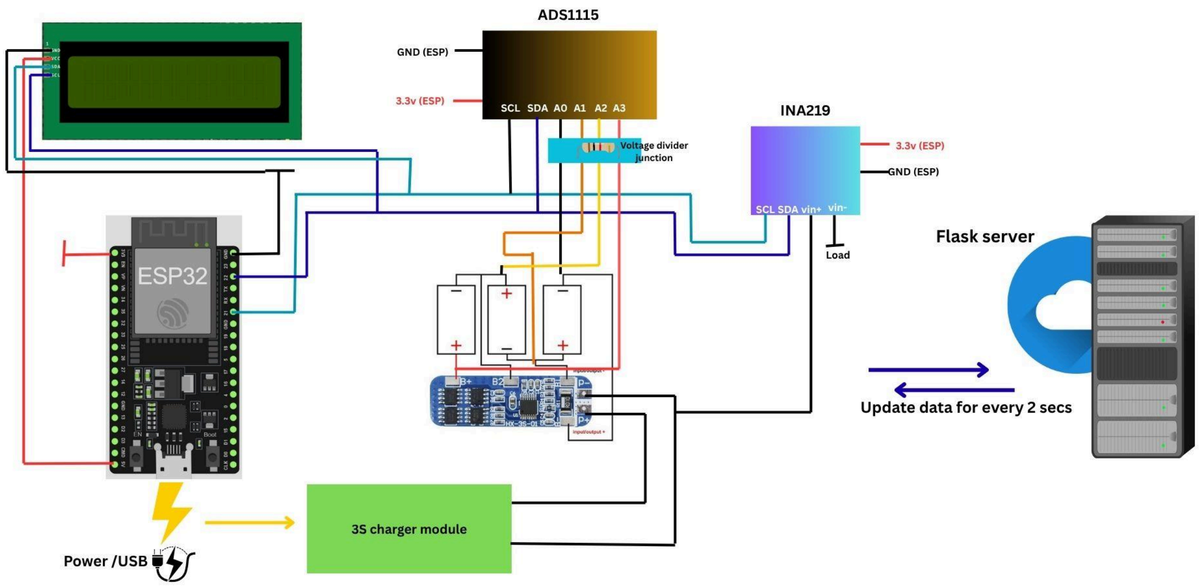
2025 TEEP Progress Report Week-13

Used Wokwi to complete the tasks for Labs 1 to 4

(Before creating the physical project, please use **Wokwi** for online simulation to complete the initial design of the circuit and code. Then, use **Thonny** together with the **ESP32** development board to build the actual project. Please **record a video of the project** in operation, upload it to YouTube, and provide the video link(LAB7) so we can better understand your learning progress.)

Lab1 – On ESP32, complete the following tasks:

This project focuses on optimizing the single-cell battery charging monitoring system completed last week, with the goal of improving the overall stability of measurement and display. Each step of the experimental process must be thoroughly documented, compiled into the project report, and also presented in video format to ensure completeness and traceability of the research process.



Experimental Record

Date: 17 / OCT / 2025

Course/Project Title: Development of IoT Situation Room for Elevator Operation Monitoring and Health Diagnosis

1. Experiment Title

Real-Time Multi-Cell (3S) Li-Ion Battery Monitoring System using ESP32, INA219, ADS1115, LCD, and Flask Web Dashboard

2. Objective

The objective of this experiment is to build a real-time battery monitoring system that:

- Reads battery voltages and current from a 3S Li-Ion pack using ESP32 with MicroPython.
- Calculates approximate battery percentage and smooths readings using an Exponential Moving Average (EMA) for stable visualization.
- Sends live data to a Flask API over Wi-Fi.
- Displays the pack voltage and battery state-of-charge (SoC) on a 16x2 LCD.
- Visualizes live data and trends on a web dashboard with charts.

3. Materials and Equipment

Hardware:

- ESP32 development board
- 3 × 3.7V Li-Ion batteries (connected in series – 3S configuration)
- 3S BMS (Battery Management System) module
- INA219 current and voltage sensor module
- ADS1115 16-bit Analog-to-Digital Converter (ADC) module
- 16x2 I²C LCD module (address 0x27)
- Voltage divider resistors (10kΩ each)
- TP4056 charger modules (for initial setup and testing)
- Jumper wires and breadboard
- USB cable and power supply

Software/Tools:

- Thonny IDE (MicroPython environment for ESP32)
- Flask (Python web framework for dashboard visualization)
- Local Flask server for data handling
- Browser (for web dashboard view)

4. Procedure

Battery Setup:

1. Three Li-Ion cells connected in series to form a 3S pack (≈12.6 V fully charged).
2. BMS module connected to balance and protect the cells during charging and discharging.

Sensor Connections:

1. INA219 placed between the pack's output (BMS P− terminal) and the system ground to measure charging/discharging current.
2. ADS1115 connected via I²C to ESP32 to measure voltages through voltage dividers:

    - A0 → Cell 1 (3.7 V max)
    - A1 → Cell 1 + 2 (≈7.4 V max)
    - A2 → Total pack (≈12.6 V max)

ESP32 and Peripherals:

- ESP32 connected to I²C devices:
    - ADS1115 (SDA = GPIO21, SCL = GPIO22)
    - INA219 (same I²C bus)
    - 16x2 LCD (I²C address = 0x27)
- Wi-Fi credentials configured to enable data transfer to the Flask server.

