or overcurrent of a single-cell lithium-ion/polymer battery. The *8205A* circuit consists of two *FS8205A* N-channel *MOSFET*s, which are controlled by the *DW01-P* circuit.

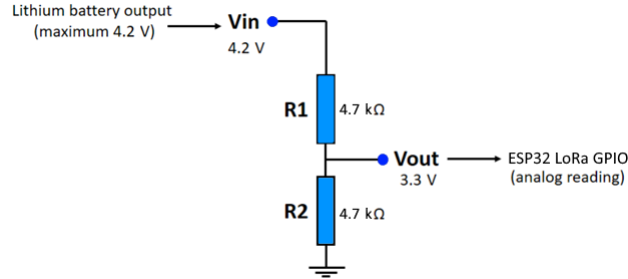


**Figure 3.8:** Battery protection circuit

The first transistor *VT1* manages the discharging process. If the battery voltage falls below the lower limit (2.8 V), then the *DW01-P* chip sends a signal that closes the *VT1* transistor, thereby disconnecting the battery from load. Second transistor *VT2* manages the charging process. If the battery voltage reaches the upper limit (4.2 V), then *VT2* closes, disconnecting the battery from the charger and the battery will stop consuming current. Also, overcurrent protection is implemented so that the *DW01-P* chip constantly monitors the discharge current at the *CS* pin. If there is a voltage of 150 mV on the *CS* meaning an overcurrent then the battery will go to sleep mode for 10 ms, disconnecting the load. If the *CS* voltage is 1.35 V meaning that the terminals are short-circuited then the battery will go into sleep mode faster (in 500 µs). So, when the output is short-circuited the battery instantly disconnects the load.



**Battery Voltage Monitoring.** In order to be able to track the battery level of every particular device and then display it for each device on the web, a voltage divider was used.



**Figure 3.9:** Voltage divider circuit

The principle of the circuit is to measure the output voltage of the *Li-Po* battery using the *ESP32* analog *GPIO* pin. Since the maximum battery voltage is 4.2 V, which is higher than the maximum operating voltage on the *GPIO* pin, that is 3.3 V, we will use the values of resistors  $R_1$  and  $R_2$  equal to 4.7 kΩ. According to the formula, the voltage on the *GPIO* pin at the maximum battery level will be:

$$V_{out} = \frac{V_{in} \cdot R_2}{R_1 + R_2} \quad (3.2)$$

So, in our case with a fully charged battery we get the following value of  $V_{out}$  that will be read by the *ESP32* module on the *GPIO* pin:

$$V_{out} = \frac{4.2 \cdot 4700}{4700 + 4700} = 2.1 \text{ V} \quad (3.3)$$

To obtain the correct battery level voltage, this value will be multiplied by 2 in the microcontroller program.

### ■ Solar panel

The solar panel works on the principle of the photovoltaic effect. It absorbs light photons from solar light, which then generate a direct current flowing inside the solar cell. Basically, we can imagine a solar panel as a DC generator powered by the Sun.

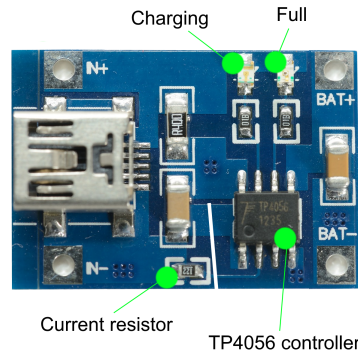
In order to find out the parameters of the solar cell we will use the next statement: the voltage is usually chosen to be 1.5 times the voltage of the battery pack. Therefore, we can write the necessary voltage of the solar cell as:

$$U_{cell} = U_{battery} \cdot 1.5 = 5.55 \text{ V} \quad (3.4)$$

Taking some extra reserve we can conclude that 6 V solar cell will be sufficient to maintain the battery voltage during the operation of the device. So, a solar panel with an output voltage of 6 V and power of 1 W was used. Besides, the additional advantage of this panel is its size. Since the solar panel will occupy a large surface of the case, the small area of the panel will make the size of the case for the device quite portable.

### Battery charger

When dealing with any lithium based battery, it is recommended to use special chargers to prevent damage of this battery type due to incorrect charging cycle. Therefore, *TP4056 mini-USB* [63] module was chosen for battery charging, which is a constant-current/constant-voltage linear charger for single cell lithium-polymer batteries.



**Figure 3.10:** TP4056 Mini Lithium Battery Charging Control Board

By default the value of charging current is set to  $1000\text{ mA}$  (what is exactly equal to the recommended charging current according to the documentation and  $0.5C$  battery rate), but it can be adjusted from  $130\text{ mA}$  to  $1000\text{ mA}$  by changing the resistance value of the current resistor. The default resistor mounted on the board is  $1.2\text{ k}\Omega$ .

Supply Voltage	5 V
Input voltage	4.5 V - 5.5 V
Accuracy in determining the charge level	1.5%
Charging current	1 A (Adjustable)
Full charging voltage	4.2 V
Operating temperature	$-10^{\circ}\text{C}$ to $+85^{\circ}\text{C}$

**Table 3.2:** TP4056 Technical data

Also, this module has two charging LEDs mounted on it: red LED indicates that charging is in progress, blue LED indicates the charging is completed.

### External timer

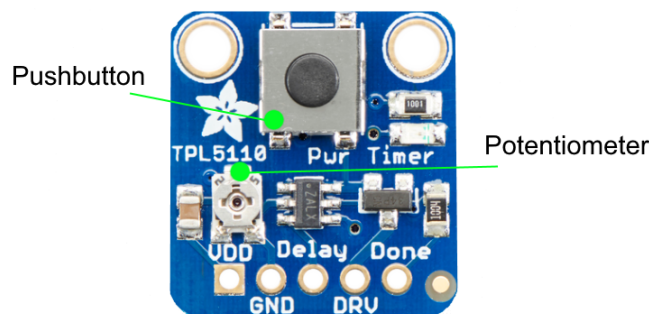
The *ESP32* module supports five sleep modes. The table below shows the data about the activity of individual modules in each mode, together with the current consumption. We can see that the minimum current consumption without considering the *LoRa SX1276* chip is  $2.5\text{ }\mu\text{A}$ . In order to achieve

even better battery savings it was decided to use the *Adafruit TPL5111 Reset Enable Timer* based on the *TPL5111* [64] from Texas Instruments. The operation of this timer is based on periodic alternation of the *HIGH* and *LOW* values on the *EN<sub>out</sub>* pin. The alternation frequency depends on the value on the *Delay* pin. By default, the *Delay* pin is connected to the trim potentiometer embedded into the module case. But it is also possible to connect our own resistor between the *Delay* and *GND* pins.

Mode	Powered – On	Consumption
Active	RF	160 - 260 mA
Modem-sleep	CPU, RTC, ULP co-processor	3 - 20 mA
Light-sleep	CPU (clock-gated), RTC, ULP co-processor	0.8 mA
Deep-sleep	ULP co-processor, RTC	10 $\mu$ A
Hibernation	RTC	2.5 $\mu$ A

**Table 3.3:** ESP32 Low Power Modes [56]

The value of this connected resistor will determine the delay for powering up. The frequency of applying the *HIGH* value to the *EN<sub>out</sub>* pin can be from *100 ms* to *7200 s*. After the *EN<sub>out</sub>* pin becomes *HIGH* (supplying the voltage equal to the voltage on the *VDD* pin), the timer waits for a signal from the microcontroller in order to set the *EN<sub>out</sub>* pin to *LOW* and stop supplying the voltage to the electrical circuit.



**Figure 3.11:** TPL5111 Low-Power timer

*DONE* pin is a signal pin that is activated by the microcontroller after the completion of the program execution. This pin is a flag indicating that the microcontroller is sending a request to disconnect the circuit from the power supply, thereby turning off the transistor on the module and cutting off the voltage supply. With the built-in pushbutton, we can manually start the power supply to the microcontroller. According to the datasheet, the time accuracy is 1%. Supply voltage range is 1.8 V - 5.5 V.

After adding this module to the circuit and measuring the consumption level of the entire system together with the *LoRa* module, the device's consumption in the sleep mode was only  $0.7 \mu A$ .

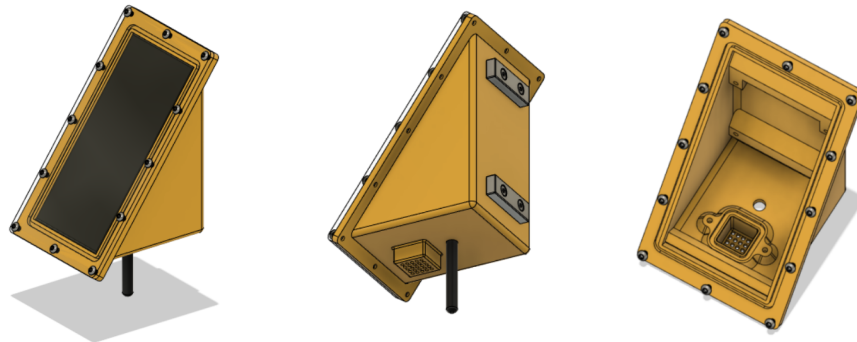
### ■ Antenna

Omnidirectional antennas with power gain of  $2 \text{ dBi}$ ,  $2.15 \text{ dBi}$  and  $6 \text{ dBi}$  were tested. In urban environments,  $2 \text{ dBi}$  and  $2.15 \text{ dBi}$  antennas showed almost the same performance.  $6 \text{ dBi}$  antenna outperformed the rest of the antennas in the signal range, but since, firstly, the existing gateway network in Prague is sufficient for use of  $2 \text{ dBi}$  antennas, and secondly, the size of the antenna itself is twice as long as the antenna of  $2 \text{ dBi}$ , 10 centimeters versus 5 centimeters respectively, reducing the compactness of the device as a whole, therefore, an antenna with  $2 \text{ dBi}$  gain was selected.

Omnidirectional type of antenna is more versatile, since the reception and transmission of a signal occurs equivalently from all directions, in contrast to a directional antenna, which can pick up a weaker signal from a certain distance, but at the same time reduces its ability to pick up a signal from all directions.

### ■ 3.1.2 Device Case

All components assembled in one circuit must be placed in a weatherproof case, which must be made in such a way that the device can be used outdoors. For the designing purposes the 3D CAD program *Autodesk Fusion 360* was used to model the case.



(a) : Case front view

(b) : Case back view

(c) : Unassembled case

**Figure 3.12:** 3D model of a meteostation case: device with assembled protective glass (a-b) and device case without embedded components (c).

Since the whole device is powered by a solar panel, the design of the entire case was based on the location of the solar panel to achieve maximum efficiency of its use. In addition, we needed to take into account the urban conditions in which the solar panel will operate [65]. Also, the angle of inclination of the solar panel depends on both the season and the hemisphere, while the

typical angles are  $20^\circ$ ,  $45^\circ$  and  $60^\circ$ . Based on the above, and given that the solar panel position on the device will not change throughout all seasons, it was decided to choose the tilt angle of  $45^\circ$  degrees. This arrangement is also suitable from the point of view of mounting the device, since it is intended for outdoor use where the device will be mounted on a vertical metal surface.

### ■ Color

The choice of color was due to the fact that the device was designed for outdoor use based on such definitions as *wavelength*, *reflection* and *absorption*. Light (e.g. visible light) is a form of energy called *electromagnetic radiation*. Visible light has wavelengths between 380 nm and 760 nm, with each color represented by a specific wavelength. The color of any object from a physical point of view means that the given object reflects an electromagnetic wave having a wavelength corresponding to the color of this object. And vice versa, it absorbs electromagnetic waves with wavelengths of other colors. Knowing that white color is a combination of all colors in the visible light spectrum, therefore, white color will reflect electromagnetic waves corresponding to most of the colors in the visible light spectrum. Accordingly, from the fact of absorbing less light energy, it follows that white objects absorb less heat (case design in the program is displayed in a darker color for better visual perception. The printing itself was carried out in white color).

### ■ Mounting mechanism

The mounting mechanism of the device is located on the back of the case and is represented by two neodymium magnets N38. This magnet type has a rectangular shape with dimensions of 40 mm(length)×12 mm(width)×6 mm(height) and mass of 18 grams. The demagnetization of this magnet is about 1% in 10 years. The adhesion force declared by the manufacturer is 9 kilograms. Each magnet has two holes for attaching the magnet to the case back with screws. In addition, each magnet is nickel-plated to protect against corrosion.

