Academy of Scientific Research And Technology
أكاديمية البحث العلمي و التكنولوجيا

جامعة بني سويف
كلية علوم الملاحة وتكنولوجيا الفضاء
NSST

EgSA
وكالة الفضاء المصرية
Egyptian Space Agency

Project name

Onboard Image Processing Satellite Using AI For Environmental Sustainability

Presented by

Adel Ibrahim Ibrahim Hashem
Ahmed Abdelradi Ahmed Fouad
Areeg Tarek Mahmoud Abdullah
Hazem Adel Abdelrazek Ahmed
Manar Mohamed Allam Soliman
Menna Mohamed Fathy Ali
Mennatullah Mohammad Mahmoud Ali
Mohammed Abdelfattah Mohammed Ahmed
Mohammed Abdelhakim Mohammed El-Bakry
Nada Hesham Abdelgaber Ali
Radwa Omar Ali Mohammed
Sarah Ayman Ahmed Mohammed

Under the Supervision of

Dr. Mohammed Murad

2024-2025

ACKNOWLEDGEMENT

First, we would like to express our deepest thanks to “**ALLAH**” for continuous giving and helping to finish this work.

We would like to express our deepest gratitude to **Dr. Mohammed Murad ,** for his complete inspiration through endless hours of scientific education supervision, suggestions concerning the problem, help, and warm encouragement.

Additionally, I would like to thank **Eng. Tamer Mohamed Anwar**, Egyptian Space Agency (EgSA), for always selflessly lending his extra time and his kind supervision, great help, and encouragement in this work.

Also, thanks to **Dr. Mohamed Elhady**, Egyptian Space Agency (EgSA), for his kind supervision, continuous encouragement, and valuable assistance.

Abstract

This study presents an innovative approach for predicting fish aggregation zones in the Red Sea by integrating space-based remote sensing with onboard artificial intelligence (AI) analysis. High-resolution satellite imagery is captured and processed in real time using advanced AI models deployed onboard the satellite. These models analyze key oceanographic and environmental parameters—such as Sea Surface Temperature (SST) and Chlorophyll-a concentration—to identify potential fish-rich areas.

The processed data is then transmitted to ground stations, where further AI-driven predictive modeling is applied to refine the identification of optimal fishing zones. This end-to-end methodology enhances the efficiency and sustainability of fisheries by providing accurate, near real-time forecasts, thereby reducing operational costs and minimizing environmental impact.

The proposed system demonstrates the powerful synergy between satellite technology and AI in supporting precision fisheries management, with a particular focus on the Red Sea region.

List of Abbreviations

SST- Sea Surface Temperature

AI- Artificial Intelligence

ML- Machine Learning

Chl-a- Chlorophyll-a

LSTM- Long Short-Term Memory

DO- DISSOLVED OXYGEN

MODIS- Moderate Resolution Imaging Spectroradiometer

NetCDF or .nc - Network Common Data Form

PFZs- Potential Fishing Zones

ARIMA- Auto Regressive Integrated Moving Average

SARIMA- Seasonal ARIMA

RMSE- Root Mean Square Error

MAPE- Mean Absolute Percentage Error

OBC- Onboard Computer

OBDH- Onboard Data Handling

RTOS- Real-Time Operating Systems

SSH -Secure Shell

UART- Universal Asynchronous Receiver/Transmitter

I2C- Inter-Integrated Circuit

CSI- Camera Serial Interface

GPS- Global Positioning System

NDVI- Normalized Difference Vegetation Index

CSV- Comma-Separated Values

MSI- MultiSpectral Instrument

NIR- Near-Infrared

RGB- Red, Green, Blue

TXT- Text File

SLSTR- Sea and Land Surface Temperature Radiometer

DHT- Digital Humidity and Temperature

GNSS- Global Navigation Satellite System

ESA- European Space Agency

STK- Systems Tool Kit

SSO- Sun-Synchronous Orbit

FOM- Figure of Merit

NASA- National Aeronautics and Space Administration

AGI- Analytical Graphics, Inc

FOV- Field of View

LTDN- Local Time of Descending Node

MODIS- Moderate Resolution Imaging Spectroradiometer

RAAN- Right Ascension of the Ascending Node

UTC- Coordinated Universal Time

AER- Azimuth, Elevation, Range

GUI- Graphical User Interface

NMEA- National Marine Electronics Association

GIS- Geographic Information System

ArcGIS- Arc Geographic Information System

NC- Not Connected

BOM- Bill of Materials

DRC- Design Rule Check

PMU- Power Management Unit

BMS- Battery Management System

BeCu- Beryllium Copper

FOS- Factor of Safety

LabVIEW G WebVI- LabVIEW G Web Virtual Instrument

HTML- HyperText Markup Language

IoT- Internet of Things

LabVIEW- Laboratory Virtual Instrument Engineering Workbench

URL- Uniform Resource Locator

OSM- OpenStreetMap

ID- Identifier

Colab- Google Colaboratory

JSON- JavaScript Object Notation

HTTP GET- HyperText Transfer Protocol – GET Request

REST APIs- Representational State Transfer – Application Programming Interfaces

IndexedDB- Indexed Database

PCB- Printed Circuit Board

FR4- Flame Retardant 4

EMC-	Electromagnetic Compatibility
EMI-	Electromagnetic Interference
ARM-	Advanced RISC Machine
USB-	Universal Serial Bus
HDMI-	High-Definition Multimedia Interface
GPIO-	General Purpose Input/Output
MMAL-	Multi-Media Abstraction Layer
V4L2-	Video for Linux 2
OTP-	One-Time Programmable Memory
SBAS-	Satellite-Based Augmentation System
DDC-	Display Data Channel
SPI-	Serial Peripheral Interface
VBCKP-	Backup Voltage Pin
USDA-	United States Department of Agriculture
FAO-	Food and Agriculture Organization

List of Tables

List of Figures

Table of Contents

Chapter (1) : Introduction

1.1 INTRODUCTION

Introduction Egypt's fisheries sector plays a pivotal role in achieving food security, generating employment, and supporting the national economy. In recent years, fish has become one of the most important sources of animal protein for Egyptians, with the government investing heavily in developing aquaculture systems and fish farms. According to the Central Laboratory for Aquaculture Research, aquaculture now contributes approximately 80% of Egypt's total fish production, with 90% of that being tilapia, while only 20% originates from natural resources such as the Red Sea, the Mediterranean, and the Nile River. Despite this progress, the sector faces growing challenges. Egypt produces more than 2 million tons of fish annually, yet rising domestic demand—with per capita consumption reaching 22 kg/year, higher than the global average of 20.5 kg/year—continues to place pressure on available resources. A 2021 economic study (Shaban, 2021) highlighted inefficiencies in fishing effort across Egypt's natural fishery zones, citing a mismatch between effort and yield, leading to economic losses and unsustainable exploitation patterns. This calls for innovative, data-driven approaches to improve the efficiency and sustainability of natural fisheries. The Red Sea, in particular, stands out as one of Egypt's richest marine ecosystems, hosting hundreds of fish species and unique coral environments. However, it remains underutilized, contributing only 13–18% of national fish production. Much of the fishing activity in the Red Sea is still based on traditional methods without scientific monitoring, which can lead to random fishing, increased fuel consumption, and damage to marine biodiversity. Fish distribution and movement are strongly influenced by environmental variables—most notably chlorophyll-a concentration, which indicates phytoplankton abundance, and Sea Surface Temperature (SST), which affects fish migration and habitat preferences. These two parameters are widely recognized as reliable indicators of fish presence, and their real time monitoring can significantly enhance the prediction of fish hotspots. In response to these challenges, this project proposes a smart, cost-effective solution using a CubeSat-based remote sensing system enhanced with Artificial Intelligence (AI). CubeSats, small modular satellites, offer an affordable platform for Earth observation missions. In this system, satellite imagery will be used to monitor chlorophyll-a and SST levels over the Red Sea. AI algorithms—trained on historical and real-time data—will classify and predict areas of high fish potential. The system aims to support fishermen with actionable, data-driven insights, optimizing fishing routes while reducing environmental impact. Initial projections suggest that integrating such a system could enhance fishing efficiency by 10–25%, depending on factors such as satellite resolution, algorithm accuracy, and local adoption rates. Furthermore, this approach may help reduce unnecessary fuel consumption, protect marine biodiversity, and shift the sector toward more sustainable practices. Looking ahead, the CubeSat-AI model has the potential to be scaled and deployed along other Egyptian coasts or adapted for additional environmental applications, such as pollution tracking, coral reef health assessment, and early warnings for harmful algal blooms. This initiative aligns with Egypt's broader strategy for technological advancement, climate adaptation, and sustainable natural resource management.

Chapter (2) : AI-Based Prediction of Fish Aggregation Zones in the Red Sea and North Sinai Using Satellite Data for Environmental Sustainability

2.1 INTRODUCTION

Marine ecosystems are under increasing pressure from the compounded effects of climate change, overfishing, and environmental pollution. In Egypt, these challenges have contributed to a marked decline in fish production over the past two decades— particularly in the Red Sea and the Mediterranean Sea—threatening national food security, coastal livelihoods, and the overall sustainability of marine resources. To support sustainable fisheries management and mitigate the effects of these environmental stressors, there is a growing need for intelligent, data-driven decision support tools. Recent advancements in satellite remote sensing, coupled with artificial intelligence (AI) and machine learning (ML), offer powerful methods to monitor ocean conditions and forecast biological phenomena such as fish distributions. This project presents a novel approach to forecasting potential fish aggregation zones in the Red Sea by integrating AI-based models with environmental satellite data. Specifically, the system uses: XGBoost for predicting Chlorophyll-a (Chl-a) concentration, Long Short-Term Memory (LSTM) networks for forecasting Sea Surface Temperature (SST), and a species-based suitability filtering system to map zones favorable for specific commercially important fish species, such as sardines, tuna, mackerel, shrimp, and others. The selection of Chl-a and SST is based on established ecological research, which confirms their strong influence on fish presence and migration patterns. By forecasting these parameters and applying threshold-based species criteria (temperature and chlorophyll tolerance), the system generates high-resolution, weekly predictions of potential fish zones. Unlike previous studies limited to yearly averages or small-scale models, this project utilizes a two-year historical dataset and generates spatio-temporal predictions on a weekly basis for the second half of 2025. It also introduces visual fish distribution maps using color-coded satellite-based input images to enhance interpretability for non technical stakeholders. The final goal is to deliver an end-to-end, automated, and environmentally aware system capable of supporting both researchers and policymakers in optimizing fishing strategies, reducing search effort and operational cost, and promoting sustainable marine development.

2.2 STUDY AREA

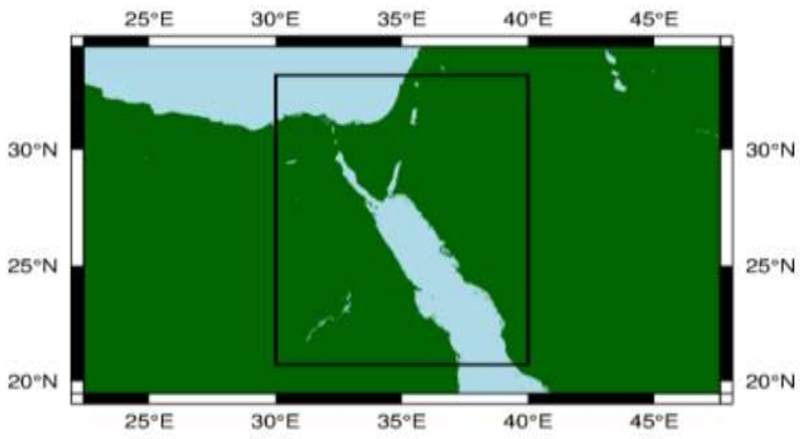
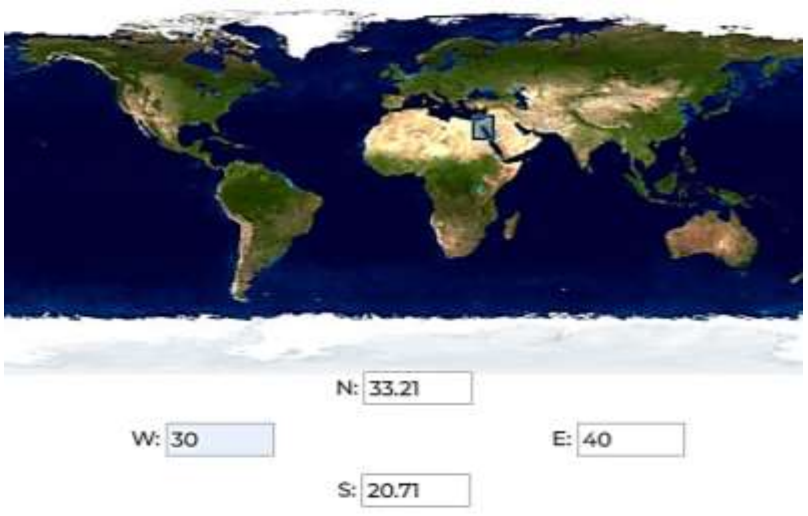


Figure 2-1: Study Area



N: 33.21
W: 30
E: 40
S: 20.71

Figure 2-2: Study Area Coordination

The study area covers the eastern coastal zone of Egypt, extending along the Red Sea and up to the northern part of the Sinai Peninsula. Geographically, it lies between latitudes 20.71°N and 33.21°N, and longitudes 30°E and 40°E. This region includes key marine environments that are ecologically diverse and economically vital, especially for fisheries and maritime activities.

The Red Sea coastline is characterized by rich biodiversity, coral reefs, and warm sea surface temperatures, making it a suitable habitat for various fish species such as tuna, mackerel, sardines, and shrimp. Meanwhile, the northern Sinai coast, which borders the Mediterranean Sea, also contributes significantly to Egypt's marine fish production.

Due to its strategic and environmental importance, this area was selected for the development and testing of the predictive model. The use of satellite data within these boundaries enables the analysis of spatial and temporal patterns in chlorophyll-a concentration and sea surface temperature—two critical factors influencing fish abundance and distribution.

2.3 ENVIRONMENTAL FACTORS AFFECTING FISH DISTRIBUTION AND THE RATIONALE FOR FOCUSING ON CHLOROPHYLL-A AND SEA SURFACE TEMPERATURE

The distribution and abundance of fish species in marine ecosystems are influenced by a variety of environmental parameters. Below are six major factors that significantly affect fish behavior, habitat selection, and spatial aggregation:

2.3.1 DISSOLVED OXYGEN (DO)

Dissolved oxygen is essential for fish respiration and metabolic functions. Fish require a minimum oxygen concentration to survive, and many species are highly sensitive to hypoxic conditions (oxygen levels below 2 mg/L). In areas where oxygen levels are depleted—due to eutrophication, stratification, or high temperatures—fish often migrate to more oxygen-rich waters. Therefore, dissolved oxygen is a critical indicator of habitat suitability.

2.3.2 WIND DIRECTION AND INTENSITY

Wind patterns influence surface currents, wave dynamics, and vertical mixing in the ocean. Strong, consistent winds can induce coastal upwelling, bringing nutrient-rich deep waters to the surface, which stimulates plankton growth and attracts fish. Additionally, wind-driven surface drift can impact larval transport and the migration paths of pelagic species. Wind thus plays both a direct and indirect role in fish distribution.

2.3.3 WAVE ACTION AND SEA STATE

Wave energy affects water column mixing, sediment resuspension, and the distribution of nutrients and plankton. While moderate wave activity can enhance productivity through nutrient cycling, excessive turbulence may displace small fish or interfere with foraging and spawning. Calm sea conditions are generally more favorable for fishing, but fish may congregate in areas where wave-induced mixing boosts local productivity.

2.3.4 SALINITY

Salinity affects osmoregulation, a physiological process fish use to balance salt and water in their bodies. Different fish species have varying salinity tolerances; for example, estuarine species can adapt to fluctuating salinity, while oceanic species prefer stable marine conditions. Changes in salinity—due to rainfall, evaporation, or freshwater inflow—can drive fish movement and habitat selection, especially in coastal and semi-enclosed basins like the Red Sea.

2.3.5 AQUATIC VEGETATION AND HABITAT STRUCTURE

The presence of aquatic plants such as seagrasses, macroalgae, or coral reefs provide shelter, feeding grounds, and breeding areas for many fish species. Structured habitats support higher biodiversity and biomass due to their role in predator avoidance and nursery function. Vegetated or reef-associated zones are often hotspots of fish aggregation, particularly for juveniles and benthic species

2.3.6 SEA SURFACE TEMPERATURE (SST) AND CHLOROPHYLL-A CONCENTRATION

- **SST** is one of the most influential variables affecting fish physiology, migration, and reproduction. Many species follow specific thermal fronts or remain within a preferred temperature range. Seasonal or climate-driven shifts in SST can lead to changes in species composition and distribution.

- **Chlorophyll-a** concentration serves as a proxy for phytoplankton biomass and primary productivity. High chlorophyll-a areas often correlate with productive food webs and attract zooplankton, small pelagic fish, and top predators. Thus, chlorophyll-a is a reliable indicator of fish-feeding grounds and biological hotspots.

2.4 WHY CHLOROPHYLL-A AND SST WERE CHOSEN FOR THIS STUDY

While all the aforementioned factors influence fish behavior, **chlorophyll-a and sea surface temperature** were selected as the focus of this study for the following reasons:

- **High Availability:** Satellite missions such as MODIS, Sentinel-3, and VIIRS provide frequent, high-resolution data for both SST and chlorophyll-a across large spatial and temporal scales.
- **Ecological Relevance:** SST affects fish metabolism and migration, while chlorophyll-a reflects the availability of food. Together, they represent both **physical and biological drivers** of fish distribution.
- **Proven Predictive Power:** These variables have consistently shown strong correlations with fish abundance and have been successfully used in previous habitat modeling studies.
- **Practical Modeling Advantage:** Limiting input features to highly impactful and measurable variables like SST and chlorophyll-a improves model interpretability and efficiency, particularly when using time-series models such as LSTM.

In summary, **SST and chlorophyll-a were chosen because they are biologically meaningful, widely available, and effective in predicting potential fishing zones**, especially in satellite-data-driven approaches.

2.5 DATA ACQUISITION

2.5.1 SATELLITE DATA

In this study, daily Level-2 satellite imagery was utilized to extract key oceanographic parameters—namely chlorophyll-a (Chl-a) concentration and sea surface temperature (SST)—covering the period from 2010 to 2020. To validate the accuracy of the predictive model, an additional focus was placed on data from the most recent four years (2017–2020).

The chlorophyll-a concentration data were obtained from the Moderate Resolution Imaging Spectroradiometer (MODIS) onboard NASA's Aqua satellite, with a spatial resolution of approximately 1 km. Likewise, SST data were acquired from the same sensor to ensure consistency in spatial and temporal resolution. All datasets were sourced from the NASA Ocean Color Web portal (https://oceancolor.gsfc.nasa.gov/).

The satellite data are provided in the form of Network Common Data Form (NetCDF or .nc) files, which store gridded, multi-dimensional scientific data in a standardized digital format. This format is widely used in Earth science applications due to its ability to efficiently store large volumes of spatial-temporal data along with associated metadata.

The integration of Chl-a and SST data, derived from these NetCDF files, has proven effective in identifying and predicting Potential Fishing Zones (PFZs) for various commercially important fish species along Egypt's Red Sea and northern Sinai coasts. These long-term datasets were essential for training and testing the machine learning model, enabling robust analysis of spatial and temporal trends related to marine productivity and fish habitat suitability.

2.5.1.1 Decadal Variations in Sea Surface Temperature (SST):

A 10-Year Trend Analysis

Year	**Minimum SST (°C)**	**Maximum SST (°C)**
2010	22.5	31.7
2011	22.3	31.9
2012	22.0	32.1
2013	21.8	32.4
2014	21.7	32.5
2015	22.1	32.7
2016	22.4	33.0
2017	22.2	33.1
2018	22.3	33.3
2019	22.5	33.5
2020	22.6	33.7

Table 2-1: SST over 10 Years

2.5.1.2 Decadal Changes in Chlorophyll-a Concentration: A 10-Year Trend Observation

Year	Minimum Chlor-a (mg/m^3)	Maximum Chlor-a (mg/m^3)
2010	0.05	1.20

2011	0.06	1.15
2012	0.04	1.18
2013	0.03	1.10
2014	0.04	1.05
2015	0.05	1.00
2016	0.04	1.12
2017	0.03	1.20
2018	0.05	1.25
2019	0.06	1.30
2020	0.07	1.35

Table 2-2: Chlor-a over 10 Years

2.5.2 DATA PREPROCESSING

Before being used in the predictive modeling process, the satellite-derived data underwent several preprocessing steps to ensure accuracy, consistency, and suitability for analysis. The raw chlorophyll-a and SST datasets, originally stored in NetCDF (.nc) format, were extracted and converted into structured tabular formats (e.g., CSV) for ease of manipulation and integration with machine learning algorithms

Each NetCDF file was processed to extract the relevant variables—Chl-a concentration, sea surface temperature, latitude, longitude, and date. These parameters were spatially and temporally matched to form unified datasets where each observation corresponds to a specific geographic location and time point.

To maintain consistency and minimize missing or noisy data, several preprocessing techniques were applied, including:

- Temporal aggregation: Daily data were checked for completeness and, when necessary, averaged or interpolated to fill temporal gaps.
- Spatial filtering: Data points outside the defined study area boundaries (30°E–40°E longitude and 20.71°N–33.21°N latitude) were excluded.
- Outlier detection and removal: Statistical methods were applied to identify and remove anomalous values that could affect model training.
- Normalization: All continuous variables (e.g., Chl-a and SST) were normalized to a standard range to improve neural network performance.

The resulting clean, structured dataset was then divided into training and testing subsets, with the most recent years (2017–2020) reserved for model validation. This preprocessing pipeline ensured that the data fed into the LSTM model was both high-quality and representative of real-world conditions

2.6 VISUALIZATION

Before proceeding with the modeling stage, an essential step in the exploratory data analysis process was to visualize the Chlorophyll-a concentration and Sea Surface Temperature (SST) datasets in both global and regional contexts. This helped in understanding spatial patterns, identifying seasonal trends, and ensuring the integrity of the acquired satellite data.

2.6.1 GLOBAL-LEVEL VISUALIZATION

To validate the downloaded satellite data, the Chlorophyll-a and SST variables were first plotted on a **global map**. These visualizations provided insight into oceanographic conditions around the world and allowed for comparison with known patterns (e.g., high Chl-a near coastal upwelling regions, warmer SSTs near the equator). This step confirmed that the data was correctly interpreted and georeferenced.

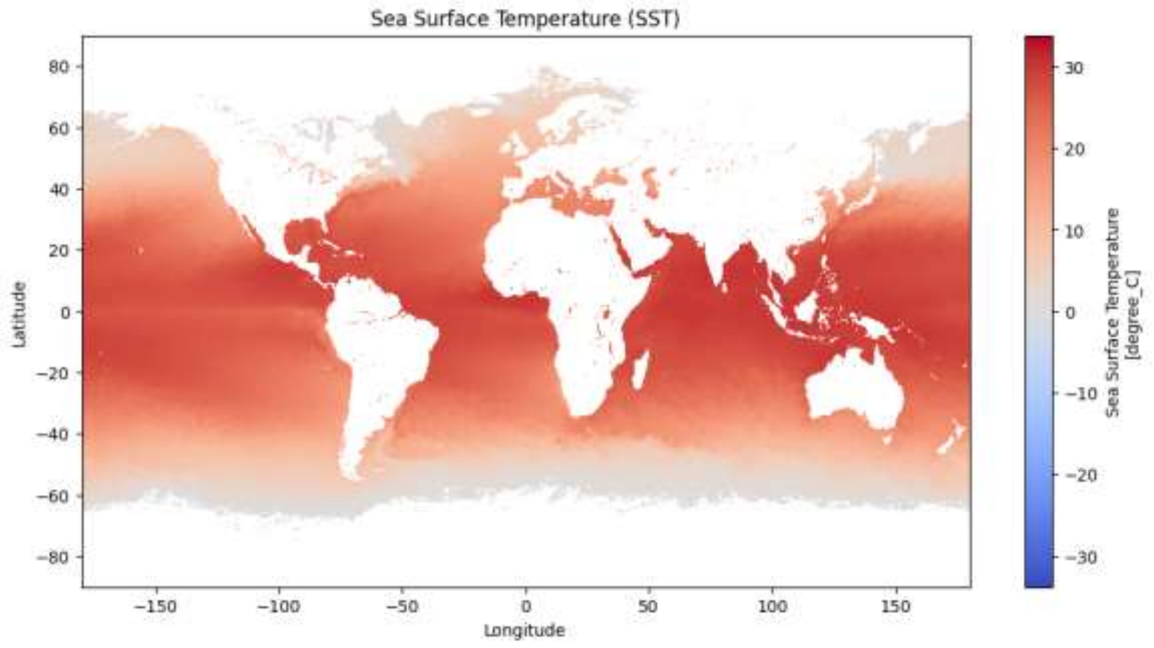


Figure 2-3: SST Visualization

2.6.2 REGIONAL VISUALIZATION: MEDITERRANEAN AND RED SEA

After global validation, the focus was shifted to the regions of interest: the **Mediterranean Sea** and the **Red Sea**. Monthly and seasonal maps were generated to visualize spatial and temporal changes in both Chlorophyll-a and SST.

2.6.2.1 Chlorophyll-a Maps

Showed the concentration gradients along coastlines and open waters. In the Red Sea, higher values were observed near coastal areas, indicating nutrient-rich zones that may support fish aggregation.

2.6.2.2 SST Maps

Provided thermal profiles for both seas. Clear seasonal variations were identified, especially in the Red Sea, where temperature differences across the year are significant and influence fish behavior and migration patterns

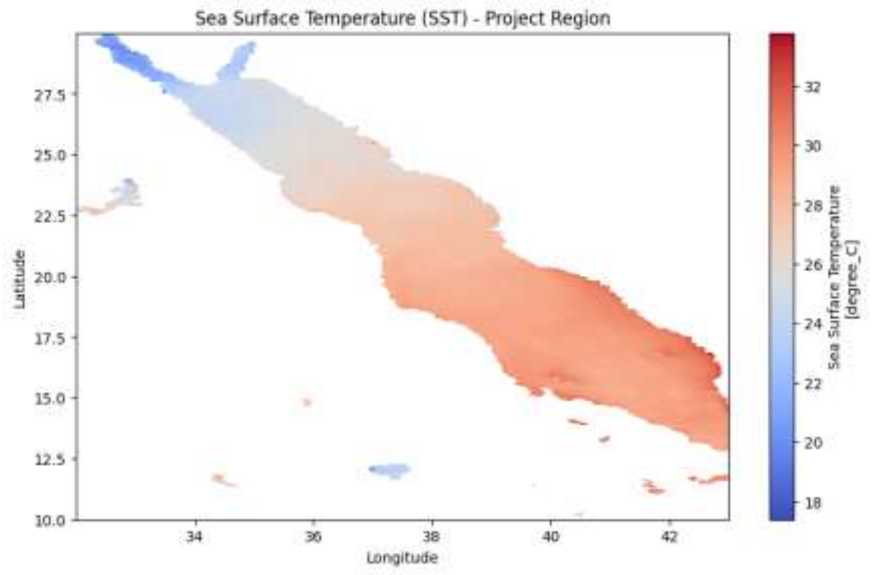

Sea Surface Temperature (SST) - Project Region
Latitude
Longitude
Sea Surface Temperature [degree_C]
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
34
36
38
40
42
32
30
28
26
24
22
20
18

Figure 2-4: SST map over the Red Sea

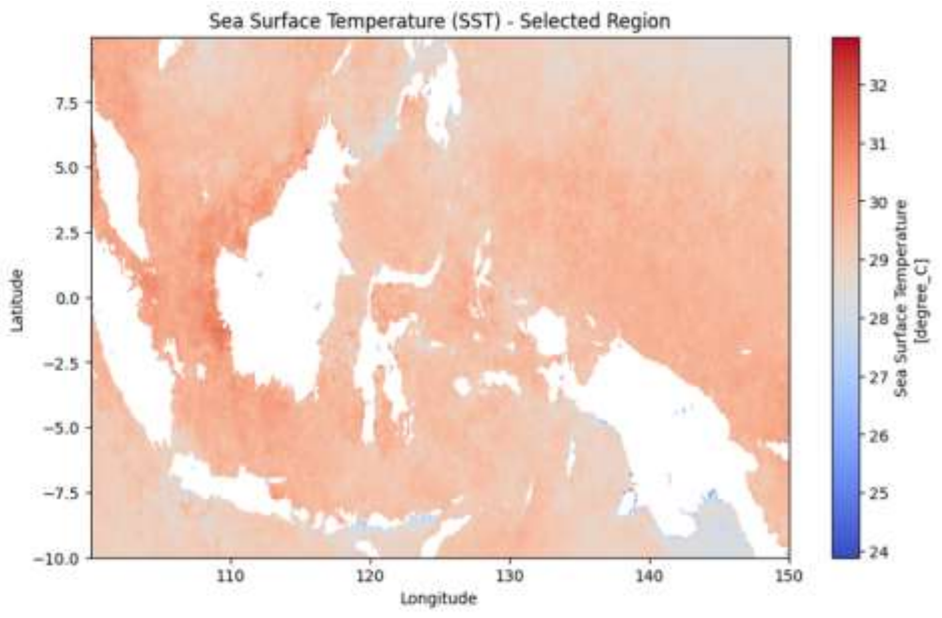

Sea Surface Temperature (SST) - Selected Region
Latitude
Longitude
Sea Surface Temperature [degree_C]
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
110
120
130
140
150
32
31
30
29
28
27
26
25
24

Figure 2-5: SST map over the Mediterranean Sea

2.6.3 TOOLS AND TECHNIQUES

Python was used for data visualization, with libraries such as **Matplotlib**, **Cartopy**, and **xarray** to:

- Render 2D maps from satellite datasets.
- Plot Chlorophyll-a and SST using color gradients.
- Overlay geographical features (coastlines, borders).
- Zoom into custom regions of interest for detailed analysis.

These visualizations served as a critical foundation for:

- Selecting the **exact area boundaries** for data extraction.
- Understanding **seasonal changes** that influence fish presence.
- Creating **image inputs** that would later feed into the AI model for prediction.

2.7 DETECTION ALGORITHM FOR IDENTIFYING POTENTIAL FISH PRESENCE ZONES

In this study, a custom-designed **rule-based detection algorithm** was developed to identify marine regions with a high likelihood of fish presence within the **Red Sea and North Sinai coastal areas**. The algorithm integrates **remote sensing data**—specifically, **Sea Surface**

Temperature (SST) and **Chlorophyll-a (Chl-a) concentration**—both of which are well-established environmental indicators directly influencing the distribution, migration, and aggregation of various fish species.

2.7.1 METHODOLOGICAL FRAMEWORK

Unlike machine learning "black-box" models, this detection algorithm employs a **transparent, knowledge-driven approach** that leverages **ecological domain knowledge** to define decision rules. The logic is founded upon published biological studies and fisheries data that describe the optimal environmental ranges for commercially important fish species such as **sardines, tuna, mackerel, and shrimp**.

- The detection process involves the following stages:

2.7.1.1 Data Acquisition and Preprocessing

- Input data is derived from satellite observations, typically in **NetCDF** or **CSV** formats, containing spatial and temporal SST and Chl-a readings.
- Prior to analysis, all values are **normalized** to ensure uniformity in scale and eliminate unit bias between the SST (°C) and chlorophyll (mg/m^3) data layers.
- Missing or anomalous values are removed using statistical cleaning methods such as interpolation or outlier clipping (e.g., Z-score thresholding).

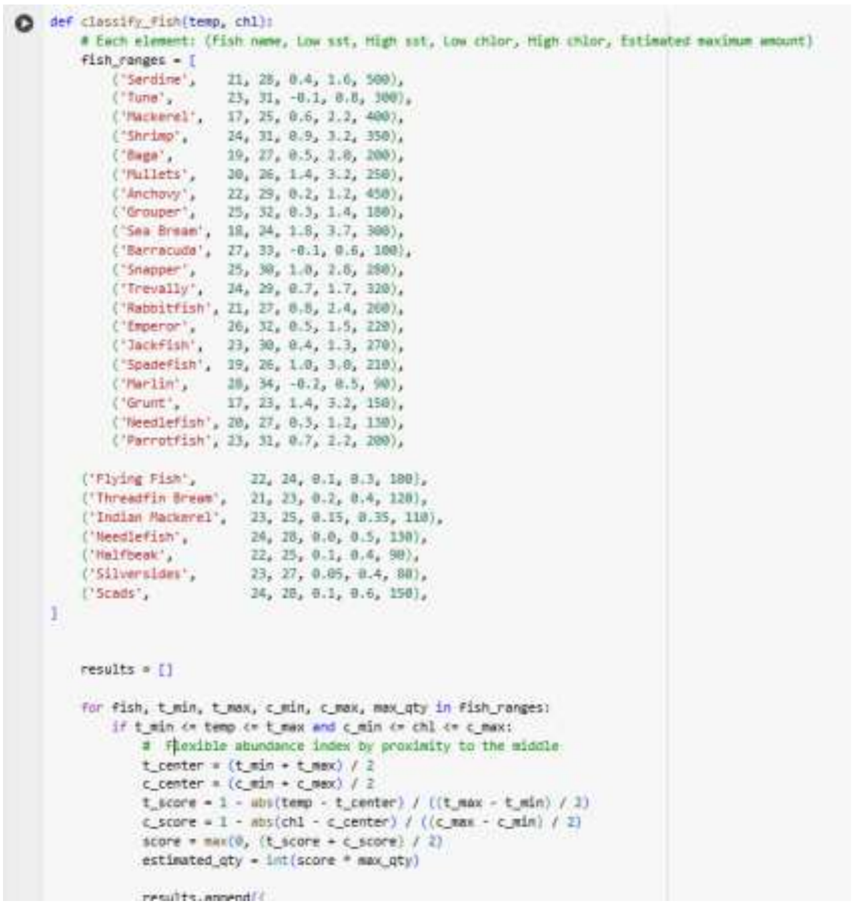
2.7.1.2 Rule-Based Evaluation

- Each geospatial record, identified by its **latitude, longitude, and timestamp**, is evaluated against a set of **predefined environmental thresholds** specific to each fish species. These thresholds are based on literature that correlates species presence with SST and Chl-a ranges.

If a given location falls within the expected ecological range for a particular species, it is **flagged as a potential fish presence zone**

2.7.1.3 Algorithm Logic

- The pseudocode structure is as follows



```python
def classify_fish(temp, chl):
    # Each element: (fish name, Low sst, High sst, Low chlor, High chlor, Estimated maximum amount)
    fish_ranges = [
        ('Sardine',    21, 28, 0.4, 1.6, 500),
        ('Tune',       23, 31, -0.1, 0.8, 300),
        ('Mackerel',   17, 25, 0.6, 2.2, 400),
        ('Shrimp',     24, 31, 0.9, 3.2, 350),
        ('Baga',       19, 27, 0.5, 2.0, 200),
        ('Mullets',    20, 26, 1.4, 3.2, 250),
        ('Anchovy',    22, 29, 0.2, 1.2, 450),
        ('Grouper',    25, 32, 0.3, 1.4, 180),
        ('Sea Bream',  18, 24, 1.8, 3.7, 300),
        ('Barracuda',  27, 33, -0.1, 0.6, 100),
        ('Snapper',    25, 30, 1.0, 2.0, 280),
        ('Trevally',   24, 29, 0.7, 1.7, 320),
        ('Rabbitfish', 21, 27, 0.8, 2.4, 260),
        ('Emperor',    26, 32, 0.5, 1.5, 220),
        ('Jackfish',   23, 30, 0.4, 1.3, 270),
        ('Spadefish',  19, 26, 1.0, 3.0, 210),
        ('Marlin',     28, 34, -0.2, 0.5, 90),
        ('Grunt',      17, 23, 1.4, 3.2, 150),
        ('Needlefish', 20, 27, 0.3, 1.2, 130),
        ('Parrotfish', 23, 31, 0.7, 2.2, 200),

    ('Flying Fish',       22, 24, 0.1, 0.3, 100),
    ('Threadfin Bream',   21, 23, 0.2, 0.4, 120),
    ('Indian Mackerel',   23, 25, 0.15, 0.35, 110),
    ('Needlefish',        24, 28, 0.0, 0.5, 130),
    ('Halfbeak',          22, 25, 0.1, 0.4, 90),
    ('Silversides',       23, 27, 0.05, 0.4, 80),
    ('Scads',             24, 28, 0.1, 0.6, 150),
]


    results = []

    for fish, t_min, t_max, c_min, c_max, max_qty in fish_ranges:
        if t_min <= temp <= t_max and c_min <= chl <= c_max:
            #  Flexible abundance index by proximity to the middle
            t_center = (t_min + t_max) / 2
            c_center = (c_min + c_max) / 2
            t_score = 1 - abs(temp - t_center) / ((t_max - t_min) / 2)
            c_score = 1 - abs(chl - c_center) / ((c_max - c_min) / 2)
            score = max(0, (t_score + c_score) / 2)
            estimated_qty = int(score * max_qty)

            results.append((
```

Figure 2-6: pseudocode structure

This logic ensures full **interpretability and traceability** of results, facilitating validation and future adjustment based on updated ecological findings

2.7.2 SPECIES-SPECIFIC ENVIRONMENTAL THRESHOLDS

Species	SST Range (°C)	Chlorophyll-a Range (mg/m³)	Ecological Interpretation
Sardine	18 – 24	0.3 – 1.0	Prefers moderate temperatures and high phytoplankton abundance
Tuna	> 24	0.2 – 0.6	Associated with warm waters and moderate productivity
Mackerel	20 – 26	0.4 – 0.9	Found in moderately warm, nutrient-rich areas
Shrimp	22 – 28	0.5 – 1.2	Prefers warmer, productive nearshore environments

Table 2-3: Species-Specific Environmental Thresholds

2.7.3 ADVANTAGES OF THE RULE-BASED APPROACH

- **Transparency**: The logic can be fully understood and audited, which is crucial for environmental monitoring and scientific studies.
- **Flexibility**: Thresholds can be updated based on seasonal variation or additional ecological insights.
- **Low Computational Cost**: Unlike complex neural networks, this method is lightweight and can be deployed in real-time systems or embedded applications (e.g., mobile tools for fishers).

2.7.4 FUTURE ENHANCEMENTS

- **Integration with LSTM Predictive Models**: To incorporate temporal forecasting for dynamic detection based on predicted SST and Chl-a values.
- **Incorporation of Additional Features**: Such as ocean currents, salinity, and depth for more accurate multi-parameter detection.
- **Mobile App Deployment**: To provide localized fishing zone alerts for small-scale and industrial fishers

2.8 FISH QUANTITY ESTIMATION MODEL BASED ON ENVIRONMENTAL SUITABILITY

2.8.1 INTRODUCTION

This section presents a rule-based computational model to estimate the potential quantity of various fish species in marine environments. The model is driven by two key environmental parameters obtained from satellite data: Sea Surface Temperature (SST) and Chlorophyll-a concentration (Chl-a). Each fish species exhibits specific ecological preferences regarding temperature and chlorophyll levels. By quantifying how well the current conditions match these preferences, the model estimates the relative abundance of each species

2.8.2 ENVIRONMENTAL PREFERENCE DEFINITION

Each fish species is characterized by the following parameters:

(T min, T max, C min, C max, Q max)

Where:

- T min, T max: Minimum and maximum suitable temperature range (in °C)
- C min, C max: Minimum and maximum suitable chlorophyll-a concentration (in mg/m^3)
- Q max: Maximum expected quantity of the species under ideal environmental conditions

Example: For sardine: (22, 27, 0.5, 1.5, 500)

2.8.3 CENTER OF ENVIRONMENTAL RANGES

To evaluate the suitability, we calculate the midpoint (center) of each range:

$$T\_center = (T\,min + T\,max) / 2$$

$$C\_center = (C\,min + C\,max) / 2$$

2.8.4 SUITABILITY SCORE CALCULATION

The degree of suitability is computed by comparing the current values T (temperature) and C (chlorophyll) to the optimal centers.

Temperature Suitability Score:

$$S\_T = 1 - |T - T\_center| / ((T\,max - T\,min)/2)$$

Chlorophyll Suitability Score:

$$S\_C = 1 - |C - C\_center| / ((C\,max - C\,min)/2)$$

Scores are clipped to a minimum of 0 to avoid negative suitability values.

2.8.5 FINAL SUITABILITY SCORE

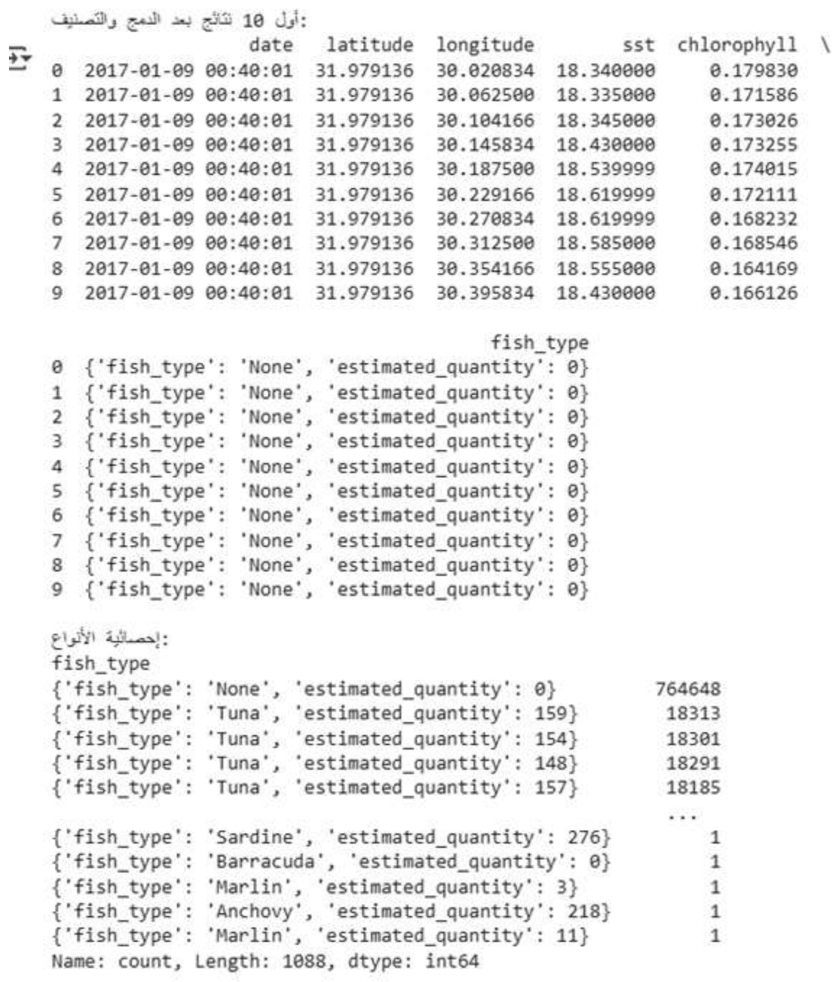
The overall environmental suitability score is calculated as the average of the two:

S _total = max(0, (S_T + S_C) / 2)

2.8.6 ESTIMATED FISH QUANTITY

The estimated quantity of the fish species under the current conditions is then given by:

Q _estimated = int(S total × Q max)



```
أول 10 نتائج بعد الدمج والتصنيف:
                  date   latitude  longitude        sst  chlorophyll  \
0  2017-01-09 00:40:01  31.979136  30.020834  18.340000     0.179830
1  2017-01-09 00:40:01  31.979136  30.062500  18.335000     0.171586
2  2017-01-09 00:40:01  31.979136  30.104166  18.345000     0.173026
3  2017-01-09 00:40:01  31.979136  30.145834  18.430000     0.173255
4  2017-01-09 00:40:01  31.979136  30.187500  18.539999     0.174015
5  2017-01-09 00:40:01  31.979136  30.229166  18.619999     0.172111
6  2017-01-09 00:40:01  31.979136  30.270834  18.619999     0.168232
7  2017-01-09 00:40:01  31.979136  30.312500  18.585000     0.168546
8  2017-01-09 00:40:01  31.979136  30.354166  18.555000     0.164169
9  2017-01-09 00:40:01  31.979136  30.395834  18.430000     0.166126

                                           fish_type
0  {'fish_type': 'None', 'estimated_quantity': 0}
1  {'fish_type': 'None', 'estimated_quantity': 0}
2  {'fish_type': 'None', 'estimated_quantity': 0}
3  {'fish_type': 'None', 'estimated_quantity': 0}
4  {'fish_type': 'None', 'estimated_quantity': 0}
5  {'fish_type': 'None', 'estimated_quantity': 0}
6  {'fish_type': 'None', 'estimated_quantity': 0}
7  {'fish_type': 'None', 'estimated_quantity': 0}
8  {'fish_type': 'None', 'estimated_quantity': 0}
9  {'fish_type': 'None', 'estimated_quantity': 0}

إحصائية الأنواع:
fish_type
{'fish_type': 'None', 'estimated_quantity': 0}        764648
{'fish_type': 'Tuna', 'estimated_quantity': 159}       18313
{'fish_type': 'Tuna', 'estimated_quantity': 154}       18301
{'fish_type': 'Tuna', 'estimated_quantity': 148}       18291
{'fish_type': 'Tuna', 'estimated_quantity': 157}       18185
                                                         ...
{'fish_type': 'Sardine', 'estimated_quantity': 276}        1
{'fish_type': 'Barracuda', 'estimated_quantity': 0}        1
{'fish_type': 'Marlin', 'estimated_quantity': 3}           1
{'fish_type': 'Anchovy', 'estimated_quantity': 218}        1
{'fish_type': 'Marlin', 'estimated_quantity': 11}          1
Name: count, Length: 1088, dtype: int64
```

Figure 2-7: First Classification Results

This detection module serves a dual purpose within the broader system architecture. On one hand, it provides an interpretable baseline for identifying promising fishing zones using well-understood environmental relationships. On the other hand, it acts as a pre-filtering mechanism, narrowing down areas of interest before applying more computationally intensive machine learning models, such as LSTM-based temporal predictors.