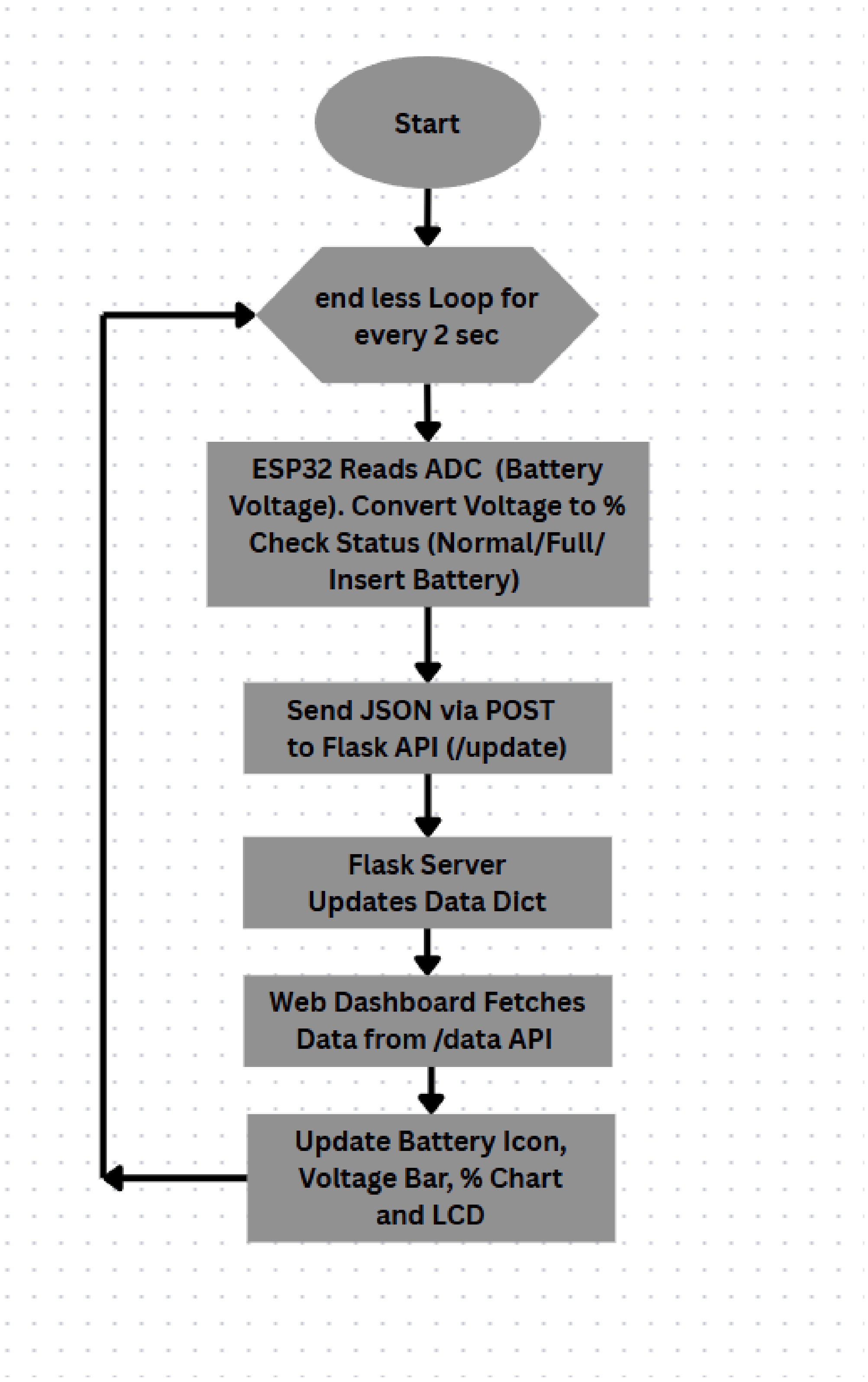
Programming:

- ESP32 programmed using Thonny (MicroPython).
- Code reads:
    - Individual cell voltages from ADS1115
    - Current and pack voltage from INA219
- Microcontroller calculates:
    - Each cell's voltage
    - Total pack voltage
    - Charging/discharging current
    - Battery percentage (SoC) and status
- EMA smoothing applied to reduce fluctuations in voltage and SoC readings.

Data Display and Communication:

- 16x2 LCD displays pack voltage and battery percentage in real time.
- ESP32 uploads the smoothed data to a Flask web server via Wi-Fi for remote monitoring.
- Flask dashboard displays live data and voltage trends graphically, providing stable and readable charts thanks to the smoothing algorithm.

Flowchart :


Start
end less Loop for every 2 sec
ESP32 Reads ADC (Battery Voltage). Convert Voltage to % Check Status (Normal/Full/ Insert Battery)
Send JSON via POST to Flask API (/update)
Flask Server Updates Data Dict
Web Dashboard Fetches Data from /data API
Update Battery Icon, Voltage Bar, % Chart and LCD