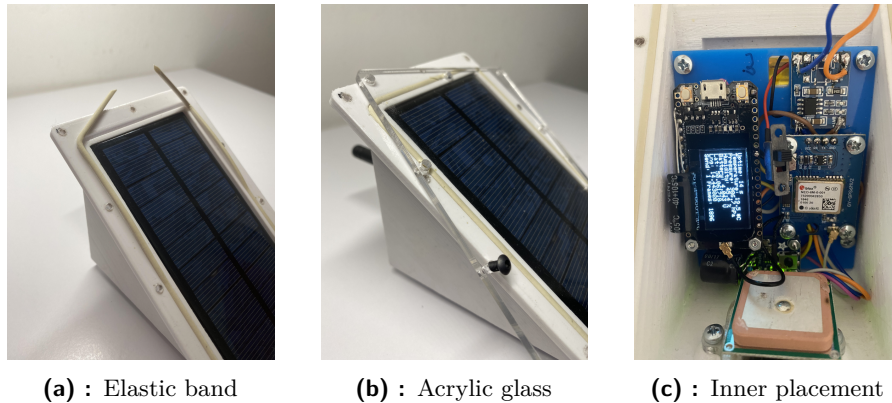
### ■ Component placement

The component placement is also determined by the position of the solar panel and the fact that the device will be mounted with the back of the case to a vertical metal surface. Another limitation of component placing is the requirement for the GPS antenna to be mounted parallel to the horizon to receive signals from the largest possible number of satellites. Therefore, the location of the BME280 sensor was on the bottom of the case. An antenna for data transmission via *LoRa* is located nearby. In order for the sensor to measure the actual temperature and humidity, it must have direct access to the air flow. For these purposes, a kind of cap has been proposed with many holes through which the air can flow to the sensor. Due to its placement on the bottom, the sensor can have open access to the environment and at the

same time be protected from precipitation. This hood is insulated from the interior of the case with an internal acrylic baffle. Due to this, if the sensor is flooded with water, the remaining modules will not be damaged. The GPS antenna is located directly on the horizontal acrylic partition inside the case. In addition, the specific placing of the sensor at the bottom was chosen in such a way as to move it as far as possible from the plane of the adjacent surface which the device is mounted to. This was done in order to reduce the effect of possible heating of the adjacent surface material in summer.

### ■ Solar panel protection

For the additional protection of the solar panel and the interior of the case from the effects of weather (e.g. rain, hail, storm), it was decided to add an acrylic glass cover on top of the solar cell. The advantages of this material are high light transmission, resistance to moisture and bacteria, low weight and frost resistance. This cover is attached to the case body with screws.



**Figure 3.13: Device appearance:** Additional elastic band for better tightness and adhesion to glass (a). Acrylic glass for extra protection to protect the solar panel and internal components from precipitation (b). Arrangement of components inside the device (c).

In addition, for better sealing of the entire device, a spot has been added for the installation of an additional rubber band. This band increases the contact of acrylic glass with the edges of the case, thereby preventing moisture from penetrating into the device.

### ■ 3D printing

The most common material used in 3D printing is plastic. There are different types of plastic with their own properties. Among filaments for outdoor use, there are two prevailing materials: *PETG* (*Polyethylene Terephthalate Glycol-modified*) and *ASA* (*Acrylonitril-butadien-Atyreen*). Based on the studied literature [66][68], *PETG* was chosen as the case material. It is a perfect choice for applications operating in extreme conditions without deforming shape or appearance [67]. Also, in addition to the price, the advantages of



this material include high strength, insensitivity to changing temperatures, ultraviolet resistance, much easier printing process due to lower printing temperatures compared with another filament type - *ABS*. Moreover, it is a recyclable and biodegradable plastic, unlike *ASA*.

As for the operating durability of this type of material, specialized forums were studied, where the clear preference was given to *PETG* plastic [69]. For example, *ABS* plastic in outdoor conditions lasted for about a year, while *PETG* operates to this day.

After all the analysis and calculations, the case was printed out and all the elements were placed inside.



**Figure 3.14:** Assembled devices

The entire device was then weighed. The weighing scale showed a mass of 334.79 grams.

### ■ 3.1.3 Device Software

A simplified version of C++ language known as *Wiring* was used for programming our device. This language does not actually exist, as there are no any *Wiring* compiler – programs written in this language are converted with minimal changes into a C/C++ language program, and then are compiled with a AVR-GCC compiler.

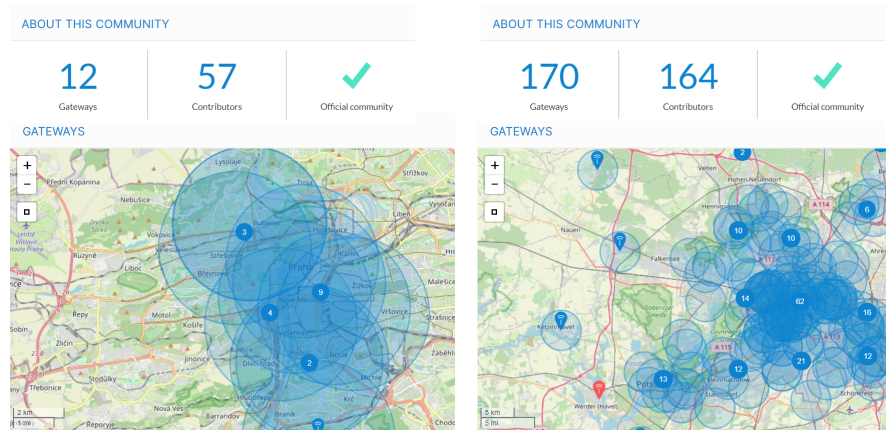
Implementation of the present program was developed using the *Arduino IDE* free environment that is a cross-platform application in Java, including a code editor, compiler and firmware transmission unit into the board. Mentioned IDE supports all popular operating systems such as Windows, MacOS X and Linux.

### ■ Implementation analysis

Before we start writing the code, it is needed to analyze the requirements for the program and the capabilities of the infrastructure itself at the place

where the network of meteorostations will be deployed.

**Infrastructure profile.** Prague has a sufficient number of public gateways for the deployment of a network of meteorostations performing its primary function, that is, sending measured data to the gateway. This communication is one-way, without confirmation messages indicating the reception of the packet on the gateway side.



**Figure 3.15: TTN infrastructure comparison:** TTN Community in Prague (on the left)[71] and in Berlin (on the right)[72]

Unlike other cities with a more developed infrastructure of gateways, in Prague the density of gateways is significantly lower than, for example, in Berlin, meaning more devices per one gateway. More devices sending data to one gateway increases the likelihood of collisions. In addition, if the gateway for some reason stopped working for some time and was rebooted, then this can lead to the fact that many devices will try to re-join the network at the same time leading to the network bloating in the area. This scenario is especially likely in areas where the density of gateways is low.

As it was already described in the theoretical part, after sending an uplink message, the device waits for a downlink from the gateway for some time. If the downlink has not been received, the device goes into sleep mode for a short time, after which it turns on and waits for a downlink message again. If the downlink message has not been received, then the device goes into sleep mode until the next scheduled activation for transmitting an uplink message with subsequent window for downlink.

According to the documentation of *LoRaWAN*, the waiting time for a downlink message before switching into sleep mode (*ACK\_TIMEOUT*) is from 1 to 3 seconds. In case of not receiving a downlink during two waiting windows, we get from 2 to 6 seconds of additional operating time meaning higher energy consumption during an hourly working cycle and a reduced operating time of the device as a whole. Moreover, even in the case of some packet loss, it is much easier to restore the missing value on the application side than to resend the package again by node, since the dynamics of the



meteorological parameters is smooth. During testing of the entire system, only 1 packet was lost from all devices altogether.

**TTN Fair Access Policy.** But the main limitation is *The Things Network's* public community *Fair Access Policy* [73]. Since the network is open, where devices are not statically-linked and send data according to their needs (i.e. *ALOHA*-like protocol), the following limitations were introduced [74]:

- Average uplink air time is equal to 30 s per device per 24 hours.
- The maximum packet size is 51 bytes.
- The maximum number of downlink messages is equal to 10 per device per 24 hours, including the *ACKs* for confirmed uplinks.

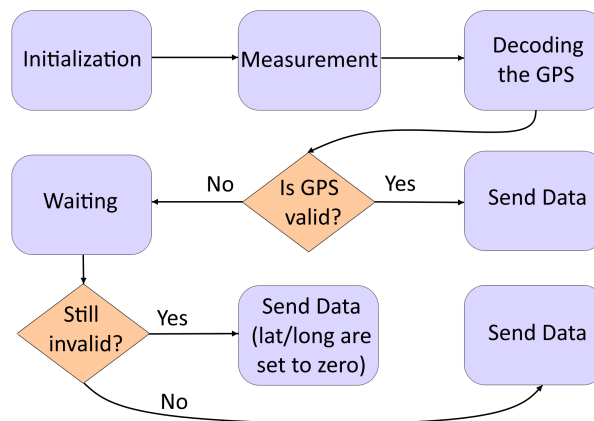
According to these points, we can see that since the data is sent with a frequency of one hour, in total we will send 24 packets in 24 hours. So, in any case, we will not be able to accompany each data packet with confirmation message. And it makes no sense in this case to waste energy on confirmation of 10 packets, while 14 packets will still remain without confirmation.

In conclusion, *TTN* recommends not to use downlink messages as far as possible, since not all gateways are able to receive messages from end devices while the gateway itself is transmitting.

### ■ Program flow

The program implements the formation of data packets with the subsequent sending of an uplink message to the *TTN* network. In addition, there is logic for working with a watchdog timer, reading data from sensors and measuring the battery voltage level using a built-in 12 bit analog-to-digital converter.

Based on the points described in the analysis part, it was decided to refuse using confirmed uplink messages, but send data and immediately switch to a sleep mode.



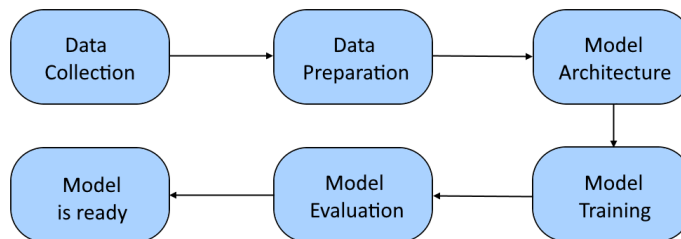
**Figure 3.16:** Logical structure of the program running on ESP32



indicating that microcontroller can turn the *TPL*'s power transistor off and disable power to the rest of the circuit.

## 3.2 Prediction Model

The process of designing and creating a neural network model consists of several parts. One of the most crucial components of any model is obtaining of a good training dataset. After that, the process continues with data preparation and the subsequent model's architecture design together with the selection of hyperparameters.



**Figure 3.17:** Machine Learning model implementation

After the process of training the model is finished, the results are evaluated and a decision is made - either the model needs to be improved and retrained, or the model meets the requirements of the project. The result of the development process is a model that is ready for deployment in real-world environment.

### 3.2.1 Training dataset

The training of any neural network starts with the selection of a training dataset. The quality and quantity of training data largely determines the properties of the model and the accuracy of the forecast. The data should also be relevant to the region in which the developed meteorological model will be used.

The training dataset was obtained from the *Czech Hydrometeorological Institute*. The service of providing historical data from selected stations is generally paid, but for educational purposes the data is provided free of charge. The *Praha - Klementinum* station was selected as the data source. Hourly data were obtained for three parameters: air temperature, relative humidity and atmospheric pressure. Historical data was obtained for the period from 01/01/2015 to 12/31/2020.

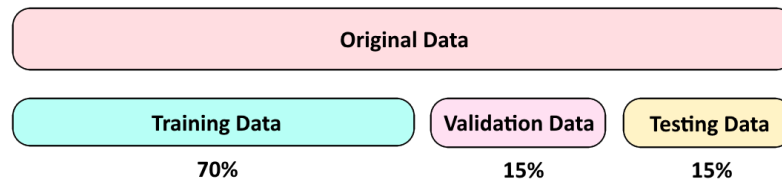
### 3.2.2 Data preprocessing

Data preprocessing is a crucial step before training each neural network. Thus, we organize, cleanse and format the real world data in order to present

them in a suitable understandable form for the created model, which will be able to learn from the input data.

### ■ Splitting the data

Before starting to train the neural network, we need to split the initial data into several parts in order to avoid overfitting and underfitting.

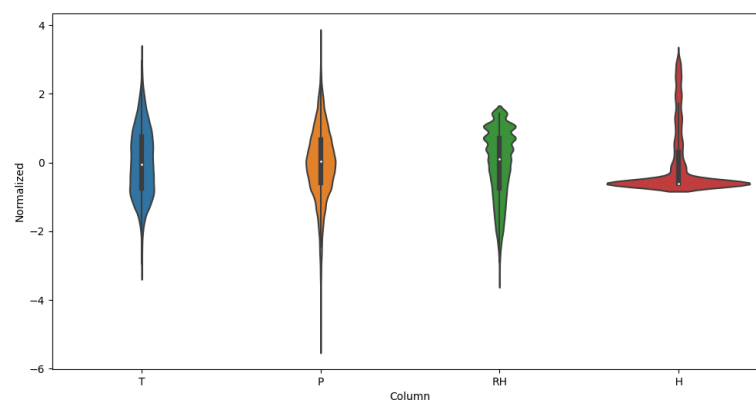


**Figure 3.18:** Dataset splitting

- *Training Data* is used to train the model.
- *Validation Data* is data that we did not use in the process of training the model and serves for preventing overfitting and for selecting proper hyperparameters of the model.
- *Testing Data* is used to determine the final accuracy of the model.

### ■ Data normalization

An important step before starting the training process is to scale the features. This is done through a normalization process, since neural networks train better when the input values have similar magnitudes.