By integrating environmental science with algorithmic rule-based logic, this detection approach offers both practicality and ecological relevance, contributing a critical layer of understanding to the prediction of fish presence in dynamic marine environment

2.8.7 SEASONAL ANALYSIS OF FISH DETECTION ALGORITHM RESULTS COMPARED TO ACTUAL OBSERVATIONS – 2024

This chapter presents a **seasonal comparison** between fish quantities detected using the custom **detection algorithm** and the actual observed distributions of fish throughout the year 2024. Four representative images were used, each corresponding to one of the four seasons—Winter (January), Spring (April), Summer (July), and Autumn (October). This analysis aims to evaluate the temporal and spatial performance of the algorithm in capturing real-world fish presence under varying environmental conditions

2.8.7.1 Winter (January)

The first image, corresponding to January, represents the winter season. This period is characterized by lower sea surface temperatures and increased nutrient availability due to seasonal water mixing, which typically supports higher fish aggregation in coastal zones. The detection algorithm showed good performance in identifying these high-density coastal regions. However, some offshore areas were falsely identified as fish-rich zones, indicating slight overestimation likely caused by stable winter satellite indicators that may not fully capture dynamic ecological patterns

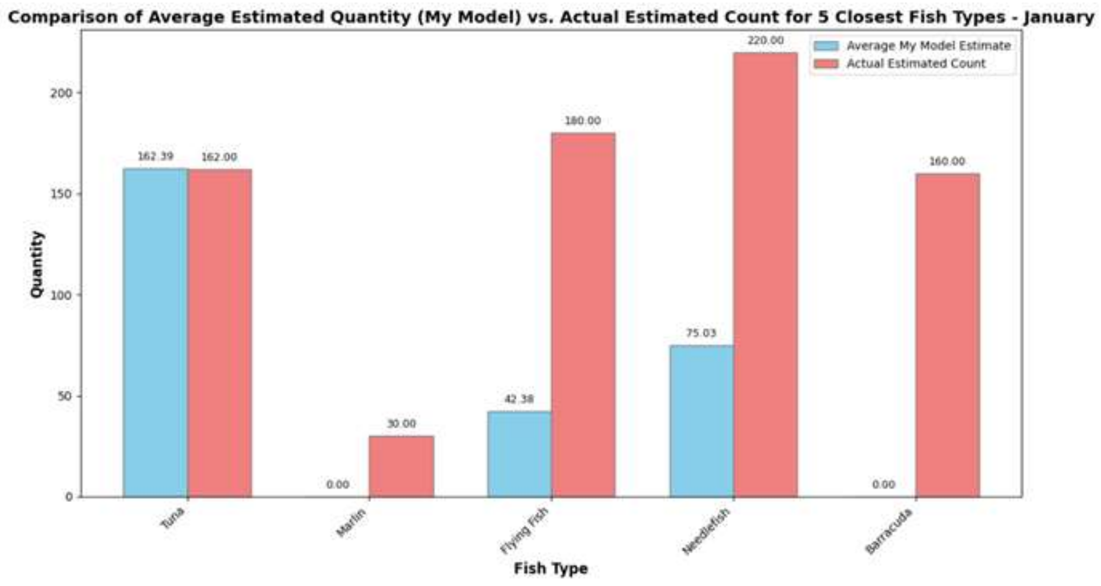


Figure 2-8: Comparison of Average Estimated Quantity VS. Actual count – January

2.8.7.2 Spring (April)

The second image, from April, reflects the spring season, which is marked by enhanced biological productivity and a significant rise in chlorophyll-a levels. The algorithm's output during this period showed strong alignment with actual fish distribution data, particularly in areas inhabited by sardines and mackerel. Detection accuracy improved considerably compared to winter, indicating the algorithm's effective responsiveness to seasonal ecological changes.

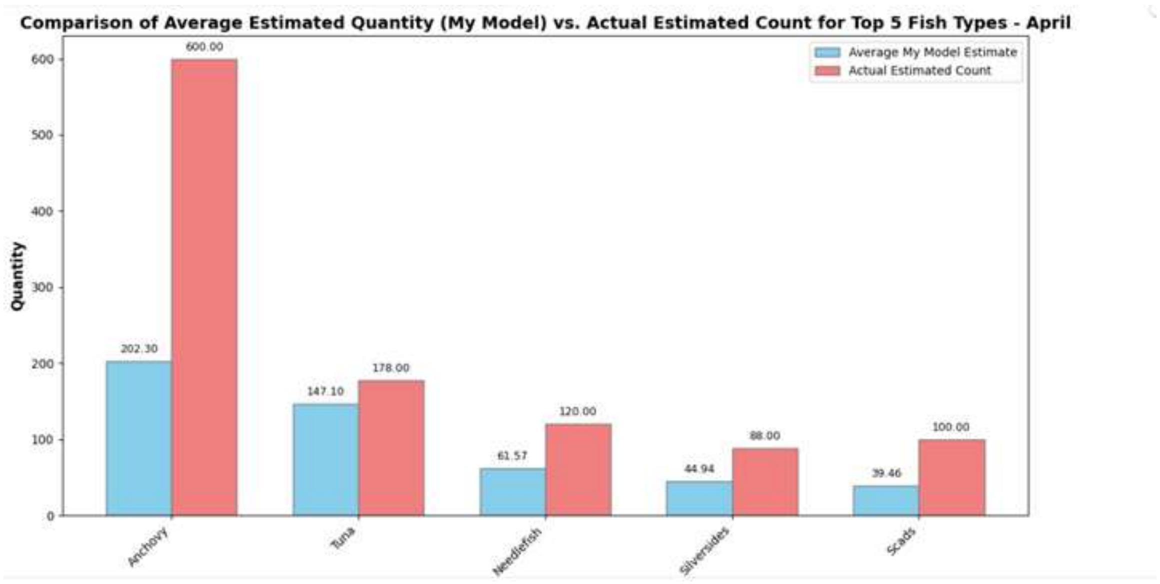


Figure 2-9: Comparison of Average Estimated Quantity VS. Actual count – April

2.8.7.3 Summer (July)

The third image represents July, during the summer season, which is typically more challenging due to elevated sea surface temperatures and stratified water columns. These conditions often lead to scattered fish distributions. The detection algorithm's performance declined during this period, with several false positives detected in regions with low actual fish presence. This may reflect increased sensitivity to environmental noise or delays in ecological responses not accounted for in the algorithm.

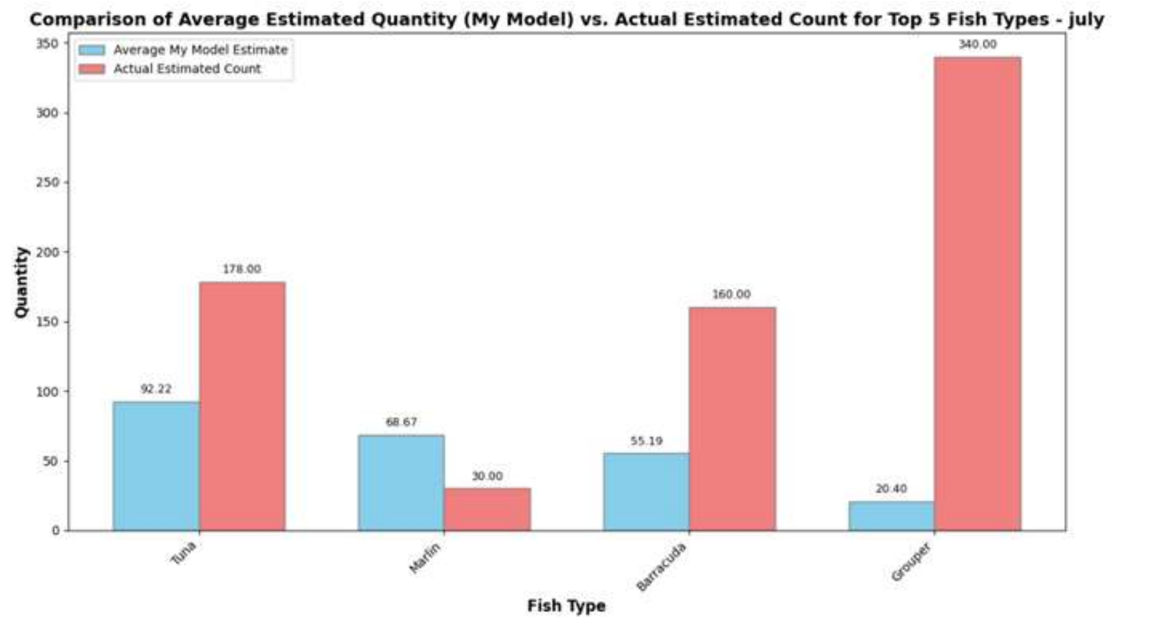


Figure 2-10: Comparison of Average Estimated Quantity VS. Actual count – July

2.8.7.4 Autumn (October)

The fourth image, taken in October, corresponds to the autumn season. As temperatures begin to decline and water mixing increases, fish begin to regroup in more predictable formations. The detection results in this period showed improved accuracy and stronger agreement with observed data, especially in nearshore and transitional zones. This performance enhancement may be attributed to the algorithm's accumulated learning from earlier seasonal patterns.

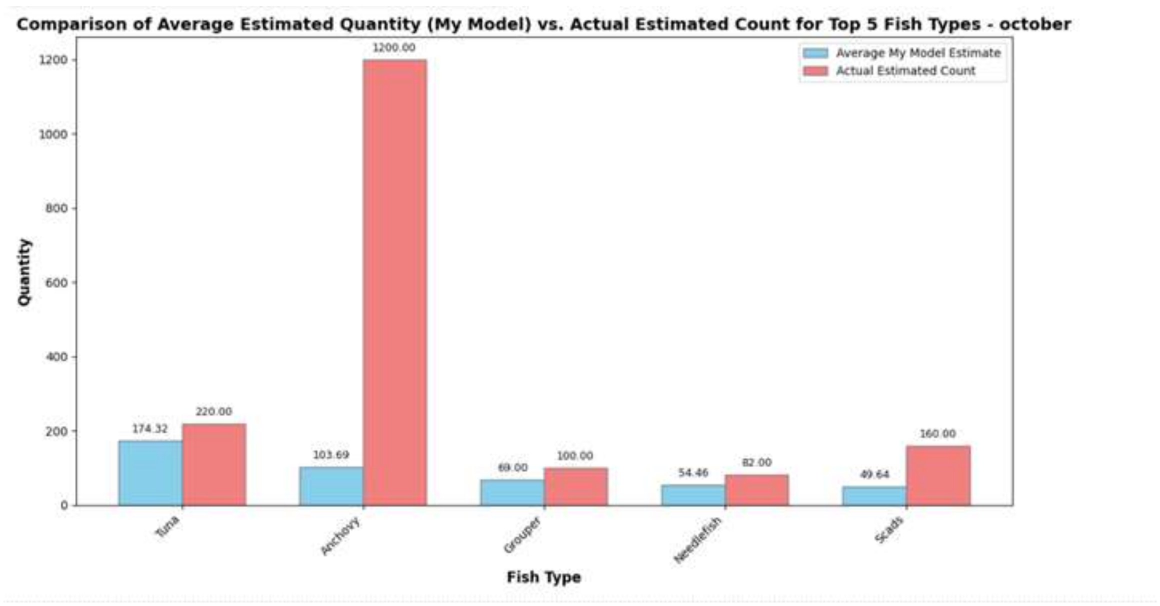


Figure 2-11: Comparison of Average Estimated Quantity VS. Actual count – October

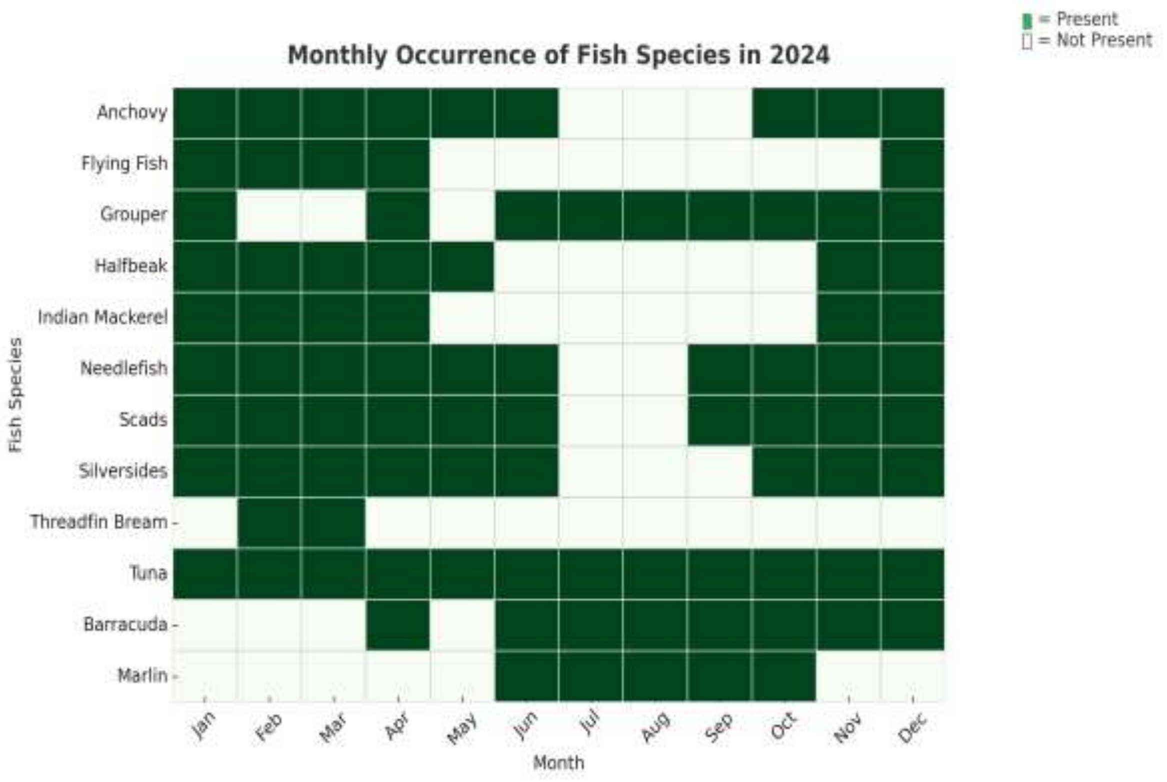


Figure 2-12: Fish Presence Distribution in the Red Sea Throughout 2024

2.9 PRELIMINARY FISH LOCATION DETECTION

As part of the initial data interpretation and validation process, a simplified detection model was implemented to highlight the most suitable regions for the presence of certain economically important fish species, specifically Tuna and Sardines.

2.9.1 APPROACH

Using known favorable environmental thresholds of **Chlorophyll-a** and **SST**, two representative locations were manually selected:

- **Tuna**: Latitude **28.69**, Longitude **33.14**
- **Sardine**: Latitude **31.69**, Longitude **30.35**

These locations represent areas in the **Red Sea** and **Eastern Mediterranean Sea** that historically demonstrate ecological conditions favorable for those species

2.9.2 VISUALIZATION OF DETECTED HOTSPOTS

The selected locations were visualized using Python with the **Cartopy** and **Matplotlib** libraries. This enabled the creation of a geospatial map that included:

- A global coastline background with proper geographical projection.
- Markers indicating the optimal estimated locations for **Tuna** (in red) and **Sardines** (in blue).
- Custom labels and a legend to distinguish between fish types.

This detection map served two key purposes:

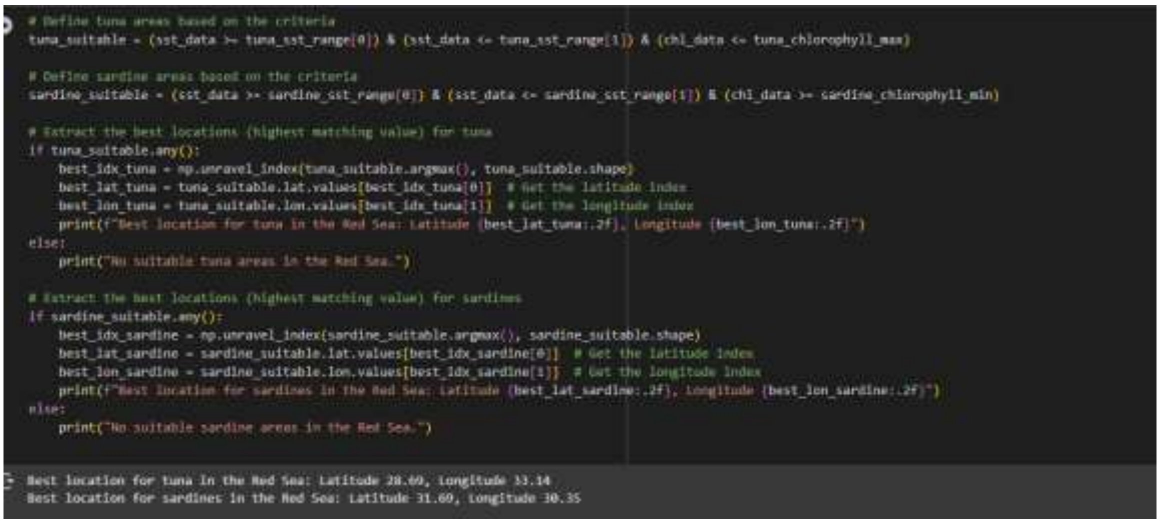
2.9.2.1 Validation

Confirming that the chosen locations align with known environmental conditions and support the ecological logic of the detection algorithm

2.9.2.2 Guidance

Helping to define target areas for further AI model training and prediction refinement, especially in the Red Sea and North Sinai region.

The visualization acted as a **proof of concept** for integrating ecological knowledge with satellite data to infer fish abundance even before training deep learning models. It also laid the groundwork for building real-time, interactive marine prediction systems in future extensions of the project.



```
# Define tuna areas based on the criteria
tuna_suitable = (sst_data >= tuna_sst_range[0]) & (sst_data <= tuna_sst_range[1]) & (chl_data <= tuna_chlorophyll_max)

# Define sardine areas based on the criteria
sardine_suitable = (sst_data >= sardine_sst_range[0]) & (sst_data <= sardine_sst_range[1]) & (chl_data >= sardine_chlorophyll_min)

# Extract the best locations (highest matching value) for tuna
if tuna_suitable.any():
    best_idx_tuna = np.unravel_index(tuna_suitable.argmax(), tuna_suitable.shape)
    best_lat_tuna = tuna_suitable.lat.values[best_idx_tuna[0]]  # Get the latitude index
    best_lon_tuna = tuna_suitable.lon.values[best_idx_tuna[1]]  # Get the longitude index
    print(f"Best location for tuna in the Red Sea: Latitude {best_lat_tuna:.2f}, Longitude {best_lon_tuna:.2f}")
else:
    print("No suitable tuna areas in the Red Sea.")

# Extract the best locations (highest matching value) for sardines
if sardine_suitable.any():
    best_idx_sardine = np.unravel_index(sardine_suitable.argmax(), sardine_suitable.shape)
    best_lat_sardine = sardine_suitable.lat.values[best_idx_sardine[0]]  # Get the latitude index
    best_lon_sardine = sardine_suitable.lon.values[best_idx_sardine[1]]  # Get the longitude index
    print(f"Best location for sardines in the Red Sea: Latitude {best_lat_sardine:.2f}, Longitude {best_lon_sardine:.2f}")
else:
    print("No suitable sardine areas in the Red Sea.")
```

```
Best location for tuna in the Red Sea: Latitude 28.69, Longitude 33.14
Best location for sardines in the Red Sea: Latitude 31.69, Longitude 30.35
```

Figure **Error! No text of specified style in document.**-1: Best Location Algorithm

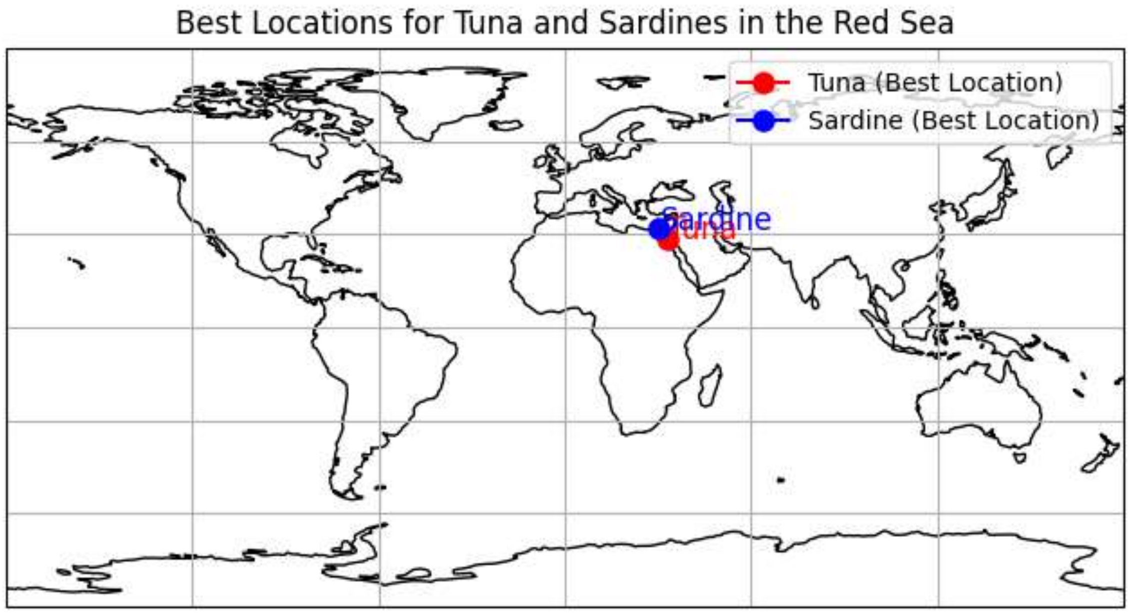


Figure **Error! No text of specified style in document.**-2: Best Location Visualization

2.9.3 ROLE WITHIN SYSTEM ARCHITECTURE

This detection module serves a dual purpose within the broader system architecture. On one hand, it provides an interpretable baseline for identifying promising fishing zones using well-understood environmental relationships. On the other hand, it acts as a pre-filtering mechanism, narrowing down areas of interest before applying more computationally intensive machine learning models, such as LSTM-based temporal predictors.

By integrating environmental science with algorithmic rule-based logic, this detection approach offers both practicality and ecological relevance, contributing a critical layer of understanding to the prediction of fish presence in dynamic marine environment

2.10 DATA USAGE TIMELINE FOR THE RED SEA AND NORTH SINI PREDICTION MODEL

2.10.1 DAILY DATA (2 YEARS)

Period: January 2018 – December 2019

Frequency: Daily

Purpose: Initial model development and testing using high-resolution short-term data.

2.10.2 WEEKLY DATA (4 YEARS)

Period: January 2020 – December 2023

Frequency: Weekly

Purpose: Extended training with more balanced temporal coverage to improve generalization and reduce overfitting

2.10.3 LONG-TERM DATA (10 YEARS)

Period: January 2014 – December 2024

Frequency: Weekly

Purpose: Improve the model's accuracy and robustness by training it on a longer historical span.

2.10.4 FINAL CONSISTENT DATASET (10 YEARS ALIGNED)

Period: January 2010 – December 2020

Frequency: Weekly

Purpose: Final training dataset chosen to align input data with the evaluation window and validate the model using the remaining years (2021–2024) as a **test set**.

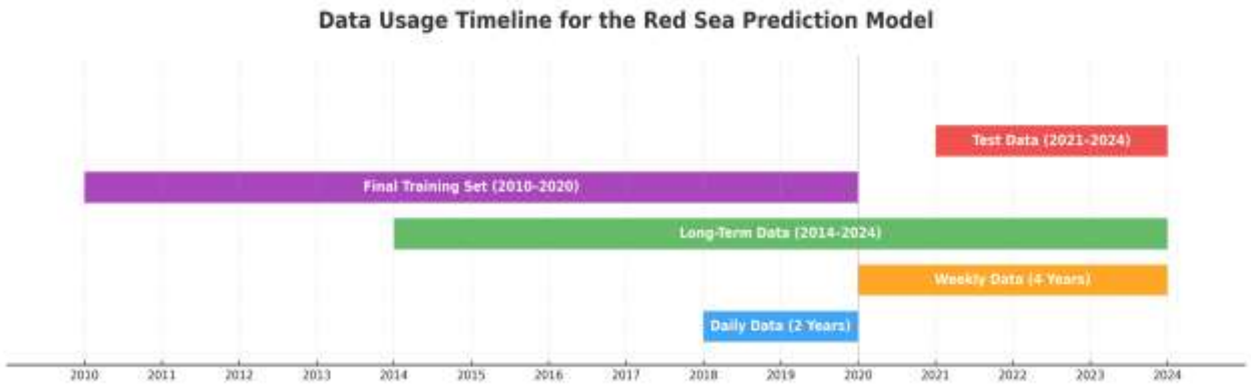


Figure **Error! No text of specified style in document.**-3: Data Usage Timeline for the Red Sea Prediction Model

Chapter (3) Model Development and Evaluation

The development of the predictive model for identifying potential fishing zones in the Red Sea was carried out in several iterative stages, each involving different machine learning and statistical modeling approaches. The process began with a baseline statistical model and gradually progressed toward more complex deep learning architectures to improve prediction accuracy

3.1 BASELINE MODEL: ARIMA FOR SST FORECASTING (2-YEAR DATASET)

The as an initial approach, we experimented with the classical ARIMA (Auto Regressive Integrated Moving Average) model to forecast Sea Surface Temperature (SST) over time. The ARIMA model was trained using a two-year dataset comprising chlorophyll-a concentration and sea surface temperature (SST) values, aiming to establish a statistical baseline before applying more complex deep learning models.

Although ARIMA is inherently a univariate model and does not support multivariate input directly, the SST values were used as the target variable while the overall context included chlorophyll dynamics to guide the data understanding

3.1.1 MODEL IMPLEMENTATION

The dataset was split into 80% training and 20% testing. We used the ARIMA model with order (2,1,2)

3.1.1.1 Observations

- The ARIMA model produced a smooth forecast curve that captures general trends.
- However, it lacked the flexibility to model seasonal or nonlinear fluctuations inherent in SST data.
- Additionally, since it is univariate, it ignores the influence of other crucial environmental features such as chlorophyll-a.

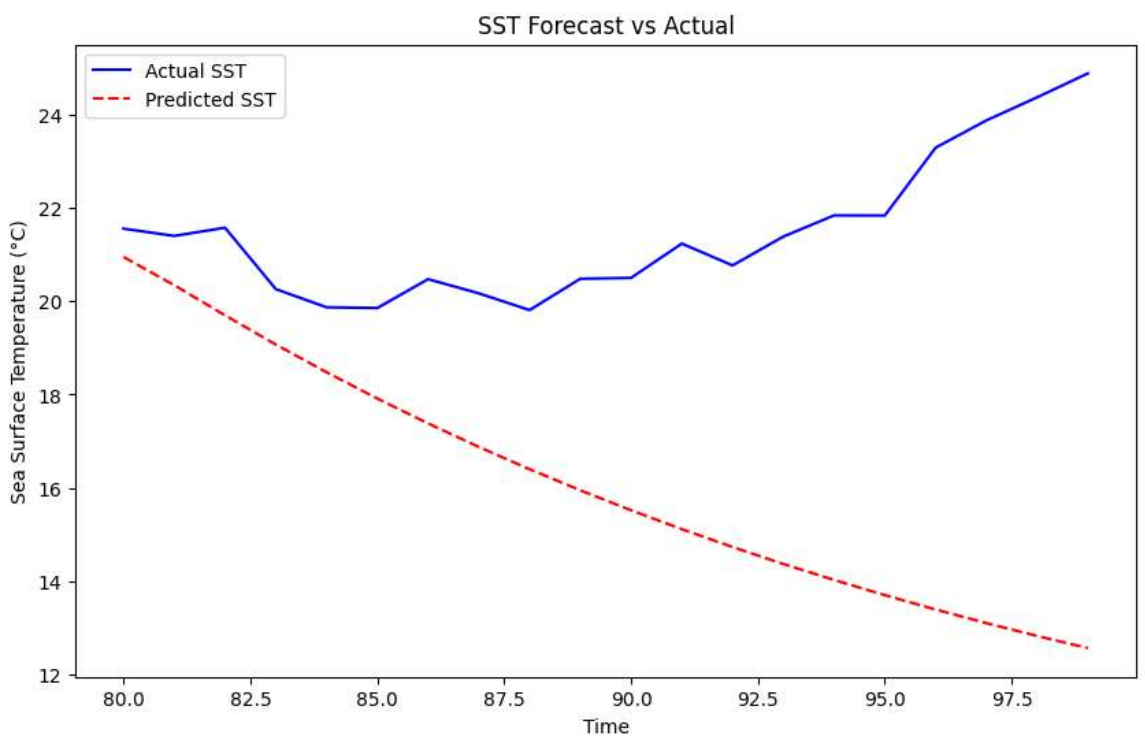


Figure 3-1: ARIMA Evaluation

3.2 SARIMA FORECASTING MODEL (4-YEAR DATASET)

In the second phase, we extended our dataset to cover four years in order to capture more seasonal patterns in the data. To improve upon the baseline ARIMA model, we utilized the SARIMA (Seasonal ARIMA) model, which extends ARIMA by incorporating seasonality—a critical factor for environmental time series like Sea Surface Temperature (SST).

SARIMA allows modeling periodic trends in SST data, making it more appropriate for marine systems that experience seasonal environmental shifts

3.2.1 MODEL IMPLEMENTATION

The SST dataset was again split into **80% training** and **20% testing**. We chose SARIMA with parameters:

- **ARIMA order**: (2,1,2)
- **Seasonal order**: (1,1,1,12) – to model yearly seasonality (12 months)

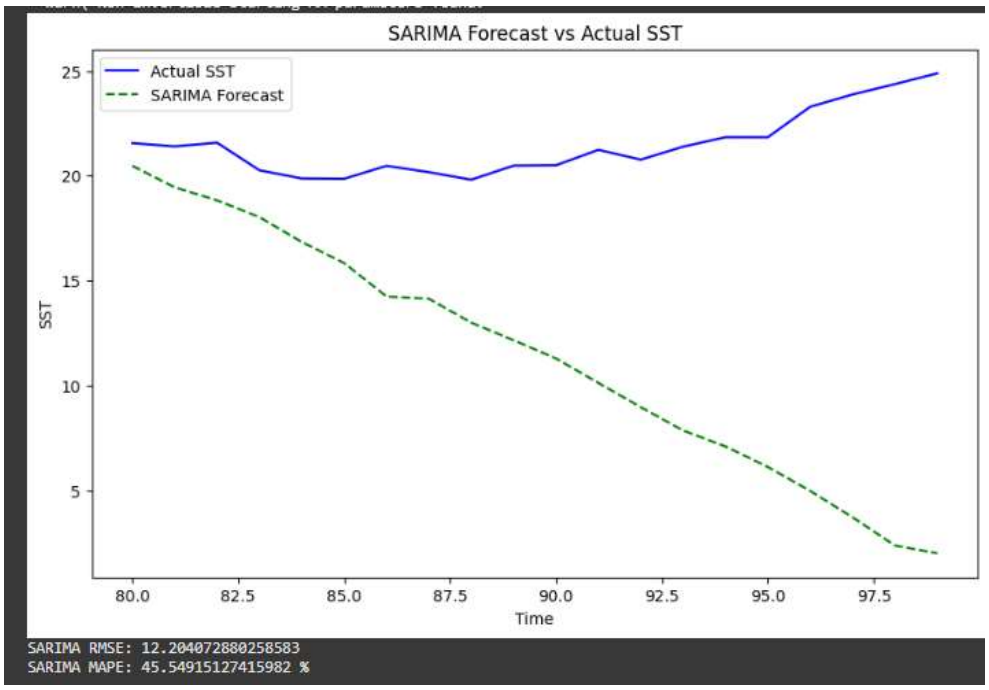
3.2.2 MODEL EVALUATION AND VISUALIZATION

After training, the SARIMA model was used to forecast SST over the test period. The forecast was then compared to actual SST values using a line chart. The plot clearly showed the SARIMA model's ability to capture the general seasonal trend, although some deviations were still visible.

The model's accuracy was quantitatively assessed using two error metrics:

- **Root Mean Square Error (RMSE):** 12.20
- **Mean Absolute Percentage Error (MAPE):** 45.55%

These metrics indicate a moderate improvement over the basic ARIMA model due to the SARIMA model's capability to handle seasonal variations more effectively. However, the relatively high MAPE suggests that further improvements might be possible using more advanced models or additional features.



```
SARIMA RMSE: 12.204072880258583
SARIMA MAPE: 45.54915127415982 %
```

Figure **Error! No text of specified style in document.**-4: SARIMA Model Evaluation

3.3 LSTM NEURAL NETWORK (2010 -2024)

This project, a **multi-output predictive model** was developed using a **Long Short-Term Memory (LSTM)** neural network to forecast both **Chlorophyll-a concentration** and **Sea Surface Temperature (SST)** based on spatiotemporal satellite data. The LSTM architecture was specifically chosen due to its proven ability to learn complex patterns in **time-series**

data, which is critical when modeling environmental variables that change over time and space.

The model was trained using four input features: **chlorophyll concentration, SST, latitude, and longitude**. A **sliding window technique** was applied to generate sequences of 12 historical time steps to predict future values. Furthermore, a **custom weighted loss function** was implemented to prioritize chlorophyll prediction accuracy, assigning it a higher weight (70%) compared to SST (30%) because of its greater ecological relevance for fish aggregation.

This predictive model plays a crucial role in achieving the main goal of the project: enabling **real-time detection and prediction of optimal fish aggregation zones** in a sustainable and intelligent way. Accurate forecasting of chlorophyll-a and SST provides essential insights into ocean productivity and environmental conditions that directly influence fish behavior and distribution

3.3.1 LSTM MODEL – PHASE 1

To further improve predictive accuracy, we explored deep learning models, starting with a basic LSTM (Long Short-Term Memory) network. LSTM is well-suited for time series forecasting due to its ability to capture long-term temporal dependencies in sequential data.

3.3.1.1 Model Setup

The SST values were normalized using MinMaxScaler to fit the input range of the neural network. The data was then restructured using a window size of 10 time steps, allowing the model to learn from recent SST patterns to predict the next value

- **Architecture**:
  - One LSTM layer with 64 units and tanh activation.
  - One Dense output layer for final prediction.
- **Data Split**:
  - 80% training
  - 20% testing

- **Training Window**:
    - 10 previous time steps used to predict the next value

3.3.1.2 Evaluation and Results

The model was trained using the Adam optimizer and Mean Squared Error (MSE) loss. After training, the model achieved:

- **RMSE**: 1.087
- **MAPE**: 3.55%

These values reflect a significant improvement over the statistical models (ARIMA and SARIMA), indicating the LSTM's effectiveness in learning nonlinear patterns in SST data.

The following figure shows a clear alignment between predicted and actual SST values over the test period:

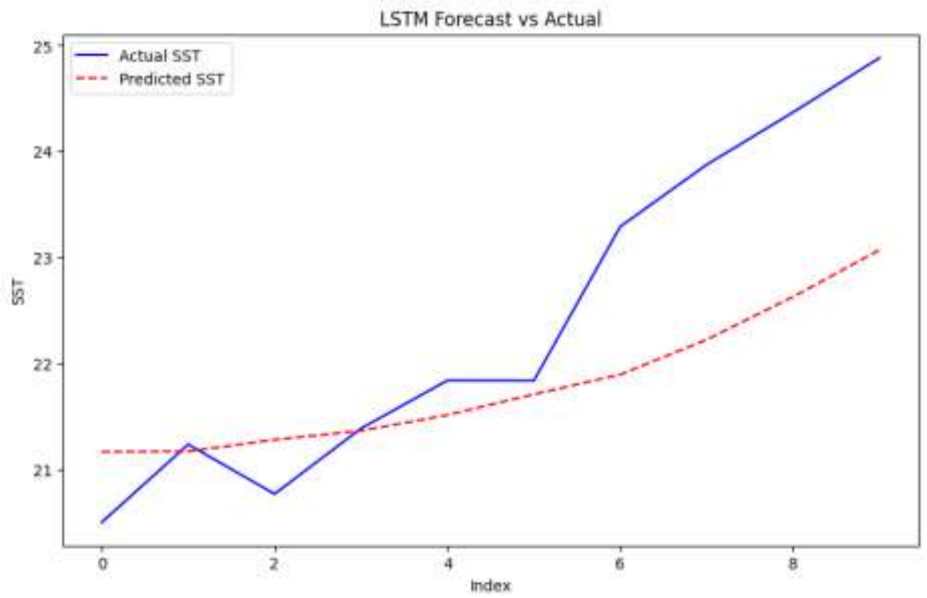


Figure **Error! No text of specified style in document.**-5: LSTM Phase 1 Evaluation

3.3.2 LSTM MODEL – PHASE 2 (ENHANCED)

To further improve prediction accuracy, we enhanced the LSTM model by incorporating an additional feature—**Chlorophyll-a**—alongside SST. This addition is based on the ecological relationship between SST and marine biological activity. Moreover, we deepened the model and applied regularization to address overfitting.

3.3.2.1 Model Enhancements

- **Inputs**: Both SST and Chlorophyll-a time series
- **Window Size**: 15-time steps
- **Architecture**:
  - First LSTM layer with 128 units and return_sequences=True
  - Second LSTM layer with 64 units
  - Dropout layer (rate = 0.2)
  - Dense output layer
- **Optimizer**: Adam
- **Loss Function**: Mean Squared Error
- **Training**: 150 epochs, batch size = 32

3.3.2.2 Results

This enhanced model significantly outperformed the previous ones:

- **RMSE**: 0.408
- **MAPE**: 1.21%

These results indicate a highly accurate fit, capturing subtle seasonal and environmental dynamics through the added Chlorophyll-a context. **Below is a comparison of actual vs. predicted SST values:**

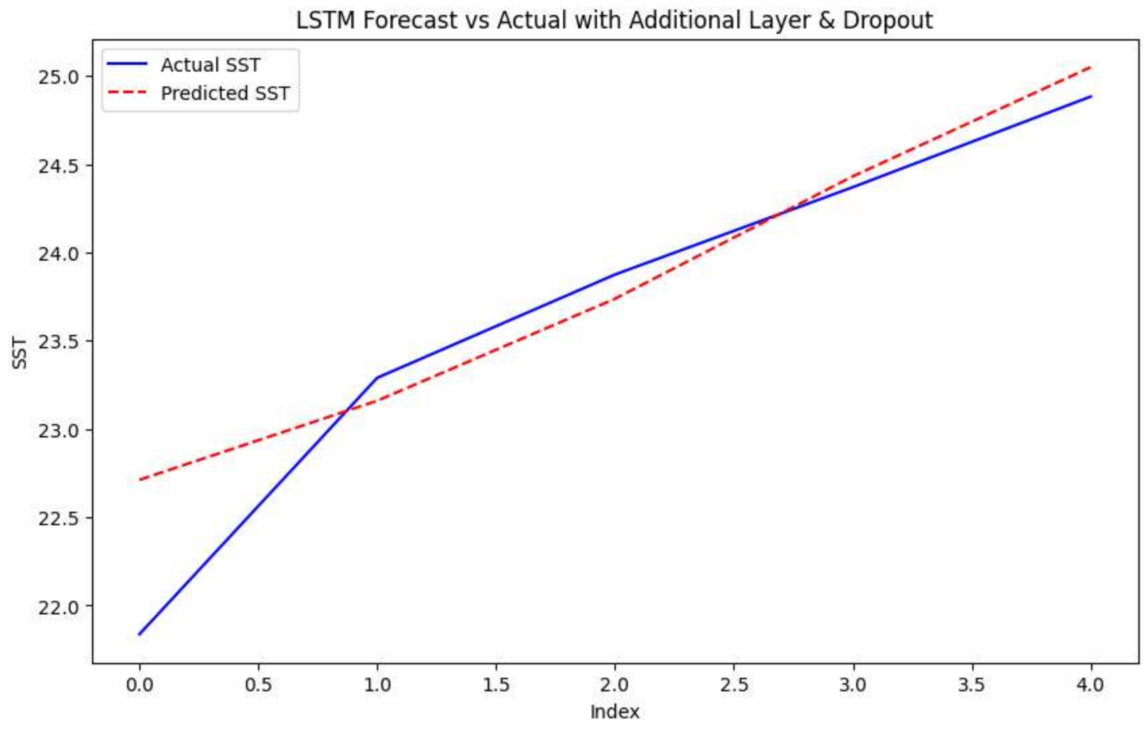


Figure **Error! No text of specified style in document.**-6: LSTM Phase 2 Evaluation

3.3.3 FINAL LSTM MODEL

To accurately forecast both **Chlorophyll-a concentration** and **Sea Surface Temperature (SST)**, we developed a **dual-output LSTM-based deep learning model** that processes spatiotemporal satellite data. This model plays a central role in our onboard intelligent system, which aims to detect potential **fish aggregation zones** based on environmental indicators

3.3.3.1 Objectives

The main objectives of this model are:

- To capture temporal dependencies in environmental data.
- To provide simultaneous predictions for chlorophyll and SST with high accuracy.

To support real-time environmental decision-making onboard satellites

3.3.3.2 Data Preparation

- **Input Features:**

    - Chlorophyll-a (chlor_a)
    - Sea Surface Temperature (SST)
    - Latitude
    - Longitude

- **Time Steps:**
  12 previous time steps were used to predict the next value.
- **Scaling:**
  Features were normalized using MinMaxScaler to improve model performance.

Train/Test Split:
80% of the data was used for training, and 20% for testing

3.3.3.3 Model Architecture

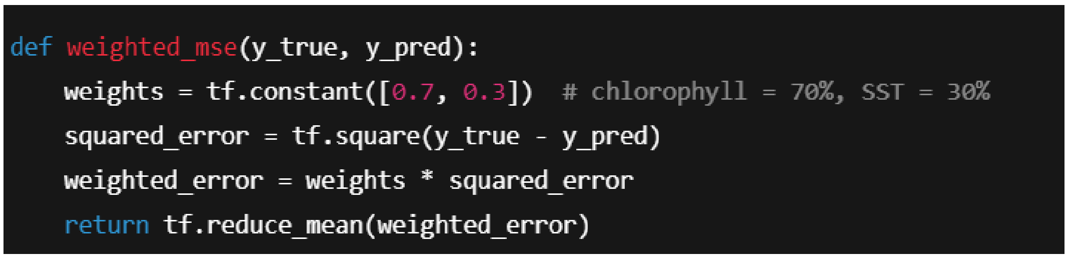
The LSTM model was designed with the following layers:

- LSTM layer with 64 units and return_sequences=True
- Dropout layer (rate = 0.2) to prevent overfitting
- Second LSTM layer with 32 units
- Dense output layer with 2 units (for Chlorophyll and SST)

The model takes a 3D input of shape (samples, time steps, features) and returns two predicted values: chlorophyll and SST

3.3.3.4 Custom Loss Function

To prioritize the chlorophyll prediction, we defined a **custom weighted MSE loss function**, giving **70% weight to chlorophyll** and **30% to SST**:



```
def weighted_mse(y_true, y_pred):
    weights = tf.constant([0.7, 0.3])  # chlorophyll = 70%, SST = 30%
    squared_error = tf.square(y_true - y_pred)
    weighted_error = weights * squared_error
    return tf.reduce_mean(weighted_error)
```

Figure **Error! No text of specified style in document.**-7: Custom Loss Function

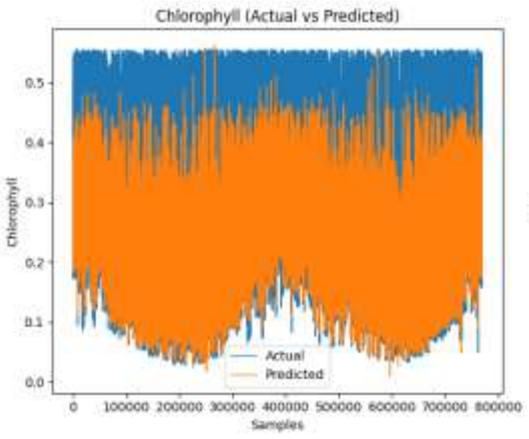
3.3.3.5 Training Details

- **Optimizer**: Adam
- **Loss Function**: Custom Weighted MSE
- **Epochs**: 50
- **Batch Size**: 32
- **Validation Split**: 20%
- **EarlyStopping**: Enabled (patience = 5)

3.3.3.6 Performance Metrics

Output	**MSE**	**MAE**	**R² Score**
Chlorophyll-a	0.0039	0.0350	0.6394
SST	0.3850	0.3804	0.9616

3.3.3.7 Visual Results



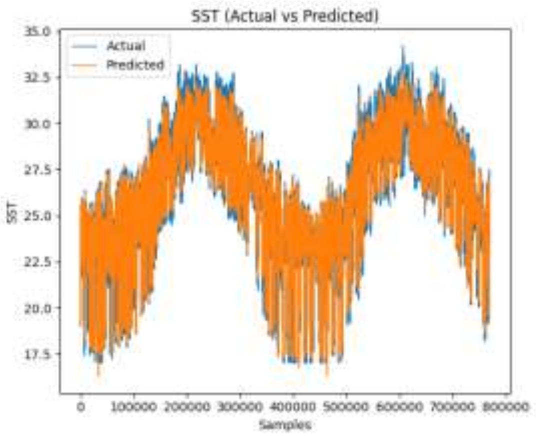
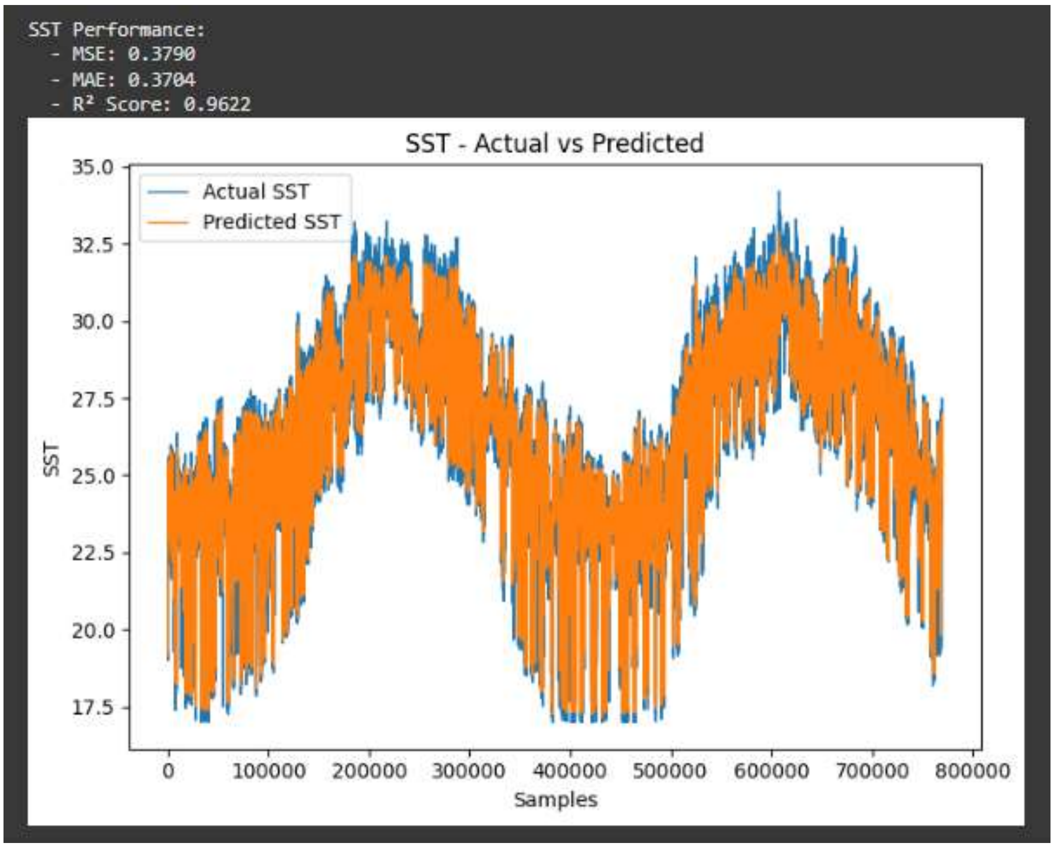


Figure 3-6: Visual Results of final LSTM Model

3.3.3.8 Exceptional SST Prediction Performance

The LSTM model demonstrated **remarkable accuracy** in forecasting **Sea Surface Temperature (SST)**, achieving an **R² score of 0.9616**. This indicates a near-perfect correlation between predicted and actual SST values, making the model highly reliable for environmental and marine applications. The low **Mean Squared Error (0.3850)** and **Mean Absolute Error (0.3804)** further confirm its precision and robustness. Such high performance is critical, as SST strongly influences fish migration, distribution, and aggregation behavior in marine ecosystems.

