LAMPIRAN

LAMPIRAN

Surat Pengantar Penelitian Tugas Akhir

UNIVERSITAS 17 AGUSTUS 1945 (UNTAG) SURABAYA
FAKULTAS TEKNIK

Kampus : Jl. Semolowaru No. 45 Surabaya 60118 Telp. +62 31 5931800 (hunting) Fax, +62 31 5927817

- Program Studi Teknik Industri
- Program Studi Teknik Mesin
- Program Studi Teknik Sipil
- Program Studi Arsitektur
- Program Studi Teknik Elektro
- Program Studi Teknik Informatika
- Program Studi Sistem dan Teknologi Informasi
- Program Studi Teknik Robotika & Kecerdasan Buatan
- Program Studi Magister Teknik Sipil
- Program Studi Magister Arsitektur
- Program Studi Program Profesi Insinyur

Homepage : ft.untag-sby.ac.id Email : teknik@untag-sby.ac.id

Nomor : **1827/K/FT/Akd/XI/2024** **Surabaya, 11 November 2024**
Lampiran : -
Perihal : **Penelitian Tugas Akhir**

Kepada Yth : **PT. PLN (PERSERO) UPT Surabaya**
Jl. Ketintang Baru No. 9 Ketintang Kec. Gayungan Surabaya

Dengan hormat,
Sebagai salah satu persyaratan untuk menyelesaikan studi pada program Strata 1, maka mahasiswa/mahasiswi diwajibkan untuk melakukan **Penelitian Tugas Akhir** sebagai penerapan teori dan praktek yang diperoleh selama masa studinya.
Sehubungan dengan hal tersebut, maka dengan ini kami mohon Bapak/Ibu berkenan untuk memberikan ijin kepada mahasiswa/mahasiswi sebagai berikut :

No	Nama	NBI	EMAIL	No.HP
1.	Riyan Agus S	1452100044	riyanagus649@gmail.com	082337907370
2.	Ifan Maulana	1452100045		
3.	Aswanda H	1452100054		

Program Studi Teknik Elektro
Guna melaksanakan **Penelitian Tugas Akhir** di :

"PT. PLN (PERSERO) UPT Surabaya"

yang akan dimulai pada : **Semester Gasal 2024/2025**
Demikian permohonan kami, atas perkenannya disampaikan terima kasih.

Dekan,

Dr. Ir. Sajiyo, M.Kes., IPU., ASEAN Eng.
NPP : 20410.90.0197

Surat Balasan Penelitian Tugas Akhir

PLN

UITJBM
UPT SURABAYA

Nomor : 0621/STH.01.04/F34050000/2024
Lampiran : 1 Lembar
Sifat : Segera - Biasa
Hal : Persetujuan Melaksanakan Penelitian

24 Desember 2024

Kepada

Yth. Universitas 17 Agustus 1945
(UNTAG) Surabaya
Fakultas Teknik

Menunjuk Surat Saudara Nomor; 1827/K/FT/Akd/XI/2024 tanggal 11 Nopember 2024 perihal Izin Penelitian Tugas Akhir , dengan ini kami sampaikan Ijin kepada mahasiswa/i sebagai berikut :

No.	Nama	NIM
1	Ifan Maulana	1452100045
2	Riyan Agus Setiawan	1452100044
3	Aswanda Harja	1452100054

Untuk melakukan pengambilan data baik melalui survey maupun pengukuran di **PT PLN (Persero) UPT Surabaya ULTG Surabaya Utara Gardu Induk Surabaya Selatan** mulai tanggal **06 Januari s/d 03 Februari 2025** dengan persyaratan sebagai berikut :

1. Mahasiswa mengisi dan menanda tangani surat pernyataan 1 (satu) lembar bermaterai Rp. 10.000,-
2. Mahasiswa yang bersangkutan agar mematuhi peraturan / ketentuan yang berlaku di PT PLN (Persero) sehingga faktor-faktor kerahasiaan harus benar-benar diutamakan. Semua biaya perjalanan, penginapan, makan dan lain sebagainya tidak menjadi tanggungan PT PLN (Persero) UPT Surabaya.
3. Mahasiswa wajib mentaati protokol kesehatan dan 5M selama melaksanakan PKL / penelitian.
4. Mahasiswa sanggup tidak membocorkan hal-hal yang bersifat rahasia perusahaan PT PLN (Persero) UIT JBM UPT Surabaya, dan bahan yang diperoleh dalam Training / Praktek Kerja / Riset, dan tidak mempergunakan untuk hal-hal yang dapat merugikan PT PLN (Persero) UIT JBM UPT Surabaya
5. Untuk informasi lebih lanjut dapat menghubungi PT PLN (Persero) UPT Surabaya Cq. Bidang ADM & UMUM.

Demikian kami sampaikan, atas perhatian dan kerjasamanya kami ucapkan terimakasih.

MANAGER UNIT PELAKSANA
TRANSMISI SURABAYA,

GUNAWAN WIJAYA

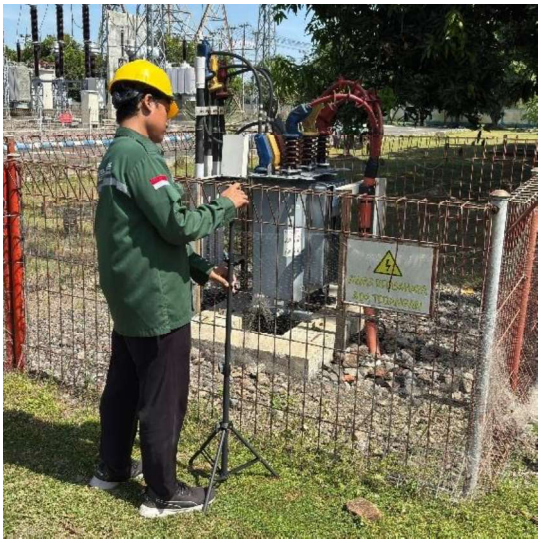
Tembusan:

1. MUL ULTG SURABAYA UTARA ULTG SURABAYA UTARA PLN

Jl. Ketintang Baru No 9, Surabaya 60231
T (031)8285505, 8285506 F (031)8284400 W www.pln.co.id