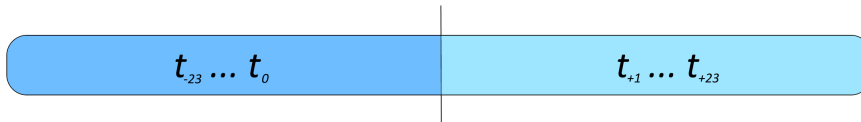


**Figure 3.19:** Data normalization

So, prior to passing the measured data through the model, we subtract the mean value and divide by the standard deviation value of each feature first.

### ■ Data windowing

Our model will generate a prediction based on sequential data immediately preceding the predicted time interval. Here we need to find a balance between the accuracy of the forecast and the length of the period of data accumulation by the newly mounted device to generate the first forecast. Also, we need to take into account the time horizon for which we want to make a forecast.



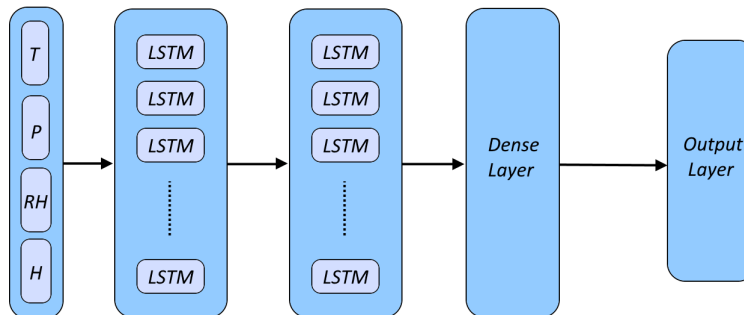
**Figure 3.20:** Data distribution over time intervals

Based on the studied literature specified in the theoretical part, and the need to maintain the fastest possible start of operation in new environmental conditions after a possible relocation of the node, it was decided to make a forecast for the next 24 hours using measurements from the past 24 hours.

Of course, in order to increase the prediction accuracy we could use both 48 past measurements or 72 or even more, but by doing so we increase the time interval between the deployment of the device and the generation of the first forecast.

### ■ 3.2.3 Model architecture

The creation of a model itself was implemented using *Keras* library and the *Sequential* class. An object of the *Sequential* class is a container to which the used layers are sequentially added. For the purposes of our project, we implement the *Stacked LSTM* model meaning that there will be multiple hidden LSTM layers.



**Figure 3.21:** Model structure

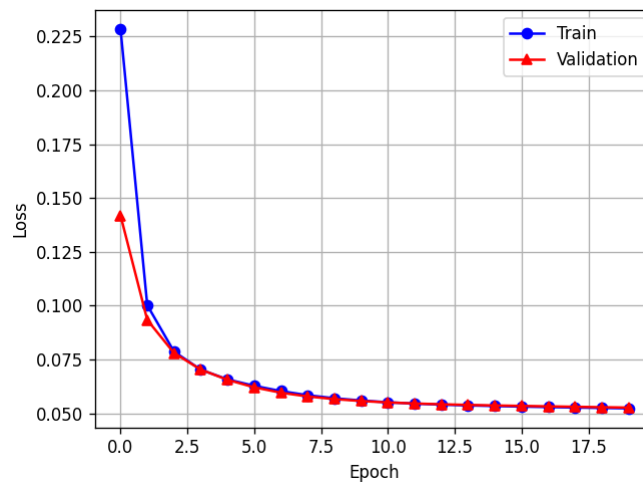
Four parameters are passed to the input of our model: temperature, relative humidity, pressure and the calculated additional auxiliary parameter of the value of solar irradiation received by surface. These parameters form the *Input Layer* of our model. This layer is then followed by two *Hidden Layers* consisting of 16 neurons each. Finally, there is a *Dense Layer* and standard

feedforward *Output Layer* that represents the predicted values of temperature, humidity and pressure.

### 3.2.4 Compilation and training

While training the model, we will use the following hyperparameters:

- *Activation function*: within the structure of the LSTM cell itself there are already three *sigmoid* activation functions and two *tanh* activation function used, so there is no need to add an additional activation function after the *LSTM* cell, since there is enough non-linearities within the structure.
- *Learning rate*: 0.001
- *Batch size*: 256
- *Cost function*: MSE
- *Early stopping*: the training process will be terminated after five iterations without decreasing the *MAE* value
- *Optimizer*: *Adam* optimizer will be used.



**Figure 3.22:** The training and validation loss while training

The algorithm of the model future operation will be in passing the input data sequentially through the model, that is, generating the forecast for one hour ahead first, then for two hours and so on, until a forecast for 24 hours is generated.

### 3.2.5 Model validation

In order to evaluate the performance of our model we will use two metrics:

- *Mean Absolute Error* (MAE) represents the average of the absolute difference between the actual value and observed value:

$$MAE = \frac{1}{n} \sum_{j=1}^n |y_i - \hat{y}_j| \quad (3.5)$$

We can conclude from the equation that all individual differences are weighted equally in the average and the MAE value follows a linear behavior.

- *Root Mean Square Error* (RMSE) represents the standard deviation of the residuals:

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_i - \hat{y}_j)^2} \quad (3.6)$$

Here we can see that RMSE stronger penalizes large errors and reduces minor deviations since the residual is squared. The RMSE will always be larger or equal to the MAE.

Both *RMSE* and *MAE* can be used together to better understand the errors variation in the forecast. The final accuracy of the trained model was calculated on testing data.

	MAE	RMSE
Temperature (K)	0.88	1.20
Atmospheric pressure (Pa)	63.48	85.03
Relative humidity (%)	4.39	6.13

**Table 3.4:** Mean errors in the test dataset for 1-hour forecast

The model was additionally tested without the inclusion of the parameter of solar irradiation received by surface. According to the obtained results, it was proved that the calculation and addition of this parameter improved the forecast accuracy by as much as 11% and 10% for temperature and humidity, respectively, since these parameters are affected most by the level of solar radiation. Also, forecasts were calculated during the year of the test dataset to display the forecast accuracy of the model (24 predictions · 365 days = 8760 steps).

	T	P	RH
MAE (%)	-11.54	-3.97	-10.20
RMSE (%)	-11.27	-3.96	-8.44

**Table 3.5:** Improvement in prediction accuracy

### 3. Implementation



**Figure 3.23:** Model performance in the test dataset



## 3.3 TTN-AWS Connection

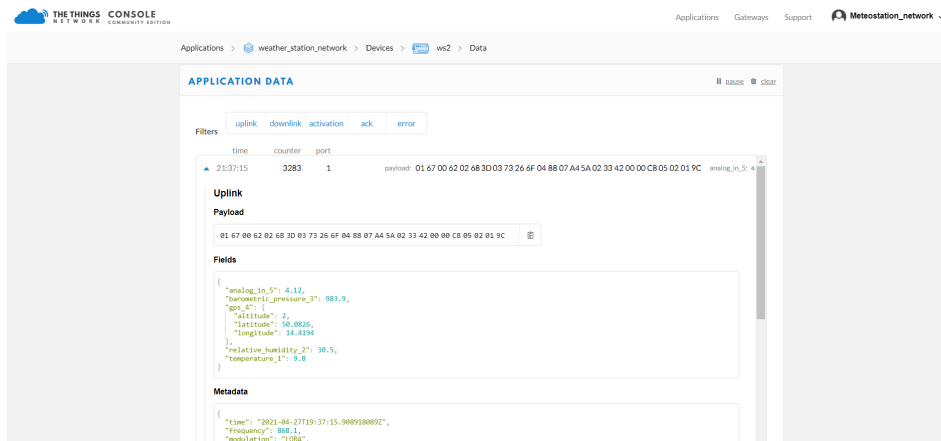


Figure 3.24: Received uplink packets showed in the TTN Console

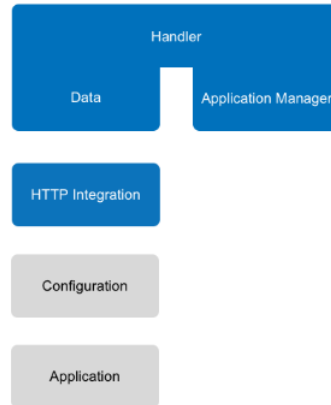
In order to connect end devices with the application, we will use build-in *Integration*. *Integration* is a way of establishing a connection between the end application and *The Things Network*. Among the various forms of integrations, the most used is the method of redirecting uplink messages to some webhook or other messaging endpoint. For our purposes, the *HTTP integration* was chosen, which acts as a bridge between the *Handler Data API* and any endpoint we configure. Messages are redirected as they arrive, that is, as soon as the message has been received by the *TTN Application Server*, it is immediately redirected to our application's endpoint via *HTTP* integration. In addition, this integration provides an endpoint on the *The Things Network* side, through which it is possible to send downlink messages in the opposite direction - from the application to the device.

### 3.3.1 TTN side

The main components of HTTP integration are:

- *Handler* is a geographic indicator of the application location (in our case - *ttn-handler-eu*). *Handler - eu* means that all *LoRaWAN* packets will be forwarded to the *eu* region of *The Things Network*.
- *Application Manager API* provides methods for integrating *The Things Network* stack into an application. Also, it is used for managing applications and devices registered with *The Things Network* in general.
- *Data API* is used to send and receive messages. We can work with *Data API* through various SDKs (Java, Python, Node-RED) or directly through MQTT.
- *HTTP Integration* is our created integration.

- *Configuration* where we configure the endpoint and *HTTP* method of our *HTTP Integration*.
- *Application* is an application on the server the endpoint of which data is sent to.



**Figure 3.25:** TTN HTTP Integration [70]

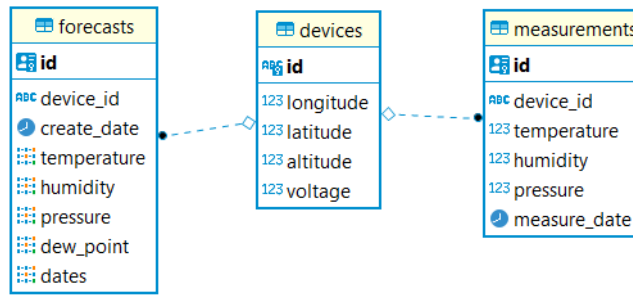
The following fields have been configured:

1. *URL* is the address of the target endpoint. In our case, the endpoint is registered on the Amazon server.
2. *Method* - *HTTP POST* method is used to send data to the server. Messages are sent in *JSON* format.
3. *Authorization* allows us to configure an authorization key for sending messages. This is done so that unauthorized devices cannot send data to the address of the created endpoint.

### ■ 3.3.2 AWS side

The first step was to create an *EC2* instance. Instance type was chosen as *t2.micro*, since, firstly, a free trial period applies to it and, secondly, the provided resources are sufficient for both the current use and future system extension. Also, after the expiration of the free trial period, based on the allocated resources and the general pricing policy, the costs of future instance using will be minimal. This instance works under *Linux* operating system. It has a number of virtual CPUs equal to one and memory size also equal to *1 GiB*. *Elastic Beanstalk* was also configured for a more convenient process of deploying and maintaining our application.

For the purposes of the project, a database was created utilizing the *PostgreSQL* object-relational database system. The overall database storage size is equal to *8 GB*. The database consists of three tables.



**Figure 3.26:** Database entity relationship diagram

The *Devices* table contains information about separate devices, including the device name, current device coordinates and relevant battery voltage level. The *Measurements* and *Forecasts* tables contain information about the last values measured by every separate device and the generated predictions, respectively.

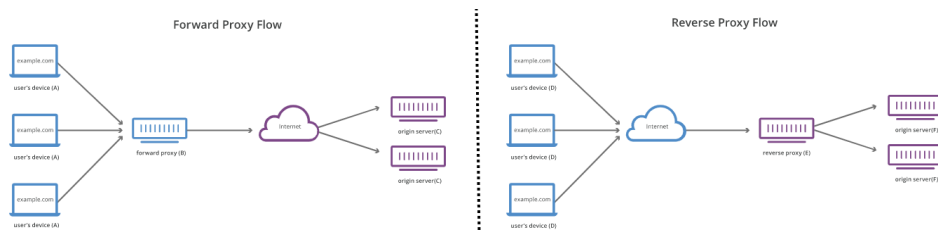
Unique identifiers for the record of each device measurement in the *Measurements* table and records for each generated forecast in the *Forecasts* table are generated using the `uuid_generate_v4` function.

## 3.4 Deployment

Our application is deployed on the *AWS* server. By itself, the structure of the overall solution on the cloud can be divided into three parts:

1. a part representing our implemented forecast application.
2. a part responsible for processing incoming traffic, the so-called proxy server.
3. and a part representing the client of our application. In our case, these are device owners who will monitor the device state and forecasts through the browser; as well as incoming uplink messages from the *Things Network Server*.

The meaning of both first and last points is clear to everyone, but the definition *proxy server* should be described in a little more detail.