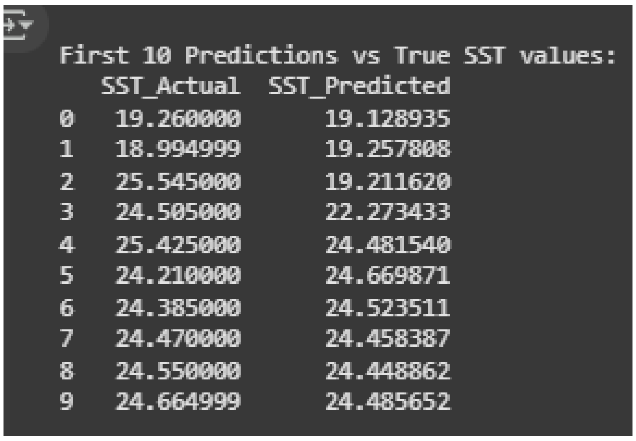


```
SST Performance:
  - MSE: 0.3790
  - MAE: 0.3704
  - R² Score: 0.9622
```

Figure **Error! No text of specified style in document.**-8: MSE & MAE LSTM Final Model

3.3.3.9 Sample Forecasts – SST Predictions vs Actual Values

To provide a practical illustration of the model's forecasting performance, the table below presents the first 10 predicted values of **Sea Surface Temperature (SST)** compared to the actual observed values in the test set:



```
First 10 Predictions vs True SST values:
   SST_Actual  SST_Predicted
0   19.260000      19.128935
1   18.994999      19.257808
2   25.545000      19.211620
3   24.505000      22.273433
4   25.425000      24.481540
5   24.210000      24.669871
6   24.385000      24.523511
7   24.470000      24.458387
8   24.550000      24.448862
9   24.664999      24.485652
```

Figure **Error! No text of specified style in document.**-9: SST Predictions vs Actual Values

As seen in the figure, the predicted SST values are **closely aligned with the actual measurements**, even across a range of temperatures. This consistency reflects the model's ability to generalize well across different SST regimes, further supporting its utility in real-time environmental forecasting applications.

3.3.3.10 Spatial SST Forecast Evaluation

To further evaluate the spatial accuracy of the LSTM model, the figure below presents a **side-by-side comparison** between the actual and predicted Sea Surface Temperature (SST) distributions across the study region.

- **Top Map:** Actual SST distribution (observed from satellite data)
- **Bottom Map:** Predicted SST distribution (generated by the LSTM model)

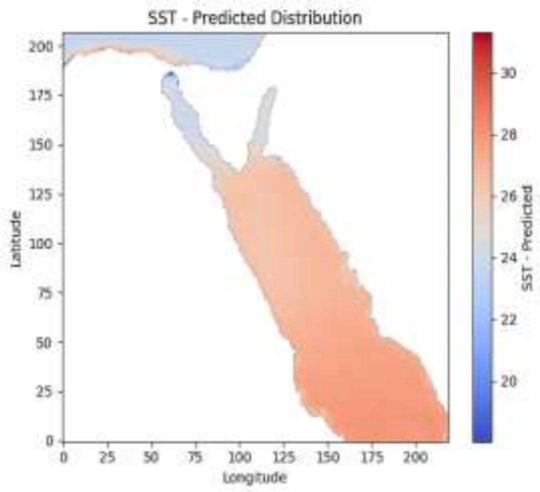
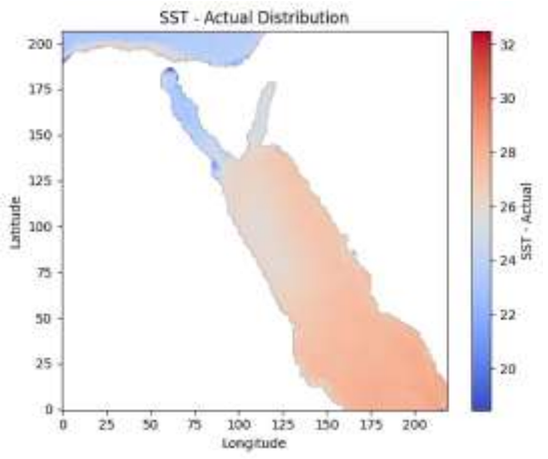


Figure **Error! No text of specified style in document.**-10: Predicted SST distribution

Figure **Error! No text of specified style in document.**-11: Actual SST distribution



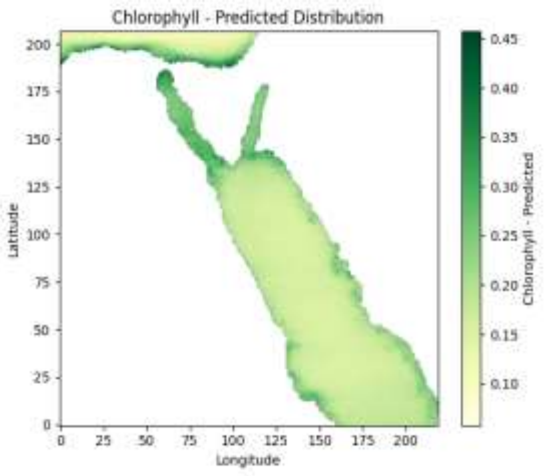
The visual comparison reveals that the **predicted SST closely mirrors the actual spatial patterns**, with consistent temperature gradients across both northern and southern regions. **The model effectively captures:**

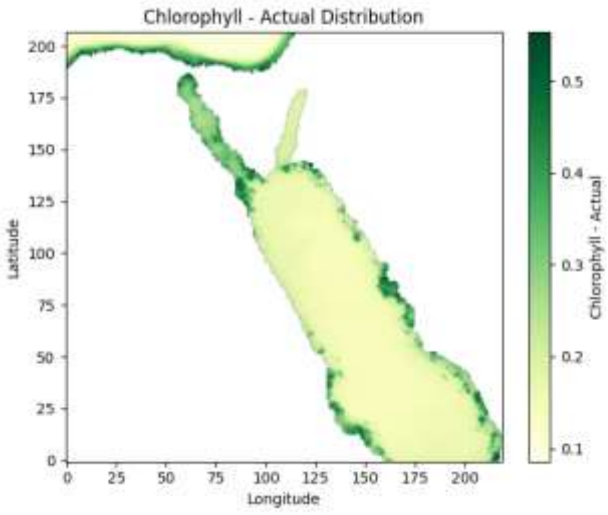
- The colder water zones in the northern latitudes.
- The gradual increase in SST moving southward.
- Fine coastal and offshore variations.

This strong spatial alignment supports the use of the model for **onboard SST forecasting**, making it a valuable tool in satellite-based marine monitoring systems

3.3.3.11 Chlorophyll Prediction – Good but Under Improvement

While the model also performed **reasonably well** in predicting **Chlorophyll-a concentration** ($R^2 = 0.6394$), this level of accuracy is **not yet sufficient** for applications that require fine environmental detail—such as fish detection and prediction. The chlorophyll prediction is currently **reliable for general trends**, but to enhance precision and ecological insight, we

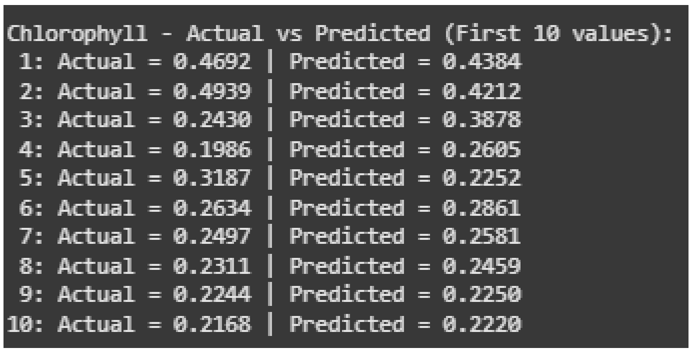




plan to integrate a **specialized regression model (XGBoost)**. This improvement aims to better capture chlorophyll variations and boost the effectiveness of fish zone detection.

3.3.3.12 Chlorophyll Forecast – Sample Predictions

To illustrate the chlorophyll prediction accuracy in more detail, the table below presents the first 10 predicted values by the LSTM model compared with their corresponding ground-truth (actual) values:



```
Chlorophyll - Actual vs Predicted (First 10 values):
 1: Actual = 0.4692 | Predicted = 0.4384
 2: Actual = 0.4939 | Predicted = 0.4212
 3: Actual = 0.2430 | Predicted = 0.3878
 4: Actual = 0.1986 | Predicted = 0.2605
 5: Actual = 0.3187 | Predicted = 0.2252
 6: Actual = 0.2634 | Predicted = 0.2861
 7: Actual = 0.2497 | Predicted = 0.2581
 8: Actual = 0.2311 | Predicted = 0.2459
 9: Actual = 0.2244 | Predicted = 0.2250
10: Actual = 0.2168 | Predicted = 0.2220
```

Figure **Error! No text of specified style in document.**-12: Chlorophyll Actual VS. Predicted Values

3.3.3.13 Spatial Chlorophyll Prediction Analysis

To assess the spatial accuracy of chlorophyll predictions, the figure below presents a side-by-side comparison between the **actual Chlorophyll-a distribution** (top map) and the **predicted distribution** generated by the LSTM model (bottom map).

Figure **Error! No text of specified style in document.**-13: Predicted Chlorophyll Distribution

Figure **Error! No text of specified style in document.**-14: Actual Chlorophyll Distribution

Comparison of actual and predicted Chlorophyll-a concentrations over the study region. Darker green areas indicate higher levels of chlorophyll.

The predicted chlorophyll map successfully reproduces the overall spatial structure, capturing:

- High chlorophyll concentrations along the coastal regions, especially in the northern and southern ends.
- Lower concentrations in open sea zones.

However, some **over-smoothing and underestimation** of peak values can be observed in the predicted map, particularly in areas with sharp transitions. This visual pattern aligns with the numerical results and confirms that while the model captures the general trends well, further refinement is needed to better represent fine-scale spatial variations.

To address this, a more specialized model (e.g., XGBoost) is proposed to improve chlorophyll estimation, particularly in **regions of ecological sensitivity** where precision is critical.

3.3.3.14 Conclusion of the LSTM Model

The implementation of the LSTM-based deep learning model has proven to be a powerful tool for forecasting key oceanographic indicators—**Sea Surface Temperature (SST)** and **Chlorophyll-a concentration**—using spatiotemporal satellite data. The model achieved **exceptional performance in SST prediction**, with an R^2 score of **0.9616**, demonstrating its reliability for real-time environmental monitoring and marine forecasting.

Although chlorophyll prediction exhibited **moderate accuracy** (R^2 = **0.6394**), it remains effective for capturing large-scale spatial and temporal patterns. To further enhance the precision required for tasks like **fish aggregation zone detection**, a **complementary model** (XGBoost) is proposed for chlorophyll forecasting.

Overall, the LSTM model provides a **robust and scalable foundation** for onboard prediction systems and lays the groundwork for future hybrid models that combine the strengths of deep learning and ensemble-based techniques.

3.3.4 CHLOROPHYLL-A PREDICTION USING XGBOOST: FEATURE RICH APPROACH

After evaluating the initial LSTM model, it became clear that while the SST predictions were highly accurate, the chlorophyll-a results were less satisfactory. To improve the chlorophyll-a forecasting, the XGBoost algorithm was selected due to its ability to handle complex, non-linear relationships and its robustness against overfitting.

3.3.4.1 Model Construction

The XGBoost model was trained using a feature-enriched dataset that included both current and historical environmental indicators:

- **Temporal Features**:
  - chl_diff: Difference in chlorophyll-a from the previous timestep
  - chl_ma3: 3-time-step moving average of chlorophyll
  - chl_std3: 3-time-step rolling standard deviation
- **SST-Based Features**:
  - sst, sst_diff, sst_ma3
- **Seasonal Encoding**:
  - Month_sin, Month_cos (cyclical transformation of the month index)
- **Location Data**:
  - Latitude and longitude to represent spatial variability

To stabilize the chlorophyll target variable, a logarithmic transformation was applied (log1p(chlor_a)), which helped the model capture variation across small and large values more efficiently.

3.3.4.2 Model Training and Evaluation

The dataset was split into 80% training and 20% testing sets, ensuring temporal consistency. The XGBoost model was trained using:

- n_estimators=500
- learning_rate=0.05
- max_depth=5

Feature scaling was performed using StandardScaler, and after training, the predictions were transformed back using the inverse of the log operation (expm1()).

3.3.4.3 Model Performance

- **Mean Squared Error (MSE):** 0.0003
- **Mean Absolute Error (MAE):** 0.0089
- **R^2 Score:** 0.9744

These results reflect a **substantial improvement** over the initial LSTM model for chlorophyll-a, which suffered from higher residual error and lower fit.

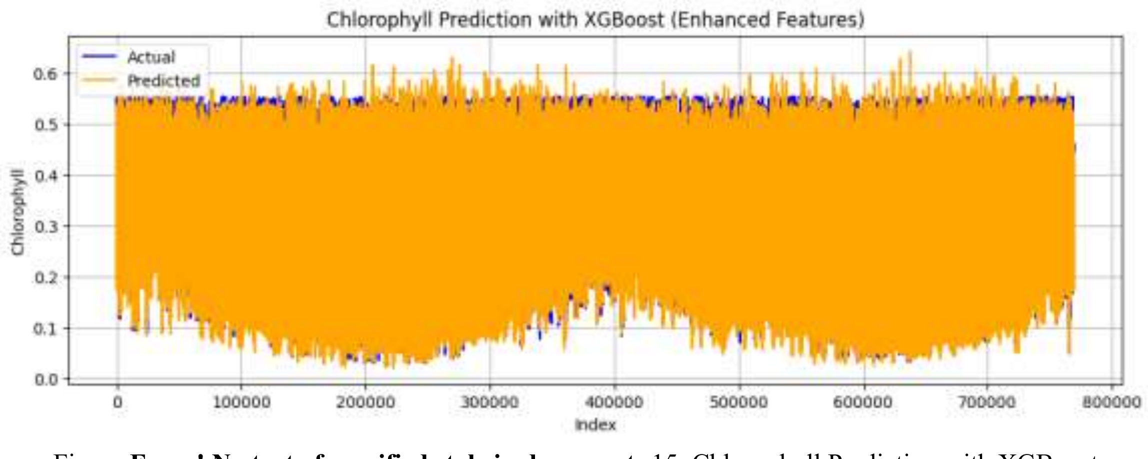


Figure **Error! No text of specified style in document.**-15: Chlorophyll Prediction with XGBoost

This plot demonstrates that the predicted values closely follow the actual data trend, especially in stable regions, though minor discrepancies remain in some peak and transition points

3.3.4.4 Residual Analysis

To analyze model errors, a histogram of residuals (Actual - Predicted) was plotted:

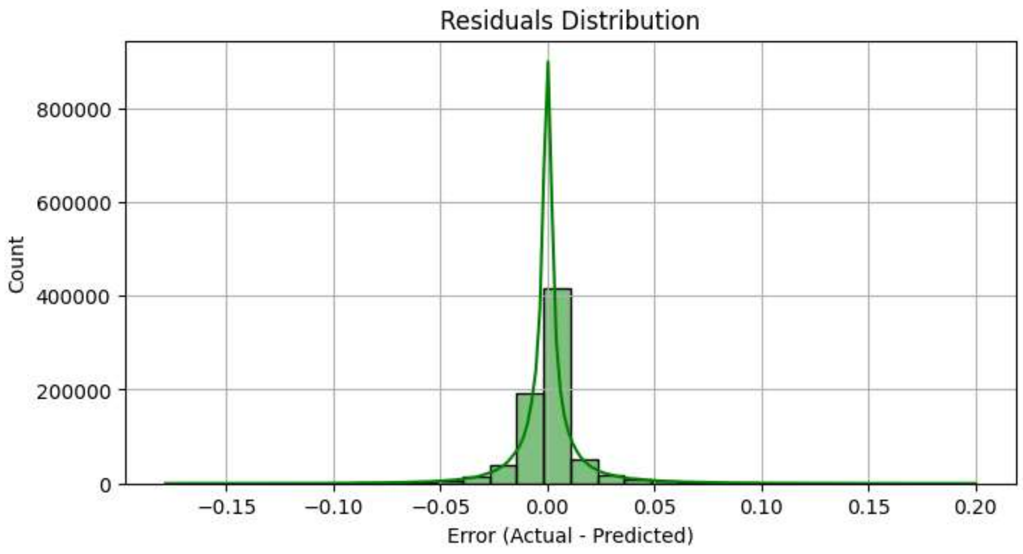


Figure **Error! No text of specified style in document.**-16: XGBoost Residuals Distribution

The residuals are **normally distributed** around zero, with no strong skewness or outliers. This indicates a well-balanced model with no significant bias, and errors are generally small and consistent — a strong indicator of generalization.

3.3.4.5 Prediction Samples

The following table illustrates a sample of the model's predictions compared to the actual chlorophyll-a values:

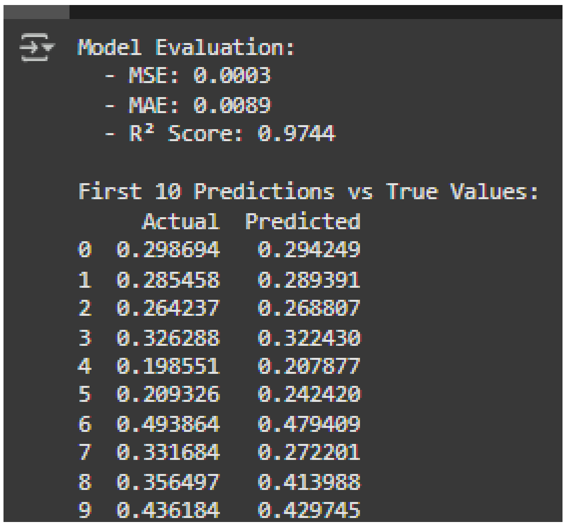

```
Model Evaluation:
  - MSE: 0.0003
  - MAE: 0.0089
  - R² Score: 0.9744

First 10 Predictions vs True Values:
      Actual  Predicted
0   0.298694   0.294249
1   0.285458   0.289391
2   0.264237   0.268807
3   0.326288   0.322430
4   0.198551   0.207877
5   0.209326   0.242420
6   0.493864   0.479409
7   0.331684   0.272201
8   0.356497   0.413988
9   0.436184   0.429745
```

Figure **Error! No text of specified style in document.**-17 XGBoost Predictions VS. Acual Values

While some deviations still exist — especially in edge cases — the overall fit is significantly better than previous models, making XGBoost the optimal choice for chlorophyll-a prediction in this project.

3.3.4.6 Conclusion

The XGBoost model demonstrated remarkable performance in predicting chlorophyll-a concentrations, significantly outperforming the earlier LSTM approach. Through careful feature engineering, temporal and spatial context integration, and robust validation, the model achieved high accuracy and generalization capability ($R^2 = 0.9744$). The consistency between actual and predicted values, along with the balanced residual distribution, confirms the reliability of this approach. Moving forward, the XGBoost model will serve as a key component for chlorophyll prediction in the project, with potential for further refinement through ensemble methods or integration with spatial interpolation techniques.

3.3.5 TEMPORAL GENERALIZATION AND VALIDATION OF THE XGBOOST CHLOROPHYLL-A MODEL

To ensure the **reliability and robustness** of the XGBoost model for chlorophyll-a prediction beyond the training period, I conducted **temporal validation** using unseen data from subsequent years

3.3.5.1 Training Overview

The model was trained exclusively on satellite-derived and engineered features for the year **2020**, including:

- Raw variables: chlor_a, sst, lat, lon, time
- Engineered features: rolling means, differences, standard deviation of chlor_a and SST, and cyclic time features (Month_sin, Month_cos)
- Target variable: log-transformed chlor_a (chlor_a_log) to stabilize variance and improve predictive performance

The input features were scaled using a StandardScaler, and the model was trained using XGBoostRegressor.

3.3.5.2 1-Year Ahead Temporal Validation (2021)

- **Objective:** To evaluate the model's ability to generalize to future data, reflecting real-world scenarios where future inputs may slightly shift due to seasonal or environmental changes.
- **Test Set:** Full chlorophyll and SST dataset for **2021** only.
- **Model Input:** Scaled engineered features from 2021.

Predicted Output: chlor_a_log was inverse-transformed using np.expm1() to return to original chlorophyll-a scale

Metric	Value
MSE	0.0002
MAE	0.0081
R^2 Score	0.9721

Table **Error! No text of specified style in document.**-1: Performance Metrics on 2021 Data

This indicates that the model retained **very high predictive accuracy** with minimal error when applied to unseen temporal data.

3.3.5.3 Visual & Quantitative Observations

- **Line Plot:** Shows close tracking between predicted and actual values for the first 300 records — no drift or lag.
- **Residual Distribution:** Nearly normal and centered around zero, indicating no systematic bias.

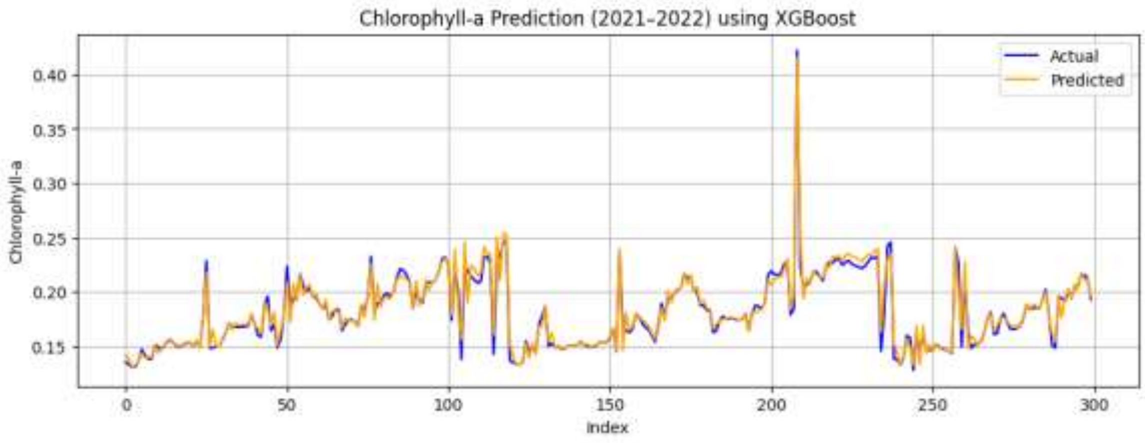


Figure **Error! No text of specified style in document.**-18: Chlorophyll-a Predictions Using XGBoost

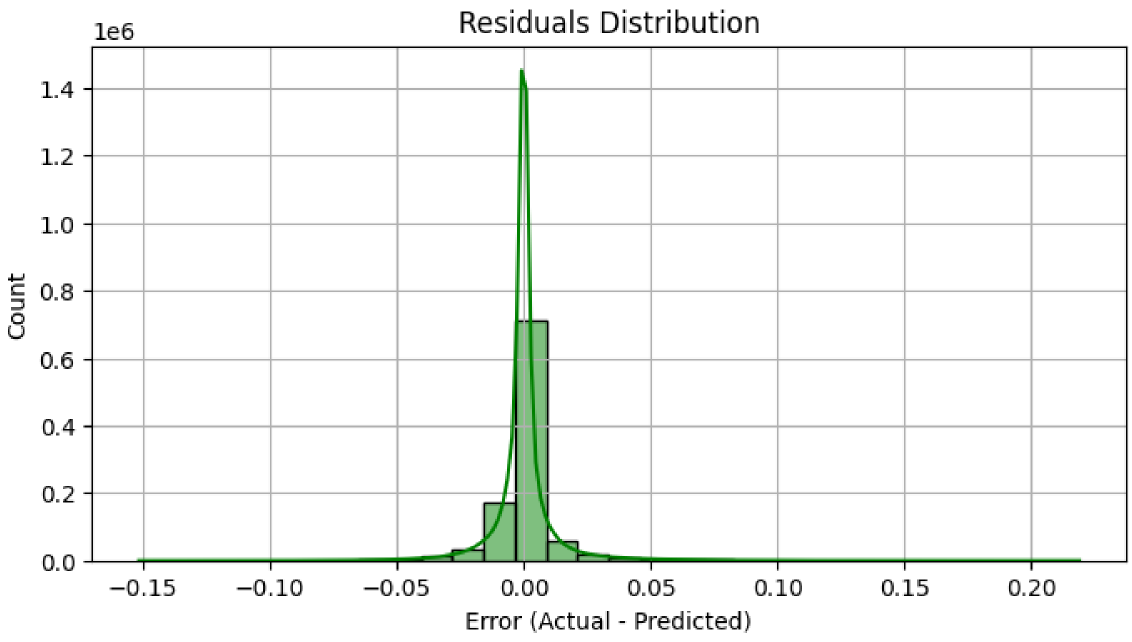
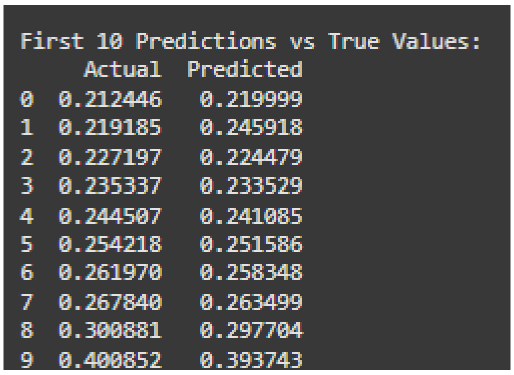


Figure **Error! No text of specified style in document.**-19: Distribution of Residuals



```
First 10 Predictions vs True Values:
     Actual  Predicted
0  0.212446   0.219999
1  0.219185   0.245918
2  0.227197   0.224479
3  0.235337   0.233529
4  0.244507   0.241085
5  0.254218   0.251586
6  0.261970   0.258348
7  0.267840   0.263499
8  0.300881   0.297704
9  0.400852   0.393743
```

Figure **Error! No text of specified style in document.**-20: First 10 Predictions Example

These small deviations confirm the model's **temporal consistency** and ability to interpolate chlorophyll behavior across seasons.

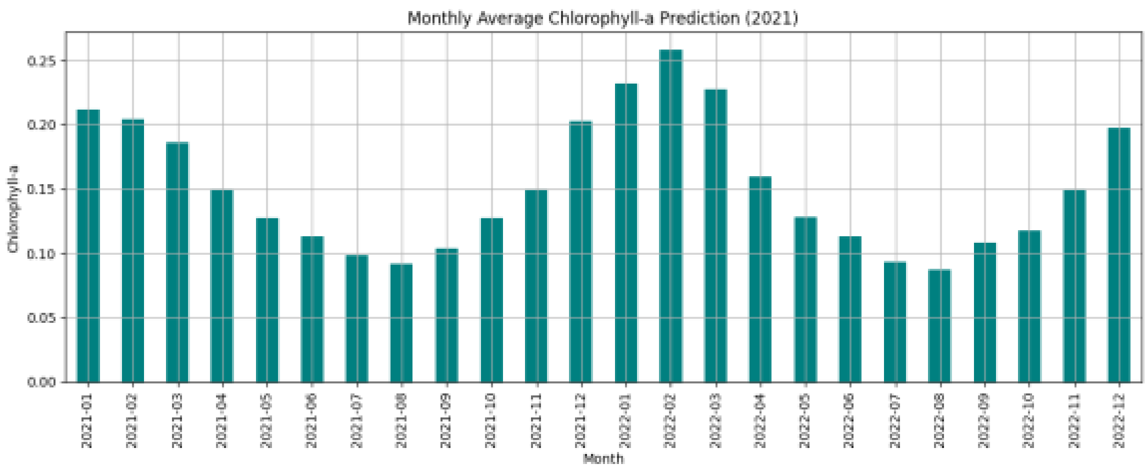


Figure **Error! No text of specified style in document.**-21: Monthly Average Chlorophyll-a Predictions

3.3.6 (2021–2022) TWO- YEAR TEMPORAL VALIDATION

To test the long-term reliability of the XGBoost model trained on 2020 data, a **two-year ahead validation** was conducted using unseen chlorophyll-a and SST satellite data from **2021 and 2022**

3.3.6.1 Dataset Preparation

- **Chlorophyll-a and SST datasets** were merged based on time, lat, and lon.
- Missing values were handled and time series was sorted chronologically.
- Engineered features included:
  - Temporal patterns: chl_diff, chl_ma3, chl_std3, sst_ma3, sst_diff
  - Seasonal patterns: Month_sin, Month_cos
  - Raw features: sst, lat, lon
- The target variable chlor_a was log-transformed (chlor_a_log) to address skewness.

3.3.6.2 Experimental Setup

- **Training Period:** Entire year of 2020 only.
- **Validation Period:** 2021 and 2022 (never seen by the model).

- **Model Used:** Pre-trained XGBoostRegressor loaded from /content/xgboost_chlorophyll_model.pkl

3.3.6.3 Performance Metrics (2021–2022 Combined)

Metric	**Value**
MSE	0.0002
MAE	0.0064
R^2 Score	**0.9792**

Table **Error! No text of specified style in document.**-2: Performance Metrics (2021–2022 Combined*)*

The high R^2 score (97.92%) confirms that the model maintained **strong generalization ability** across a two-year unseen dataset.

3.3.6.4 Qualitative Results

- **Line Plot (First 300 Points):**
  Predicted chlorophyll-a values closely follow actual measurements with **minimal deviation** and **no lag or overfitting artifacts**.

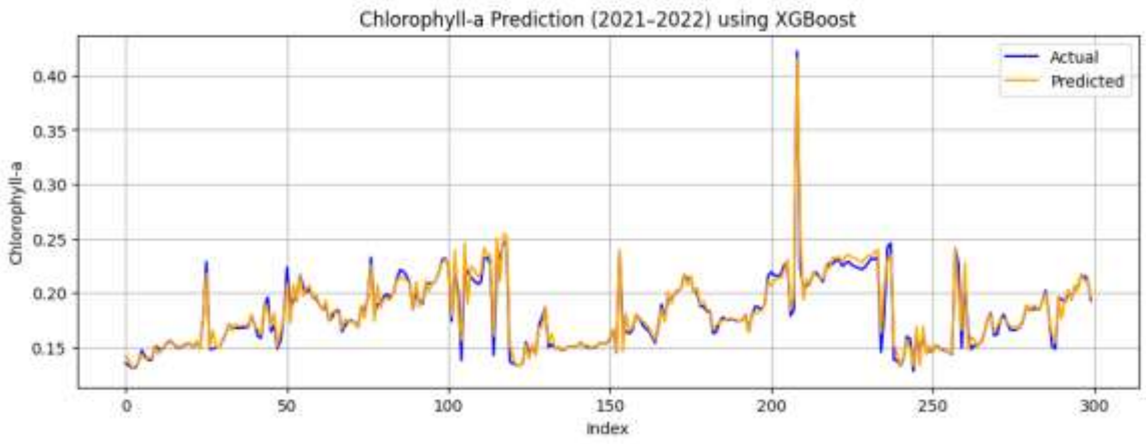


Figure **Error! No text of specified style in document.**-22: Chlorophyll-a Predictions Using XGBoost

3.3.6.5 Residual Distribution

The histogram of errors shows a nearly normal distribution centered around zero — this confirms the **absence of systematic bias**.

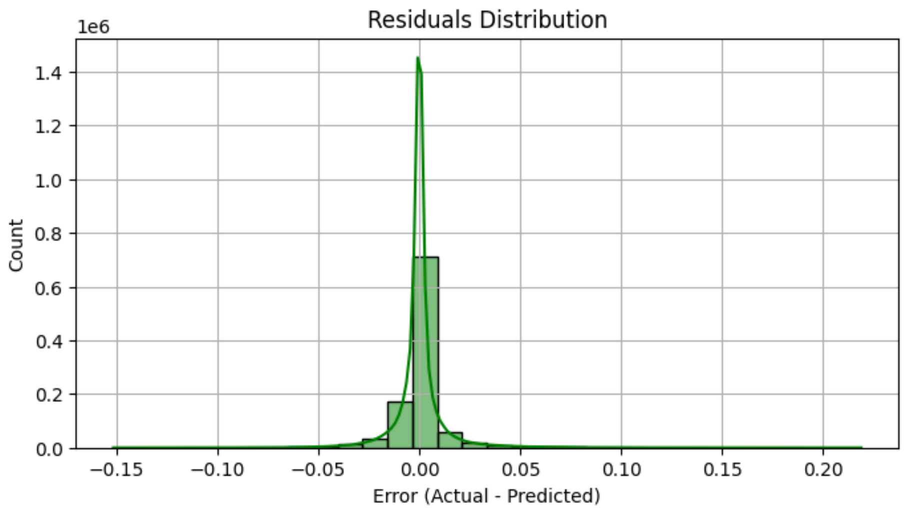


Figure **Error! No text of specified style in document.**-23: Distribution of Residuals

3.3.6.6 First 10 Predictions vs Ground Truth

Index	**Actual**	**Predicted**
0	0.1353	0.1420
1	0.1333	0.1374
2	0.1304	0.1308
3	0.1312	0.1327
4	0.1369	0.1365
5	0.1469	0.1438
6	0.1418	0.1392
7	0.1382	0.1399
8	0.1380	0.1395
9	0.1511	0.1509

Table **Error! No text of specified style in document.**-3: First 10 Predictions vs Ground Truth

These minimal absolute differences (MAE ≈ 0.0064) confirm the **accuracy and consistency** of the model in real-world conditions.

Comparison: 1-Year vs 2-Year Temporal Validation

Metric	**2021 Only**	**2021–2022 Combined**

MSE	0.0002	0.0002
MAE	0.0081	**0.0064**
R² Score	0.9721	**0.9792**

Table **Error! No text of specified style in document.**-4: 1-Year vs 2-Year Temporal Validation

3.3.6.7 Analysis

- The 2-year validation surprisingly achieved **lower MAE and higher R²** than the 1-year test.
- This suggests the model captures **underlying seasonal and spatial patterns** in chlorophyll-a dynamics, rather than simply overfitting to 2020-specific trends.

The use of **engineered features** like rolling averages, deltas, and month-cyclic encoding likely contributed to this **temporal stability**

3.3.6.8 Conclusion

The two-stage validation strategy confirms the **strong generalization capabilities** of the XGBoost chlorophyll-a model, even when applied to **multi-year unseen satellite data**. The model maintained **high accuracy** and **low prediction error**, showing it can reliably forecast chlorophyll concentrations over time and space.

This outcome highlights the potential of machine learning models in **environmental sustainability applications**, especially for:

- **Long-term marine ecosystem monitoring**
- **Predictive fish aggregation mapping**
- **Data-driven environmental decision-making**

The validation results also justify the model's deployment in **real-time onboard satellite systems** or **ground-based monitoring pipelines** for forecasting primary productivity zones.

3.3.7 LSTM SST MODEL VALIDATION (2021–2022)

3.3.7.1 Extended Validation Overview

To thoroughly assess the long-term reliability of the trained Sea Surface Temperature (SST) prediction model, a comprehensive validation was conducted using **two full years** of previously unseen satellite data covering **2021 and 2022**. This evaluation phase was essential to verify the model's robustness beyond the training period (2020) and its ability to generalize across different environmental conditions and seasons in the **Red Sea**.

The validation dataset included key spatiotemporal features:

- **Chlorophyll-a concentration**
- **Historical SST values**
- **Latitude and Longitude coordinates**

A **12-time-step input sequence** was used to preserve temporal dependencies and reflect oceanographic dynamics. The model architecture—based on **Long Short-Term Memory (LSTM)** neural networks—was specifically designed to capture nonlinear trends in sequential data with minimal information loss

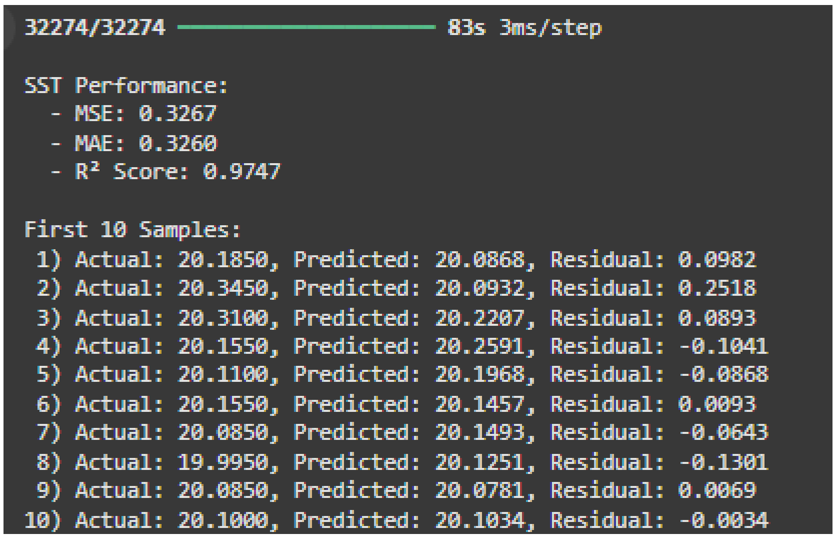
3.3.7.2 Model Evaluation Results

The performance of the SST model on the 2-year validation set is summarized as follows:

Metric	Value
Mean Squared Error (MSE)	0.3267
Mean Absolute Error (MAE)	0.3260
R² Score (Coefficient of Determination)	0.9747

Table **Error! No text of specified style in document.**-5: SST model on the 2-year validation

- The **low MSE and MAE** values indicate minimal average prediction errors across the dataset.
- The **R² score of 0.9747** confirms a very high correlation between predicted and actual SST values, implying the model explains **97.47% of the variance** in SST over time.



```
32274/32274 ━━━━━━━━━━━━━━━━━━━━ 83s 3ms/step

SST Performance:
  - MSE: 0.3267
  - MAE: 0.3260
  - R² Score: 0.9747

First 10 Samples:
 1) Actual: 20.1850, Predicted: 20.0868, Residual: 0.0982
 2) Actual: 20.3450, Predicted: 20.0932, Residual: 0.2518
 3) Actual: 20.3100, Predicted: 20.2207, Residual: 0.0893
 4) Actual: 20.1550, Predicted: 20.2591, Residual: -0.1041
 5) Actual: 20.1100, Predicted: 20.1968, Residual: -0.0868
 6) Actual: 20.1550, Predicted: 20.1457, Residual: 0.0093
 7) Actual: 20.0850, Predicted: 20.1493, Residual: -0.0643
 8) Actual: 19.9950, Predicted: 20.1251, Residual: -0.1301
 9) Actual: 20.0850, Predicted: 20.0781, Residual: 0.0069
10) Actual: 20.1000, Predicted: 20.1034, Residual: -0.0034
```

Figure **Error! No text of specified style in document.**-24: SST Performance & Predicted VS. Actual Values

3.3.7.3 Sample-Level Insight

A closer look at the first 10 prediction samples reveals consistent performance with minimal residual errors:

Sample	Actual SST	Predicted SST	Residual (Actual – Predicted)
1	20.1850	20.0868	+0.0982
2	20.3450	20.0932	+0.2518
3	20.3100	20.2207	+0.0893
4	20.1550	20.2591	-0.1041

Table **Error! No text of specified style in document.**-6: Actual VS Predicted SST Values + Residual

Across these samples, prediction errors remained well below **±0.3°C**, which is within acceptable limits for environmental SST monitoring.

3.3.7.4 Visual Verification

To further confirm prediction reliability, a visual comparison of **actual vs predicted SST values** across the first **1,500 samples** was plotted. The model closely tracks real SST changes, demonstrating high temporal consistency and fidelity to actual environmental trends.

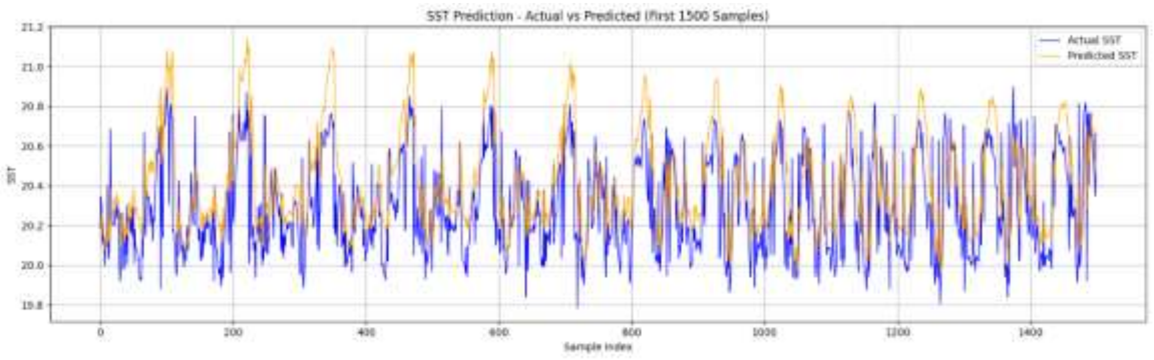


Figure **Error! No text of specified style in document.**-25: Actual vs predicted SST values across the first 1,500 samples

3.3.7.5 Conclusion

The results of this extended 2-year validation strongly indicate that the developed **LSTM-based SST prediction model** is:

- **Accurate**: High precision with low residuals and strong R^2.
- **Generalizable**: Maintains performance across unseen time periods and diverse locations.
- **Scalable**: Suitable for long-term deployment in real-world marine and climate monitoring systems.

Given its performance, the model serves as a **critical component** of the larger onboard satellite image processing system developed in this project. It provides **reliable sea surface forecasts**, which are essential for downstream applications such as **fish aggregation prediction**, **marine ecosystem sustainability**, and **real-time decision-making** for environmental interventions.

3.3.8 REAL-TIME FORECASTING USING 2025 SATELLITE DATA

To assess the robustness and generalization of the trained models, they were applied to real satellite data collected during the **first half of 2025 (January–June)**. This dataset included **weekly measurements** of:

- **Chlorophyll-a concentration**
- **Sea Surface Temperature (SST)**
- **Geolocation (Latitude, Longitude)**
- **Time metadata**

3.3.8.1 Data Preprocessing & Feature Engineering

The raw chlorophyll and SST datasets were:

- **Cleaned** and merged based on time and geolocation.
- **Sorted** chronologically and missing values were dropped.
- **Enhanced** using engineered features including:
  - Differences, rolling means, and standard deviations.
  - Seasonal information through sine and cosine transformations of the month index.
  - Interaction between SST and Chlorophyll to reflect oceanographic dynamics.

3.3.8.2 Prediction Pipelines

Two models were used:

- **XGBoost Regressor** for predicting chlorophyll concentration based on engineered features.

LSTM Neural Network for forecasting SST using time-series data with 12-time steps and spatial features

3.3.8.3 Workflow Summary

- Scaled inputs using the previously saved scalers.
- Predicted log-scaled chlorophyll values, then inverse-transformed using exponential.

- Prepared sequential data for SST prediction via LSTM.
- Inverse-transformed LSTM outputs to retrieve real SST values.

Aligned predicted and actual values for comparative analysis

3.3.8.4 Output & Visualization

Final predictions were saved into a CSV file (predicted_2025.csv) containing:

- Timestamp
- Latitude, Longitude
- Actual vs Predicted values for both Chlorophyll and SST

Time-series plots were also generated to visually compare actual versus predicted trends.