6. Source Code :

main.py

```
from machine import Pin, I2C
import network, urequests, time

# ---------------- I2C Setup ----------------
i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=100000)

ADS_ADDR = 0x48   # ADS1115 default I2C
INA_ADDR = 0x40   # INA219 default I2C
LCD_ADDR = 0x27   # typical 16x2 I2C LCD

SSID = "ISILAB CR"
PASSWORD = "isilab.ncut.CR"
URL = "http://192.168.50.205:5000/update"

# ---------------- Wi-Fi Connect ----------------
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect(SSID, PASSWORD)
while not wifi.isconnected():
    time.sleep(0.5)
print("Connected:", wifi.ifconfig())

# ---------------- LCD Driver ----------------
class I2CLcd:
    # Basic 16x2 I2C LCD driver
    def __init__(self, i2c, addr, rows=2, cols=16):
        self.i2c = i2c
        self.addr = addr
        self.rows = rows
        self.cols = cols
        self.backlight = 0x08
        self.init_lcd()

    def init_lcd(self):
        self._write_cmd(0x33)
        self._write_cmd(0x32)
        self._write_cmd(0x06)
        self._write_cmd(0x0C)
        self._write_cmd(0x28)
        self.clear()

    def _write_cmd(self, cmd):
        self.i2c.writeto(self.addr, bytes([cmd & 0xF0 | self.backlight]))
        self.i2c.writeto(self.addr, bytes([(cmd << 4) & 0xF0 | self.backlight]))
        time.sleep_ms(2)

    def _write_char(self, char):
        self.i2c.writeto(self.addr, bytes([char & 0xF0 | 0x01 | self.backlight]))
        self.i2c.writeto(self.addr, bytes([(char << 4) & 0xF0 | 0x01 | self.backlight]))
        time.sleep_ms(1)

    def clear(self):
        self._write_cmd(0x01)
        time.sleep_ms(2)

    def move_to(self, row, col):
        self._write_cmd(0x80 | (col + 0x40 * row))

    def putstr(self, string):
        for c in string:
            self._write_char(ord(c))

lcd = I2CLcd(i2c, LCD_ADDR)

# ---------------- ADS1115 ----------------
ADS_REG_CONVERT = 0x00
ADS_REG_CONFIG = 0x01

def read_ads(channel):
    config = 0x8400 | (channel << 12) | 0x0083
    i2c.writeto_mem(ADS_ADDR, ADS_REG_CONFIG, config.to_bytes(2,'big'))
    time.sleep(0.01)
    raw = i2c.readfrom_mem(ADS_ADDR, ADS_REG_CONVERT, 2)
    val = int.from_bytes(raw,'big')
    if val > 32767:
        val -= 65536
    return val * 4.096 / 32768

def get_cell_voltages():
    v1_raw = read_ads(0) * 2
    v12_raw = read_ads(1) * 3
    v123_raw = read_ads(2) * 4

    c1 = v1_raw
    c2 = v12_raw - c1
    c3 = v123_raw - (c1 + c2)
    pack = v123_raw
    return c1, c2, c3, pack

# ---------------- INA219 ----------------
def read_ina219():
    REG_BUSV = 0x02
    REG_SHUNTV = 0x01
    REG_CAL = 0x05

    i2c.writeto_mem(INA_ADDR, REG_CAL, (4096).to_bytes(2,'big'))

    raw_bus = i2c.readfrom_mem(INA_ADDR, REG_BUSV, 2)
    bus_v = (int.from_bytes(raw_bus,'big') >> 3) * 0.004

    raw_shunt = i2c.readfrom_mem(INA_ADDR, REG_SHUNTV, 2)
    shunt = int.from_bytes(raw_shunt,'big')
    if shunt > 32767: shunt -= 65536
    shunt_v = shunt * 0.01 / 1000
    current = shunt_v / 0.1
    return bus_v, current

# ---------------- EMA Smoothing ----------------
alpha = 0.2
ema_pack = None
ema_c1 = ema_c2 = ema_c3 = None
ema_soc = None
ema_current = None

def ema(prev, new):
    if prev is None:
        return new
    return prev * (1-alpha) + new * alpha

def battery_percentage(voltage):
    vmin, vmax = 9.0, 12.6
    pct = (voltage - vmin)/(vmax-vmin)*100
    return max(0, min(100,int(pct)))

# ---------------- Main Loop ----------------
while True:
    try:
        c1, c2, c3, pack = get_cell_voltages()
        bus_v, current = read_ina219()
        soc = battery_percentage(pack)
        # Apply EMA smoothing
        ema_c1 = ema(ema_c1, c1)
        ema_c2 = ema(ema_c2, c2)
        ema_c3 = ema(ema_c3, c3)
        ema_pack = ema(ema_pack, pack)
        ema_current = ema(ema_current, current)
        ema_soc = ema(ema_soc, soc)

        data = {
            "cell1": round(ema_c1,2),
            "cell2": round(ema_c2,2),
            "cell3": round(ema_c3,2),
            "pack": round(ema_pack,2),
            "current": round(ema_current,2),
            "soc": int(ema_soc)
        }

        print("Sending:", data)
        urequests.post(URL, json=data)

        # LCD Display
        lcd.clear()
        lcd.move_to(0,0)
        lcd.putstr("Pack: {:.2f}V".format(ema_pack))
        lcd.move_to(1,0)
        lcd.putstr("SoC: {}%".format(int(ema_soc)))

        time.sleep(2)

    except Exception as e:
        print("Error:", e)
        time.sleep(2)
```

