

BANDWIDTH OPTIMIZATION FOR SENSOR NETWORKS WIRELESS IN AGRICULTURE WITH PYTHON COMPRESSION IoT-Based

Ayu Nasyua¹, Cut Widya Fatwa²

^{1,2} Informatics Engineering, STMIK KAPUTAMA

E-mail: ayunasyua@gmail.com, Cutwidyaf@gmail.com

ABSTRACT

This research aims to improve the efficiency of data transmission to wireless sensor networks in agriculture by compression using the IoT-based Python programming language. This research provides novelty in the field of Internet of Things (IoT) and Wireless Sensor Networks (WSN) for smart agricultural systems, especially in terms of bandwidth optimization and data compression by proposing a Python-based data compression method. This study uses a soil moisture sensor to measure soil moisture as a source of experimental data, which will collect data for 12 hours within 7 days. This research succeeded in significantly reducing the size of the data transmitted from the field sensor to the data center, without sacrificing the accuracy of the data needed to monitor crop conditions. The results of the comparison of data before and after compression can be seen in the boxplot diagram and bar chart to see the average. From the following research results, it opens up opportunities for optimizing bandwidth usage and reducing operational costs in IoT-based agricultural systems.

Keywords: *Wireless Sensor Network (WSN), Agricultural Industry, Bandwidth, Data Compression, Python, Performance Optimization*

Article History

Received: Januari 2026

Reviewed: Januari 2026

Published: Januari 2026

Plagiarism Checker No
235

Prefix DOI :

[10.8734/Kohesi.v1i2.365](https://doi.org/10.8734/Kohesi.v1i2.365)

Copyright : Author

Publish by : Kohesi



This work is licensed
under a [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/)

1. INTRODUCTION

The Industrial Revolution 4.0 has brought significant transformations to various sectors, including agriculture. The concept of precision agriculture, which combines information and communication technology with conventional farming practices, has become a major focus for increasing agricultural productivity and efficiency. One key technology supporting

precision agriculture is the wireless sensor network (WSN).

Wireless sensor networks consist of a large number of small sensors distributed across agricultural areas to collect environmental data such as temperature, humidity, light, and soil nutrient levels. The collected data is then processed to generate useful information for farmers in making decisions regarding irrigation, fertilization, and pest control.

While WSN offers great potential for increasing agricultural productivity, there are several challenges that need to be addressed. One of the main challenges is bandwidth limitations. The more sensors used and the higher the frequency of data collection, the greater the bandwidth requirements. Limited bandwidth can lead to delays in data transmission, data loss, and overall system performance degradation.

One approach to overcome the problem of limited bandwidth is to apply data compression techniques. Data compression allows data size to be reduced without significant information loss. This allows data to be transmitted more efficiently over networks with limited bandwidth, with database development in Python.

2. MATERIALS AND METHODS

2.1 Research Approach

Research uses a quantitative approach with an experimental method with an agricultural case study design. A wireless sensor network system was developed to optimizing bandwidth in IoT-based agricultural management.

2.2 Research Approach

The formulation of the research problem is based on the research objectives:

1. How does data compression using Python affect bandwidth savings in wireless sensor networks in IoT agricultural applications?

2. How does bandwidth optimization affect the resilience and reliability of wireless sensor networks in dynamic agricultural environments?

2.3 Research Approach

The subject of this research is a wireless sensor network system used in IoT-based agricultural applications. This includes sensor devices, a Python-based data compression algorithm, and wireless communication technology that supports data transmission in smart farming systems.

This study uses an experimental approach to test the effectiveness of compression techniques in optimizing bandwidth. The research was conducted with the following steps:

- **Selection of sensor devices:** The sensors used in this study include temperature, soil moisture, and pH.
- **Wireless sensor network design:** WSN systems are designed to connect sensors to a central server using wireless communication technologies such as LoRa or WiFi.
- **Implementation of data compression with Python:** Data compression will be implemented using Python algorithms, such as Huffman Coding or other suitable compression techniques.
- **Performance measurement:** Network performance will be evaluated by comparing bandwidth usage and data

latency before and after compression is applied

2.4 Data Collection

The collected data includes the size of data transmitted by sensors with and without compression, as well as network performance in terms of latency and bandwidth usage. Data was collected every 12 hours for 7 days

Data Hasil Monitoring Sensor (Suhu, Kelembaban Tanah, pH) selama 7 Hari:

Hari	Jam	Suhu (°C)	Kelembaban Tanah (%)	pH Tanah
1	00:00	25.3	45.2	6.5
	12:00	28.1	43.5	6.6
2	00:00	24.8	46.0	6.4
	12:00	27.4	44.0	6.7
3	00:00	25.5	47.0	6.5
	12:00	28.0	42.5	6.8
4	00:00	24.6	48.2	6.6
	12:00	27.3	44.3	6.7
5	00:00	25.2	45.5	6.5
	12:00	28.2	43.2	6.6
6	00:00	24.7	46.3	6.4
	12:00	27.6	44.1	6.7
7	00:00	25.4	45.8	6.5
	12:00	28.3	43.0	6.6

2.4 Internal Data Analysis

Data analysis will be conducted using statistical methods to compare the results of bandwidth usage and data quality before and after the compression technique is applied. Significance tests will be conducted. was conducted to see whether the differences found between the two conditions had a significant effect.

2.4.1 Organizing Data

- **Data before compression:** Sensor data collected without any compression.
- **Data after compression:** The same data, but after going through a compression technique (using Python, for example)

Hari	Jam	Suhu (°C)	Kelembaban Tanah (%)	pH Tanah
1	00:00	25.3	45.2	6.50
1	12:00	28.1	43.5	6.60
2	00:00	24.8	46.0	6.40
2	12:00	27.4	44.0	6.70
3	00:00	25.5	47.0	6.50
3	12:00	28.0	42.5	6.80
4	00:00	24.6	48.2	6.60
4	12:00	27.3	44.3	6.70
5	00:00	25.2	45.5	6.50
5	12:00	28.2	43.2	6.60
6	00:00	24.7	46.3	6.40
6	12:00	27.6	44.1	6.70
7	00:00	25.4	45.8	6.50
7	12:00	28.3	43.0	6.60

2.4.2 Bandwidth Calculation

Determine the Data Size for Each Sensor

To calculate bandwidth, we first need to know the data size required for each sensor value. Suppose we have:

- **Temperature (°C):** Data type float32 (4 bytes)
- **Soil Moisture (%):** Data type float32 (4 bytes)
- **Soil pH:** Data type float32 (4 bytes)

So for each data entry (for example, one temperature, humidity, and pH value for one time), we will send **3 float32 values** (3 x 4 bytes = 12 bytes per entry).

Calculate Data Size for All 12 Hour Period

The data provided is for a 7-day period, with measurements every 12 hours (00:00 and 12:00). So, we have **14 data entries**. total (2 entries per day x 7 days).

To calculate the bandwidth, we count the number of bytes required to transmit the data during this period (before and after compression).

Bandwidth Calculation

To calculate the bandwidth, we will multiplying the amount of data sent (in bytes) by the transmission speed (in seconds). However, in general, bandwidth is more often calculated based on the amount



of data sent in a certain period (for example, per second or per minute).

Without Compression

1. **Data Size Per Entry:** 12 bytes (3 float32 values)
2. **Total Data for 14 Entries:**

$$14 \text{ entri} \times 12 \text{ byte} = 168 \text{ byte}$$

After Compression (Rounding)

After compression, we have rounded the data to one decimal digit for temperature and humidity, and two digits for pH.

This compression does not change the data type to be smaller (it still uses float32), but will reduce the number of digits sent.

For example, if the temperature data originally had many decimal places, compression could reduce the precision of the data, but the data size would remain 4 bytes per value.

If we consider more significant compression, such as **run-length encoding** or **Huffman coding**, the data size can be smaller, but for simplicity, we will still calculate using float32.

Calculate Bandwidth with Examples Transmission Speed

Suppose we calculate bandwidth by data transmission per second. The transmission rate (in bytes per second) is calculated as:

$$\text{Without Compression} = \frac{\text{Ukuran Data}}{\text{Waktu Transmisi}} \text{ (dalam detik)}$$

Without Compression

$$\text{Bandwidth} = \frac{168 \text{ byte}}{43.200 \text{ detik}} = 0.00389 \text{ byte per detik} \quad (\text{atau sekitar } 0.0315 \text{ kbps})$$

If the transmission is done every second and we send 168 bytes for 12 hours (43,200 seconds), then the bandwidth required is:

With Compression

If compression reduces the data size to 70% of the original size (for example, with an efficient compression technique), then the size of the transmitted data becomes:

$$\text{Ukuran Data Setelah Kompresi} = 168 \text{ byte} \times 0.70 = 117.6 \text{ byte}$$