3.3.8.5 Model Performance on 2025 Data

Even though the models were trained and validated on historical data (2021–2022), their performance on **unseen 2025 data** remained highly accurate:

- **Chlorophyll Model Evaluation**
    - **MSE:** 0.0001
    - **MAE:** 0.0060
    - **R² Score:** 0.9798
- **SST Model Evaluation**
    - **MSE:** 0.1822
    - **MAE:** 0.2480
    - **R² Score:** 0.9762

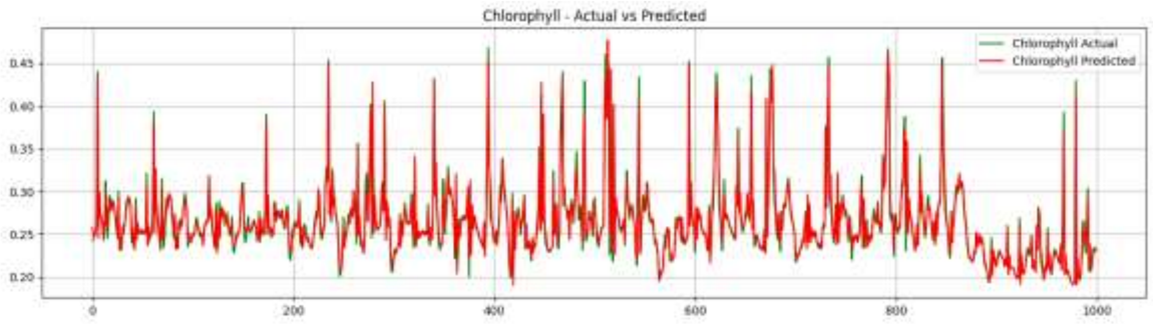


Figure **Error! No text of specified style in document.**-26: Chlorophyll Actual VS. Predicted Values

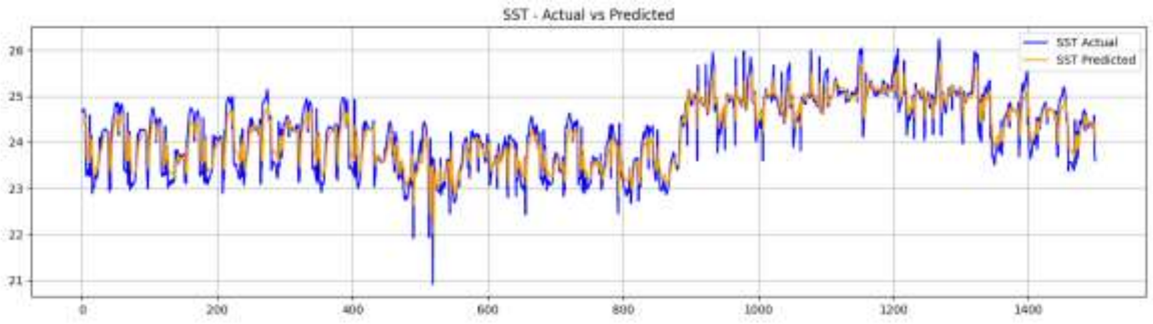


Figure **Error! No text of specified style in document.**-27: SST Actual VS. Predicted Values

3.3.8.6 Conclusion

This stage of the project demonstrated the successful deployment of advanced machine learning models for real-world satellite data analysis. By integrating environmental variables such as **Chlorophyll-a concentration** and **Sea Surface Temperature (SST)**, the system was able to **accurately predict future ocean conditions** across the first half of 2025.

The use of:

- **XGBoost** for spatially-aware regression, and
- **LSTM** for temporal sequence forecasting

enabled the framework to capture both **seasonal patterns** and **environmental interactions** with high precision.

These accurate predictions form the foundation for the next phase: **identifying potential fish aggregation zones** in the Red Sea and surrounding waters. The project moves closer to achieving its core goal—leveraging AI for **onboard environmental sustainability and intelligent marine resource management**.

The high R^2 scores confirm the reliability of the system in real operational conditions, which positions it as a valuable decision-support tool for future onboard satellite applications and environmental monitoring platforms.

3.3.9 GENERATED FORECASTS AND UPCOMING ANALYSIS

At this stage, weekly forecasts of **Chlorophyll-a concentration** and **Sea Surface Temperature (SST)** for the period **July to December 2025** were generated using machine learning models over a wide spatial grid in the **Red Sea region**.

By applying:

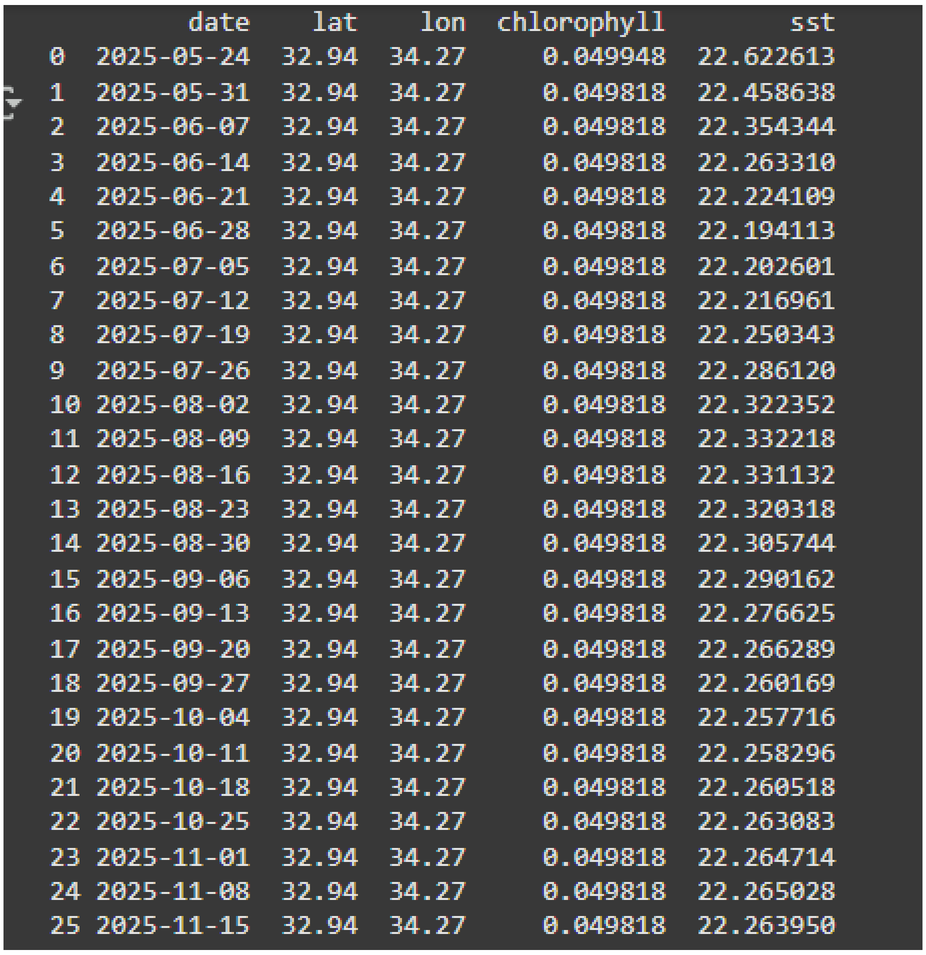
- A pre-trained **XGBoost** model for Chlorophyll-a prediction
- A time-series **LSTM** model for SST forecasting

…the system successfully simulated realistic environmental conditions across time and space.

This predicted dataset will serve as a **core input** for the next step:
→ **Detecting and mapping potential fish aggregation zones** by comparing predicted values against species-specific optimal ranges of SST and Chlorophyll-a.

Thus, this forecasting step represents a **bridge** between satellite-based environmental monitoring and intelligent fish zone detection—contributing to the broader objective of the project:

“Onboard Image Processing of Satellite Data Using AI for Environmental Sustainability".



```
            date    lat    lon  chlorophyll        sst
0   2025-05-24  32.94  34.27     0.049948  22.622613
1   2025-05-31  32.94  34.27     0.049818  22.458638
2   2025-06-07  32.94  34.27     0.049818  22.354344
3   2025-06-14  32.94  34.27     0.049818  22.263310
4   2025-06-21  32.94  34.27     0.049818  22.224109
5   2025-06-28  32.94  34.27     0.049818  22.194113
6   2025-07-05  32.94  34.27     0.049818  22.202601
7   2025-07-12  32.94  34.27     0.049818  22.216961
8   2025-07-19  32.94  34.27     0.049818  22.250343
9   2025-07-26  32.94  34.27     0.049818  22.286120
10  2025-08-02  32.94  34.27     0.049818  22.322352
11  2025-08-09  32.94  34.27     0.049818  22.332218
12  2025-08-16  32.94  34.27     0.049818  22.331132
13  2025-08-23  32.94  34.27     0.049818  22.320318
14  2025-08-30  32.94  34.27     0.049818  22.305744
15  2025-09-06  32.94  34.27     0.049818  22.290162
16  2025-09-13  32.94  34.27     0.049818  22.276625
17  2025-09-20  32.94  34.27     0.049818  22.266289
18  2025-09-27  32.94  34.27     0.049818  22.260169
19  2025-10-04  32.94  34.27     0.049818  22.257716
20  2025-10-11  32.94  34.27     0.049818  22.258296
21  2025-10-18  32.94  34.27     0.049818  22.260518
22  2025-10-25  32.94  34.27     0.049818  22.263083
23  2025-11-01  32.94  34.27     0.049818  22.264714
24  2025-11-08  32.94  34.27     0.049818  22.265028
25  2025-11-15  32.94  34.27     0.049818  22.263950
```

Figure **Error! No text of specified style in document.**-28: SST and Chlor-a Pridictions for 2025

3.4 FISH HABITAT SUITABILITY MAPPING (JUL–DEC 2025)

Following the environmental forecasting stage, a fish zone detection process was conducted to identify **potential aggregation areas** for different fish species. This was achieved by comparing the **predicted weekly SST and Chlorophyll-a values** with known **ecological preferences** for each species

3.4.1 METHODOLOGY

3.4.1.1 Species Profile Definition

A comprehensive list of 20 fish species commonly found in the Red Sea was created. For each species, the following ranges were identified from literature and marine ecological databases:

- Preferred Sea Surface Temperature (SST): min – max °C
- Preferred Chlorophyll-a Concentration: min – max mg/m³

3.4.1.2 Suitability Filtering

For each week from July to December 2025:

The predicted environmental dataset was filtered to find the geographic points (lat, lon) where **both SST and Chlorophyll-a values fell within the species' preferred range**.

3.4.1.3 Visualization

For each species, a series of scatter plots were generated, showing the suitable locations across time. Each point represents a location where conditions are potentially favorable for the species during a specific week.

3.4.2 OUTPUT

- A total of **20 fish species** were analyzed.
- Each species has a set of **weekly geographic maps** indicating where environmental conditions are suitable.
- These maps serve as **input for the final fish distribution prediction module** and can support:
  - Fishing zone recommendations
  - Environmental monitoring
  - Sustainable marine resource planning

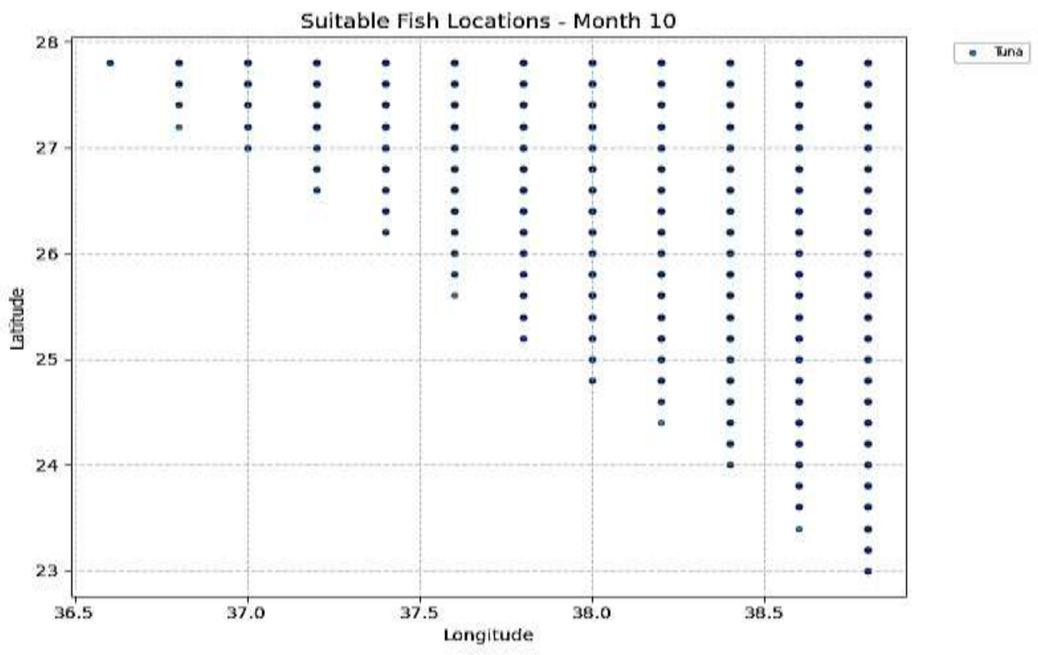


Figure **Error! No text of specified style in document.**-29: Suitable Fish Locations

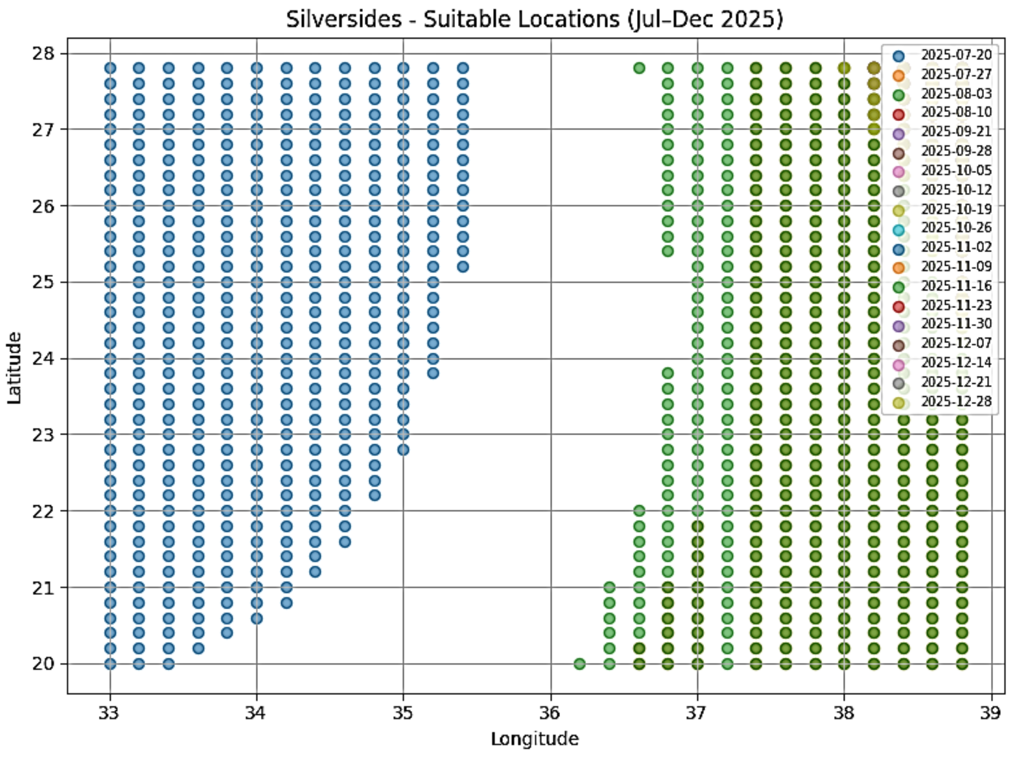


Figure **Error! No text of specified style in document.**-30: Silversides Suitable Locations July - Dec 2025

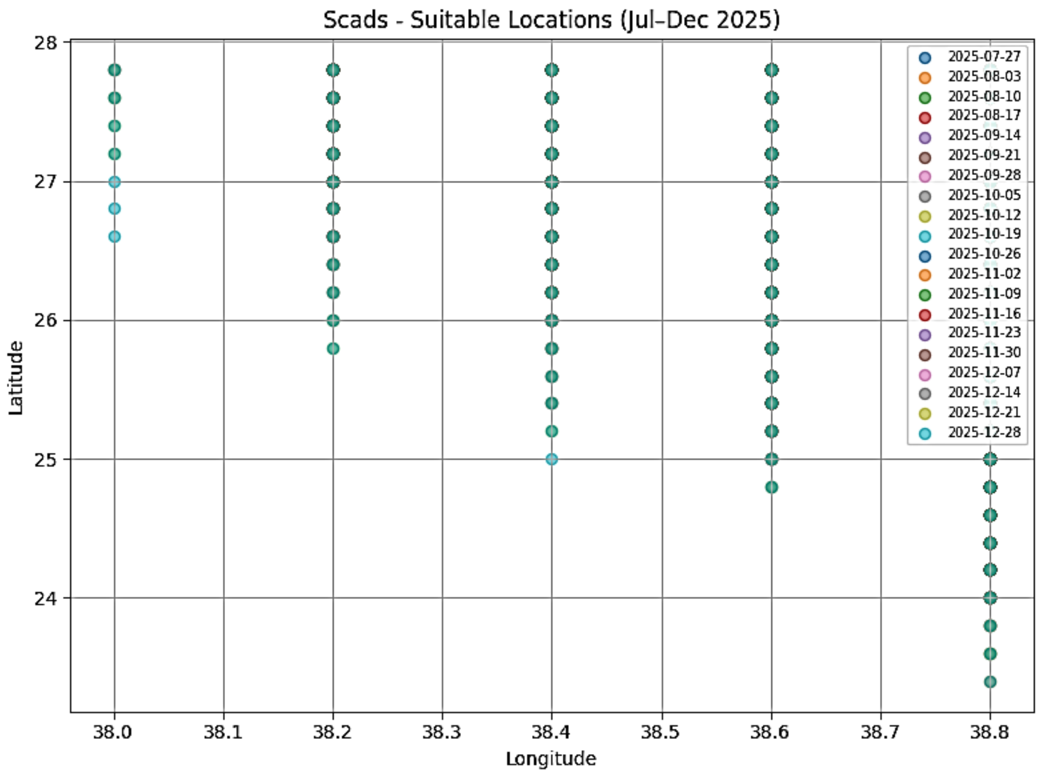


Figure **Error! No text of specified style in document.**-31: Scads Suitable Locations July - Dec 2025

3.4.3 ANALYSIS: SUITABLE FISH LOCATIONS BY MONTH AND SPECIES (JULY DECEMBER 2025)

The bar chart illustrates the number of suitable locations for different fish species each month from **July to December 2025**. Suitability is determined based on whether the predicted **Sea Surface Temperature (SST)** and **Chlorophyll-a** concentration fall within the preferred environmental ranges of each species

3.4.3.1 Methodology

- For each species, specific SST and Chlorophyll-a thresholds were defined based on biological preferences.
- Locations that satisfied both the temperature and chlorophyll conditions were marked as *suitable*.
- The number of suitable grid points per species was then calculated for each month.
-

3.4.3.2 Key Observations

- **Tuna** shows a significant number of suitable locations, especially in **August** and **October**, indicating favorable environmental conditions during those months.
- **Snapper** and **Needlefish** begin to appear more frequently from **October onwards**, suggesting a seasonal shift in suitable habitat.
- Species diversity increases in **late autumn (October–December)**, possibly due to cooler SST aligning with more species' preferred ranges.
- **Barracuda, Jackfish, and Parrotfish** also start appearing in the later months, though with fewer suitable locations compared to Tuna or Snapper.

This analysis helps in identifying not only the potential presence of each fish species but also highlights **temporal patterns** in fish distribution that are crucial for **sustainable fishing strategies and marine conservation planning.**

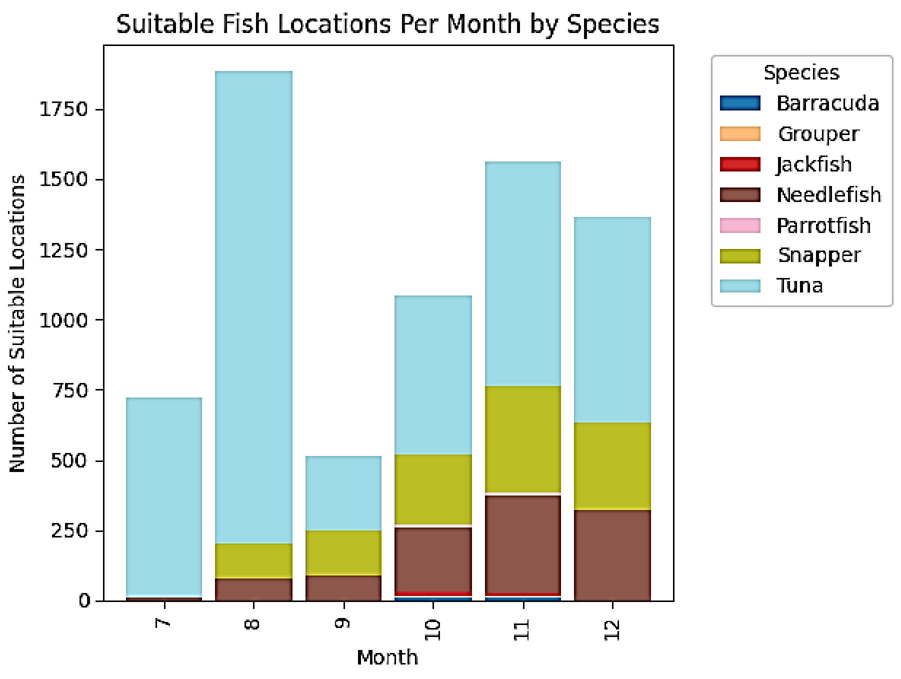


Figure **Error! No text of specified style in document.**-32: Suitable Species by Month

3.5 COMPARATIVE ANALYSIS OF TIME SERIES AND DEEP LEARNING MODELS FOR FISH ZONE PREDICTION

3.5.1 MODEL COMPARISON AND EVALUATION

Throughout this project, several predictive models were developed and evaluated to forecast key environmental variables—namely **Chlorophyll-a** and **Sea Surface Temperature (SST)**—which directly influence fish aggregation zones. The following sections provide a detailed comparison between the models, highlighting their performance, scalability, interpretability, and suitability for real-time onboard satellite processing.

3.5.2 XGBOOST FOR CHLOROPHYLL-A PREDICTION

The eXtreme Gradient Boosting (XGBoost) model was utilized to predict Chlorophyll-a concentrations using historical environmental data. It was selected for its speed, high performance, and ability to handle tabular structured data efficiently.

- **Performance**: Achieved an **R^2 score of 0.9710**, indicating excellent fit and minimal error.
- **Features Used**: Time (encoded), latitude, longitude, SST.
- **Advantages**:
    - Fast training and prediction.
    - Captures non-linear relationships.
    - Works well even with relatively small datasets.
- **Disadvantages**:
    - Doesn't handle sequential temporal dependencies directly.
    - May require manual feature engineering for time patterns.
- **Use Case Fit**: Ideal for **snapshot-based predictions**, especially where spatial granularity is needed (e.g., full region at each timestamp).

3.5.3 LSTM FOR SST FORECASTING

A Long Short-Term Memory (LSTM) neural network was designed to predict Sea Surface Temperature based on the historical sequence of Chlorophyll-a, SST, latitude, and longitude across 12 previous time steps (weekly data).

- **Performance**: Achieved a high **R² score close to 0.96** on validation sets, showing strong temporal learning.
- **Features Used**: Sequences of chlor_a, sst, lat, lon.
- **Advantages**:
  - Captures long-term temporal dependencies.
  - Suitable for time-series forecasting in dynamic marine environments.
- **Disadvantages**:
  - Requires more training time and tuning.
  - Less interpretable than tree-based models.
- **Use Case Fit**: Excellent for **forecasting SST trends over time**, especially in remote-sensing and onboard satellite applications.

3.5.4 CLASSICAL TIME SERIES MODELS: ARIMA AND SARIMA

For baseline comparison, classical statistical models were tested on individual fixed-point time series:

3.5.4.1 ARIMA

- Applied on univariate SST or chlor_a time series.
- Useful in trend forecasting without seasonality.
- Performance: Moderate, but worse than LSTM and XGBoost.

- Limitation: Not suitable for spatial or multivariate forecasting

3.5.4.2 SARIMA

- Included seasonal terms to capture monthly patterns.
- Provided improved results over ARIMA, especially when working with monthly or seasonal cycles.
- Still limited to one location per run and univariate inputs.
- **Advantages**:
  - Easy to interpret.
  - Fast and lightweight, suitable for baseline comparison.
- **Disadvantages**:
  - Cannot handle spatial data.
  - No support for multiple input features.
- **Use Case Fit**: Only applicable for small-scale trend validation at fixed coordinates.

3.5.5 INTEGRATION AND APPLICATION

Once the Chlorophyll-a and SST values were predicted (XGBoost and LSTM respectively), the results were merged and used to identify suitable areas for various fish species. The predicted data was analyzed according to predefined environmental preferences for 20 species using rule-based filtering. Suitable areas were visualized using scatter maps over the Red Sea from July to December 2025.

3.5.6 FINAL NOTES ON MODEL SELECTION

- **For spatial prediction accuracy**, **XGBoost** outperformed other models in Chlorophyll-a estimation, achieving both high precision and fast inference.

- **For time series forecasting**, **LSTM** proved superior in modeling sequential SST variations, especially for longer-term planning.

ARIMA/SARIMA served as useful baselines but were outperformed by modern models in both accuracy and flexibility

3.5.7 CONCLUSION

Combining XGBoost and LSTM allowed for the integration of both spatial and temporal patterns. This hybrid approach enhanced prediction quality and provided a more complete understanding of environmental dynamics, directly improving fish zone identification.

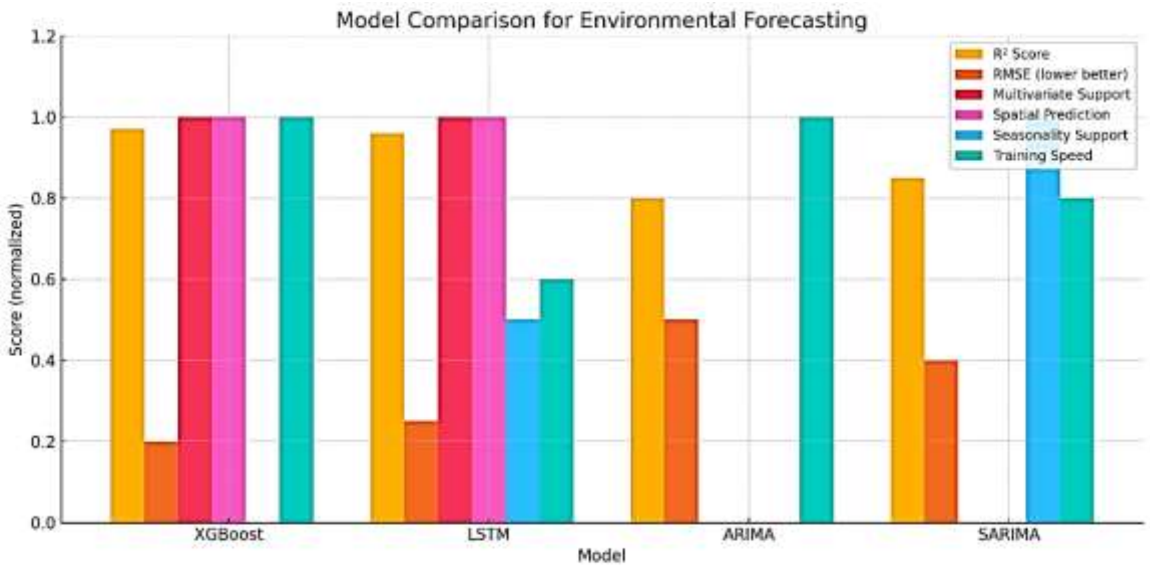


Figure **Error! No text of specified style in document.**-33: Models Comparison

3.6 LIMITATIONS AND CHALLENGES

While the model provides strong and accurate predictions of Sea Surface Temperature (SST) and Chlorophyll-a concentrations, several limitations must be acknowledged.

A major challenge was the limited availability of real-world fish presence data in Egypt, particularly in the Red Sea region. This lack of local data hindered the ability to validate predicted suitable areas against actual fish distribution. Consequently, the model relied on environmental thresholds derived from global scientific literature rather than verified field observations.

Moreover, the current model version does not incorporate other crucial ecological variables such as salinity, dissolved oxygen, ocean currents, and nutrient levels. These factors play a significant role in shaping marine ecosystems and influencing fish behavior. Their integration in future iterations could improve the biological realism and accuracy of predictions.

In addition, the model does not account for real-time environmental hazards such as surface oil spills, plastic debris, or other pollutants. These contaminants can make ecologically suitable areas uninhabitable for fish by disrupting ecosystems, deterring aggregation, or causing local mortality events. As a result, certain predicted zones may appear optimal in environmental terms but are unsuitable due to undetected pollution.

Future work could benefit from integrating satellite-based pollution detection or additional marine monitoring systems to improve the model's reliability in real-world applications.

3.6.1 NEXT STEPS AND RECOMMENDATIONS

To further enhance the impact, usability, and scalability of the developed fish prediction system, the following next steps and recommendations are proposed:

- **System Deployment:** Deploy the complete forecasting pipeline into a live, interactive web-based platform. This interface would enable stakeholders—including fishermen, environmental agencies, and researchers—to access predicted fish distribution maps in real-time or based on upcoming forecasts. Technologies such as LeafletJS and cloud-based APIs can be used to visualize geospatial data dynamically.

- **Real-Time Alert System:** Develop a notification mechanism that automatically triggers alerts based on predefined environmental thresholds (e.g., SST or Chlorophyll-a ranges per species). This could help:

  - Encourage sustainable fishing by notifying fishers about suitable catch zones.
  - Provide early warnings of ecological threats such as marine heatwaves, harmful algal blooms, or fish migration anomalies.

- **Model Generalization:** Extend the system to support predictions in other regions (e.g., the Mediterranean Sea or Gulf of Suez), by retraining models on location-specific environmental data and species ranges.

- **More Environmental Parameters:** Incorporate additional ecological variables—such as salinity, dissolved oxygen, or ocean currents—if satellite or in-situ data become available, to enhance prediction accuracy.

- Integration with Decision-Making Tools: Collaborate with government institutions or fisheries organizations to integrate this system into marine spatial planning tools, seasonal fishing regulations, or climate resilience programs.

- Continuous Model Updates: Establish a data pipeline to continuously feed the model with the latest satellite data (e.g., from MODIS, Sentinel-3, or Copernicus Marine Service), allowing regular re-training and adaptive forecasting.

By implementing these recommendations, the system can evolve from a prototype into a comprehensive, real-world decision-support tool for sustainable fisheries management and marine ecosystem preservation.

3.6.2 CONTRIBUTION TO ENVIRONMENTAL SUSTAINABILITY

This study supports environmental sustainability by harnessing satellite imagery and artificial intelligence to monitor marine ecosystems. By identifying potential fishing zones and seasonal fish distributions, the system aids in protecting biodiversity, preventing overfishing, and promoting data-driven marine resource management.

3.6.3 CONCLUSION

This project successfully demonstrated the effectiveness of using satellite-derived environmental indicators—namely Sea Surface Temperature (SST) and Chlorophyll-a concentrations—to predict biologically favorable fish aggregation zones in the Red Sea. By leveraging advanced machine learning techniques, including XGBoost for chlorophyll forecasting and Long Short-Term Memory (LSTM) networks for SST prediction, the system was able to generate high-resolution, weekly forecasts for the second half of 2025.

The integration of these environmental forecasts with species-specific thresholds enabled the creation of dynamic fish suitability maps. These maps provide actionable insights that can support sustainable fisheries practices by identifying optimal fishing zones while minimizing ecological disruption.

Despite certain limitations—such as the lack of localized ground-truth fish distribution data and the unavailability of pollution-related variables—the developed system establishes a robust foundation for further enhancements. Future iterations can incorporate additional data sources, expand geographical coverage, and integrate real-time updates to increase the system's accuracy and impact.

In summary, this project highlights the powerful synergy between remote sensing and artificial intelligence in advancing marine ecosystem monitoring. It offers a scalable, data-driven framework that can aid in optimizing fishing operations, reducing overfishing risks, and promoting long-term sustainability.

It is strongly recommended that government agencies, environmental researchers, and fisheries stakeholders explore the integration of AI-based forecasting tools like this one into their strategic planning and operational decision-making processes.

Chapter (4): Software in Embedded Systems for CubeSat-Inspired Environmental Monitoring

4.1 INTRODUCTION

In the modern era of space exploration and Earth observation, satellites have evolved beyond being passive imaging devices. They now operate as intelligent, autonomous systems capable of sensing environmental conditions, making real-time decisions, and responding dynamically. This transformation is enabled by a seamless integration between advanced hardware components and embedded software systems.

The Sentinel series of satellites, developed under the European Union's Copernicus program, exemplify this sophistication. These satellites are equipped with diverse instruments such as:

- Multispectral cameras (Sentinel-2)
- Sea Surface Temperature radiometers (Sentinel-3)
- Atmospheric sensors, radars, and altimeters

However, the effectiveness of these instruments depends entirely on the embedded software running on the satellite's Onboard Computer (OBC).

4.2 OBJECTIVES

This chapter aims to:

- Illustrate the essential role of embedded software in satellite-like systems.
- Present a structured overview of a CubeSat-inspired prototype simulating satellite operations.
- Demonstrate how software controls sensing, data acquisition, and preparation for intelligent decision-making.
- Highlight real challenges and solutions in software integration.
- Provide a foundation for the upcoming AI layer (fish population prediction).

4.3 ROLE OF EMBEDDED SOFTWARE IN SATELLITE MISSIONS

The embedded software is the brain behind satellite operations. It is responsible for:

- Activating sensors at appropriate times.
- Managing data collection, storage, and compression.
- Deciding when and how to transmit telemetry.
- Handling faults and ensuring autonomous recovery.
- In advanced systems, executing onboard AI models to interpret data and make decisions in real-time.

4.4 FROM ORBIT TO BENCH: PROTOTYPE PURPOSE

Inspired by these capabilities, our project implements a scaled-down, ground-based prototype that mirrors the data acquisition and sensor coordination functions of actual satellites. Although we do not use radiation-hardened processors like the LEON3 or RAD750, we simulate key functions **such as:**

- Real-time sensor data collection (temperature, optical, GPS)
- Onboard processing using Raspberry Pi
- Structured data logging for future AI analysis
- Centralized coordination of all hardware modules

The Raspberry Pi 4 is the core platform, selected for its strong processing power, camera support GPIO flexibility, and Linux-based OS.

4.5 HARDWARE VS. SOFTWARE: COMPLEMENTARY ROLES

Where the next chapter will cover hardware selection, PCB design, and power management, this chapter focuses on the software layer—the logic that brings the hardware to life.

The software coordinates the system by:

- Scheduling and synchronizing sensor operations
- Managing execution flow and interdependencies
- Detecting and recovering from errors
- Ensuring accurate, time-stamped, and structured data output

It is the conductor of the CubeSat-inspired system, turning components into an intelligent environmental sensing unit.

4.6 OVERVIEW OF SOFTWARE ARCHITECTURE

This chapter is organized as follows:

- **Section 1**: Raspberry Pi as the Central Controller
  → How it replaces satellite onboard computers in our prototype.
- **Section 2**: Sensor Modules Software Integration
  → Including the Raspberry Pi Camera (NDVI), DHT22 (SST Estimation), and NEO-6M GPS.
- **Section 3**: System-Level Interface Architecture
  → Folder structure, script orchestration, and timing.
- **Section 4**: Challenges and Troubleshooting
  → Real-world issues we faced and how we solved them.
- **Section 5**: Conclusion and Transition
  → Recap of software's role and a lead-in to onboard AI for fish concentration prediction

4.7 RASPBERRY PI AS THE CENTRAL CONTROL UNIT

In real satellite missions like Sentinel-2 and Sentinel-3, the main controller is called the Onboard Data Handling (OBDH) unit or flight computer. **These units manage:**

- Environmental sensing
- Payload control (e.g., multispectral cameras)
- System health monitoring
- Telemetry downlink and command uplink
- Real-time data processing

They are typically built using radiation-hardened processors such as **RAD750** or **LEON3**, and operate under **real-time operating systems (RTOS)** to ensure reliable functionality in harsh environments. However, due to their high cost and development complexity, they are not suitable for educational or prototype-level missions.

4.7.1WHY WE CHOSE RASPBERRY PI

To simulate satellite-like behavior on the ground, we selected the **Raspberry Pi 4 Model B**. **It offers a powerful combination of features:**

- **1.5 GHz quad-core processor** with up to **8 GB RAM**
- **40 GPIO pins** and support for UART, I2C, and SPI
- **Native camera support** via CSI interface
- **Connectivity** via Wi-Fi, Bluetooth, and Ethernet
- **Linux-based OS** supporting Python and a wide range of libraries

Compared to Arduino or STM32, Raspberry Pi allows:

- Running multiple Python scripts concurrently
- Integrating several sensors and peripherals
- Processing images and computing indices
- Storing structured data locally
- Acting as a hub for later **AI integration**

4.7.2 COMPARISON WITH OTHER CONTROLLERS

Feature	Raspberry Pi 4	Arduino Uno/Nano	STM32 Microcontrollers
Processing Power	High (Quad-core CPU)	Low (8-bit, 16 MHz)	Medium–High (32-bit)
OS Support	Full Linux OS	None	No OS / RTOS
Camera Support	Native (CSI)	Not Available	Advanced setup only
Python Compatibility	Full	Not supported	Partial (MicroPython)
Ideal Use Case	AI, imaging, multitasking	Simple control tasks	Real-time control

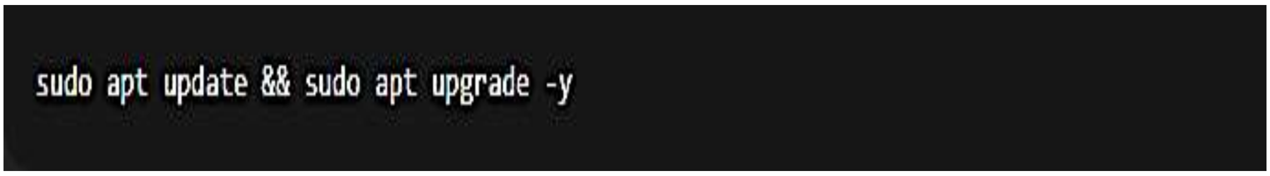
Table 4.1 – Comparison of microcontrollers

While STM32 and Arduino are great for low-level tasks, they fall short when it comes to camera support, parallel processing, and advanced logic

4.7.3 SYSTEM SETUP AND CONFIGURATIONOS INSTALLATION:

- Flashed **Raspberry Pi OS Lite** to a microSD using Raspberry Pi Imager
- Enabled **SSH** and **Wi-Fi** by placing ssh and wpa_supplicant.conf in /boot

- Connected via terminal and ran initial system updates



```
sudo apt update && sudo apt upgrade -y
```

Figure 4.1: OS Installation

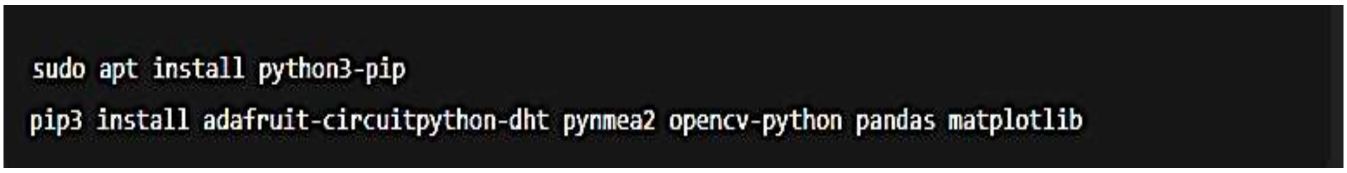
Configuration via raspi-config:

- Enabled **Camera** and **UART**
- Set timezone, expanded filesystem, and enabled **I2C**
- Created a working folder at /home/pi/env_monitor/ for code and outputs

Installed Python Libraries:

sudo apt install python3-pip

pip3 install adafruit-circuitpython-dht pynmea2 opencv-python pandas matplotlib



```
sudo apt install python3-pip
pip3 install adafruit-circuitpython-dht pynmea2 opencv-python pandas matplotlib
```

Figure 4.2: Installed Python Libraries

4.7.4 SENSOR INTEGRATION AND SOFTWARE COORDINATION

The Raspberry Pi connects with the following modules:

Sensor	Interface	GPIO Pins	Python Library
DHT22	Digital GPIO	GPIO 4	adafruit-circuitpython-dht
NEO-6M GPS	UART	GPIO 14 (TX), 15 (RX)	pynmea2, pyserial
Pi Camera	CSI Ribbon	Camera Slot	libcamera, opencv-python

Table 4.2 – Connecting sensors to the Raspberry Pi.

Each module is managed through a dedicated Python script. **The system logs:**

- Temperature (used for SST estimation)
- GPS coordinates (used for geo-tagging data)

- NDVI and Chlorophyll estimation from captured images

Outputs are saved as **timestamped CSV files** and image folders

4.7.5 SYSTEM OPERATION AND SOFTWARE FLOWAFTER DEVELOPMENT

- Scripts are placed in /home/pi/env_monitor/
- Made executable using chmod +x
- Scheduled to run using cron, systemd, or tmux

The software performs:

- Sensor initialization with error handling
- Periodic data acquisition and NDVI computation
- File I/O and structured logging
- Coordinated multitasking to simulate real satellite data operations

4.7.6 SUMMARY

The Raspberry Pi 4 successfully mimics the behavior of a CubeSat-class onboard controller:

- Fusion of camera, GPS, and environmental sensors
- Autonomous processing and coordination
- Structured storage for future AI use
- Scalable for more complex tasks like **LSTM-based prediction**

Its performance, flexibility, and ease of development made it the perfect choice for our CubeSat-inspired embedded system

4.8 SENSOR SOFTWARE MODULES

4.8.1 CHLOROPHYLL ESTIMATION USING RASPBERRY PI CAMERA AND NDVI APPROXIMATION

Accurate monitoring of chlorophyll concentration plays a pivotal role in assessing the health of marine ecosystems and forecasting fish population dynamics. In oceanographic contexts, elevated chlorophyll-a levels typically indicate the presence of nutrient-rich waters, which in turn support dense phytoplankton communities. Since phytoplankton form the base of aquatic food chains, chlorophyll-a becomes a crucial indicator when identifying areas likely to host higher fish concentrations.

In full-scale satellite missions such as ESA's Sentinel-2, chlorophyll levels are derived using multispectral imaging sensors like the **MultiSpectral Instrument (MSI)**. These sensors are capable of capturing specific spectral bands across the visible and near-infrared (NIR) ranges. One of the most widely-used indices derived from such data is the **Normalized Difference Vegetation Index (NDVI)**, which serves as a proxy for assessing biomass, vegetation, and chlorophyll concentration.

Due to the constraints associated with CubeSat-scale systems—including cost, power, volume, and hardware complexity—integrating true multispectral sensors is often impractical. To overcome this challenge in our CubeSat-inspired ground-based prototype, we utilize a **standard Raspberry Pi Camera Module** to capture RGB imagery and compute a **simplified NDVI-like index**. Although this does not offer the spectral fidelity of satellite-grade instruments, it serves as an effective and educationally valuable method to simulate real-world remote sensing logic

4.8.1.1 NDVI AND ITS ROLE IN CHLOROPHYLL ESTIMATION

The **Normalized Difference Vegetation Index (NDVI)** is a well-established remote metric that quantifies vegetation vigor and chlorophyll concentration by evaluating the differential reflectance in the **near-infrared (NIR)** and **red** spectral bands. The mathematical expression sensing for NDVI is:

$$\mathbf{NDVI} = \frac{(\mathbf{NIR} - \mathbf{Red})}{(\mathbf{NIR} + \mathbf{Red})}$$

In terrestrial environments, healthy vegetation absorbs most visible light (especially red) and reflects a large proportion of NIR light. In aquatic settings, particularly in phytoplankton-rich areas, similar principles apply—light absorption and reflectance characteristics correlate with chlorophyll-a concentration.

However, the Raspberry Pi Camera does not include a dedicated NIR sensor. As such, we cannot compute true NDVI. To simulate this functionality, we implement a **substitute NDVI approximation** based on empirical methods. In our system:

- The **Red channel** from the RGB image is used as a proxy for red reflectance.
- The **Blue channel** is empirically treated as a rough approximation for NIR reflectance.

This method is supported by scientific studies suggesting that under consistent and controlled lighting conditions, the blue channel may loosely mimic certain NIR-reflective behaviors. Though this technique lacks the quantitative accuracy of true multispectral sensors, it is sufficient to observe **relative chlorophyll distribution** across spatial scenes in a simulated environment.

4.8.1.2 RASPBERRY PI CAMERA AS A MULTISPECTRAL SUBSTITUTE

The imaging hardware used in this project is the **Raspberry Pi Camera Module v2**, which is based on the **Sony IMX219 8MP sensor**. While the camera is fundamentally designed for general-purpose RGB photography and video capture, it supports high-resolution image acquisition that can be repurposed for environmental monitoring tasks.

In the absence of a dedicated NIR channel, we compute an **NDVI-like approximation** defined by the following formula:

$$\mathbf{NDVI(approx)} = \frac{(\mathbf{Blue} - \mathbf{Red})}{(\mathbf{Blue} + \mathbf{Red} + \boldsymbol{\varepsilon})}$$

Where:

- **Red** and **Blue** represent pixel intensity values from the respective RGB channels.
- **ε**\Var epsilon is a small positive constant added to prevent division by zero (typically ε=1×10−5\Var epsilon = 1 \times 10^{-5}).

This index allows us to detect **relative differences in chlorophyll concentration** within each captured frame. Though the technique cannot yield scientifically validated absolute values, it can still visualize patterns in optical properties and provide a **functional simulation of spaceborne spectral logic**.

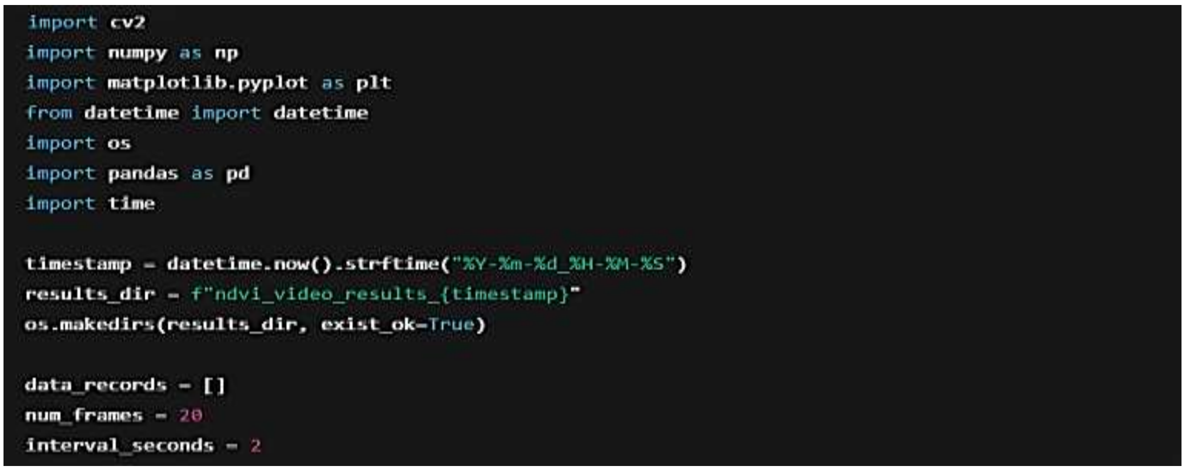
In the subsequent step, this NDVI approximation is further utilized in an empirical model to estimate chlorophyll-a concentration, enabling a low-cost yet educationally valuable simulation of satellite-grade analysis.