app.py

```
from flask import Flask, render_template, request, jsonify
from datetime import datetime

app = Flask(__name__)

battery_data = {
    "cell1": 0,
    "cell2": 0,
    "cell3": 0,
    "pack": 0,
    "current": 0,
    "soc": 0,
    "status": "Idle",
    "timestamp": None
}

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/update', methods=['POST'])
def update_data():
    global battery_data
    data = request.get_json()
    if data:
        battery_data.update(data)
        battery_data["timestamp"] = datetime.now().strftime("%H:%M:%S")

        # Determine charge status
        if battery_data["current"] > 0.2:
            battery_data["status"] = "Charging"
        elif battery_data["soc"] >= 98:
            battery_data["status"] = "Full"
        else:
            battery_data["status"] = "Idle"
    return jsonify({"message": "Data received"})

@app.route('/data')
def get_data():
    return jsonify(battery_data)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Index.html

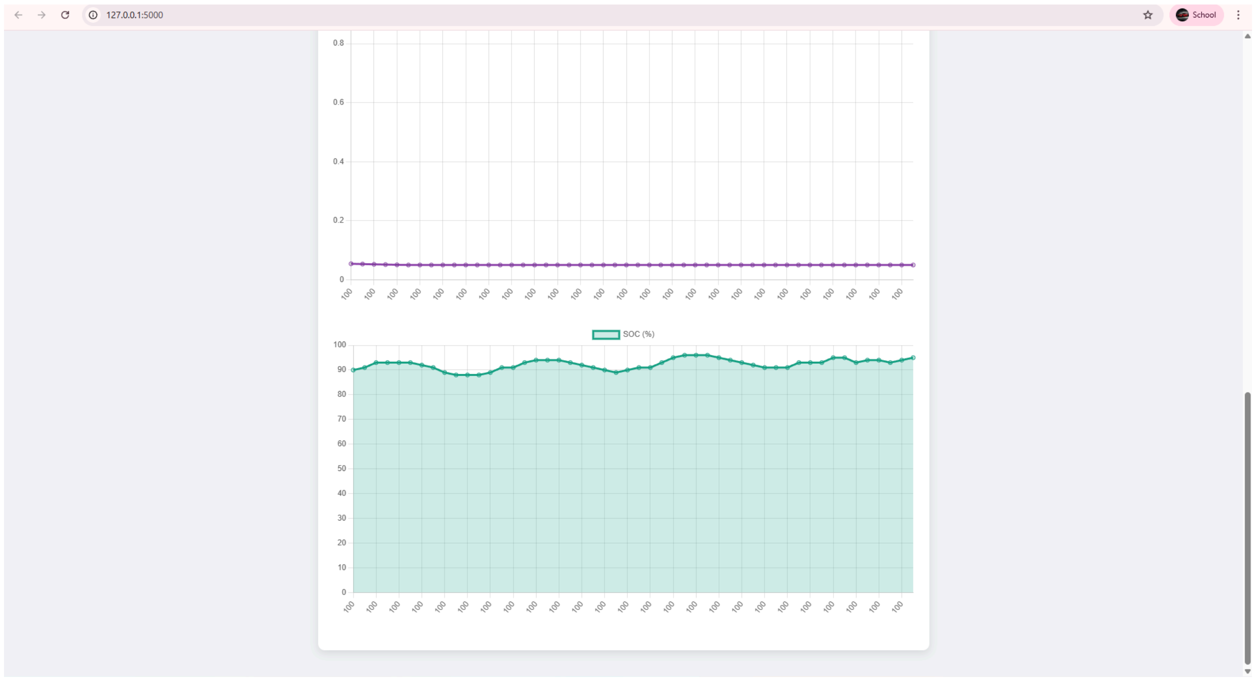
```
<!DOCTYPE html>
<html lang="en">
```

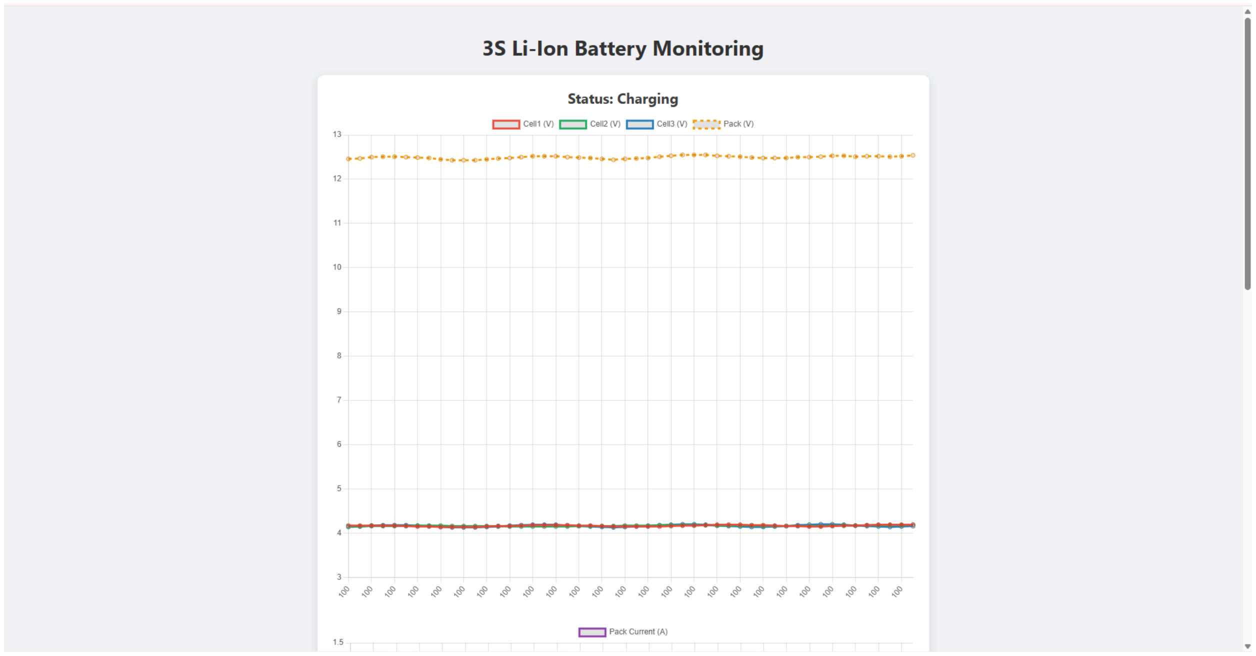
```html
<head>
  <meta charset="UTF-8">
  <title>Smart Battery Dashboard</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body { font-family: "Poppins", sans-serif; background-color: #f3f6fa;
color: #333; text-align: center; margin: 0; padding: 0; }
    h1 { background-color: #2d6cdf; color: white; margin: 0; padding:
20px; font-weight: 500; box-shadow: 0 2px 8px rgba(0,0,0,0.2); }

