

# Machine Learning-based Environmental Monitoring and Analysis System<sup>\*</sup>

Bohdan Zhurakovskiy<sup>1,\*†</sup>, Oleksandr Pliushch<sup>2,†</sup>, Volodymyr Saiko<sup>3,†</sup> and Viktor Shutenko<sup>1,†</sup>

<sup>1</sup> National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," 37 Peremogy ave., 03056 Kyiv, Ukraine

<sup>2</sup> Taras Shevchenko National University of Kyiv, 60 Volodymyrska str., 01601 Kyiv, Ukraine

<sup>3</sup> Kruty Heroes Military Institute of Telecommunications and Information Technology, 45/1 Knyaz Ostrozki str., 01011 Kyiv, Ukraine

## Abstract

Growing environmental threats require modern solutions for environmental monitoring. Integration of Internet of Things technologies with machine learning allows you to automate the process of collecting, processing, and analyzing environmental data, ensuring accuracy and speed. The work aims to create a system for automated environmental monitoring capable of quickly analyzing environmental data. For this purpose, modern approaches were studied, the system architecture was developed, machine learning algorithms were implemented and testing was conducted. The main emphasis is on the use of IoT technologies for automated data collection, machine learning algorithms for predicting changes in the environmental state, and local data processing using TinyML to reduce the load on cloud services. The developed system can be used in agriculture, urban structures, and industry, optimizing the costs of environmental monitoring and improving data quality.

## Keywords

Internet of Things, analysis, monitoring, environment, database, machine learning, model, ecology

## 1. Introduction

The topic of research is relevant and important not only within the framework of environmental issues. It covers a wider range of industries, including business, agriculture, socio-economic development, smart cities, etc. [1]. Effective management and forecasting of the state of the environment is a significant factor of influence in the conditions of constant development of urbanization, the introduction of automation of business processes from various industries, and the emergence of technological innovations. Enterprises, farms, and urban agglomerations need to implement modern tools to guarantee sustainable development, preserve ecosystems, and ensure the greatest efficiency of production. The combination of IoT technologies and machine learning can provide an alternative approach to the development of systems for environmental risk management and real-time environmental monitoring [2].

Thus, the relevance of the research topic lies not only in the environmental component but also in its significance for modern business and urban agglomeration management. Investments in such systems allow to increase the efficiency of resource management, reduce the environmental impact on the environment, and also create new opportunities for business through process optimization, cost reduction, and productivity increase.

The practical significance of this research lies in the creation of a universal and adaptive system for environmental monitoring, which can be used in various fields of activity. This system can be used in public administration to track the state of the environment and respond to environmental threats, in industrial enterprises to monitor compliance with environmental norms and standards,

<sup>\*</sup> CPITS 2025: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, February 28, 2025, Kyiv, Ukraine

<sup>\*</sup> Corresponding author.

<sup>†</sup> These authors contributed equally.

✉ zhurakovskiyb@tk.kpi.ua (B. Zhurakovskiy); oplushch@yahoo.com (O. Pliushch); vgsaiko@gmail.com (V. Saiko); victor.shutenko@gmail.com (V. Shutenko)

ORCID 0000-0003-3990-5205 (B. Zhurakovskiy); 0000-0001-5310-0660 (O. Pliushch); 0000-0002-3059-6787 (V. Saiko); 0009-0002-3537-4101 (V. Shutenko)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

as well as in scientific research to model environmental processes and study their interrelationships. The proposed system can also be used to assess the impact on the environment during the construction of new infrastructure facilities, urban development planning, and development of programs for adaptation to climate change. In addition, it can serve as a platform for conducting environmental education programs, helping to raise public awareness of the importance of preserving natural resources and being environmentally responsible. In the long term, such a system can contribute to the sustainable development of society by preventing ecosystem degradation, reducing the impact of anthropogenic factors on nature, and ensuring the preservation of natural resources for future generations.

## **2. Description of the subject area**

### **2.1. General analysis of the research object**

Environmental monitoring systems are becoming increasingly important in the context of global challenges such as climate change, depletion of natural resources, and environmental pollution. The integration of modern technologies such as the Internet of Things (IoT) and machine learning (ML) [3] allows for the creation of effective solutions for the analysis and prediction of environmental changes. An environmental monitoring system based on IoT and machine learning is just such a solution that combines modern approaches to data collection, processing, and analysis.

The IoT system consists of:

- Sensors—devices that collect data on temperature, humidity, gas concentrations (CO<sub>2</sub>, CO, NO<sub>x</sub>), water level, soil pollution, and other factors.
- Network protocols—means for wireless data transmission, such as LoRa, Zigbee, and Wi-Fi, which provide remote transmission of information from sensors to servers.
- Management platforms—software for data collection and processing, which can be located both in the cloud and on local servers.

Machine learning (ML) is an important component of modern data analysis systems. Its use allows you to automatically find patterns and regularities in large amounts of information, provide forecasts, and help make decisions [4].

An environmental analysis system based on IoT and machine learning is a promising solution that combines modern technologies to ensure effective monitoring of environmental conditions. It not only allows you to respond to current changes in the environment but also to predict possible problems, ensuring sustainable development for various industries and sectors of the economy.

### **2.2. Analysis and comparison of existing systems**

Existing environmental monitoring systems based on IoT and machine learning technologies are diverse, and each of them has its unique characteristics. Let us conduct a comparative analysis of the main systems that have already been implemented in different regions and sectors [5].

Table 1 shows how the analyzed existing systems and technologies differ in key indicators (scalability, forecast accuracy, energy efficiency, infrastructure requirements, cost, flexibility of configuration).

**Table 1**  
Comparison of existing systems

System name	Scalability	Accuracy of forecasts	Energy Efficiency	Flexibility of setting
Smart Santander	High for one city, limited globally	Average	High	Limited to other cities
Google Air Quality Monitoring	Global	High. Thanks to access to big data	Good optimization for long-term work in the city	Limited at the local level
SUEZ Smart Environment	Average	Average	Average. The requirement of large energy costs due to the need to maintain infrastructure in hard-to-reach areas	Narrow (for water and waste only)
Microsoft Azure IoT Central	High	High	Average	High. Thanks to the ability to integrate with different platforms and support for several types of environmental data.

SmartSantander and Microsoft Azure IoT Central have the most versatile approach to data collection, covering various environmental aspects such as air quality, water, noise, lighting, and others. Google Air Quality Monitoring, SUEZ Smart Environment, and SmartWater are more specialized in one type of data, namely air or water quality, which limits their flexibility for complete environmental monitoring.

Google Air Quality Monitoring and Microsoft Azure IoT Central have global coverage thanks to satellite monitoring capabilities and cloud infrastructure. SmartSantander and SUEZ Smart Environment are more localized and effective for specific regions or cities but require large investments to expand to other regions.

Google Air Quality Monitoring and Microsoft Azure IoT Central demonstrate the highest accuracy of predictions thanks to powerful machine learning algorithms and access to large amounts of data [6].

Common disadvantages of existing systems include the high cost of their installation and integration. All systems, except Google Air Quality Monitoring, require significant financial investments in sensor installation, infrastructure support, and maintenance. Most systems, such as SmartSantander, and SUEZ Smart Environment, have limited ability to quickly adapt to small geographical regions or scale to large areas without significant investments [7].

