ENEL 500A- Final Report

Computer, Electrical, and Software Engineering Design Team

Instructor: Dr. Hamidreza Zareipour
Sponsor: Simply Embedded
Sponsor Representative: Chris Karaplis
chris.karaplis@simplyembedded.ca
Academic Advisor: Dr. Sara Saeedi

Group 4
Project Manager: Samira Khan (30113862)
samira.khan@ucalgary.ca

Team:
Ammar Elzeftawy (30113872)
Thevin Mahawatte (30130509)
Brandon McGee (30125635)
Jonas Wong (30096750)
Zaima Zubaida (30141690)

Table of Contents

1. Glossary

Term	Definition
IOT (Internet of Things)	A network of physical devices embedded with sensors and software to collect and exchange data.
MCU (Microcontroller Unit)	A compact integrated circuit designed to govern specific operations in embedded systems
I2C Protocol	A two-wire serial communication protocol used to connect low-speed peripherals to microcontrollers.
LTE-M	A low-power wide-area cellular technology optimized for IoT devices.
MQTT (Message Queuing Telemetry Transport)	A lightweight messaging protocol for small sensors and mobile devices optimized for low-bandwidth networks.
InfluxDB	A time-series database optimized for storing and querying data with timestamps, such as sensor readings.
Django	A high-level Python web framework that enables rapid development of secure and maintainable websites.
SDK (Software Development Kit)	A collection of software development tools for creating applications on a specific platform or hardware

2. Executive Summary

This report presents the final design and implementation of a wildfire monitoring system developed by Toasty Transistors, in collaboration with Simple Embedded. The project aims to provide an innovative, real-time monitoring solution that enhances early wildfire detection and prevention while optimizing power consumption and data accuracy.

The increasing frequency and severity of wildfires highlight the need for efficient, real-time wildfire monitoring systems capable of detecting environmental changes before fires escalate. Traditional monitoring methods often lack precision, responsiveness, or accessibility in remote areas, leading to delayed detection and intervention. This project seeks to bridge that gap by developing a cost-effective, low-power solution that provides accurate, real-time environmental data to support early warning systems and wildfire prevention efforts.

The system consists of sensor nodes equipped with humidity, temperature, particulate matter, and light intensity sensors, which continuously collect environmental data. These nodes communicate with a central gateway using optimized wireless protocols, ensuring seamless data transmission to a dashboard interface for real-time monitoring. The firmware was designed to efficiently process and transmit data while minimizing power consumption, allowing for sustained operation in remote locations. A structured iterative design process was followed, focusing on hardware and software integration, gateway communication, and power management strategies. Additionally, a risk assessment and mitigation plan was developed to address potential hardware failures, network disruptions, and environmental challenges affecting long-term sustainability.

A structured timeline with well defined milestones and deliverables guided the project's progression, ensuring steady development and timely completion. Technical specifications, cost estimation, and ethical considerations were carefully analyzed to maintain compliance with regulatory and financial constraints. The final validation tests confirmed the system's effectiveness in meeting both the sponsor's expectations and the initial problem statement.

By completing this project, our team has demonstrated a higher level skill in embedded systems design, software development, and project management, delivering a functional and scalable

engineering solution. The findings and lessons learned contribute valuable insights to the broader field of wildfire monitoring technology, with potential for further improvements.

3. Project Motivation and Objectives

Forest fires are rampant every summer in densely forested areas in Canada. Particularly in regions like Alberta and British Columbia, these fires not only affect the environment and wildlife but also put nearby towns and municipalities at risk. When a forest fire escalates beyond control, it can lead to the displacement of entire communities and may take decades to recover. This chain of events highlights the need for a fast and accurate way to detect fires early to stop them from reaching dangerous levels.

Currently, the early detection of forest fires relies heavily on human intervention. Fire watchtowers are manned during the dry season by individuals who monitor for signs of smoke and flames. While this approach is effective in certain cases, there is large potential for human error. The responsibility of spotting an initial spark in miles of dense forest is challenging. Humans are prone to losing focus while on watch and making errors. Even the most attentive individual requires breaks, risks fatigue and is susceptible to misjudgments.

Beyond the operational risks, maintaining a fire watch program increases logistical costs. Transporting, housing, and providing food and supplies for the workers stationed in remote areas consumes resources that could otherwise be directed toward fire fighting equipment or prevention programs. The unpredictability of human error, along with the costs associated with sustaining these watch posts, shows the need for new, automated solutions that can improve early detection and response to help protect nature and nearby communities from uncontrolled wildfires.

The proposed solution is to introduce a network of sensors for the early detection of forest fires in boreal regions of Alberta and British Columbia. The system contains sensory nodes to measure quantitative data and uses gateway nodes to distribute the data to a main server downstream to be processed on a visual dashboard. Each sensory node will extract and measure at minimum the temperature, humidity, and infrared readings of its surrounding area in real time. The raw data is delivered to a gateway node, which collects similar data from other designated sensory nodes assigned to it. The gateway node performs simple transformations to the raw data

it collects in batches then transfers that batch to a main server which loads the data into a dashboard for the end user.

This solution is cost effective from a logistical standpoint and from a technical perspective, as it reduces the requirement for inconsistent human monitoring while providing a reliable, continuous stream of real-time data. The forest detection network has the advantage of being able to operate around the clock, requiring minimal maintenance and resources compared to watchtowers operated by humans. Integration of the selected sensors, with room for additional sensors if need be, allows the system to quickly detect indicators of fire. The historical data collected over time from the system allows for the eventual implementation of a predictive model to identify high-risk zones, a feature that further enhances placement (or displacement) of sensory and gateway nodes to increase cost effectiveness. This method of continuous monitoring allows for a swift and rapid response when detecting forest fires, ultimately containing them before they get out of control.

4. Methodology and Design

4.1. Hardware Methodology

This section outlines the hardware methodology employed in the development of the final product. This section will include a detailed account of the hardware design choices, processes, tools and components in creating our product. The methodologies discussed in this section will include hardware design choices, hardware system architecture and Printed Circuit Board (PCB) design.

I. Sensor Choice

The sensor implementations in KiCad are given below as well as the implementation notes in the design:

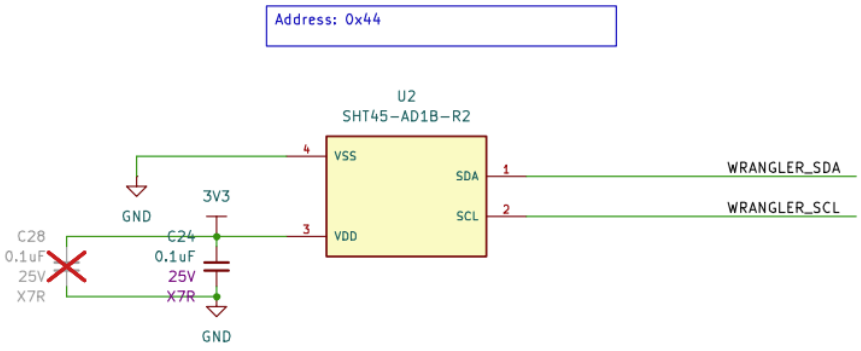

Figure 1: Humidity/Temperature Implementations

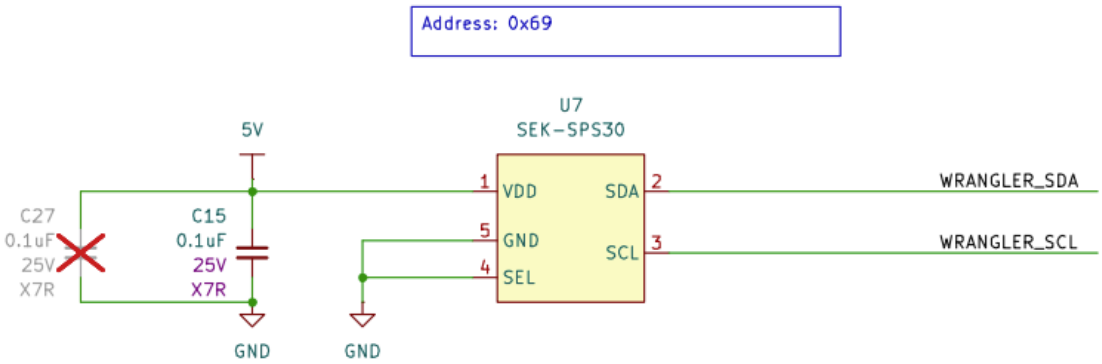

Figure 2: Particulate Matter Sensor Implementation

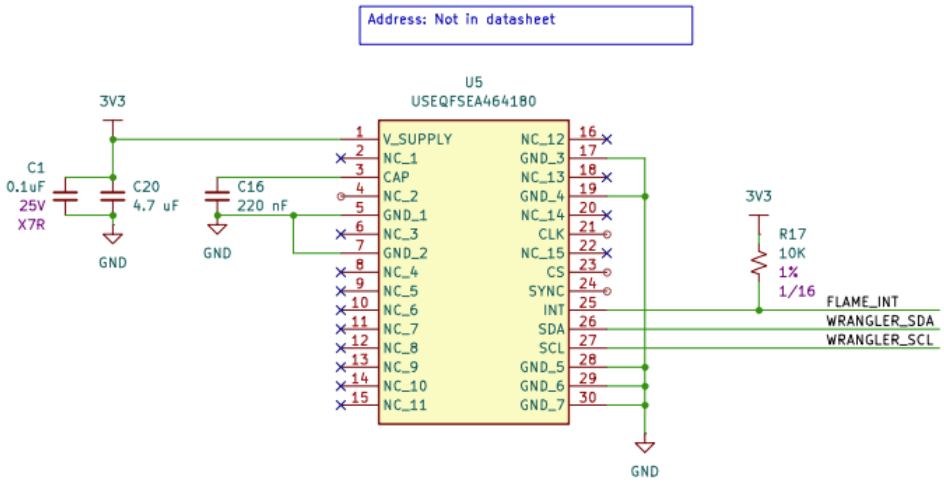

Figure 3: IR Sensor Implementation

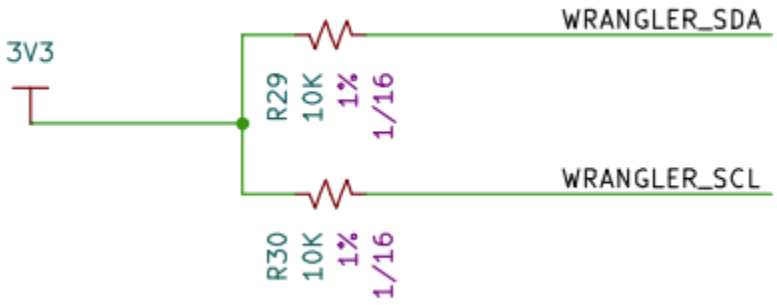

Figure 4: Pull-up Resistors Connected to MCU

Sensor	Model Used	Implementation Notes
Particulate Matter	Sensiorion SEK-SPS30	Measures the concentration of particulate matter in the air like PM1.0, PM2.5 and PM10. Increase in particulates in the air could be an indication of the onset or growth of a fire in the region.