    .battery-container { display: flex; justify-content: space-around;
flex-wrap: wrap; margin: 20px; }
    .card { background: white; border-radius: 15px; box-shadow: 0 2px 10px
rgba(0,0,0,0.1); padding: 15px; width: 250px; transition: transform 0.2s
ease; margin-bottom: 20px; }
    .card:hover { transform: translateY(-5px); }
    .bar-container { background: #e0e0e0; border-radius: 8px; overflow:
hidden; margin: 10px 0; height: 20px; }
    .bar { height: 100%; width: 0%; background: linear-gradient(90deg,
#47cf73, #2d6cdf); transition: width 0.5s ease; }
    .status { font-size: 18px; font-weight: bold; margin-top: 10px; }
    #chart-container { width: 90%; max-width: 1000px; margin: 30px auto;
background: white; border-radius: 15px; padding: 20px; box-shadow: 0 2px
10px rgba(0,0,0,0.1); }
    .charging { color: #2d6cdf; animation: blink 1s infinite; }
    @keyframes blink { 50% { opacity: 0.5; } }
    .timestamp { font-size: 14px; color: #777; margin-top: 10px; }
  </style>
</head>
<body>
  <h1>⚡ Smart Battery Dashboard</h1>

  <div class="battery-container">
    <div class="card">
      <h3>Cell 1</h3>
      <div class="bar-container"><div id="bar1" class="bar"></div></div>
      <p id="cell1">0.00 V</p>
      <p id="cell1_current">0.00 A</p>
    </div>
    <div class="card">
```

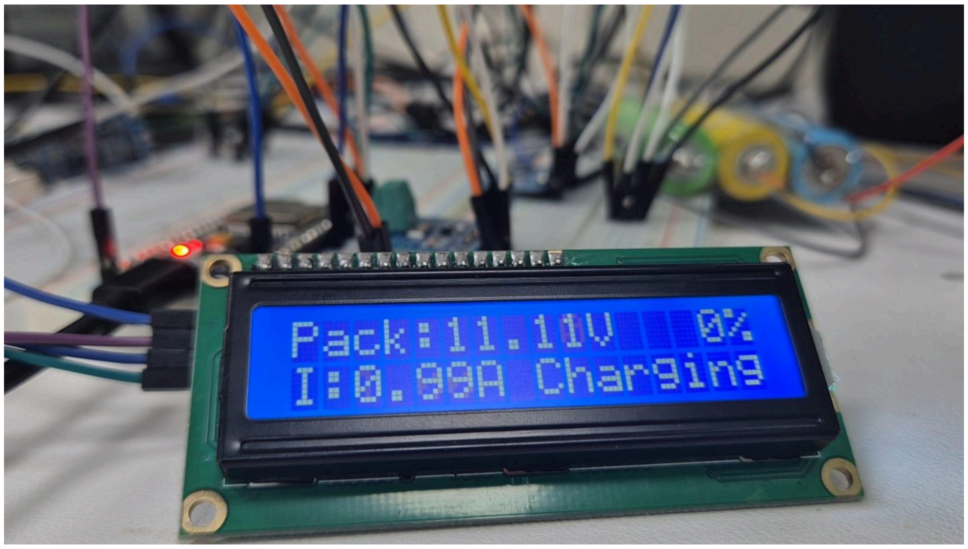
```
      <h3>Cell 2</h3>
      <div class="bar-container"><div id="bar2" class="bar"></div></div>
      <p id="cell2">0.00 V</p>
      <p id="cell2_current">0.00 A</p>
    </div>
    <div class="card">
      <h3>Cell 3</h3>
      <div class="bar-container"><div id="bar3" class="bar"></div></div>
      <p id="cell3">0.00 V</p>
      <p id="cell3_current">0.00 A</p>
    </div>
    <div class="card">
      <h3>Total Pack</h3>
      <div class="bar-container"><div id="barPack"
class="bar"></div></div>
      <p id="pack">0.00 V</p>
      <p id="pack_current">0.00 A</p>
      <div class="status" id="status">Idle</div>
    </div>
  </div>

  <div id="chart-container">
    <canvas id="voltageChart" height="300"></canvas>
    <canvas id="currentChart" height="300" style="margin-top:
40px;"></canvas>
  </div>

  <div class="timestamp" id="timestamp">Updated: --:--:--</div>

  <script>
    // -------------------- CHART SETUP --------------------
    const voltageCtx =
document.getElementById('voltageChart').getContext('2d');
    const currentCtx =
document.getElementById('currentChart').getContext('2d');

    const voltageChart = new Chart(voltageCtx, {
      type: 'line',
      data: {
        labels: [],
        datasets: [
          { label: 'Cell 1 (V)', borderColor: '#2d6cdf', data: [], fill:
false, tension: 0.3 },
          { label: 'Cell 2 (V)', borderColor: '#47cf73', data: [], fill:
false, tension: 0.3 },
          { label: 'Cell 3 (V)', borderColor: '#f1c40f', data: [], fill:
false, tension: 0.3 },
          { label: 'Pack (V)', borderColor: '#e67e22', data: [], fill:
false, tension: 0.3 }
        ]
      },
      options: { responsive: true, animation: false }
    });

    const currentChart = new Chart(currentCtx, {
      type: 'line',
      data: {
        labels: [],
        datasets: [
          { label: 'Cell 1 (A)', borderColor: '#2d6cdf', data: [], fill:
false, tension: 0.3 },
          { label: 'Cell 2 (A)', borderColor: '#47cf73', data: [], fill:
false, tension: 0.3 },
          { label: 'Cell 3 (A)', borderColor: '#f1c40f', data: [], fill:
false, tension: 0.3 },
          { label: 'Pack (A)', borderColor: '#e67e22', data: [], fill:
false, tension: 0.3 }
        ]
      },
      options: { responsive: true, animation: false }
    });

