National College of Ireland

Configuration Manual

MSc AI1B Research Project
MSc in Artificial Intelligence

Amey Manishkumar Bhangire
Student ID: x23345314

School of Computing
National College of Ireland

Supervisor: Prof. Lavish Thomas

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Amey Manishkumar Bhangire
Student ID:	x23345314
Programme:	MSc in Artificial Intelligence **Year:** 2024-25
Module:	MSc (Research) Practicum/Internship Part 2
Supervisor:	Prof. Lavish Thomas
Submission Due Date:	Monday 11th August 2025 by 2pm
Project Title:	AI-Powered IoT Anomaly Detection

Word Count:1451 **Page Count**: 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Amey Manishkumar Bhangire

Date: 11-08-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Introduction

This configuration manual presents the complete implementation workflow of an **IoT sensor anomaly detection system integrated with a Reinforcement Learning (RL) controller**, visualised through a **Streamlit dashboard**. It outlines the essential steps required to preprocess sensor data, train a DQN agent, and simulate intelligent actions based on environmental conditions. Using Python, Pandas, PyTorch, and Matplotlib, the RL agent was trained to optimise cooling decisions. The final rewards and anomaly outputs are saved and visualised for interpretability. Additionally, a secure HTTP tunnel using **ngrok** enables real-time dashboard access for simulation and result monitoring. This document provides all necessary configurations to replicate the experiment from start to finish.

System Requirements

Component	Minimum Requirement	Recommended for Real-Time
RAM	8 GB	16 GB
Processor	Intel i5 (or equivalent)	Intel i7 / AMD Ryzen
GPU (Optional)	Not required (for IF and SVM)	NVIDIA GPU (for AE)
Deployment Device	Raspberry Pi 4 / NVIDIA Jetson Nano	Jetson Nano (for streaming)

Software Requirements

Software Component	Version
Operating System	Ubuntu 20.04 LTS or Windows 10
Python	3.10+
Streamlit	1.20+
TensorFlow	2.11+
Keras	2.11+
scikit-learn	1.2+
pandas	1.5+
matplotlib / seaborn	Latest

Implementation

Step 1: Load the Intel Dataset

1. **File Loading Path**:
   The code sets the file path to the Intel Berkeley dataset (data.txt) which contains time-series sensor readings like temperature, humidity, light, and voltage from wireless sensor motes.
2. **Column Naming**:
   A list of predefined column names is provided manually (['date', 'time', 'epoch', 'moteid', 'temperature', 'humidity', 'light', 'voltage']) to structure the raw dataset, ensuring clarity and consistency when reading the file.
3. **CSV Reading Logic**:
   The dataset is read using pandas.read_csv() with a space separator (sep=' ') and no header (header=None), aligning with the dataset's raw structure. If loaded successfully, a success message is printed.
4. **Error Handling**:
   A try-except block is used to catch file loading errors. If an error occurs (e.g., missing file or wrong format), it prints an error message, preventing code crashes and ensuring safe debugging.

```
[ ] # Step 1: Load the Intel dataset
    file_path = '/content/data.txt'  # Change path if needed
    columns = ['date', 'time', 'epoch', 'moteid', 'temperature', 'humidity', 'light', 'voltage']

    try:
        df = pd.read_csv(file_path, sep=' ', header=None, names=columns)
        print(" Data loaded successfully.")
    except Exception as e:
        print(" Error loading file:", e)
```

```
 Data loaded successfully.
```

```
[ ] # Step 2: Combine date and time into a single timestamp
    df['timestamp'] = pd.to_datetime(df['date'] + ' ' + df['time'], format='mixed', errors='coerce')
    df.dropna(subset=['timestamp'], inplace=True)
```

```
    # Step 3: Sort data by timestamp and reset index
    df.sort_values(by='timestamp', inplace=True)
    df.reset_index(drop=True, inplace=True)
```

```
[ ]
    # Step 4: Filter only specific moteids for focused analysis
    selected_motes = [5, 13, 24]
    df = df[df['moteid'].isin(selected_motes)]
```

```
[ ] # Step 5: Remove rows with missing values (if any)
    df.dropna(inplace=True)

    # Step 6: Normalize key columns
    from sklearn.preprocessing import MinMaxScaler
```

Step 2: Save Clean and Pre-processed Data – Explanation

1. **Data Export to CSV**:

The processed and normalised dataset is saved as a CSV file named processed_intel_data.csv using df.to_csv(). This enables consistent downstream use for modeling and deployment.