Therefore, existing systems for environmental monitoring based on IoT and machine learning have their unique advantages and disadvantages. The main advantages of such systems are the ability to collect large amounts of environmental data in real-time, the ability to use machine learning to analyze and predict the state of the environment, as well as reducing the cost of resource management [8]. At the same time, there are some challenges, such as high implementation and maintenance costs, limited scalability in different geographical conditions, the need for energy-efficient solutions, and the need to improve machine learning models to increase the accuracy of forecasts.

### 2.3. Problem statement

The main objective of the work is to develop a system architecture for environmental analysis based on IoT and machine learning with improved efficiency of data collection, transmission, and analysis. This includes optimizing the processes of data collection from IoT devices [9], their transmission and storage, as well as improving the accuracy of predictions based on machine learning algorithms.

The system should meet the following functional requirements:

- Real-time data collection: The system should provide continuous data collection from various IoT devices, such as temperature, humidity, air pollution, noise level sensors, etc.
- Data transmission: Data collected by IoT devices should be transmitted to a central server or cloud platform for further processing. The transmission should occur with minimal delays [10].
- Data storage: The system should ensure the storage of large volumes of environmental data in the storage for further analysis [11].
- Analysis and forecasting: A built-in machine learning model should analyze the collected data and provide predictions about the state of the environment [12].
- User interface: The system should have a user-friendly interface for data visualization and displaying analysis results in the form of graphs and analytical reports [13].
- Anomaly analysis: The system should detect anomalies in the data and warn of possible environmental disasters [14].

Non-functional requirements of the system include:

- Reliability: The system must operate uninterruptedly and provide high reliability of data transmission even in cases of partial loss of connection between IoT devices.
- Scalability: The architecture of the IoT system must be scalable to ensure the ability to connect a large number of sensors without loss of performance [15].
- Energy efficiency: IoT devices must operate for a long time with minimal energy consumption to ensure their use in remote areas.
- Security: The data collected by the system must be protected from unauthorized access. Encryption and authentication tools must be implemented [16].
- Accuracy: Machine learning models must provide high accuracy of predictions based on the collected data [17].
- Speed: The time for data processing and forecast generation must not exceed specified limits (usually no more than a few minutes after data receipt) [18].

Requirements for system integration, scaling, and support:

- Integration with cloud services: The system should be able to integrate with popular cloud platforms (e.g. AWS, Azure) for processing and storing large amounts of data [19].
- Support for different types of sensors: The system should be flexible and support the connection of different types of sensors, including those that measure different environmental parameters [20, 21].
- Cross-platform: The software should be available for use on different platforms (mobile devices, PCs) to provide access to data at any time and from any place.

The result of the work will be a developed and tested system for analyzing the state of the environment based on IoT and machine learning, which will have an improved data collection and analysis architecture. The system will provide more efficient and accurate monitoring of environmental indicators in real-time, increase the accuracy of forecasts, and optimize the cost of resource use.

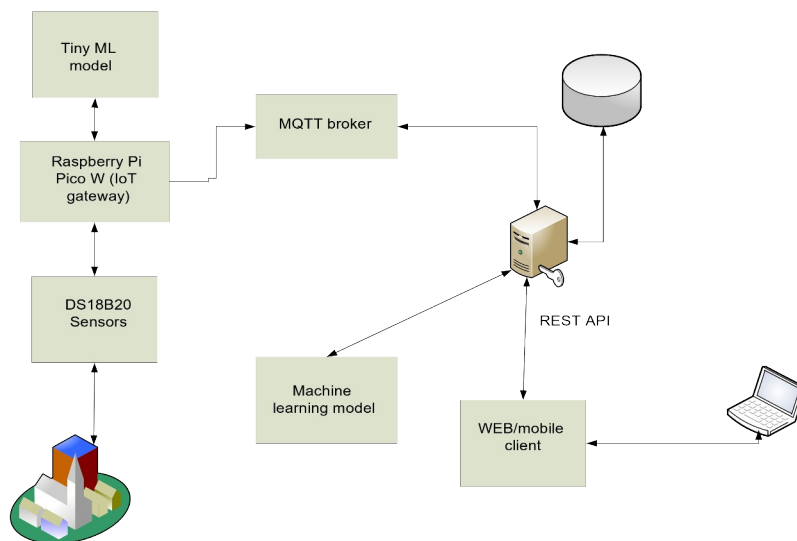
### 3. System design

#### 3.1. System components and their interaction

The entire system is divided into three main groups according to its functionality. Each group has its purpose and characteristics and requires separate design and development. These functional groups include:

- IoT devices—the hardware of the system, consisting of physical devices that can read data from the environment and interact with it. Actuators—devices that, after receiving commands, perform certain actions, which may include influencing the environment, for example, notifications, sound, or light. Sensors—devices that read data from the environment (for example, temperature, humidity, air quality, light level, etc.) and transmit them to IoT gateway devices. The IoT gateway is the point of the IoT system that connects all other devices. It serves to manage all devices, collect data, initially process them, and then send them in the selected way.
- Machine learning—a module responsible for advanced analysis, classification, and evaluation of data received from IoT devices. TinyML is a technology that allows for simple data analysis using machine learning algorithms on low-power devices with limited memory [22]. Within the framework of the system under development, TinyML can be used for initial data processing directly at the IoT gateway level, which will help to use system resources more efficiently. The data analysis module is responsible for the main data processing and preparation of the final analysis results [23].
- Software is the part of the system that is responsible for working with data, processing it, and storing it. An important task of this part of the system is to provide a convenient interface for interacting with the obtained system results [24]. The software should be scalable and independent of IoT devices and the machine learning module.

Fig. 1 shows the structural diagram of the system.



**Figure 1:** Structural diagram of the system

The central component of the system is the Internet of Things (IoT) devices, which are divided into two groups: sensors and actuators. The physical gateway acts as an intermediate link between the sensors/actuators and the cloud gateway.

The cloud gateway receives data from the physical gateway and performs the following tasks:

- Storage of large amounts of data and their initial processing.
- Data transmission for further processing and analysis.
- Providing access to the system through available networks.

The machine learning module performs the following tasks [25]:

- Analysis of the received data.
- Building forecasting models.
- Detection of anomalies, optimization, and decision-making.

Data processed by machine learning algorithms is returned to the system for application (for example, optimizing the operation of devices or providing results for users).

The control application is the main interface for user interaction with the system. It consists of the following components:

- Interface—a graphical or text interface for convenient data display and interaction with the system.
- Business analysis—tools for analyzing data to obtain valuable conclusions and recommendations.

The application is available as a mobile application for convenient use using phones and tablets. This will provide the opportunity to use the system more flexibly in various conditions. A web version of the application is also available for use via a browser. This allows you to conveniently use the system in stationary conditions.