Paraf

Dokumentasi Pengambilan Data Tugas Akhir



Program Keseluruhan Alat Kamera *Thermal* dari Arduino IDE

```
#include <Wire.h>
#include <Adafruit_AMG88xx.h>
#include <SD.h>
#include <SPI.h>
#include <U8g2lib.h>
#include <WiFiS3.h>
#include <ArduinoJson.h>
#include <WiFiSSLClient.h>

// ===== ENHANCED DISCORD FUNCTIONS - ADAPTED FROM MAY26C =====
bool shouldSendDiscordNotification(bool isDanger, float percentage,
String classification, float avgTemp);
void checkAndSendDiscordAlert(bool isDanger, float percentage,
String classification, float avgTemp);
String createDiscordMessage(String classification, float
percentage, float avgTemp);
unsigned long getDiscordCooldown(String classification, float
percentage, float avgTemp);

// U8G2 OLED Configuration
U8G2_SSD1306_128X64_NONAME_F_HW_I2C display(U8G2_R0, /* reset=*/
U8X8_PIN_NONE);

// Objek sensor thermal
Adafruit_AMG88xx amg;

// WiFi settings - Multiple networks for backup
struct WiFiNetwork {
  const char* ssid;
  const char* password;
};

const int numNetworks = sizeof(wifiNetworks) /
sizeof(wifiNetworks[0]);

WiFiServer server(80);
int currentNetworkIndex = 0;

// SD Card dan pin
const int chipSelect = 10;
const int CAL_UP_PIN = 2;
const int CAL_SAVE_PIN = 3;
const int CAL_DOWN_PIN = 4;

// Variabel kalibrasi
float tempOffset = 0.0;
const float CAL_STEP = 0.5;

// Data training
float dataLatih[10][2] = {{0,0},{0,0},{0,0},{0,0},{0,0},
```

```
                                {0,0},{0,0},{0,0},{0,0},{0,0}};
  int label[10] = {0,0,0,0,0,0,0,0,0,0};
  bool trainingSelesai = false;

  const int neighborOptions[] = {3, 5, 7};
  const int neighborCount = sizeof(neighborOptions) /
  sizeof(neighborOptions[0]);
  int currentNeighborIndex = 0;

  String lastClassification = "NONE";
  float lastredPercentage = 0.0;
  int lastCount1 = 0;
  int lastCount0 = 0;

  // ===== ENHANCED DETECTION PARAMETERS FOR SMALL OBJECTS =====
  const float DISTANCE_FACTOR = 1.0;
  const float ROOM_TEMP_BASE = 25.0;

  // IMPROVED: Multi-level thresholds for different object sizes
  const float SMALL_OBJECT_THRESHOLD = 3.0;    // BARU: Threshold
  rendah untuk objek kecil (korek api)
  const float MEDIUM_OBJECT_THRESHOLD = 8.0;   // BARU: Threshold
  medium untuk objek sedang
  const float LARGE_OBJECT_THRESHOLD = 15.0;   // BARU: Threshold
  tinggi untuk objek besar

  const float PEAK_TEMP_THRESHOLD = 8.0;       // BARU: Threshold
  untuk peak temperature detection
  const float TEMP_HYSTERESIS = 2.0;           // Hysteresis untuk
  stabilitas
  const float DANGER_THRESHOLD = 20.0;         // DIPERBAIKI:
  Threshold bahaya yang lebih sensitif
  const float HIGH_TEMP_THRESHOLD = 50.0;      // Threshold suhu
  tinggi
  const float NOISE_FILTER_MIN = 10.0;         // Filter noise
  minimum
  const float NOISE_FILTER_MAX = 80.0;         // Filter noise
  maximum

  // ===== ENHANCED STABILITY SYSTEM FOR SMALL OBJECTS =====
  const int SMALL_OBJECT_STABILITY = 2;        // BARU: Stabilitas
  untuk objek kecil (lebih cepat)
  const int MEDIUM_OBJECT_STABILITY = 3;       // Stabilitas untuk
  objek sedang
  const int LARGE_OBJECT_STABILITY = 4;        // Stabilitas untuk
  objek besar

  int consecutiveDanger = 0;
  int consecutiveSafe = 0;
  bool stableClassification = false;
  String detectionMode = "NONE";               // BARU: Mode deteksi
  (SMALL/MEDIUM/LARGE)
```

```
83.
84.   // ===== ENHANCED AUTO-CALIBRATION =====
85.   float ambientTempSum = 0;
86.   int ambientReadings = 0;
87.   float ambientBaseline = 25.0;
88.   bool autoCalibrationDone = false;
89.   const int AMBIENT_SAMPLES = 30;              // DIPERBAIKI: Kurangi
      sampel untuk kalibrasi lebih cepat
90.
91.   // ===== NEW: SMALL OBJECT DETECTION SYSTEM =====
92.   float maxPixelTemp = 0;                      // BARU: Suhu pixel
      tertinggi
93.   float tempGradient = 0;                      // BARU: Gradien suhu
94.   int hotSpotCount = 0;                        // BARU: Jumlah hot
      spot
95.   float hotSpotIntensity = 0;                  // BARU: Intensitas
      hot spot
96.   bool peakDetected = false;                   // BARU: Peak
      temperature terdeteksi
97.
98.   // Status sistem
99.   bool systemReady = false;
100.  bool sdCardOK = false;
101.
102.  // ===== HELPER FUNCTIONS =====
103.  void displayOLEDMessage(String line1, String line2, bool clear) {
104.    if (clear) display.clearBuffer();
105.    display.setFont(u8g2_font_6x10_tf);
106.    display.setCursor(3, 10);
107.    display.print(line1.c_str());
108.    display.setCursor(3, 25);
109.    display.print(line2.c_str());
110.    display.sendBuffer();
111.  }
112.
113.  float ecludian(float x1, float y1, float x2, float y2) {
114.    return sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2));
115.  }
116.
117.  String getFormattedTime() {
118.    return String(millis()/1000);
119.  }
120.
121.  bool isNoiseFiltered(float temp) {
122.    return (temp > NOISE_FILTER_MIN && temp < NOISE_FILTER_MAX);
123.  }
124.
125.  // ===== NEW: ENHANCED DETECTION FUNCTIONS =====
126.
127.  // Deteksi objek kecil berdasarkan peak temperature
128.  bool detectSmallObject(float pixels[], float baseline) {
129.    float maxTemp = -100;
```

```
130.    float secondMaxTemp = -100;
131.    int maxTempIndex = -1;
132.    int hotPixelCount = 0;
133.    float totalTemp = 0;
134.
135.    // Cari suhu maksimum dan hitung statistik
136.    for (int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
137.      float temp = pixels[i] + tempOffset;
138.
139.      if (temp > maxTemp) {
140.        secondMaxTemp = maxTemp;
141.        maxTemp = temp;
142.        maxTempIndex = i;
143.      } else if (temp > secondMaxTemp) {
144.        secondMaxTemp = temp;
145.      }
146.
147.      // Hitung pixel panas dengan threshold rendah
148.      if (temp > baseline + SMALL_OBJECT_THRESHOLD) {
149.        hotPixelCount++;
150.        totalTemp += temp;
151.      }
152.    }
153.
154.    // Update global variables
155.    maxPixelTemp = maxTemp;
156.    tempGradient = maxTemp - secondMaxTemp;
157.    hotSpotCount = hotPixelCount;
158.    hotSpotIntensity = hotPixelCount > 0 ? totalTemp / hotPixelCount
      : 0;
159.
160.    // KONDISI DETEKSI OBJEK KECIL:
161.    // 1. Ada peak temperature yang signifikan
162.    bool hasSignificantPeak = (maxTemp - baseline) >
      PEAK_TEMP_THRESHOLD;
163.
164.    // 2. Gradien suhu cukup tajam (menandakan objek kecil panas)
165.    bool hasSteepGradient = tempGradient > 2.5;
166.
167.    // 3. Jumlah hot spot terbatas (1-6 pixel untuk objek kecil)
168.    bool isSmallHotSpot = (hotPixelCount >= 1 && hotPixelCount <= 8);
169.
170.    // 4. Intensitas hot spot tinggi
171.    bool hasHighIntensity = hotSpotIntensity > (baseline +
      SMALL_OBJECT_THRESHOLD);
172.
173.    peakDetected = hasSignificantPeak;
174.
175.    return hasSignificantPeak && hasSteepGradient && isSmallHotSpot
      && hasHighIntensity;
176.  }
177.
```

```
178.  // Deteksi objek medium berdasarkan area coverage
179.  bool detectMediumObject(float pixels[], float baseline) {
180.    int mediumHotPixelCount = 0;
181.
182.    for (int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
183.      float temp = pixels[i] + tempOffset;
184.      if (temp > baseline + MEDIUM_OBJECT_THRESHOLD) {
185.        mediumHotPixelCount++;
186.      }
187.    }
188.
189.    // Medium object: 6-20 hot pixels
190.    return (mediumHotPixelCount >= 6 && mediumHotPixelCount <= 20);
191.  }
192.
193.  // Deteksi objek besar berdasarkan coverage luas
194.  bool detectLargeObject(float pixels[], float baseline) {
195.    int largeHotPixelCount = 0;
196.
197.    for (int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
198.      float temp = pixels[i] + tempOffset;
199.      if (temp > baseline + LARGE_OBJECT_THRESHOLD) {
200.        largeHotPixelCount++;
201.      }
202.    }
203.
204.    // Large object: >15 hot pixels
205.    return (largeHotPixelCount > 15);