IR Sensor	USEQFSEA464180	Employs a long-distance and wide-view (>90°) pyroelectric infrared sensor to detect flame presence in the vicinity of sensor nodes. The sensor outputs analog signals corresponding to infrared radiation from flames, facilitating early fire detection.
Humidity/Temperature	Sensirion SHT-45[4]	Used to measure the ambient temperature and humidity of the environment. Any abnormalities, such as extremely low humidity and high temperature can be used for wildfire prediction.

II. Hardware Design- Power Supplies

The hardware design consists of some key components which include the sensors, power supplies, battery management and MCU. The power supplies were chosen based on the sensor requirements. The sensors we chose to include in our prototype include a temperature sensor and humidity sensor (SHT-45), IR sensor and particulate matter sensor (SEK-SPS30). Most of our sensors require 3.3V, however, the particulate matter sensor requires 5V to operate. We also needed to include a 1.8V supply for the battery fuel gauge (BQ35100-pwr). Since we are using 3.6V batteries, we needed to include two voltage regulators. One voltage regulator was used to boost the voltage to 5V for the particulate matter sensors. Whereas, the second voltage regulator lowers the input voltage

to 3.3V for the rest of the sensors and the MCU. The 1.8 V was supplied using a small LDO regulator which directly converts 3.3V to 1.8V through a simple topology. The following figures include the KiCad symbols which is where we designed our PCB.

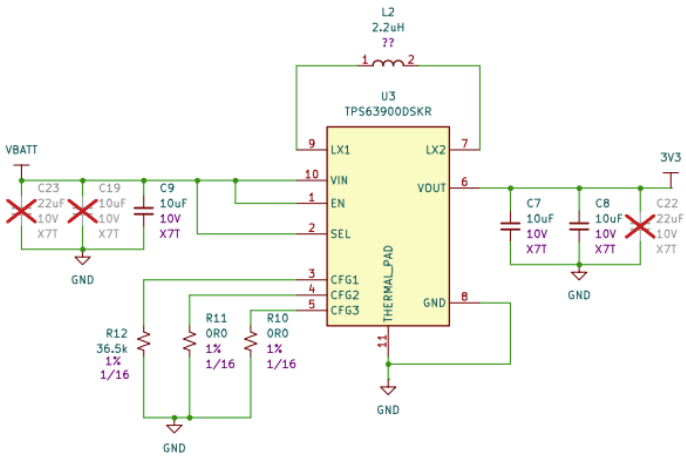

Figure 5: 3.3V Regulator Design

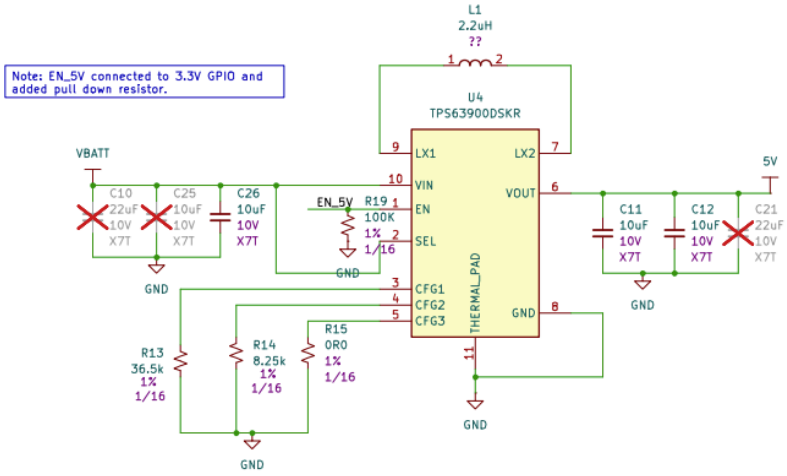

Figure 6: 5.0V Regulator Design

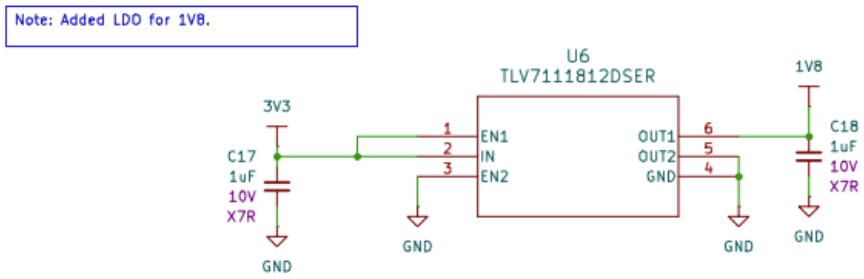

Figure 7: 1.8V LDO Design

Note the enable (EN_5V) for the 5V is initially pulled down to zero using a 100k pull down resistor. This was done to save power and to stagger the sensor activation. In the current implementation of this project, we have the enable set to an arbitrary value, however, this can be adjusted in the field to enable the particulate matter sensor. As mentioned before, this design choice was implemented in order to save power in the field. The temperature and IR sensors are the primary sensors detecting a fire.

III. Hardware Design- Battery Management

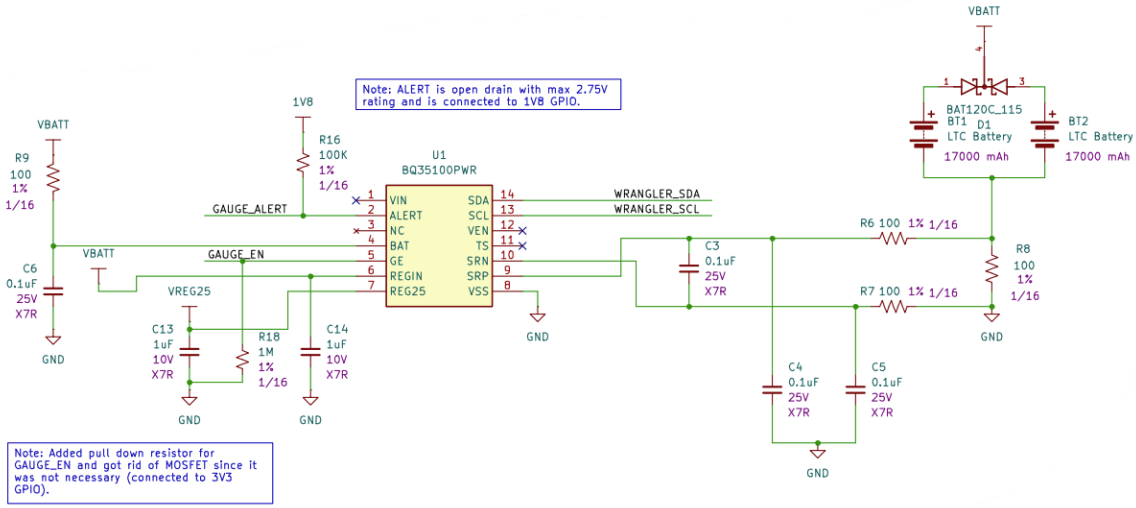

Figure 8: Battery Fuel Gauge

We selected the BQ35100-pwr battery fuel gauge due to its compatibility with lithium thionyl chloride chemistry. The main objective of adding a battery fuel gauge is to monitor the state of the battery. The fuel gauge was implemented hardware wise,

however, as this was not an agreed upon requirement from our sponsor, we did not include this in our firmware design. In future works the fuel gauge can be seamlessly integrated into the firmware. In our design we used D-cell lithium thionyl chloride batteries which have about 17-19 Ah of capacity. In the figure above, we included the option of increasing the capacity by placing two batteries in parallel. In order to prevent one battery current from flowing into another battery, reverse current protection using a Schottky diode was implemented into the design. Ideally, our product would last an entire summer, up to 4 months in the wilderness, without intervention as that adds to our products cost and maintenance. Due to time constraints we were not able to test the full capacity of our product, however, by our estimations, this product should last up to 4 months if it operates in sleep mode or idle mode most of the time. In future interactions of this project, the fuel gauge would be added to the firmware to enhance the maintainability of this project making it more robust.