The key connections of the system are as follows:

- Sensors and actuators transmit data to the physical gateway.
- The physical gateway interacts with the cloud gateway in two directions: transmits collected data and receives commands for sensors/actuators.
- The cloud gateway provides data transmission to the machine learning module. This component of the system is a module that is responsible only for receiving and processing input data. It should be as flexible as possible to be able to expand the list of data-sending protocols that it can process if necessary.
- Machine learning analyzes the data and transmits the results to the application for control. This module is closely related to the server part of the control application, as it stores and aggregates data in a convenient form for client applications [26].
- The control application provides user interaction with the entire system. The mobile application and the web version of the application are implemented in such a way that they contain only specific logic to correctly and conveniently display data on the platform they implement. At the same time, all specific business logic that is common to all client applications is implemented on the server part of the application. The system is designed in such a way that it is easy and fast to add new client platforms, regardless of the others. Each client platform receives all data from the server part, where it is stored and aggregated. The single point of truth of the system is the server, which in turn is connected to the database, machine learning module, and cloud gateway.

This architecture provides convenient data collection, processing, and analysis with subsequent decision-making to optimize processes in real-time.

System features:

- Iterative approach: A closed-loop provides a continuous optimization process.
- Automatic adjustment: The system can adapt its actions based on feedback.
- Modularity: Each stage can be scaled or changed without interfering with the others.

The system is suitable for automated monitoring of complex processes, such as resource management in agriculture, smart cities, industry, etc.

**Table 2**

Characteristics of system states

State	Description	Examples/ Technologies
Start the process	Starting a data collection and processing cycle.	Schedulers (cron jobs), IoT triggers
Data collection	Reading raw data from IoT sensors	Sensors DHT22, MQ135; Protocols MQTT, CoAP
Data transfer	Data is transmitted from sensors to the edge device for pre-processing	Wireless protocols: Wi-Fi, Bluetooth, Zigbee
Preliminary processing	The edge device performs noise filtering, data aggregation, and basic analysis using TinyML	TinyML (TensorFlowLite); Kalman signal filtering
Centralized analysis	Deep data analysis on the server using machine learning models and comparison with historical data	Servers AWS, Google Cloud; Python, Pandas, Scikit-learn
Prognostication	Based on the analysis, predictions of future events or system states are formed	Models LSTM; TensorFlow, PyTorch
Evaluation of results	Analysis of forecast accuracy and its compliance with benchmarks	Metrics: Accuracy, MSE Tools Jupyter Notebook, Dash
Recommendation generation	Making recommendations to optimize system performance or warn of potential risks	Tableau, Power BI; Integration through API
Sending results	Send analysis results and recommendations to responsible individuals or systems	Mobile notifications (Firebase); integration with Telegram, Slack
Feedback	Obtaining feedback from users or systems used to improve models and data collection parameters	Feedback Modules; API for collecting automated feedback
Completion of the cycle	End of the current cycle after all actions have been performed.	A Cyclic Approach to Continuous Optimization

### 3.2. Selection of system hardware components

Designing an IoT system requires consideration of many factors: system purpose, component types, energy efficiency, security, scalability, and cost requirements. Hardware selection should be comprehensive and based on an analysis of the needs and constraints of a specific application to ensure the effectiveness and durability of the IoT system.

#### 3.2.1. Purpose and system requirements

When choosing hardware, it is important to understand what the IoT system will be used for. Requirements may vary depending on the purpose:

- Environmental monitoring systems: These applications require sensors that can operate in extreme conditions, such as changes in temperature, humidity, or pollution levels. The selection of components should take into account the appropriate weather protection standards (IP rating for sensors).
- Smart home control systems: These systems require sensors that work with different types of interfaces (e.g. Zigbee, Z-Wave, Wi-Fi), as well as actuators to control various devices (light, temperature).
- Industrial IoT systems: Components are required that can withstand difficult conditions (high loads, explosive areas), as well as the ability to process large amounts of data in real-time.

#### 3.2.2. Types of hardware components

The choice of components has a huge impact on the performance of an IoT system. Microcontrollers are the heart of IoT devices, and their choice depends on the number of connected sensors, the required computing power, and energy efficiency requirements. The most popular ones are:

- ESP32: has built-in Wi-Fi and Bluetooth, making it a great choice for wireless IoT devices.
- Raspberry Pi Pico W: a compact microcontroller based on the ARM Cortex M0+, suitable for projects with limited resources.
- STM32: a family of microcontrollers based on the ARM Cortex, well suited for complex applications.
- Arduino: a popular choice for initial projects due to its ease of use and a wide selection of boards and accessories.

Sensors are an integral part of an IoT system, and choosing the right sensor depends on the requirements for accuracy, measurement range, and external operating conditions. By purpose, sensors are as follows:

- Temperature and humidity: DHT22, BME280.
- Gas sensors: MQ series for detecting gases such as CO<sub>2</sub>, CO, methane, etc.
- Motion and acceleration: motion sensors, and accelerometers for tracking movement and orientation.
- Pressure: pressure sensors (e.g. BMP280). Actuators (mechanical devices that perform actions such as turning lights or valves on/off) should also be selected according to the tasks set by the system.

Communication modules: For IoT devices, it is important to have an efficient connection for data transmission.

Communication modules are as follows:



- Wi-Fi: Suitable for short distances where there is access to the Internet or a local network.
- Bluetooth: Good for low-power devices such as BLE.
- Zigbee/Z-Wave: Protocols for multi-device networks (mesh networks), well suited for smart homes.
- LoRa: Low-power, long-range communication, ideal for agricultural and other long-distance applications.
- NB-IoT: High-speed mobile networks for IoT with low power consumption.

Power supply: Battery-based or accumulator-based systems must have optimized power consumption. You can use:

- Li-ion or Li-Po batteries for autonomous systems.
- Solar panels provide constant power in ecological systems.

Energy saving is a critical aspect, as many IoT devices operate autonomously and must have minimal power consumption.

- Power saving modes: Most modern microcontrollers have special sleep modes where they can be in an inactive state (deep sleep, standby) to reduce power consumption.
- Low-power wireless communication technologies: The use of protocols such as LoRa, Zigbee or NB-IoT helps reduce power consumption during data transmission.

IoT systems must be scalable, both in terms of the number of devices and the amount of data processed. Scalability can be defined by the following parameters:

- Modularity: Adding new components should be easy. For example, the ability to add new sensors or connect new types of network communication (LoRa or 5G).
- Compatibility with other protocols: Choosing hardware that supports multiple wireless communication standards allows for integration into a variety of networks.

Since IoT systems work with sensitive data, security is extremely important.

- Encryption and authentication: Choosing components that support reliable protection mechanisms, such as hardware data encryption or secure protocols for information exchange (TLS, SSL) [27].
- Access control: Implementing multi-level authentication to ensure the security of devices and data.

For use in extreme conditions (for example, outdoors or in industrial areas), components must have an appropriate level of protection, in particular, according to IP (Ingress Protection) standards [28].

## **4. Software development**

### **4.1. The general structure of the machine learning module**

The IoT and machine learning-based environmental condition analysis system is an innovative approach to environmental condition monitoring, analysis, and forecasting. The use of modern technologies such as TinyML for preliminary data analysis and traditional machine learning algorithms for in-depth analysis and forecasting ensures the efficiency and accuracy of the system. The main advantage of this approach is the optimization of computing resources and minimizing data transmission costs, which is important for IoT systems.