206.  }
207.
208.  // Tentukan mode deteksi dan stability requirement
209.  void determineDetectionMode(bool smallObj, bool mediumObj, bool
      largeObj) {
210.    if (smallObj) {
211.      detectionMode = "SMALL";
212.    } else if (mediumObj) {
213.      detectionMode = "MEDIUM";
214.    } else if (largeObj) {
215.      detectionMode = "LARGE";
216.    } else {
217.      detectionMode = "NONE";
218.    }
219.  }
220.
221.  // Get required stability count based on detection mode
222.  int getRequiredStability() {
223.    if (detectionMode == "SMALL") {
224.      return SMALL_OBJECT_STABILITY;
225.    } else if (detectionMode == "MEDIUM") {
226.      return MEDIUM_OBJECT_STABILITY;
227.    } else if (detectionMode == "LARGE") {
228.      return LARGE_OBJECT_STABILITY;
```

```
229.    }
230.    return MEDIUM_OBJECT_STABILITY; // default
231.  }
232.
233.  // ===== ENHANCED CALIBRATION =====
234.  void performAutoCalibration(float pixels[]) {
235.    if (autoCalibrationDone) return;
236.
237.    float sum = 0;
238.    int validReadings = 0;
239.
240.    for (int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
241.      float temp = pixels[i] + tempOffset;
242.      if (temp > 15.0 && temp < 35.0) { // Filter suhu ruangan normal
243.        sum += temp;
244.        validReadings++;
245.      }
246.    }
247.
248.    if (validReadings > 32) {
249.      float avgTemp = sum / validReadings;
250.      ambientTempSum += avgTemp;
251.      ambientReadings++;
252.
253.      Serial.print("Auto-calibration: ");
254.      Serial.print(ambientReadings);
255.      Serial.print("/");
256.      Serial.print(AMBIENT_SAMPLES);
257.      Serial.print(" - Current avg: ");
258.      Serial.println(avgTemp);
259.
260.      if (ambientReadings >= AMBIENT_SAMPLES) {
261.        ambientBaseline = ambientTempSum / ambientReadings;
262.        autoCalibrationDone = true;
263.
264.        Serial.println("=== AUTO-CALIBRATION COMPLETE ===");
265.        Serial.print("Ambient Baseline: ");
266.        Serial.print(ambientBaseline);
267.        Serial.println("°C");
268.
269.        displayOLEDMessage("Auto-Calibration", "Complete: " +
      String(ambientBaseline, 1) + "C");
270.        delay(2000);
271.      } else {
272.        displayOLEDMessage("Calibrating...", String(ambientReadings)
      + "/" + String(AMBIENT_SAMPLES));
273.      }
274.    }
275.  }
276.
277.  // ===== ADAPTED OLED DISPLAY FROM MAY26C STYLE =====
```

```
278.  void updateOLEDDisplay(float avgTemp, float redPercentage, String
      klasifikasi, int k, int count1, int count0) {
279.    display.clearBuffer();
280.
281.    // Header dengan style lebih sederhana seperti may26c
282.    display.setFont(u8g2_font_6x10_tf);
283.    display.setCursor(3, 10);
284.    display.print("Thermal Monitor");
285.
286.    // Temperature info (simplified)
287.    display.setCursor(3, 20);
288.    display.print("Temp: ");
289.    display.print(avgTemp, 1);
290.    display.print("C");
291.
292.    // Red percentage (adapted from may26c)
293.    display.setCursor(3, 30);
294.    display.print("Hot%: ");
295.    display.print(redPercentage, 1);
296.    display.print("%");
297.
298.    // KNN info (adapted from may26c style)
299.    display.setCursor(3, 40);
300.    display.print("K=");
301.    display.print(k);
302.    display.print(" (");
303.    display.print(count1);
304.    display.print("/");
305.    display.print(count0);
306.    display.print(")");
307.
308.    // Status line (simplified classification display)
309.    display.setFont(u8g2_font_7x14B_tf);
310.    display.setCursor(3, 55);
311.    if (!autoCalibrationDone) {
312.      display.print("Calibrating...");
313.    } else if (stableClassification) {
314.      if (klasifikasi.startsWith("BAHAYA")) {
315.        display.print("BAHAYA!");
316.      } else {
317.        display.print("AMAN");
318.      }
319.    } else {
320.      display.print("Checking...");
321.    }
322.
323.    // Status indicators (adapted from may26c - simple and clean)
324.    display.setFont(u8g2_font_6x10_tf);
325.    // if (wifiConnected) {
326.    //   display.setCursor(85, 10);
327.    //   display.print("WiFi");
328.    // }
```

```
329.
330.    if (discordEnabled && wifiConnected) {
331.      display.setCursor(100, 10);
332.      display.print("DC");
333.    }
334.
335.    // Detection mode indicator (new but simple)
336.    if (autoCalibrationDone && detectionMode != "NONE") {
337.      display.setCursor(85, 20);
338.      display.print(detectionMode.c_str());
339.    }
340.
341.    // Peak indicator (simple)
342.    if (peakDetected) {
343.      display.setCursor(85, 30);
344.      display.print("PEAK");
345.    }
346.
347.    // Offline indicator
348.    if (!wifiConnected) {
349.      display.setCursor(90, 50);
350.      display.print("OFFLINE");
351.    }
352.
353.    display.sendBuffer();
354.  }
355.
356.  // ===== ADAPTED DISCORD FUNCTIONS FROM MAY26C =====
357.
358.
359.    // Danger classification
360.    // if (classification.startsWith("BAHAYA")) {
361.    //   return DISCORD_DANGER_COOLDOWN; // 10 detik untuk bahaya
362.    // }
363.
364.    // Safe classification
365.    return DISCORD_SAFE_COOLDOWN; // 60 detik untuk aman
366.  }
367.
368.  // Cek apakah perlu mengirim notifikasi Discord (adapted from
      may26c)
369.  bool shouldSendDiscordNotification(bool isDanger, float percentage,
      String classification, float avgTemp) {
370.    unsigned long currentTime = millis();
371.    unsigned long requiredCooldown =
      getDiscordCooldown(classification, percentage, avgTemp);
372.
373.    // Cooldown check with dynamic timing
374.    if (currentTime - lastDiscordNotification < requiredCooldown) {
375.      return false;
376.    }
377.
```

```
  bool shouldSend = false;
  String reason = "";

  // Emergency conditions - send immediately regardless of cooldown
  if (avgTemp >= EMERGENCY_TEMP_THRESHOLD) {
    shouldSend = true;
    reason = "Emergency temperature: " + String(avgTemp, 1) + "°C";
  }
  else if (percentage >= CRITICAL_PERCENTAGE_THRESHOLD &&
(percentage - lastNotifiedPercentage > 5.0)) {
    shouldSend = true;
    reason = "Critical percentage: " + String(percentage, 1) + "%";
  }
  else if (isDanger && !lastDangerState) {
    // Transisi dari aman ke bahaya
    shouldSend = true;
    reason = "Safe to Danger transition";
  }
  else if (isDanger && (percentage - lastNotifiedPercentage >
PERCENTAGE_INCREASE_THRESHOLD)) {
    // Peningkatan signifikan dalam kondisi bahaya
    shouldSend = true;
    reason = "Significant percentage increase: +" +
String(percentage - lastNotifiedPercentage, 1) + "%";
  }
  else if (classification.startsWith("BAHAYA") &&
!lastClassification.startsWith("BAHAYA")) {
    // Perubahan klasifikasi menjadi bahaya
    shouldSend = true;
    reason = "Classification changed to DANGER";
  }
  else if (!isDanger && lastDangerState && (currentTime -
lastDiscordNotification > DISCORD_SAFE_COOLDOWN)) {
    // Transisi dari bahaya ke aman (dengan cooldown lebih lama)
    shouldSend = true;
    reason = "Danger to Safe transition";
  }

  if (shouldSend) {
    Serial.println("Discord trigger: " + reason);
    Serial.println("Cooldown used: " +
String(requiredCooldown/1000) + " seconds");
  }

  return shouldSend;
}

// Buat pesan Discord (adapted from may26c style - cleaner and more
readable - FOR EMBED ONLY)
String createDiscordMessage(String classification, float
percentage, float avgTemp) {
  String message = "";
```

```
422.
423.   if (classification.startsWith("BAHAYA")) {
424.     // message = "⚠️ **DANGER CLASSIFICATION ACTIVE**\n\n";
425.     // message = "**📊 Alert Details:**\n\n";
426.     message = "🌡️ **Temperature:** " + String(avgTemp, 1) + "°C\n";
427.     message += "🔥 **Peak Temperature:** " + String(maxPixelTemp,
    1) + "°C\n";
428.     message += "🤖 **Classification:** " + classification + "\n\n";
429.
430.     if (wifiConnected) {