IV. MCU Implementation and Hardware Features

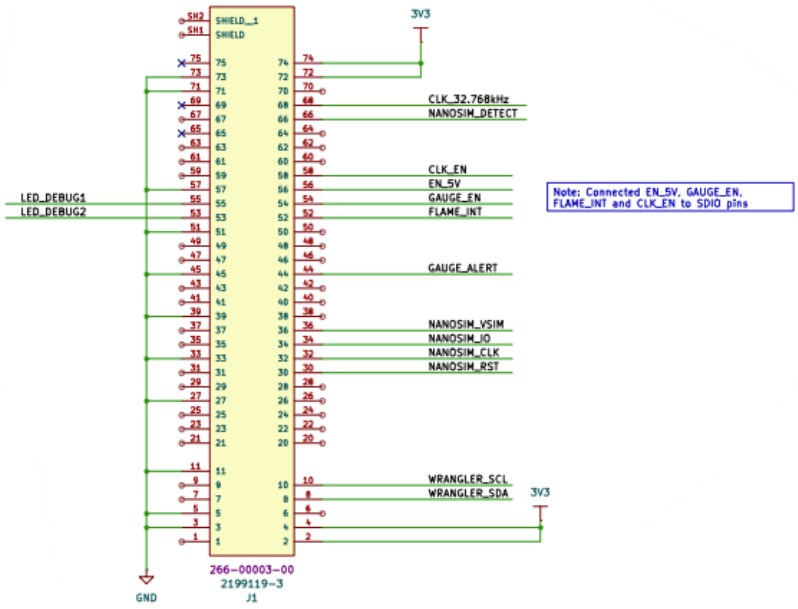

Figure 9: MCU Implementation

The MCU used in this project was the MIMXRT1051-NXP which our sponsors have on

the “Wrangler” device they have developed for IOT development. The pin connections for the MCU are given above, however, since their device is closed source, we aren’t able to include further details in this project. Our sensors were interfaced with the MCU using I2C protocol. The MCU also controls various devices such as the fuel gauge, 5V supply and external clock. We also included two debug LEDs in case we needed to debug and test the functionality of the MCU when developing our firmware.

For external communication, we integrated a nano-SIM card holder and an LTE antenna into our device. This setup enabled us to connect to cellular networks via an LTE-compatible modem, allowing for real-time data transmission and reception over the LTE network. This hardware was easy to install as it was included on the “Wrangler” that our sponsors developed. All we had to do was choose an appropriate antenna and attach a sim card holder.

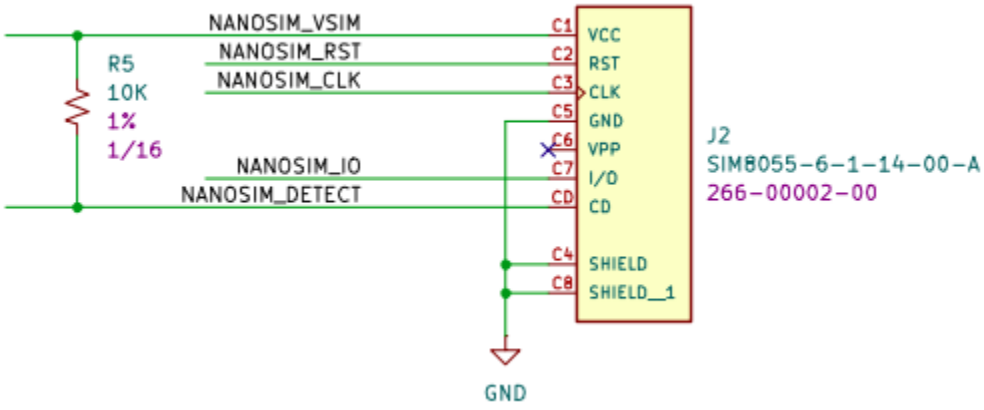

Figure 10: Nano-sim Holder Implementation

We included some other features onto our PCB which we did not have to include in our project as per our goal from this project such as the external clock. This would allow our system to run at much higher speeds if needed, however, this application was only requested for hardware purposes and not firmware integration.

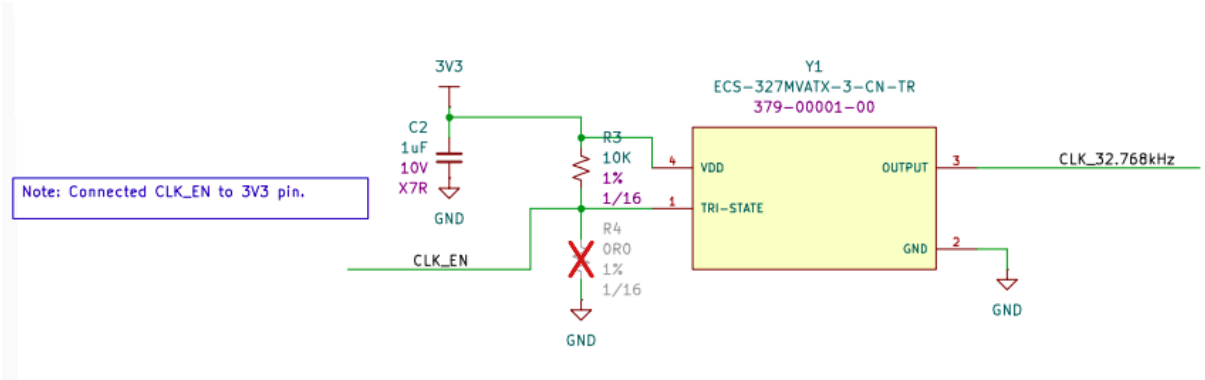

Figure 11: External Clock Implementation

V. PCB Design

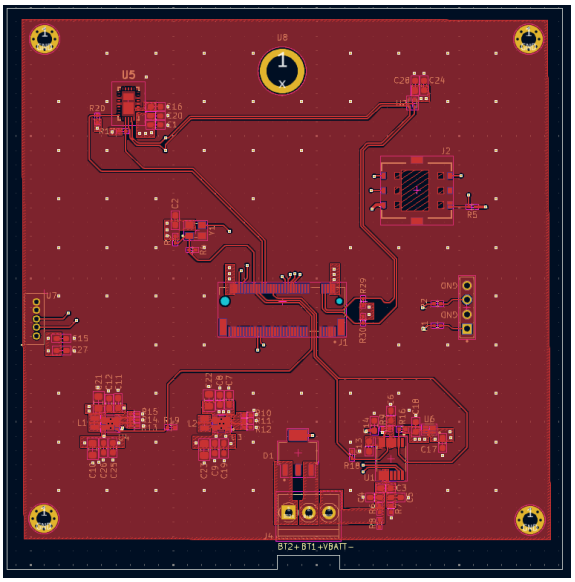

Figure 12: PCB Layout (Top Layer)

There were a few considerations we had to make for the PCB layout. The first one was picking an appropriate size for the PCB. We found that 10cmx10cm was small enough to meet loose size constraints which we set as a non-specific standard to create the smallest board possible. The size of our PCB was not a huge design consideration for our project but we still minimized it to maintain cost and portability for ease of deployment. Next, we had to make design constraints to ensure that the quality and functionality of the board was reliable. This meant that we would need to have solid ground planes in order to prevent any signal integrity issues, such as noise coupling especially when considering

I2C. Maintaining a solid going plane also reduced EMI to ensure a low impedance return path for the signal. This consideration was especially important to our sponsor although EMI is not a big deal in our project. We implemented our project on a four layer board to increase reliability and performance. The top layer consists of signals, the second layer was a solid GND plane, the third layer was a solid power plane (3.3V) and the last layer was another signal layer. This stack allowed us to implement more solid GND planes.

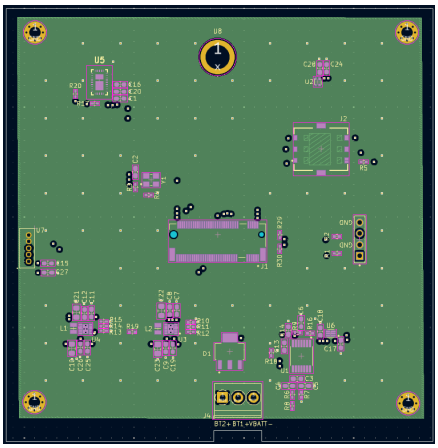

Figure 13: Second Layer Solid Ground Plane

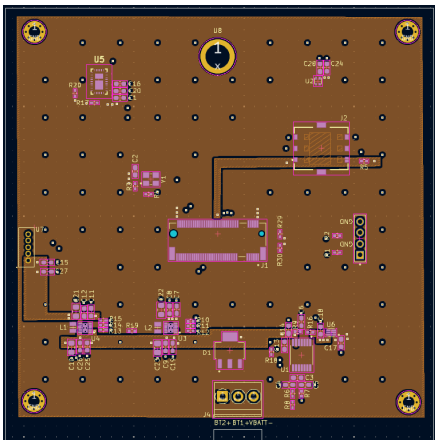

Figure 14: Third Layer Solid Power Plane

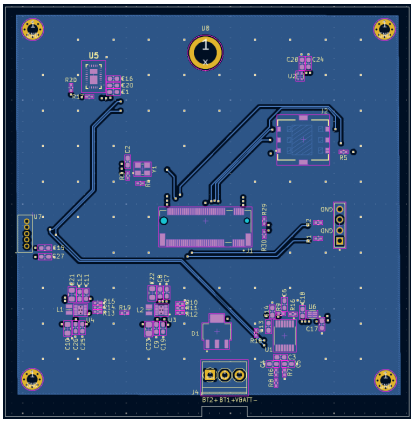

Figure 15: Bottom Layer Signal Layer

Other design choices we made for the PCB layout includes wider trace widths for power signals and trace spacing. This was important to consider since our product is low power and has high frequency signal lines so we want to reduce voltage drops. Additionally, we routed power using vias instead of traces in most areas for lower resistance and inductance in the power paths. We also incorporated via stitching as seen in the figures above. This was done to solidify the ground planes on the different layers. This also improves EMI shielding and reduces ground impedance. This is an industry precaution and it is best practice to include via stitching in projects like this.

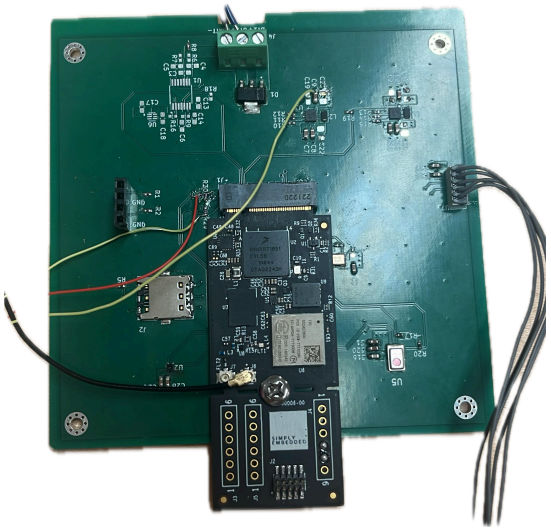

Figure 16: PCB Board

4.2. Software Methodology

The wildfire detection system is built using a modular and scalable architecture that enables real-time monitoring of environmental conditions across various devices deployed in the field. The system is designed to ingest time-series data from sensor-equipped IoT devices and display it on a live, user-friendly dashboard.