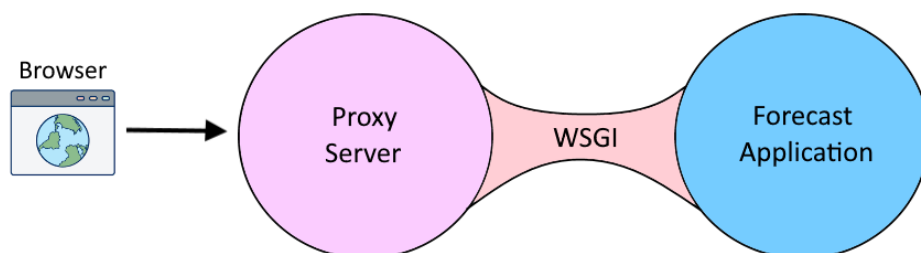


**Figure 3.27:** Proxy server working principle [75]

*Proxy server* is an intermediate server between the user and the target server (i.e. *origin server*) acting as an intermediate node handling users requests. There are two types of proxy servers: *forward proxy server* and *reverse proxy server*. *Forward proxy server* is located between the one or more users and the *World Wide Web*. The main purpose of this server is to filter passing traffic according to the configured rules in order to avoid direct origin server communication to the user. Unlike the *forward proxy server*, the *reverse proxy server* is located between the *World Wide Web* and one or more origin servers, meaning that no direct traffic goes straight from the user to the server, but only with passing this intermediate node. Among the main purposes and advantages of using this server are: caching, load balancing and protection from attacks [75].

Among the most popular reverse proxy servers at the moment are *Apache* and *Nginx*. After analysis, the choice fell on *Nginx*. *Nginx* is an almost ready-made solution for many tasks that require the deployment of a full-fledged web server and proxy. *Nginx* outperforms *Apache* in a number of ways. Among the main ones are the lack of resource requirements and the ability to handle a large number of connections at the same time. This is achieved by creating worker processes, each of which can handle thousands of connections. Each connection processed by the worker is placed in the event loop along with other connections. In this loop, events are processed asynchronously, allowing tasks to be processed in a non-blocking manner. This approach of handling connections allows *Nginx* to scale incredibly with limited resources. Since the server is single-threaded and does not create processes for every connection, the memory and CPU usage is relatively stable, even under high loads.

The forecast application itself represents a web server implemented using *Flask* microframework. But the standalone *Flask server* is not enough for the production deployment where we require our application to be able to handle multiple users and multiple requests in order to avoid situations when users wait for the page to load. Therefore, our application will be handled by a *Waitress WSGI (Web Server Gateway Interface)* application server that is a production-quality pure-Python with good performance.

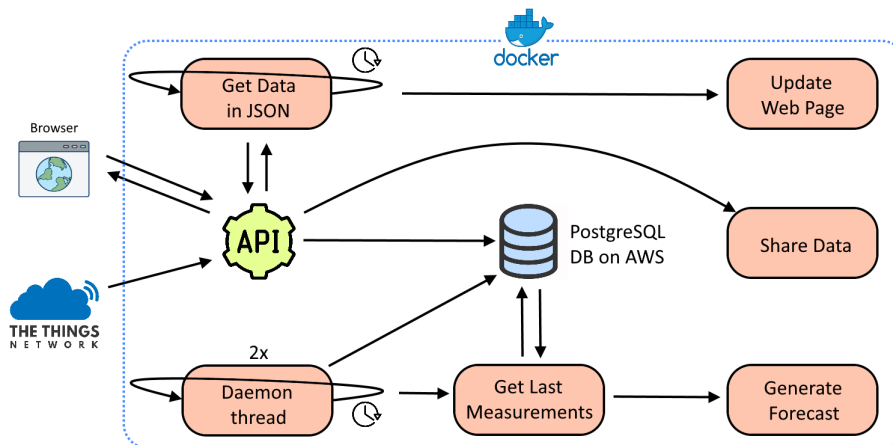


**Figure 3.28:** Application deployment structure on the server

*WSGI* represents an interface specification according to which the proxy server and our forecast application communicate with each other. While frameworks are not designed neither to handle thousands of requests nor to

route them from the proxy server in a best way, *WSGI* servers can deal with an enormous number of concurrent requests.

The general concept of the *Forecast Application* can be divided into two parts: backend and frontend. The backend is responsible for the logic of working with the database, processing incoming browser connections and uplink messages sent from *The Things Network*, generating forecasts, sharing measured data with community weather monitoring projects and data preparation for displaying on the web. The function of the frontend is to display devices on the map along with the last received data for each device, as well as display the generated forecasts.



**Figure 3.29:** Forecast application workflow

The entire *Forecast Application* runs in a *Docker* container. Like a virtual machine, *Docker* runs its processes in its own pre-configured operating system. But at the same time, all docker processes run on a physical host server, sharing all processors and all available memory with all other processes running on the host system. *Docker's* approach is halfway between running everything on a physical server and the full virtualization offered by virtual machines using *Hypervisors*. Among the many advantages of using containers, three main ones can be underlined: a time-consuming process of installing an entire OS to a virtual machine (i.e. *VirtualBox*) is omitted; container itself is extremely portable and will run in a similar way regardless the host OS they are executed on; *Docker* creates a consistent environment that can be easily ported to production without the need for a lengthy development environment re-creation process.

### ■ 3.4.1 Backend

As described earlier, the backend can be conditionally divided into two parts:

- *API part* handling both external incoming requests and requests related to the logic of the application including data preparation for displaying on the web page and data sharing.



Path	Method	Description
/measure	POST	Saves and shares uplink data
/devices	GET	Returns list of all devices together with all corresponding data
/devices/<id>	GET	Returns information regarding a specific device
/devices/<id>/measurements/last	GET	Returns last measurements of a device specified
/devices/<device_id>/forecasts	GET	Returns all forecasts for a specific device
/devices/<device_id>/forecasts/<forecast_id>	GET	Returns specific forecast for a device specified
/devices/<device_id>/forecasts/last	GET	Returns last forecast for a device specified
/favicon.ico	GET	Returns application icon
/	-	Returns an HTML file

Table 3.6: API description

### Forecast generation

The forecast generation process is started by the daemon thread at regular intervals, in our case every hour. Using daemon threads is appropriate when some background task needs to be performed, since we can start the execution of the thread and leave it running without having to keep track of it.

**Daemon threads.** There are two daemon threads used within the developed system. The first thread starts the process of generating a new forecast for every device every hour. The second thread starts every 7 days and serves as a cleanup thread that cleans up forecast and measurement data in the database in order to avoid database storage shortage in case the number of connected devices is large, since this data is not used after the forecast is generated and displayed on the web page.

**Period check.** After the forecast flow is started, a request is made to the database to retrieve measurements for the last 24 hours for each device. If the device has just been mounted and has not passed the *"measurement accumulation period"* of 24 hours, the forecast will not be generated and empty lists of values will be passed for displaying. But the last measured data from every meteostation will still be sent to the web for displaying by clicking on the device marker.

**Interpolation.** In case of TTN uplink data packet loss, a gap in measured values for a certain hour is produced. In order to recover missing values we use linear interpolation. The algorithm works in such a way that we make a

request to the database, and if there are no measured parameters for some hour for some of the devices, then we designate these missing values as *NaN*, and send an array of all values containing both *NaN* and measured values to the function, which returns an array without missing values. The values are restored based on the existing measured parameters for the adjacent hours. This solution is appropriate, since the weather in Prague does not change with huge sharp jumps, so interpolation is quite capable of accurately restoring the missing values.

Finally, the data is fed to the input of a machine learning model, which produces an array of predictions for three parameters: temperature accompanied by the calculated values of a Dew Point for every temperature point, sea-level atmospheric pressure and relative humidity. *Dew Point* is calculated according to the following equation [77]:

$$T_{DP} = \frac{237.7 \cdot \left( \ln\left(\frac{RH}{100}\right) + \frac{17.62 \cdot T}{237.7 + T} \right)}{17.62 - \left( \ln\left(\frac{RH}{100}\right) + \frac{17.62 \cdot T}{237.7 + T} \right)} \quad (3.8)$$

where:

- $RH$  = measured relative humidity [%]
- $T$  = measured temperature [°C]

The predicted data together with additional information for each device including relevant measurements is then sent in *JSON* format to the frontend.

### ■ Source code

*model\_train.py*. Neural network model's creation and training. This model is then used to forecast meteorological values.

*model\_predict.py*. This module uses the *db\_helper.py* module's function to retrieve last measurements for each individual device and generate forecasts.

*prediction\_thread.py*. This module contains the *PredictionThread* class, which is implemented in the form of a daemon thread running continuously on the background. This thread is inactive the majority of time and wakes up every hour in order to make a request to the *model\_predict.py* module and get an updated forecast.

*cleanup\_thread.py*. This module contains the *CleanupThread* class that performs regular data cleaning.

*db\_helper.py*. This module contains the logic for working with the database:

- function of connecting to the database and data cleanup functions.



- functions of parsing incoming uplink messages from *The Things Network* with the subsequent saving of measurements to the database along with creating/updating devices' records.
- functions of retrieving the list of last measurements of each separate device for a configured period of  $n$  hours. Output of this function serves as an input parameter for the forecast generation function.
- dew point and sea-level pressure calculation functions.
- function to retrieve the last measurement for each device to display this information within the device marker on the map.

*application.py*. Represents implemented *API* service.

*share\_data.py*. Function to share received measurements with community weather monitoring projects.

*interpolation.py*. Interpolation function implementation.

*data.py*. Creation and definition of classes *Measure*, *Forecast*, *Device*.

*waitress\_server.py*. This module represents a *WSGI* that handles our *Flask server*.

*Docker file*. This file contains commands to run our application in a *Docker* container.

### ■ 3.4.2 Frontend

Web page design was developed using *Bootstrap Studio* software. This software supports *HTML*, *CSS*, *JavaScript* and allows one to create responsive web pages. The frontend itself can be divided into several constituent parts:

- displaying devices on the map together with relevant information.
- the mechanism of displaying devices depending on the status of the device.
- displaying generated forecasts for each device.

#### ■ Current device status

The device itself can be in one of three states. The following logic was implemented in which the device marker changes its color depending on the device status:

- *Green marker* status in which data is received without lossage and the battery voltage level of the device is sufficient.
- *Orange marker* status when one of the following conditions is met:

### 3. Implementation

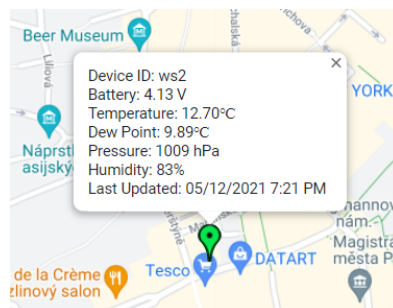
- the data has not been received from the device for more than 1 hour but still less than 5 hours. This values were determined experimentally, since the interpolation of missing values for a period exceeding the last five measurements does not work accurately enough.
- the battery voltage level is 3.7 V or less. This value was selected based on the non-linear *Li-Po* battery discharge curve. Because when the battery is further discharged after the nominal voltage of 3.7 V, the subsequent discharge goes on very rapidly, so the user should be warned in advance.
- *Red marker* status in which data has not been received from the device for more than 5 hours. So, the red marker color can indicate either the absence of a signal, or the complete discharge of the device.



**Figure 3.30: Device states on the map:** Sufficient battery voltage and data are sent properly (a). Low device battery voltage (b). The device is completely discharged or there is no signal at all (c).

#### ■ The latest data

Each device is represented by a marker on the map. When user clicks on the marker of a specific device, information about the last measured data, battery voltage level, last update time and the identifier of a specific device will be displayed.



**Figure 3.31: Device info**

## Generated forecast

The weather forecast is generated for each device separately. With the help of implemented pagination, we can switch between the predictions of individual devices. When user switches forecasts with pages, the device marker indicating the specific device the forecast is generated for is zoomed to.

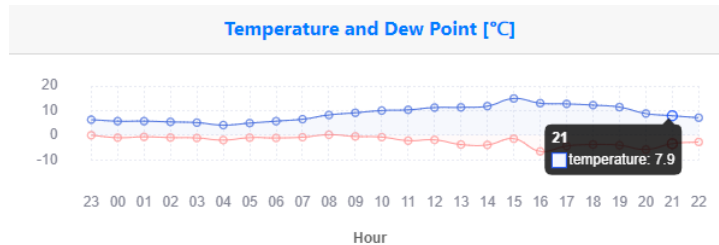


Figure 3.32: Generated forecasts

If enough data was not yet received for the forecast, meaning that the 24-hour period of data accumulation has not passed, then the charts will be empty. As soon as the first forecast is generated, the forecast charts will be built. At the same time, it will still be possible to find out the last measured data by clicking on the markers of individual devices on the map, before the first forecast is generated.

Each forecast chart is click-responsive. For example, when user clicks on a certain point, the value and hour of the forecast are displayed.

The temperature chart contains two sets of values at once - the air temperature forecast and the corresponding dew point value.

## Source code


*index.html*. It is the basis of the web page. It is able to automatically adjust screen resolution based on the used device used. This module displays the visual components:

- map with displayed devices in the form of markers with corresponding device information.
- plots displaying the predicted values for each device.
- pagination to switch between devices.

*theme.js*. This module is responsible for all the logic of the frontend:

- functions responsible for the page's response to user actions and map initialization.
- functions responsible for displaying device information after clicking on the marker and changing marker colors.
- function responsible for updating values on charts.
- function responsible for updating all data on the page.