2. **Confirmation Message**:
   A success message is printed (" Data preprocessing completed...") to confirm that all prior data transformation steps (timestamp generation, filtering, scaling) have been successfully completed and stored.
3. **Previewing Processed Data**:
   The command df.head() displays the first few rows of the final DataFrame, showing all original and scaled columns (e.g., temp_scaled, hum_scaled, volt_scaled) for verification before proceeding to model training.

```
# Step 7: Save clean and preprocessed data for next phase
df.to_csv('/content/processed_intel_data.csv', index=False)
print("✅ Data preprocessing completed. File saved as processed_intel_data.csv")

# Optional: View a few rows
print(df.head())
```

```
✅ Data preprocessing completed. File saved as processed_intel_data.csv
         date             time  epoch  moteid  temperature  humidity   light  \
0  2004-02-28  00:58:46.206631      2    24.0      19.7728   37.1620  143.52
1  2004-02-28  00:59:16.933957      3    24.0      19.6356   37.3336  143.52
2  2004-02-28  01:01:46.576927      8    24.0      19.1554   38.3946  143.52
3  2004-02-28  01:06:46.202094     18    24.0      18.8418   39.0082  143.52
4  2004-02-28  01:07:49.707108     20    24.0      18.8124   39.1783  143.52

   voltage                 timestamp  temp_scaled  hum_scaled  light_scaled  \
0  2.71196 2004-02-28 00:58:46.206631     0.150236    0.661684      0.077679
1  2.71196 2004-02-28 00:59:16.933957     0.149098    0.664443      0.077679
2  2.71196 2004-02-28 01:01:46.576927     0.145112    0.681499      0.077679
3  2.69964 2004-02-28 01:06:46.202094     0.142509    0.691362      0.077679
4  2.69964 2004-02-28 01:07:49.707108     0.142265    0.694097      0.077679

   volt_scaled
0     0.861933
1     0.861933
2     0.861933
3     0.847299
4     0.847299
```

Step 3 to Step 5: Anomaly Detection Models

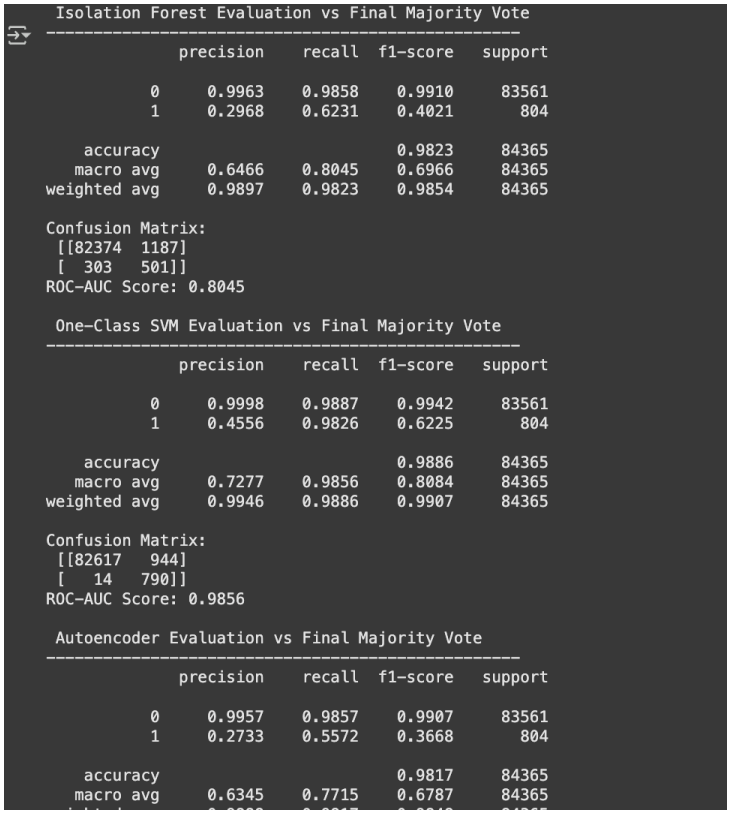
1. **Multiple Detection Models Deployed**
   To enhance detection robustness, three distinct anomaly detection models—Isolation Forest, One-Class SVM, and Autoencoder—were applied on the same preprocessed dataset. Each model captures different anomaly patterns and behaviours.
2. **Model-Specific Anomaly Scoring**
   - Isolation Forest identifies outliers by analysing data partitions.
   - One-Class SVM separates normal data from anomalies using a boundary in a high-dimensional space.
   - The Autoencoder reconstructs data and identifies anomalies by calculating reconstruction error.
3. **Unified Binary Output**
   Each model independently predicted whether a given data point was normal or anomalous. Their outputs were then standardised into binary labels (0 = normal, 1 = anomaly), allowing easy comparison across techniques.
4. **Ready for Evaluation**