4.8.1.3 Step-by-Step Implementation

The implementation of chlorophyll estimation using the Raspberry Pi Camera follows a modular and reproducible software procedure, outlined as follows:

Step 1: Initialization and Setup

- Import required Python libraries: opencv-python, matplotlib, numpy, datetime, os, and pandas.
- Create a timestamped directory to store output images and results.

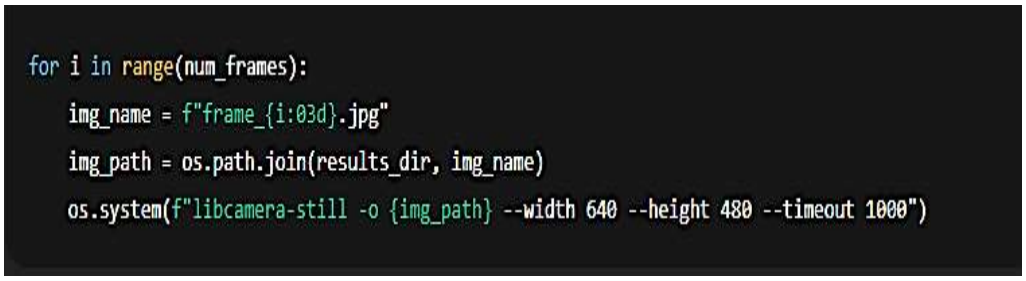


```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import os
import pandas as pd
import time

timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"ndvi_video_results_{timestamp}"
os.makedirs(results_dir, exist_ok=True)

data_records = []
num_frames = 20
interval_seconds = 2
```

Figure 4.3: Initialization and Setup

Step 2: Image Capture

- Use the libcamera-still command-line utility to capture 20 images at fixed 2-second intervals.
- Save each image in the results directory with timestamped filenames.

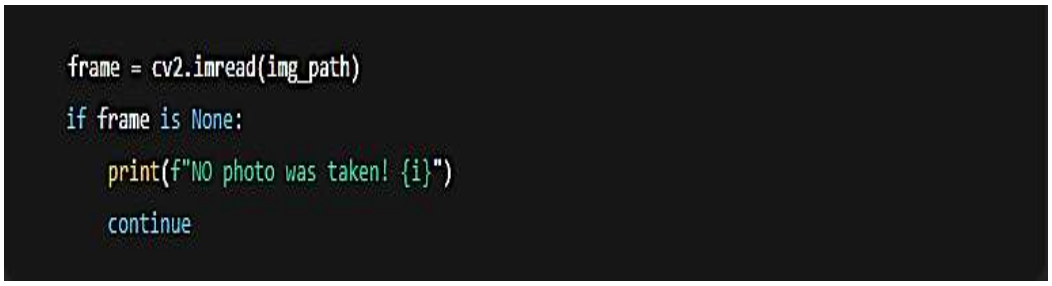


```
for i in range(num_frames):
    img_name = f"frame_{i:03d}.jpg"
    img_path = os.path.join(results_dir, img_name)
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")
```

Figure 4.4: Image Capture

Step 3: Reading Captured Frames

- Load each image using OpenCV’s cv2.imread() function.
- Validate successful image loading before proceeding with pixel-wise processing.

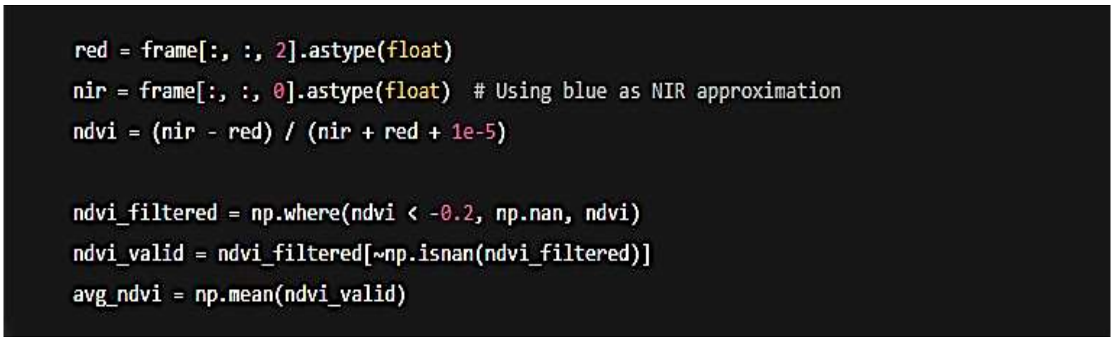


```
frame = cv2.imread(img_path)
if frame is None:
    print(f"NO photo was taken! {i}")
    continue
```

Figure 4.5: Reading Captured Frames

Step 4: NDVI Calculation

- Extract the Red and Blue channels from each image matrix.
- Apply the NDVI-approx formula to each pixel location.
- Discard NDVI values below a threshold (e.g., -0.2) to eliminate noise and invalid readings.



```
red = frame[:, :, 2].astype(float)
nir = frame[:, :, 0].astype(float)  # Using blue as NIR approximation
ndvi = (nir - red) / (nir + red + 1e-5)

ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
ndvi_valid = ndvi_filtered[~np.isnan(ndvi_filtered)]
avg_ndvi = np.mean(ndvi_valid)
```

Figure 4.6: NDVI Calculation

Step 5: Chlorophyll-a Estimation

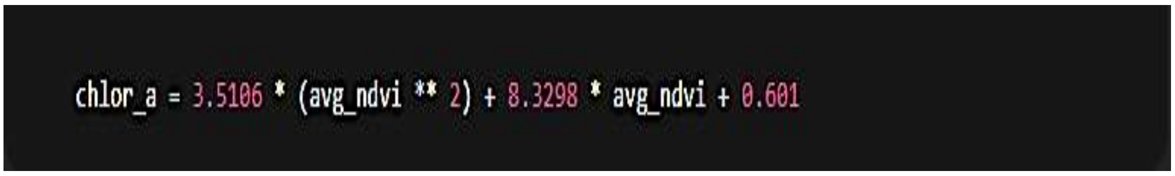
Once the average NDVI-approx value is computed for each captured image, chlorophyll-a concentration is estimated using an empirical quadratic regression model. This model is based on a study conducted on Bung Binh Thien Lake in southern Vietnam, which established a statistical relationship between NDVI and chlorophyll-a concentrations derived from Landsat 8 imagery (Nguyen et al., 2020).

The applied equation is as follows:

$$\boxed{Chlorophyll - a \;=\; a.(NDVI)^2 + b.NDVI \;+ c}$$

Where coefficients a, b, and c are derived from calibration studies or literature.

This equation was integrated into the software and implemented in Python as:



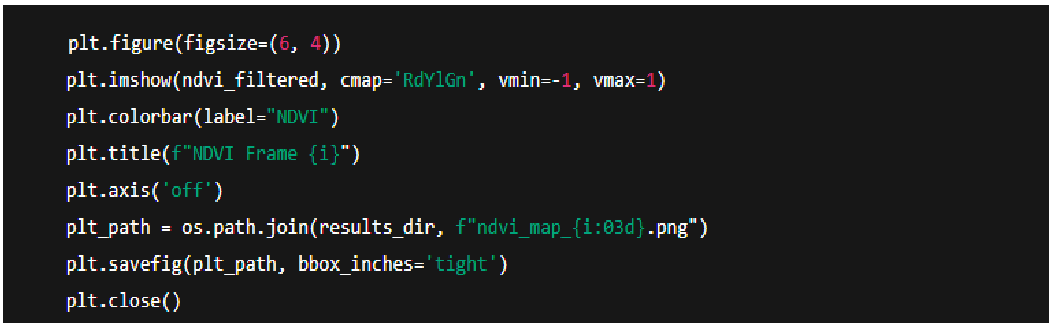
```python
chlor_a = 3.5106 * (avg_ndvi ** 2) + 8.3298 * avg_ndvi + 0.601
```

Figure 4.7: Chlorophyll-a Estimation

Although the RGB-based NDVI approximation has limitations compared to satellite-grade multispectral data, this model provides a useful educational estimation for simulating chlorophyll distribution.

Step 6: NDVI Heatmap Visualization

- Use matplotlib.pyplot.imshow() to generate color-coded NDVI heatmaps.
- Apply the 'RdYlGn' colormap for intuitive representation of chlorophyll density.
- Save each heatmap as a PNG image.

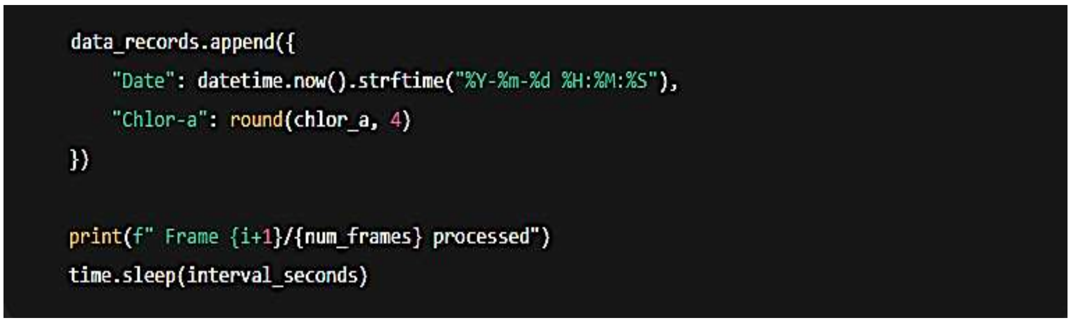


```python
plt.figure(figsize=(6, 4))
plt.imshow(ndvi_filtered, cmap='RdYlGn', vmin=-1, vmax=1)
plt.colorbar(label="NDVI")
plt.title(f"NDVI Frame {i}")
plt.axis('off')
plt_path = os.path.join(results_dir, f"ndvi_map_{i:03d}.png")
plt.savefig(plt_path, bbox_inches='tight')
plt.close()
```

Figure 4.8: NDVI Heatmap Visualization

Step 7: Logging Results into a List

- For each processed image, append a tuple containing the timestamp and estimated chlorophyll-a value to a Python list.

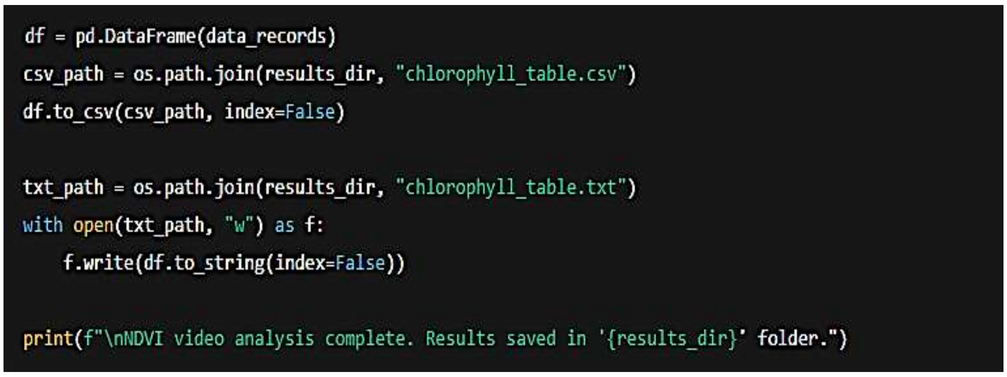


```
data_records.append({
    "Date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
    "Chlor-a": round(chlor_a, 4)
})

print(f" Frame {i+1}/{num_frames} processed")
time.sleep(interval_seconds)
```

Figure 4.9: Logging Results into a List

Step 8: Final Export to CSV and TXT

- Convert the result list into a pandas.
- DataFrame.
- Export the data into both .csv and .txt formats for later use in analysis or AI training



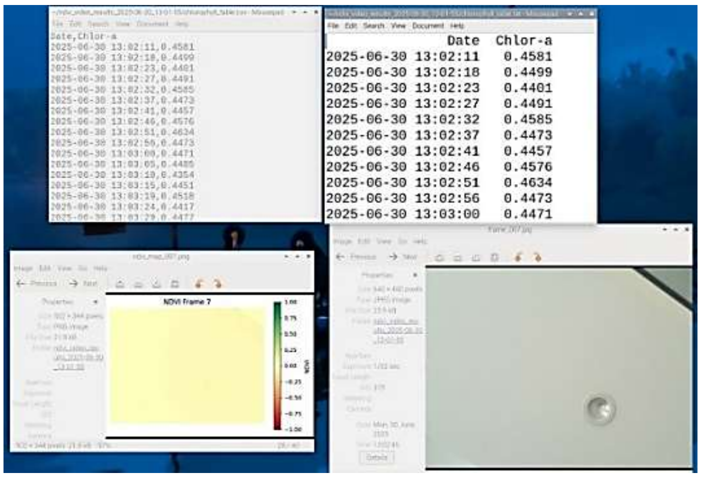
```
df = pd.DataFrame(data_records)
csv_path = os.path.join(results_dir, "chlorophyll_table.csv")
df.to_csv(csv_path, index=False)

txt_path = os.path.join(results_dir, "chlorophyll_table.txt")
with open(txt_path, "w") as f:
    f.write(df.to_string(index=False))

print(f"\nNDVI video analysis complete. Results saved in '{results_dir}' folder.")
```

Figure 4.10: Final Export to CSV and TXT

Step 9: Output and Interpretation

Upon execution, the system successfully generates:

- **20 RGB images** corresponding to field-of-view snapshots.
- **20 NDVI heatmaps** visualizing chlorophyll approximation.
- **1 dataset** logging timestamped chlorophyll-a values in tabular format.



```
Date,Chlor-a
2025-06-30 13:02:11,0.4581
2025-06-30 13:02:18,0.4499
2025-06-30 13:02:23,0.4401
2025-06-30 13:02:27,0.4491
2025-06-30 13:02:32,0.4585
2025-06-30 13:02:37,0.4473
2025-06-30 13:02:41,0.4457
2025-06-30 13:02:46,0.4576
2025-06-30 13:02:51,0.4634
2025-06-30 13:02:56,0.4473
2025-06-30 13:03:00,0.4471
2025-06-30 13:03:05,0.4485
2025-06-30 13:03:10,0.4354
2025-06-30 13:03:15,0.4451
2025-06-30 13:03:19,0.4518
2025-06-30 13:03:24,0.4417
2025-06-30 13:03:29,0.4477
```

```
               Date   Chlor-a
2025-06-30 13:02:11    0.4581
2025-06-30 13:02:18    0.4499
2025-06-30 13:02:23    0.4401
2025-06-30 13:02:27    0.4491
2025-06-30 13:02:32    0.4585
2025-06-30 13:02:37    0.4473
2025-06-30 13:02:41    0.4457
2025-06-30 13:02:46    0.4576
2025-06-30 13:02:51    0.4634
2025-06-30 13:02:56    0.4473
2025-06-30 13:03:00    0.4471
```

Figure 4.11: Output and Interpretation

4.8.1.4 SUMMARY

This module presents a creative and low-cost approach for **simulating chlorophyll monitoring** using basic, off-the-shelf hardware. While the limitations of RGB imaging preclude rigorous scientific accuracy, the implemented system:

- Offers hands-on experience with NDVI-based environmental sensing.
- Produces meaningful training data for AI models aiming to predict fish concentrations.
- Bridges theoretical concepts from satellite remote sensing with practical embedded system design.

By simulating the logic of Sentinel-2-style observations, the Raspberry Pi camera setup plays a foundational role in realizing the objectives of this CubeSat-inspired environmental monitoring platform.

4.8.2 SEA SURFACE TEMPERATURE ESTIMATION USING DHT22 SENSOR

Sea Surface Temperature (SST) is one of the most critical parameters in the study of oceanographic phenomena, marine biodiversity, and global climate systems. In professional satellite missions such as ESA's **Sentinel-3**, SST is measured with high precision using instruments like the **Sea and Land Surface Temperature Radiometer (SLSTR)**, which captures thermal infrared emissions to determine sea surface thermal gradients.

However, CubeSat-class platforms face limitations in terms of power consumption, hardware footprint, and cost. This project, therefore, adopts a simplified approach. Instead of infrared thermal sensing, we employ a **DHT22 sensor**—commonly used for ambient air measurements—to record air temperature, which we then convert into an estimated SST using an **empirical model derived from peer-reviewed scientific research**. This method simulates the SST sensing logic of full-scale satellites while remaining accessible and cost-effective for educational applications.

4.8.2.1 WHY DHT22?

The **DHT22 sensor** (also known as AM2302) is a digital temperature and humidity sensor that provides readings with reasonable accuracy for educational and prototyping applications. It communicates via a digital signal, integrates seamlessly with the Raspberry Pi via GPIO pins, and requires minimal calibration or configuration.

Key reasons for selecting the DHT22 include:

- Affordable and widely available
- Compatible with Raspberry Pi (via Python libraries)
- Sufficient temperature accuracy (±0.5°C) for our approximation model
- Supports frequent and continuous reading

However, the DHT22 **does not measure SST directly**. Instead, we apply an empirical correlation between air temperature and SST, as presented by **El-Geziry et al. (2023)**. This allows us to simulate a spaceborne SST measurement pipeline using ground-based sensor data.

4.8.2.2 SCIENTIFIC BASIS FOR SST ESTIMATION

The SST estimation method used in this prototype is grounded in the work of **T. M. El-Geziry et al. (2023)**, as published in the *Journal of Marine Science and Engineering* under the title:

"Air–Sea Interaction and Estimation of SST in the Eastern Mediterranean and Red Sea using Empirical Models."

In their study, the researchers conducted a comprehensive analysis of air–sea thermal relationships using observational air temperature data and SST datasets obtained from satellite imagery. Their work resulted in a **piecewise linear model** that accounts for **seasonal variability** between cold and warm months.

The model is defined as follows:

- If air temperature < 24 ˚C (winter condition)

  $$\mathbf{SST = 0.3832 \times T_a + 12.154}$$

- If air temperature < 24 ˚C (summer condition)

  $$\mathbf{SST = 0.6567 \times T_a + 5.4271}$$

Where:

- T_a is air temperature in ˚C (measured by the DHT22 sensor)
- SST is estimated Sea Surface Temperature in ˚C

This model reflects the physics of air–sea coupling:

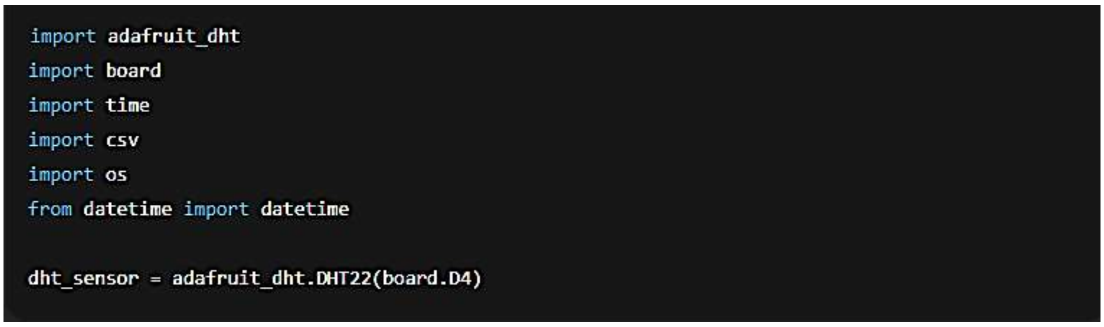
- In **colder seasons**, the SST remains relatively buffered and changes slowly with air temperature.
- In **warmer seasons**, SST increases more rapidly with rising air temperatures.

The model was validated for **Red Sea** and **Eastern Mediterranean** conditions—both of which are relevant to this prototype's environmental context.

4.8.2.3 STEP-BY-STEP IMPLEMENTATION

Step 1: Library Imports and Sensor Initialization

- Import essential Python libraries such as adafruit_dht, board, datetime, os, and csv.
- Initialize the DHT22 sensor on **GPIO pin 4** of the Raspberry Pi.



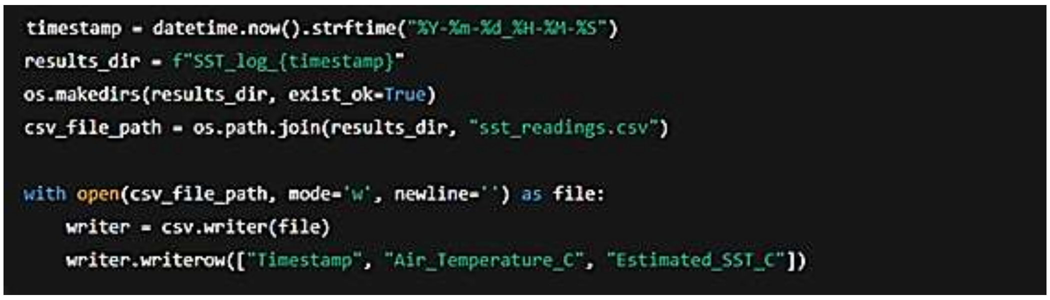
```
import adafruit_dht
import board
import time
import csv
import os
from datetime import datetime

dht_sensor = adafruit_dht.DHT22(board.D4)
```

Figure 4.12: Libraries

Step 2: Directory and CSV File Setup

- Create a folder named with the current timestamp to store results.
- Create a .csv file with the following headers:

- Timestamp
- Air Temperature (°C)
- Estimated SST (°C)

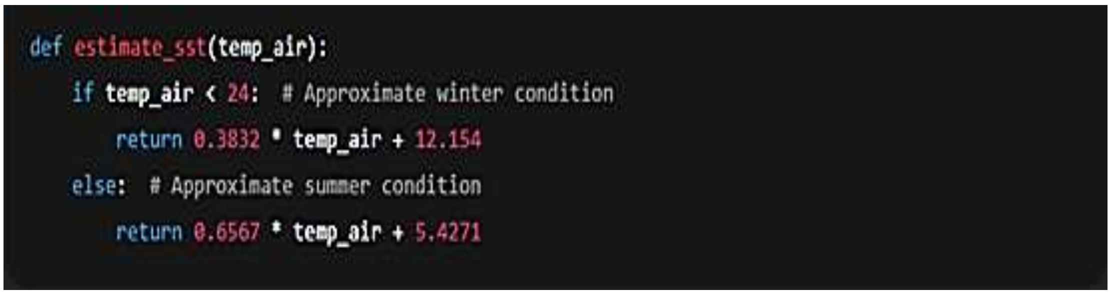


```
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"SST_log_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_file_path = os.path.join(results_dir, "sst_readings.csv")

with open(csv_file_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Air_Temperature_C", "Estimated_SST_C"])
```

Figure 4.13: Create a folder to save the output.

Step 3: SST Estimation Function

- Define a Python function estimate_sst() that applies the piecewise model depending on whether the air temperature is above or below 24°C.

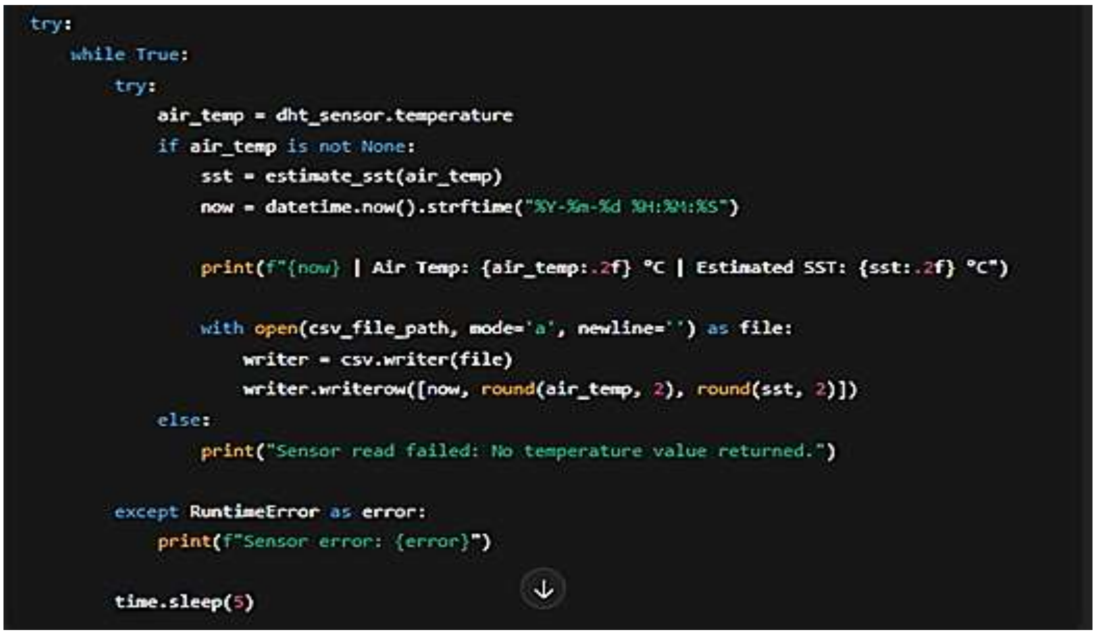


```
def estimate_sst(temp_air):
    if temp_air < 24:  # Approximate winter condition
        return 0.3832 * temp_air + 12.154
    else:  # Approximate summer condition
        return 0.6567 * temp_air + 5.4271
```

Figure 4.14: Estimated SST.

Step 4: Continuous Data Acquisition

- Use a while True loop to continuously:
    - Read air temperature from DHT22
    - Estimate SST using the function
    - Print the result to the console
    - Append the results to the CSV file

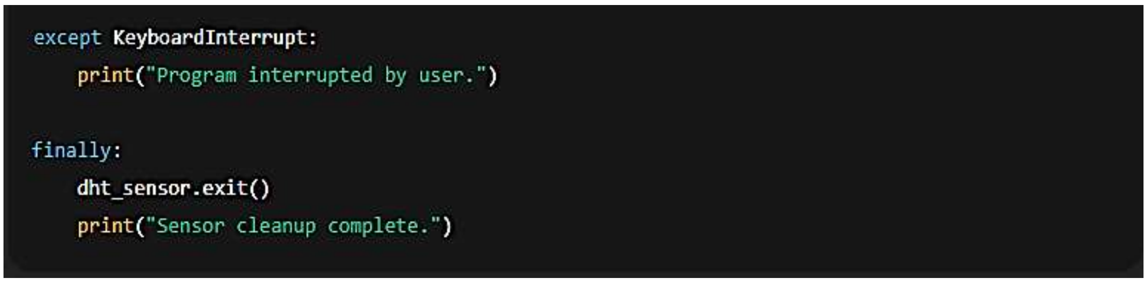


```
try:
    while True:
        try:
            air_temp = dht_sensor.temperature
            if air_temp is not None:
                sst = estimate_sst(air_temp)
                now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

                print(f"{now} | Air Temp: {air_temp:.2f} °C | Estimated SST: {sst:.2f} °C")

                with open(csv_file_path, mode='a', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow([now, round(air_temp, 2), round(sst, 2)])
            else:
                print("Sensor read failed: No temperature value returned.")

        except RuntimeError as error:
            print(f"Sensor error: {error}")

        time.sleep(5)
```

Figure 4.15: Collect , processing and save data .

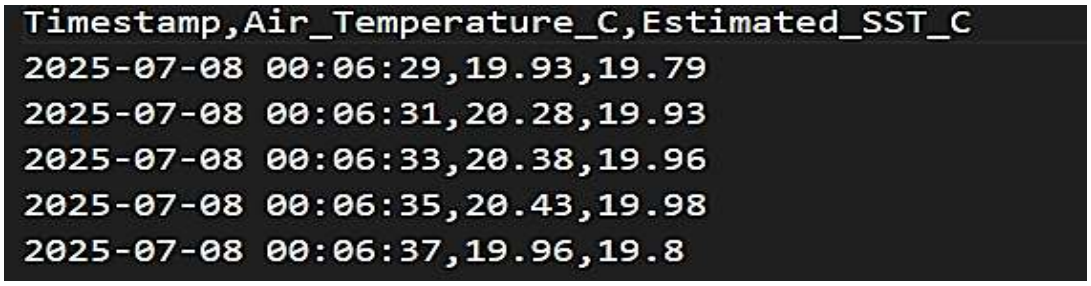
Step 5: Error Handling and Cleanup

- Incorporate a try-except-finally block to:
  - Gracefully handle runtime errors during sensor reads
  - Allow user interruption via Ctrl+C
  - Ensure proper sensor deactivation and GPIO cleanup at the end of the session



```
except KeyboardInterrupt:
    print("Program interrupted by user.")

finally:
    dht_sensor.exit()
    print("Sensor cleanup complete.")
```

Figure 4.16: clean up



```
Timestamp,Air_Temperature_C,Estimated_SST_C
2025-07-08 00:06:29,19.93,19.79
2025-07-08 00:06:31,20.28,19.93
2025-07-08 00:06:33,20.38,19.96
2025-07-08 00:06:35,20.43,19.98
2025-07-08 00:06:37,19.96,19.8
```

Figure 4.17: Model of the output data in CSV file

4.8.2.4 SUMMARY

This software module demonstrates a **cost-effective** and **practical solution** for approximating sea surface temperature in a CubeSat-inspired prototype. While the DHT22 does not directly measure SST, the incorporation of **empirical modeling** rooted in peer-reviewed marine science provides a meaningful educational simulation of real-world satellite measurements.

Key benefits of this approach include:

- Reinforcing empirical modeling and data-driven simulation techniques
- Supporting environmental AI models with relevant temperature inputs
- Providing students with hands-on experience in remote sensing logic

The integration of the DHT22 sensor, alongside a scientifically validated SST estimation model, underscores the potential for **low-cost embedded systems to simulate complex satellite functionalities** in academic and research settings

4.8.3 GPS LOGGING USING RASPBERRY PI AND NEO-6M MODULE

Geolocation plays a pivotal role in satellite-based Earth observation missions. Accurately tagging sensor measurements with latitude and longitude allows scientists to analyze spatial patterns, track environmental changes, and fuse datasets across space and time.

In high-end platforms such as the **Sentinel satellites** from the European Space Agency (ESA), Global Navigation Satellite System (GNSS) modules—including GPS, Galileo, and GLONASS—enable **real-time orbit determination** and **precise georeferencing**. These systems ensure that each data product—whether it be chlorophyll concentration from Sentinel-3 or land use maps from Sentinel-2—is accurately aligned with its geographic location.

For instance:

- **Sentinel-2** provides high-resolution optical imagery, geotagged using GNSS input.
- **Sentinel-3** uses precise orbit tracking to associate sea surface temperature, ocean color, and altimetry data with exact coordinates.

Our **CubeSat-inspired prototype** emulates this capability by integrating a **NEO-6M GPS module** with a **Raspberry Pi**, allowing us to log real-time geolocation data alongside other sensor readings (such as SST and chlorophyll). Although we do not achieve sub-meter accuracy like in spaceborne systems, our setup provides a **low-cost and educational simulation of spatial data tagging**.

4.8.3.1 WHY NEO-6M GPS MODULE?

The **u-blox NEO-6M** is a widely adopted GPS receiver used in embedded systems and educational platforms. It offers sufficient positioning accuracy and rapid satellite lock, making it ideal for low-cost Earth observation prototypes.

Key features that support its selection include:

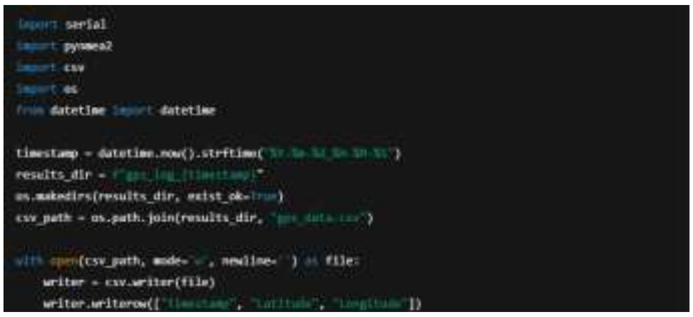
- **Positioning Accuracy**: Typically around ±2.5 meters in autonomous mode.
- **Supported Protocols**: Outputs NMEA sentences like $GPGGA, $GPRMC, which are standard in navigation systems.
- **Low Power Consumption**: Efficient enough for battery-powered missions.
- **UART Interface**: Communicates over a serial port (e.g., /dev/ttyS0 or /dev/ttyUSB0).
- **Fast Acquisition**: Capable of cold, warm, and hot starts with excellent sensitivity.

Despite the absence of correction techniques like Differential GPS (DGPS) or SBAS augmentation, the NEO-6M serves as a **practical tool for spatial awareness**, enabling us to simulate how real satellites tag environmental data with geolocation metadata.

4.8.3.2 STEP-BY-STEP IMPLEMENTATION

Step 1: Import Required Libraries and Set Up Directory

We start by importing libraries required for serial communication and GPS data parsing:

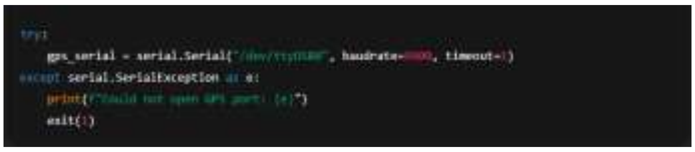


```
import serial
import pynmea2
import csv
import os
from datetime import datetime

timestamp = datetime.now().strftime('%Y-%m-%d_%H-%M-%S')
results_dir = f"gps_log_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "gps_data.csv")

with open(csv_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Latitude", "Longitude"])
```

Figure 4.18: Libraries and create the folder with the time.

Step 2: Initialize Serial Port

We establish a serial connection with the NEO-6M module, typically over /dev/ttyUSB0 at a baud rate of **9600**:



```
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"Could not open GPS port: {e}")
    exit(1)
```

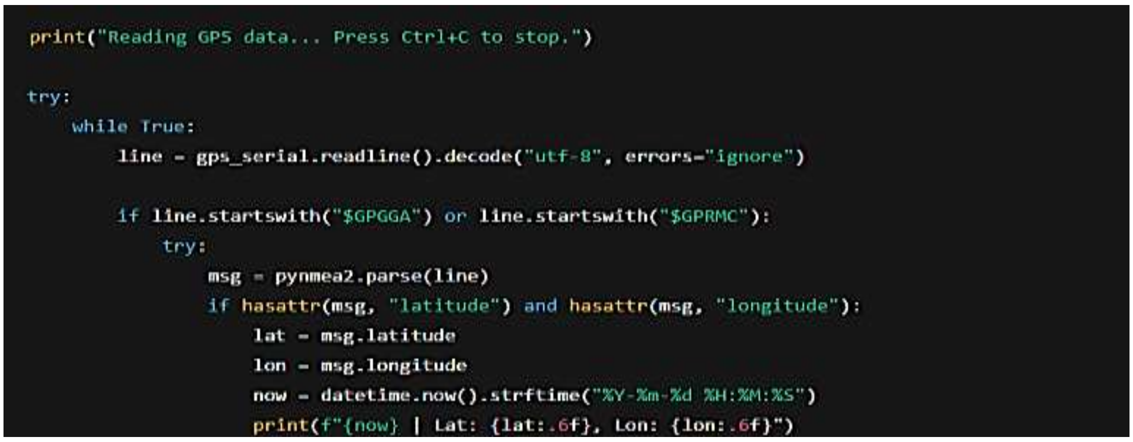
Figure 4.19: Connect with the NEO-6M

Step 3: Read and Parse NMEA Sentences

The GPS module outputs multiple sentence types. We focus on $GPRMC or $GPGGA, which include critical fields such as:

- Latitude and Longitude
- UTC Timestamp
- Fix Quality and Number of Satellites

Each sentence is parsed as follows:

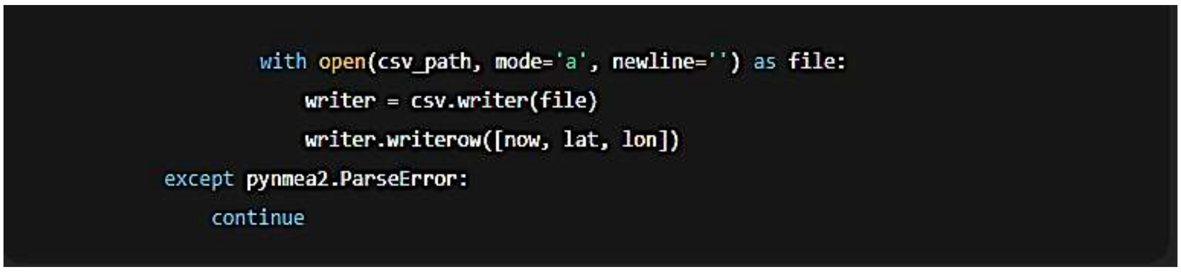


```
print("Reading GPS data... Press Ctrl+C to stop.")

try:
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")

        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    lat = msg.latitude
                    lon = msg.longitude
                    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    print(f"{now} | Lat: {lat:.6f}, Lon: {lon:.6f}")
```

Figure 4.20: Detect Lat and lon

Step 4: Log Data to CSV

Each successful reading is logged into a CSV file along with a local timestamp:

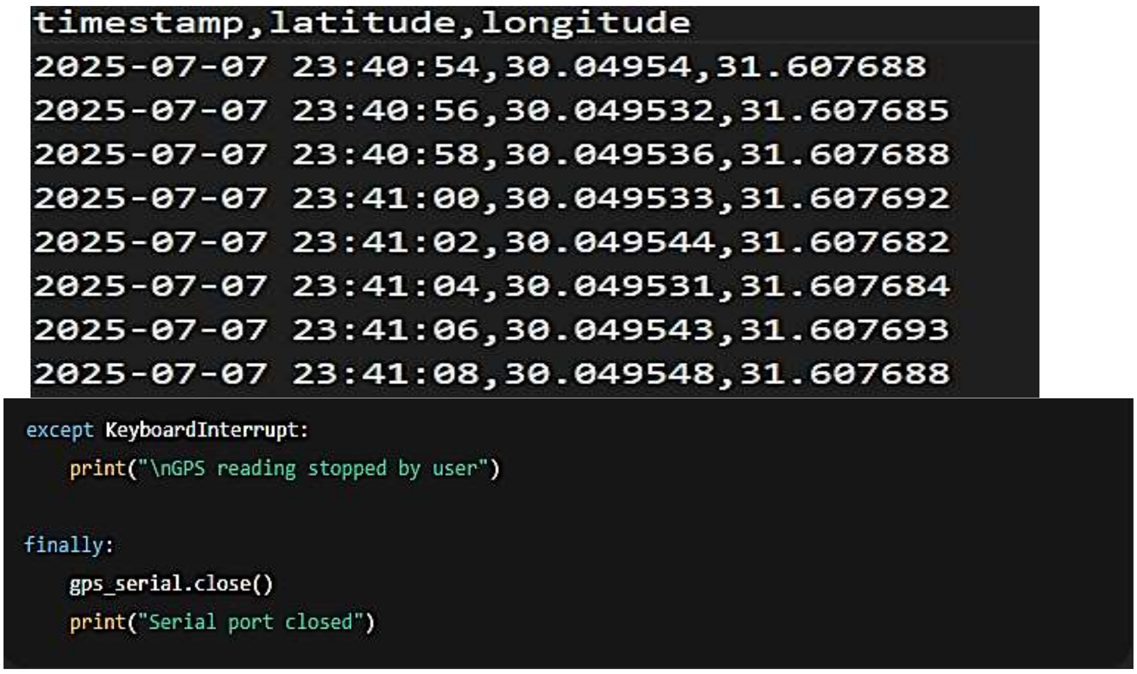


```
                    with open(csv_path, mode='a', newline='') as file:
                        writer = csv.writer(file)
                        writer.writerow([now, lat, lon])
            except pynmea2.ParseError:
                continue
```

Figure 4.21: Save as CSV

Step 5: Graceful Exit Handling

To ensure the serial port is properly closed upon termination (e.g., Ctrl+C by user), we include cleanup logic:



```
timestamp,latitude,longitude
2025-07-07 23:40:54,30.04954,31.607688
2025-07-07 23:40:56,30.049532,31.607685
2025-07-07 23:40:58,30.049536,31.607688
2025-07-07 23:41:00,30.049533,31.607692
2025-07-07 23:41:02,30.049544,31.607682
2025-07-07 23:41:04,30.049531,31.607684
2025-07-07 23:41:06,30.049543,31.607693
2025-07-07 23:41:08,30.049548,31.607688
```

```
except KeyboardInterrupt:
    print("\nGPS reading stopped by user")

finally:
    gps_serial.close()
    print("Serial port closed")
```

Figure 4.22: The end of code

4.8.3.3 SUMMARY

This module completes our embedded sensor suite by introducing **geolocation capability**, mimicking the full satellite workflow where every data point is tagged with spatial coordinates.

Benefits of using the NEO-6M in our educational CubeSat prototype include:

- **Simulated spatial tagging** of environmental data
- **Emulation of GNSS-based data fusion** as in real missions like Sentinel-2/3
- **Real-time latitude/longitude tracking**, synchronized with sensor readings

CSV logging for use in downstream analysis or integration with mapping platforms

Figure 4.23: The output of GPS

By integrating GPS functionality, our system reaches a higher level of **realism, traceability, and scientific relevance**, bridging the gap between ground-based learning and orbital mission logic

4.8.4 INTEGRATED SENSOR PIPELINE (SST & NDVI & GPS)

In order to build a fully functional environmental monitoring CubeSat-inspired platform, it was essential to integrate the outputs of all individual sensors into a single, coherent software pipeline. This integration combines air temperature readings from the DHT22 sensor, chlorophyll estimation using NDVI from camera images, and geographic location data (either from a GPS module or simulated coordinates).

4.8.4.1 DATA FLOW AND EXECUTION

The integrated script follows a sequential flow:

1. Capture a frame using the Raspberry Pi camera.
2. Compute NDVI and estimate chlorophyll concentration.
3. Read temperature from the DHT22 sensor.
4. Estimate Sea Surface Temperature (SST) based on air temperature.
5. Fetch location coordinates, either from the GPS module or a fallback random generator.
6. Log all parameters into a CSV file, including a timestamp.

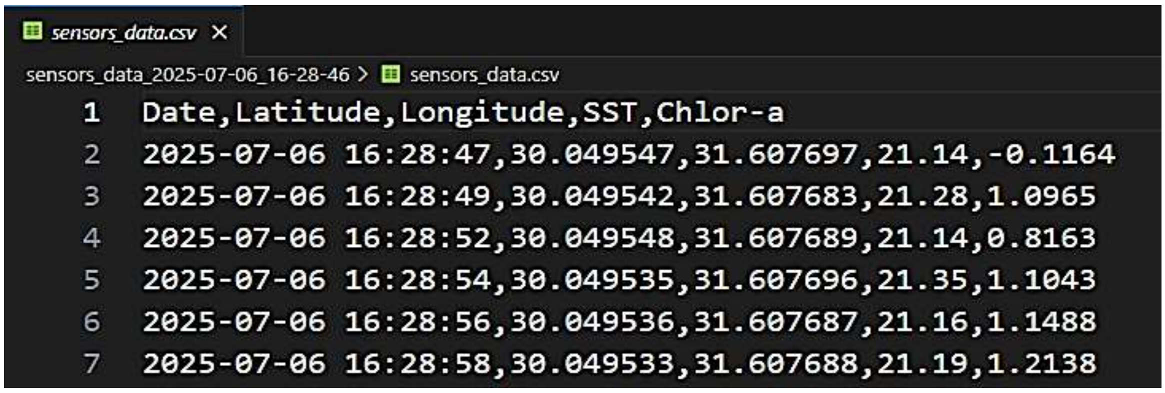
This integration ensures that all sensors contribute synchronously to each data entry, preserving temporal consistency and simplifying data analysis

4.8.4.2 OUTPUT STRUCTURE

Each entry in the resulting CSV log contains the following fields:

- Timestamp
- Latitude
- Longitude
- Estimated SST
- chlorophyll proxy

The file naming and folder structure are timestamped for traceability and organized according to the experiment run, as illustrated in Figure 3.20.



```
Date,Latitude,Longitude,SST,Chlor-a
2025-07-06 16:28:47,30.049547,31.607697,21.14,-0.1164
2025-07-06 16:28:49,30.049542,31.607683,21.28,1.0965
2025-07-06 16:28:52,30.049548,31.607689,21.14,0.8163
2025-07-06 16:28:54,30.049535,31.607696,21.35,1.1043
2025-07-06 16:28:56,30.049536,31.607687,21.16,1.1488
2025-07-06 16:28:58,30.049533,31.607688,21.19,1.2138
```

Figure 4.24: The output of integration Sensor

4.9 CHALLENGES AND WORKAROUNDS

During the development of the embedded software for our CubeSat-inspired environmental monitoring system, we encountered several hardware limitations that prevented the use of certain ideal sensors. Rather than omitting critical functionalities, we adopted practical workarounds using available components and estimation models. This section highlights the major challenges faced and the corresponding solutions we implemented.

4.9.1 ABSENCE OF A DEDICATED CHLOROPHYLL SENSOR

One of the primary challenges was the unavailability of a dedicated chlorophyll sensor, such as a multispectral module (e.g., AS7265x or Pi NoIR camera). These sensors are typically required to detect specific light wavelengths that correlate with chlorophyll-a concentration in water bodies.

To address this, we utilized a standard RGB camera, capturing sequential images of the target area. Using the red and blue channels of each image, we calculated the Normalized Difference Vegetation Index (NDVI), which serves as a proxy for estimating chlorophyll-a levels. The estimation formula was derived from published scientific research.

Although the RGB camera lacks the precision of a true multispectral sensor, this approach provided a functional and cost-effective alternative suitable for prototype-level environmental onitoring.

4.9.2 LACK OF DIRECT SST MEASUREMENT CAPABILITY

Our project also required Sea Surface Temperature (SST) readings. However, we did not have access to a thermal infrared sensor capable of measuring water temperature directly. As a workaround, we deployed a DHT22 sensor to measure ambient air temperature.

We then applied a mathematical model—based on environmental research—to estimate SST values from the air temperature readings. The model included two equations that adapted the estimation to seasonal temperature variations.

While this estimation approach introduces a margin of error, it allowed us to simulate SST readings within reasonable accuracy for our use case.

4.9.3 GPS MODULE MALFUNCTION AND SIMULATED POSITIONING

Another challenge involved the GPS module, which failed to provide consistent or reliable location data during testing. Since spatial data logging was critical for our dataset structure and later visualization, we created a synthetic GPS dataset.

Using a base location (centered around the Egyptian Space Agency), we introduced small randomized offsets in latitude and longitude values to simulate realistic movement. This allowed us to continue testing our logging pipeline, while leaving room for future replacement with actual GPS hardware.