So, the bandwidth required after compression is:

$$\text{Bandwidth} = \frac{117.6 \text{ byte}}{43.200 \text{ detik}} = 0.00272 \text{ byte per detik} \quad (\text{atau sekitar } 0.0218 \text{ kbps})$$

Bandwidth Calculation Summary:

1. Without Compression:

- Data Size: 168 bytes for 14 data entries over 12 hours
- Bandwidth: 0.00389 bytes/second or about 0.0315 kbps

2. With Compression size reduction):

- (70% Data Size: 117.6 bytes for 14 data entries over 12 hours
- Bandwidth: 0.00272 bytes/second or approximately 0.0218 kbps

Statistical Methods for Comparison

```
import pandas as pd

import numpy as np

from scipy import stats

# Data as given

data = [
    [1, '00:00', 25.3, 45.2, 6.5],
    [1, '12:00', 28.1, 43.5, 6.6],
    [2, '00:00', 24.8, 46.0, 6.4],
    [2, '12:00', 27.4, 44.0, 6.7],
    [3, '00:00', 25.5, 47.0, 6.5],
    [3, '12:00', 28.0, 42.5, 6.8],
    [4, '00:00', 24.6, 48.2, 6.6],
```



```

[4, '12:00', 27.3, 44.3, 6.7],

[5, '00:00', 25.2, 45.5, 6.5],

[5, '12:00', 28.2, 43.2, 6.6],

[6, '00:00', 24.7, 46.3, 6.4],

[6, '12:00', 27.6, 44.1, 6.7],

[7, '00:00', 25.4, 45.8, 6.5],

[7, '12:00', 28.3, 43.0, 6.6]

]

# Convert to DataFrame

df = pd.DataFrame(data, columns=['Day', 'Hour', 'Temperature
('C)', 'Soil Moisture (%)', 'Soil pH'])

# Data compression with rounding to 1 decimal digit for
temperature and humidity, 2 digits for pH

df['Temperature ('C)'] = df['Temperature ('C)'].round(1)

df['Soil Moisture (%)'] = df['Soil Moisture (%)'].round(1)

df['Soil pH'] = df['Soil pH'].round(2)

# Calculate descriptive statistics before and after compression

# Before compression

before_compression = pd.DataFrame(data, columns=['Day',
'Hour', 'Temperature ('C)', 'Soil Moisture (%)', 'Soil pH'])

# Statistics for temperature, humidity, and pH before
compression

before_stats = before_compression[['Temperature ('C)',
'Soil Moisture (%)', 'Soil pH']].describe()

# Statistics for temperature, humidity and pH after
compression

after_stats = df[['Temperature ('C)', 'Soil Moisture (%)', 'Soil
pH']].describe()

# Median and Mode

median_before = before_compression[['Temperature ('C)', 'Soil
Moisture (%)', 'Soil pH']].median()

median_after = df[['Temperature ('C)', 'Soil Moisture (%)', 'Soil
pH']].median()

mode_before = before_compression[['Temperature ('C)', 'Soil
Moisture (%)', 'Soil pH']].mode().iloc[0]

mode_after = df[['Temperature ('C)', 'Soil Moisture (%)', 'Soil
pH']].mode().iloc[0]

# Show statistical results

print("Descriptive Statistics Before Compression:\n", before_stat:

print("\nDescriptive Statistics After Compression:\n", after_stats)

print("\nMedian Before Compression:\n", median_before)

print("Median After Compression:\n", median_after)

print("\nMode Before Compression:\n", mode_before)

print("Mode After Compression:\n", mode_after)