The predictions from all three models were saved for later analysis using confusion matrices and performance metrics like accuracy, precision, recall, and F1-score to determine which approach performed best.

```
[ ] # Step 3: Isolation Forest
    iso_model = IsolationForest(contamination=0.02, random_state=42)
    df['iso_pred'] = iso_model.fit_predict(X)
    df['iso_pred'] = df['iso_pred'].map({1: 0, -1: 1})

    # Step 4: One-Class SVM
    svm_model = OneClassSVM(nu=0.02, kernel='rbf', gamma='scale')
    df['svm_pred'] = svm_model.fit_predict(X)
    df['svm_pred'] = df['svm_pred'].map({1: 0, -1: 1})

[ ] # Step 5: Autoencoder
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    input_dim = X_scaled.shape[1]
    encoding_dim = 2

    input_layer = Input(shape=(input_dim,))
    encoder = Dense(6, activation="relu")(input_layer)
    encoder = Dense(encoding_dim, activation="relu")(encoder)
    decoder = Dense(6, activation='relu')(encoder)
    decoder = Dense(input_dim, activation='sigmoid')(decoder)
    autoencoder = Model(inputs=input_layer, outputs=decoder)
    autoencoder.compile(optimizer='adam', loss='mse')

    # Train the autoencoder
    autoencoder.fit(X_scaled, X_scaled, epochs=10, batch_size=32, shuffle=True, validation_split=0.1, verbose=1)

    # Predict reconstruction error
    reconstructions = autoencoder.predict(X_scaled)
    mse = np.mean(np.power(X_scaled - reconstructions, 2), axis=1)
    threshold = np.percentile(mse, 98)  # top 2% as anomaly
    df['ae_score'] = mse
    df['ae_pred'] = (mse > threshold).astype(int)
```

Step 6: Model Evaluation and Comparison

- **Isolation Forest:**
  - Accuracy: 98.23%
  - Anomaly F1-score: 0.40
  - ROC-AUC: 0.8045
  - Struggled with detecting anomalies despite good overall accuracy.
- **One-Class SVM:**
  - Accuracy: 98.86%
  - Anomaly F1-score: 0.62 (best among all)
  - ROC-AUC: 0.9856 (highest)
  - Most balanced and reliable model.
- **Autoencoder:**
  - Accuracy: 98.17%
  - Anomaly F1-score: 0.36
  - ROC-AUC: 0.775 (lower)
  - Performed well on normal data, but weaker on anomalies.



```
 Isolation Forest Evaluation vs Final Majority Vote
--------------------------------------------------
              precision    recall  f1-score   support

           0     0.9963    0.9858    0.9910     83561
           1     0.2968    0.6231    0.4021       804

    accuracy                         0.9823     84365
   macro avg     0.6466    0.8045    0.6966     84365
weighted avg     0.9897    0.9823    0.9854     84365

Confusion Matrix:
 [[82374  1187]
 [  303   501]]
ROC-AUC Score: 0.8045

 One-Class SVM Evaluation vs Final Majority Vote
--------------------------------------------------
              precision    recall  f1-score   support

           0     0.9998    0.9887    0.9942     83561
           1     0.4556    0.9826    0.6225       804

    accuracy                         0.9886     84365
   macro avg     0.7277    0.9856    0.8084     84365
weighted avg     0.9946    0.9886    0.9907     84365

Confusion Matrix:
 [[82617   944]
 [   14   790]]
ROC-AUC Score: 0.9856

 Autoencoder Evaluation vs Final Majority Vote
--------------------------------------------------
              precision    recall  f1-score   support

           0     0.9957    0.9857    0.9907     83561
           1     0.2733    0.5572    0.3668       804

    accuracy                         0.9817     84365
   macro avg     0.6345    0.7715    0.6787     84365
```