    // -------------------- DUMMY DATA --------------------
    let timeCount = 0;
    function generateDummyData() {
      timeCount++;
      const timestamp = new Date().toLocaleTimeString();

      // Simulate voltages (charging from 3.6V -> 4.2V)
      const cell1 = 3.6 + 0.006 * timeCount + Math.random() * 0.02;
      const cell2 = 3.6 + 0.0055 * timeCount + Math.random() * 0.02;
      const cell3 = 3.6 + 0.0062 * timeCount + Math.random() * 0.02;
      const pack = cell1 + cell2 + cell3;

      // Simulate currents (random charging behavior 0.4A -> 1A)
      const cell1_current = 0.4 + Math.random() * 0.6;
      const cell2_current = 0.4 + Math.random() * 0.6;
      const cell3_current = 0.4 + Math.random() * 0.6;
      const pack_current = cell1_current + cell2_current + cell3_current;

      return {
        cell1, cell2, cell3, pack,
        cell1_current, cell2_current, cell3_current, pack_current,
        soc: Math.min((pack/12.6)*100, 100),
        status: "Charging",
        timestamp
      };
    }

    // -------------------- UPDATE DASHBOARD --------------------
    function updateDashboard() {
      const data = generateDummyData();

      // Update text
      document.getElementById('cell1').innerText = data.cell1.toFixed(3) +
' V';
      document.getElementById('cell2').innerText = data.cell2.toFixed(3) +
' V';
      document.getElementById('cell3').innerText = data.cell3.toFixed(3) +
' V';
      document.getElementById('pack').innerText = data.pack.toFixed(3) + '
V';

      document.getElementById('cell1_current').innerText =
data.cell1_current.toFixed(3) + ' A';
      document.getElementById('cell2_current').innerText =
data.cell2_current.toFixed(3) + ' A';
      document.getElementById('cell3_current').innerText =
data.cell3_current.toFixed(3) + ' A';
```

```
      document.getElementById('pack_current').innerText =
data.pack_current.toFixed(3) + ' A';

      document.getElementById('status').innerText = data.status;
      document.getElementById('status').classList.add('charging');
      document.getElementById('timestamp').innerText = "Updated: " +
data.timestamp;

      // Update bars
      document.getElementById('bar1').style.width = (data.cell1 / 4.2 *
100) + '%';
      document.getElementById('bar2').style.width = (data.cell2 / 4.2 *
100) + '%';
      document.getElementById('bar3').style.width = (data.cell3 / 4.2 *
100) + '%';
      document.getElementById('barPack').style.width = data.soc + '%';
      // Update voltage chart
      voltageChart.data.labels.push(data.timestamp);
      voltageChart.data.datasets[0].data.push(data.cell1);
      voltageChart.data.datasets[1].data.push(data.cell2);
      voltageChart.data.datasets[2].data.push(data.cell3);
      voltageChart.data.datasets[3].data.push(data.pack);
      if (voltageChart.data.labels.length > 30) {
        voltageChart.data.labels.shift();
        voltageChart.data.datasets.forEach(ds => ds.data.shift());
      }
      voltageChart.update('none');
      // Update current chart
      currentChart.data.labels.push(data.timestamp);
      currentChart.data.datasets[0].data.push(data.cell1_current);
      currentChart.data.datasets[1].data.push(data.cell2_current);
      currentChart.data.datasets[2].data.push(data.cell3_current);
      currentChart.data.datasets[3].data.push(data.pack_current);
      if (currentChart.data.labels.length > 30) {
        currentChart.data.labels.shift();
        currentChart.data.datasets.forEach(ds => ds.data.shift());
      }
      currentChart.update('none');
    }
    setInterval(updateDashboard, 1000);
    updateDashboard();
  </script>
</body>
</html>
```

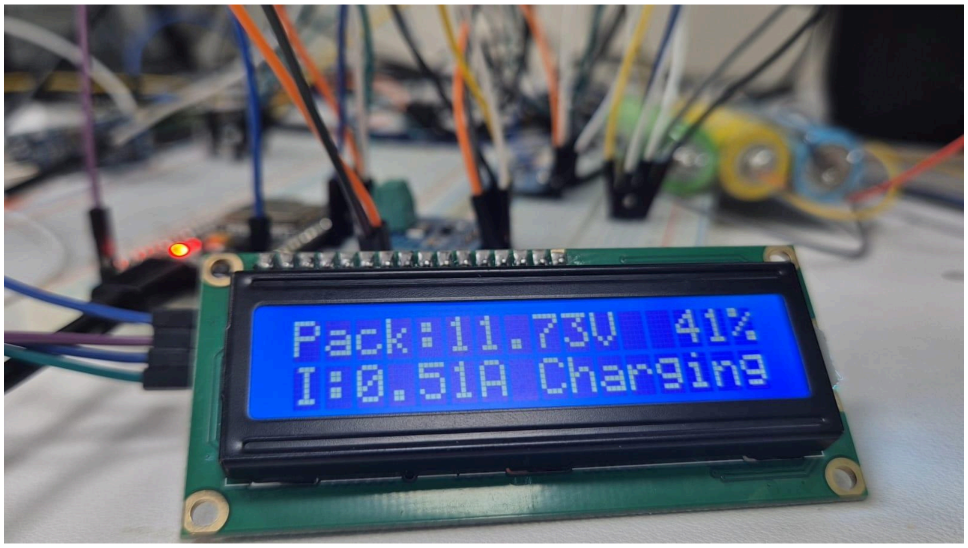
Results :



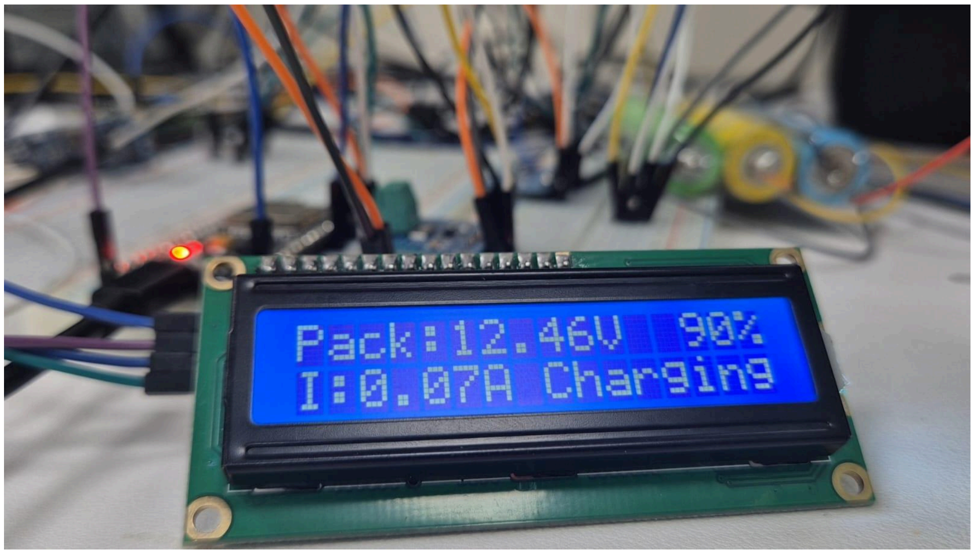