At the sensor layer, devices collect environmental data such as temperature, humidity, infrared (IR) levels, and PM2.5 concentrations. These values are encoded in a lightweight custom protocol and transmitted to a centralized cloud service hosted by Simply Embedded, which serves as the intermediary for device communication and routing. The cloud system enables flexible device-to-server communication without requiring direct IP

access to the sensors.

Once received, the data is stored in an InfluxDB time-series database. Each sensor type is recorded as a specific field within a measurement inside the shared "data" bucket. All readings are time stamped and tagged with the corresponding device_id, enabling efficient filtering and querying per device. This structure makes it easy to query recent sensor values and extract data for a specific device, time range, or measurement type.

On the backend, a Django web application serves as the core of the system. Django handles APIs to query sensor values from InfluxDB and serialize the data for frontend use.

The frontend features interactive dashboards powered by Chart.js and Leaflet, enabling real-time visualization of both sensor data and device locations. Users can filter by date, select custom data ranges, and toggle between datasets such as temperature, humidity, IR, and PM2.5. Live sensor readings and system logs are refreshed every 10 seconds through JavaScript-based polling, ensuring that the dashboard stays up to date without requiring a full page reload.

This layered approach offers a balance between flexibility, performance, and maintainability, making it ideal for environmental monitoring systems with live data needs.

I. Django Web Framework

The Django web framework is the backbone of the wildfire detection system's backend. It powers the user interface, manages authentication, defines device views, and acts as the bridge between the time-series database (InfluxDB) and the frontend dashboard. Django’s Model-View-Template (MVT) architecture and admin features made it an ideal choice for rapid development, scalability, and maintainability.

At the core of the application, Django handles user sessions, device management, and permissions. Custom models are defined for each key component in the system, such as Device, Group, FluxRecord, and Organization, allowing the backend to keep track of every sensor node, its status, metadata, and historical data.

```
class Device(models.Model):
    name = models.CharField(max_length=100)
    device_id = models.CharField(max_length=64)
    latitude = models.FloatField(null=True, blank=True)
    status = models.CharField(...)
    …
```

The view_device view is a core part of the application. It accepts dynamic parameters such as device ID and date range through GET requests, queries time-filtered sensor records from InfluxDB, and serializes the results into JSON. This JSON is then embedded directly into the Django template using a <script type="application/json"> tag, making it accessible to the frontend JavaScript without the need for a separate API call. This approach enables fast, dynamic chart rendering and table updates with minimal backend load.

```
def view_device(request, pk=None):
    start_dt = datetime.fromisoformat(start_str)
    ensure_device_flux_data(device, request, start=start_dt,
    end=end_dt)
    records = FluxRecord.objects.filter(
        device_id=device.device_id,
        time__gte=start_dt,
        time__lte=end_dt
    ).order_by("time")
    sensor_data = [{"time": r.time.isoformat(), "value_0":
    r.value_0, ...} for r in records]
```

Django templates are responsible for rendering the core interactive components of the user interface, including sensor charts, data tables, and device overview cards. Each Django view is registered in urls.py and linked to a corresponding HTML template. These templates are built using Bootstrap for layout styling and Chart.js for dynamic chart rendering.

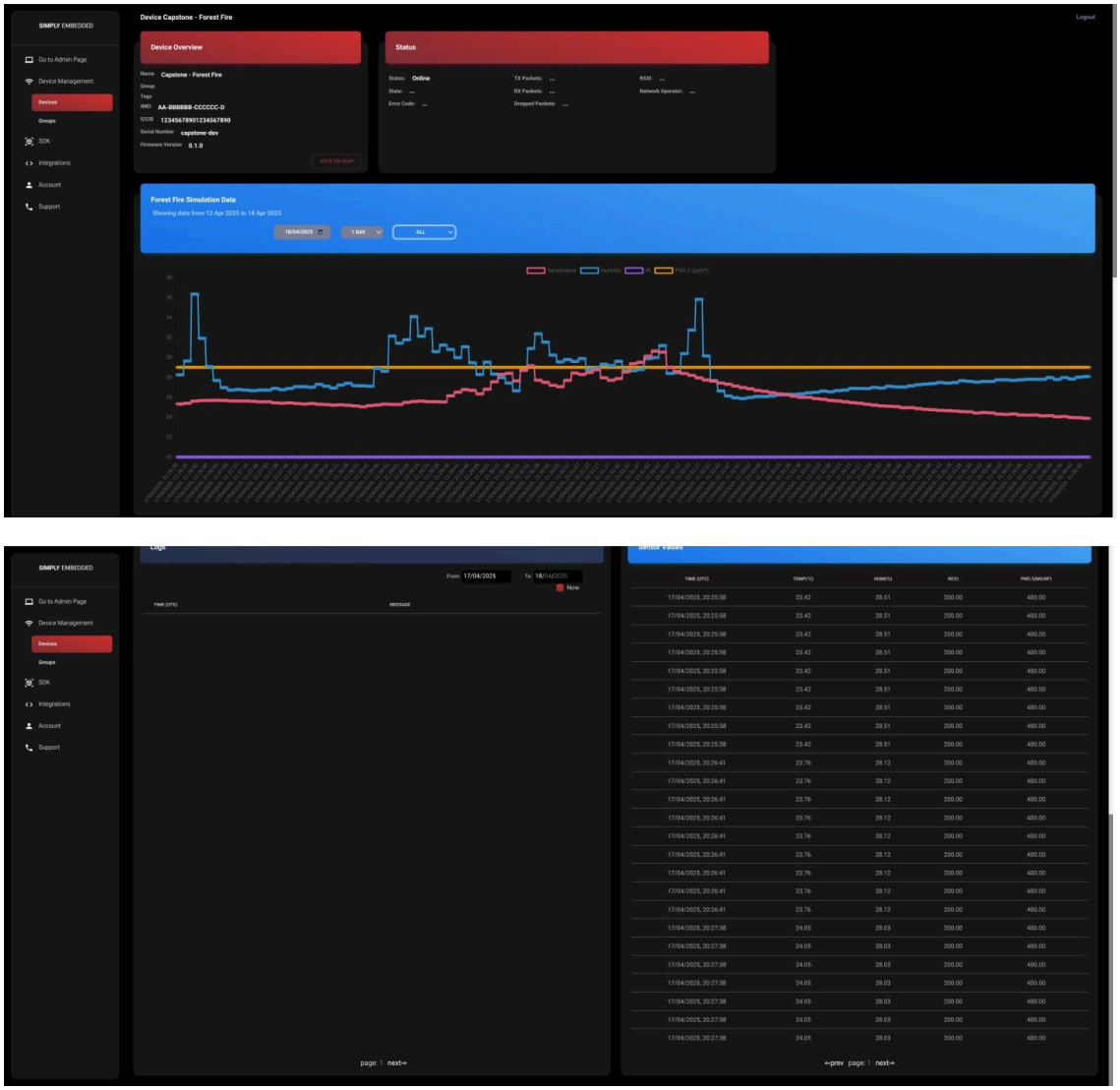

Figure 17: Dashboard

This full-stack integration between Django (backend and templating), InfluxDB (time-series storage), and JavaScript (interactive UI) enables real-time, responsive interaction with sensor data through a secure, user-friendly web dashboard.

Leaflet Map Integration

The project uses Leaflet, an open-source JavaScript library, for rendering device locations on an interactive map. This allows users to monitor device status and latest sensor readings in a geographical context.

Each device is plotted on the map using its latitude and longitude coordinates. Clicking a device marker reveals a tooltip with:

- Device name
- Last update timestamp
- Sensor values (Temperature, Humidity, IR, PM2.5)
- Calculated wildfire risk level

Wildfire Risk Calculation Algorithm

The wildfire detection system includes a custom-built algorithm to estimate the risk level of wildfire activity based on real-time sensor readings from each deployed device. This algorithm is implemented on the backend and executed during map data rendering and device status updates.

Input Sensors

The risk scoring system uses four key environmental parameters collected from each IoT device:

- Temperature (°C) – measured via environmental sensors.
- Humidity (%) – a critical factor affecting fire spread.
- Infrared (IR) Intensity (V) – high IR radiation can suggest combustion or abnormal thermal behavior.
- PM2.5 (μg/m³) – elevated particulate levels may indicate smoke or pollution.