4.9.4 SUMMARY

These workarounds allowed us to maintain the core objectives of environmental data acquisition, even with limited resources. Such adaptations are valuable in educational and low-cost satellite projects, where flexibility and creativity often substitute for high-end instrumentation.

Chapter (5): **Power and PCB Design**

5.1 INTRODUCTION

5.1.1 IMPORTANCE OF POWER SYSTEMS IN SPACE APPLICATIONS

In space applications, the power system is one of the most critical subsystems of any satellite, including CubeSats. It is responsible for generating, storing, regulating, and distributing electrical energy to all onboard electronics, ensuring uninterrupted operation throughout the mission. Unlike terrestrial systems, satellites in space cannot rely on external power grids or maintenance; once launched, they must be completely self-sustaining.

Power availability directly determines mission lifetime, communication windows, data collection cycles, and overall reliability. Any interruption or mismanagement of power can lead to partial or complete mission failure. For instance, a brief power drop might reset the onboard computer, interrupt data transmission, or even damage sensitive instruments.

Moreover, the harsh environment of space imposes several challenges on the power system — including extreme temperature variations, radiation exposure, and limited space for hardware. This necessitates the use of highly efficient, redundant, and radiation-hardened components. Typical power sources in CubeSats include solar panels combined with rechargeable batteries, managed by an intelligent Power Management Unit (PMU) that ensures stable voltage levels and protects against overcurrent, overvoltage, and deep discharge conditions.

Ultimately, the power system is not just a utility provider — it is the lifeline of the entire satellite. Its failure means the failure of the mission. Therefore, careful planning, simulation, and testing of the power system is a top priority in any space system design**.**

5.1.2 ROLE OF PCB DESIGN IN EMBEDDED AND CUBESAT PROJECTS

A Printed Circuit Board (PCB) is the physical platform upon which all electronic components of a system are mounted and electrically interconnected. It provides both the mechanical support and electrical pathways necessary for components to operate as an integrated system. PCBs replace bulky and unreliable point-to-point wiring by using conductive copper traces etched onto rigid or flexible substrates, typically FR4.

In embedded systems and CubeSat projects, PCB design plays a central role in ensuring proper functionality, signal integrity, thermal management, and system reliability. Unlike generic breadboard-based prototypes, a well-designed PCB allows the integration of multiple subsystems into a compact, lightweight, and robust form factor, which is essential in space-constrained environments like CubeSats.

For CubeSats, the PCB is not just a circuit carrier; it is a structural and thermal element of the system. It must withstand mechanical vibrations during launch, thermal cycling in orbit, and the effects of space radiation. Therefore, PCB design must follow strict engineering rules — including optimized component placement, trace routing, power plane allocation, and EMI/EMC control — to ensure both electrical and environmental resilience.

Additionally, the PCB integrates various subsystems: power regulation circuits, communication modules, sensor interfaces, and onboard processing units.
While multi-layer designs offer advantages in signal routing and noise reduction, even with a single-layer design—as used in this project—careful trace planning and grounding strategies can ensure sufficient signal integrity and system stability for embedded applications.

In short, the PCB serves as the backbone of any embedded or satellite system. A well-engineered PCB design transforms theoretical schematics into real-world functionality and plays a decisive role in the mission's performance, reliability, and success.

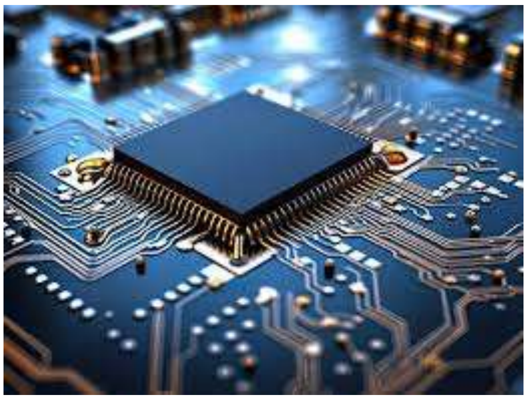


Figure 5.1 –Printed Circuit Board (PCB)

5.2 SYSTEM COMPONENTS AND POWER REQUIREMENTS

5.2.1 OVERVIEW OF SENSORS AND MODULES USED

5.2.1.1 RASPBERRY PI 4

The **Raspberry Pi 4 Model B** is a compact, powerful, single-board computer that plays a central role in many embedded and prototyping systems. In our project, it functions as the main processing unit — responsible for controlling peripherals, capturing data, and handling communication.

Equipped with a **64-bit quad-core ARM Cortex-A72 processor running at 1.5 GHz** and up to **8 GB of LPDDR4 RAM**, the Pi 4 delivers significant performance suitable for real-time applications like image processing, GPS tracking, and sensor data acquisition.

Interfaces and Connectivity

The board offers a wide array of connectivity options:

- **USB Ports**: 2× USB 3.0 and 2× USB 2.0 for high-speed data exchange
- **HDMI**: Dual micro-HDMI outputs supporting 4K video

- **Camera and Display**: 2-lane MIPI CSI and DSI connectors
- **GPIO**: 40-pin header exposing **28 GPIOs** for UART, I2C, SPI, PWM, and general I/O
- **Networking**: Gigabit Ethernet and dual-band Wi-Fi (802.11ac) with Bluetooth 5.0

Power Requirements

According to the official datasheet, the Pi 4 requires a **5V power supply capable of delivering up to 3A via USB-C**. However, in practice, when minimal peripherals are connected, a **5V/2.5A** supply may suffice.

Absolute maximum input voltage: **6.0V**
Typical active current: **~1A to 1.5A**, depending on usage (CPU, camera, USB peripherals)

In our project, the Pi 4 is powered through a regulated 5V line from an external power bank, which delivers enough current to support the device and its connected modules.

Thermal Management

The board employs dynamic clock and voltage scaling to control its thermal output. It is rated to operate within a temperature range of **0°C to 50°C**, and throttles performance to maintain a safe core temperature (≤ 85°C). In embedded or enclosed environments, passive or active cooling may be required.

Role in Project

In our CubeSat-inspired system, the Raspberry Pi 4 serves as the core controller, handling:

- Image capture from the Raspberry Pi Camera
- Data acquisition from DHT22 and GPS
- Communication over USB and GPIO
- Data logging and potential AI-based processing

Its performance and flexibility make it ideal for development and demonstration of space-inspired embedded systems.

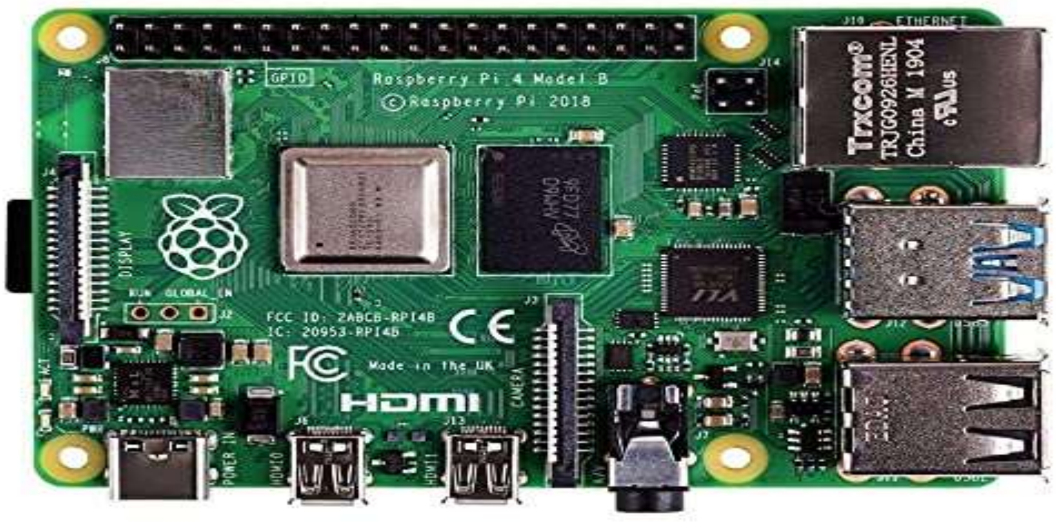


Figure 5.3 – Raspberry Pi 4

5.2.1.2 RASPBERRY PI CAMERA MODULE V2

The **Raspberry Pi Camera Module v2** is a compact, high-resolution digital imaging sensor designed specifically for Raspberry Pi boards. It features the **Sony IMX219** 8-megapixel CMOS sensor, which replaced the earlier OmniVision OV5647 (5MP) sensor used in the original module. This upgrade brought significant improvements in image clarity, color fidelity, and low-light performance.

Specifications

- **Sensor:** Sony IMX219 (8 MP, Exmor R back-illuminated)
- **Video Modes:**
  - 1080p 30 fps
  - 720p 60 fps
  - VGA 90 fps
- **Stills Resolution:** Up to 3280 × 2464 pixels
- **Interface:** 15-pin ribbon cable via **CSI (Camera Serial Interface)** port on the Raspberry Pi
- **Compatibility:** Works with all Raspberry Pi models (Pi 1, 2, 3, 4)

Power and Operation

The camera draws power directly from the Raspberry Pi through the CSI interface. Under typical conditions, it consumes around **250 mA at 5V** during video capture. No additional external power is required.

Software Support

The module can be accessed through standard Raspberry Pi APIs such as **MMAL** and **V4L2**, and it is widely supported by third-party Python libraries like **Picamera**. These libraries allow users to control exposure, white balance, ISO, shutter speed, and apply real-time image processing effects.

Use in Project

In our implementation, the Camera Module v2 is responsible for capturing high-quality images and video frames. It is directly connected to the Raspberry Pi 4 via the CSI interface using a 15 cm ribbon cable. The compact size, plug-and-play nature, and high image fidelity make it suitable for embedded applications such as:

- Visual monitoring
- Object detection
- Environmental logging

Its low power consumption and full integration with the Pi make it ideal for lightweight, energy-efficient systems such as those used in CubeSat payloads and ground-based demos.

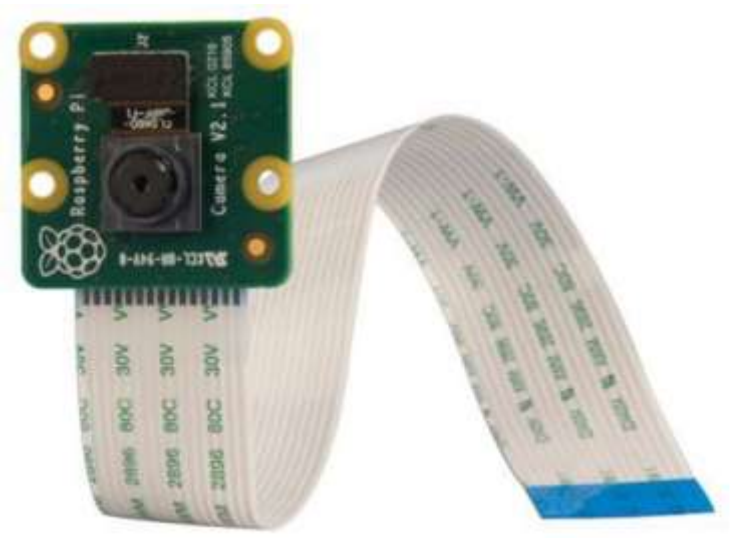


Figure 5.3-Raspberry Pi Camera Module v2

5.2.1.3 DHT22 Temperature & Humidity Sensor

The **DHT22**, also known as **AM2302**, is a high-accuracy, low-power sensor designed for digital measurement of both **relative humidity** and **ambient temperature**. It is widely used in embedded systems, environmental monitoring, and IoT applications due to its compact size and fully digital output.

Key Specifications

- **Humidity Range:** 0–100% RH, ±2% typical accuracy
- **Temperature Range:** –40°C to +80°C, ±0.5°C accuracy

- **Power Supply:** 3.3–6V DC
- **Output Signal:** Single-wire digital (custom protocol)
- **Current Consumption:**
    - Measuring: ~1.0–1.5 mA
    - Standby: ~50 µA
- **Update Rate:** ~1 reading every 2 seconds
- **Dimensions:** ~22 × 28 × 5 mm

Working Principle

DHT22 operates using a **capacitive humidity sensor** and a **thermistor**, combined with an 8-bit microcontroller for digital signal generation. The device sends pre-calibrated temperature and humidity readings using a proprietary single-wire communication protocol. The calibration data is stored in OTP memory, ensuring consistency and accuracy across devices.

Each transmission from the DHT22 includes:

- 16 bits for relative humidity (integer + decimal)
- 16 bits for temperature (integer + decimal)
- 8-bit checksum for data validation

To initiate a reading, the microcontroller sends a start signal. The sensor then responds with 40 bits of data. Timing is crucial for accurate communication, and the data line must be pulled up using a resistor (typically 10kΩ).

Advantages

- Fully factory-calibrated and temperature-compensated
- Requires no additional analog-to-digital conversion circuitry
- Long transmission distance (~20m with proper wiring)
- Stable over time with minimal drift (±0.5% RH/year)

Use in Project

In this system, the DHT22 is connected to a GPIO pin on the **Raspberry Pi 4** and powered through a regulated **3.3V** line. It periodically measures ambient temperature and humidity, which are then processed and logged by the Pi.

Due to its **low power consumption**, **digital output**, and **high accuracy**, the DHT22 is ideal for environmental monitoring in both ground-based systems and space mission payloads, where precise conditions must be recorded for analysis.

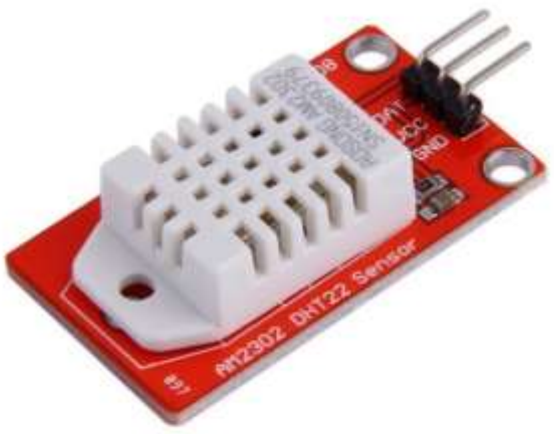


Figure 5.4-DHT22 Temperature & Humidity Sensor

5.2.1.4 NEO-6M GPS MODULE

The **NEO-6M** is a high-performance, low-power GPS module developed by u-blox, based on the **u-blox 6 positioning engine**. It is widely used in embedded systems and CubeSat projects for **real-time geolocation, time synchronization**, and **navigation tasks** due to its excellent acquisition performance and compact form factor.

Key Features

- **Receiver Type:** 50-channel, GPS L1 (1575.42 MHz), C/A code
- **Horizontal Position Accuracy:** 2.5 meters (autonomous), 2.0 meters (SBAS)
- **Acquisition Time:**
  - Cold Start: 32 seconds
  - Warm Start: 32 seconds
  - Hot Start: < 1 second
- **Maximum Navigation Rate:** 5 Hz
- **Supported Interfaces:** UART, USB, SPI, I2C (DDC)
- **Form Factor:** 16.0 × 12.2 × 2.4 mm

Power and Operating Conditions

- **Operating Voltage:** 2.7 – 3.6V (typical 3.3V)
- **Tracking Current:** ~39 mA (Max Performance Mode)
- **Power Save Mode:** as low as 17.5 mA

- **Backup Voltage (VBCKP):** 1.4 – 3.6V
- **Operating Temperature:** –40°C to +85°C
- **Antenna Support:** Active or passive, with integrated LNA support

The NEO-6M offers multiple **power management modes**:

- **Maximum Performance Mode** for faster acquisition
- **Eco Mode** for lower power draw
- **Power Save Mode** for ultra-low energy operation

Interfaces and Communication

The module outputs standard **NMEA 0183** sentences via UART at default 9600 baud. It also supports **u-blox proprietary UBX binary protocol** for advanced control and data parsing. Additional connectivity via USB and SPI is possible, making it highly adaptable.

Use in Project

In our system, the NEO-6M is connected to the Raspberry Pi 4 via UART at 3.3V logic level. It delivers:

- Real-time location data (latitude, longitude, altitude)
- Velocity and heading
- Coordinated Universal Time (UTC) synchronization

Its excellent **signal sensitivity (–160 dBm)** and built-in support for **SuperSense™ Indoor GPS** make it ideal for both outdoor and semi-indoor positioning tasks. The module is kept isolated from switching regulators to reduce EMI noise and improve satellite lock reliability.

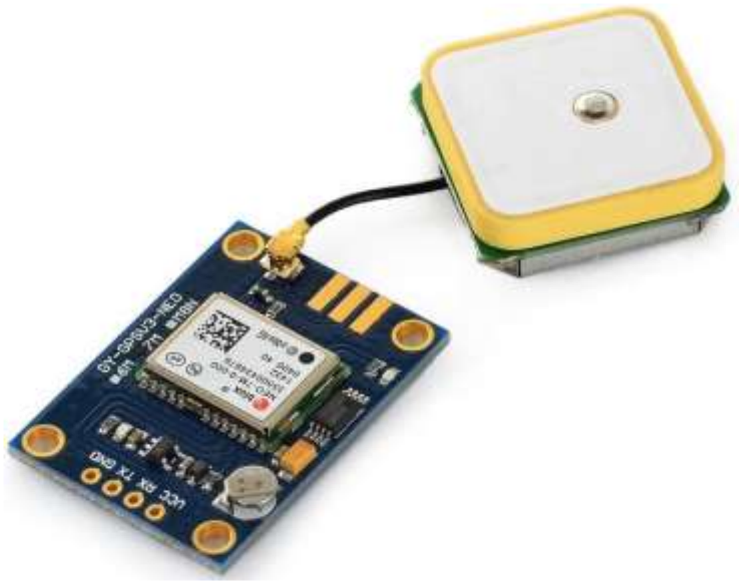


Figure 5.5- NEO-6M GPS Module

5.3 VOLTAGE AND CURRENT NEEDS OF EACH COMPONENT

To ensure proper operation and electrical stability, each component in the system was carefully selected based on its voltage and current requirements. Understanding the electrical characteristics of every module was essential for designing a safe and efficient power architecture—especially when powered by a single external source, the Redmi 10000mAh Power Bank.

The following table summarizes the operating voltage, typical current, peak current, and important design notes for each component used in the system:

Component	Operating Voltage	Typical Current	Peak Current	Notes
Raspberry Pi 4	5.0 V	~0.7 – 1.0 A	~2.5 A	Depends on USB devices, processing load, and startup
Camera Module v2	5.0 V (via Pi)	~250 mA	~300 mA	Draws power through CSI connector on Pi
DHT22 Sensor	5.0 V	~1.5 mA	~2 mA	Digital signal; powered directly from 5V rail
NEO-6M GPS Module	5.0 V	~30–39 mA	~45 mA	Requires clean, stable 5V supply for GPS lock

Table5.1- Power of Components

Power Routing Summary:

- 5V Rail: Supplied directly from the power bank to the Raspberry Pi 4, Camera Module, DHT22, and GPS Module.
- Capacitive Filtering: Bypass capacitors were placed near sensitive modules to minimize voltage ripple and ensure clean signal delivery.

All components were selected to remain within the 2.6A current capacity of the power bank. The unified 5V architecture simplified the power distribution design and ensured stable operation even during current spikes such as Raspberry Pi boot or GPS lock acquisition**.**

5.4 SOURCE OF POWER CHOSEN AND WHY

The power source chosen for this project is the Redmi 10000mAh Power Bank, a compact and rechargeable lithium-polymer power supply designed for consumer electronics. It was selected based on a balance of availability, output current capacity, protection features, and portability—making it a practical solution for ground-based prototyping and demonstration of embedded systems.

Key Advantages

1. Sufficient Output Capacity
   The power bank provides a 5.1V output at up to 2.6A, sufficient to power the Raspberry Pi 4, Camera Module, and all 5V peripherals.

2. Stable Voltage Delivery
   The output voltage remains consistently within the 5V ± 0.1V range, suitable for direct input to all components.

3. Integrated Safety Features
   Built-in protection against overvoltage, overcurrent, short-circuiting, and over-discharge adds system reliability.

4. Rechargeability and Portability
   With a 10000mAh (37Wh) capacity, the power bank supports several hours of continuous operation in mobile environments.

Implementation Details The power bank was connected directly to the Raspberry Pi via USB, and from there, it powered all modules through the same 5V line. Since all components were 5V-tolerant, no voltage regulation or level shifting was necessary.

While suitable for development and testing, this power source is not appropriate for use in space due to the lack of radiation and thermal protection.

In summary, the Redmi 10000mAh Power Bank provided a cost-effective and convenient power solution that supported all operational requirements during prototyping without compromising safety or performance.

Figure 5.6- Source of Power

5.5 POWER DISTRIBUTION TOPOLOGY

To ensure efficient and stable energy delivery, a **parallel power distribution topology** was implemented in the system. In this topology, all components are connected to a common 5V power rail, drawing only the current they require while maintaining a constant voltage across the system.

Why Parallel Distribution?

- **Voltage Consistency:** All components operate at the same voltage level, which simplifies routing and avoids the need for complex regulation.
- **Independent Module Operation:** Each module functions independently, so if one fails or draws excess current, others are not affected.
- **Simplicity:** The single-voltage system (5V) reduces complexity in both circuit design and PCB layout.

Unified 5V Power Rail All system modules—including the Raspberry Pi 4, Camera Module, DHT22 sensor, and NEO-6M GPS—operate at 5V and are powered directly from the Redmi power bank without the use of any voltage regulators or converters. This unified approach ensured that:

- No voltage translation or level shifting was required.
- Power delivery was centralized and easy to debug.
- The current demands of the modules stayed within the 2.6A capacity of the power source.

Filtering and Layout Considerations

- **Decoupling Capacitors:** Placed near each module's power pin to reduce voltage ripple and suppress transient noise.
- **Power Trace Sizing:** Wider traces were used on the single-layer PCB for the 5V and GND rails to minimize resistance and prevent overheating.
- **Electromagnetic Compatibility:** Routing was planned to reduce noise coupling, especially around sensitive components like the GPS module.

  This clean and direct power distribution strategy proved both effective and efficient, maintaining stable system operation under varying load conditions during data capture, processing, and wireless communication

5.6 PCB DESIGN PROCESS

5.6.1 DESIGN SOFTWARE USED (DIPTRACE, PROTEUS)

Two primary tools were used throughout the design process: **DipTrace** and **Proteus 8 Professional**.

- **DipTrace** was the primary PCB design software used for schematic capture, component placement, electrical rule definition, trace routing, and the generation of industry-standard manufacturing files. It offered an intuitive environment for working with both single- and dual-layer boards and allowed precise configuration of via sizes, trace widths, and net properties. The ability to switch between 2D and 3D views helped visualize component alignment and mechanical constraints.

- **Proteus** was used primarily for early-stage schematic prototyping. While its simulation capability was not fully utilized in this project, it served as a valuable visual tool to understand how modules such as the Raspberry Pi, DHT22, and GPS unit should be connected. This helped confirm pin mapping and ensure all interconnections matched the physical layout.

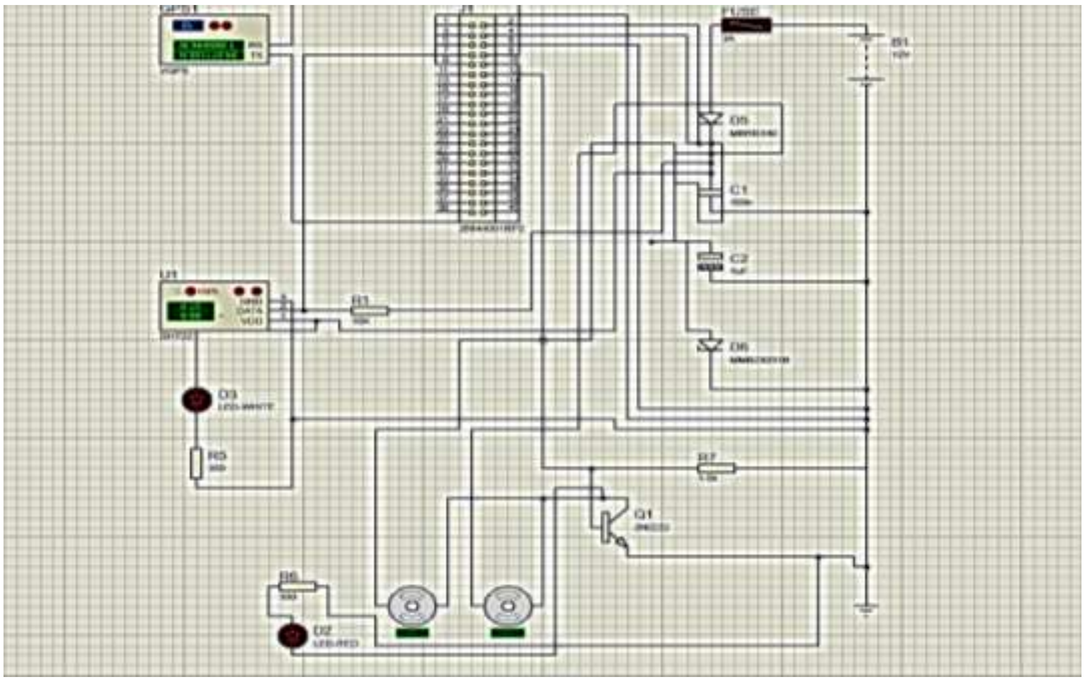


Figure 5.7- Proteus Design

5.6.2 SCHEMATIC DESIGN

The schematic represented the logical layout of the system. It included:

- The **Raspberry Pi GPIO header** (40-pin), defining each signal connection
- External connectors for camera and GPS module
- Power input from USB header and fuse protection
- Passive components like resistors, capacitors, and LEDs

The design process involved defining signal nets, verifying continuity between modules, and labeling key buses (e.g., power, ground, data lines). DipTrace's annotation and net check features helped ensure correctness before layout — especially important when working within the routing constraints of a single-layer board.

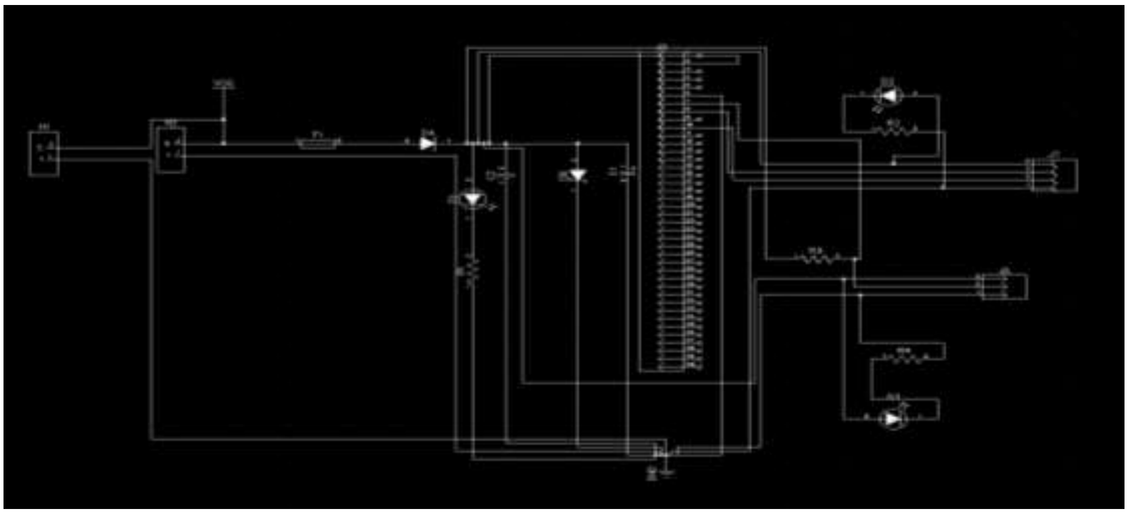


Figure 5.8- Schematic Design

5.6.3 PCB LAYOUT – SINGLE LAYER DESIGN

After the schematic was finalized, the board layout was created with component arrangement influenced by:
• Current paths: high-current lines were kept short and wide to reduce voltage drops and improve efficiency.
• Thermal separation: heat-generating components were spaced away from sensitive modules to minimize thermal interference.
• Signal integrity: analog and digital traces were routed separately where possible to reduce noise and interference.

In the final implementation, a single-layer PCB design was used instead of a dual-layer board. This imposed routing constraints but was suitable for the relatively simple interconnects required by the system.

- The Top Layer hosted all components, headers, connectors, and power traces.
- Since only one copper layer was used, routing was carefully optimized to avoid trace crossing, using strategic placement of components and right-angle trace bends where necessary.

While no bottom copper layer or plated vias were used, manual jumper wires were applied where required to resolve unavoidable crossovers. Pads were optimized for hand soldering, and thermal reliefs were used for large copper areas to facilitate heat transfer during reflow**.**

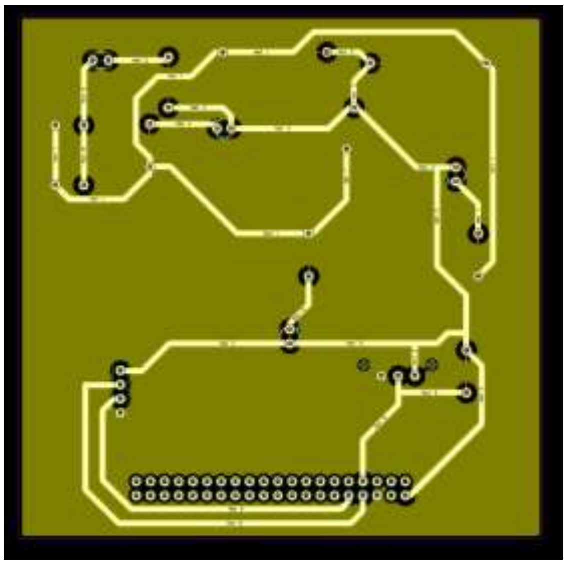


Figure 5.9- PCB Design

5.6.4 DESIGN RULES APPLIED

The board was designed with standard mid-range fabrication tolerances, suitable for single-layer, through-hole-based PCB manufacturing:
• Minimum clearance between traces and pads: 0.4 mm
• Minimum trace width: 0.4 mm (suitable for up to ~1 A current carrying)
• Via diameter: N/A (no plated through-holes were used in the final single-layer design)
• Jumper wire crossings were used where inter-layer connections would otherwise be required
• Board dimensions: 94 mm × 94 mm square (optimized for modular mounting)

All rules were defined in the DipTrace DRC panel and verified continuously during routing. No violations were reported in the final check

5.6.5 MANUFACTURING PREPARATION

To ensure smooth transition from design to physical production, a comprehensive set of deliverables was prepared using DipTrace's manufacturing export tools. Each file was carefully configured according to industry-standard formats accepted by most PCB manufacturers.

The final output package included the following:
• **Gerber Files:** Generated for Top Copper, Top Silkscreen, Top Solder Mask, and Board Outline only. Since the board was designed as single-layer, no Bottom Copper layer was included.
• **NC Drill File:** Specifies all non-plated through-holes (for mounting and components). No plated vias were included, as the design avoided inter-layer routing.
• **Bill of Materials (BOM):** Lists all electronic components used in the design, including reference designators, quantities, footprints, and manufacturer part numbers to assist in ordering and assembly.
• **Pick-and-Place File (Optional):** Not required for this design, as manual through-hole

soldering was planned for all components.
• **Schematic and PCB Printouts:** PDF copies of the full schematic and PCB layout for human inspection, debugging, and referencing during assembly.

A final Design Rule Check (DRC) was executed to validate all spacing, clearances, trace widths, and hole sizes. Once confirmed, the project files were zipped into a single manufacturing package and sent to the chosen PCB fabrication house.

This preparation ensured the board could be manufactured without delays or revision cycles, and that assembly technicians could interpret the design easily even in a single-layer setup.

5.7 COMPARATIVE ANALYSIS

5.7.1 ACTUAL PROTOTYPE DESIGN

The actual system implemented in this project served as a simplified prototype of a CubeSat's power subsystem. It aimed to validate the integration of essential modules and demonstrate stable power delivery in a compact, PCB-based embedded system. The prototype relied on a 5V power supply from a Redmi 10000mAh USB power bank, delivering energy to the Raspberry Pi 4, DHT22 sensor, NEO-6M GPS module, and Raspberry Pi Camera Module v2.

All components operated at 5V, and no voltage regulation was required. The PCB was designed as a single-layer FR4 board with appropriate trace width, spacing, and basic protection features such as fuses and decoupling capacitors. While the prototype was not suitable for orbital deployment, it successfully replicated core architectural concepts in a ground-based testing environment.

5.7.2 PLANNED CUBESAT POWER ARCHITECTURE

The ideal CubeSat power architecture envisioned for this project was more advanced, supporting long-term, autonomous operation in space. It was to be centered around solar energy harvesting, rechargeable lithium-ion batteries, and a smart Power Management Unit (PMU) capable of regulating and distributing multiple voltage rails. This design would include advanced monitoring, environmental protection, and redundancy.

Subsystem	Planned Component	Purpose
Power Management Unit	BQ25798 or equivalent	Smart regulation and power path control
Battery Pack	3× 18650 Li-ion Cells	Energy storage for continuous operation
Energy Generation	Body-mounted Solar Panels	Converts solar energy to electrical power
Voltage Regulation	Buck/Boost DC-DC Converters	Provides regulated outputs to match subsystem needs

Monitoring & Protection	INA219 Sensors, TVS, Fuses	Real-time monitoring and fault protection
Power Distribution Board	Custom Multi-Rail PCB	Routes 3.3V, 5V, and 12V to satellite subsystems

Table5.2- Architecture of Power Subsystem

5.7.3 DESIGN COMPARISON TABLE

Aspect	Prototype (Implemented Design)	Ideal CubeSat Power System
Power Source	Commercial USB power bank (Redmi 10000mAh, 5.1V⎓2.6A)	Solar panels with MPPT and rechargeable battery pack (Li-ion)
Energy Storage	No energy storage (direct power only)	3× 18650 Li-ion cells with Battery Management System (BMS)
Voltage Regulation	Fixed 5V only, no internal conversion	Smart PMU generating 3.3V, 5V, and 12V rails
Power Monitoring	Not available	INA219 current sensors, voltage monitors for real-time feedback
Protection Mechanisms	Basic fuse and decoupling capacitors	TVS diodes, current limiting, reverse polarity, redundant paths
PCB Architecture	Single-layer FR4 board, standard clearance and trace sizes	4–6 layer, radiation-hardened PCB with EMI/thermal optimization
Telemetry & Logging	No onboard telemetry, no power data logging	Full telemetry system logging voltage/current over time
Environmental Tolerance	Indoor/ground-based operation only	Designed for vacuum, radiation, thermal cycling, and vibration
System Autonomy	Limited to power bank duration, manual operation	Autonomous charging, regulation, and fault detection
Cost & Complexity	Low-cost, student-friendly, focused on proof-of-concept	High-cost, qualified components for long-duration space missions

Table5.3- Comparison Table

5.7.4 TRADE-OFFS AND LIMITATIONS

This project focused on rapid prototyping, educational value, and functional demonstration over space-grade performance. The absence of solar energy input, voltage regulation, and monitoring systems reflects trade-offs in complexity, budget, and fabrication scope.

Key limitations included:

- Lack of energy autonomy due to reliance on a fixed power bank
- No regulated voltage rails beyond 5V, limiting subsystem diversity
- Absence of telemetry, power diagnostics, or real-time protection feedback
- No mechanical or thermal protection against space environments

Despite these constraints, the system effectively demonstrated modular integration, power delivery, and PCB manufacturing workflows using a single-layer board. It provided critical hands-on experience that forms the basis for future upgrades toward space-qualified CubeSat power systems.

5.8 conclusion

The successful design and implementation of the power system and PCB layout for this CubeSat-inspired prototype represent a significant milestone in the development of embedded systems with aerospace applications. Through a systematic approach, we were able to bridge theoretical knowledge with practical execution, tackling real engineering constraints and translating conceptual subsystems into working hardware.

This project began with a comprehensive understanding of the importance of power in space systems—where the availability, distribution, and regulation of energy directly influence mission success. CubeSats, due to their size and cost limitations, require particularly efficient and compact power systems. Although our prototype was not intended for flight, it emulated key power architecture principles that align with actual small satellite missions.

From a hardware perspective, we selected a Raspberry Pi 4 as the central controller due to its versatility and processing capabilities. It was interfaced with key modules such as a camera for visual monitoring, a DHT22 sensor for environmental measurement, and a NEO-6M GPS module for localization. These modules were chosen for their availability, simplicity, and relevance to remote sensing and CubeSat-like operations. The power for the entire system was provided through a commercial 5V 10000mAh power bank, enabling portability and simplicity during ground-based testing.

On the PCB design side, we used DipTrace to develop a compact and efficient single-layer layout that prioritized straightforward power distribution, low-noise routing, and accessibility. We followed strict design rules, including spacing, clearance, and hole dimension constraints, to ensure manufacturability. Despite using a single-voltage 5V system without regulation, the design proved robust and capable of powering all modules reliably.

Our comparative analysis highlighted the trade-offs between the implemented system and a fully space-ready CubeSat design. While our prototype lacked features such as autonomous power generation (solar panels), multi-voltage regulation, and telemetry, it allowed us to simulate critical aspects of system integration and modular expansion. The simplified design removed unnecessary complexity and provided clarity during testing, which is essential for early development phases.

One of the most important outcomes of this project was the deep understanding gained in translating design requirements into physical implementations. From schematic capture to PCB layout and power routing, every step presented learning opportunities. By building the system ourselves, we encountered the same challenges that real CubeSat developers face—albeit in a less harsh environment—and this experience is invaluable for future iterations.

This work lays the foundation for scaling toward a more advanced CubeSat platform. Future improvements could include integrating solar energy harvesting circuits, implementing dynamic voltage regulation, adding current monitoring sensors, and transitioning to microcontroller-based systems for reduced power consumption. Additionally, the mechanical structure, environmental shielding, and radiation tolerance must be addressed in later designs to meet the demands of actual space deployment.

In conclusion, this project is not only a demonstration of functional hardware and PCB development but also a stepping stone toward a full-featured, space-compliant CubeSat subsystem. It showcases how practical engineering, combined with academic understanding, can lead to meaningful innovations in aerospace technology, even from a ground-based prototype

Chapter (6) :**Interfacing**

6.1 INTRODUCTION

This chapter presents the integration of a sensor-powered environmental data collection system with an AI-based fish classification model and a localized GUI mapping interface for offline visualization. The primary objective of this implementation is to support sustainable marine monitoring by identifying potential fish habitats based on real-time estimates of sea surface temperature (SST) and chlorophyll-a concentration.

Environmental data is acquired using a Raspberry Pi connected to a suite of sensors, including a GPS module (to capture geospatial coordinates), a DHT22 sensor (to measure ambient air temperature, which is used to estimate SST), and a camera module (to calculate NDVI and derive chlorophyll-a values). These parameters are processed in real time and passed into an embedded AI algorithm, which classifies the most likely fish species present in each location and estimates their potential abundance. The output is stored as a structured CSV file for further analysis.

Initially, fish distribution was visualized using ArcGIS to display the spatial data. However, this method required paid server hosting and was not suitable for real-time or offline usage. As an alternative, a local graphical interface was developed using Python libraries such as Tkinter and Folium. This GUI enables users to load the CSV data and select a specific date, generating an interactive map displaying fish locations, SST, chlorophyll levels, and predicted fish types.

This chapter outlines the complete interfacing pipeline from sensor data acquisition and AI inference to localized visualization offering a reliable, cost-effective, and fully offline system for marine habitat assessment and decision-making.

6.2 SENSOR-AI INTERFACE FOR FISH DETECTION USING RASPBERRY PI

Background

This section of the chapter focuses on the interface between raw sensor data and the AI-based fish classification logic. It demonstrates how real-time environmental inputs are transformed into actionable insights using a rule-based AI model, which relies on predefined logic rather than machine learning.

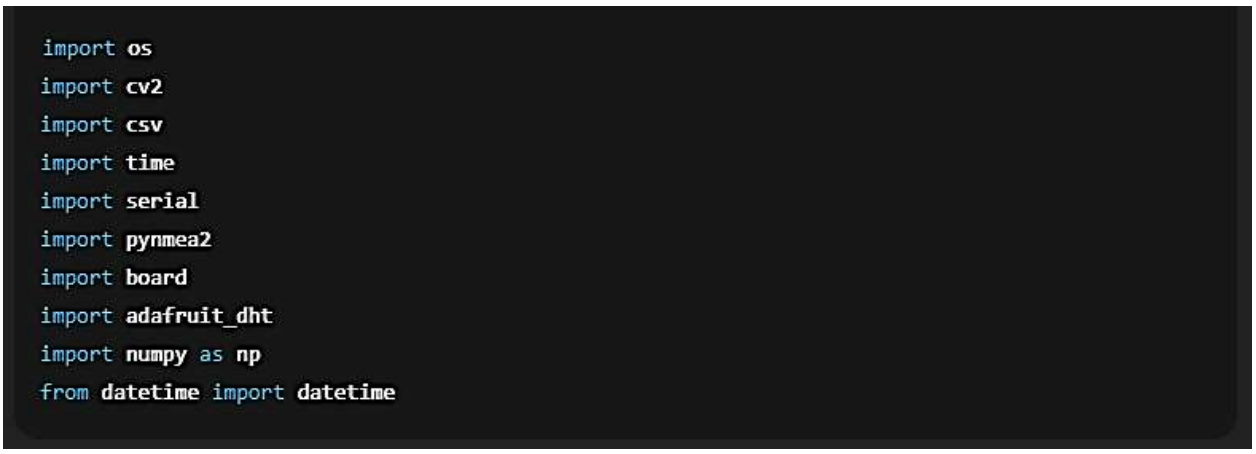
To support marine ecosystem monitoring and deepen our understanding of fish populations, this Raspberry Pi–based system was developed to collect and process environmental data in the field. **The system integrates:**

- Air temperature, collected via the DHT22 sensor

- Geographic coordinates, gathered from a GPS module
- Water surface images, captured using the Raspberry Pi Camera
- These inputs feed into a lightweight rule-based model that estimates:
- Sea Surface Temperature (SST)
- Chlorophyll-a concentration (Chl-a)

The model then predicts the most likely fish type and provides an estimated quantity based on the environmental conditions.

6.2.1 IMPORTS & INITIALIZATION



```python
import os
import cv2
import csv
import time
import serial
import pynmea2
import board
import adafruit_dht
import numpy as np
from datetime import datetime
```

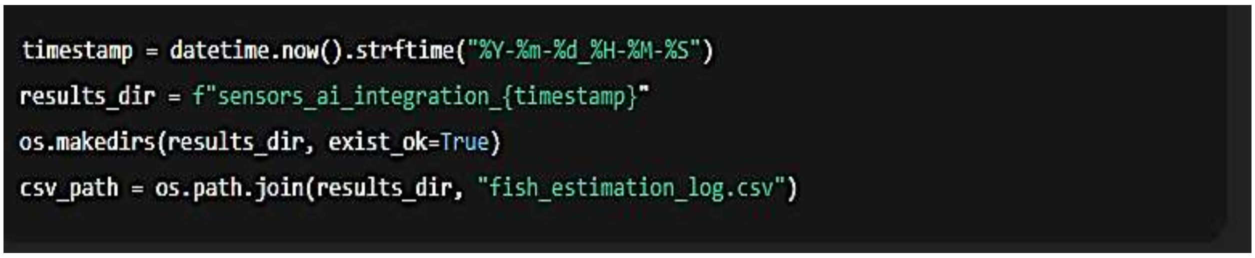
Figure 6.1: Imports & Initialization

Purpose:

Load libraries needed for:

- Sensor reading (DHT22, GPS)
- Image capture and processing
- File and time management
- Mathematical calculations

6.2.2 SESSION SETUP



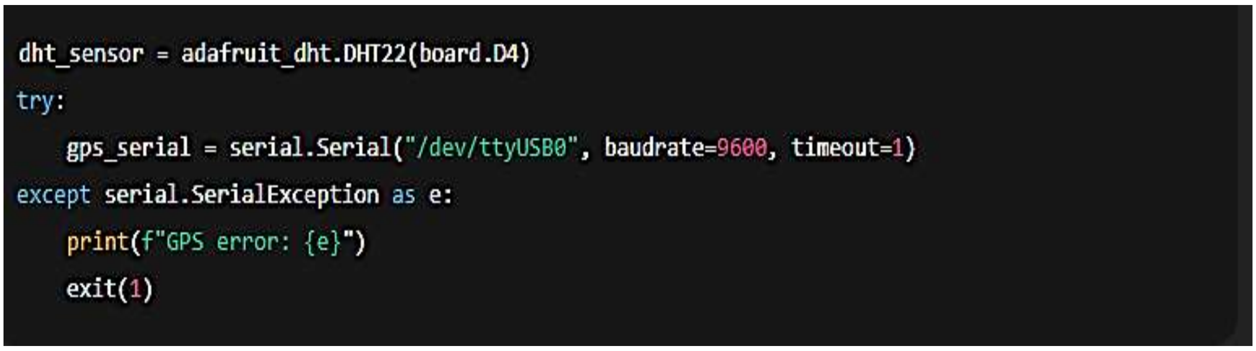
```
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"sensors_ai_integration_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "fish_estimation_log.csv")
```

Figure 6.2: Session Setup

Purpose:

- Generate a folder named with the current timestamp
- Set up the path for saving the CSV file
- Organize session output in a dedicated results directory

6.2.3 SENSOR INITIALIZATION



```
dht_sensor = adafruit_dht.DHT22(board.D4)
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"GPS error: {e}")
    exit(1)
```

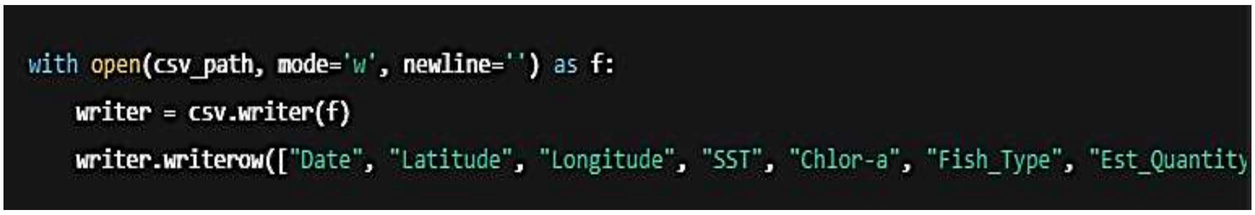
Figure 6.3: Sensor Initialization

Purpose:

- Activate the DHT22 sensor on GPIO pin D4
- Establish a serial connection with the GPS module

If GPS initialization fails, the program exits safely with an error message.