431.     message += "📡 **WiFi:** Connected\n";
432.   } else {
433.     message += "📡 **WiFi:** Disconnected\n";
434.   }
435.
436.   message += "🔧 **Auto-Calib:** " + String(autoCalibrationDone ?
    "Complete" : "Pending") + "\n";
437.   message += "_Enhanced thermal monitoring with small object
    detection_";
438.     // message = "• **Temperature Rise:** +" + String(avgTemp -
    ambientBaseline, 1) + "°C above baseline\n";
439.     // message += "• **Temperature Gradient:** " +
    String(tempGradient, 1) + "°C\n";
440.     // message += "• **Baseline Temperature:** " +
    String(ambientBaseline, 1) + "°C\n";
441.     // message += "• **Hot Spot Intensity:** " +
    String(hotSpotIntensity, 1) + "°C\n";
442.     // message += "• **Peak Detected:** " + String(peakDetected ?
    "YES" : "NO") + "\n";
443.     // message += "• **Consecutive Readings:** " +
    String(consecutiveDanger) + " confirmed\n\n";
444.     message += "🔔 **This is a CONFIRMED alert with enhanced small
    object detection!**\n";
445.     message += "_Immediate attention required!_";
446.   } else {
447.     message = "✅ **System Status: NORMAL**\n\n";
448.     message += "**📊 Current Reading:**\n";
449.     message += "• Temperature is within normal range\n";
450.     message += "• Hot area coverage: " + String(percentage, 1) +
    "%\n";
451.     message += "• Detection mode: " + detectionMode + "\n";
452.     message += "• Baseline: " + String(ambientBaseline, 1) +
    "°C\n\n";
453.     message += "_Enhanced monitoring system operating normally_";
454.   }
455.
456.   return message;
457. }
458.
459. // Enhanced scheduled message (adapted style from may26c)
```

```
460.  String createScheduledMessage(float avgTemp, float percentage,
      String classification, bool isDanger) {
461.    String
462.    // message = "📊 **Enhanced Status Report**\n\n";
463.    // message = "🕐 **Time:** " + String(millis()/1000) + " seconds
      since boot\n";
464.    message = "🌡️ **Temperature:** " + String(avgTemp, 1) + "°C\n";
465.    message += "🔥 **Peak Temperature:** " + String(maxPixelTemp, 1)
      + "°C\n";
466.    // message += "📈 **Hot Area:** " + String(percentage, 1) +
      "%\n";
467.    // message += "🎯 **Detection Mode:** " + detectionMode + "\n";
468.    message += "🚨 **Classification:** " + classification + "\n";
469.    message += "⚠️ **Status:** " + String(isDanger ? "⚠️ DANGER" :
      "✅ NORMAL") + "\n\n";
470.
471.    if (wifiConnected) {
472.      message += "📡 **WiFi:** Connected\n";
473.    } else {
474.      message += "📡 **WiFi:** Disconnected\n";
475.    }
476.
477.    message += "🔧 **Auto-Calib:** " + String(autoCalibrationDone ?
      "Complete" : "Pending") + "\n";
478.    message += "_Enhanced thermal monitoring with small object
      detection_";
479.
480.    return message;
481.  }
482.
483.  // Danger message creation (enhanced but cleaner)
484.  String createDangerMessage(float avgTemp, float percentage, String
      classification) {
485.    return createDiscordMessage(classification, percentage, avgTemp);
486.  }
487.
488.  // Send Discord notification (adapted from may26c with SSL support
      - ALL TEXT IN EMBED)
489.  bool sendDiscordNotification(String message, String classification,
      float percentage, float avgTemp) {
490.    if (!wifiConnected || !discordEnabled) {
491.      return false;
492.    }
493.
494.    unsigned long currentTime = millis();
495.    if (currentTime - lastDiscordAttempt < DISCORD_RETRY_INTERVAL) {
496.      Serial.println("Discord rate limited, waiting...");
497.      return false;
498.    }
499.
500.    lastDiscordAttempt = currentTime;
```

```
501.    Serial.println("Attempting Discord notification...");
502.
503.    if (!isWiFiStillConnected()) {
504.      Serial.println("WiFi not ready for Discord notification");
505.      return false;
506.    }
507.
508.    WiFiSSLClient discordClient;
509.    bool success = false;
510.
511.    discordClient.setTimeout(10000);
512.
513.    if (discordClient.connect(DISCORD_HOST, DISCORD_PORT)) {
514.      Serial.println("Connected to Discord API");
515.
516.      // Create JSON payload - ALL CONTENT IN EMBED ONLY
517.      StaticJsonDocument<1536> doc; // Increased size for embed
      content
518.
519.      // Create embeds array - NO CONTENT FIELD OUTSIDE EMBED
520.      JsonArray embeds = doc.createNestedArray("embeds");
521.      JsonObject embed = embeds.createNestedObject();
522.
523.      // Enhanced title based on classification
524.      if (classification.startsWith("BAHAYA")) {
525.        embed["title"] = "🚨 THERMAL ALERT DETECTED! 🚨";
526.      } else {
527.        embed["title"] = "📊 Enhanced Thermal Monitor Status";
528.      }
529.
530.      // ALL MESSAGE CONTENT IN DESCRIPTION
531.      embed["description"] = message;
532.
533.      // Set color based on classification
534.      if (classification.startsWith("BAHAYA")) {
535.        embed["color"] = 15158332; // Red color
536.      } else {
537.        embed["color"] = 3066993;  // Green color
538.      }
539.
540.      // Add detailed fields (enhanced but clean)
541.      JsonArray fields = embed.createNestedArray("fields");
542.
543.      JsonObject field1 = fields.createNestedObject();
544.      field1["name"] = "🌡️ Average Temperature";
545.      field1["value"] = String(avgTemp, 1) + "°C";
546.      field1["inline"] = true;
547.
548.      JsonObject field2 = fields.createNestedObject();
549.      field2["name"] = "🔥 Peak Temperature";
550.      field2["value"] = String(maxPixelTemp, 1) + "°C";
551.      field2["inline"] = true;
```

```
552.
553.      // JsonObject field3 = fields.createNestedObject();
554.      // field3["name"] = "📈 Hot Area Coverage";
555.      // field3["value"] = String(percentage, 1) + "%";
556.      // field3["inline"] = true;
557.
558.      // JsonObject field4 = fields.createNestedObject();
559.      // field4["name"] = "🎯 Detection Mode";
560.      // field4["value"] = detectionMode;
561.      // field4["inline"] = true;
562.
563.      JsonObject field3 = fields.createNestedObject();
564.      field3["name"] = "🤖 Classification";
565.      field3["value"] = classification;
566.      field3["inline"] = true;
567.
568.      // JsonObject field6 = fields.createNestedObject();
569.      // field6["name"] = "🔍 Hot Spots";
570.      // field6["value"] = String(hotSpotCount) + " pixels";
571.      // field6["inline"] = true;
572.
573.      // Add timestamp
574.      embed["timestamp"] = ""; // Discord will auto-fill current time
575.
576.      // Add footer
577.      JsonObject footer = embed.createNestedObject("footer");
578.      footer["text"] = "Enhanced Small Object Detection";
579.
580.      String jsonString;
581.      serializeJson(doc, jsonString);
582.
583.      Serial.println("Discord JSON payload:");
584.      Serial.println(jsonString);
585.
586.      // Extract webhook path from URL
587.      String webhookPath = String(DISCORD_WEBHOOK_URL);
588.      int pathStart = webhookPath.indexOf("/api/webhooks/");
589.      if (pathStart == -1) {
590.        Serial.println("Invalid webhook URL format");
591.        discordClient.stop();
592.        return false;
593.      }
594.      String path = webhookPath.substring(pathStart);
595.
596.      // Send HTTP POST request
597.      discordClient.println("POST " + path + " HTTP/1.1");
598.      discordClient.println("Host: " + String(DISCORD_HOST));
599.      discordClient.println("Content-Type: application/json");
600.      discordClient.println("Content-Length: " +
      String(jsonString.length()));
601.      discordClient.println("Connection: close");
602.      discordClient.println();
```

```
603.      discordClient.print(jsonString);
604.
605.      // Wait for response
606.      unsigned long timeout = millis();
607.      String response = "";
608.
609.      while (discordClient.connected() && millis() - timeout < 8000)
      {
610.        if (discordClient.available()) {