At the data collection stage, the system uses IoT devices such as the Raspberry Pi Pico W microcontroller, which are connected to DS18B20 temperature sensors, which provide accurate temperature data in a wide range of conditions. These sensors transmit the collected data to local computing devices for pre-processing using TinyML.

TinyML is a technology that allows you to run optimized machine learning models on devices with limited resources. In the system under development, TinyML is used for data preprocessing, which includes noise filtering, anomaly detection, and temperature condition classification. For example, if a sensor transmits a series of temperature values, TinyML can detect in real-time whether these values correspond to normal conditions or are signs of anomalous changes, such as sudden temperature changes [29].

Preprocessing data at the IoT device level has several important advantages. First, it reduces the amount of data that is transmitted over the network to a server or cloud, which reduces bandwidth requirements and saves energy. For example, instead of transmitting all measured temperature values, an IoT device can transmit only aggregated data, such as the average, maximum, or minimum, or signals about detected anomalies. Second, it increases the autonomy of the system: even if the connection to the server is temporarily unavailable, the devices can perform basic analysis locally [30].

After preliminary analysis, the data is transferred to a central server or the cloud, where deeper analysis is performed using more sophisticated machine learning models. These models can take into account more parameters than are available at the IoT device level, including historical data, climate trends, and external factors such as weather conditions or seasonal changes. This allows the system to provide not only an analysis of the current temperature state but also to make predictions about future changes. For example, the server model can use time series algorithms or recurrent neural networks (RNNs) to predict the temperature several days ahead, based on collected data and trends [31].

The integration of TinyML and traditional machine learning also allows for the creation of a multi-level analysis system. The first layer, based on TinyML, provides a quick response to local events, such as real-time warnings of a sharp drop in temperature. The second layer, running on the server, provides more complex insights and long-term forecasts [32]. This combination allows for a balance between data processing speed and analysis accuracy.

To predict temperature based on machine learning, the system uses historical data stored on the server, as well as data from other sources, such as weather stations or satellites. Combining these data sources allows you to increase the accuracy of forecasts. For example, a machine learning model can detect patterns in seasonal temperature changes or predict the consequences of extreme weather events. Overall, the combination of TinyML and traditional ML in a system for analyzing the state of the environment ensures its efficiency, accuracy, and scalability. This approach allows you to combine the advantages of local data processing provided by IoT devices and complex analysis and forecasting performed at the server level. This makes the system not only high-tech but also cost-effective, suitable for use in a wide range of tasks, from environmental monitoring to climate management in smart cities or agricultural lands.

## 4.2. TinyML

The development of the TinyML module for the environmental analysis system is a critical element that provides pre-analysis of data directly on IoT devices. The system will use the One-Class SVM algorithm, optimized for operation on devices with limited computing resources, such as the Raspberry Pi Pico W. This algorithm can train on normal data and detect anomalies in real-time, which makes it ideal for monitoring temperature in a changing environment. Compared to other algorithms, such as K-Nearest Neighbors or Random Forest, One-Class SVM has significantly lower memory consumption and higher speed. For example, K-Nearest Neighbors requires storage of all training data, which is not optimal for microcontrollers, and Random Forest is too complex for limited resources [33].

At the same time, One-Class SVM provides a high level of accuracy with minimal hardware requirements, which makes it particularly effective for detecting abnormal temperature changes. A more detailed comparative characteristic of the algorithms is given in Table 2.

**Table 3**

Comparison of machine learning algorithms

Algorithm	Complexity	Resource	Precision	Suitability For TinyML
K-Nearest Neighbors (KNN)	Low	Average	High	Suboptimal due to large memory
Decision Trees	Low	Low	Average	A good option for basic tasks
Random Forest	Average	High	High	Too complex for limited resources
Logistic Regression	Low	Low	Average	Suitable, but limited in complex tasks
One-Class SVM	Average	Average	High	Optimal for detection of anomaly

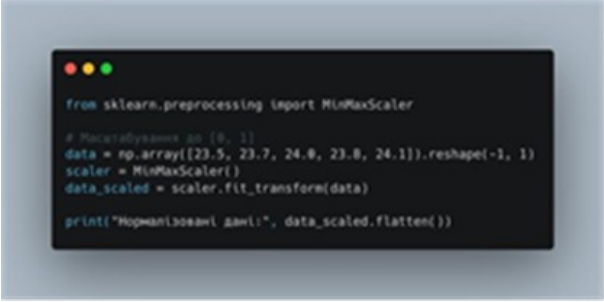
The model was run using normalized temperature data collected under normal conditions. This allows the algorithm to create a hyperplane that separates normal values from potential anomalies. For example, if the system records temperature values that fall outside the trained hyperplane, this signals a possible problem, such as a technical failure or environmental risk. The model was built using the sci-kit-learn library in Python and then optimized for use on a microcontroller. The data was normalized using MinMaxScaler to ensure consistency with the limited resources of the device. The algorithm uses a radial basis function as the kernel for building the model, which provides better adaptation to nonlinear data characteristic of temperature changes. The finished model was saved in a format compatible with TinyML and loaded onto a Raspberry Pi Pico W. This allows for real-time analysis, even if the device is offline. In practice, this works as follows: a sensor, such as a DS18B20, transmits temperature data to a microcontroller, where it is normalized and fed into the model input. The model classifies the values as normal or abnormal in real-time. If an anomaly is detected, the system can instantly signal a problem, which is especially important for environmental monitoring.

The chosen approach is optimal for several reasons. First, the One-Class SVM has a high level of generalization, which allows it to detect new, previously unknown anomalies. This is important for systems operating in a changing environment and where unpredictable situations may arise [34]. Second, the model is quite compact and does not require large computational resources, which allows it to be integrated even on devices with very limited memory. Third, local data processing significantly reduces the load on the network, since only critical events or aggregated results are transmitted, and not all raw data. This also reduces the system's power consumption, which is a key factor for autonomous IoT devices [35].

Compared to alternatives such as neural networks, One-Class SVM takes up less memory and has a faster execution time, making it suitable for fast real-time responses. Although neural networks can provide more complex analysis, their implementation at the microcontroller level requires significantly more resources, which is not efficient for our task [36]. That is why One-Class SVM is chosen as the base algorithm for the TinyML module.

Overall, the developed module allows you to integrate preliminary data analysis into the system, making it more flexible and efficient. The ability to detect anomalies in real-time, reduce the amount of transmitted data, and reduce power consumption allows you to build an environmentally and economically efficient system that can be used for a wide range of environmental monitoring tasks. This solution combines accuracy, speed, and resource optimization, which is critical for new-generation IoT systems.