Risk Scoring Logic

Each parameter is individually scored using threshold-based logic, and the total risk score is computed as the sum of these individual scores.

```
def calculate_risk(temp, humidity, ir=None, pm25=None):
    risk_score = 0

    # Temperature
    if temp >= 38:
```

```
    risk_score += 3
elif temp >= 35:
    risk_score += 2
elif temp >= 30:
    risk_score += 1

# Humidity
if humidity <= 15:
    risk_score += 3
elif humidity <= 25:
    risk_score += 2
elif humidity <= 35:
    risk_score += 1

# IR
if ir is not None:
    if ir >= 2.0:
        risk_score += 3
    elif ir >= 1.5:
        risk_score += 2
    elif ir >= 1.2:
        risk_score += 1

# PM2.5
if pm25 is not None:
    if pm25 >= 150:
        risk_score += 3
    elif pm25 >= 100:
        risk_score += 2
    elif pm25 >= 75:
        risk_score += 1

# Total score
if risk_score >= 8:
    return "fire"
elif risk_score >= 5:
    return "fire_possible"
elif risk_score >= 3:
    return "suspicious"
else:
    return "safe"
```

The map view uses real-time updates and colour-coded markers to indicate current risk severity:

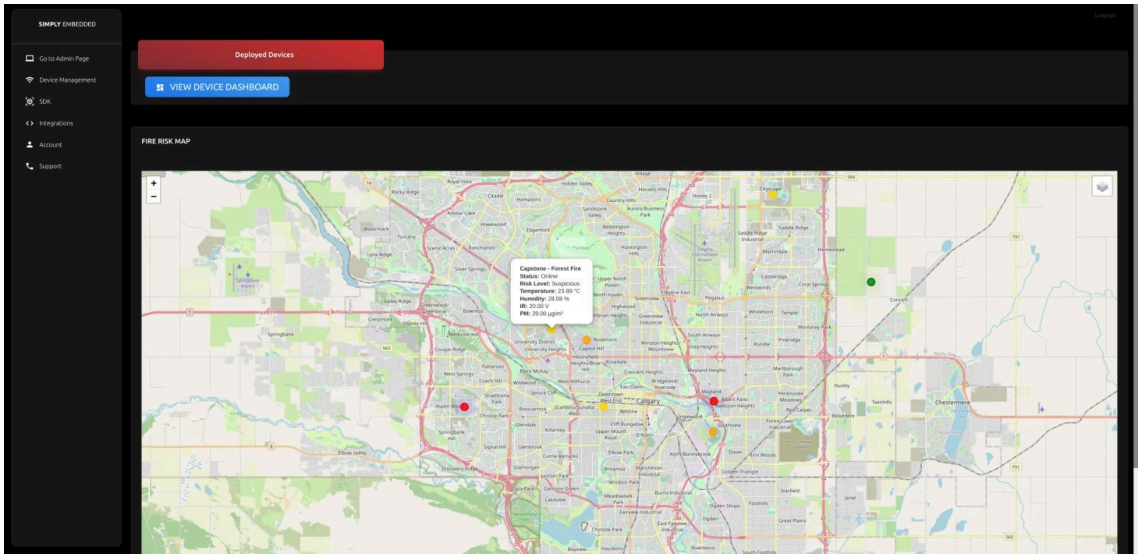

Figure 18: Dashboard Map Feature

I. InfluxDB Time-Series Database

InfluxDB is the main storage engine for the wildfire detection system. As a time-series database, it is optimized for handling high-frequency, timestamped data, making it ideal for storing environmental sensor readings such as temperature, humidity, particulate levels (PM2.5), gas concentrations, and infrared radiation.

The system uses a single bucket named data. Within this bucket, each sensor type is stored under a measurement, most commonly, env, which stores environmental sensor data. This measurement contains multiple fields like value_0, value_1, etc., representing different sensor readings from the device.

All sensor records are tagged with a device_id, allowing for device-specific queries. Conceptually, each measurement in InfluxDB is like a table in SQL, with fields acting as columns and each record carrying a precise timestamp.

```
env:
  values:
    value_0: Gas
    value_1: Humidity (%)
    value_2: Pressure (kPa)
```

value_3: Temperature (°C)

```
env | 2025-04-17 21:14:32 MDT | value_1 | 27.932249069213867
env | 2025-04-17 21:14:37 MDT | value_1 | 28.048599243164062
env | 2025-04-17 21:14:42 MDT | value_1 | 27.636606216430664
env | 2025-04-17 21:14:47 MDT | value_1 | 27.75295639038086
```

The system uses the Python influxdb-client library to interact with the database. Django views (such as view_device) construct Flux queries dynamically based on device and date ranges.

```
query = f'''
from(bucket:"data")
  |> range(start: -2h)
  |> filter(fn:(r) => r.device_id == "{device_id}")
  |> filter(fn:(r) => r._measurement == "env")
'''
query_result = query_api.query(org=org, query=query)
```

Each record is parsed and serialized into JSON for use on the frontend. InfluxDB is not directly mapped as a Django model. Instead, a helper function (ensure_device_flux_data) ensures recent data is available by triggering ingestion before querying. The resulting values are returned as a list of FluxRecord objects and passed to the template.

II. Simply Embedded Cloud System & Data Transfer Protocol

The wildfire detection system uses the Simply Embedded cloud platform as the backbone for managing device connectivity, secure data routing, and integration with the backend. This cloud service acts as the intermediary between field-deployed IoT sensors and the Django-based monitoring dashboard.

Each device sends environmental data (e.g., temperature, humidity, IR, PM2.5) to a central MQTT broker hosted on the Simply Embedded platform. The data is encoded using CBOR (Concise Binary Object Representation), a compact and efficient binary format ideal for embedded systems. This combination (MQTT + CBOR) ensures low-bandwidth communication with minimal overhead, making it well-suited for remote wildfire monitoring.

Data Flow Overview:

1. IoT devices collect environmental readings.
2. Data is encoded via CBOR and published to an MQTT topic.
3. Simply Embedded cloud receives the message, validates it, and routes it to the backend.
4. The Django backend parses and inserts the decoded values into InfluxDB for storage and analysis.

This managed infrastructure handles:

- Device provisioning and authentication
- Health monitoring
- Reliable data delivery
- Real-time communication at scale

It simplifies backend development and allows the team to focus on frontend visualization, alert systems, and wildfire detection logic, while ensuring robustness in distributed deployments.

4.3. Firmware Methodology

Details of the principles, layers, and toolchain that guided the development of the firmware running on the i.MX RT1051 microcontroller (Cortex-M7, 528MHz) are detailed here. The ultimate goal was a low-power, field-upgradeable codebase that performed deterministic sensor polling, first-stage processing, and LTE with MQTT telemetry while remaining portable to future iterations of the MCU. Due to the proprietary nature of the SDK flashed onto the MCU, this section may be lacking in technical details, but will provide as much information as possible.

The firmware is built from three main layers: Hardware Abstraction, Device Drivers, and Application.

- The Hardware Abstraction layer is where our sponsor's SDK HAL calls are performed to expose I2C primitives. Key components include common headers such as hal_i2c.h, though the actual code is not available for outside use.

- The Device Drivers layer contains the driver for each sensor and the bare-metal system calls are used to perform read/write operations. Raw bytes sent from the MCU to sensors include that sensor's I2C address and sensor command operations. For the SHT45 this would be a one byte command (i.e., 0x44 is the I2C address and 0x94 is a soft reset command). When the MCU sends a command where it expects to read raw bytes from the the sensor, it expects a buffer packet returned with the raw bytes (i.e., MCU sends a command to the SHT45 to measure temperature and humidity, a buffer of size six bytes is returned). The raw bytes are converted into useful information within each sensor's driver before being sent to the database. Key components include compiled C files such as sht45driver.c along with SDK headers and compiled code for bare-metal system calls.
- The Application layer is the layer where the main code is run. This code initializes the sensor's by calling their drivers, initializes the network the data is transferred by, and publishes telemetry via that network to the database. Key components include the app_main.c and configuration files from the SDK.

I. Toolchain & Build System

The firmware is built utilizing the Arm GNU Toolchain 12.2 (arm-none-eabi-gcc). The SDK's CMake wrapper injects standard Cortex-M7 flags such as:

- –mcpu=cortex-m7
- -mthumb
- -mfpu=fpv5-d16
- -mfloat-abi=hard

These options allow the linker to strip unused code and keep the final image small. The link stage itself is driven by the sponsor's proprietary .ld script that is bundled with the sponsor's SDK. Due to the nature of it being closed source, the exact memory-map details are outside the scope of the report.

For debugging we rely on a SEGGER J-Link probe connected over Serial Wire Debug (SWD) at 500 kHz. After flashing the device, a probe is launched with a GDB server for run-control, break-points, and single-stepping. The stepping includes not only code stepping but assembly instruction single stepping also, which was important during the debugging process of the closed-source SDK as the actual code implementation for some functions were not provided.

II. Sensor Driver Integration

The firmware treats each device as a C module with a similar public API:

```
int <sensor>_init(void);
void <sensor>_sample(<sensor>_t*);
```

Under the hood, each driver follows the sequence recommended in the manufacturer's datasheet but hides all I2C framing with the Hardware Abstraction Layer.

A. SHT45 (Temperature & Humidity)

i. `init()` issues a soft-reset (0x94), waits 1 ms, then stores calibration constants

ii. `sample()` sends the single-shot high-repeatability command (`0xFD`), has a small delay, and reads 6 bytes. The driver verifies the CRC for both words, converts raw bytes to °C/ %RH with the formula in the data-sheet, and fills the `<sensor>_t structure`.

B. SPS30 (Particulate Matter)

i. `init()` toggles the 5 V regulator, has a delay of about 500 ms for the fan, then transmits the Sensirion High-Level Data Link Control (SHDLC) "Start Measurement, float format" frame (`0x00 0x00 0x02 0x01 0x03 CRC`)

ii. `sample()` polls "Read Measured Values" (`0x03`) once per second. The reply is 60 bytes and contains nine 32-bit floats. The driver then lifts PM2.5 and packs them into a payload as fixed-point integers.

iii. SPS30 has a `sleep()` which sends a "Stop Measurement" (`0x01`) command that turns off the 5 V rail, and the fan slows to a stop, this drastically drops the current draw from approximately 55 mA to almost 50 μA.

5. Product Scope

The finalized product was developed within a refined scope, focusing on delivering a cost-effective, and deployable wildfire monitoring node. The primary goals were shaped by sponsor priorities, emphasizing simplicity, modularity, and hardware verification within constrained budget and timeframes.

Key Scope Elements Achieved:

- **Environmental Compatibility:**
    a. Designed to operate within -10°C to 85°C and 0–95% RH, matching expected wildfire-prone environmental conditions.
    b. Testing of environmental conditions was not performed since all selected components were rated to operate within the required range.
- **Optimized Electrical Design:**
    a. System powered by a 3.3V, 5V, and 1.8V supply, narrowed down from initial options to better align with sensor requirements.
    b. Maintained a total power consumption under 5W, including transmission loss.
- **Compact Power System:**
    a. Integrated battery with ≥15,000 mAh capacity, weighing ≤1 kg, and designed to operate in passive mode for 30+ days.
    b. Battery dimensions are constrained to fit within the compact enclosure alongside all components.
- **Scope Refinement:**
    a. Flame detection and PM sensors, along with power-saving firmware development, were excluded from this phase at the sponsor's discretion.
    b. Focus was placed on delivering a minimal but functional prototype using the SHT45 sensor.