```
Pack:11.11V   0%
I:0.00A Charging
```



```
Pack:11.73V  41%
I:0.51A Charging
```



```
Pack:12.46V  90%
I:0.07A Charging
```

Lab2- Question and Answers (Q&A)

1. How to choose model?

Ans:
Choose a model based on:

- The **type of problem** (regression, classification, clustering).
- **Data characteristics** (size, dimensionality, labeled or unlabeled).
- **Computational resources** (some models need GPUs or long training).
- **Performance trade-offs** (accuracy vs interpretability vs speed).

Choosing the right machine learning model depends on several key factors.
First, consider **the type of problem** you are trying to solve. If it is a **regression** problem (predicting continuous values such as temperature or price), models like Linear Regression, Random Forest Regressor, or Neural Networks may be suitable. For **classification** tasks (predicting categories such as spam or not spam), models like Logistic Regression, Decision Trees, Support Vector Machines, or Deep Learning models can be used. For **clustering** problems, where the data is unlabeled, models like K-Means or DBSCAN are appropriate.

Second, analyze the **characteristics of your data** — including dataset size, dimensionality, and whether it contains labeled examples. For instance, deep learning models require large amounts of data, while smaller datasets may perform better with simpler algorithms like Decision Trees or Naïve Bayes.

Third, take into account **computational resources**. Some models, especially deep neural networks, require powerful hardware such as GPUs and can take a long time to train. Finally, consider **performance trade-offs** such as accuracy versus interpretability versus speed. For example, while neural networks may offer higher accuracy, simpler models like Linear Regression or Decision Trees are more interpretable and easier to deploy.

2. Does it make sense to choose multiple models?

Ans:
Yes, it is often beneficial to choose and test multiple models rather than relying on just one. Different algorithms may perform differently depending on the dataset and problem type. Comparing multiple models allows you to identify which one generalizes best to unseen data and performs most effectively based on your chosen evaluation metrics.

In addition, multiple models can be combined using ensemble techniques to improve performance. Examples include bagging (Bootstrap Aggregating), which reduces variance; boosting, which reduces bias and improves accuracy; and stacking, which combines multiple models to leverage their individual strengths. Ensemble methods like Random Forests and Gradient Boosting are powerful because they integrate multiple learners to achieve better overall predictions, reduce overfitting, and improve model robustness.

3. How to judge the quality of the model?

Ans:
The quality of a machine learning model is judged using evaluation metrics that measure its predictive performance on unseen or validation data. The choice of metric depends on the type of problem.

For classification problems, common metrics include **Accuracy, Precision, Recall** (Sensitivity), **F1-Score, and AUC (Area Under the ROC Curve)**. These metrics help evaluate how well the model classifies positive and negative samples. A Confusion Matrix is often used to visualize these results, showing counts of True Positives, False Positives, True Negatives, and False Negatives.

For regression problems, metrics such as **Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE)** are used. These measure how close the predicted values are to the actual target values. In general, the lower these error values are, the better the model's predictive accuracy.

4. How to calculate various metrics?

Ans:
From the **confusion matrix**:

- **True Positive (TP):** correctly predicted positive samples
- **False Positive (FP):** predicted positive but actually negative
- **True Negative (TN):** correctly predicted negative samples
- **False Negative (FN):** predicted negative but actually positive

Formulas:

- Accuracy = (TP + TN) / (TP + FP + TN + FN)
- Precision = TP / (TP + FP)
- Recall (Sensitivity) = TP / (TP + FN)
- Specificity = TN / (TN + FP)
- F1-score = 2 × (Precision × Recall) / (Precision + Recall)

5. In neural networks, “Neuron” is most similar to which of the following?

Ans: A. A neuron in the human brain.
The concept of an **artificial neuron** in a neural network is inspired by biological neurons in the human brain. In the artificial version, each neuron receives multiple input signals, each multiplied by a corresponding weight. These inputs are summed and passed through an **activation function** such as ReLU, Sigmoid, or Tanh to determine the output. This process mimics how biological neurons activate based on stimuli. Over time, through training, the model adjusts the weights to minimize error and improve learning performance.

6. Which of the following metrics mainly measures the model’s ability to detect “true positive samples”?

Ans: B. Sensitivity (Recall).
Sensitivity, also known as **Recall**, measures the model’s ability to correctly identify actual positive samples. It is calculated as TP / (TP + FN). High sensitivity means that the model is good at detecting true positives, making it especially important in applications like medical diagnosis or fraud detection, where missing a positive case (false negative) can have serious consequences.

7. What does the value of AUC represent?

Ans: B. The Area Under the ROC Curve.
The **AUC (Area Under the ROC Curve)** measures a model’s ability to distinguish between classes across all possible classification thresholds. A higher AUC value (closer to 1) indicates better performance. It represents the probability that the model ranks a randomly chosen positive instance higher than a randomly chosen negative instance. Thus, AUC provides a more comprehensive evaluation of model performance than accuracy alone, especially in imbalanced datasets.