The implementation of the TinyML module looks like this. First, data collection occurs, which involves obtaining values from sensors. At this stage, it is important to make sure that the data is read correctly, without errors or equipment malfunctions. Next, data cleaning occurs, which is necessary to remove noise, erroneous measurements, or missing values. Next, data normalization (scaling) occurs, which is necessary to ensure that all values are in the same range (for example, from 0 to 1). The script responsible for normalization is shown in Fig. 2.




```
from sklearn.preprocessing import MinMaxScaler

# Normalization to [0, 1]
data = np.array([23.5, 23.7, 24.0, 23.8, 24.1]).reshape(-1, 1)
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

print("Нормализован!:", data_scaled.flatten())
```

**Figure 2:** Data normalization

The script responsible for initializing the model is shown in Fig. 3. The figure describes in detail the set of parameters that adjust the model's operation. Next, the model is trained using the method shown in Fig. 4.



```
from sklearn.svm import OneClassSVM

kernel = 'rbf' # Радиально-базисная функция
nu = 0.1 # Число параметров
gamma = 'scale' # Автоматическое масштабирование
degree = 3 # Степень полиномиальной ядра
coef0 = 0.0 # Коэффициент полиномиального ядра
tol = 1e-4 # Точность завершения
shrinking = True # Сокращение
cache_size = 100 # Размер кэша в МБ
verbose = False # Диагностика
max_iter = -1 # Максимальное количество итераций (по умолчанию)

model = OneClassSVM(
    kernel=kernel,
    nu=nu,
    gamma=gamma,
    degree=degree,
    coef0=coef0,
    tol=tol,
    shrinking=shrinking,
    cache_size=cache_size,
    verbose=verbose,
    max_iter=max_iter
)
```

**Figure 3:** Model initialization



```
model.fit(data_scaled)
```

**Figure 4:** Model training

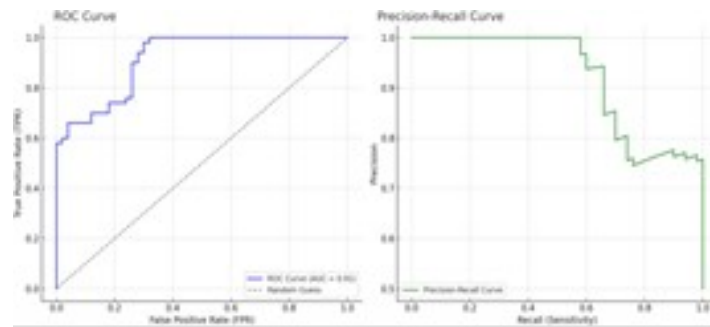
After training with the script, the model is saved for further use on the device.

Fig. 5 shows the results of measuring the model's performance using the ROC curve (operating characteristic curve) and Precision-Recall curve (Precision-Recall curve). The ROC curve is used to

evaluate the model's ability to distinguish between classes by comparing the sensitivity (True Positive Rate) and the level of false positives (False Positive Rate) at different decision thresholds. The closer the ROC curve is to the upper left corner, the better the model's performance, and the area under the curve (AUC) is an integral indicator that reflects the overall quality of classification. The Precision-Recall curve focuses on assessing the relationship between precision (Precision) and sensitivity (Recall), which is especially useful for analyzing models in problems with class imbalance. It shows how well the model retains precision as the number of true positives found increases, allowing you to assess the trade-off between missing positive cases and avoiding false positives.

Analyzing the data in Fig. 5, we can conclude that:

- The AUC is 0.91, which indicates good performance, but not perfect.
- The curve has several deviations from the ideal slope to the upper left corner, indicating the presence of false predictions (false positive and false negative classifications).
- Precision gradually decreases with increasing Recall, which is typical for models with good performance.
- The curve shows a trade-off between Precision and Sensitivity (Recall).



**Figure 5:** ROC and Precision-Recall curves

### 4.3. Development of a temperature forecasting model

LSTM (Long Short-Term Memory) is one of the modifications of recurrent neural networks (RNN), developed to solve the problem of long-term dependencies in time series [37]. The main idea of LSTM is to implement special mechanisms (memory cells) that allow the network to store important information for long periods and ignore unimportant ones. This feature makes LSTM an ideal choice for working with sequential data, such as temperature series, text data, audio, and video.

In standard RNNs, each layer passes all the information to the next stage, which leads to the problem of exploding or vanishing gradients, where important information is lost due to many steps in the sequence. LSTM solves this problem by introducing forget, input, and output gates [38].

- The forget gate decides what information from the previous state should be removed. It uses a sigmoid activation function to determine how important each element is.
- The input gate adds new information to the memory cell, also through sigmoid activation.
- The output gate controls how much of the information from the memory cell will be passed as the current-time output.

These three gateways work together to ensure that important information is preserved and irrelevant information is ignored, allowing the model to maintain context and understand sequences even over large time intervals [39].

The created model for temperature forecasting is based on the architecture of a recurrent neural network (RNN) using a modification of LSTM (Long Short-Term Memory). This architecture allows for both short-term and long-term dependencies in time series data to be taken into account, making it ideal for forecasting tasks.

The model consists of three consecutive LSTM layers. The first two LSTM layers are configured to return sequences, which allows all information about the dependencies between time elements to be passed to the following layers. The third LSTM layer completes the data processing, condensing them into a highly informative vector [40]. Each LSTM layer uses 50–100 neurons to provide sufficient capacity to process complex dependencies in the data. To reduce overfitting, Dropout regularization with levels of 0.2–0.3 was used, which randomly “turns off” some of the neurons during training, thereby increasing the model’s resistance to noise and irregularities in the data [41].

After processing by the recurrent layers, the data is passed to the Dense layers. The first Dense layer with 64 neurons uses the ReLU (Rectified Linear Unit) activation function, which adds nonlinearity and allows the model to detect complex dependencies. The second Dense layer with 32 neurons performs a similar function, but with fewer parameters, preparing the data for the output layer. The output layer has one neuron with linear activation, which allows the model to predict a specific temperature value.

The model architecture includes optimization using the Adam algorithm, which works well with complex optimization problems and quickly converges to the optimal solution. The training uses the mean square error loss function (MeanSquaredError), which is suitable for regression problems, in particular, for predicting numerical values [42]. The input data is formed in the form of sequences with a fixed length, which allows the models to calculate the temperature dynamics over time.


The main advantages of this model are its ability to process time series data and take into account both short-term and long-term dependencies. The use of multiple LSTM layers provides a deep understanding of the patterns in the data, and Dense layers allow the model to adapt to complex dependencies. Dropout regularization prevents overtraining, which is important for ensuring the stability of the model in conditions of a limited amount of data.

Among the disadvantages of such an architecture, it is worth noting the high computational complexity. Three LSTM layers with a large number of neurons require significant resources for training, which can be a problem for less powerful hardware. In addition, the model depends on high-quality data preparation, in particular, scaling and formatting sequences. Improper data preparation can significantly reduce the accuracy of predictions. Another limitation is the potential difficulty of integrating the model into real-time systems due to its long inference time, especially on less powerful devices [43].

Overall, the model is a powerful tool for temperature forecasting. It can accurately predict future values based on historical data and works well with time series due to its deep architecture. However, its computational requirements need to be optimized or adapted to work under resource-constrained conditions.