6. Final Product Functionality

The product successfully demonstrated core system functionalities required for remote wildfire risk monitoring. These include environmental sensing, wireless data transmission, and battery-powered operation in a lightweight enclosure.

<u>Key Functional Achievements:</u>

1. Real-Time Monitoring with SHT45 Sensor:

1.1. Successfully interfaced the SHT45 sensor with the MCU to measure temperature and humidity in real time.

2. **Long-Term Power Supply**:

    2.1. The system runs passively for over a 30-day period, meeting fire season duration targets under low-power conditions.

3. **Data Transmission:**

    3.1. Designed for long-range communication protocols (LTE-M) with a target transmission range of ≥1 km (validation pending).

    3.2. System supports data streaming for high-risk scenarios and batching for low-risk events, optimizing communication and energy use.

4. **Web-Based GUI (Future Integration):**

    4.1. The system design supports integration with a web browser GUI for data visualization and potential alert notifications.

5. **Compact and Deployable Hardware:**

    5.1. All components were fit within a small enclosure making the node suitable for field deployment.

7. Technical Specifications of the Final Product

This section provides details of the technical specifications that were met in the project. The details of how these specifications were measured can be found in section 8- Measuring Success and Validating Test Results. This section will also discuss the changes made to the technical specifications between our final product and the specifications made in the fall term Design Roadmap, as well as the justification behind these changes.

Test	Technical Goal	Implementation Notes
Environmental Conditions	**Operating Temp:** -10°C to 85°C **Humidity Range:** 0–95% RH	Ensures operation in extreme weather conditions for year round deployment in remote outdoor environments.
Electrical	**Voltage Supply:** 3.3V, 5V, 1.8V **Power Consumption:** ≤ 5W (incl. transmission loss)	Matches voltage requirements of selected microcontroller, sensors, and modem while minimizing power usage

Power	**Battery Capacity:** ≥ 15,000 mAh **Battery Dimensions:** Fit in enclosure **Battery Weight:** ≤ 1kg **Battery Life:** ≥ 30 days (mostly passive)	Enables long-term, unattended operation in remote locations. Battery size and weight optimized for field deployment.
Communication	**Protocols:** LTE, MQTT **Transmission Range:** ≥ 1 km	Supports reliable long-range, real-time data transmission even in areas without Wi-Fi or close infrastructure.
Software & Firmware	**GUI**: Web browser–based interface **SDK:** Integrated closed-source SDK for read/write **Data Capture:** I2C	Chosen for ease of use, quick development, and compatibility with sponsor hardware. I2C used for low-pin-count sensor communication.
Budget	**Cost Limit:** $2000 Sensor Node Development **Cost:** $400	Ensures affordability and scalability of the solution while balancing performance and reliability.

The changes made to the technical specifications had to be changed from the fall-term 'Design Roadmap' document because as we researched and worked on our project, our technical specifications became more relevant. We got rid of some specifications such as enclosure requirements and design since that was out of the scope of our project. All of the technical specifications above are the most relevant and significant to our final product.

8. Measuring Success and Validating Test Results

This section documents the quantitative and qualitative measurements initially described to judge project success, the validation tests executed on the prototype, and the extent to which each predefined criterion from the 'Technical Specifications of the Final Product' was met.

Test	Technical Goal	Validation
Environmental Conditions	**Operating Temp:** -10°C to	**Sensors selected to meet**

	85°C **Humidity Range:** 0–95% RH	**these ranges.** Testing not needed as components fell within.
Electrical	**Voltage Supply:** 3.3V, 5V, 1.8V **Power Consumption:** ≤ 5W (incl. transmission loss)	Voltages specified for project use. Power measured with a multimeter. Worst case ~**1W**
Power	**Battery Capacity:** ≥ 15,000 mAh **Battery Dimensions:** Fit in enclosure **Battery Weight:** ≤ 1kg **Battery Life:** ≥ 30 days (mostly passive)	Two **D-cell lithium thionyl chloride batteries** used, can be paralleled (**max ~30 Ah**). Fit enclosure, weight well under limit. Battery life not field-tested.
Communication	**Protocols:** LTE, MQTT **Transmission Range:** ≥ 1 km	**LTE** chosen for better compatibility and long-distance performance. **Tested range >1 km**, real-time updates confirmed
Software & Firmware	**GUI**: Web browser–based interface **SDK:** Integrated closed-source SDK for read/write **Data Capture:** I2C	Data visualized with **<1s latency.** Closed SDK used to interact with sensors via I2C.
Budget	**Cost Limit:** $2000 Sensor Node Development **Cost:** $400	Actual cost per unit ~**$450** (still under budget). Higher cost due to sensor quality assurance.

A. Hardware Functionality Validation

Test	Purpose	Result
PCB Probing	Check for shorts and ensure continuity in signal traces using a multimeter	Passed: No shorts, good continuity
I2C Probing	Verify I2C signals on	Passed: I2C CLK and data

	oscilloscope to ensure they are active	observable when toggled
Address Scanning	Ensure all sensors are addressable for programming	Passed: All sensor addresses visible during scan
Testing Sensors on Different Master	Confirm sensor functionality when controlled by an alternate master device	Passed: Sensors worked correctly with a different master

B. Software Functionality Validation

Test	Purpose	Result
Backend Validation	**Dynamic Data Retrieval:** Use Django views to fetch sensor data by device ID and date range	Passed: Data retrieved correctly
	InfluxDB Integration: Use Django views to fetch sensor data by device ID and date range	Passed: Validated with independent queries
Frontend Validation	**Live Chart/Table Updates:** Ensure sensor data updates correctly in charts/tables based on filters	Passed: Data reflected user-selected filters
	Interactive Filters: Allow users to choose date ranges and datasets (e.g., temperature, humidity)	Passed: View updated accordingly
	Paging: Display sensor logs in paginated format, synced with live updates	Passed: Logs displayed and updated as expected
Map Functionality (Leaflet)	**Device Markers on Map:** Show device locations accurately on map	Passed: All devices displayed
	Real-Time Data Popups: Display latest sensor readings and risk level in map popups	Passed: Popups updated in real time

	Color-Coded Risk Display: Visually indicate fire risk (Safe → Fire) with intuitive color scheme	Passed: Risk clearly visualized

C. Firmware & Sensor Functionality

Test	Purpose	Result
MCU Validation	Ensure firmware flashes and MCU boots reliably	Passed: Firmware flashed via J-Link; device boots to main loop every time
SDK Validation	Verify SDK builds and operates correctly with MCU HAL via I2C	Passed: Project built with CMake + Makefile; custom syscalls worked without NACKs
Sensor Validation	Validate communication between sensors and MCU using I2C and confirm correct sensor data	Passed: I2C worked for data capture; SHT45 gave accurate temp/humidity readings; single/multi-byte reads confirmed

Validation Tests

Power Consumption Testing

- Purpose:
    - Confirm individual sensor power < 5 W
- Methodology:
    - Bench supply set 3.6 V @ 0.3 A
    - Multimeter on Vcc lines

Data Capture Integrity (Functional)

- Purpose:
    - Verify accurate data capture
- Methodology:

    - ESP32 EDK polled SHT45
- Result:
    - Ambient: 24.9 ± 0.5 ºC, 27.6 ± 1 %RH
    - Lighter Flame: 31.4 to 34.1 ºC, 29.0 ± 1 %RH
- Status:
    - SHT45: **PASS**
    - SPS30: **NOT PERFORMED**
    - Flame Detection IR: **NOT PERFORMED**

Hardware / Firmware Integration (Functional)

- Purpose:
    - Confirm MCU firmware controls sensors within power limits
- Methodology:
    - i.MX RT1051 (528 MHz) running sponsor closed-source SDK
    - Oscilloscope on I2C lines
- Result:
    - Firmware executes without over-current events
- Status:
    - **PASS**

Cloud Communication (Performance)

- Purpose:
    - Validate MQTT → InfluxDB pipeline
- Methodology:
    - Publish raw data
    - Timer measured sample → graph delay
- Result:
    - Latency is approximately 1 s
- Status:
    - **PASS**

Dashboard Availability (User Interface)

- Purpose:
    - Ensure UI reflects live device state
    - Ensure UI receives live device updates
- Methodology:
    - Manual refresh
    - Automatic update
    - Offline simulation
- Result:
    - Graph renders with data
    - Graph updates with data
    - No heartbeat from device → cannot detect device offline
- Status:
    - Refresh: **PASS**
    - Update: **PASS**
    - Offline: **FAIL**

Power-Consumption Profile (Performance)

- Purpose:
    - Measure idle current
    - Estimate battery life
- Methodology:
    - *Not executed* - battery pack never integrate
    - Leave device on while measuring current drawn
    - Check device often until no more current being drawn
- Result:
    - N/A
- Status:
    - **NOT PERFORMED**

Environmental Stress (Performance)

- Purpose:

    - Test the device in various weather temperatures from -10 ºC to 80 ºC
- Methodology:
    - *Not executed* - no enclosure
    - Test in cold temperatures or in a chamber to reach lower temperatures
    - Use oven for higher temperatures
- Result:
    - N/A
- Status:
    - **NOT PERFORMED**

Enclosure (Compliance)

- Purpose:
    - IP-rating water ingress
    - Animal intrusion check
- Methodology:
    - *Not complete* - enclosure not complete
    - Spray with water to ensure none gets in
    - Leave outside for 24 hours to make sure animals could not get in
- Result:
    - N/A
- Status:
    - **NOT PERFORMED**

Results vs. Criteria

Functional goals were met for temperature and humidity data acquisition, SHT45 communication on our designed PCB via I2C, reliable MQTT transfer via Rogers LTE network, and MCU-sensor integration on our designed PCB ensuring proper distribution of power to sensors.