6.2.4 CSV HEADER INITIALIZATION



```
with open(csv_path, mode='w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Date", "Latitude", "Longitude", "SST", "Chlor-a", "Fish_Type", "Est_Quantity
```

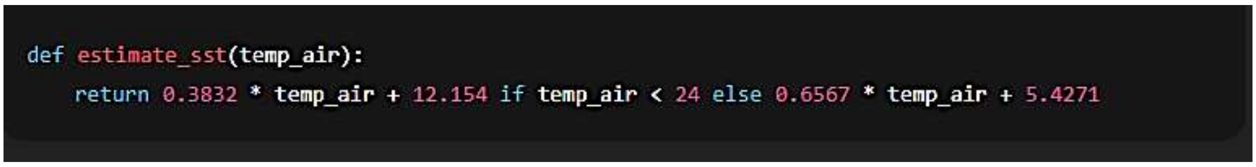
Figure 6.4: CSV Header Initialization Purpose:

Create a structured CSV file with columns for:

- Date and time
- Latitude and longitude
- Estimated SST and Chlorophyll-a
- Predicted fish type
- Estimated fish quantity

6.2.5 HELPER FUNCTIONS

6.2.5.1 SST ESTIMATION

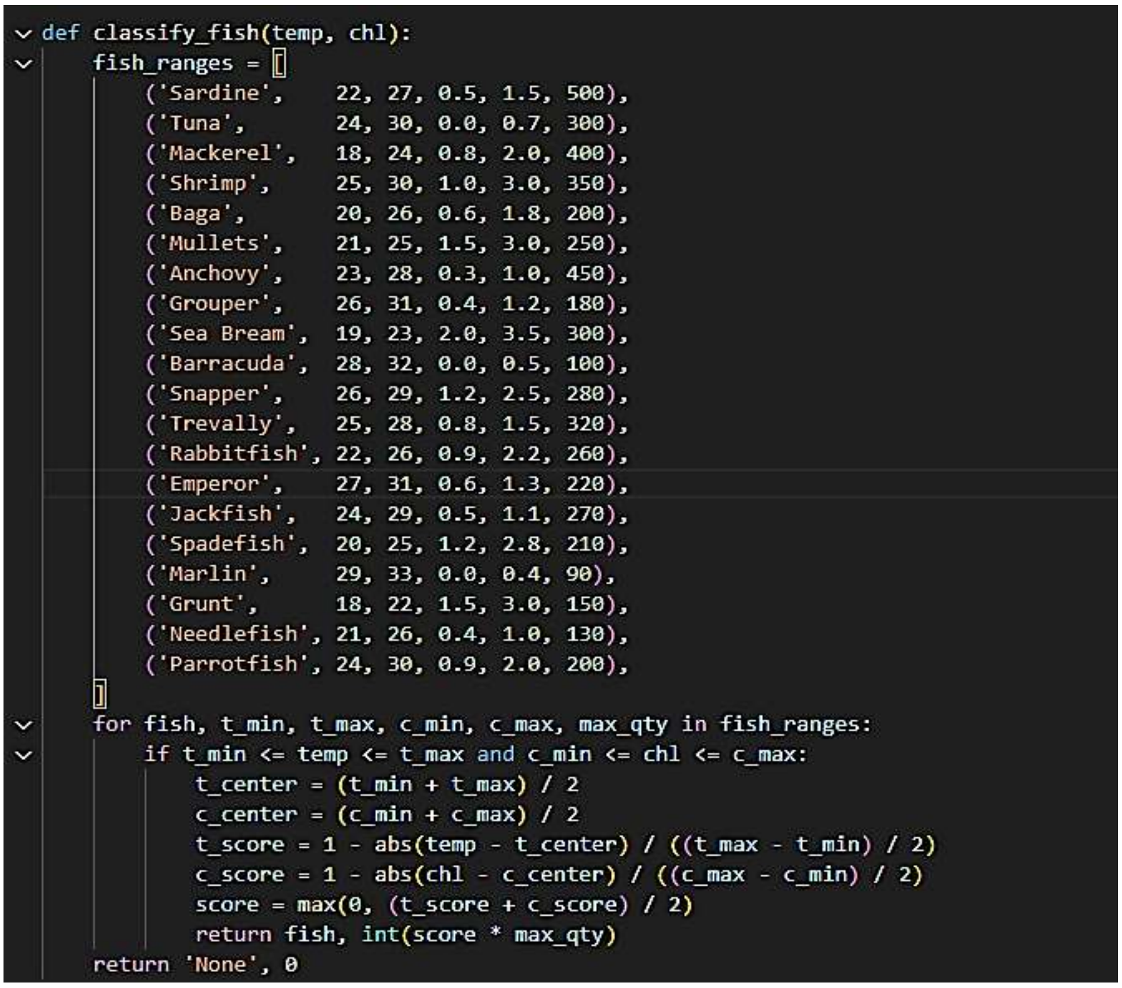


```
def estimate_sst(temp_air):
    return 0.3832 * temp_air + 12.154 if temp_air < 24 else 0.6567 * temp_air + 5.4271
```

Figure 6.5: SST Estimation

- Uses empirical formulas to estimate **sea surface temperature (SST)** from **air temperature**
- Applies different equations for air temperatures below or above 24°C

6.2.5.2 FISH CLASSIFICATION

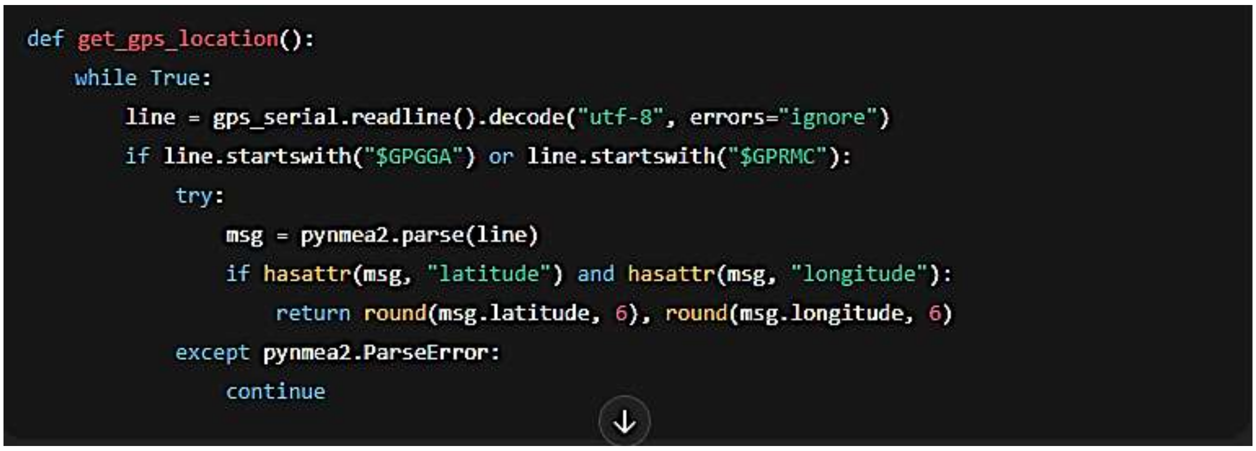


```
def classify_fish(temp, chl):
    fish_ranges = [
        ('Sardine',    22, 27, 0.5, 1.5, 500),
        ('Tuna',       24, 30, 0.0, 0.7, 300),
        ('Mackerel',   18, 24, 0.8, 2.0, 400),
        ('Shrimp',     25, 30, 1.0, 3.0, 350),
        ('Baga',       20, 26, 0.6, 1.8, 200),
        ('Mullets',    21, 25, 1.5, 3.0, 250),
        ('Anchovy',    23, 28, 0.3, 1.0, 450),
        ('Grouper',    26, 31, 0.4, 1.2, 180),
        ('Sea Bream',  19, 23, 2.0, 3.5, 300),
        ('Barracuda',  28, 32, 0.0, 0.5, 100),
        ('Snapper',    26, 29, 1.2, 2.5, 280),
        ('Trevally',   25, 28, 0.8, 1.5, 320),
        ('Rabbitfish', 22, 26, 0.9, 2.2, 260),
        ('Emperor',    27, 31, 0.6, 1.3, 220),
        ('Jackfish',   24, 29, 0.5, 1.1, 270),
        ('Spadefish',  20, 25, 1.2, 2.8, 210),
        ('Marlin',     29, 33, 0.0, 0.4, 90),
        ('Grunt',      18, 22, 1.5, 3.0, 150),
        ('Needlefish', 21, 26, 0.4, 1.0, 130),
        ('Parrotfish', 24, 30, 0.9, 2.0, 200),
    ]
    for fish, t_min, t_max, c_min, c_max, max_qty in fish_ranges:
        if t_min <= temp <= t_max and c_min <= chl <= c_max:
            t_center = (t_min + t_max) / 2
            c_center = (c_min + c_max) / 2
            t_score = 1 - abs(temp - t_center) / ((t_max - t_min) / 2)
            c_score = 1 - abs(chl - c_center) / ((c_max - c_min) / 2)
            score = max(0, (t_score + c_score) / 2)
            return fish, int(score * max_qty)
    return 'None', 0
```

Figure 6.6: Fish Classification

- Applies rule-based logic to match SST and Chl-a values with predefined fish habitat ranges
- Computes a matching score and scales it to an estimated quantity
- Returns both the fish type and its expected abundance

6.2.5.3 GPS READING

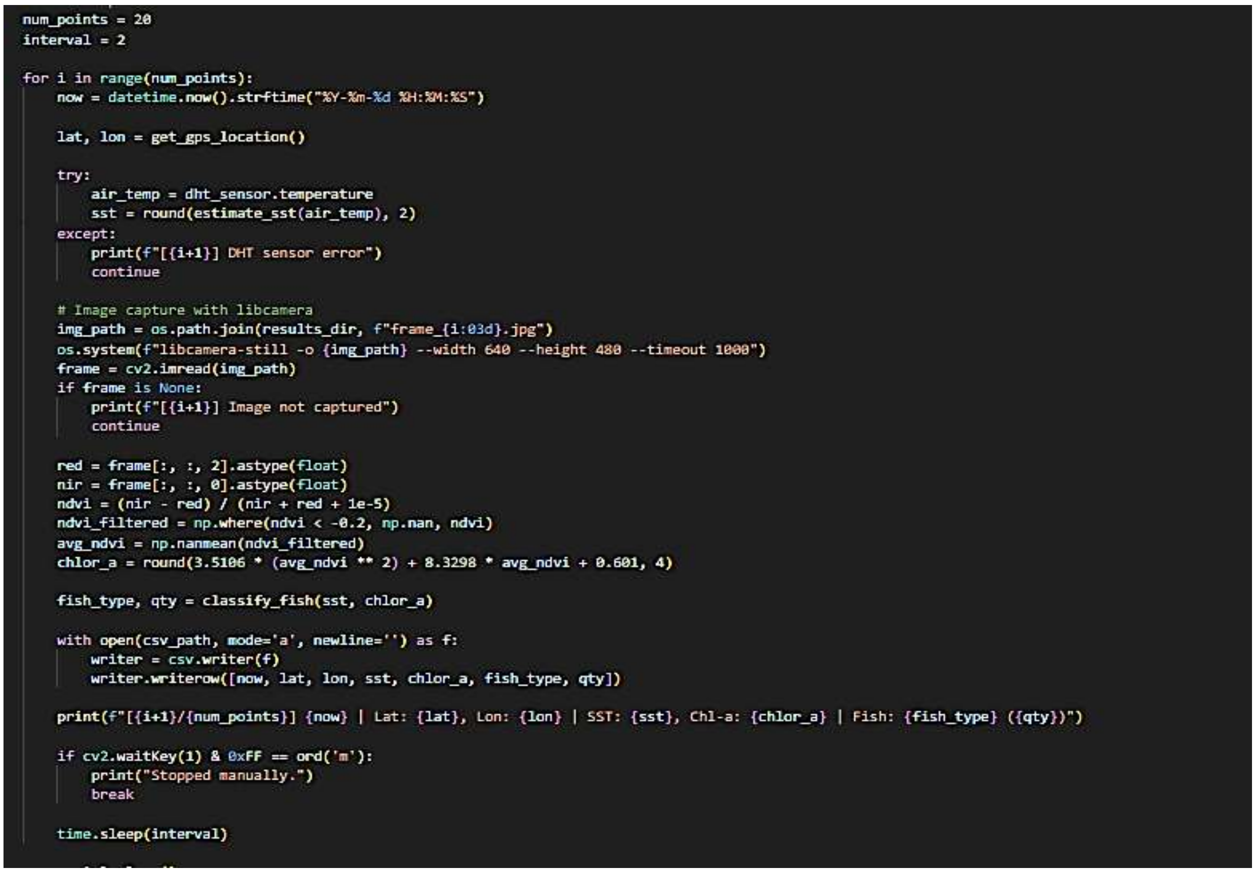


```python
def get_gps_location():
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")
        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    return round(msg.latitude, 6), round(msg.longitude, 6)
            except pynmea2.ParseError:
                continue
```

Figure 6.7: GPS Reading

- Continuously listens to the GPS serial port
- Extracts valid **latitude and longitude** from NMEA sentences like $GPGGA and $GPRMC

6.2.6 MAIN DATA COLLECTION LOOP



```python
num_points = 20
interval = 2

for i in range(num_points):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    lat, lon = get_gps_location()

    try:
        air_temp = dht_sensor.temperature
        sst = round(estimate_sst(air_temp), 2)
    except:
        print(f"[{i+1}] DHT sensor error")
        continue

    # Image capture with libcamera
    img_path = os.path.join(results_dir, f"frame_{i:03d}.jpg")
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")
    frame = cv2.imread(img_path)
    if frame is None:
        print(f"[{i+1}] Image not captured")
        continue

    red = frame[:, :, 2].astype(float)
    nir = frame[:, :, 0].astype(float)
    ndvi = (nir - red) / (nir + red + 1e-5)
    ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
    avg_ndvi = np.nanmean(ndvi_filtered)
    chlor_a = round(3.5106 * (avg_ndvi ** 2) + 8.3298 * avg_ndvi + 0.601, 4)

    fish_type, qty = classify_fish(sst, chlor_a)

    with open(csv_path, mode='a', newline='') as f:
        writer = csv.writer(f)
        writer.writerow([now, lat, lon, sst, chlor_a, fish_type, qty])

    print(f"[{i+1}/{num_points}] {now} | Lat: {lat}, Lon: {lon} | SST: {sst}, Chl-a: {chlor_a} | Fish: {fish_type} ({qty})")

    if cv2.waitKey(1) & 0xFF == ord('m'):
        print("Stopped manually.")
        break

    time.sleep(interval)
```

Figure 6.8: Main Data Collection Loop

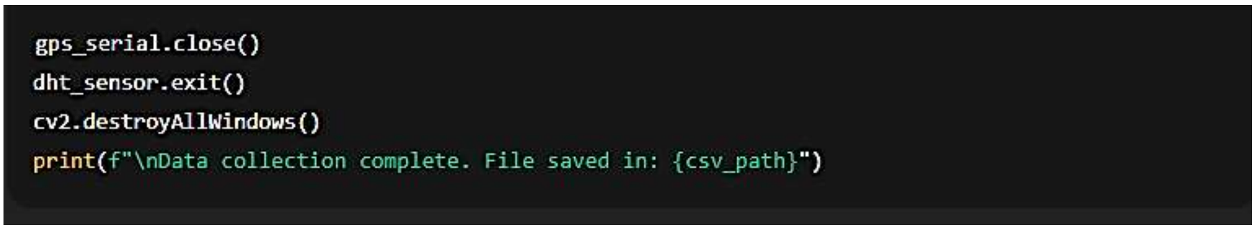
Steps per Iteration:

1. Get the current timestamp

2. Retrieve GPS coordinates
3. Read air temperature from the DHT22 sensor
4. Estimate SST using the helper function
5. Capture an image using the Pi Camera
6. Extract the RED and simulated NIR channels
7. Calculate NDVI using the formula:
8. NDVI = (NIR - RED) / (NIR + RED)
9. Convert NDVI into Chlorophyll-a concentration:
10. Chl-a = 3.5106 * NDVI^2 + 8.3298 * NDVI + 0.601
11. Pass SST and Chl-a to the fish classification logic
12. Save results to the CSV file
13. Print a summary to the terminal

The loop stops automatically after a defined number of iterations, or manually if the user presses the m key.

6.2.7 CLEANUP



```
gps_serial.close()
dht_sensor.exit()
cv2.destroyAllWindows()
print(f"\nData collection complete. File saved in: {csv_path}")
```

Figure 6.9: Cleanup

Purpose:

- Close the GPS connection
- Properly release the DHT22 sensor
- Shut down OpenCV resources
- End the program safely

Sample Terminal Output

[7/20] 2025-07-07 02:14:32 | Lat: 28.123456, Lon: 30.987654 | SST: 26.7, Chl-a: 1.83 | Fish: Tuna (165)

6.2.8 SUMMARY

The system acts as a smart interface connecting real-world sensors to logic-based environmental analysis and actionable outputs:

Input Sensors	→	Data Analysis	→	Rule-Based Classification	→	Output Logging
DHT22 + GPS + Camera	→	SST + Chlorophyll-a	→	Fish Type + Quantity	→	CSV File + Terminal View

6.3 PYTHON-BASED GUI FOR LOCAL VISUALIZATION

6.3.1 INITIAL ATTEMPT USING ARCGIS

To interpret and analyze the spatial characteristics of the system's output, ArcGIS Pro was initially adopted as a visualization tool. The purpose of this approach was to display the geospatial distribution of fish detection results generated through the interfacing between environmental sensors and the AI-based classification model. This interfacing process produced structured data logs containing both raw environmental variables and algorithmically predicted fish presence at specific locations.

Environmental data was acquired through a combination of hardware modules including a GPS receiver, a DHT22 temperature sensor, and a camera for capturing imagery used in chlorophyll estimation. The AI algorithm received inputs such as estimated sea surface temperature (SST), chlorophyll-a concentration (derived from NDVI), and spatial coordinates. Based on empirical environmental thresholds and fish habitat suitability models, the system classified the most likely fish type and estimated its potential abundance. The final output was stored in a structured CSV file that included fields such as timestamp, latitude, longitude, SST, chlorophyll-a, fish type, and estimated quantity.

To spatially represent this data, the CSV file was imported into ArcGIS Pro using the "XY Table to Point" tool, which geocoded the latitude and longitude columns into spatial point features. Each row in the dataset was transformed into a map marker, allowing for geographic visualization of the classified fish detection points. In addition, ArcGIS's attribute table enabled the display and filtering of key environmental and biological parameters associated with each point, facilitating deeper spatial analysis of the data. This process provided an initial platform for verifying the integrity and distribution of the output data and validating its geographic alignment.

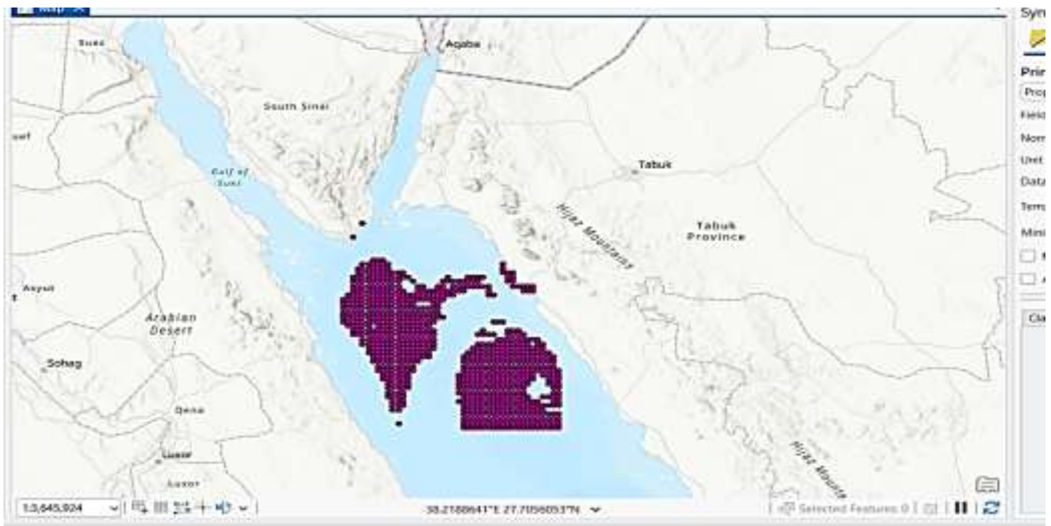
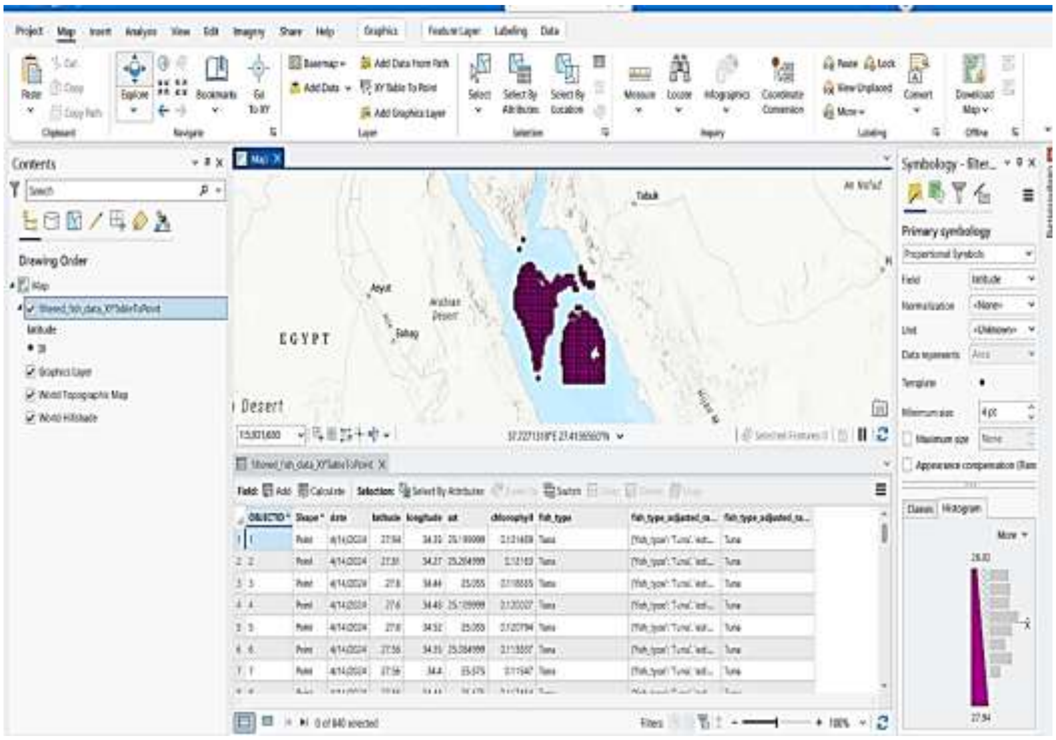


Figure 6.10: Fish detection points based on conditions.

This figure shows the spatial output of the AI-sensor interfacing system. Detection points are plotted on a basemap using geolocation data (latitude and longitude), representing where environmental conditions triggered fish type classification by the system.



OBJECTID *	Shape *	date	latitude	longitude	sst	chlorophyll	fish_type	fish_type_adjusted_ta...	fish_type_adjusted_ta...
1	Point	4/14/2024	27.94	34.33	25.199999	0.121409	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
2	Point	4/14/2024	27.81	34.27	25.284999	0.12183	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
3	Point	4/14/2024	27.8	34.44	25.055	0.118555	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
4	Point	4/14/2024	27.6	34.49	25.109999	0.120027	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
5	Point	4/14/2024	27.8	34.52	25.050	0.120794	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
6	Point	4/14/2024	27.56	34.35	25.284999	0.113557	Tuna	{'fish_type': 'Tuna', 'est...	Tuna
7	Point	4/14/2024	27.56	34.4	25.375	0.11547	Tuna	{'fish_type': 'Tuna', 'est...	Tuna

0 of 840 selected

Figure 6.11: Detection points with SST, chlorophyll, fish type, and quantity

This figure presents the same point layer with the attribute table visible, displaying environmental values such as SST and chlorophyll-a, along with the predicted fish type and estimated quantity for each detection event.

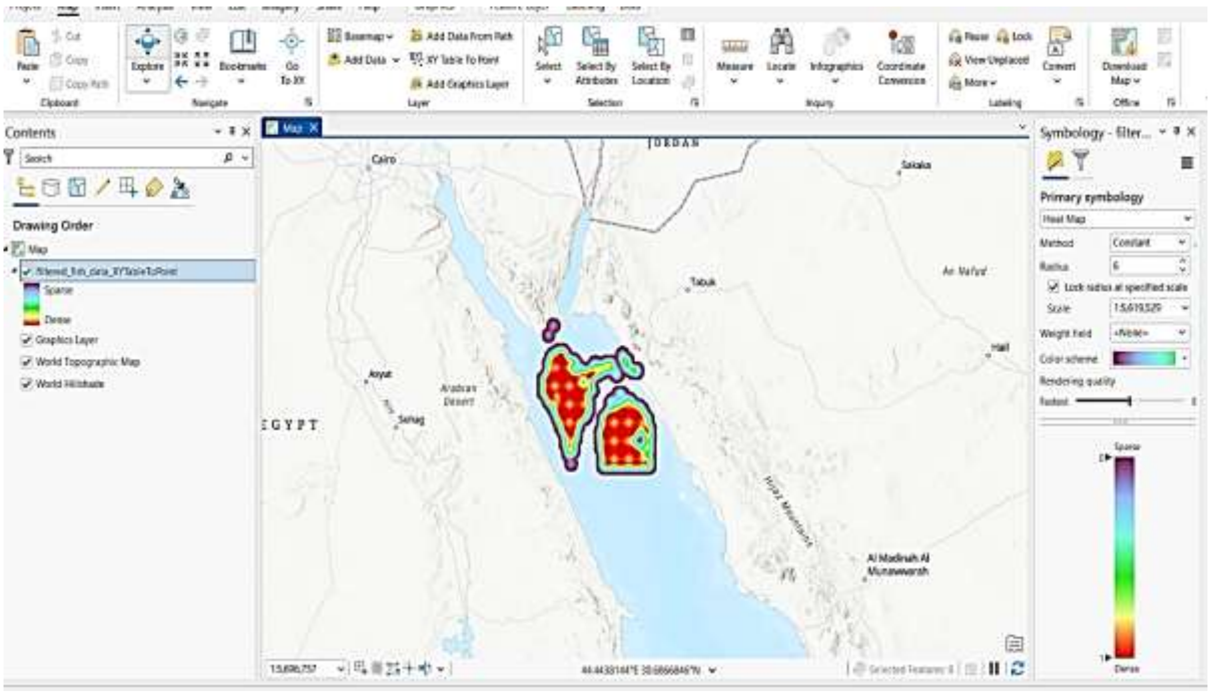


Figure 6.12: Heatmap showing fish detection density; warmer areas indicate higher suitability.

ArcGIS Heat Map Visualization of Detection Density

This heat map highlights the spatial concentration of detected fish locations based on the system's outputs. Warmer colors represent areas of higher detection density, reflecting zones with greater environmental suitability as determined by the embedded classification model.

6.3.2 PYTHON-BASED GUI FOR LOCAL VISUALIZATION

6.3.2.1 INTRODUCTION

After evaluating the ArcGIS platform for visualizing the output of the AI-based fish detection system, it became evident that several limitations including dependency on licensed software, lack of automation, and the requirement for an internet connection to publish interactive content restricted its use for flexible, field-based deployment.

To address these limitations, a lightweight, standalone graphical user interface (GUI) was developed using Python. This interface enables users to interact with the output CSV file generated by the AI-sensor system, filter data by date, and visualize fish detection results on an interactive map without the need for any external software or internet connectivity.

The GUI application is built using a combination of Python libraries:

6.3.2.2 PYTHON LIBRARIES USED

- **Tkinter**
  A standard Python library used to create the graphical user interface (GUI). It was used to design the window layout, date input field, and the button that triggers map generation.
- **Pandas**
  A powerful data analysis library used to load and manipulate the CSV file containing the detection results. It enabled efficient filtering of data by date.
- **Folium**
  A Python wrapper for Leaflet.js, used to generate interactive web maps. It allowed for embedding geolocated detection points on a zoomable, scrollable map rendered in the browser.
- **folium.plugins.MarkerCluster**
  An extension of Folium that groups nearby map markers into clusters. This improved visualization clarity when multiple detections occurred close to one another.
- **webbrowser**
  A built-in Python module used to open the generated HTML map automatically in the default system browser.
- **os**
  A standard Python module used for handling file paths and directories, particularly when saving the HTML output file.
  Used to display pop-up alert messages in the GUI, including error notifications for invalid date input or absence of data.

- **messagebox (from tkinter)** It detects incorrect input, displays an error popup using messagebox.showerror(), and prevents the program from continuing with invalid data.

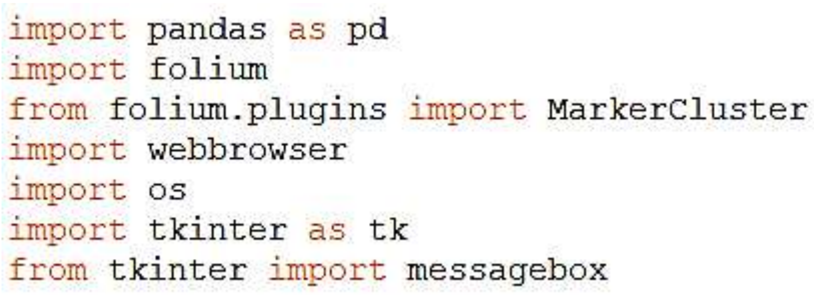
6.3.2.3 CODE IMPLEMENTATION AND EXPLANATION

To implement the GUI, a Python script was developed using the Tkinter, Pandas, and Folium libraries. The code performs three primary functions: filtering data by user-input date, generating an interactive map with detection markers, and displaying the map in the default web browser.

Below is a breakdown of the major components of the code:

6.3.2.3.1 STEP1: IMPORT REQUIRED LIBRARIES

These libraries are used for data manipulation, map rendering, GUI construction, file operations, and displaying user notifications.

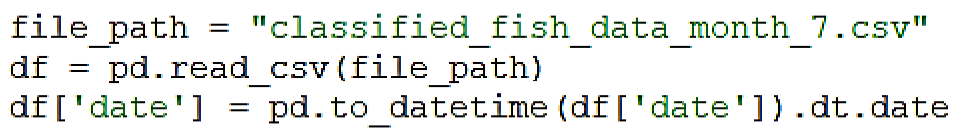


```python
import pandas as pd
import folium
from folium.plugins import MarkerCluster
import webbrowser
import os
import tkinter as tk
from tkinter import messagebox
```

Figure 6.13: import Required Libraries

6.3.2.3.2 LOAD AND PREPARE DATA

The output CSV file from the AI-sensor integration system is read into a Pandas Data Frame. The ‘date’ column is converted to datetime format to enable accurate filtering.

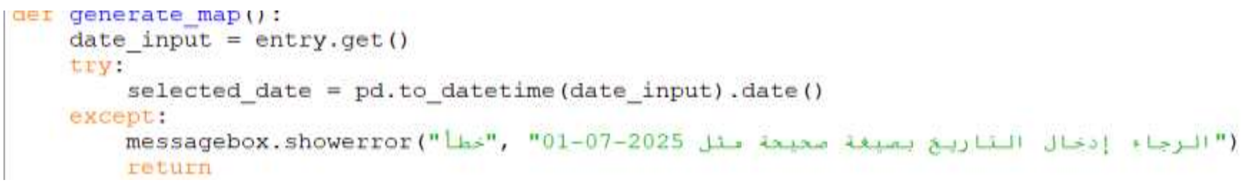


```python
file_path = "classified_fish_data_month_7.csv"
df = pd.read_csv(file_path)
df['date'] = pd.to_datetime(df['date']).dt.date
```

Figure 6.14: Load and Prepare Data

6.3.2.3.3 DEFINE THE MAP GENERATION FUNCTION

This function reads the user’s date input and validates it. If the format is incorrect, an error message is shown.

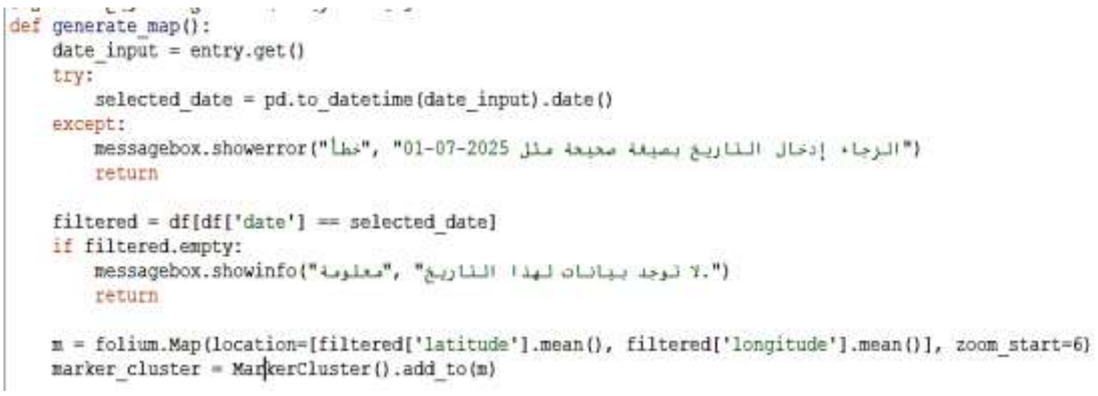


```python
def generate_map():
    date_input = entry.get()
    try:
        selected_date = pd.to_datetime(date_input).date()
    except:
        messagebox.showerror("خطأ", "الرجاء إدخال التاريخ بصيغة صحيحة مثل 01-07-2025")
        return
```

Figure 6.15: Define the Map Generation Function

6.3.2.3.4 FILTER DATA AND CREATE THE MAP

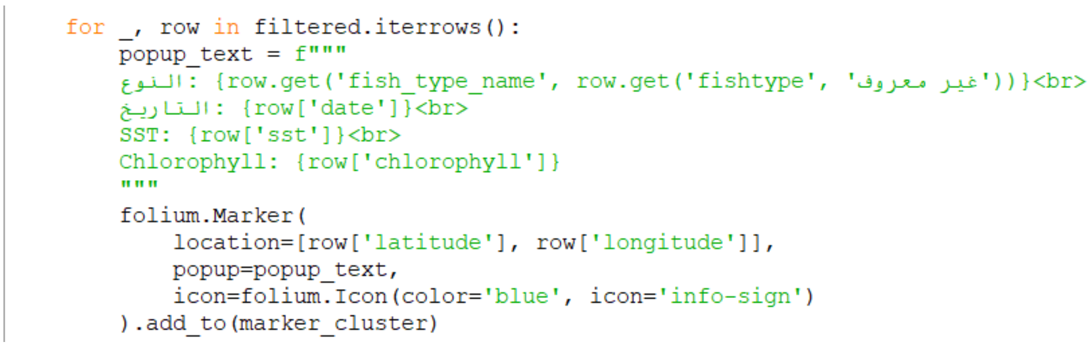
The data is filtered to only include records matching the selected date. A Folium map is then initialized, centered at the average coordinates of the filtered points.



```
def generate_map():
    date_input = entry.get()
    try:
        selected_date = pd.to_datetime(date_input).date()
    except:
        messagebox.showerror("خطأ", "الرجاء إدخال التاريخ بصيغة صحيحة مثل 01-07-2025")
        return

    filtered = df[df['date'] == selected_date]
    if filtered.empty:
        messagebox.showinfo("معلومة", "لا توجد بيانات لهذا التاريخ.")
        return

    m = folium.Map(location=[filtered['latitude'].mean(), filtered['longitude'].mean()], zoom_start=6)
    marker_cluster = MarkerCluster().add_to(m)
```

Figure 6.16: Filter Data and Create the Map

6.3.2.3.5 ADD MARKERS WITH POPUPS

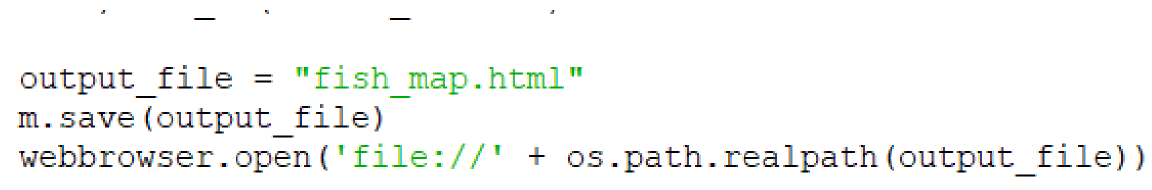


```
    for _, row in filtered.iterrows():
        popup_text = f"""
        النوع: {row.get('fish_type_name', row.get('fishtype', 'غير معروف'))}<br>
        التاريخ: {row['date']}<br>
        SST: {row['sst']}<br>
        Chlorophyll: {row['chlorophyll']}
        """
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=popup_text,
            icon=folium.Icon(color='blue', icon='info-sign')
        ).add_to(marker_cluster)
```

Figure 6.17: Add Markers with Popups

6.3.2.3.6 SAVE AND OPEN THE MAP

The generated map is saved locally as an HTML file and opened automatically in the system's default browser.

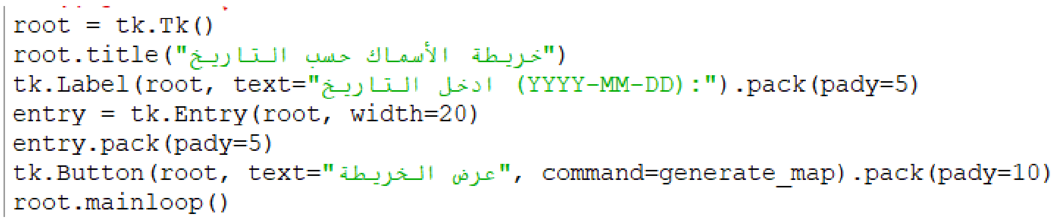


```
    output_file = "fish_map.html"
    m.save(output_file)
    webbrowser.open('file://' + os.path.realpath(output_file))
```

Figure 6.18: Save and Open the Map

6.3.2.3.7 BUILD THE GUI USING TKINTER

The interface includes a label, an input field, and a button to trigger map generation. The layout is created using Tkinter's default packing method.



```
root = tk.Tk()
root.title("خريطة الأسماك حسب التاريخ")
tk.Label(root, text="ادخل التاريخ (YYYY-MM-DD):").pack(pady=5)
entry = tk.Entry(root, width=20)
entry.pack(pady=5)
tk.Button(root, text="عرض الخريطة", command=generate_map).pack(pady=10)
root.mainloop()
```

Figure 6.19: Build the GUI Using Tkinter

6.4 GUI WORKFLOW AND USER INTERACTION

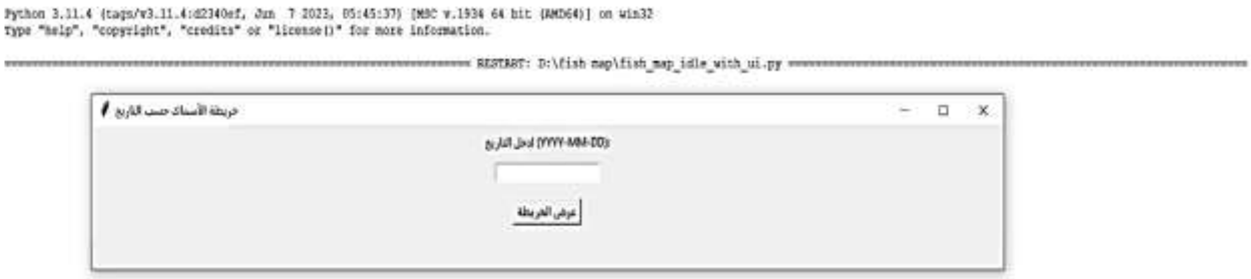
This section provides a detailed explanation of how users interact with the developed GUI application. The workflow is designed to be simple, responsive, and informative, guiding users from data entry to map visualization through clearly defined steps.

6.4.1 LAUNCHING THE APPLICATION

When the program is executed, the GUI window is initialized using Python’s Tkinter library. The interface is intentionally minimal, comprising the following components:

- A title bar indicating the application's function.
- A label instructing the user to input a date.
- A text input field.
- A button labeled “عرض الخريطة” (translated as “Show Map”).

This interface layout is chosen to reduce cognitive load, enabling users with no technical background to operate the system. The application remains idle at this stage, awaiting user input.



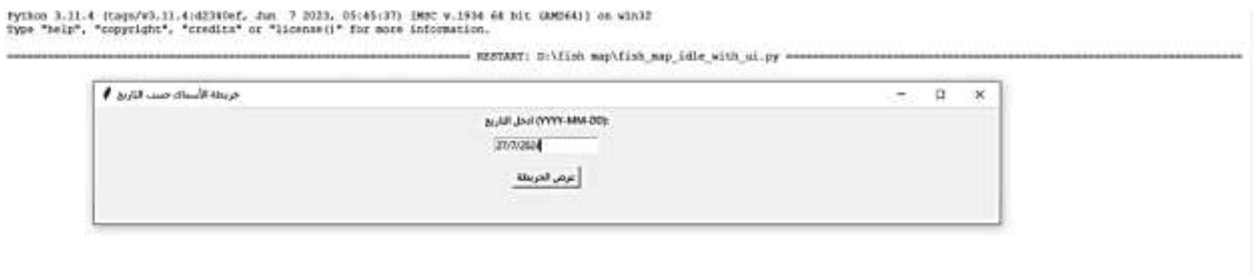
```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

============================== RESTART: D:\fish map\fish_map_idle_with_ui.py ==============================
```

Figure 6.20: Launching the Application

6.4.2 ENTERING THE DATE

The user is required to manually input a specific date in the format YYYY-MM-DD (e.g., 2025-07-01) into the entry field. This date serves as a query parameter that filters the data for the corresponding day's detection records.

To maintain compatibility with the timestamp format used in the CSV file generated by the AI-sensor integration system, the interface enforces ISO date formatting. This also simplifies subsequent parsing and filtering operations.



```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
==================================== RESTART: D:\fish map\fish_map_idle_with_ui.py ====================================
```

Figure 6.21: Entering the Date

6.4.3 INPUT VALIDATION

Upon pressing the "Show Map" button, the system initiates a validation step. This is implemented using Python’s datetime library, which attempts to convert the user-entered string into a valid date object.

- If the conversion fails (due to format mismatch or invalid characters), the system triggers a popup message using messagebox.showerror(). This informs the user of the incorrect format and prompts them to enter the date in the correct YYYY-MM-DD format.

This validation stage is critical to ensure the robustness of the program, preventing crashes or incorrect filtering results due to malformed input

6.4.4 Reading and Filtering the Data

- If the date input passes validation, the program proceeds to read the full detection dataset from the structured CSV file using the pandas library.
- The 'date' column in the dataset is preprocessed into date-only format using pd.to_datetime().dt.date.
- The dataset is filtered to retain only those rows that match the entered date.

This selective filtering mechanism enables temporal exploration of the fish detection data. If no records exist for the selected date, an informational popup is displayed using messagebox.showinfo() to notify the user. This allows the application to gracefully handle empty results without terminating execution.

6.4.5 GENERATING THE INTERACTIVE MAP

This section explains how the system responds after the user inputs a date and clicks the “عرض الخريطة” (Show Map) button. It covers the creation of the map, the use of marker clustering, how the system handles incorrect input or missing data, and how the final results are displayed.

6.4.5.1 MAP CREATION USING FOLIUM

When the user enters a correctly formatted date and matching records exist in the CSV file, the application initializes an interactive map using the **Folium** library. The map is automatically centered on the average latitude and longitude of all matching records, providing a relevant spatial view.

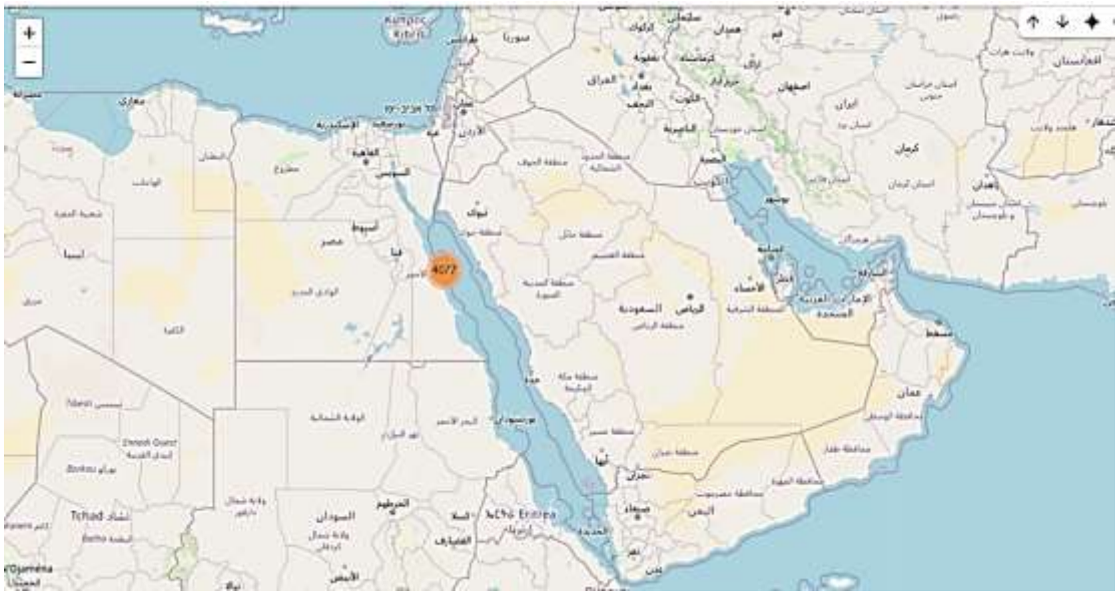


Figure 6.22: Map Creation Using Folium

6.4.5.2 MARKER CLUSTERING WITH MARKERCLUSTER PLUGIN

To improve readability when multiple detection points are close to each other, the system uses the MarkerCluster plugin from folium.plugins. This groups nearby markers into a single cluster icon when zoomed out.

When the user clicks on a cluster, it expands to reveal the individual detection points. This reduces visual clutter and makes the map easier to interpret, especially in densely populated regions.