## Chapter 4

### Data sharing

There are many commercial and non-commercial projects that collect data from amateur and professional meteorological stations. Many of these projects offer their users access to local forecasts based on the collected data. By sharing the measured data, the weather station owner not only contributes to improving the accuracy of such forecasts, but also often has an access to additional analytics on their data. For example, peak temperatures over time, average of the measurement values and comparison with other stations nearby.

Globally, all community meteorological projects can be roughly divided into:

- those that only support professional stations, allowing data to be uploaded only after both a specific manufacturer's device together with a serial number are identified.
- those that are available to both professional stations and non-commercial ones.

Our developed station belongs to the second type of projects. In addition to the division described above, community meteorological projects can also be divided into two subtypes:

- station registration can be implemented in an automatic way using requests to the available API service. An unambiguous advantage of this method is that the entire system can be programmed in such a way that the entire process of data sharing is performed fully automatically when additional devices are added to the system.
- it is necessary to register each station manually. The disadvantage of this approach is the need to add each new station manually on the site of the provider of a particular project.

Within the framework of this project, data sharing of measured values was implemented by each of the methods.

## 4.1 Sharing data with Weather Observations Website

The *Weather Observations Website (WOW)* is an open platform for sharing of current weather observations [79]. It was launched in 2011 in the United Kingdom with the support of the *Royal Meteorological Society*.

When creating a new station, one needs to specify its geographic coordinates and assign an *Authentication Key*. After registration, each station receives a unique identifier. *Authentication Key* and *Station ID* are used for subsequent data transmission.

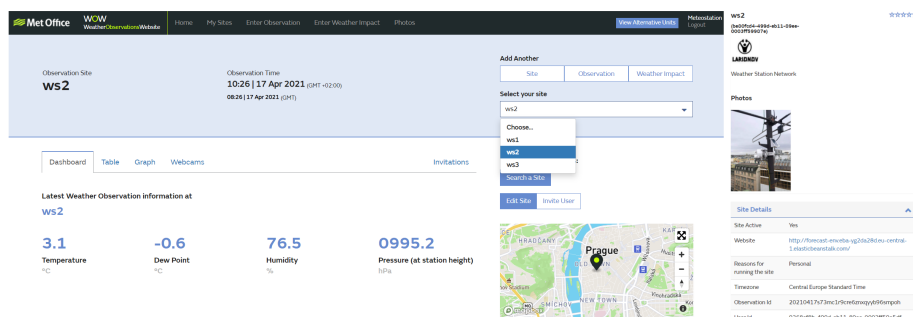


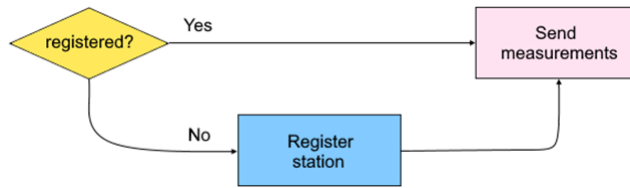
Figure 4.1: Shared data showed on the WOW

The data is sent using a *HTTP POST* request. Information about existing meteorostations is coded into *dict* data structure in the *share\_date\_wow* function where each station name (*ws1* - *ws3*) has a matching *ID* and *Authentication Key*. The disadvantage already described above is that new devices are registered manually on the provider's website, and the existing *API* does not allow to make a request in order to determine existing stations or add new ones. It means that after adding a new station, it will be necessary to update the source code on the Amazon server.

## 4.2 Sharing data with OpenWeatherMap

*Open Weather Map (OWM)* is an online service that was launched in 2012 [80]. This service provides a convenient *API* for receiving and sharing current local meteorological data, historical data and weather forecasts. In addition, *OpenWeatherMap* is a provider of weather information in applications from Google and Samsung.

The provided *API* allows us to register new devices and send and receive data as well as additional information about the device.



**Figure 4.2:** Workflow of sharing the data with OWM

Sharing of measured data is implemented using the *share\_data\_owm* function. Each device is associated with a specific account. Upon registration, a new device receives a unique internally generated system identifier. Furthermore, the station names can be repeated (*ws1 - ws3*) meaning that it is impossible to refer to a specific station by its name. Therefore, for maximum automation and in order not to hardcode the existing stations' identifiers and to make it easy to add new stations to the network, a *HTTP GET* request is first made to get a list of all stations associated with our account. Then, comparing the registered station names with the current one, we determine if it has already been registered or not. Based on the result, we either immediately send the measurements to the service, or we first register the station and then send the measurements. Also, it is necessary to check that the station coordinates are not zero when registering a new station.

With this algorithm, the sharing of data from a new meteostation is fully automated meaning that there is no need to add every new station manually or update the code on the Amazon server's side.





## Chapter 5

### System evaluation

In this chapter, the achieved results of the entire project will be discussed in detail. The evaluation of the results is divided into two parts: a description of the final product as a whole and a comparison of the developed solution with commercial products.

#### 5.1 Results

In this part, the results obtained will be analyzed, including the characteristics of the device and a description of the testing and operation of the network as a whole.

##### 5.1.1 Device lifetime

In order to calculate the approximate battery life, we first need to determine the individual consumption of each of the components:

- *ESP32* - according to the datasheet, board's consumption in *Active Mode* without *LoRa* chip is *160-260 mA*, - that is provided that all integrated modules work (e.g. *WiFi*, *Bluetooth*). The minimum consumption of this module can be achieved in *Hibernation Mode* and is equal to *2.5 μA*.
- *LoRa chip* - in order to calculate the separate current consumption of the *LoRa* module we used the *SX1276* module's datasheet from where we can see that for the very good signal conditions (+20 dBm) the current consumption will be equal to:

$$P = 100 \text{ mW} \quad I_{LoRa} = \frac{U_{ESP32}}{P} = \frac{5 \text{ V}}{0.1 \text{ W}} = 50 \text{ mA} \quad (5.1)$$

In the case that the gateways will be located further, or any barriers on the signal path or bad weather conditions, more current and power will be required to send the data.

- *GPS* - according to the datasheet, the consumption of this module in normal mode is *45 mA*. But it is possible to switch module to power save mode in which the consumption will be *11 mA*.



the ESP32 module was selected based on the average consumption value, taking into account the fact that we do not transmit data through Wi-Fi or Bluetooth but at the same time, there is an energy consumption for powering the display. Also, this value includes a possible higher consumption by the *LoRa* module, in case of unfavorable environmental conditions. The duration of the transmission of data by the *LoRa* module was calculated using an online calculator that [76] takes into account both spreading factor, bandwidth and payload length (26 bytes in our case).

Now we can calculate the average current consumption during one hour as:

$$I_{avg} = \frac{I_{ON} + T_{OFF}I_{OFF}}{T_{ON} + T_{OFF}} = \frac{6126 + 2.5}{3600} = 1.7 \text{ mA} \quad (5.6)$$

Finally, the battery life of the system can be calculated as:

$$t_{life} = \frac{C_{battery}}{I_{avg}} = \frac{2000 \text{ mAh}}{1.7 \text{ mA}} = 1176 \text{ hours} \quad (5.7)$$

And translating into the number of days:

$$n_{days} = \frac{t_{life}}{24} = \frac{1176 \text{ hours}}{24 \text{ hours}} \approx 49 \text{ days} \quad (5.8)$$

We can see that if the solar panel receives absolutely zero sunlight, then our device will operate from a fully charged battery for approximately 49 days. But in real conditions, the battery is constantly charged by the solar panel, which was confirmed by the real-world exploitation, when the battery level on any of the devices did not fall below 4 V. This fact provides our device with theoretically unlimited operating time.

### ■ 5.1.2 Operating environment

At the time of this writing, the devices have been operating in the open air for two months. End nodes were installed at selected locations in mid-March. This year, both March and April were quite cold months with very variable weather, making this time period ideal for testing the work of the developed devices.

During the outdoor operation of the devices, all possible test scenarios were created: from subzero temperatures to abnormally hot April days. In addition, during this testing period all devices were exposed to both heavy rain, snow, hail and strong winds. The devices (weather stations: *ws1-ws3*) were mounted in different locations:

- in the courtyard of the CTU building on Charles Square (*ws1*).
- on the roof of the building on Národní třída (*ws2*).
- on the open balcony of the house in Nusle district (*ws3*).

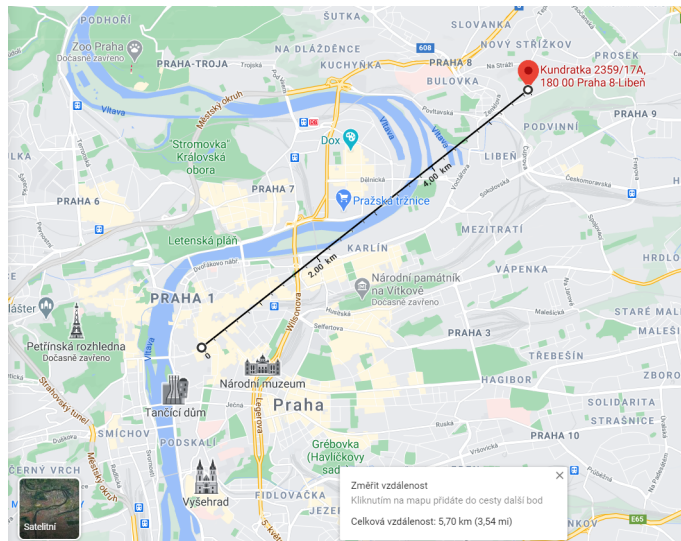


**Figure 5.1:** Deployed sensor nodes

All devices have worked without a single failure since the beginning of the experiment. Despite different locations, the battery voltage level always remained above  $4\text{ V}$  meaning that regardless of weather conditions and cloudiness, the light falling on the solar panel was enough to constantly maintain the battery voltage level.

The transfer of data was carried out via public gateways. The reach of individual gateways is determined by several criteria:

- the mounting height of both the gateway and the device itself.
- the presence of obstacles in the path of the transmitted signal.
- momentary weather conditions and precipitation.



**Figure 5.2:** Maximum data transmission range achieved in urban conditions using 2 dBi antenna

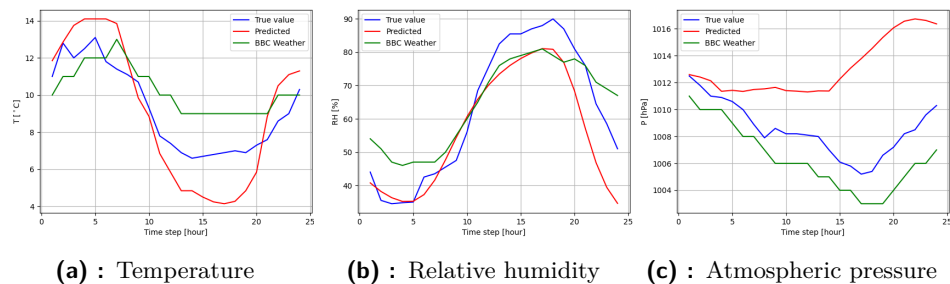
Thanks to the coverage of the *LoRaWAN* network in Prague, each of the devices had at least one gateway in their transmission range. The best results

in urban transmission were detected for a device located on the roof of the building on Národní třída. The most distant gateway to which it was possible to transfer the data was located almost 6 kilometers away from the device location.

### 5.1.3 Weather forecast

After the system was fully deployed, it was necessary to verify the accuracy of the predicted values while operating in real environment. In order to do this, first of all, it was necessary to select the reference and subsequent predicted values from some meteorostation, since the forecasts are generated for each station separately. Also, we needed prediction values representing any commercial product, so that it was possible to determine both the forecast accuracy of our model in relation to the real measured parameters and to widely used service.

The *BBC Weather* service was chosen as a widely available and popular weather forecast service. The government *Met Office*, serving as one of our end points for data sharing, used to be a data provider for this service. Presently, *MeteoGroup* is a new data provider for *BBC Weather*. This organization uses a model operated by *ECMWF (European Centre for Medium-Range Weather Forecasts)* to generate accurate weather forecasts. To predict parameters the method of Numerical Weather Prediction (NWP) is used based on the measurements of temperature, pressure and relative humidity, measurements of wind speed and direction, level of cloud coverage, maximum reflectivity and many more additional parameters. In addition, many post-processing algorithms are used for combining data from many sources and subsequent analysis [81].