Performance goals which we deemed partially met was the latency of the data transfering to the dashboard. We would have liked to see more immediate feedback from the data collected from the sensor and displayed on the dashboard. We also would like to see the data in tabular format alongside the graph, to better read the results.

Two performance test goals (power consumption profile and environmental stress) were unmet. Since we did not have the battery connected to our designed PCB, we were unable to fully test the battery capacity or the stress that harsh temperature would have on the device. The device also did not send a heartbeat to the dashboard, so if the device somehow was offline, the dashboard would have no way of knowing and would continually output data with NULL values.

Since the enclosure was not completed, it could not undergo compliance testing to make sure that it could withstand harsh weather conditions.

Lessons Learned

The validation phase revealed critical insight that will help inform the next hardware and software revision and future IoT field projects.

Firstly, performing incremental bring-up on an inexpensive test harness pays for itself. When the i.MX RT1051 began issuing I2C NAK's late in the schedule, using an ESP32 microcontroller let us decouple firmware from hardware and confirm that the SHT45 and the wiring on the PCB were not at fault. Had we followed an approach that tested the sensor on breadboard first with MCU integration second, we would have surfaced this issue earlier and avoided a tedious debugging exercise to determine why we were receiving NAKs.

Secondly, the energy budget should not be an afterthought. Our power consumption targets were dependent on empirical battery-drain data, though the battery pack was never implemented on to the PCB due to time constraints. Without real discharge curves we could neither tune duty-cycles nor verify that our device would last a four-month field test as we hoped for. Future projects will have battery-profiling sprints immediately and capacity data can be determined by running the cells on benchtop if need be.

Thirdly, live telemetry is much more valuable than rich telemetry. At the end of the day, the end users for the product care more about whether a remote node is still breathing compared to visually appealing rendered graphs. Since the dashboard and the device lacked a heartbeat between each other, a severed LTE link would have looked like a flat-line of data containing 0 or NULL for all values, masking the most critical failure mode. Going forward, a heartbeat will be added on every transaction in idle mode and the dashboard will colour-code stale devices.

Fourthly, supply-chain buffers must be explicitly accounted for. Shipping delays of critical parts and the printing of our PCB caused almost three weeks of delays, severely compressing our schedule, and forcing numerous “not performed” verdicts during the validation phase. Allocating a proper buffer for shipping delays and defining a go/no-go contingency such as keeping a breadboard sensor kit set up will reduce the risk of this type of delay.

Lastly, the project highlighted the importance of cross-disciplinary documentation. Hardware, firmware, and software teams kept separate logs; a discovery that led to much confusion when trying to communicate with each other about certain aspects of the project. Since the teams worked largely in isolation from each other, there was little cross discipline ownership, and it could be difficult to convey important messages about why an error in the project was happening.

Taken together, these lessons enforce that early modular testing, disciplined power analysis, resilient UI design, proactive supply-chain planning, and ownership of all parts of the project are important for the success of a production-grade wildfire-detection platform.

9. Materials and Cost

This section provides a detailed economic breakdown of the components procured for the Early Wildfire Detection System, including the manufacturer product number, Digikey part number, category, unit cost, and total cost in CAD. The selected components were chosen based on performance, compatibility, and cost-effectiveness to ensure the system remains financially viable while meeting the project's technical requirements.

Component Procurement and Cost Breakdown

Component	Manufacturer Product Number	Digikey Part Number	Category	Quantity Ordered	Total Cost (CAD)
Temperature/ Humidity Sensor	SHT45-AD1B-R3	1649-SHT45-AD1B-R3CT-ND	Environmental Sensor	5	$36.29

Particulate Matter Sensor	SEK-SPS30	1649-1107-ND	Air Quality Sensor	1	$155.31
Flame Sensor	USEQFSEA464180	399-USEQFSEA464180CT-ND	Fire Detection	5	$140.18
Regulator	TPS63900DSKR	296-TPS63900DSKRCT-ND	Power Management	5	$17.45
Fuel Gauge	BQ35100PWR	296-47251-1-ND	Power Monitoring	5	$29.95
Oscillator	ECS-327MVATX-3-CN-TR	ECS-327MVATX-3-CN-TR	XC3155CT-ND	5	$8.4
SIM Module	SIM8055-6-1-14-00-A	SIM8055-6-1-14-00-A	2073-SIM8055-6-1-14-00-ACT-ND	5	$9.85
LDO Regulator	TLV7111812DSER	TLV7111812DSER	296-30550-1-ND	5	$4.45
Diode	BAT120C,115	1727-5450-1-ND	Circuit Protection	5	$5.9
MOSFET	DMN21D2UFB-7B	DMN21D2UFB-7BDICT-ND	Switching	5	$1.7
Panel for Enclosure	1554QPL	164-1554QPL-ND	Enclosure	1	$14.58
Enclosure	1554Q2GY	164-1554Q2GY-ND	Enclosure	2	$38.07
Battery Holder	176	36-176-ND	Battery	1	$14.42

Total Cost Summary:

- Total expenditure on all components: 476.55 $CAD
- All components were sourced through Digikey, ensuring high quality, reliability, and compatibility with the system's architecture.
- All other essential components, including circuit elements such as resistors, capacitors, connectors, wiring, and PCB materials, were provided by sponsors, ensuring cost efficiency while maintaining system integrity.

Economic Considerations and Design Decisions

Economic constraints played a significant role in component selection and procurement. Several cost-benefit trade-offs were made to balance performance and affordability:

1. Sensor Selection and Optimization:
    1.1. Instead of using high-end industrial sensors, commercial-grade sensors were selected to reduce costs while maintaining accuracy.
    1.2. Multiple sensors were integrated to compensate for potential limitations in individual sensor accuracy while remaining within budget.
2. Power Supply Cost Efficiency:
    2.1. The system was designed to run on a low-power consumption model, reducing battery and solar panel requirements.
    2.2. The selected LDO regulator and power monitoring components ensure efficient energy usage, minimizing long-term maintenance costs.
3. Communication Cost Management:
    3.1. The SIM module was chosen for cost-effective data transmission, avoiding expensive satellite communication unless absolutely necessary.
    3.2. Edge processing techniques will be implemented to send only critical data, reducing bandwidth costs.
4. Scalability and Future Cost Reductions:
    4.1. The initial prototype was designed for a small-scale deployment, minimizing upfront costs.
    4.2. Future iterations may use alternative components or bulk purchasing strategies to

drive down per-unit costs.

10. Ethics and Equity

During the course of our capstone work, our team encountered some ethical dilemmas that we had to navigate as a group. Throughout our collaboration, we had to make crucial decisions in order to meet professional codes of conducts and industry standards. As a group we tried to maintain equality and equity with regards group work and decision making. First, we will begin by describing engineering ethics and what ethics means to us as engineers. We will then cover some of the ethical dilemmas and equity issues we encountered during our capstone journey. Lastly, we will end this reflection and discuss how equity considerations influenced our design and team interactions.

Engineering ethics is fundamental in the process of becoming a professional engineer. As engineering students we have had to learn about engineering ethics in our learning journey so far. Capstone has allowed us to apply much of our knowledge about engineering ethics into our project dynamics. Many of our group mates have participated in internships and developed a deep sense of what ethics could mean in the workplace. APEGA, which regulates engineering practices in Alberta, has five rules of conduct that we maintained in our group [1]. When considering ethics in our project, we had to think about safety, competency, integrity, honesty and fairness. This includes building a conducting our group in a safe manner and building a product that is safe to deploy in public. We had to ensure that our group members were confident in their own skills and competency by being honest and bridging gaps in our knowledge when we don't understand something. This was done mostly by seeking advice from our sponsors on industry practices. We aimed to maintain objectivity and unbiased opinions in our project. Lastly, our group valued equity, inclusivity and respect in our group.

Our group had to identify and address a few ethical dilemmas while working on our project. When we first considered our project, a wildfire monitoring system, we first had to consider the boundaries of our project in order to ensure public safety and the integrity of our product. We had considered the distance one sensor could cover, where to deploy our product and how we could alert someone of risks. Since our project is a prototype, we thought it would be best to approach this project in a more experimental light as much time would be required to test our product which is beyond the scope of this course. We had to limit the scope of the project

to our competency level as this project can get complicated fast. We decided to deploy this project close to city limits in order to have better testability of our project. These limitations were necessary when addressing ethical concerns such as safety. We also had to maintain safety within our group during the project by adhering to safety protocols during dangerous tasks like soldering. We made sure we got the proper training and used the correct tools in order to uphold safety and health. In order to maintain a healthy group dynamic, we had to maintain integrity and honesty. For example, our firmware development team fell behind due to miscommunication and had to be upfront with our other group members in order to manage risks. Our group members communicated this delay and we were able to prioritise our software development instead of waiting around and further delaying our project. Delays in our project would have severely delayed our testing time which would add more risk to our product.

Lastly, we will address how we considered equity in our design and group dynamic. Our group is quite diverse with members with different backgrounds and experiences. We implemented inclusive decision making which encouraged all of our group members to share their thoughts. We wanted to ensure that everyone, regardless of their major or background, shared their thoughts even if they didn't directly contribute to that part of the project. We maintained fairness by distributing tasks evenly to avoid burdening one team member. If any team member had any concerns about their workload then our project manager would take that into consideration and consult other team members in order to maintain fair workloads. In order to maintain unbiased and fair team dynamics, we adhered to our team contract which we set up at the beginning of our capstone journey. In our project, we maintained a fair design in our software by having a user friendly dashboard that is easy to navigate and read. Our dashboard includes multiple graphs and a map and we wanted to ensure that they are intuitive for our end user. Our dashboard was our main concern with addressing equity in our design since that is what the user will be interacting with rather than our hardware.

In conclusion, throughout our capstone journey, we had to address ethical dilemmas in our design and teamwork. As a team, we valued safety, competency, integrity, honesty and fairness. We implemented the ethical philosophies outlined by the APEGA rules of conduct. Our team was committed to fairness and inclusivity in our decision making and collaboration. Capstone has reinforced the responsibilities, roles and ethics of engineers.