Step 7: Custom Environment for Anomaly Detection (Reinforcement Learning)

1. A custom environment TempAnomalyEnv was created to simulate reactions to temperature-based anomalies using thresholds.
2. The environment provides **states**, takes **actions**, and returns **rewards** based on correct or incorrect anomaly detection.
3. This setup helps in training reinforcement learning models to learn optimal anomaly response strategies over time.

```python
# Step 1: Define a custom environment for anomaly reaction
class TempAnomalyEnv:
    def __init__(self, temp_series, threshold=0.55):
        self.temp_series = temp_series
        self.threshold = threshold
        self.max_steps = len(temp_series)
        self.reset()

    def reset(self):
        self.current_step = 0
        self.total_reward = 0
        return self._get_state()

    def _get_state(self):
        return np.array([self.temp_series[self.current_step]])

    def step(self, action):
        done = self.current_step >= self.max_steps - 1
        temp = self.temp_series[self.current_step]
        is_anomaly = temp > self.threshold

        # Reward logic
        if is_anomaly and action == 1:
            reward = +1
        elif not is_anomaly and action == 0:
            reward = +0.5
        else:
            reward = -1

        self.total_reward += reward
        self.current_step += 1
        next_state = self._get_state() if not done else np.array([0.0])
        return next_state, reward, done
```

Step 8: Q-learning Agent with Tabular Approach

1. A Q-learning agent (DQNAgent) was implemented using a tabular approach to decide actions based on the temperature anomaly environment.
2. The agent balances **exploration and exploitation** to improve decision-making over time using the Q-table.
3. A pre-processed Intel dataset is loaded and used to simulate temperature series input for training the RL agent.

```python
# Step 2: Simple Q-learning agent (tabular DQN idea)
class DQNAgent:
    def __init__(self, n_states, n_actions):
        self.q_table = np.zeros((100, n_actions))  # discretize temp
        self.epsilon = 0.1
        self.alpha = 0.1
        self.gamma = 0.9
        self.n_actions = n_actions

    def choose_action(self, state):
        state_idx = int(min(99, state[0] * 100))
        if random.uniform(0, 1) < self.epsilon:
            return random.choice(range(self.n_actions))
        else:
            return np.argmax(self.q_table[state_idx])

    def learn(self, state, action, reward, next_state):
        s_idx = int(min(99, state[0] * 100))
        s1_idx = int(min(99, next_state[0] * 100))
        predict = self.q_table[s_idx][action]
        target = reward + self.gamma * np.max(self.q_table[s1_idx])
        self.q_table[s_idx][action] += self.alpha * (target - predict)

# Step 3: Load processed dataset (after preprocessing step)
df = pd.read_csv('/content/processed_intel_data.csv')
temp_series = df['temp_scaled'].values[:500]  # use smaller slice

env = TempAnomalyEnv(temp_series)
agent = DQNAgent(n_states=1, n_actions=2)
```