**Figure 5.3:** Comparison of forecast model performance with both online service and reference measured values

As the reference station, the station mounted on the roof was selected. The testing algorithm was as follows: at a certain hour, the predicted values of temperature, relative humidity and atmospheric pressure for Prague were recorded on the *BBC Weather* web page [82] for the next 24 hours. At the same time, the forecast values of all three parameters at the selected station for the next 24 hours were recorded. Then, after 24 hours, the measured values of our station were collected from the database to obtain true local

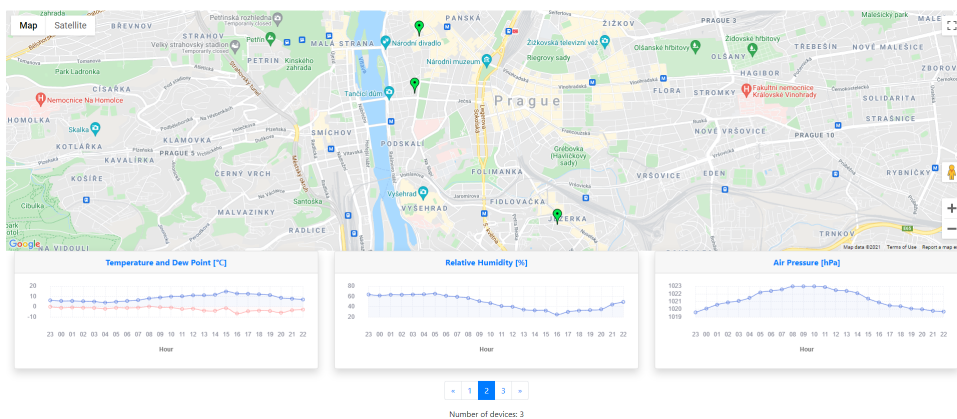


The last point worth mentioning is that our model predicts the development of parameters for a specific location where the station is mounted meaning that measured values are exclusively local, that is, in other locations, devices can measure different parameters, which will affect the subsequent forecast making it, on the one hand, dependent on the measured data in a specific location, without the ability to make a generalized forecast based on many data sources, but on the other hand, we get more accurate local parameters, since services like *BBC Weather* do not always have data available in the vicinity of your specific location, which is why the forecast is not entirely accurate in relation to a specific place.

### 5.1.4 Web page

A user-friendly web page has been developed to display the devices along with their status, latest measured data and forecasts for each meteorostation. A huge advantage of the solution is the ability to instantly display new devices and the scalability of the system.

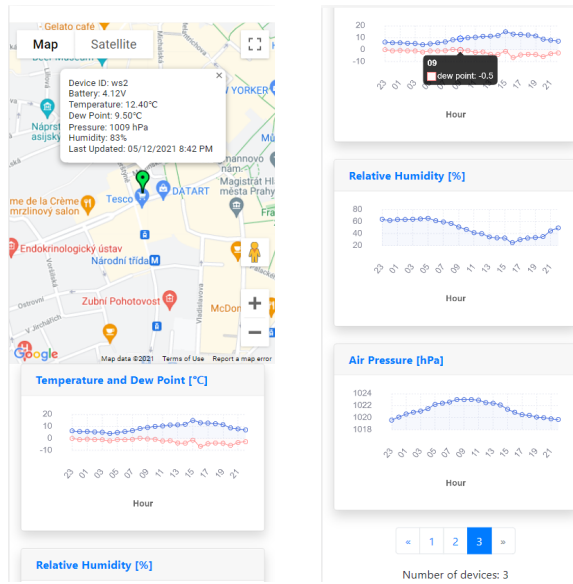
Moreover, the pagination was implemented, with the help of which one can switch between individual devices. When switching, the selected device is automatically zoomed on the map. Also, when one clicks on a device marker, the charts are updated to show the forecast for the selected particular device.



**Figure 5.4:** Main web page appearance on PC or tablet

In addition to displaying in browsers on computers or tablets, the page automatically adjusts the size of the elements when using mobile devices, allowing one to use the page without limiting functionality or degrading the user experience on any device. On the mobile device display, charts are arranged in a column, thereby adjusting to the resolution of a particular device.





(a) : Device info (b) : Generated forecasts

**Figure 5.5:** Main page appearance on mobile device: After clicking on the device marker, the current information will be displayed (a). Generated forecasts for each device and switch between devices using pagination (b).

## 5.2 Comparison with commercial products

Two meteostations were selected as popular commercial products for comparison, each representing a different price category. Both stations differ by a set of sensors, the level of autonomous operation and the complexity of the installation.



**Figure 5.6:** Commercial products selected for comparison. La Crosse Technology C85845 Weather Station (on the left)[83]. Ambient Weather WS-2000 (on the right)[84]

We will also not take into account the fact that our system supports many stations at once, while the stations above work in a "single mode" displaying the measured data for only one station. Therefore, the comparison will be made 1-to-1: comparison of one developed station to the one commercial station.



### 5.2.1 Operation

The first step in making our comparison is to determine the accuracy of the measured values. Based on the studied technical parameters of commercial counterparts, a comparative table was created.

		Our	WS-2000	C85845
Range	Temperature[°C]:	-40 to +85	-40 to +65	-39.9 to +59.9
	Humidity[%]:	0 - 100	10 - 99	1 - 99
	Pressure[hPa]:	300 - 1100	300 - 1100	940 - 1045
Resolution	Temperature[°C]:	0.01	0.1	0.1
	Humidity[%]:	0.008	1	1
	Pressure[Pa]:	0.18	0.34	0.34
Accuracy	Temperature[°C]:	± 1	± 2	± 2
	Humidity[%]:	± 3	± 5	± 3
	Pressure[hPa]:	± 1	± 2.7	± 3

**Table 5.1:** Measured parameters comparison between the assembled device and commercial counterparts

We can see that on the basis of technical characteristics, our device is either has not inferior technical performance, but even outperforms commercial products in most parameters.

### Ambient Weather WS-2000

Given meteostation consists of two blocks. A basic outdoor unit is mounted on a mast and measures wind speed and wind direction, rainfall, temperature, humidity, ultraviolet and solar radiation. Outdoor unit is powered by a super capacitor, which is recharged by a solar panel. In addition, a battery backup is implemented using  $2 \times AAA$  lithium batteries, in case of lack of sunlight. Moreover, the kit includes an additional small indoor unit for measuring indoor barometric pressure, humidity and temperature. The unit is powered by  $2 \times AA$  batteries.

Meteostation's kit also includes a console - a wireless LCD display that displays measured values and generated forecast. The console is powered by conventional batteries that need to be replaced with time. There are also additional indicators of sunrise/sunset, moon phase and in-build alarm. In addition, there is the possibility of integration with Google Home. The weather forecast is generated based on the rate of change of pressure: if the given rate increases, the weather is improving (sunny), otherwise the weather is degrading (cloudy or rainy). If the rate of change is relatively steady, it will show partly cloudy or partly sunny weather.



signal via which the console transmits data to the server, it will be impossible to get the last measured parameters through this application.

The next disadvantage is the forecast of the local meteorological situation using only one parameter - atmospheric pressure. According to information from the manufacturer, generated prediction is only 70% accurate on average. Manufacturer also clarifies that it may take up to 30 days to generate a more or less accurate forecast. Furthermore, the forecast is very general - "sunny"/"rainy" and there is no hourly forecast of the measured parameters. In contrast to the above, our device generates an hourly forecast that is close to real values and is able to compete with the forecast provided by a commercial service. Also, the "measurements accumulation period" for the first forecast generation is only 24 hours, after which the forecast will be continuously generated every hour.

Finally, the indoor unit measuring temperature, humidity and pressure is powered by regular 2-AA batteries, which need to be replaced over time.

### ■ C85845

This *La Crosse Technology C85845 Weather Station* consists of one single outdoor unit capable of measuring three parameters - temperature, humidity and atmospheric pressure. This module is powered by  $2 \times AA$  batteries.

Meteostation's kit also includes a console. It is powered by  $3 \times AA$  batteries that can be recharged through a 5 V DC adapter. The sensors send data to this console (every 30 seconds) via the RF (915 MHz) link, where the data is then displayed on colored LCD display. The maximum range of signal transmission from the sensors to the console is the same as that of the previously reviewed commercial model: 100 meters in the line of sight. The forecast of the meteorological situation, as in the previous case, is generated based on the rate of change of pressure.

**Similar.** Given meteostation measures same weather parameters: pressure, humidity and temperature. Also, there is a sensor unit's battery level monitoring. The measured data is displayed on the console, while ours data is displayed on the web page.

**Pros.** Among the advantages is the presence of both a regular clock and an alarm clock. Otherwise, in my opinion, this station has no more advantages over the station developed within the framework of this work.

**Cons.** Among the same disadvantage, as in the previous commercial model, is the attachment of the console to the installed sensor, since the maximum distance of data transmission from the sensor to the console is only 100 meters. Also, the data cannot be viewed if you are not near the console, which is a huge disadvantage. In addition, there is neither dew point calculation nor sharing the measured data with other community weather monitoring projects. The sensor module itself must be installed using screws, which also makes this product not universal in all locations.

Another disadvantage relates to the forecast. Due to the prediction method based only on pressure values, the accuracy is 70% - 75%. The forecast is generated for the next 6-12 hours. Also, according to the information from the manufacturer, it takes about 7-10 days to get a more or less accurate forecast.

The last point remaining to highlight is the autonomy of a sensor unit, which is limited by the battery lifetime, since the device does not have an integrated solar panel. According to the official information from the manufacturer, outdoor unit's battery lifetime is approximately 2 months.

### 5.2.2 Price

To compare the price of one developed device with described commercial offers, we first need to calculate the total price of the entire developed device, including all physical components and the cost of hosting the software on Amazon server.

The calculation of the cost of individual components was made in two directions:

- *Distributor* means the price of the module when purchased in a store in the Czech Republic.
- *E-shop* column contains the price of the module when buying online on large retail services with further delivery to the Czech Republic. The disadvantage of this option is the longer waiting time for the order, which can take about a month. But, in case of placing an order in advance, this point is leveled. Also, large overseas online shops often pay for delivery themselves. *AliExpress* was chosen as the benchmark for the E-shop column.

Module	Distributor	E – shop
<i>ESP32 LoRa</i>	350	220
<i>BME280</i>	180	22
<i>NEO-6M GPS</i>	169	64
<i>Li-Po battery</i>	230	21
<i>Solar panel</i>	125	25
<i>TP4056 miniUSB</i>	14	6
<i>TPL5111 timer</i>	190	150
<i>Magnets</i>	50	30
<i>Other</i>	80	80
<b>Total [CZK] :</b>	<b>1388</b>	<b>638</b>

**Table 5.2:** Calculation of the final price of one meteorostation unit

The *Other* parameter in the table represents the price of assembling the case, as well as screws, acrylic protective cover, resistors/capacitors and wires. The main objects of expenditure here are:

- *Case.* The current market value of 3D printing is 4.5 CZK per gram. The printed case weighed 130 grams meaning that the price of printing would be about 600 CZK. But in case of owning a 3D printer and printing the case by ourselves the cost will drop significantly. The market value of 1 kilogram of *PETG* filament is 400 CZK. Now we can calculate that printing the case by ourselves will cost about 50 CZK, since the costs go only for the material.
- *Acrylic protective cover.* The market value of one square meter of acrylic glass is around 700 CZK. For this money we get  $1,000,000 \text{ mm}^2$  of acrylic glass. The parameters of the cover used in the project are  $83 \text{ mm}$  (width) and  $150 \text{ mm}$  (length) meaning that the area we need is equal to  $12,450 \text{ mm}^2$ . Also, we must take into account the presence of a double acrylic baffle that protects the inside of the module from moisture in case of flooding of the *BME280* sensor. Baffle parameters are:  $25 \text{ mm}$  (width) and  $25 \text{ mm}$  (length) and getting the area of the baffle equal to  $625 \text{ mm}^2$ . Since it is double (i.e. consisting of two layers), we multiply it by 2 and get  $1,250 \text{ mm}^2$ . So, the total area of the required acrylic glass for one device is:  $12,450 \text{ mm}^2 + 1,250 \text{ mm}^2 = 13,700 \text{ mm}^2$ . Based on all calculations, we can state that one square meter of acrylic glass is enough for us to assemble about 70 devices. At the cost we get the price for one device: 10 CZK.

As for the cost of using infrastructure and services:

- *The Things Network* - free of charge.
- *AWS* - the cost of use depends on specific loads, but since we use the *t2.micro* instance and the number of devices within this solution will not exceed dozens of units, the price will be no more than a couple of dollars per month.
- *Google Maps API* - implemented as pay-as-you-go pricing. We use the *Maps JavaScript API*, which costs: 0.15 CZK for each map load. Assuming that maps are loaded 50 times per day by all users, the price for using the API per day is 7.5 CZK, or 225 CZK per month for 11,250 requests per month. But there is one big advantage - every month Google provides each user with a balance of 200\$ (4200 CZK) for using the *Google Maps API* meaning that the payment starts after the use of this balance is exceeded. In other words, if we recalculate this balance by the number of requests, we will receive almost 210,000 free requests per month, which will definitely be enough for us as the network expands over time.

	WS – 2000	C85845	Our
Price [CZK] :	6200	1280	1388/638

**Table 5.3:** Comparison of the final price of the assembled device with commercial products

There are two prices for our device: the first is when using items purchased in shops in the Czech Republic and the second is when utilizing items purchased online.

Based on the pricing table, we can see that the price of our device is in the price range of devices like *La Crosse Technology C85845*, or even cheaper if using components purchased online. At the same time, the developed device outperforms the *C85845* weather station in all parameters.

In addition, it can be argued that based on the operating information and the price/range ratio of functions provided, our developed device can compete with devices of a premium segment, since in addition to the lower price and the already described advantages, the developed device is designed in such a way that its use is easy in urban environment and, in addition, it is possible to add more devices to the same network.

## Chapter 6

### Conclusion

Within the framework of this work, the possibilities of implementing a distributed network of personal meteorological stations in a city were studied. In addition to the hardware component, work was carried out on the analysis and solution of the problem of centralized collection of measured data, storing data on the cloud with its subsequent processing, as well as the implementation of a user-friendly interface to display the results.