11. Impact of Engineering on Society/Environment

Wildfire seasons are growing longer and more destructive in provinces such as Alberta and British Columbia, and time-to-detection remains a decisive variable: an ignition of fire that is located within the first five minutes is statistically far easier (and cheaper) to suppress than one confirmed hours later from a remotely manned-watchtower. Recent field trials of low-cost IoT fire-detection nodes demonstrate that sensor arrays similar to the one we designed can recognise a fire-signature within one to five minutes of its ignition, a performance window that conventional manned watch towers rarely achieve [2].

The public-health upside of earlier alerts is just as pronounced. Health Canada reports that short-term exposure to fine particulate matter from wildfire smoke aggravates conditions such as asthma, pulmonary disease, and cardiovascular conditions, with children, seniors, and immunocompromised individuals at the highest risk [3]. By shortening the period during which a fire can spread and spew particulate matter, our system reduces the likelihood of prolonged smoke events like those that blanketed the City of Calgary in 2023.

Reduced fire size also translates into tangible economic gains. Federal data shows wildfire-protection costs in Canada have risen by approximately $150 million per decade since the 1970's, topping $1 billion in six of the last ten years [4]. Limiting the spread of wildfires through early detection therefore cushions provincial budgets and shields tourist hubs located in heavily boreal hubs (Canmore, Banff, the Kootenays) from devastating closures that affect revenue, such as the situation in Jasper in 2024.

Environmentally, the technology protects both wildlife in the region and the climate. Boreal forests are the world's largest terrestrial carbon sink; modelling work indicates that stand-replacing fires immediately flip these ecosystems from a net sink to a substantial source of carbon in the atmosphere for decades [5][6]. Early knock-down preserves mature canopies, maintains habitat survivability, and prevents the surge of CO2 that follows large-area combustion caused by wildfires.

Our prototype does, however, carry environmental liability. Lithium primary cells and electronic assemblies contribute to the global electronic waste stream, where studies show that 98% of Li-ion batteries still end up in landfills. These batteries can leach heavy metals and contain the

possibility of igniting secondary fires [7]. Physical deployment of the prototype, which is about the size of a shoebox and strapped to a tree, may disturb nesting birds or small mammals if placed carelessly.

To mitigate such downsides on the environment, our team is dedicated to ensuring that when a node is placed it is enrolled in a provincial battery-take-back program, publish siting guidelines that prioritize placing the nodes on existing lookout poles or dead-standing trees away from active nests, and releasing anonymous sensor feeds via an open API so that the municipalities and First-Nations governments utilizing the product can integrate the data without incurring licensing fees. These measures, coupled with future hardware revisions that target year long battery life and modular sensor replacements, strengthen the project's net-positive balance of social benefit versus environmental cost.

12. Project Management Reflection

Our team's approach to project management evolved significantly over the course of this project. At the beginning, progress was slower as we were all working on a project of this scale for the first time. The steep learning curve, especially in developing firmware and working with complex hardware, meant that much of our first semester was spent understanding the technical aspects rather than making significant progress on implementation. However, our sponsors were supportive throughout the entire process and helped us bridge knowledge gaps and provided guidance.

Looking back, our biggest takeaway is the importance of detailed planning from the start. Having a more structured breakdown of deliverables and a clearer timeline would have helped us manage the workload more effectively. Additionally, asking more questions early on would have provided us with the clarity we needed to move forward with more confidence. Since we underestimated potential setbacks related to debugging hardware-firmware interactions, we didn't initially account for how time-consuming troubleshooting would be. Delays in sourcing certain components also impacted our timeline. To mitigate these risks, we adapted by setting internal deadlines ahead of our actual deliverables, allowing buffer time for debugging and unexpected delays.

Overall, our experience highlighted the importance of proactive planning, clear communication, and structured risk management in successfully executing a complex project. Flexibility and adaptability were key to overcoming challenges, whether in adjusting timelines, reallocating tasks, or troubleshooting technical issues. As we progressed, we learned to distribute tasks more effectively and implement

contingency plans, such as testing sensor modules separately before integrating them into the full system. These adjustments helped us identify and resolve issues faster, ultimately leading to a smoother second half of the project.

13. Economics and Cost Control

Economic constraints played a significant role in component selection and procurement. Several cost-benefit trade-offs were made to balance performance and affordability:

1. Sensor Selection and Optimization:

    - Instead of using high-end industrial sensors, commercial-grade sensors were selected to reduce costs while maintaining accuracy.
    - Multiple sensors were integrated to compensate for potential limitations in individual sensor accuracy while remaining within budget.

2. Power Supply Cost Efficiency:

    - The system was designed to run on a low-power consumption model, reducing battery and solar panel requirements.
    - The selected LDO regulator and power monitoring components ensure efficient energy usage, minimizing long-term maintenance costs.

3. Communication Cost Management:

    - The SIM module was chosen for cost-effective data transmission, avoiding expensive satellite communication unless absolutely necessary.
    - Edge processing techniques will be implemented to send only critical data, reducing bandwidth costs.

4. Scalability and Future Cost Reductions:

- The initial prototype was designed for a small-scale deployment, minimizing upfront costs.
- Future iterations may use alternative components or bulk purchasing strategies to drive down per-unit costs.

14. Approach to Life-long Learning Reflection

Throughout our wildfire detection capstone project, our team consistently engaged in lifelong learning by identifying knowledge gaps and collaboratively finding ways to overcome them. This project exposed us to a broad range of technical areas, from firmware development and sensor integration to cloud communication, database systems, and full-stack web development, many of which were entirely new to us. Navigating these challenges helped us grow as engineers and problem-solvers.

One of the most significant learning curves we faced was with firmware development. None of us had extensive prior experience building production-level firmware, especially for embedded systems with constrained memory, real-time sensor polling, and low-level communication protocols. We had to understand how to collect and encode data from multiple sensors, including environmental, infrared, and particulate matter sensors, and how to package that information reliably for cloud transmission. Early issues with payload inconsistencies and serial communication bugs taught us how easily downstream systems could break from minor mistakes at the device level.

To overcome these challenges, we closely collaborated with our sponsor and dug into existing firmware codebases, technical datasheets, and online documentation. We also held internal walkthroughs to map out how data moved from sensor polling loops to cloud ingestion, helping us build shared understanding across the firmware and backend teams. This part of the project taught us a lot about debugging in low-visibility environments, efficient memory usage, and the importance of reliable communication protocols.

On the software and backend side, we encountered technologies like InfluxDB, Django, and Leaflet, which most of us were unfamiliar with at the start. Working with InfluxDB introduced us to time-series data modelling and Flux queries, while Django's MVT architecture required us to rethink how frontend and backend components interacted. Leaflet enabled us to integrate real-time map views showing device locations and wildfire risk levels, which added a meaningful spatial dimension to our dashboard.

To manage our learning as a team, we adopted several strategies:

- Assigning individuals to research unfamiliar technologies (e.g., CBOR encoding, MQTT) and present findings to the group.
- Hosting whiteboard sessions to diagram system architecture and clarify integration points.
- Maintaining shared documentation for common issues and their solutions (e.g., decoding errors, data format mismatches).
- Pair programming across teams, especially between firmware and backend, to close knowledge gaps and streamline integration.

This project reinforced that lifelong learning is about more than just acquiring new technical skills; it's about embracing uncertainty, working collaboratively, and staying persistent when solutions aren't immediately clear. Firmware development, in particular, tested our patience and taught us the value of attention to detail, but solving those problems made the end result far more rewarding.

By the end of the project, I felt more capable not only in specific technologies but also in navigating complex, interdisciplinary problems. This experience has better prepared me for future roles that involve full-stack development, embedded systems, and collaborative engineering across hardware and software domains.

15. References

[1] "5 rules of Conduct Reference Guide " APEGA, https://www.apega.ca/docs/default-source/pdfs/standards-guidelines/code-of-ethics-5-rules-of-conduct-reference-guide-march-2023.pdf?sfvrsn=ef0d8c00_3 (accessed Feb. 8, 2025).

[2] M. N. A. Ramadan *et al.*, "Towards early forest fire detection and prevention using AI-powered drones and the IoT," *Internet of Things*, vol. 27, pp. 1–22, 2024. [Online]. Available: https://doi.org/10.1016/j.iot.2024.101248. [Accessed: Apr. 18, 2025].

[3] Health Canada, *Human Health Effects of Wildfire Smoke*. Ottawa: Government of Canada, 2024. [Online]. Available: https://www.canada.ca/en/services/health/healthy-living/environment/air-quality/wildfire-smoke/human-health-effects-report-summary.html. [Accessed: Apr. 18, 2025].

[4] Canadian Climate Institute, “FACT SHEET: Climate change and wildfires in Canada,” *Canadian Climate Institute*, Jul. 23, 2024. [Online]. Available: https://climateinstitute.ca/news/fact-sheet-wildfires/. [Accessed: Apr. 18, 2025].

[5] Y. Xu, X. Zhang, M. Chen, and L. Wang, "Impacts of wildfires on boreal forest ecosystem carbon dynamics from 1986 to 2020," *Environmental Research Letters*, vol. 19, no. 6, p. 064023, 2024. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-9326/ad489a**.** [Accessed: Apr. 18, 2025].

[6] S. Doerr, “Global CO2 emissions from forest fires increase by 60% | Newsroom,” *UC Merced News*, Oct. 17, 2024. [Online]. Available: https://news.ucmerced.edu/news/2024/global-co2-emissions-forest-fires-increase-60. [Accessed: Apr. 18, 2025].

[7] Institute for Energy Research, “Environmental Impacts of Lithium-Ion Batteries - IER,” *The Institute for Energy Research*, May 11, 2023. [Online]. Available: https://www.instituteforenergyresearch.org/renewable/environmental-impacts-of-lithium-ion-batteries. [Accessed: Apr. 18, 2025].