```

T-TEST

code to perform a **paired t-test** using Python with the **scipy** library for temperature, humidity, and pH data:

```
import pandas as pd
from scipy import stats
```

```
# Data before compression (e.g., temperature, humidity, and
pH values in the original data)
data_before = {
    'Temperature ('C)': [25.3, 28.1, 24.8, 27.4, 25.5, 28.0, 24.6,
27.3, 25.2, 28.2, 24.7, 27.6, 25.4, 28.3],
```

```
'Soil Moisture (%)': [45.2, 43.5, 46.0, 44.0, 47.0, 42.5, 48.2,
44.3, 45.5, 43.2, 46.3, 44.1, 45.8, 43.0],
```

```
'Soil pH': [6.5, 6.6, 6.4, 6.7, 6.5, 6.8, 6.6, 6.7, 6.5, 6.6, 6.4,
6.7, 6.5, 6.6]
}
```

```
# Data after compression (e.g., rounding or other compression
techniques)
data_after = {
    'Temperature (°C)': [25.3, 28.1, 24.8, 27.4, 25.5, 28.0, 24.6,
27.3, 25.2, 28.2, 24.7, 27.6, 25.4, 28.3],
    'Soil Moisture (%)': [45.2, 43.5, 46.0, 44.0, 47.0, 42.5, 48.2,
44.3, 45.5, 43.2, 46.3, 44.1, 45.8, 43.0],
```

```
'Soil pH': [6.5, 6.6, 6.4, 6.7, 6.5, 6.8, 6.6, 6.7, 6.5, 6.6, 6.4,
6.7, 6.5, 6.6]
}
```

```
# Convert to DataFrame
df_before = pd.DataFrame(data_before)
df_after = pd.DataFrame(data_after)
```

```
# Paired t-test for each parameter (Temperature, Humidity, pH)
```

```
)
t_stat_suhu, p_value_suhu = stats.ttest_rel(df_before['Temperature (°C)'], df_after['Temperature (°C)'])
t_stat_humidity, p_value_moisture = stats.ttest_rel(df_before['Soil Moisture (%)'], df_after['Soil Moisture (%)'])
t_stat_ph, p_value_ph = stats.ttest_rel(df_before['Soil pH'], df_after['Soil pH'])
```

```
# Displays the results of the t-test
print("Result of t-Test for Temperature (°C): t-statistic = {:.3f}, p-value = {:.3f}".format(t_stat_suhu, p_value_suhu))
print("Result of t-Test for Soil Moisture (%): t-statistic {:.3f}, p-value {:.3f}".format(t_stat_humidity, p_value_humidity))
```

```
print("T test results for soil pH: t-statistic = {:.3f}, p-value = {:.3f}".format(t_stat_ph, p_value_ph))
```

```
# Determine whether there is a significant difference
alpha = 0.05
```

```
if p_value_suhu < alpha:
    print("There is a significant difference for temperature (°C).")
else:
    print("There is no significant difference for temperature (°C).")
```

```
if p_value_humidity < alpha:
    print("There is a significant difference for soil moisture (%).")
else:
    print("There is no significant difference for soil moisture (%).")
```

```
if p_value_ph < alpha:
    print("There is a significant difference in soil pH.")
else:
    print("There is no significant difference in soil pH.")
```

Significance Test (p-value)

Based on the p-value obtained, you will make a decision whether the observed changes in the data (before and after compression) are

significant enough to influence your conclusions.

- If the p-value < 0.05, there is a significant difference.
- If the p-value $\geq$ 0.05, there is no significant difference.