Fig. 6 shows the script used to create this model. First, the model is created using the Sequential class, which allows layers to be added sequentially. The first layer is an LSTM with 100 neurons, configured to return the entire array of output sequences (`return_sequences=True`). This allows all information about temporal dependencies to be passed to the next layer. The layer accepts input data of the form `(X_train.shape, X_train.shape [2])`, where the first parameter is the length of the sequence, and the second is the number of features at each time step. After the first LSTM layer, a Dropout with a probability of 0.3 is added to avoid overtraining by randomly turning off some neurons during training. The second layer is another LSTM with 100 neurons, which also returns sequences, deepening the model’s understanding of the dependencies in the data. This is followed by another Dropout with the same probability of 0.3. The third LSTM layer has 50 neurons and does not return sequences (`return_sequences = False`), which means that the output will be condensed into a vector that represents the entire context of the sequence. This output is fed to the

Dropout layer with a probability of 0.2. Two Dense layers are then used. The first has 64 neurons with ReLU (Rectified Linear Unit) activation, which provides nonlinear data transformation and helps the model find complex dependencies. The second Dense layer with 32 neurons further reduces the vector size, providing a generalization of information before the final layer. The output layer has one neuron with linear activation (linear), which allows for predicting one numerical value—the temperature forecast. After determining the architecture, the model is compiled using the adam optimizer, which effectively updates the weights of the neural network during training, and the mean\_squared\_error loss function, which is suitable for regression problems. The code is completed by calling the model summary(), which outputs a brief description of the model architecture, including the number of parameters to be optimized and the total number of layers. Such an architecture is well suited for working with time series, as it takes into account both short-term and long-term dependencies thanks to the combination of LSTM and Dense layers. Dropout regularization helps avoid overfitting, ensuring model stability even in cases where the amount of training data is limited.



```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

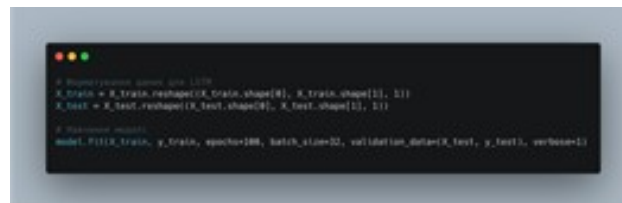
model = Sequential()
model.add(LSTM(100, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2]),
              dropout=0.2))
model.add(LSTM(100, return_sequences=True,
              dropout=0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='linear'))

model.compile(optimizer='adam', loss='mean_squared_error')
model.summary()

```

**Figure 6:** Creating a model

Fig. 7 shows how data formatting and model training takes place.



```

X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

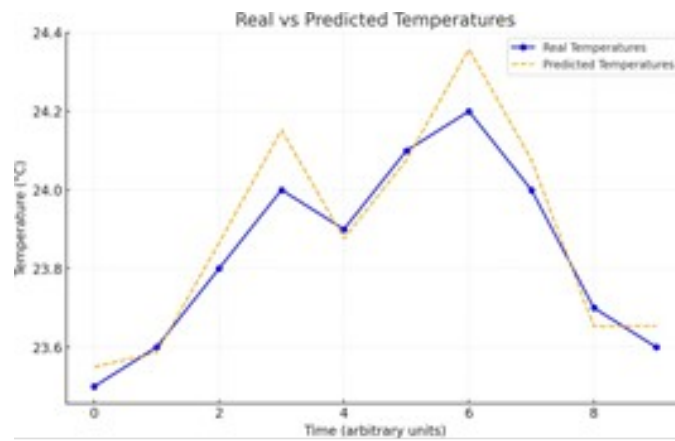
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test, y_test), verbose=1)

```

**Figure 7:** Model training

The evaluation of the model performance is shown in the graph (Fig. 8).

The graph shows a comparison of actual and predicted temperatures over some time. The blue line represents the actual temperature, and the orange line represents the model's predicted values. The predicted values show a good fit to the actual data, indicating the overall accuracy of the model. At most points, the deviation between the actual and predicted values is minimal and within the acceptable level of noise or uncertainty that may be inherent in the model.



**Figure 8:** Evaluation of the model

The predicted line follows the general trend of the actual data, in particular, when the temperature increases from 23.8°C to 24.1°C, the model correctly predicts this increase. This indicates its ability to account for local variations in the time series. At some points, there are minor discrepancies between the actual and predicted values, for example, when the actual temperature is 23.7°C, the model predicts slightly lower values. Such errors can be caused by noise in the data or by insufficient training examples for specific temperature ranges. The differences between the predictions and the actual values remain stable within  $\pm 0.1^{\circ}\text{C}$ , which is acceptable for such a task and indicates the stability of the model even when random factors affect the actual values. The model also predicts peak temperature values well, for example, maxima at 24.2°C, which indicates its ability to take into account both long-term and short-term dependencies that could lead to local maxima. Overall, the model demonstrates a high ability to predict temperature with small deviations, correctly modeling the general trend and local changes, which makes it a reliable tool for predicting temperature in real conditions. Minor discrepancies can be reduced by further optimizing the model or increasing the amount of training data. The conclusion shows that the model is suitable for temperature analysis and prediction tasks in IoT systems or environmental monitoring.

#### 4.4. Hardware script development

The code implements a complex system that reads temperature data from a DS18B20 sensor, analyzes it using a pre-trained machine-learning model, and sends the results via the MQTT protocol. First, the MQTT connection is configured. To do this, an MQTT client with a unique identifier Pico W\_Temperature is created, which connects to the public broker test.mosquitto.org via port 1883. If the connection is interrupted, the `connect_mqtt` function automatically tries to restore the connection, which ensures the system's resilience to network failures.

Next, the DS18B20 sensor is configured. The OneWire bus connected to the microcontroller's GPIO15 is used. All available sensors on this bus are scanned, and their unique addresses are identified. If the sensors are not found, the program displays a corresponding message and enters an infinite wait loop. This ensures that without access to the sensor, the system will not perform unnecessary operations.

After the sensor is initialized, the machine learning model is stored in the `temperature_model.pkl` file is loaded, along with a normalizer that will scale the input data to the same ranges used during model training. If the model file is missing or corrupt, the program reports an error and does not continue execution.

The main part of the code works in a loop. Each loop starts with the DS18B20 sensor starting a temperature measurement, after which the data is read. If this is the first loop, the system saves the current temperature as a base value and continues to the next loop. For each subsequent reading, the temperature gradient is calculated, that is, the change in temperature compared to the previous



value. This data (temperature and gradient) forms an input vector that is passed to the machine learning model. The model predicts whether the current state is normal or abnormal. The result of the analysis is formed as a string: “Normal” or “Anomaly”. Next, a JSON object is formed, which includes the current temperature, gradient, status (model result), and timestamp. This object is encoded in JSON format and sent via MQTT to the specified temperature/data topic.

Outputting data to the console allows you to monitor the system status: current temperature, analysis result, and confirmation of sending a message to the MQTT broker. In case of any error, for example, loss of communication with the broker or failure in data processing, the program reconnects to the broker or continues the cycle with the next available data.

This code provides both local temperature analysis using a machine learning model and sending results to a remote system via a lightweight MQTT protocol. Its structure allows you to integrate the solution into larger IoT systems for monitoring the state of the environment and notifying you about anomalies in real-time [44–46]. Thanks to error handling and automatic connection recovery, the system remains resilient to network or hardware failures.