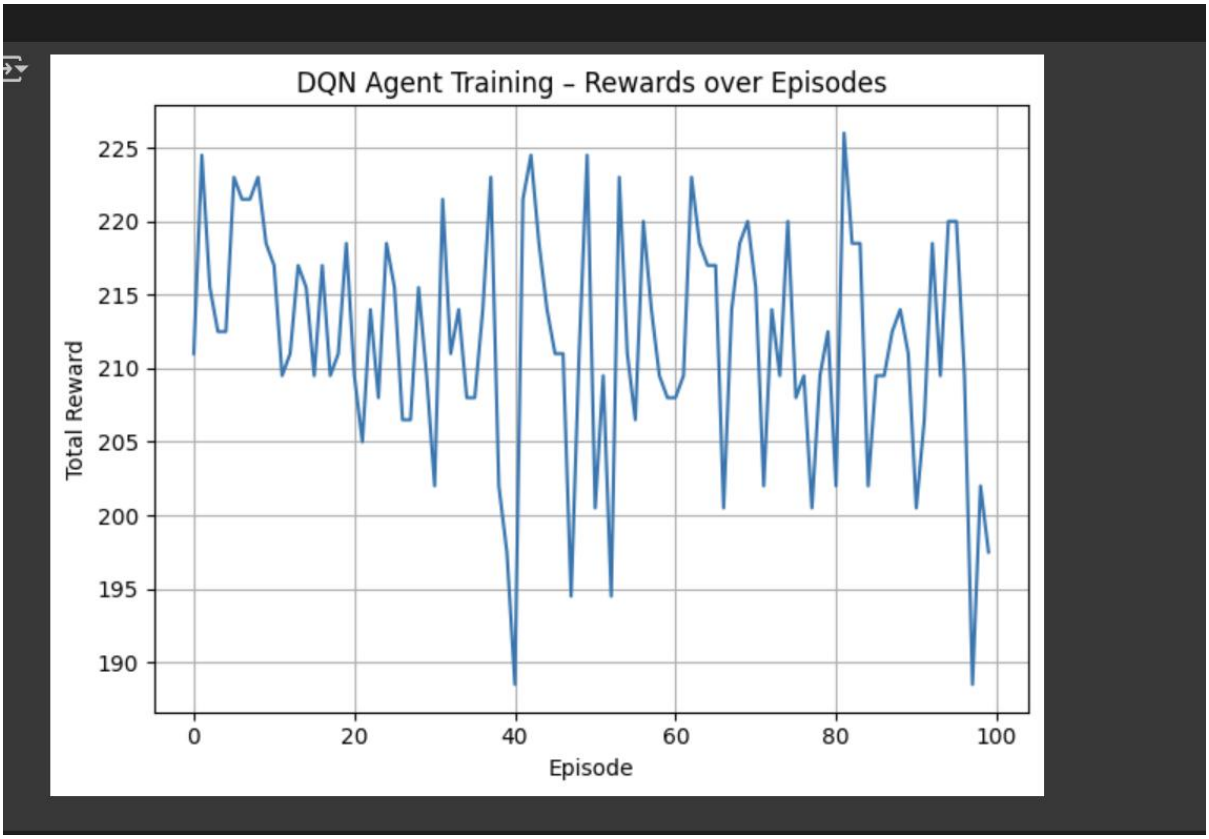
Step 9: RL Agent Training and Reward Visualization