8. Briefly explain the difference between Sensitivity (Recall) and Specificity.

Ans: Sensitivity (Recall) and **Specificity** are complementary metrics used to evaluate classification models.

- **Sensitivity (Recall):** Measures the proportion of actual positive samples correctly identified. It focuses on detecting positive cases and is crucial when missing a positive instance (false negative) has a high cost.
  Formula: Recall = TP / (TP + FN).

- **Specificity:** Measures the proportion of actual negative samples correctly identified as negative. It reflects how well the model avoids false alarms.
  Formula: Specificity = TN / (TN + FP).

9. Why is AUC (Area Under the ROC Curve) considered a better evaluation metric than simple Accuracy?

Ans:
AUC is often preferred over simple accuracy because it evaluates a model's performance across **all possible classification thresholds**, rather than a single cutoff. Accuracy can be misleading, particularly when the dataset is **imbalanced** (e.g., when one class heavily outnumbers the other).

AUC, on the other hand, considers both **Sensitivity (True Positive Rate)** and **Specificity (False Positive Rate)**, providing a holistic measure of how well the model distinguishes between positive and negative classes. A higher AUC means better discriminative ability. Therefore, AUC is a more robust and informative performance metric for evaluating classifiers in real-world applications.

10. In medical diagnosis, if a model has high Recall but low Precision, what problems might this cause?

Ans:
When a model in medical diagnosis exhibits **high recall but low precision**, it means it correctly identifies most actual sick patients (few false negatives), but it also incorrectly labels many healthy patients as sick (many false positives).

This situation can cause several problems. From a healthcare perspective, it may lead to **false alarms**, unnecessary medical tests, patient anxiety, and inefficient use of medical resources. While high recall ensures that no actual patient is missed, low precision decreases the trustworthiness of the model's positive predictions. Therefore, an optimal diagnostic model should maintain a good balance between recall and precision to ensure both reliability and efficiency..

11. Disease detection model (given data):

Given data:

- Actually sick = 40 → TP = 30, FN = 10
- Actually healthy = 60 → FP = 20, TN = 40

Now calculate key metrics:

- **Sensitivity (Recall)** = TP / (TP + FN) = 30 / 40 = **0.75 (75%)**
- **Specificity** = TN / (TN + FP) = 40 / 60 = **0.6667 (67%)**
- **Precision** = TP / (TP + FP) = 30 / 50 = **0.6 (60%)**

These results show that while the model can correctly identify 75% of sick patients, it still incorrectly classifies some healthy individuals as sick, indicating a need for improvement in precision and specificity.

12. Suppose a model has an AUC value of 0.95. Explain what this number indicates about the model's performance.

Ans:
An **AUC value of 0.95** indicates that the model has **excellent discriminative ability**. It means that 95% of the time, the model will correctly rank a randomly chosen positive sample higher than a randomly chosen negative sample.

In practical terms, this signifies that the model is highly capable of distinguishing between positive and negative cases with minimal overlap. Such a high AUC score implies strong predictive power, reliability, and suitability for deployment in real-world applications, such as disease detection, fraud prevention, or customer churn prediction.

Lab4 learning reflection or comments:

Through completing these three laboratory exercises, I developed a strong understanding of how IoT systems integrate sensing, communication, and real-time data visualization for smart monitoring applications. In Lab 1, I worked on the 3-cell Li-ion battery charging and monitoring system using ESP32, INA219, ADS1115, and a Flask web dashboard. Although the basic system was completed earlier, I enhanced it this week by adding a smoothing function that made the voltage and current graphs more stable and visually clear, helping me understand how data processing improves measurement accuracy and readability.

The set of long-answer questions guided me through the complete process of selecting, comparing, and evaluating different models. I gained insights into how the choice of model depends on the type of data, problem nature, and available computational resources. I also learned the importance of testing multiple models and the benefits of ensemble techniques like bagging, boosting, and stacking. The detailed study of metrics such as accuracy, precision, recall, specificity, and F1-score helped me understand how to evaluate models more effectively, especially in imbalanced datasets.

Additionally, the connection between neural networks and biological neurons enhanced my conceptual understanding of how learning occurs within a model. Through this week's tasks, I developed a clearer view of how metrics are calculated, interpreted, and used to judge model quality. Overall, these exercises not only improved my theoretical knowledge but also strengthened my ability to apply model evaluation techniques in practical machine learning scenarios.

Lab5 Video Link

Record a video of approximately 3 minutes to explain your research process. The video should include the following:

1.A description of the tools or methods used during your research process.

2. Present all the processes and outcomes from LAB1 to LAB2.

Link: **https://youtu.be/I9LKuj9xHIg**

Thank you