#### **4.5. Application software development**

The developed server software for the environmental analysis system using IoT and machine learning is based on Node.js and MQTT technologies for processing data from sensors in real-time. The server system receives data from IoT devices, such as Raspberry Pi Pico W, analyzes them using a pre-built machine learning model, and provides centralized storage and access to the results via a RESTful API.

The main part of the system is implemented on Node.js, which provides high-performance and asynchronous processing of messages from numerous IoT devices. The MQTT protocol is used to exchange data between the sensors and the server, which is ideal for lightweight IoT systems. The server is connected to an MQTT broker, which receives messages about the current temperature and gradient published by the sensors. The server is subscribed to the corresponding topic, for example, temperature/data, and receives all data in real-time. Incoming messages are processed using the mqtt library for Node.js, which provides easy integration with the MQTT broker and convenient access to data.

After receiving the data, it is passed to the machine learning module, integrated through the Python Shell library. This module works with a machine learning model previously created using sci-kit-learn. The model is loaded as a file and is used to analyze the received data. For each new message, the server calls a Python script, passing the data via standard input, where the model processes the information and returns it. This allows the server to classify the system state in real-time and generate the appropriate status for each sensor.

The processing results are stored in a PostgreSQL relational database. Each record includes temperature, gradient, timestamp, system status (normal or abnormal), and a unique sensor identifier. This structure allows you to store historical data and quickly execute queries for trend analysis. PostgreSQL’s high performance allows you to work with large amounts of data and provides fast access to information for reporting and analytics.

A RESTful API is implemented to access the system using the Express.js framework. The API provides endpoints for retrieving current data, accessing historical records, and obtaining aggregated statistics, such as the average temperature per day or the number of recorded anomalies. The API also supports authentication using JWT tokens to ensure data access security.

A web application has been developed for visualization, allowing users to monitor the system status in real-time. The web application is built on React.js and receives data from the server API. The interface includes temperature trend graphs, tables with historical data, and a notification panel about anomalies. Additionally, push notification functionality via web sockets has been implemented, allowing users to be instantly informed about critical anomalies.

The development of the client part on React for the environmental status analysis system was aimed at creating an intuitive and functional interface that allows users to monitor temperature

indicators, trends, anomalies, and other data coming from the server part in real-time. The interface provides both analytical functions and the ability to quickly respond to critical situations.

The client part is built using the React library, which allows the creation of a component-based approach to building the interface. React Context API is used to manage the state, which provides centralized data transfer between components. Modern React functionalities, such as `useState`, `useEffect`, and `useContext` hooks, were actively used in the development. The Axios library was additionally used to process HTTP requests to the server API.

In addition, the web application is adapted to work on mobile devices. For this, adaptive design using CSS Flexbox and Grid is used. Components automatically adjust to the screen width, which provides convenient viewing of data on both large screens and smartphones.

The interface development also included ensuring accessibility. Descriptive attributes for buttons and interactive elements were added, and the color scheme was optimized for users with color vision impairments. Testing of the client part was carried out using Jest and React Testing Library to ensure the stable operation of key components.

The result of the development was a powerful and user-friendly interface that allows users to monitor the state of the environment in real-time and receive analysis of historical data. Thanks to React, it was possible to create a dynamic application with a modern design that provides a high level of user interaction with the system.

The entire system is tested to work with a large number of connected devices and provides data processing with high speed and accuracy.

The developed software is a reliable tool for collecting, analyzing, and visualizing data about the state of the environment. It integrates IoT devices, machine learning, and modern server technologies, providing users with deep analysis and the ability to quickly respond to detected anomalies.

## Conclusions

As a result of the analysis of the subject area, the relevance and prospects of the selected topic of work in the context of modern technological development were studied in detail. The combination of technologies such as IoT and machine learning is relevant and promising, allowing to ensure maximum productivity of data collection and deep analysis. Additionally, a comparative analysis of existing technologies and systems was conducted. Based on a comparison of their capabilities, technical and functional characteristics, pricing, and marketing policies, a list of basic requirements and characteristics of the system was formed.

An important stage of the work on the design of the system was the definition of the main structural components of the system by role, structure, purpose, and characteristics. The system is conventionally divided into two main interconnected parts: hardware and software. The hardware part of the system includes physical devices of the Internet of Things, which are responsible for interacting with the environment by collecting data and influencing it. The software part includes software running on hardware, client, and server. An important part of the software is the machine learning module, which by architecture works both on the server and hardware. The system architecture was created in such a way that the system was as simple as possible to develop and scale. This result was achieved by designing each element of the system, observing maximum independence from other elements of the system. Although all components closely interact with each other, this cooperation is carried out only under predetermined contracts. This approach also allows for parallel development of the system by different teams to quickly achieve new results in the form of new features and system expansion. The system was successfully created in the Wokwi virtual environment and prepared for work with the software.

The software part of the system for analyzing the state of the environment based on machine learning and IoT has been created. The structure of the machine learning module includes two parts, the first of which is a TinyML model that runs on an IoT device. The second part of the machine learning module is a more powerful model that runs on the server and deeply analyzes

data. Software for an IoT device has been created and tested in a virtual environment. The process of developing application software has been created and described, which includes a server part on NodeJs and a client part on ReactJs.

During the testing of the system, all developed components were integrated and their performance results were collected. Based on the results of the system, the system's weaknesses and strengths were identified, and the main technical characteristics and features of the system's performance were described.

The developed system is an effective tool for analyzing a large amount of environmental data and can solve real-world problems related to improving decision-making efficiency in various areas and directions. The use of TinyML technology helped to increase the efficiency of anomaly detection by up to 20 percent, compared to similar systems that do not integrate this technology. Due to the high energy efficiency of the selected components and the combination with TinyML technologies for local data processing, the system's energy consumption is 10-15 percent more efficient than existing analogs. Due to the low price of the selected devices, the full payback time of the system and the initial costs for installing or integrating the system are 20 percent lower than existing systems. Due to the high modularity and flexibility of the system components, the speed of installation, integration, and adaptation of the system is twice as fast as existing systems.

## Declaration on Generative AI

While preparing this work, the authors used the AI programs Grammarly Pro to correct text grammar and Strike Plagiarism to search for possible plagiarism. After using this tool, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