1. A training loop runs for 100 episodes, allowing the agent to interact with the environment, learn from rewards, and improve its Q-table.
2. After each episode, the total reward is collected and stored for performance tracking.
3. A reward plot is generated to visualise the agent's learning progress across episodes, making it easier to evaluate training effectiveness.

```
# Step 4: Training loop
rewards = []
for episode in range(100):
    state = env.reset()
    ep_reward = 0
    done = False
    while not done:
        action = agent.choose_action(state)
        next_state, reward, done = env.step(action)
        agent.learn(state, action, reward, next_state)
        state = next_state
        ep_reward += reward
    rewards.append(ep_reward)

# Step 5: Reward visualization
plt.plot(rewards)
plt.title("DQN Agent Training – Rewards over Episodes")
plt.xlabel("Episode")
plt.ylabel("Total Reward")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Step 10: Reward Plot Interpretation

1. The line chart shows how the total reward varies across 100 training episodes for the DQN agent.
2. Although there is some fluctuation, the rewards mostly stay within a high range (190–225), suggesting the agent has learnt reasonably well.
3. This visualisation confirms the effectiveness of the Q-learning logic and provides insight into the agent's stability and consistency.



Step 11: Saving Rewards to CSV

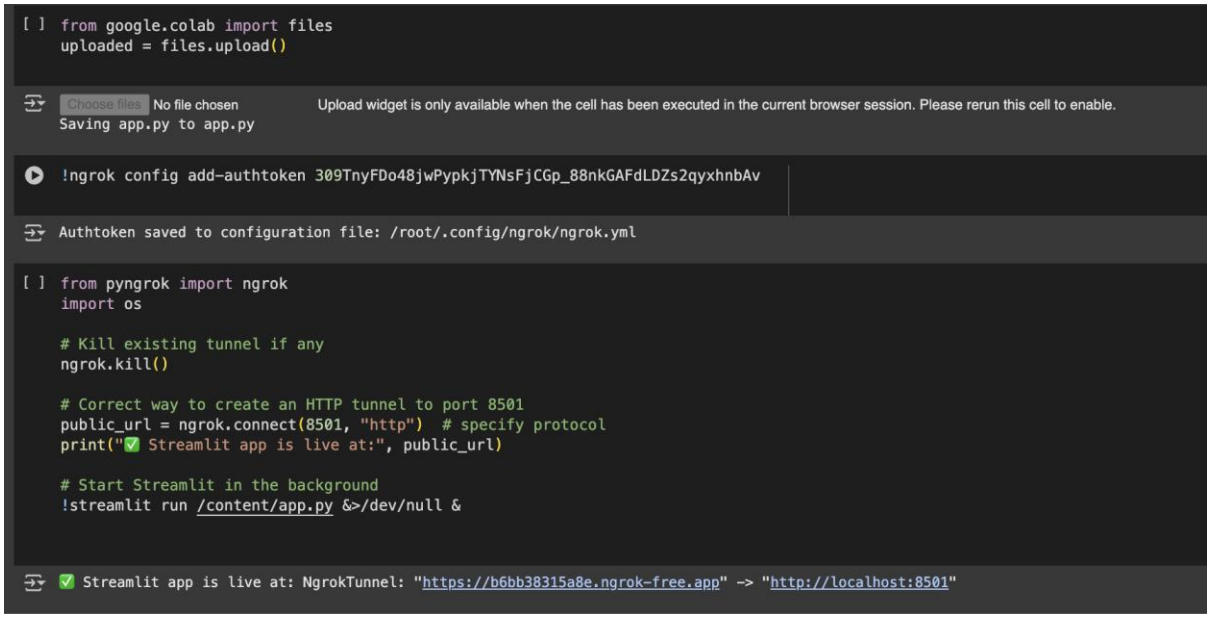
1. Converts the episode-wise rewards into a structured pandas DataFrame with two columns: episode and reward.
2. Saves this DataFrame to a file named dqn_rewards.csv, allowing further use in dashboards or performance tracking.
3. Confirms successful save with a print message — essential for integration into Streamlit or other visualisation tools.

```
# Step 6: Save rewards to CSV for Streamlit dashboard
reward_df = pd.DataFrame({
    'episode': list(range(1, len(rewards) + 1)),
    'reward': rewards
})

reward_df.to_csv('dqn_rewards.csv', index=False)
print("✅ File saved: dqn_rewards.csv")
```