611.          response = discordClient.readStringUntil('\n');
612.          Serial.println("Discord Response: " + response);
613.
614.          if (response.indexOf("HTTP/1.1 204") >= 0 ||
      response.indexOf("HTTP/1.1 200") >= 0) {
615.            success = true;
616.            break;
617.          }
618.        }
619.        delay(10);
620.      }
621.
622.      discordClient.stop();
623.
624.      if (success) {
625.        Serial.println("Discord notification sent successfully!");
626.        discordFailCount = 0;
627.      } else {
628.        Serial.println("Discord notification failed or timed out");
629.        discordFailCount++;
630.      }
631.
632.    } else {
633.      Serial.println("Failed to connect to Discord API");
634.      discordFailCount++;
635.    }
636.
637.    if (discordFailCount >= MAX_DISCORD_FAIL_COUNT) {
638.      Serial.println("Too many Discord failures, disabling for a
      while...");
639.      discordEnabled = false;
640.      discordDisabledTime = millis();
641.    }
642.
643.    return success;
644.  }
645.
646.  // Check and send Discord alert (adapted from may26c)
647.  void checkAndSendDiscordAlert(bool isDanger, float percentage,
      String classification, float avgTemp) {
648.    if (!discordEnabled || !wifiConnected) {
649.      return;
650.    }
```

```
651.
652.   if (shouldSendDiscordNotification(isDanger, percentage,
     classification, avgTemp)) {
653.     String message = createDiscordMessage(classification,
     percentage, avgTemp);
654.
655.     if (sendDiscordNotification(message, classification,
     percentage, avgTemp)) {
656.       lastDiscordNotification = millis();
657.       lastDangerState = isDanger;
658.       lastNotifiedPercentage = percentage;
659.
660.       Serial.println("Discord notification sent successfully");
661.
662.       // Show different messages based on severity (adapted from
     may26c)
663.       if (avgTemp >= EMERGENCY_TEMP_THRESHOLD) {
664.         displayOLEDMessage("EMERGENCY ALERT", "Discord Sent!",
     false);
665.       } else if (classification.startsWith("BAHAYA")) {
666.         displayOLEDMessage("Discord Alert", "Danger Sent!", false);
667.       } else {
668.         displayOLEDMessage("Discord Update", "Status Sent!",
     false);
669.       }
670.     } else {
671.       Serial.println("Failed to send Discord notification");
672.       displayOLEDMessage("Discord Error", "Send Failed", false);
673.     }
674.   }
675. }
676.
677. // ===== DISCORD FUNCTIONS =====
678. void checkDiscordReEnable() {
679.   if (!discordEnabled && discordDisabledTime > 0) {
680.     unsigned long currentTime = millis();
681.     if (currentTime - discordDisabledTime >=
     DISCORD_REENABLE_DELAY) {
682.       discordEnabled = true;
683.       discordFailCount = 0;
684.       discordDisabledTime = 0;
685.       Serial.println("Discord re-enabled after timeout");
686.     }
687.   }
688. }
689.
690. void checkDiscordNotifications(bool isDanger, float percentage,
     String classification, float avgTemp) {
691.   if (!wifiConnected) {
692.     return;
693.   }
694.
```

```
695.    checkDiscordReEnable();
696.
697.    unsigned long currentTime = millis();
698.
699.    // 1. CEK NOTIFIKASI TERJADWAL (1 jam)
700.    if (currentTime - lastScheduledNotification >=
      SCHEDULED_INTERVAL) {
701.      String scheduledMessage = createScheduledMessage(avgTemp,
      percentage, classification, isDanger);
702.
703.      Serial.println("Sending scheduled hourly report...");
704.      displayOLEDMessage("Discord Report", "Sending hourly...");
705.
706.      if (sendDiscordNotification(scheduledMessage, classification,
      percentage, avgTemp)) {
707.        lastScheduledNotification = currentTime;
708.        Serial.println("Hourly Discord report sent successfully");
709.        displayOLEDMessage("Discord Report", "Hourly sent!");
710.      } else {
711.        Serial.println("Failed to send hourly Discord report");
712.        displayOLEDMessage("Discord Report", "Hourly failed!");
713.      }
714.
715.      delay(1000);
716.    }
717.
718.    // 2. CEK NOTIFIKASI BAHAYA (HANYA JIKA STABIL DAN BARU)
719.    if (isDanger && stableClassification && !lastWasDanger) {
720.      String dangerMessage = createDangerMessage(avgTemp, percentage,
      classification);
721.
722.      Serial.println("Sending CONFIRMED danger alert...");
723.      displayOLEDMessage("CONFIRMED DANGER", "Sending Discord...");
724.
725.      if (sendDiscordNotification(dangerMessage, classification,
      percentage, avgTemp)) {
726.        Serial.println("Confirmed danger Discord alert sent
      successfully");
727.        displayOLEDMessage("CONFIRMED DANGER", "Discord sent!");
728.      } else {
729.        Serial.println("Failed to send confirmed danger Discord
      alert");
730.        displayOLEDMessage("CONFIRMED DANGER", "Discord failed!");
731.      }
732.
733.      delay(1000);
734.    }
735.
736.    // 3. ADDITIONAL CHECK using may26c style logic
737.    checkAndSendDiscordAlert(isDanger, percentage, classification,
      avgTemp);
738.
```

```
    if (stableClassification) {
      lastWasDanger = isDanger;
    }
  }

  // ===== WIFI FUNCTIONS =====
  String getMacAddress() {
    uint8_t mac[6];
    WiFi.macAddress(mac);
    String macStr = "";
    for (int i = 0; i < 6; i++) {
      if (mac[i] < 16) macStr += "0";
      macStr += String(mac[i], HEX);
      if (i < 5) macStr += ":";
    }
    macStr.toUpperCase();
    return macStr;
  }

  void printWiFiStatus() {
    Serial.println("\n==== WiFi Status ====");
    Serial.print("Status: ");
    Serial.println(getWiFiStatusString(WiFi.status()));

    if (WiFi.status() == WL_CONNECTED) {
      Serial.print("SSID: ");
      String ssid = WiFi.SSID();
      Serial.println(ssid.length() > 0 ? ssid : "Unknown");
      Serial.print("IP Address: ");
      Serial.println(WiFi.localIP());
      Serial.print("Signal Strength (RSSI): ");
      Serial.print(WiFi.RSSI());
      Serial.println(" dBm");
      Serial.print("MAC Address: ");
      Serial.println(getMacAddress());
      Serial.print("Connected Time: ");
      Serial.print((millis() - wifiConnectedTime) / 1000);
      Serial.println(" seconds");
    }
    Serial.println("====================\n");
  }

  String getWiFiStatusString(int status) {
    switch (status) {
      case WL_IDLE_STATUS: return "Idle";
      case WL_NO_SSID_AVAIL: return "No SSID Available";
      case WL_SCAN_COMPLETED: return "Scan Completed";
      case WL_CONNECTED: return "Connected";
      case WL_CONNECT_FAILED: return "Connection Failed";
      case WL_CONNECTION_LOST: return "Connection Lost";
      case WL_DISCONNECTED: return "Disconnected";
      default: return "Unknown (" + String(status) + ")";
```

```
  }
}

void initWiFiStability() {
  wifiConnectedTime = 0;
  totalWiFiDisconnections = 0;
  lastWiFiKeepAlive = millis();
  wifiReconnectAttempts = 0;
  wifiRetryInProgress = false;

  Serial.println("WiFi stability system initialized");
}

bool isWiFiStillConnected() {
  int status = WiFi.status();
  bool connected = (status == WL_CONNECTED);

  if (connected) {
    IPAddress localIP = WiFi.localIP();
    connected = (localIP != IPAddress(0, 0, 0, 0));
  }

  return connected;
}

void performWiFiKeepAlive() {
  unsigned long currentTime = millis();

  if (currentTime - lastWiFiKeepAlive > WIFI_KEEPALIVE_INTERVAL) {
    if (isWiFiStillConnected()) {
      IPAddress localIP = WiFi.localIP();
      Serial.println("WiFi Keep-Alive OK: " + localIP.toString());
    } else {
      Serial.println("WiFi Keep-Alive FAILED");
      wifiConnected = false;
    }

    lastWiFiKeepAlive = currentTime;
  }
}

void checkWiFiStability() {
  unsigned long currentTime = millis();

  if (currentTime - lastWiFiCheck < WIFI_CHECK_INTERVAL) {
    return;
  }

  lastWiFiCheck = currentTime;

  bool actuallyConnected = isWiFiStillConnected();