In Chapter [1], an analysis of the future implementation was made. Important criteria and existing limitations in the development of the hardware unit were highlighted. The selection of measured meteorological parameters was also justified. In addition, relevant implementation examples of systems for the centralized data collection were described.

The Chapter [2] describes the basic concepts that served as the basis for the implementation of the present work. At the beginning, the architecture of both *WSN* networks as a whole and end devices in particular was illustrated. Next, the detailed description of the data transfer method using the *LoRa* protocol along with a description of the whole *LoRaWAN* architecture was presented. After that, the principles of neural networks were described, starting from the characteristics of the operation of a simple neuron, and ending with a description of the architecture of *LSTM* networks for time series predictions. The *AWS* architecture and the individual products used were also specified. Finally, we explained the methodology for calculating the additional meteorological parameter to improve prediction accuracy.

Chapter [3] presents the implementation of the whole system that can be roughly divided into three parts:

1. Meteostation's development, which includes both the selection of individual components, the design of a protective case and implementation of the controller program.
2. Development of a machine learning algorithm for predicting future meteorological parameters based on the measured values.
3. Infrastructure development and deploying on cloud hosting. This subpart includes both the configuration of individual components and the development of the backend and frontend.

Chapter [4] describes the process of sharing measured parameters of all meteostations with community weather monitoring projects.

Finally, Chapter [5] summarizes the overall project, describes the results and compares developed meteostation with commercially available products.

As a result of the present work is a fully operating network of meteorological stations with centralized data collection and processing. Results are displayed in a convenient and user-friendly form. The results are displayed in a convenient and user-friendly way in the form of a web page. It can be argued that present solution is able to compete with commercial products.

But the work doesn't stop there. As a future improvement of the system, the emphasis will be put on the processing of open-source images from geostationary weather satellites for detecting impending cyclones and combining these results with local measurements to develop a more accurate model for predicting local climatic conditions.





## Appendix A

### List of Acronyms

API	Application Programming Interface
AWS	Amazon Web Services
Adam	Adaptive Moment Estimation
BPTT	Backpropagation Through Time
CPU	Central Processing Unit
CSS	Chirp Spread Spectrum
DSSS	Direct-Sequence Spread Spectrum
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IoT	Internet of Things
LPWAN	Low-power Wide Area Network
LSTM	Long Short Term Memory
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
MAE	Mean Absolute Error
MSE	Mean Squared Error
PETG	Polyethylene Terephthalate Glycol Modified
PSK	Phase Shift Keying
PWM	Pulse Width Modulation
QPSK	Quadrature Phase Shift Keying
RF	Radio Frequency
RMSE	Root Mean Square Error
RNN	Recurrent neural network
RSSI	Received Signal Strength Indicator
SNR	Signal-To-Noise Ratio
SPI	Serial Peripheral Interface
TTN	The Things Network
ULP	Ultra-Low-Power
USB	Universal Serial Bus
WSN	Wireless Sensor Network



## Appendix B

### Example of TTN uplink message

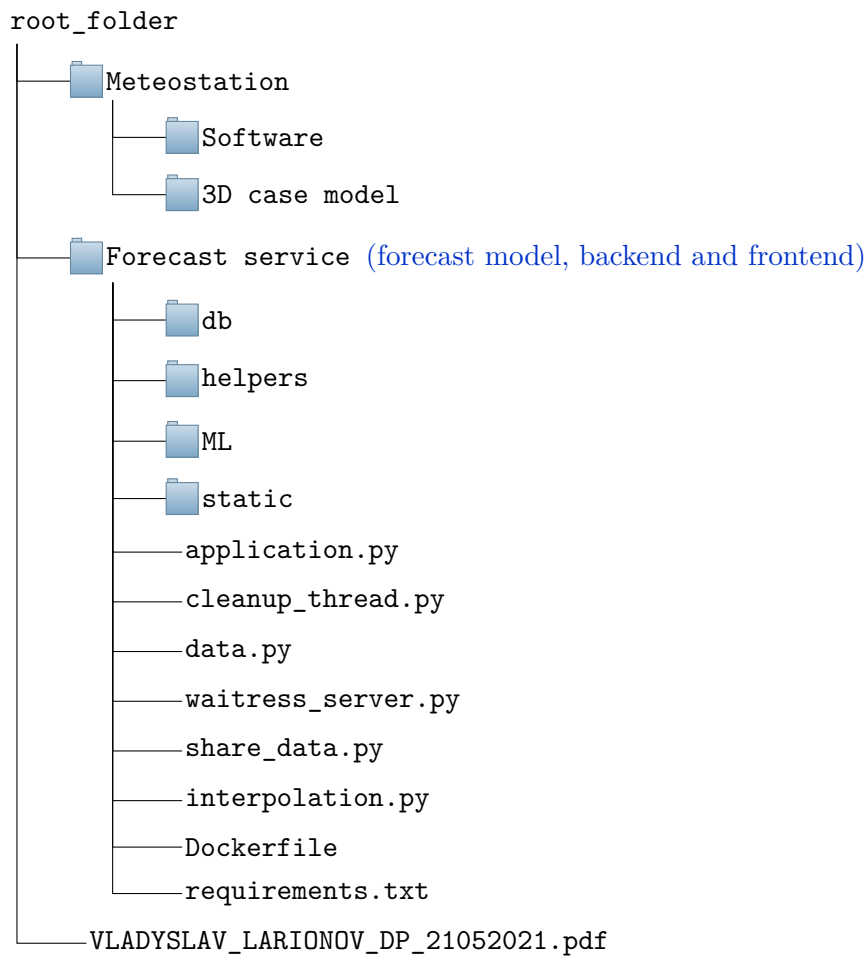
```
1 {
2   "app_id": "weather_station_network",
3   "dev_id": "ws2",
4   "hardware_serial": "AABBCCAABBCCAAB1",
5   "port": 1,
6   "counter": 4142,
7   "payload_raw": "AWcAqAJoUQNzJnYEiAekWgIzQgAAyAUCAZw=",
8   "payload_fields": {
9     "analog_in_5": 4.12,
10    "barometric_pressure_3": 984.6,
11    "gps_4": {
12      "altitude": 2,
13      "latitude": 50.0826,
14      "longitude": 14.4194
15    },
16    "relative_humidity_2": 40.5,
17    "temperature_1": 16.8
18  },
19  "metadata": {
20    "time": "2021-05-09T22:09:04.898296107Z",
21    "frequency": 868.1,
22    "modulation": "LORA",
23    "data_rate": "SF7BW125",
24    "coding_rate": "4/5",
25    "gateways": [
26      {
27        "gtw_id": "satoshilabs02",
28        "gtw_trusted": true,
29        "timestamp": 3442332483,
30        "time": "",
31        "channel": 0,
32        "rssi": -120,
33        "snr": -8.25,
34        "rf_chain": 1,
35        "latitude": 50.114483,
```

B. Example of TTN uplink message

```
36     "longitude": 14.481862
37   },
38   {
39     "gtw_id": "eui-0000024b080e0539",
40     "timestamp": 2979816891,
41     "time": "2021-05-09T22:09:03.972719Z",
42     "channel": 0,
43     "rssi": -99,
44     "snr": 2.5,
45     "rf_chain": 0,
46     "latitude": 50.08947,
47     "longitude": 14.42204,
48     "altitude": 221
49   },
50   {
51     "gtw_id": "cesgw4",
52     "timestamp": 4211674867,
53     "time": "2021-05-09T22:09:04.877282Z",
54     "channel": 0,
55     "rssi": -99,
56     "snr": 3.5,
57     "rf_chain": 0,
58     "latitude": 50.08189,
59     "longitude": 14.425832,
60     "altitude": 229
61   }
62 ]
63 },
64 "downlink_url": "https://integrations.thethingsnetwork.org/
ttn-eu/api/v2/down/weather_station_network/aws_integration
?key=ttn-account-v2.tlkkJ4Le5tf0YUIO4cFOX0oOF0z2
zjRITAwffQDkNFY"
65 }
```

# Appendix C

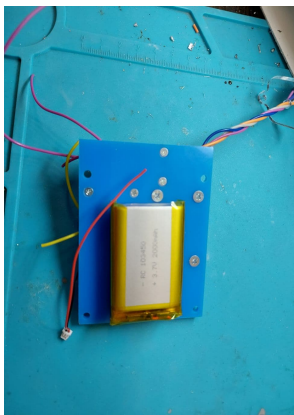
## CD Content



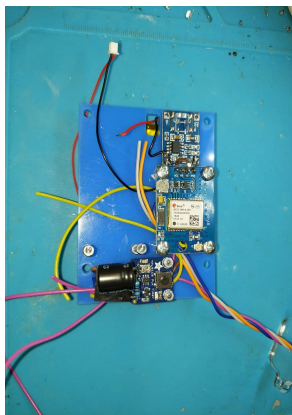


## Appendix D

### Photos



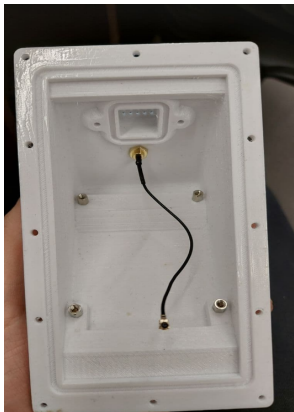
(a)



(b)



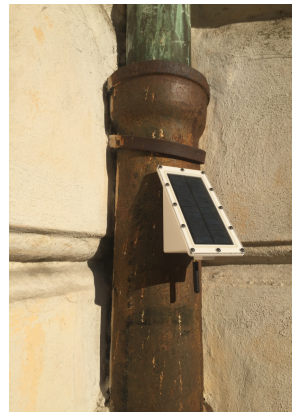
(c)



(d)



(e)



(f)

**Figure D.1: Device assembly process and nodes deployment.** The assembly and screwing process of individual components onto a common mount is depicted (a-c). Printed case without embedded components (d). Deployed devices on the roof (e) and in the courtyard (f).





## Appendix E

### Bibliography

- [1] Schultz, Martin & Betancourt, Clara & Gong, Bing & Kleinert, Felix & Langguth, Michael & Leufen, Lukas & Mozaffari, Amirpasha & Stadtler, Scarlet. (2021). *Can deep learning beat numerical weather prediction?*. Philosophical Transactions of The Royal Society A Mathematical Physical and Engineering Sciences. 379. 10.1098/rsta.2020.0097.
- [2] Farhan, Laith & Shukur, Sinan & Alissa, Ali E. & Alrweg, Mohmad & Raza, Umar & Kharel, Rupak. (2017). *A survey on the challenges and opportunities of the Internet of Things (IoT)*. 1-5. 10.1109/IC-SensT.2017.8304465.
- [3] Bagiorgas, Haralambos & Assimakopoulos, Margarita & Patentalaki, A & Konofaos, Nikos & Matthopoulos, Demetrios & Mihalakakou, G.. (2007). *The design, installation and operation of a fully computerized, automatic weather station for high quality meteorological measurements*. Fresenius Environmental Bulletin. 16. 948-962.
- [4] Tim R. Oke *Initial Guidance To Obtain Representative Meteorological Observations At Urban Sites*. Instruments and observing methods report No. 81, 2006. <https://www.weather.gov/media/epz/mesonet/CWOP-WM01250.pdf> [online]
- [5] Ed Oswald. *Weather Station Mounting Ideas and Solutions With Siting Instructions*. <https://www.weatherstationadvisor.com/weather-station-mounting-ideas-and-solutions/> [online]
- [6] The Things Network mapper. <https://ttnmapper.org/> [online]
- [7] Citizen Weather Observer Program (CWOP). *Weather Station Siting, Performance, and Data Quality Guide*, 2005. <https://www.weather.gov/media/epz/mesonet/CWOP-OfficialGuide.pdf> [online]
- [8] Wind measurement. [https://cumuluswiki.org/a/Wind\\_measurement](https://cumuluswiki.org/a/Wind_measurement) [online]
- [9] B. C. Csáji, Z. Kemény, G. Pedone, A. Kuti and J. Váncza, *Wireless Multi-Sensor Networks for Smart Cities: A Prototype System With Statistical*