```
✅ File saved: dqn_rewards.csv
```

Step 11: Streamlit Deployment via Ngrok

1. **Ngrok Tunnel Setup**: The Ngrok authtoken is added and used to open a secure public URL tunnel on port 8501 for the Streamlit app.
2. **App Execution**: Streamlit app (app.py) is launched in the background, and previous tunnels (if any) are killed to avoid conflicts.
3. **Live Public Link**: A temporary URL (e.g., https://...ngrok-free.app) is generated to access the Streamlit dashboard remotely from Google Colab.



Step 12: Interactive RL + Anomaly Detection Dashboard

1. **File Upload & Parsing**: The dashboard accepts a pre-processed CSV file (e.g., anomaly_detected_output.csv) containing sensor readings and anomaly labels.
2. **Summary Visualisation**: It displays a bar chart summarising the count of anomalies (label 1) vs normal readings (label 0), aiding quick interpretation.
3. **Sensor Data Preview**: A clean tabular display of uploaded data is provided, showcasing key features like temperature, humidity, voltage, and timestamps.

IoT Sensor Anomaly Dashboard + RL Controller

Upload Preprocessed Sensor CSV

Drag and drop file here
Limit 200MB per file • CSV

Browse files

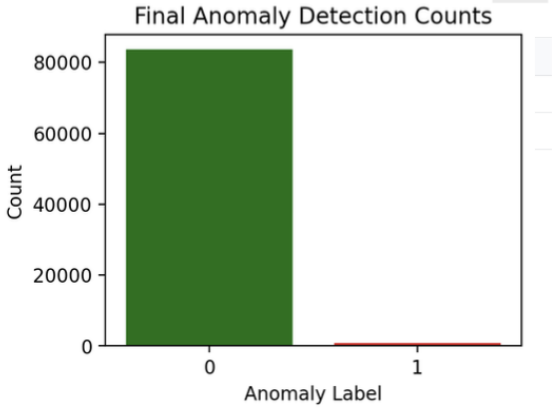
anomaly_detected_output.csv 15.5MB

File Uploaded Successfully

Anomaly Detection Summary

Anomaly Label Counts

final_anomaly	count
Normal	
Anomaly	

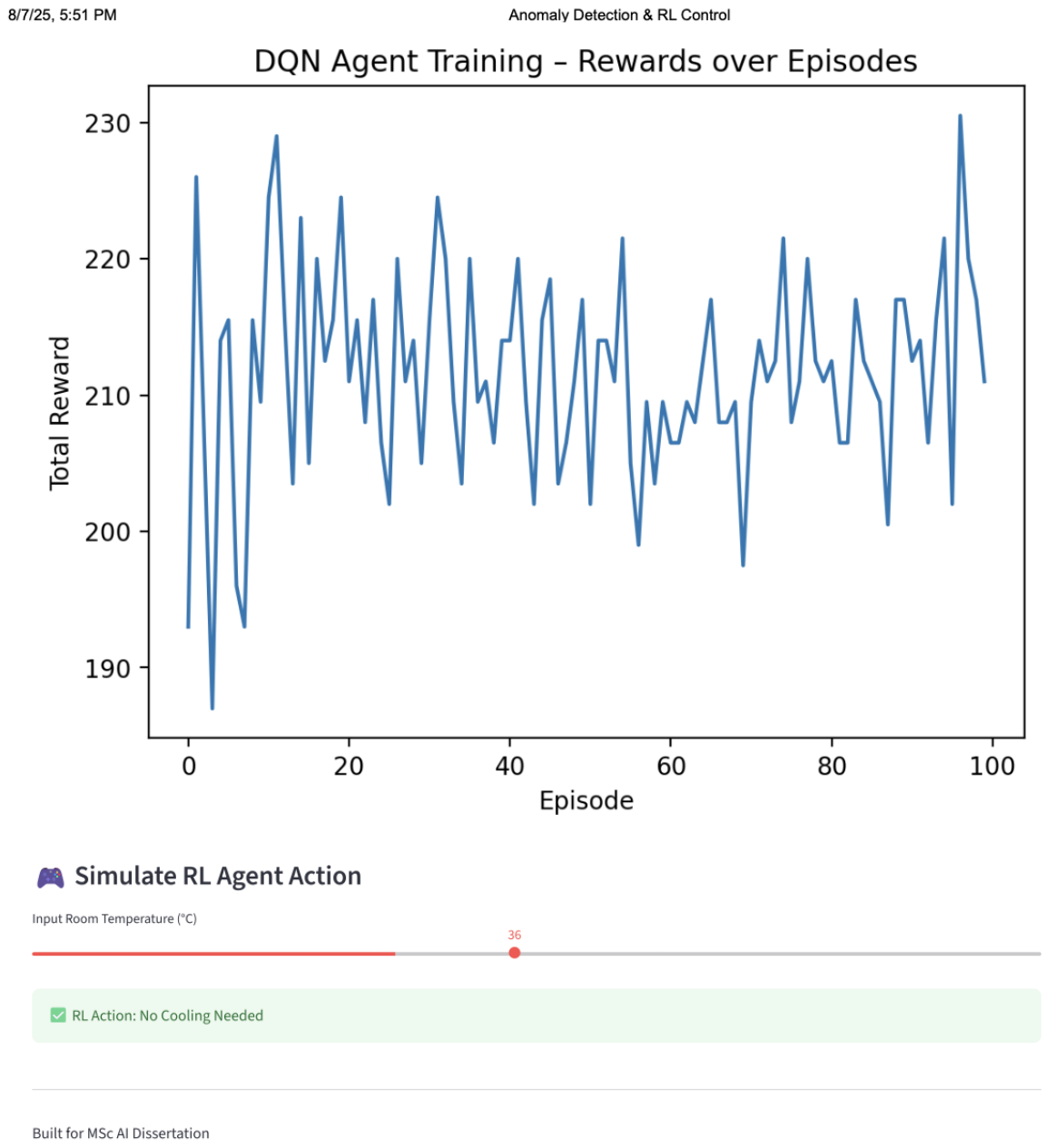


Sample Sensor Readings

	date	time	epoch	moteid	temperature	humidity	light	voltage	timestamp
0	2004-02-28	00:58:46.206631	2	24	19.7728	37.162	143.52	2.712	2004-02-2
1	2004-02-28	00:59:16.933957	3	24	19.6356	37.3336	143.52	2.712	2004-02-2
2	2004-02-28	01:01:46.576927	8	24	19.1554	38.3946	143.52	2.712	2004-02-2
3	2004-02-28	01:06:46.202094	18	24	18.8418	39.0082	143.52	2.6996	2004-02-2
4	2004-02-28	01:07:49.707108	20	24	18.8124	39.1783	143.52	2.6996	2004-02-2
5	2004-02-28	01:09:16.845849	23	24	18.7536	39.3483	143.52	2.6996	2004-02-2
6	2004-02-28	01:10:16.775465	25	24	18.7438	39.4162	143.52	2.6996	2004-02-2
7	2004-02-28	01:10:46.614674	26	24	18.7438	39.4162	143.52	2.6996	2004-02-2