```

```
843.    if (actuallyConnected && !wifiConnected) {
844.      wifiConnected = true;
845.      wifiConnectedTime = currentTime;
846.      wifiReconnectAttempts = 0;
847.      wifiRetryInProgress = false;
848.
849.      Serial.println("WiFi connection restored");
850.      displayOLEDMessage("WiFi Restored", WiFi.localIP().toString());
851.
852.    } else if (!actuallyConnected && wifiConnected) {
853.      wifiConnected = false;
854.      totalWiFiDisconnections++;
855.
856.      Serial.println("WiFi disconnected - attempting reconnection");
857.      displayOLEDMessage("WiFi Lost", "Reconnecting...");
858.
859.      if (!wifiRetryInProgress && wifiReconnectAttempts <
     MAX_WIFI_RECONNECT_ATTEMPTS) {
860.        wifiRetryInProgress = true;
861.        wifiReconnectStartTime = currentTime;
862.        wifiReconnectAttempts++;
863.
864.        WiFiNetwork network = wifiNetworks[currentNetworkIndex];
865.
866.        Serial.println("Attempting quick reconnection to: " +
     String(network.ssid));
867.
868.        if (strlen(network.password) == 0) {
869.          WiFi.begin(network.ssid);
870.        } else {
871.          WiFi.begin(network.ssid, network.password);
872.        }
873.
874.        unsigned long reconnectStart = millis();
875.        while (millis() - reconnectStart < 15000) {
876.          if (isWiFiStillConnected()) {
877.            wifiConnected = true;
878.            wifiRetryInProgress = false;
879.            wifiReconnectAttempts = 0;
880.
881.            Serial.println("Quick reconnection successful!");
882.            displayOLEDMessage("WiFi Restored",
     WiFi.localIP().toString());
883.
884.            server.begin();
885.            return;
886.          }
887.          delay(500);
888.        }
889.
890.        wifiRetryInProgress = false;
891.        Serial.println("Quick reconnection failed");
```

```
892.
893.       if (wifiReconnectAttempts >= MAX_WIFI_RECONNECT_ATTEMPTS) {
894.         Serial.println("Trying all networks...");
895.         if (tryAllWiFiNetworks()) {
896.           server.begin();
897.           Serial.println("Successfully reconnected to WiFi");
898.           displayOLEDMessage("WiFi Restored",
      WiFi.localIP().toString());
899.         } else {
900.           Serial.println("All reconnection attempts failed");
901.           displayOLEDMessage("WiFi Failed", "All attempts failed");
902.           discordEnabled = false;
903.         }
904.
905.         wifiReconnectAttempts = 0;
906.       }
907.     }
908.   }
909.
910.   if (wifiConnected) {
911.     performWiFiKeepAlive();
912.   }
913. }
914.
915. void scanWiFiNetworks() {
916.   Serial.println("Scanning for WiFi networks...");
917.   displayOLEDMessage("WiFi Scan", "Searching...");
918.
919.   int n = WiFi.scanNetworks();
920.
921.   if (n == 0) {
922.     Serial.println("No networks found");
923.     displayOLEDMessage("WiFi Scan", "No networks found");
924.   } else {
925.     Serial.print("Found ");
926.     Serial.print(n);
927.     Serial.println(" networks:");
928.
929.     for (int i = 0; i < n; ++i) {
930.       Serial.print(i + 1);
931.       Serial.print(": ");
932.       Serial.print(WiFi.SSID(i));
933.       Serial.print(" (");
934.       Serial.print(WiFi.RSSI(i));
935.       Serial.print(" dBm) ");
936.
937.       uint8_t encType = WiFi.encryptionType(i);
938.       if (encType == 0) {
939.         Serial.println("[OPEN]");
940.       } else {
941.         Serial.println("[ENCRYPTED]");
942.       }
```

```
943.      }
944.
945.      displayOLEDMessage("WiFi Scan", String(n) + " networks found");
946.    }
947.
948.    Serial.println();
949.    delay(2000);
950.  }
951.
952.  bool connectToWiFi(int networkIndex, int maxAttempts) {
953.    if (networkIndex >= numNetworks) return false;
954.
955.    WiFiNetwork network = wifiNetworks[networkIndex];
956.
957.    Serial.print("Attempting to connect to: ");
958.    Serial.println(network.ssid);
959.
960.    displayOLEDMessage("Connecting WiFi", String(network.ssid));
961.
962.    if (WiFi.status() == WL_CONNECTED) {
963.      WiFi.disconnect();
964.      delay(1000);
965.    }
966.
967.    if (strlen(network.password) == 0) {
968.      WiFi.begin(network.ssid);
969.    } else {
970.      WiFi.begin(network.ssid, network.password);
971.    }
972.
973.    int attempts = 0;
974.    unsigned long startTime = millis();
975.
976.    while (WiFi.status() != WL_CONNECTED && attempts < maxAttempts) {
977.      delay(500);
978.      Serial.print(".");
979.      attempts++;
980.
981.      if (attempts % 4 == 0) {
982.        displayOLEDMessage("Connecting WiFi",
983.                           String(network.ssid) + " (" +
      String(attempts) + "/" + String(maxAttempts) + ")");
984.      }
985.
986.      if (millis() - startTime > 30000) {
987.        Serial.println("\nConnection timeout!");
988.        break;
989.      }
990.    }
991.
992.    if (WiFi.status() == WL_CONNECTED) {
993.      Serial.println("\nWiFi connected successfully!");
```

```
994.      Serial.print("IP address: ");
995.      Serial.println(WiFi.localIP());
996.      Serial.print("Signal strength: ");
997.      Serial.print(WiFi.RSSI());
998.      Serial.println(" dBm");
999.
1000.     displayOLEDMessage("WiFi Connected!",
      WiFi.localIP().toString());
1001.     currentNetworkIndex = networkIndex;
1002.     wifiConnectedTime = millis();
1003.     lastWiFiKeepAlive = millis();
1004.     return true;
1005.   } else {
1006.     Serial.println("\nFailed to connect to WiFi");
1007.     Serial.print("Final status: ");
1008.     Serial.println(getWiFiStatusString(WiFi.status()));
1009.
1010.     displayOLEDMessage("WiFi Failed",
      getWiFiStatusString(WiFi.status()));
1011.     return false;
1012.   }
1013. }
1014.
1015. bool tryAllWiFiNetworks() {
1016.   Serial.println("Trying all configured WiFi networks...");
1017.
1018.   scanWiFiNetworks();
1019.
1020.   for (int i = 0; i < numNetworks; i++) {
1021.     Serial.print("Trying network ");
1022.     Serial.print(i + 1);
1023.     Serial.print(" of ");
1024.     Serial.print(numNetworks);
1025.     Serial.print(": ");
1026.     Serial.println(wifiNetworks[i].ssid);
1027.
1028.     if (connectToWiFi(i, 15)) {
1029.       return true;
1030.     }
1031.
1032.     delay(2000);
1033.   }
1034.
1035.   Serial.println("Failed to connect to any WiFi network!");
1036.   displayOLEDMessage("WiFi Error", "All networks failed");
1037.   return false;
1038. }
1039.
1040. // ===== DATA MANAGEMENT FUNCTIONS =====
1041. bool loadTrainingData() {
1042.   File dataFile = SD.open("training.txt");
1043.
```

```
  if (dataFile) {
    int index = 0;

    while (dataFile.available() && index < 10) {
      String line = dataFile.readStringUntil('\n');
      line.trim();

      int firstComma = line.indexOf(',');
      int secondComma = line.indexOf(',', firstComma + 1);

      if (firstComma != -1 && secondComma != -1) {
        dataLatih[index][0] = line.substring(0, firstComma).toFloat();
        dataLatih[index][1] = line.substring(firstComma + 1, secondComma).toFloat();
        label[index] = line.substring(secondComma + 1).toInt();
        index++;
      }
    }

    dataFile.close();

    if (index == 10) {
      Serial.println("Data training berhasil dimuat");
      trainingSelesai = true;
      return true;
    } else {
      Serial.println("File data training tidak lengkap");
      return false;
    }
  } else {
    Serial.println("File training.txt tidak ditemukan");
    return false;
  }
}

void loadCalibration() {
  if (SD.exists("calib.txt")) {
    File calibFile = SD.open("calib.txt", FILE_READ);
    if (calibFile) {
      String calibStr = calibFile.readStringUntil('\n');
      tempOffset = calibStr.toFloat();
      calibFile.close();

      Serial.print("Calibration Offset: ");
      Serial.print(tempOffset);
      Serial.println("°C");
    }
  }
}

void saveCalibration() {
```

```
  File calibFile = SD.open("calib.txt", FILE_WRITE);
  if (calibFile) {
    calibFile.println(tempOffset);
    calibFile.close();
    Serial.print("New Calibration Offset: ");
    Serial.print(tempOffset);
    Serial.println("°C");
  }
}

void logClassificationResult(float redPercentage, int count1, int count0, String klasifikasi) {
  if (!sdCardOK) return;

  String timestamp = getFormattedTime();

  File logFile = SD.open("enhanced_log.txt", FILE_WRITE);
  if (logFile) {
    String logEntry = timestamp + "," +
                      String(redPercentage, 1) + "," +
                      String(count1) + "," +
                      String(count0) + "," +
                      klasifikasi + "," +
                      detectionMode + "," +
                      String(maxPixelTemp, 1) + "," +
                      String(tempGradient, 1) + "," +
                      String(hotSpotCount) + "," +
                      String(hotSpotIntensity, 1) + "," +
                      String(peakDetected ? "PEAK" : "NO_PEAK") + "," +
                      String(stableClassification ? "STABLE" : "CHECKING") + "," +
                      String(consecutiveDanger) + "," +
                      String(consecutiveSafe) + "," +
                      (wifiConnected ? "WIFI_ON" : "WIFI_OFF") + "," +
                      (discordEnabled ? "DC_ON" : "DC_OFF");

    logFile.println(logEntry);
    logFile.close();
  }

  if (klasifikasi.startsWith("BAHAYA") && stableClassification) {
    File alertFile = SD.open("enhanced_alerts.txt", FILE_WRITE);
    if (alertFile) {
      alertFile.println(timestamp + "," + String(redPercentage, 1) + "," + klasifikasi + "," + detectionMode + ",CONFIRMED,DISCORD_" + (discordEnabled ? "SENT" : "DISABLED"));
      alertFile.close();
    }
  }
}
```

```
1140.
1141. bool tryLoadTrainingData(int maxRetries) {
1142.   for (int i = 0; i < maxRetries; i++) {
1143.     Serial.print("Attempting to load training data (attempt ");
1144.     Serial.print(i + 1);
1145.     Serial.println(")");
1146.
1147.     displayOLEDMessage("Loading Data", "Attempt " + String(i + 1));
1148.
1149.     if (SD.begin(chipSelect) && loadTrainingData()) {
1150.       sdCardOK = true;
1151.       return true;
1152.     }
1153.
1154.     delay(1000);
1155.   }