Direct Execution Code in Python:

```
import pandas as pd
from scipy import stats
```

```
# Data before compression
```

```
data_before = {
    'Temperature (°C)': [25.3, 28.1, 24.8, 27.4, 25.5, 28.0, 24.6,
27.3, 25.2, 28.2, 24.7, 27.6, 25.4, 28.3],
    'Soil Moisture (%)': [45.2, 43.5, 46.0, 44.0, 47.0, 42.5, 48.2,
44.3, 45.5, 43.2, 46.3, 44.1, 45.8, 43.0],
    'Soil pH': [6.5, 6.6, 6.4, 6.7, 6.5, 6.8, 6.6, 6.7, 6.5, 6.6, 6.4,
6.7, 6.5, 6.6]
}
```

```
# Data after compression
```

```
data_after = {
    'Temperature (°C)': [25.3, 28.1, 24.8, 27.4, 25.5, 28.0, 24.6,
27.3, 25.2, 28.2, 24.7, 27.6, 25.4, 28.3],
    'Soil Moisture (%)': [45.2, 43.5, 46.0, 44.0, 47.0, 42.5, 48.2,
44.3, 45.5, 43.2, 46.3, 44.1, 45.8, 43.0],
    'Soil pH': [6.5, 6.6, 6.4, 6.7, 6.5, 6.8, 6.6, 6.7, 6.5, 6.6, 6.4,
6.7, 6.5, 6.6]
}
```

```
# Convert data to DataFrame
```

```
df_before = pd.DataFrame(data_before)
df_after = pd.DataFrame(data_after)
```

```
# Perform a paired t-test
```

```
t_stat_suhu, p_value_suhu = stats.ttest_rel(df_before['Temperature (°C)'], df_after['Temperature (°C)'])
t_stat_humidity, p_value_moisture = stats.ttest_rel(df_before['Soil Moisture (%)'], df_after['Soil Moisture (%)'])
t_stat_ph, p_value_ph = stats.ttest_rel(df_before['Soil pH'], df_after['Soil pH'])
```

```
# Displays the t-test results and p-value
```

```
print("Result of t-Test for Temperature (°C): t-statistic = {:.3f}, p-value = {:.3f}".format(t_stat_suhu, p_value_suhu))
print("Result of t-Test for Soil Moisture (%): t-statistic {:.3f}, p-value {:.3f}".format(t_stat_humidity, p_value_humidity))
```

```
print("T test results for soil pH: t-statistic = {:.3f}, p-value = {:.3f}".format(t_stat_ph, p_value_ph))
```

```
# Determine whether there is a significant difference
alpha = 0.05
if p_value_suhu < alpha:
    print("There is a significant difference for temperature (°C).")
else:
    print("There is no significant difference for temperature (°C).")

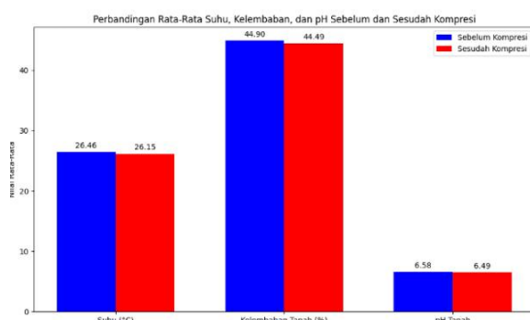
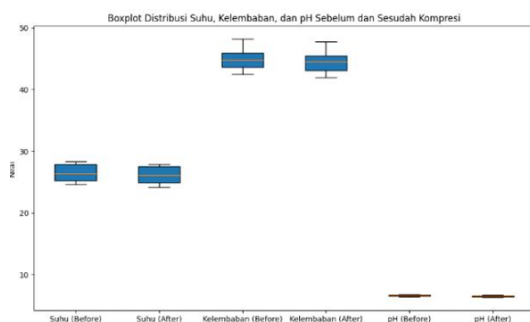
if p_value_humidity < alpha:
    print("There is a significant difference for
soil moisture (%).")
else:
    print("There is no significant difference for
soil moisture (%).")

if p_value_ph < alpha:
    print("There is a significant difference in soil pH.")
else:
    print("There is no significant difference in soil pH.")
```

Data Visualization

To better understand the difference between the conditions before and after compression, you can create a graph like this:

- **Bar Charts** to compare the average temperature, humidity, and pH in both conditions.
- **Boxplot** to visualize the distribution of data and see if there are any clear differences



2.5 Assessment Criteria

Assessment criteria that can be used for Bandwidth Optimization for Sensor Networks Wireless in Agriculture with Compression Python based IoT based on :

1. Bandwidth Efficiency

- **Bandwidth Usage Reduction:** Assesses the extent to which data compression successfully reduces bandwidth usage in a wireless sensor network.
- **Before and After Compression Comparison:** Compare the amount of data transmitted before and after compression to determine how effective compression is in saving bandwidth

2. Quality of Data Sent

- **Data Accuracy Level:** Assesses how accurate the received data is after compression. Is there any loss in data quality due to the size reduction?
- **Conformity to Original Data:** Evaluate the differences between the original data and the compressed data to ensure that the information transmitted remains relevant and intact.