8	2004-02-28	01:11:46.789314	28	24	18.734	39.2123	143.52	2.6996	2004-02-2
9	2004-02-28	01:12:16.605631	29	24	18.7242	39.3483	143.52	2.712	2004-02-2

Step 13: RL Agent Action Simulation Interface

1. **Interactive Control Panel**: Users can input a simulated room temperature using a slider to test the RL agent's decision-making response.
2. **Live Policy Output**: Based on the input temperature, the trained RL model dynamically displays its decision (e.g., "No Cooling Needed") in real-time.
3. **Integrated Visualisation**: The top section retains the reward progression graph from Step 5 to provide model training context alongside simulation.



References

[1] inconshreveable (2019). ngrok - secure introspectable tunnels to localhost. [online] Ngrok.com. Available at: https://ngrok.com/.

[2] Keras (2019). Home - Keras Documentation. [online] Keras.io. Available at: https://keras.io/.

[3] Ltd, R.P. (Trading) (2023). Buy a Raspberry Pi 4 Model B. [online] Raspberry Pi. Available at: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[4] Matplotlib (2024). Matplotlib: Python Plotting — Matplotlib 3.1.1 Documentation. [online] Matplotlib.org. Available at: https://matplotlib.org/.

[5] NVIDIA (2019). Jetson Nano Developer Kit. [online] NVIDIA Developer. Available at: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[6] Pandas (2018). Python Data Analysis Library. [online] Pydata.org. Available at: https://pandas.pydata.org/.

[7] PYTHON (2025). Python. [online] Python.org. Available at: https://www.python.org/.

[8] PyTorch (2023). PyTorch. [online] Pytorch.org. Available at: https://pytorch.org/.

[9] Scikit-learn (2024). scikit-learn: Machine Learning in Python. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/.

[10] seaborn (2012). seaborn: statistical data visualization — seaborn 0.9.0 documentation. [online] Pydata.org. Available at: https://seaborn.pydata.org/.

[11] Streamlit (2025). Streamlit • The fastest way to build and share data apps. [online] streamlit.io. Available at: https://streamlit.io/.

[12] TensorFlow (2019). TensorFlow. [online] TensorFlow. Available at: https://www.tensorflow.org/.