- Data Analysis*, in IEEE Sensors Journal, vol. 17, no. 23, pp. 7667-7676, 1 Dec.1, 2017, doi: 10.1109/JSEN.2017.2736785.
- [10] Polonelli, Tommaso & Brunelli, Davide & Bartolini, Andrea & Benini, Luca. (2019). *A LoRaWAN Wireless Sensor Network for Data Center Temperature Monitoring*. 10.1007/978-3-030-11973-7\_20.
- [11] Gresl, Jonathan & Fazackerley, Scott & Lawrence, Ramon. (2021). *Practical Precision Agriculture with LoRa based Wireless Sensor Networks*. 131-140. 10.5220/0010394401310140.
- [12] Mainwaring, Alan & Polastre, Joseph & Szewczyk, Robert & Culler, David & Anderson, John. (2002). *Wireless Sensor Networks for Habitat Monitoring*. Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications. 10.1145/570738.570751.
- [13] G. Werner-Allen et al., *Deploying a wireless sensor network on an active volcano* in IEEE Internet Computing, vol. 10, no. 2, pp. 18-25, March-April 2006, doi: 10.1109/MIC.2006.26.
- [14] García, Sebastián & Larios, Diego & Barbancho, Julio & Personal, Enrique & Mora-Merchan, Javier & León, Carlos. (2019). *Heterogeneous LoRa-Based Wireless Multimedia Sensor Network Multiprocessor Platform for Environmental Monitoring*. Sensors. 19. 3446. 10.3390/s19163446.
- [15] McGrath M.J., Scanail C.N. *Sensor Network Topologies and Design Considerations*. In: Sensor Technologies. Apress, Berkeley, CA. 2013 [https://doi.org/10.1007/978-1-4302-6014-1\\_4](https://doi.org/10.1007/978-1-4302-6014-1_4) [online]
- [16] Huang, Gongsheng & Zhou, Pei & Zhang, Linfeng. (2013). *Wireless sensor network for HVAC applications: a review*.
- [17] Ismail, Dali & Rahman, Mahbubur & Saifullah, Abusayeed. (2018). *Low-power wide-area networks: opportunities, challenges, and directions*. 1-6. 10.1145/3170521.3170529.
- [18] LoRa® and LoRaWAN® White Paper. <https://lora-developers.semtech.com/library/tech-papers-and-guides/lora-and-lorawan/> [online]
- [19] Orthogonal variable spreading factor. [https://en.wikipedia.org/wiki/Chip\\_\(CDMA\)](https://en.wikipedia.org/wiki/Chip_(CDMA)) [online]
- [20] Kim, Dong-Hoon & Lee, Eun-Kyu & Kim, Jibum. (2019). *Experiencing LoRa Network Establishment on a Smart Energy Campus Testbed*. Sustainability. 11. 1917. 10.3390/su11071917.
- [21] LoRa™ Modulation Basics *Semtech Corporation*, Application Note 2015. <https://www.frugalprototype.com/wp-content/uploads/2016/08/an1200.22.pdf> [online]

- [22] Augustin, Aloÿs & Yi, Jiazi & Clausen, Thomas Heide & Townsley, William. (2016). *A Study of LoRa: Long Range & Low Power Networks for the Internet of Things*. Sensors. 16. 1466. 10.3390/s16091466.
- [23] LoRa ISM Bands. <http://pdacontrolen.com/introduction-lora-module-rfm95-hoperf/> [online]
- [24] LoRa Frame Format. <https://commons.wikimedia.org/wiki/File:LoRaFrameFormat.png> [online]
- [25] Afisiadis, Orion ; Burg, Andreas ; Balatsoukas-Stimming, Alexios. *Coded LoRa Frame Error Rate Analysis*. 2020 IEEE International Conference on Communications, ICC 2020 - Proceedings. Institute of Electrical and Electronics Engineers, 2020.
- [26] Seneviratne P. (2019) *Introduction to LoRa and LoRaWAN*. In: *Beginning LoRa Radio Networks with Arduino*. Apress, Berkeley, CA. [https://doi.org/10.1007/978-1-4842-4357-2\\_1](https://doi.org/10.1007/978-1-4842-4357-2_1)
- [27] Overtraining. <https://wiki.loginom.ru/articles/overtraining.html> [online]
- [28] Kingma, Diederik & Ba, Jimmy. (2014). *Adam: A Method for Stochastic Optimization*. International Conference on Learning Representations.
- [29] The Things Network. <https://www.thethingsnetwork.org/> [online]
- [30] Grossi, Enzo & Buscema, Massimo. (2008). *Introduction to artificial neural networks*. European journal of gastroenterology & hepatology. 19. 1046-54. 10.1097/MEG.0b013e3282f198a0.
- [31] Suzuki, Kenji. (2013). *ARTIFICIAL NEURAL NETWORKS – ARCHITECTURES AND APPLICATIONS*.
- [32] Brain Neuron Structure. <https://www.smartsheet.com/neural-network-applications> [online]
- [33] Schmidhuber, Juergen. (2014). *Deep Learning in Neural Networks: An Overview*. Neural Networks. 61. 10.1016/j.neunet.2014.09.003.
- [34] Guo, Yanan & Cao, Xiaoqun & Bainian, Liu & Peng, Kecheng. (2020). *El Niño Index Prediction Using Deep Learning with Ensemble Empirical Mode Decomposition*. Symmetry. 12. 893. 10.3390/sym12060893.
- [35] Aurlien Gron. 2017. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (1st. ed.). O'Reilly Media, Inc. pages: 379-410.
- [36] Sherstinsky, Alex. (2020). *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network*. Physica D: Non-linear Phenomena. 404. 132306. 10.1016/j.physd.2019.132306.

- [37] Pascanu, Razvan & Gulcehre, Caglar & Cho, Kyunghyun & Bengio, Y.. (2013). How to Construct Deep Recurrent Neural Networks.
- [38] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). *Long Short-term Memory*. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.
- [39] Qing, Xiangyun & Niu, Yugang. (2018). *Hourly day-ahead solar irradiance prediction using weather forecasts by LSTM*. Energy. 148. 10.1016/j.energy.2018.01.177.
- [40] Park, & Kim, Hyun Soo & Lee, & Song,. (2019). *Temperature Prediction Using the Missing Data Refinement Model Based on a Long Short-Term Memory Neural Network*. Atmosphere. 10. 718. 10.3390/atmos10110718.
- [41] Varsamopoulos, Savvas & Bertels, Koen & Almudever, Carmen. (2018). *Designing neural network based decoders for surface codes*.
- [42] Fei-Fei Li & Justin Johnson & Serena Yeung. *CS231n: Convolutional Neural Networks for Visual Recognition*, 2017. [http://cs231n.stanford.edu/slides/2017/cs231n\\_2017\\_lecture10.pdf](http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture10.pdf) [online]
- [43] AWS Regions and Availability Zones. [https://aws.amazon.com/about-aws/global-infrastructure/regions\\_az/](https://aws.amazon.com/about-aws/global-infrastructure/regions_az/) [online]
- [44] AWS Global Infrastructure Map. <https://aws.amazon.com/about-aws/global-infrastructure/> [online]
- [45] Somckit Khemmanivanh. *Deploying highly available SAP systems using SIOS Protection Suite on AWS*, 2019. <https://aws.amazon.com/blogs/awfsorsap/deploying-highly-available-sap-systems-using-sios-protection-suite-on-aws/> [online]
- [46] AWS Regional Services. <https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/>
- [47] Bony, Sandrine & Stevens, Bjorn & Frierson, Dargan & Jakob, Christian & Kageyama, Masa & Pincus, Robert & Shepherd, Ted & Sherwood, Steven & Siebesma, A.P. & Sobel, Adam & Watanabe, Masahiro & Webb, Mark. (2015). *Clouds, circulation and climate sensitivity*. Nature Geoscience. 8. 261-268. 10.1038/ngeo2398.
- [48] Stefan Becker. *Calculation Of Direct Solar And Difuse Radiation In Israel* International Journal Of Climatology Int. J. Climatol. 21: 1561–1576 (2001).
- [49] H.D. Kambezidis. *Comprehensive Renewable Energy 3.02 - The Solar Resource*. Volume 3, 2012, Pages 27-84
- [50] Padilla, Ricardo. (2011). *Simplified Methodology for Designing Parabolic Trough Solar Power Plants*.

- [51] Tomáš Matuška. Lecture materials for subject *Fundamentals of Alternative Energy Sources*, 2017. [http://users.fs.cvut.cz/tomas.matuska/wordpress/wp-content/uploads/2015/02/AES-L1-solar\\_energy\\_2017.pdf](http://users.fs.cvut.cz/tomas.matuska/wordpress/wp-content/uploads/2015/02/AES-L1-solar_energy_2017.pdf) [online]
- [52] Makhotkin L.G. *Bemporad's air-mass equivalent*. Proceedings of the Main Geophysical Observatory of A.I. Voeikov №100. Investigation of radiation processes, page 15–16, 1960.
- [53] Pages of The AERONET (AErosol RObotic NETwork) project. <https://aeronet.gsfc.nasa.gov/> [online]
- [54] J.W. Spencer. *Fourier series representation of the position of the sun*. Search, 2 (1971), p. 172
- [55] G. Notton. *Solar Radiation for Energy Applications*. Encyclopedia of Sustainable Technologies, Elsevier, 2017, Pages 339-356.
- [56] ESP32 Series Datasheet. [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf) [online]
- [57] SX1276/77/78/79 Datasheet, 2015. [https://cdn-shop.adafruit.com/product-files/3179/sx1276\\_77\\_78\\_79.pdf](https://cdn-shop.adafruit.com/product-files/3179/sx1276_77_78_79.pdf) [online]
- [58] BME280 Sensor Datasheet. <https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf> [online]
- [59] u-blox 6 GPS Modules Datasheet. [https://www.u-blox.com/sites/default/files/products/documents/NEO-6\\_DataSheet\\_\(GPS.G6-HW-09005\).pdf](https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_(GPS.G6-HW-09005).pdf) [online]
- [60] Li-Polymer Battery Technology Specification. [https://www.mouser.com/catalog/additional/Particle\\_Battery\\_LP103450\\_2000mAh\\_3.7V\\_20151103.pdf](https://www.mouser.com/catalog/additional/Particle_Battery_LP103450_2000mAh_3.7V_20151103.pdf) [online]
- [61] Lithium Polymer Battery Complete Guide. <https://www.dnkpowers.com/lithium-polymer-battery-guide/> [online]
- [62] Battery Protection Board for Li-Po Battery. [https://www.banggood.com/1S-3\\_7V-2A-li-ion-BMS-PCM-18650-Battery-Protection-Board-PCB-for-18650-Lithium-ion-li-Battery-p-1538099.html](https://www.banggood.com/1S-3_7V-2A-li-ion-BMS-PCM-18650-Battery-Protection-Board-PCB-for-18650-Lithium-ion-li-Battery-p-1538099.html) [online]
- [63] TP4056 Datasheet. <http://www.tp4056.com/d/tp4056.pdf> [online]
- [64] TPL5111 Nano-Power System Timer for Power Gating Datasheet. <https://www.ti.com/product/TPL5111> [online]
- [65] Siraki, Arbi & Pillay, P. (2012). *Study of optimum tilt angles for solar panels in different latitudes for urban applications*. Solar Energy. 86. 1920–1928. 10.1016/j.solener.2012.02.030.

- [66] Kováčová, Mária & Kozakovičová, Jana & Prochazka, Michal & Janigová, Ivica & Vysopal, Marek & Černičková, Ivona & Krajčovič, Jozef & Spital'sky, Zdenko. (2020). *Novel Hybrid PETG Composites for 3D Printing*. Applied Sciences. 10. 3062. 10.3390/app10093062.
- [67] Tractus3D.com *Filaments for Outdoor Use*. <https://tractus3d.com/knowledge/learn-3d-printing/filaments-for-outdoor-use/> [online]
- [68] Yusuf Vihaan on 3DRIFIC. *5 best 3D printer filaments for outdoor use*, 2021. <https://3drific.com/best-3d-printer-filaments-for-outdoor-use/> [online]
- [69] Which is more durable to sunlight/weather - PLA, ABS or PETG. *3D printing forum on StackOverflow* <https://3dprinting.stackexchange.com/questions/3853/which-is-more-durable-to-sunlight-weather-pla-abs-or-petg/3857> [online]
- [70] The Things Network integrations. <https://www.thethingsnetwork.org/docs/applications/integrations/index.html> [online]
- [71] The Things Network Prague community. <https://www.thethingsnetwork.org/community/prague/> [online]
- [72] The Things Network Berlin community. <https://www.thethingsnetwork.org/community/berlin/> [online]
- [73] The Things Network's public community network Fair Access Policy. <https://www.thethingsnetwork.org/docs/lorawan/duty-cycle/index.html> [online]
- [74] The Things Network Limitations. <https://www.thethingsnetwork.org/docs/lorawan/limitations/index.html> [online]
- [75] CloudFlare webpage. *What Is A Reverse Proxy? | Proxy Servers Explained* [online]
- [76] Calculate the air time of your LoRa frame. <https://www.loratools.nl/#/airtime> [online]
- [77] Dew Point. [https://en.wikipedia.org/wiki/Dew\\_point](https://en.wikipedia.org/wiki/Dew_point) [online]
- [78] Atmospheric pressure. [https://en.wikipedia.org/wiki/Atmospheric\\_pressure](https://en.wikipedia.org/wiki/Atmospheric_pressure) [online]
- [79] Weather Observations Website. <https://www.metoffice.gov.uk/> [online]
- [80] Open Weather Map. <https://openweathermap.org/> [online]
- [81] European Centre for Medium-Range Weather Forecasts. <https://www.ecmwf.int/> [online]

- [82] BBC Weather Forecast for Prague. <https://www.bbc.com/weather/> [online]
- [83] La Crosse Technology's C85845 Wireless Color Weather Station. <https://www.lacrossetechnology.com/products/c85845> [online]
- [84] Ambient Weather WS-2000 Smart Weather Station with WiFi Remote Monitoring and Alerts & Thermo Hygrometer. <https://ambientweather.com/amws2000.html> [online]