1156.
1157.   return false;
1158. }
1159.
1160. // ===== ENHANCED CLASSIFICATION FUNCTION =====
1161. void doKNNClassification(float pixels[], float& avgTemp, float&
      redPercentage, String& classification, int& k, int& count1, int&
      count0, bool& isDanger) {
1162.
1163.   // LANGKAH 1: AUTO-CALIBRATION
1164.   if (!autoCalibrationDone) {
1165.     performAutoCalibration(pixels);
1166.     classification = "CALIBRATING";
1167.     isDanger = false;
1168.     avgTemp = ambientBaseline;
1169.     redPercentage = 0;
1170.     count1 = 0;
1171.     count0 = 0;
1172.     k = 0;
1173.     return;
1174.   }
1175.
1176.   // LANGKAH 2: ENHANCED TEMPERATURE CALCULATION
1177.   int redPixelCount = 0;
1178.   float sumTemp = 0;
1179.   float maxTemp = -100;
1180.   int validPixels = 0;
1181.
1182.   for (int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
1183.     float adjustedTemp = pixels[i] * DISTANCE_FACTOR + tempOffset;
1184.
1185.     if (isNoiseFiltered(adjustedTemp)) {
1186.       sumTemp += adjustedTemp;
1187.       maxTemp = max(maxTemp, adjustedTemp);
1188.       validPixels++;
1189.
```

```
1190.       float dynamicThreshold = ambientBaseline +
      MEDIUM_OBJECT_THRESHOLD;
1191.
1192.       if (adjustedTemp > dynamicThreshold) {
1193.         redPixelCount++;
1194.       }
1195.     }
1196.   }
1197.
1198.   if (validPixels > 0) {
1199.     avgTemp = sumTemp / validPixels;
1200.     redPercentage = (float)redPixelCount / validPixels * 100.0;
1201.   } else {
1202.     avgTemp = ambientBaseline;
1203.     redPercentage = 0;
1204.   }
1205.
1206.   // LANGKAH 3: ENHANCED OBJECT DETECTION
1207.   bool smallObjectDetected = detectSmallObject(pixels,
      ambientBaseline);
1208.   bool mediumObjectDetected = detectMediumObject(pixels,
      ambientBaseline);
1209.   bool largeObjectDetected = detectLargeObject(pixels,
      ambientBaseline);
1210.
1211.   determineDetectionMode(smallObjectDetected, mediumObjectDetected,
      largeObjectDetected);
1212.
1213.   // LANGKAH 4: KNN CLASSIFICATION
1214.   float dataUji[2] = {redPercentage, 100.0 - redPercentage};
1215.
1216.   float jarakTemp[10];
1217.   int labelTemp[10];
1218.
1219.   for (int i = 0; i < 10; i++) {
1220.     jarakTemp[i] = ecludian(dataLatih[i][0], dataLatih[i][1],
      dataUji[0], dataUji[1]);
1221.     labelTemp[i] = label[i];
1222.   }
1223.
1224.   for (int i = 0; i < 10; i++) {
1225.     for (int j = i + 1; j < 10; j++) {
1226.       if (jarakTemp[i] > jarakTemp[j]) {
1227.         float tmpJarak = jarakTemp[i];
1228.         jarakTemp[i] = jarakTemp[j];
1229.         jarakTemp[j] = tmpJarak;
1230.
1231.         int tmpLabel = labelTemp[i];
1232.         labelTemp[i] = labelTemp[j];
1233.         labelTemp[j] = tmpLabel;
1234.       }
1235.     }
```

```
1236.   }
1237.
1238.   k = neighborOptions[currentNeighborIndex];
1239.
1240.   count1 = 0;
1241.   count0 = 0;
1242.
1243.   for (int n = 0; n < k; n++) {
1244.     if (labelTemp[n] == 1)
1245.       count1++;
1246.     else
1247.       count0++;
1248.   }
1249.
1250.   // LANGKAH 5: ENHANCED CLASSIFICATION DECISION
1251.   bool tempDanger = false;
1252.
1253.   // ENHANCED: Priority detection based on object type
1254.   if (smallObjectDetected) {
1255.     classification = "BAHAYA";
1256.     tempDanger = true;
1257.   } else if (mediumObjectDetected) {
1258.     classification = "BAHAYA";
1259.     tempDanger = true;
1260.   } else if (largeObjectDetected) {
1261.     classification = "BAHAYA";
1262.     tempDanger = true;
1263.   } else if (count1 > count0) {
1264.     classification = "BAHAYA";
1265.     tempDanger = true;
1266.   } else if (count1 < count0) {
1267.     classification = "AMAN";
1268.     tempDanger = false;
1269.   } else {
1270.     classification = "TIDAK TERIDENTIFIKASI";
1271.     tempDanger = false;
1272.   }
1273.
1274.   // Additional threshold check
1275.   float absoluteDangerThreshold = ambientBaseline +
      DANGER_THRESHOLD;
1276.   if (avgTemp >= absoluteDangerThreshold) {
1277.     classification = "BAHAYA_SUHU_TINGGI";
1278.     tempDanger = true;
1279.   }
1280.
1281.   // LANGKAH 6: ADAPTIVE STABILITY SYSTEM
1282.   int requiredStability = getRequiredStability();
1283.
1284.   if (tempDanger) {
1285.     consecutiveDanger++;
1286.     consecutiveSafe = 0;
```

```
1287.   } else {
1288.     consecutiveSafe++;
1289.     consecutiveDanger = 0;
1290.   }
1291.
1292.   // Tentukan apakah klasifikasi sudah stabil
1293.   if (consecutiveDanger >= requiredStability) {
1294.     stableClassification = true;
1295.     isDanger = true;
1296.   } else if (consecutiveSafe >= requiredStability) {
1297.     stableClassification = true;
1298.     isDanger = false;
1299.   } else {
1300.     stableClassification = false;
1301.     isDanger = false;
1302.   }
1303.
1304.   // LANGKAH 7: UPDATE DISPLAY
1305.   updateOLEDDisplay(avgTemp, redPercentage, classification, k,
      count1, count0);
1306.
1307.   currentNeighborIndex = (currentNeighborIndex + 1) %
      neighborCount;
1308.
1309.   lastClassification = classification;
1310.   lastredPercentage = redPercentage;
1311.   lastCount1 = count1;
1312.   lastCount0 = count0;
1313.
1314.   // LANGKAH 8: ENHANCED DEBUG OUTPUT
1315.   Serial.println("=== ENHANCED DETECTION DEBUG ===");
1316.   Serial.println("Detection Mode: " + detectionMode);
1317.   Serial.println("Required Stability: " +
      String(requiredStability));
1318.   Serial.println("Max Pixel Temp: " + String(maxPixelTemp, 1) +
      "°C");
1319.   Serial.println("Temp Gradient: " + String(tempGradient, 1) +
      "°C");
1320.   Serial.println("Hot Spot Count: " + String(hotSpotCount));
1321.   Serial.println("Hot Spot Intensity: " + String(hotSpotIntensity,
      1) + "°C");
1322.   Serial.println("Peak Detected: " + String(peakDetected ? "YES" :
      "NO"));
1323.   Serial.println("Small Object: " + String(smallObjectDetected ?
      "YES" : "NO"));
1324.   Serial.println("Medium Object: " + String(mediumObjectDetected ?
      "YES" : "NO"));
1325.   Serial.println("Large Object: " + String(largeObjectDetected ?
      "YES" : "NO"));
1326.   Serial.println("Avg Temp: " + String(avgTemp, 1) + "°C");
1327.   Serial.println("Baseline: " + String(ambientBaseline, 1) + "°C");
1328.   Serial.println("Hot %: " + String(redPercentage, 1) + "%");
```

```
1329.   Serial.println("Raw Classification: " + classification);
1330.   Serial.println("Consecutive Danger: " +
      String(consecutiveDanger));
1331.   Serial.println("Consecutive Safe: " + String(consecutiveSafe));
1332.   Serial.println("Stable: " + String(stableClassification ? "YES" :
      "NO"));
1333.   Serial.println("Final Danger: " + String(isDanger ? "YES" :
      "NO"));
1334.   Serial.println("================================");
1335.
1336.   // LANGKAH 9: CHECK DISCORD
1337.   checkDiscordNotifications(isDanger, redPercentage,
      classification, avgTemp);
1338.
1339.   // LANGKAH 10: LOGGING
1340.   logClassificationResult(redPercentage, count1, count0,
      classification);
1341. }
1342.
1343. // ===== WEB INTERFACE FUNCTIONS =====
1344. void sendThermalData(WiFiClient client, float pixels[], float
      avgTemp, float redPercentage, bool isDanger, int k, int count1, int
      count0, String classification) {
1345.   StaticJsonDocument<2048> doc;
1346.   JsonArray pixelArray = doc.createNestedArray("pixels");
1347.
1348.   float minTemp = 100;
1349.   float maxTemp = -100;
1350.
1351.   for(int i = 0; i < AMG88xx_PIXEL_ARRAY_SIZE; i++) {
1352.     float adjustedTemp = pixels[i] * DISTANCE_FACTOR + tempOffset;
1353.     if(adjustedTemp < minTemp) minTemp = adjustedTemp;
1354.     if(adjustedTemp > maxTemp) maxTemp = adjustedTemp;
1355.     pixelArray.add(adjustedTemp);
1356.   }
1357.
1358.   doc["averageTemp"] = avgTemp;
1359.   doc["hotPercentage"] = redPercentage;
1360.   doc["isDanger"] = isDanger;
1361.   doc["minTemp"] = minTemp;
1362.   doc["maxTemp"] = maxTemp;
1363.   doc["kValue"] = k;
1364.   doc["class1Count"] = count1;
1365.   doc["class0Count"] = count0;
1366.   doc["classification"] = classification;
1367.
1368.   // Enhanced detection info
1369.   doc["detectionMode"] = detectionMode;
1370.   doc["maxPixelTemp"] = maxPixelTemp;
1371.   doc["tempGradient"] = tempGradient;
1372.   doc["hotSpotCount"] = hotSpotCount;
1373.   doc["hotSpotIntensity"] = hotSpotIntensity;
```

```
1374.   doc["peakDetected"] = peakDetected;
1375.   doc["requiredStability"] = getRequiredStability();
1376.
1377.   doc["consecutiveDanger"] = consecutiveDanger;
1378.   doc["consecutiveSafe"] = consecutiveSafe;
1379.   doc["stableClassification"] = stableClassification;
1380.   doc["autoCalibrationDone"] = autoCalibrationDone;
1381.   doc["ambientBaseline"] = ambientBaseline;
1382.
1383.   unsigned long currentTime = millis();
1384.   unsigned long timeUntilNext = SCHEDULED_INTERVAL - (currentTime -
      lastScheduledNotification);
1385.   if (timeUntilNext > SCHEDULED_INTERVAL) timeUntilNext = 0;
1386.
1387.   doc["nextReportTime"] = timeUntilNext / 1000;
1388.   doc["progressPercent"] = ((float)(SCHEDULED_INTERVAL -
      timeUntilNext) / SCHEDULED_INTERVAL) * 100;
1389.
1390.   doc["wifiConnected"] = wifiConnected;
1391.   doc["wifiUptime"] = wifiConnected ? (millis() -
      wifiConnectedTime) / 1000 : 0;
1392.   doc["discordEnabled"] = discordEnabled; // Discord status
1393.
1394.   client.println("HTTP/1.1 200 OK");
1395.   client.println("Content-Type: application/json");
1396.   client.println("Access-Control-Allow-Origin: *");
1397.   client.println("Cache-Control: no-cache, no-store, must-
      revalidate");
1398.   client.println("Connection: close");
1399.
1400.   String jsonOutput;
1401.   serializeJson(doc, jsonOutput);
1402.   client.print("Content-Length: ");
1403.   client.println(jsonOutput.length());
1404.   client.println();
1405.   client.print(jsonOutput);
1406. }
1407.
1408. void handleWebRequest(WiFiClient client, float pixels[], float
      avgTemp, float redPercentage, bool isDanger, int k, int count1, int
      count0, String classification) {
1409.   String currentLine = "";
1410.   String requestLine = "";
1411.   unsigned long startTime = millis();