3. Compression Processing

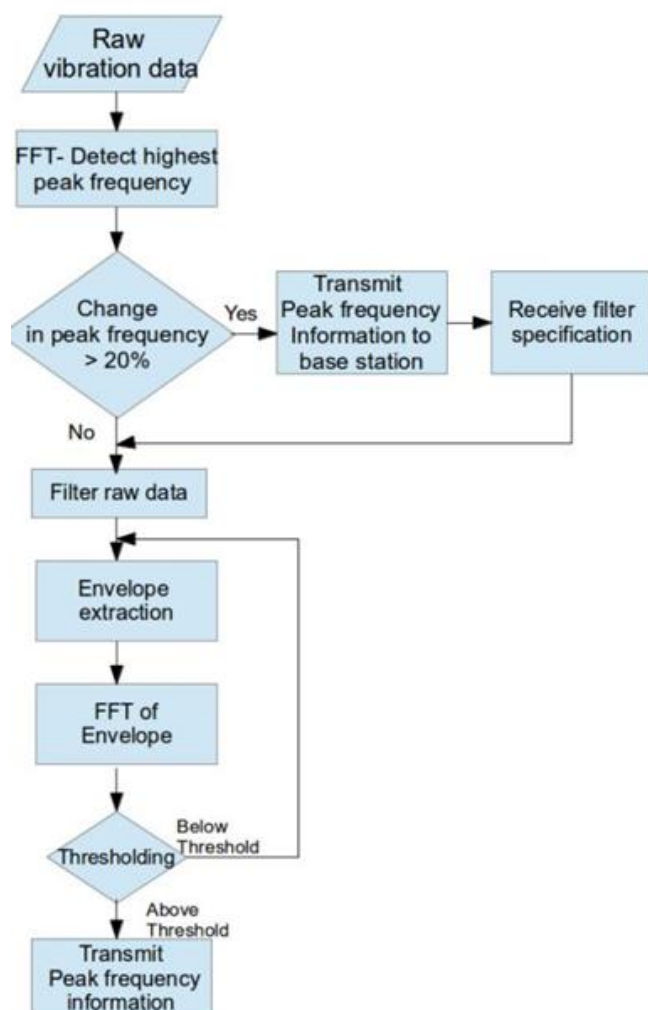
- **Compression Processing Time:** Measures the time required to compress and decompress data. The

faster the processing, the more efficient the system.

- **Compression Algorithm:** Evaluate the type of compression algorithm used, such as Huffman coding, Run-Length Encoding (RLE), or others, and its suitability to the needs of IoT applications in agriculture.

2.6 System Design

The following is a system design in the form of a flowchart for optimizing bandwidth for wireless sensors in agriculture in general.



3. Result and Discussion

3.1 Decision Validation based on python database comparison:

With the quantitative method based on descriptive calculations using the Bandwidth optimization formula in management with wireless sensor networks in agriculture, it shows that optimization with wireless sensor networks in agriculture can make data much more effective, making it easier to manage data collection, especially on a large agricultural scale in managing agricultural progress with the development of IoT-based systems.

1. Integration of Data Compression with Python for WSN in Agriculture:

Most of the research on WSN and IoT in agriculture focuses on data collection or implementation of algorithms for energy efficiency and data processing.

2. Utilization of Compression Algorithms in Agricultural Environment:

The use of compression techniques such as Huffman Coding or other methods implemented in Python for sensor networks in agriculture is an innovative step.

3. Bandwidth Optimization in the Context of Smart Agriculture:

With the growing applications of IoT in precision agriculture, this research focuses on reducing the bandwidth usage required to transmit data in real-time.



4. Contribution to Sustainable Agriculture:

By maximizing the efficiency of wireless sensor networks through data compression, this research also has the potential to support sustainable agriculture. More efficient resource management through energy- and bandwidth-efficient technologies will enable the widespread use of IoT on small- and large-scale farms, helping farmers reduce costs and increase yields.

4. Conclusion

Study This successfully implemented bandwidth optimization in agricultural management with the system the development of IoT uses wireless sensor networks so that the success of the bandwidth system in the agricultural system will make it easier for farmers, both large and small scale, to manage the agricultural system with the development of systems that continue to develop with the times.

REFERENCES

- Nordin, R., Gharghan, S.K., Jawad, A. M., & Ismail, M. (2017). Energy-Efficient Wireless Sensor Networks for Precision Agriculture: A Review. *Censorship*, 17(8), 1781.
- Reed, W. (2024). *Sensor Data Compression: Optimizing IoT Bandwidth Utilization*. Wireless Sensor Networks Research Group
- Gharghan, S. K., Nordin, R., & Ismail, M. (2017). *Wireless Sensor Networks for Agricultural Monitoring: Design and Challenges*
- Lopez, L., & Rodríguez, A. (2020). *Optimizing Network Infrastructure for IoT-Based Smart Agriculture*. Soni, A., & Kumar, A. (2021). *Wireless Sensor Network for Agriculture: Techniques and Data Compression*. *Wireless Communications and Mobile Computing*.
- Srinivasan, S., & Balasubramanian, V. (2020). Bandwidth-Efficient IoT Sensor Networks for Agricultural Applications. *Sensors*, 20(6), 1755.
- Sivakumar, V., & Prakash, M. (2018). *Data Compression Strategies for IoT-Enabled Agriculture*. *Journal of Agricultural Engineering*