## References

- [1] B. Zhurakovskiy, et al., Smart house management system, in: *Emerging Networking in the Digital Transformation Age, TCSET 2022, Lecture Notes in Electrical Engineering*, vol. 965, 2023, 268–283. doi:10.1007/978-3-031-24963-1\_15
- [2] J. Holler, et al., *From machine-to-machine to the internet of things: Introduction to a new age of intelligence*, Academic Press, 2014.
- [3] K. Kolbasova, et al., Smart home network based on Cisco equipment, in: *Cybersecurity Providing in Information and Telecommunication Systems II*, vol. 3550, 2023, 70–80.
- [4] V. Zhebka, et al., Methodology for predicting failures in a smart home based on machine learning methods, in: *Workshop on Cybersecurity Providing in Information and Telecommunication Systems, CPITS*, vol. 3654 (2024) 322–332.
- [5] K. P. Murphy, *Machine learning: A probabilistic perspective*, MIT Press, 2012.
- [6] A. Geron, *Hands-on machine learning with Scikit-learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*, O'Reilly Media, 2019.
- [7] I. Goodfellow, Y. Bengio, A. Courville, *Deep learning*, MIT Press, 2016.
- [8] S. Raschka, V. Mirjalili, *Python machine learning: Machine learning and deep learning with Python, Scikit-Learn, and TensorFlow 2*, 3<sup>rd</sup> Edition, Packt Publishing, 2019.
- [9] J. Gubbi, et al., Internet of things (IoT): A vision, architectural elements, and future directions, *Future Gener. Comput. Syst.* 29(7), 2013, 1645–1660.
- [10] B. Zhurakovskiy, N. Tsopa, Assessment technique and selection of interconnecting line of information networks, in: *3<sup>rd</sup> International Conference on Advanced Information and Communications Technologies (AICT)*, 2019, 71–75 doi:10.1109/AIACT.2019.8847726
- [11] B. Zhurakovskiy, et al., Comparative Analysis of Modern formats of Lossy Audio Compression, in: *Cyber Hygiene*, vol. 2654, 2020, 315–327.
- [12] M. Chen, J. Wan, F. Li, Machine-to-machine communications: Architectures, standards and applications, *KSII Trans. Inter. Inf. Syst.* 6(2) (2012) 480–497. doi:10.3837/tiis.2012.02.002

- [13] V. Andrii, et al., Fault identification in linear dynamic systems by the method of locally optimal separate estimation, in: *Emerging Networking in the Digital Transformation Age, Part of the Lecture Notes in Electrical Engineering book series*, vol. 965, 2022, 634–651. doi:10.1007/978-3-031-24963-1\_37
- [14] N. Fedorova, et al., Software system for processing and visualization of big data arrays, in: *Advances in Computer Science for Engineering and Education, Lecture Notes on Data Engineering and Communications Technologies*, vol. 134, 2022, 324–336. doi:10.1007/978-3-031-04812-8\_28
- [15] B. Zhurakovskiy, et al., Processing and analyzing images based on a neural network, in: *Cybersecurity Providing in Information and Telecommunication Systems*, vol. 3654, 2024, 125–136.
- [16] B. Zhurakovskiy, et al., Secured remote update protocol in IoT data exchange system, in: *Cyber Security Providing in Information and Telecommunication Systems*, vol. 3421, 2023, 67–76.
- [17] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.
- [18] B. Zhurakovskiy, et al., Traffic control system based on neural network, in: *Digital Ecosystems: Interconnecting Advanced Networks with AI Applications, Lecture Notes in Electrical Engineering*, vol. 1198, 2024, 522–542. doi:10.1007/978-3-031-61221-3\_25
- [19] B. Zhurakovskiy, et al., Modifications of the correlation method of face detection in biometric identification systems, in: *Cybersecurity Providing in Information and Telecommunication Systems*, vol. 3288, 2022, 55–63.
- [20] M. Moshchenko, et al., Optimization algorithms of smart city wireless sensor network control, in: *Cybersecurity Providing in Information and Telecommunication Systems II*, vol. 3188, 2021, 32–42.
- [21] V. Druzhynin, et al., Features of processing signals from stationary radiation sources in multi-position radio monitoring systems, in: *Cybersecurity Providing in Information and Telecommunication Systems*, vol. 2746, 2020, 46–65.
- [22] E. Alpaydin, *Introduction to machine learning*, MIT Press, 2020.
- [23] P. Balakrishnan, S. K. Srivatsa, *Programming the internet of things: An introduction to building integrated IoT solutions*, Mc Graw-Hill, 2018.
- [24] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, *Comput. Netw.* 54(15) (2010) 2787–2805.
- [25] T. Erl, Z. Mahmood, R. Puttini, *Cloud computing: concepts, technology & architecture*, Prentice Hall, 2013.
- [26] B. Zhurakovskiy, et al., Calculation of quality indicators of the future multiservice network, in: *Future Intent-Based Networking. Lecture Notes in Electrical Engineering*, vol. 831, 2022, 197–209. doi:10.1007/978-3-030-92435-5\_11
- [27] B. Zhurakovskiy, et al., Enhancing information transmission security with stochastic codes, in: *CQPC-2024: Classic, Quantum, and Post-Quantum Cryptography*, vol. 3829, 2024, 62–69.
- [28] J. W. Creswell, J. D. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, SAGE Publications, 2017.
- [29] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.
- [30] S. Marsland, *Machine Learning: An Algorithmic Perspective*, Chapman and Hall/CRC, 2015.
- [31] S. Shah, P. Mishra, S. Jain, *IoT-Enabled Environmental Monitoring System for Smart Cities*, Springer, 2020.
- [32] M. Richardson, S. Wallace, *Getting started with Raspberry Pi*, O'Reilly Media, 2014.
- [33] P. Warden, D. Situnayake, *TinyML: Machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*, O'Reilly Media, 2019.
- [34] C. Banbury, et al., Micronets: Neural network architectures for deploying Tiny ML Applications, *arXiv*, 2021. doi:10.48550/arXiv.2010.11267
- [35] W. Shi, et al., Edge computing: Vision and challenges, *IEEE Inter. Things J.* 3(5) (2016) 637–646.

- [36] X. Shi, et al., Convolutional LSTM network: A machine learning approach for precipitation nowcasting, in: 28<sup>th</sup> International Conference on Neural Information Processing Systems, 2015, 802–810.
- [37] D. Gope, G. Dasika, M. Mattina, Ternary hybrid neural-tree networks for highly constrained IoT applications, in: Proceedings of Machine Learning and Systems, 2019, 190–200.
- [38] H.-P. Cheng, et al., Msnet: Structural wired neural architecture search for internet of things, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2019.
- [39] R. David, et al., TensorFlow Lite Micro: Embedded machine learning on Tiny ML systems, arXiv preprint arXiv, 2020. doi:10.48550/arXiv.2010.08678
- [40] C. R. Banbury, et al., Benchmarking Tiny ML systems: Challenge sand direction, arXiv preprint arXiv, 2020. doi:10.48550/arXiv.2003.04821
- [41] I. Fedorov, et al., SpArSe: Sparse architectures earch for CNNs on resource constrained microcontrollers, in: Advances in Neural Information Processing Systems, 2019, 4977–4989.
- [42] A. Wan, et al., Fbnetv2: Differentiable neural architecture search for Spatia land channel dimensions, in: The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
- [43] A. Agrawal, A. An, M. Papagelis, Learning emotion-enriched word representations, in: 27<sup>th</sup> International Conference on Computational Linguistics, 2018, 950–961.
- [44] V. Dudykevych, et al., Platform for the security of cyber-physical systems and the IoT in the intellectualization of society, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 449–457.
- [45] Z. Hu, et al., Bandwidth research of wireless IoT switches, in: IEEE 15<sup>th</sup> International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (2020). doi:10.1109/tcset49122.2020.2354922
- [46] V. Sokolov, et al., Method for increasing the various sources data consistency for IoT sensors, in: IEEE 9<sup>th</sup> International Conference on Problems of Infocommunications, Science and Technology (2023) 522–526. doi:10.1109/PICST57299.2022.10238518