1412.
1413.   while (client.connected() && (millis() - startTime < 2000)) {
1414.     if (client.available()) {
1415.       char c = client.read();
1416.
1417.       if (c == '\n') {
1418.         if (currentLine.length() == 0) {
1419.           if (requestLine.indexOf("GET /thermal-data") >= 0) {
```

```
1420.             sendThermalData(client, pixels, avgTemp, redPercentage,
      isDanger, k, count1, count0, classification);
1421.           } else {
1422.             client.println("HTTP/1.1 200 OK");
1423.             client.println("Content-Type: text/html");
1424.             client.println("Connection: close");
1425.             client.println();
1426.             client.print(WEBPAGE);
1427.           }
1428.           break;
1429.         } else {
1430.           if (currentLine.startsWith("GET")) {
1431.             requestLine = currentLine;
1432.           }
1433.           currentLine = "";
1434.         }
1435.       } else if (c != '\r') {
1436.         currentLine += c;
1437.       }
1438.     }
1439.   }
1440. }
1441.
1442. // ===== MAIN MONITORING SYSTEM =====
1443. void runMonitoringSystem() {
1444.   Serial.println("\n===== ENHANCED THERMAL MONITORING ACTIVE
      =====");
1445.   Serial.println("Enhanced Features:");
1446.   Serial.println("   🎯 Small object detection (korek api
      optimized)");
1447.   Serial.println("   🔥 Peak temperature detection");
1448.   Serial.println("   📊 Multi-level thresholds
      (SMALL/MEDIUM/LARGE)");
1449.   Serial.println("   ⚡ Adaptive stability (2/3/4 readings)");
1450.   Serial.println("   🌡️ Temperature gradient analysis");
1451.   Serial.println("   📍 Hot spot counting & intensity");
1452.   Serial.println("Discord Notifications (Adapted from may26c):");
1453.   Serial.println("   📅 Status Reports: Every 1 hour");
1454.   Serial.println("   🚨 Danger Alerts: Clean format with enhanced
      details");
1455.   Serial.println("   ⏱️ Smart cooldown timing (5s/10s/60s)");
1456.   Serial.println("WiFi Status: " + String(wifiConnected ?
      "CONNECTED" : "OFFLINE"));
1457.
      Serial.println("================================================");
1458.
1459.   displayOLEDMessage("Enhanced Monitor",
1460.                      wifiConnected ? "IP: " +
      WiFi.localIP().toString() : "Offline Mode");
1461.   delay(5000);
1462.
```

```
1463.   initWiFiStability();
1464.
1465.   unsigned long lastPrint = 0;
1466.
1467.   while (true) {
1468.     unsigned long currentTime = millis();
1469.
1470.     if (currentTime - lastWiFiCheck > WIFI_CHECK_INTERVAL) {
1471.       checkWiFiStability();
1472.     }
1473.
1474.     if (currentTime - lastPrint > 30000) {
1475.       Serial.println("Enhanced Status: WiFi=" +
      String(wifiConnected ? "OK" : "NO") +
1476.                      ", Discord=" + String(discordEnabled ? "OK" :
      "NO") +
1477.                      ", AutoCalib=" + String(autoCalibrationDone ?
      "DONE" : "PENDING") +
1478.                      ", Stable=" + String(stableClassification ?
      "YES" : "NO") +
1479.                      ", Mode=" + detectionMode +
1480.                      ", Peak=" + String(peakDetected ? "YES" : "NO")
      +
1481.                      ", HotSpots=" + String(hotSpotCount) +
1482.                      ", Uptime=" + String(currentTime/1000) + "s");
1483.       lastPrint = currentTime;
1484.     }
1485.
1486.     // Calibration controls
1487.     if (digitalRead(CAL_UP_PIN) == LOW) {
1488.       tempOffset += CAL_STEP;
1489.       displayOLEDMessage("Calibration", "Offset: " +
      String(tempOffset) + "C");
1490.       delay(250);
1491.     }
1492.
1493.     if (digitalRead(CAL_DOWN_PIN) == LOW) {
1494.       tempOffset -= CAL_STEP;
1495.       displayOLEDMessage("Calibration", "Offset: " +
      String(tempOffset) + "C");
1496.       delay(250);
1497.     }
1498.
1499.     if (digitalRead(CAL_SAVE_PIN) == LOW) {
1500.       saveCalibration();
1501.       displayOLEDMessage("Calibration", "Saved!");
1502.       delay(500);
1503.     }
1504.
1505.     // Thermal monitoring
1506.     float pixels[AMG88xx_PIXEL_ARRAY_SIZE];
1507.     amg.readPixels(pixels);
```

```
1508.
1509.     float avgTemp;
1510.     float redPercentage;
1511.     String classification;
1512.     int k;
1513.     int count1, count0;
1514.     bool isDanger;
1515.
1516.     doKNNClassification(pixels, avgTemp, redPercentage,
      classification, k, count1, count0, isDanger);
1517.
1518.     // Handle web requests
1519.     if (wifiConnected) {
1520.       WiFiClient client = server.available();
1521.       if (client) {
1522.         handleWebRequest(client, pixels, avgTemp, redPercentage,
      isDanger, k, count1, count0, classification);
1523.         client.stop();
1524.       }
1525.     }
1526.
1527.     delay(100);
1528.   }
1529. }
1530.
1531. void setup() {
1532.   Serial.begin(115200);
1533.   Serial.println("\n===== ENHANCED THERMAL SYSTEM WITH ADAPTED
      DISCORD & LCD =====");
1534.   Serial.println("🔧 ADAPTATIONS FROM MAY26C:");
1535.   Serial.println("    ✅ Clean Discord message format");
1536.   Serial.println("    ✅ Simplified LCD display");
1537.   Serial.println("    ✅ Smart cooldown timing (5s/10s/60s)");
1538.   Serial.println("    ✅ Embed support with colors");
1539.   Serial.println("    ✅ Percentage visualization");
1540.   Serial.println("🎯 Enhanced Detection: SMALL/MEDIUM/LARGE
      objects");
1541.   Serial.println("⚡ Stability Requirements: 2/3/4 readings
      respectively");
1542.   Serial.println("🎮 Discord Settings:");
1543.   Serial.println("    🚨 Emergency (≥80°C): 5 seconds");
1544.   Serial.println("    ⚠️ Danger: 10 seconds");
1545.   Serial.println("    ✅ Normal: 60 seconds");
1546.   Serial.println("    🔥 Critical (≥70%): 5 seconds");
1547.
      Serial.println("==================================================
      =");
1548.
1549.   display.begin();
1550.   displayOLEDMessage("Enhanced System", "Adapted Messages");
1551.   delay(2000);
```

```
1552.
1553.   pinMode(CAL_UP_PIN, INPUT_PULLUP);
1554.   pinMode(CAL_DOWN_PIN, INPUT_PULLUP);
1555.   pinMode(CAL_SAVE_PIN, INPUT_PULLUP);
1556.
1557.   Serial.println("Initializing thermal sensor...");
1558.   displayOLEDMessage("Initializing", "Thermal Sensor");
1559.
1560.   Wire.begin();
1561.   if (!amg.begin()) {
1562.     Serial.println("AMG8833 sensor not found!");
1563.     displayOLEDMessage("ERROR", "AMG8833 Not Found");
1564.     while(1);
1565.   }
1566.
1567.   Serial.println("Initializing SD Card...");
1568.   displayOLEDMessage("Initializing", "SD Card");
1569.
1570.   if (!SD.begin(chipSelect)) {
1571.     Serial.println("SD Card initialization failed! Retrying...");
1572.     displayOLEDMessage("Error", "SD Card Failed");
1573.
1574.     for (int i = 0; i < 3; i++) {
1575.       delay(1000);
1576.       if (SD.begin(chipSelect)) {
1577.         sdCardOK = true;
1578.         break;
1579.       }
1580.     }
1581.   } else {
1582.     sdCardOK = true;
1583.   }
1584.
1585.   if (sdCardOK) {
1586.     loadCalibration();
1587.
1588.     Serial.println("Loading training data...");
1589.     displayOLEDMessage("Loading", "Training Data");
1590.
1591.     if (loadTrainingData()) {
1592.       systemReady = true;
1593.     } else {
1594.       while (!trainingSelesai) {
1595.         displayOLEDMessage("Waiting", "Insert SD with data");
1596.         delay(2000);
1597.
1598.         if (tryLoadTrainingData(3)) {
1599.           systemReady = true;
1600.           break;
1601.         }
1602.       }
1603.     }
```

```
1604.   }
1605.
1606.   // WiFi Connection
1607.   Serial.println("=== WiFi Connection Process ===");
1608.   displayOLEDMessage("WiFi Setup", "Connecting...");
1609.
1610.   initWiFiStability();
1611.
1612.   if (tryAllWiFiNetworks()) {
1613.     wifiConnected = true;
1614.     server.begin();
1615.
1616.     printWiFiStatus();
1617.
1618.     displayOLEDMessage("WiFi Connected!",
1619.                        "IP: " + WiFi.localIP().toString());
1620.
1621.     Serial.println("Testing Discord connection...");
1622.     displayOLEDMessage("Testing Discord", "Connection...");
1623.
1624.     String testMessage = "🤖 **Enhanced Thermal Monitor
      Started**\n\n";
1625.     testMessage += "✅ **System Status:** Online and ready\n";
1626.     testMessage += "🔧 **Features Active:**\n";
1627.     testMessage += "• Enhanced small object detection\n";
1628.     testMessage += "• Multi-level threshold system\n";
1629.     testMessage += "• Adaptive stability control\n";
1630.     testMessage += "• Smart cooldown notifications\n\n";
1631.     testMessage += "📅 **Notification Schedule:**\n";
1632.     testMessage += "• Status reports: Every 1 hour\n";
1633.     testMessage += "• Danger alerts: Real-time (with cooldown)\n";
1634.     testMessage += "• Emergency alerts: Immediate\n\n";
1635.     testMessage += "_System ready for enhanced thermal
      monitoring_";
1636.
1637.     if (sendDiscordNotification(testMessage, "AMAN", 0.0, 25.0)) {
1638.       Serial.println("Discord integration successful!");
1639.       displayOLEDMessage("Discord Ready", "Adapted Style");
1640.
1641.       lastScheduledNotification = millis();
1642.     } else {
1643.       Serial.println("Discord test failed - continuing anyway");
1644.       displayOLEDMessage("Discord Warning", "Test Failed");
1645.     }
1646.
1647.   } else {
1648.     Serial.println("=== WiFi Connection Failed ===");
1649.     Serial.println("Continuing in OFFLINE mode...");
1650.
1651.     displayOLEDMessage("WiFi Failed!", "Offline mode");
1652.
1653.     wifiConnected = false;
```

```
    discordEnabled = false;

    delay(3000);
  }

  if (systemReady) {
    displayOLEDMessage("Enhanced Ready", "Auto-calibrating...");
    delay(1000);
  } else {
    displayOLEDMessage("Error", "Training Data Needed");
  }

  delay(1000);
}

void loop() {
  if (systemReady && trainingSelesai) {
    runMonitoringSystem();
  } else {
    displayOLEDMessage("Error", "Training Data Needed");

    if (tryLoadTrainingData(3)) {
      systemReady = true;
      displayOLEDMessage("Enhanced Ready", "Auto-calibrating...");
      delay(1000);
    } else {
      delay(5000);
    }
  }
}
```