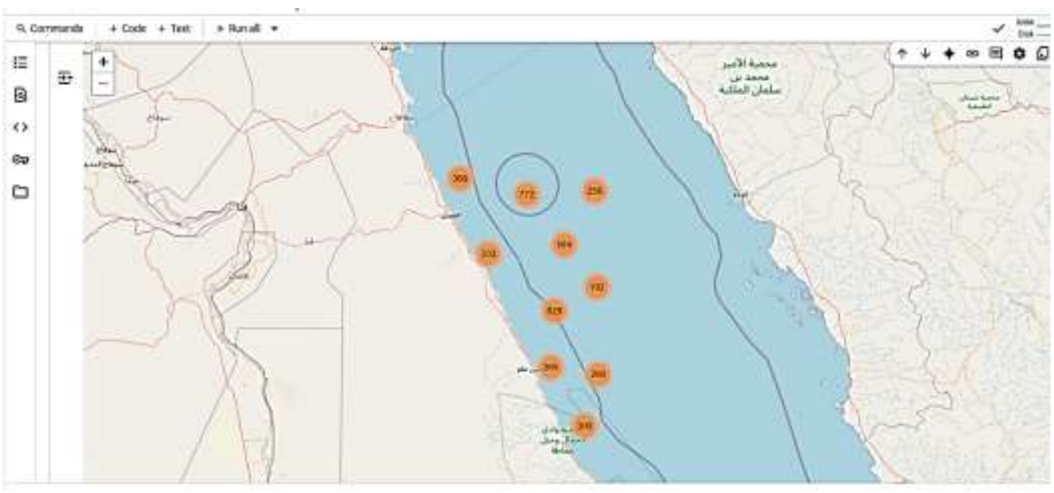
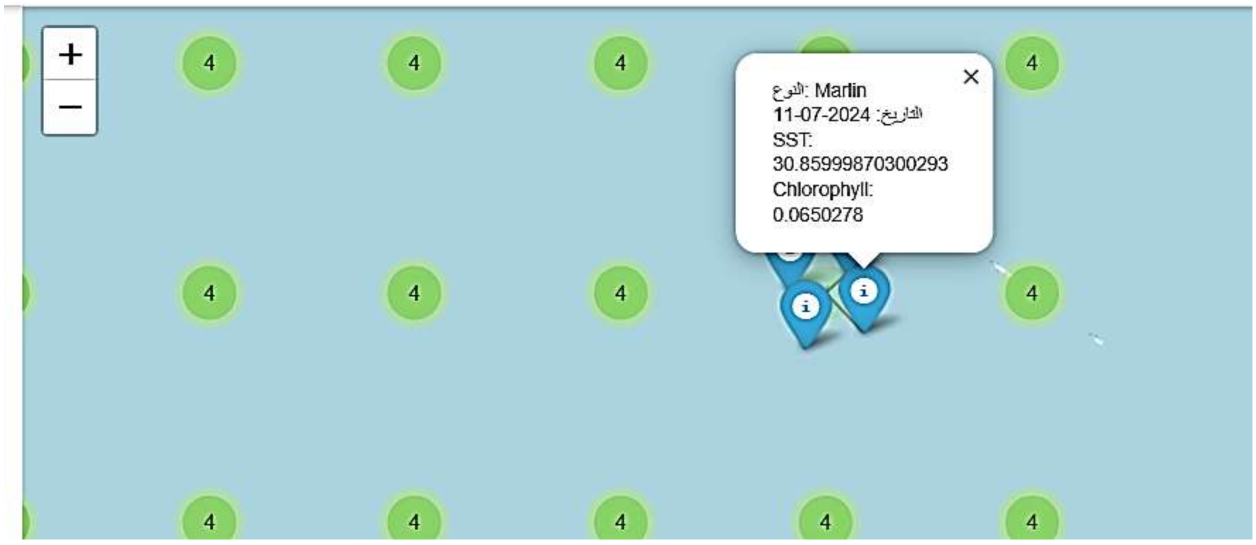


Figure 6.23: Marker Clustering with MarkerCluster Plugin

Each detection point is marked using an individual folium.Marker**. These markers display popups containing:**

- The predicted fish type
- The detection date
- Sea Surface Temperature (SST)
- Chlorophyll-a concentration



النوع: Marlin
التاريخ: 11-07-2024
SST: 30.85999870300293
Chlorophyll: 0.0650278

Figure 6.24: Each marker shows fish type, date, SST, and chlorophyll-a.

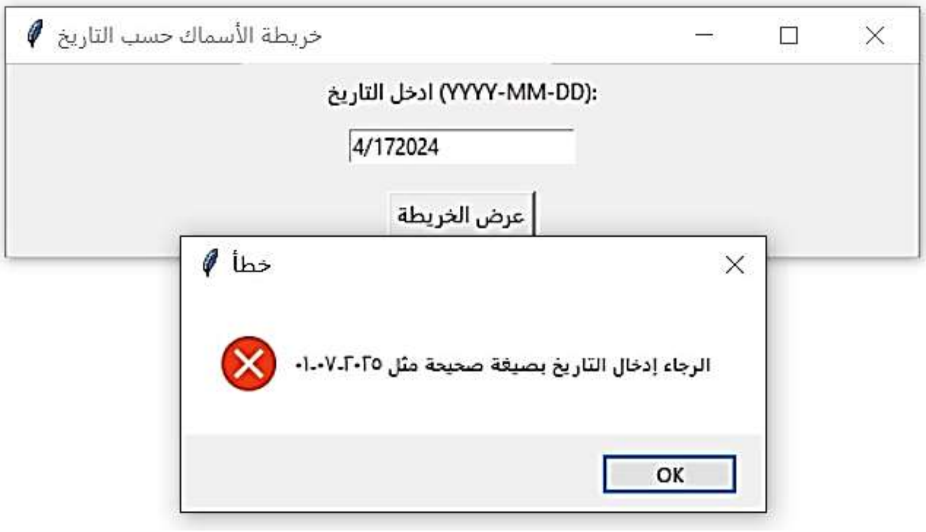
6.4.5.3 INVALID DATE INPUT HANDLING

If the user enters the date in an incorrect format (e.g., 01/07/2025 instead of 2025-07-01), the application stops processing. It immediately displays a popup message using messagebox.showerror() as ("خطأ", "الرجاء إدخال التاريخ بصيغة صحيحة مثل 01-07-2025") to notify the user that the date format is invalid and must follow the correct YYYY-MM-DD format.

This prevents unexpected errors and ensures data is handled reliably.

After all markers are added, the map is saved locally as an HTML file and opened automatically in the user's default web browser using Python's webbrowser module. This allows users to explore the data immediately without requiring any GIS software or internet connection.

6.4.5.4 NO DATA FOR SELECTED DATE

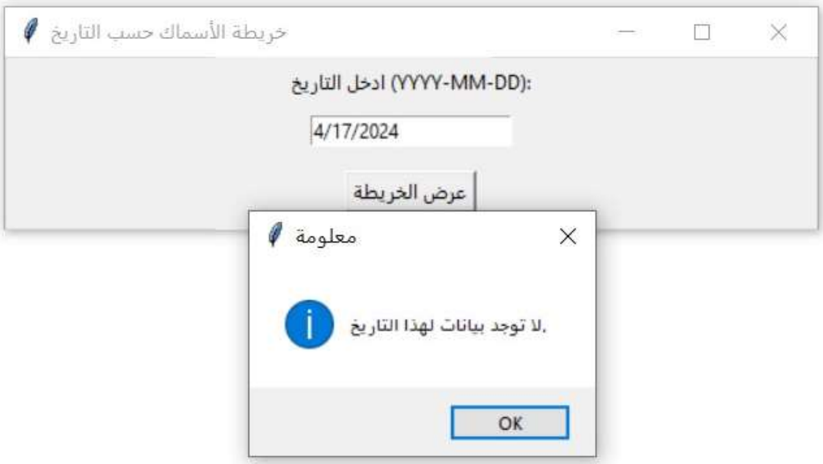


الرجاء إدخال التاريخ بصيغة صحيحة مثل ٢٠٢٥-٠٧-٠١

Figure 6.25: Invalid Date Input Handling

If the entered date is valid but no data records match it in the CSV file, the system notifies the user with an informational popup using messagebox.showinfo("معلومة" ,"لا توجد بيانات لهذا التاريخ.") to notify the user that no data points are available for that date.

- No map is generated in this case.



لا توجد بيانات لهذا التاريخ.

Figure 6.26: No Data for Selected Date

- The message clearly states that no data is available for the selected date.

6.5 ADVANTAGES OF THE OFFLINE GUI SOLUTION

This section outlines the strengths of the Python-based GUI developed for offline fish detection data visualization

6.5.1 OFFLINE ACCESSIBILITY AND FIELD USABILITY

One of the primary advantages of the developed GUI is its ability to operate entirely offline. This makes it particularly suitable for use in field conditions—such as marine environments—where internet connection may be limited or unavailable. By removing the need for online access, the tool ensures that fish detection data can be explored and visualized immediately at the site of collection, enhancing responsiveness and reducing data lag.

6.5.2 COST-EFFECTIVE AND LICENSE-FREE

The system is built entirely using free and open-source Python libraries such as Tkinter, Pandas, Folium, and webbrowser. This eliminates the need for commercial software licenses, such as those required by ArcGIS or proprietary GIS platforms. As a result, the GUI can be distributed and used by students, researchers, or small organizations without financial barriers.

6.5.3 LIGHTWEIGHT OPERATION ON EMBEDDED SYSTEMS

Unlike web-based applications that may require strong processors or cloud infrastructure, the developed GUI is optimized for lightweight execution. It can run smoothly on devices with limited computational power—such as Raspberry Pi—making it ideal for operation in remote settings. This design also supports data collection and analysis directly from the source device, without the need for transferring files to higher-end computers.

6.5.4 USABILITY FOR NON-TECHNICAL USERS

The interface is designed with simplicity in mind, allowing users with no programming or GIS background to operate the system with ease. The user is only required to input a date, after which the system handles all data filtering, map generation, and browser display. Built-in popup messages offer guidance in case of errors or missing data, improving accessibility and reducing the learning curve.

6.5.5 MODULARITY AND EXPANDABILITY

The application is modular and structured in a way that facilitates future enhancements. Filters for fish type, SST range, or chlorophyll concentration can be added, as well as data export tools, charts, or web integration features. This makes the GUI not only a working prototype but also a scalable solution that can evolve into a full-fledged environmental monitoring platform.

6.6 SUMMARY

This section summarizes the contents and outcomes of this Chapter, focusing on the transformation from raw data collection to interactive visualization.

6.6.1 OVERVIEW OF GUI DEVELOPMENT

The chapter introduced the software system designed to convert environmental sensor data into a usable, visual format. Starting with a CSV file containing GPS coordinates, temperature readings, and chlorophyll estimates, the system connected these outputs to a graphical user interface capable of filtering and rendering spatial data.

6.6.2 COMPARISON BETWEEN GIS AND PYTHON GUI

An initial attempt was made to visualize the data using ArcGIS. However, limitations such as licensing requirements, lack of automation, and dependence on an internet connection prompted the development of an offline solution. The Python-based GUI, created using Tkinter and Folium, provided a lightweight, interactive, and cost-effective alternative that better matched the project's field-based requirements.

6.6.3 TRANSITION TOWARD A SCALABLE DATA-DRIVEN INTERFACE

The development process laid the groundwork for further system expansion. With its modular architecture, the GUI is positioned as a base for future web-based solutions that enable real-time sharing and broader accessibility of environmental monitoring data.

6.7 WEB-BASED VISUALIZATION SYSTEM USING LABVIEW G WEBVI

6.7.1 INTRODUCTION

The previous sections detailed the development of an offline visualization system using Python and ArcGIS, which served as a local solution for displaying fish detection results based on sensor data stored in CSV format. Although this approach was effective for field-based deployment and did not require internet access, it was constrained by its inability to support real-time updates, remote accessibility, or multi-user access. The GUI was restricted to local execution on the same device where the data was stored or processed.

To overcome these limitations, this section introduces a web-based visualization system developed using the LabVIEW G Web Development Environment (G WebVI). This new system enables users to visualize prediction data on an interactive map directly from any modern web browser—without the need to install specialized software or access the local storage of the processing device.

Unlike the offline GUI, the G WebVI system connects to a Firebase Realtime Database, allowing it to display fish detection data in real time as it is generated by the AI model running in Google Colab. The web interface also leverages Leaflet.js, a lightweight, open-source mapping library, to deliver an interactive and dynamic geospatial visualization experience.

By transitioning from a desktop-bound solution to a browser-based interface, the system becomes more scalable, accessible, and suitable for remote monitoring applications, supporting potential future use cases such as mobile dashboards, collaborative platforms, or integration with other web services.

6.7.1.1 G WEB OFFERS THE FOLLOWING KEY FEATURES

- Fully graphical programming: Design of the user interface and logic through visual block-based programming.
- Browser-based execution: Applications are exported in HTML5 and JavaScript formats and can run on any device with a browser.
- Database integration: Support for reading data from JSON, RESTful APIs, and Realtime Databases.
- Interactive map support: Integration with the Leaflet library to display live geographic data.
- Scalability: Can be deployed in Internet of Things (IoT) systems or integrated with RT/FPGA processors via LabVIEW.

This environment was chosen for the project due to its ease of development, fast responsiveness to geospatial data, and compatibility with platforms such as Google Drive, Firebase, and real-time databases—making it ideal for live data updates and interactive visualization.

6.7.1.2 WHY LEAFLET WAS CHOSEN

- It is open source.
- Completely free to use.
- Easily integrates with G Web.
- Performs efficiently in browser-based interactive applications.
- Offers rich features such as marker clustering, popups, and shapes without unnecessary complexity.

6.7.1.3 SUMMARY BEFORE DIVING INTO DETAILS

This code represents a complete project for displaying geospatial data through a web-based interface using open-source technologies, connected to live databases, to provide a flexible and interactive user experience—without relying on paid solutions like Google Maps.

6.7.2 OVERALL CODE DESCRIPTION

This code is responsible for building an interactive web interface to display a dynamic map with the ability to add and update location data in real time. It relies on the Leaflet library, an open-source, lightweight mapping library that integrates easily with G Web Development Software. The map tile data is loaded from the free OpenStreetMap service.

The main objectives of the code are:

- Creating an interactive map within the web browser.
- Loading geographic coordinates (latitude/longitude) from JSON files or external databases.
- Displaying these data points as markers with pop-up windows.
- Supporting real-time updates via Firebase integration.
- Displaying geographical data such as polylines or shapes.

- Handling interactive events with map elements.

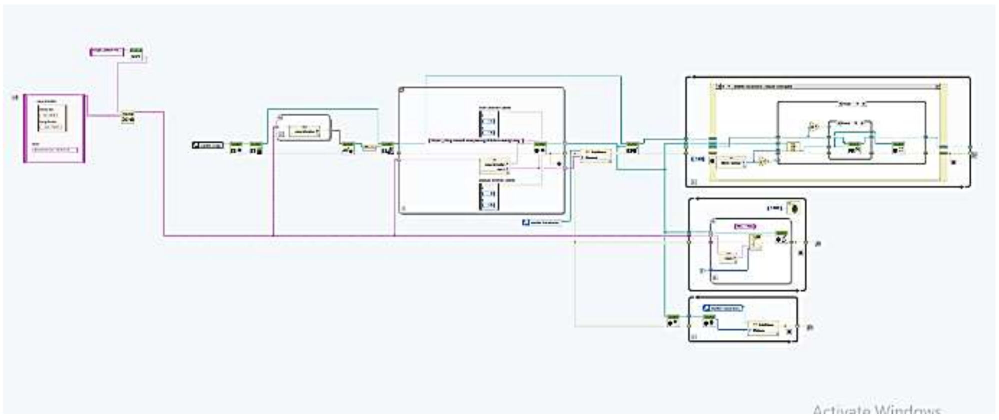


Figure 6.27: Overall Code Description

6.7.2.1 BLOCK 1: CREATE MAP

Functionality

This block is responsible for initializing the map inside the WebVI interface from the start**. It defines:**

- The container where the map will be displayed
- The center point of the map (latitude, longitude)
- The default zoom level

Once executed, an interactive map element appears on the web page, and the user can interact with it freely.

Structure and Connections

Inputs of the Block:

- Container ID: The HTML element ID where the map will be displayed (e.g., mapContainer)
- Center: The coordinates of the map's center ([30.033, 31.233] for Cairo)
- Zoom Level: The zoom level, from 1 (farthest view) to 18 (closest view)
- Map Options: Additional settings such as enabling or disabling scroll zoom, dragging, etc.

Outputs of the Block:

Map Reference: A reference to the created map instance. This output is essential and will be used in subsequent blocks—for example, to add markers, draw shapes, or destroy the map.

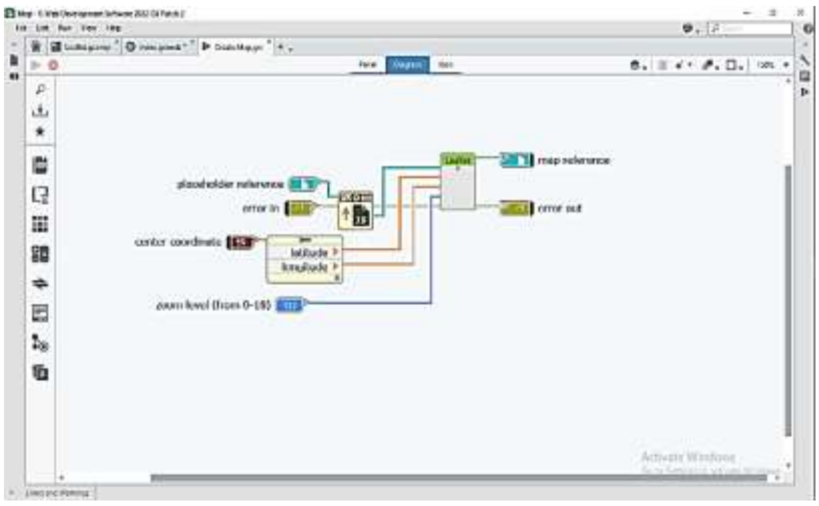


Figure 6.28: Outputs of the Block

6.7.2.2 BLOCK 2: ADD OPENSTREETMAP TILE LAYER

Functionality

This block adds the tile layer to the map that was previously created. The tile layer contains the street images, roads, and map visuals that the user sees.
It uses OpenStreetMap (OSM) as a free and open-source tile server, allowing detailed maps to be displayed without requiring a paid API like Google Maps.

Structure and Connections

Inputs of the Block:

- **Map Reference**: The map reference output from the previous block (**Create Map**)
- **URL Template**: The URL of the tile server (https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png)
- **Attribution**: A text string to credit the map source, **mandatory for OSM**

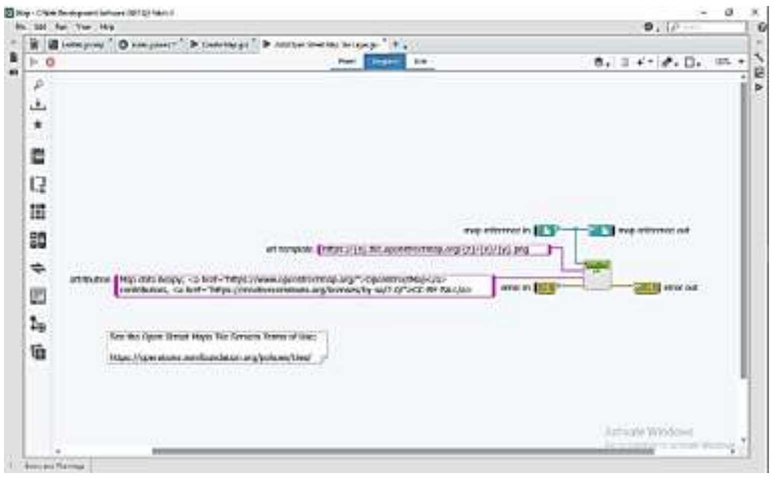
("Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, <a href="https://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>")

Outputs of the Block:

Map Reference: The same map instance, but now with the tile layer added, allowing other blocks to continue building upon it.

Why Is This Block Used?

The map created in the first block is empty by default—it's just a blank frame.
To render the actual geographic content, you must add a tile layer, which provides the visual map details (roads, landmarks, etc.).
This block ensures the map appears with full geographic context to the user.



See the Open Street Maps Tile Servers Terms of Use:

https://operations.osmfoundation.org/policies/tiles/

Figure 6.29: Adds OSM tile layer for map display.

6.7.2.3 BLOCK 3: CREATE POLYLINE SHAPE

Functionality
This block is used to create a Polyline object that represents a path composed of multiple sequential points. The created shape can then be displayed on the map.

Inputs:

Input	Description
Coordinates	An array of coordinates (latitude, longitude) that define the points of the line.
Options	Optional settings such as line color, thickness, dash style, and opacity.

Outputs:

Output	Description
Shape Reference	A reference to the created Polyline shape, to be passed later to the **Add Shapes to Map** block.

Usage
This block is commonly used to draw lines that represent routes, tracking paths, or geographic coverage areas.
The Create Polyline Shape block generates a multi-segment line object that can be added to the map through its Shape Reference, enabling visual representation of paths on an interactive map.

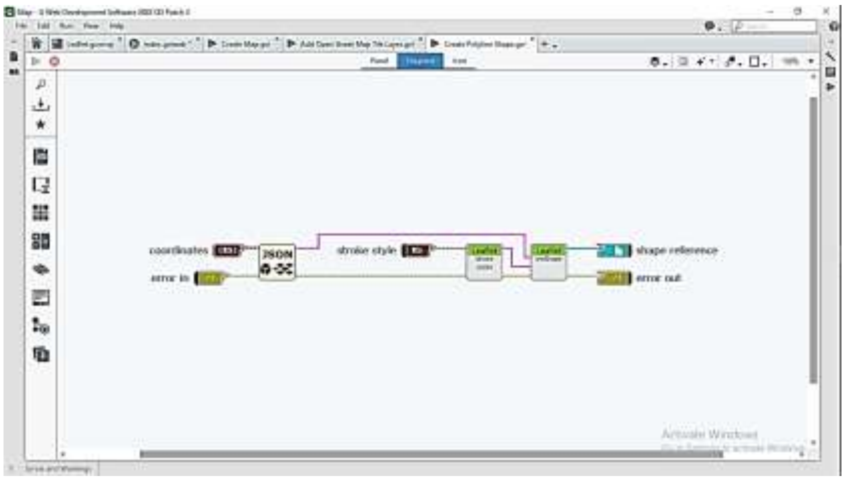


Figure 6.30: Draws multi-segment lines to show paths or routes on the map.

6.7.2.4 BLOCK 4: ADD SHAPES TO MAP

Functionality

This block allows you to add geometric shapes to the map—such as polylines or polygons—to highlight specific paths, routes, or areas on the interactive map. It is useful for visualizing regions of interest or drawing boundaries directly on the map.

Structure and Connections

Inputs of the Block:

- Map Reference: The map instance passed from the previous blocks
- Shape Reference: A reference to the shape object you created (e.g., a polyline or polygon)
- Options: Optional settings to control the stroke color, line thickness, opacity, and other visual properties

Outputs of the Block:

Map Reference: The same map, now updated with the added shape(s), allowing further interaction or visualization layers

In Summary

The “Add Shapes to Map” block draws graphical objects over the map to visually emphasize routes, zones, or boundaries, enhancing the interpretability of spatial data.

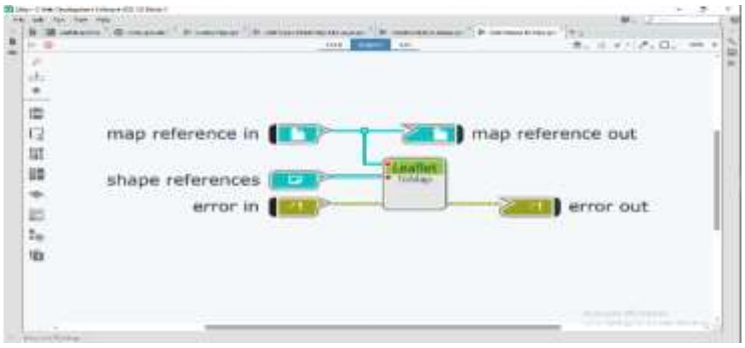


Figure 6.31: Adds shapes to highlight paths or areas on the map.

6.7.2.5 BLOCK 5: CREATE MARKER

Purpose

This block is used to create a marker on the interactive map to precisely pinpoint a specific location. It also allows for customizing the marker's appearance and displaying a popup window with additional information when the marker is clicked.

Structure and Connections

Inputs:

- coordinate: The geographic coordinates of the marker (latitude, longitude)
- iconUrl: (Optional) The URL of a custom icon to represent the marker
- iconAnchor: Defines how the icon is anchored to the geographic point (e.g., center, bottom-left)
- popupText: Text to be displayed in the popup window when the marker is clicked
- popupAnchor: Determines the popup's position relative to the icon
- error in: Standard input for error handling

Outputs:

- marker reference: A reference to the created marker object, which can be used to add it to the map or link it to events
- error out: Standard output for error reporting and debugging

Note

This block allows full customization of the marker's appearance and enables attaching informative text to increase user interaction when clicking on specific locations.

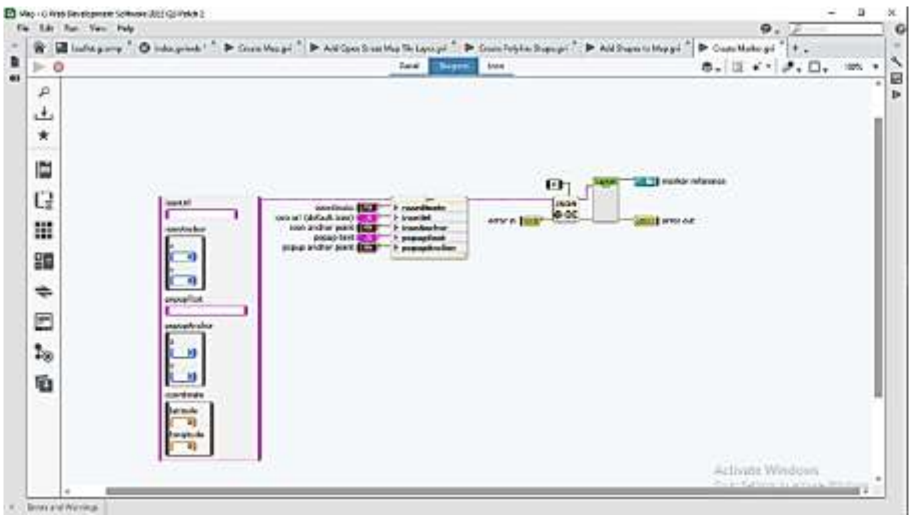


Figure 6.32: Creates a customizable map marker with optional popup info

6.7.2.6 BLOCK 6: ADD MARKERS TO MAP

Purpose

This block is used to add a group of markers to the map all at once. Whether the markers were created individually or as a collection, this block allows them to be displayed directly on the WebVI interface.

Structure and Connections

Inputs:

- map reference: A reference to the map on which the markers will be placed
- markers: An array of marker references to be added to the map
- error in: Standard input for error handling

Outputs:

- map reference: The same map reference, now with the markers added
- error out: Standard output for error handling and debugging

Note

This block enables displaying multiple markers at once, making it easier to manage and visualize multiple locations on the interactive map. It also simplifies the process compared to adding each marker individually.

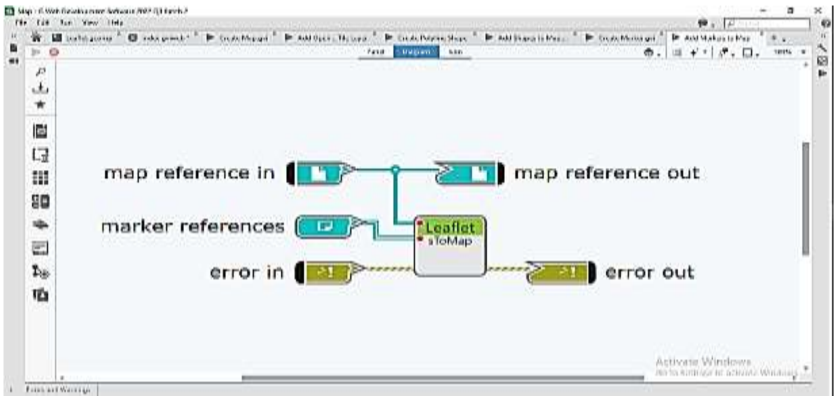


Figure 6.33: Adds multiple markers to the map in one step for easier visualization.

6.7.2.7 BLOCK 7: ZOOM TO MARKER IN MARKER GROUP

Purpose

This block allows the map to automatically zoom in on a specific marker within a group of markers. It enhances the user experience by making it easier to focus on a particular location when an item or event is selected in the user interface.

Structure and Connections

Inputs:

- map reference: A reference to the map where the zoom will occur
- marker group: A reference to the group containing the markers
- marker ID: The unique identifier (ID) of the marker to zoom in on
- error in: Standard input for error handling

Outputs:

- map reference: The same map reference, now adjusted to focus on the selected marker
- error out: Standard output for error handling and debugging

Note

This block is especially useful when dealing with large clusters of markers, allowing the application to quickly zoom in on a specific marker to highlight and center it, improving both navigation and interactivity on the map.

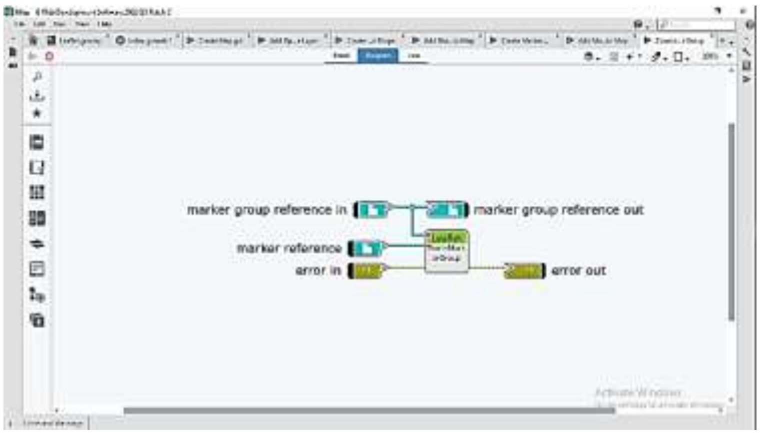


Figure 6.34: Zooms the map to a selected marker within a group for easier focus and navigation.

6.7.2.8 BLOCK 8: SHOW MARKER GROUP

Purpose

This block is used to display an entire group of markers on the map at once, after they have been created and organized into a Marker Group object. It helps manage markers in a structured and synchronized way.

Structure and Connections

Inputs:

- ➢ map reference: A reference to the map where the marker group will be shown
- ➢ marker group: A reference to the marker group that should be displayed
- ➢ error in: Standard input for error handling

Outputs:

- ➢ map reference: The same map reference, now updated to include the visible marker group
- ➢ error out: Standard output for error handling and debugging

Note

This block allows you to control the visibility of a large number of markers as a group, rather than managing them individually. It simplifies handling large datasets on the interactive WebVI map interface by enabling efficient group-level operations.

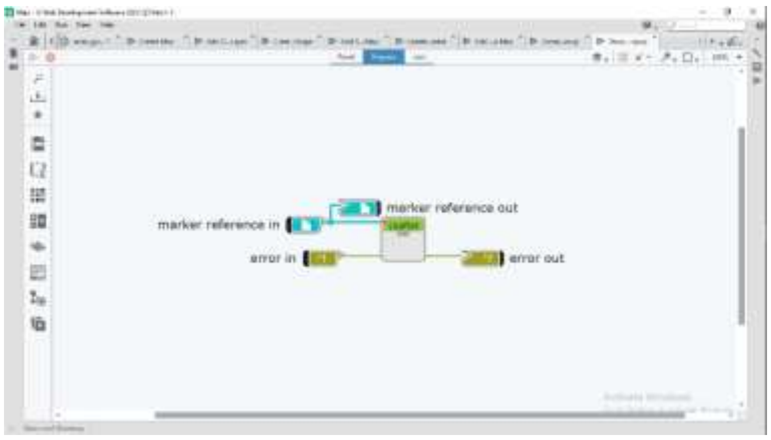


Figure 6.35: Displays a group of markers on the map for easier management and visualization.

6.7.2.9 BLOCK 9: UPDATE MARKER POPUP

Purpose

This block is used to update the content of a popup window associated with a specific marker on the map. It enables dynamic display of new information during application runtime without needing to recreate the marker.

Structure and Connections

Inputs:

- marker reference: A reference to the marker whose popup needs to be updated
- popupText: The new text content to be displayed in the popup window
- error in: Standard input for error handling

Outputs:

- marker reference: The same marker reference, now updated with the new popup text
- error out: Standard output for error handling and debugging

Note

This block is essential for real-time updates to marker information—such as device status, sensor readings, or other dynamic data. It ensures the popup content reflects the latest information for the user, without modifying the marker's position or style.

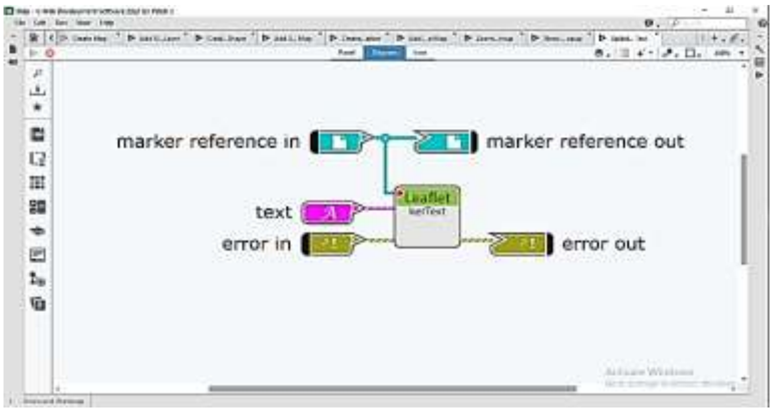


Figure 6.36: Updates a marker's popup text in real time without recreating the marker.

6.7.2.10 BLOCK 10: ADD MARKER EVENT LISTENER

Purpose

This block is used to attach an event listener to a marker on the map, allowing the application to execute specific actions when certain events occur—such as a click, hover, or other user interactions.

Structure and Connections

Inputs:

- marker reference: A reference to the marker that will be linked to the event
- event type: The type of event to listen for (e.g., "click", "mouseover")
- event handler: The name or reference of the callback function to be executed when the event occurs
- error in: Standard input for error handling

Outputs:

- marker reference: The same marker reference, now with the event listener attached
- error out: Standard output for error handling and debugging

Note

This block enhances interactivity within the application by enabling custom behavior in response to user actions. It supports building a more responsive and dynamic user experience, where map elements can trigger data displays, alerts, or navigation based on how users interact with them.

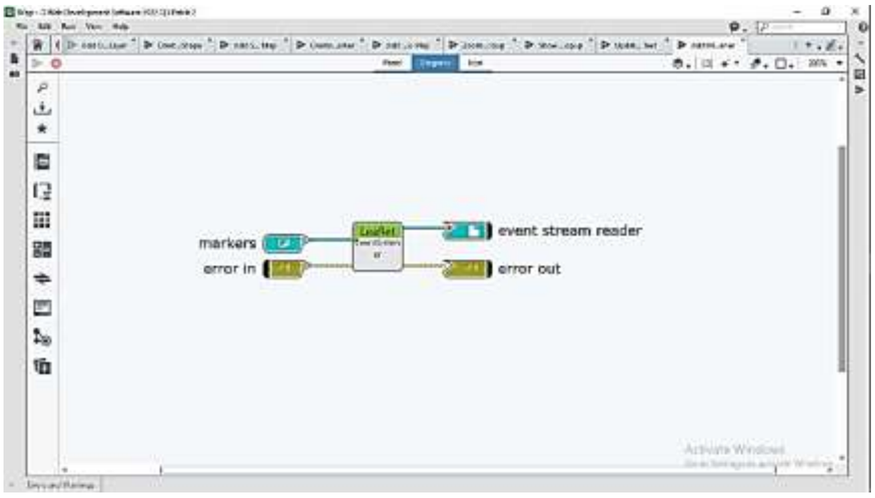


Figure 6.37: Adds event (e.g., click) to marker for interactive actionS

6.7.2.11 BLOCK 11: WAIT FOR MARKER EVENT

Purpose

This block allows you to wait for a specific event to occur on a given marker—such as a click or mouseover—and then capture and handle that event within the WebVI data flow.

Structure and Connections

Inputs:

- marker reference: A reference to the marker on which the event will be monitored
- event type: The type of event to wait for (e.g., "click", "mouseover")
- error in: Standard input for error handling

Outputs:

- event data: The data associated with the triggered event (e.g., click coordinates, marker ID, or other relevant information)
- error out: Standard output for error handling and debugging

Note

This block enables synchronous interaction programming, where a specific behavior is triggered as soon as the event occurs. It is especially useful for building highly interactive map applications, allowing users to engage with map elements in real time and receive immediate visual or functional feedback**.**

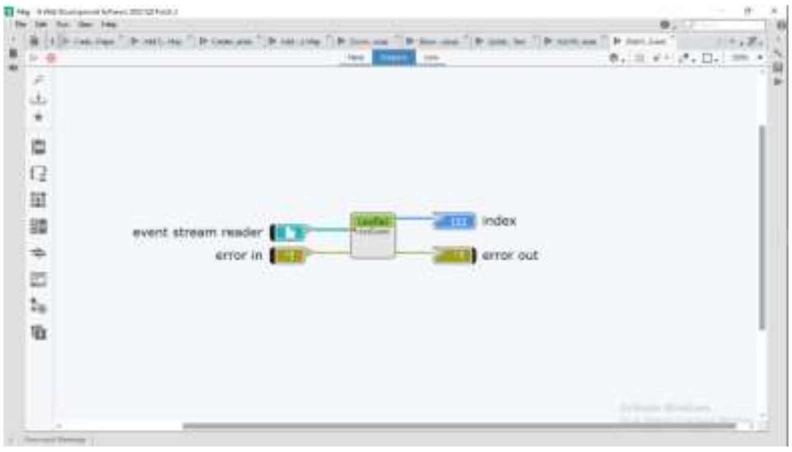


Figure 6.38: Waits for a marker event (e.g., click) and captures its data for interaction

6.8 DATA FLOW

6.8.1 SETTING UP FIREBASE REALTIME DATABASE

The Firebase Realtime Database was configured as a central data hub to store all prediction results generated by the AI model. This setup ensures real-time availability of data for all users, with the added benefit of historical access to past predictions at any point in time.

Setup Story and Steps

- **Creating a Firebase Project**
  - A new project was created on the Firebase Console
  - Realtime Database was enabled
  - Test mode rules were applied during the development phase, with future plans to secure access using authentication rules
  - Purpose: Provide a single endpoint for both Google Colab and WebVI to read from and write .

.**Purpose of the Realtime Database**:

- Serves as a centralized store for all prediction results generated by the AI model
- Allows direct integration with the WebVI interface for live visualization
- Maintains a complete log of all prediction sessions
- Designed for future scalability, including support for mobile apps or REST API endpoints

Official Summary

The Firebase Realtime Database was set up under the path /fish_monitor to serve as a centralized, structured, and secure repository for storing all AI model prediction results. It enables instant availability of data for visualization and analysis, while also preserving a permanent archive of all detected records across sessions.

6.8.2 SETTING UP GOOGLE DRIVE FOR TEMPORARY JSON STORAGE

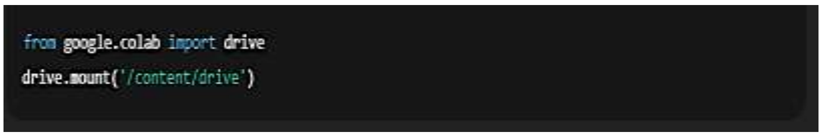
Google Drive was used as a cloud-based intermediary storage for the JSON file containing prediction results during AI model execution within Google Colab. This setup ensures that the file is automatically accessible at the beginning of each session—without requiring manual uploads—and that the prediction data remains continuously available in a seamless and efficient manner.

Setup Story and Steps

- **Preparing the Google Drive Environment**
    - A dedicated folder was created in Google Drive (e.g., /FishMonitor)
    - This folder contains a copy of the main JSON file
    - Sharing permissions were configured to allow read and write access from Google Colab

- **Linking Google Colab with Google Drive**

  Using the google.colab library, the following code was executed in Colab:



```
from google.colab import drive
drive.mount('/content/drive')
```

Figure 6.41: Linking Google Colab with Google Drive

This command mounts Google Drive as a local directory inside the Colab environment, allowing for direct read/write operations on the file system.

Purpose of Using Google Drive:

- At the start of each new session, Colab reads the existing JSON file from Drive
- No need to re-upload the file manually every time
- Ensure that initial data is ready and available quickly
- Accelerates the execution cycle and reduces operational errors

Runtime Workflow Inside a Session

- **At session start:**
  - Colab reads the JSON file directly from the mounted Google Drive path
  - Begins processing predictions and writing back to the file
- **During the session:**
  - The JSON file is updated in real time as the model runs
  - It stays synchronized with the live predictions being generated
- **At session end:**
  - The file contents can be cleared if a fresh session is required
  - Or preserved as a log/archive of past predictions, depending on use case

Official Summary

Google Drive was set up as a temporary, cloud-accessible storage layer for the prediction JSON file, ensuring its availability at the start of every new Colab session. This eliminates the need for manual file uploads, streamlines model execution, and enables real-time synchronization between prediction outputs and the stored data.

The AI model running in Google Colab is designed to interact with two parallel data storage pipelines:

- A temporary JSON file on Google Drive
- A Firebase Realtime Database for persistent long-term storage

This dual-path strategy ensures real-time availability for front-end visualization and long-term archival of all prediction results.

<u>Step-by-Step Integration Workflow</u>

- **Loading the JSON File from Google Drive at Session Start**
  - The pre-prepared JSON file stored on Google Drive is loaded at the start of each Colab session
  - Colab opens the file immediately and begins writing new data to it
  - Purpose: Provide ready-to-use prediction data without manual intervention at each session start
- **Updating the JSON File During Model Execution**
  - Each new prediction generated by the AI model is written directly to the JSON file
  - This allows any front-end system (like WebVI) to access up-to-date results instantly
  - Enables real-time monitoring by reading the file continuously
- **Synchronizing with Firebase**

- In parallel with writing to the JSON file, the same prediction data is also sent to the Firebase Realtime Database
- This ensures long-term preservation of all prediction sessions
- Provides a reliable archive for historical data analysis and system auditing

Data Handling Strategy

- **During the Session:**
  - The JSON file acts as a real-time buffer for immediate visualization
  - Firebase logs and stores every result persistently for historical access
- **After the Session:**
  - The JSON file can be cleared to start fresh in the next session
  - All data remains securely stored in Firebase, maintaining a complete log of past sessions

Output Consumed by WebVI

The WebVI interface reads fish-related data directly from the JSON file served by Firebase:

https://fish-monitor-efd05-default-rtdb.firebaseio.com/fishdata.json

- This provides real-time updates that are displayed immediately on the map
- No manual refresh or synchronization is needed
- Ensures complete consistency between AI predictions and displayed data

[illegible]

Figure 6.42: Output Consumed by WebVI

Official Summary

The AI model in Colab operates in synchronous dual-write mode, updating both the temporary JSON file (for real-time visualization) and Firebase Realtime Database (for historical record-keeping). This guarantees instant availability, reliable performance, and permanent storage of all prediction data, enabling seamless integration with systems like WebVI.

6.8.4 INTEGRATION OF THE WEBVI INTERFACE WITH FIREBASE

The WebVI interface is designed to act as the final layer responsible for visually displaying fish prediction data—extracted by the AI model—in an interactive map-based format. It relies on Firebase Realtime Database as the main data bridge, ensuring live, automated updates without any manual intervention.

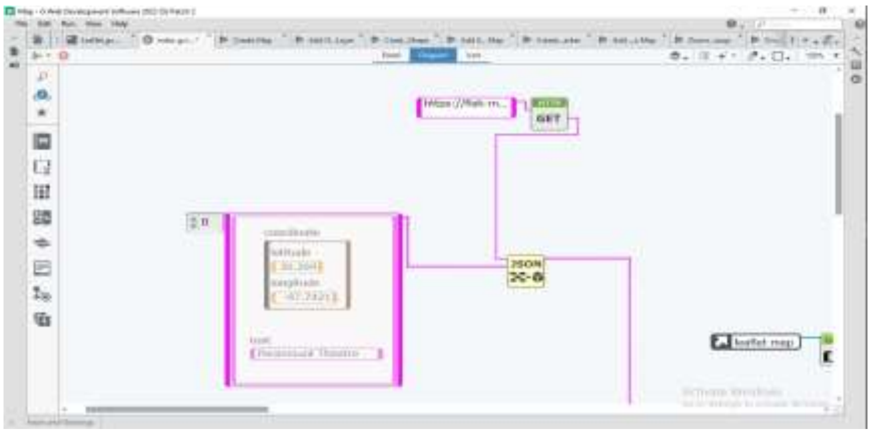


Figure 6.43: Links WebVI to Firebase for live fish prediction display.

How WebVI Connects to Firebase

- **Reading Data from Realtime Database**
    - WebVI performs a direct HTTP GET request to the Firebase JSON endpoint:
    - https://fish-monitor-efd05-default-rtdb.firebaseio.com/fishdata.json
    - The interface fetches the most recent data in real time
    - The map is updated using the Leaflet.js library to reflect live changes
    - Periodic polling ensures the display stays in sync with any updates

- **Displaying Data on the Map**
    - WebVI uses a dedicated Map Container (HTML element) as the canvas
    - It dynamically adds markers on the map using the parsed JSON data
    - Each marker represents a fish with:
    - Location coordinates (latitude, longitude)
    - Fish type
    - Estimated weight
    - Any additional metadata
    - Coordinates and attributes are updated during each refresh cycle

Purpose of Using Firebase

- Instant data updates with no delay
- Eliminates the need for manual uploads or manual refresh
- Ensure fish data is displayed immediately once predicted by the model
- Reduces human error or file handling issues
- Supports scalability (mobile apps, REST APIs, etc.)

Relation to Other System Components

WebVI serves as the critical bridge between:

- Live prediction results from the AI model (via JSON)
- Persistent storage in Firebase

In other words, it visualizes real-time prediction data from the Firebase JSON endpoint and provides a user-friendly, interactive front end for final users.

6.8.5 SUMMARY

The WebVI interface is directly connected to the Firebase Realtime Database, reading live JSON data and rendering it as interactive fish markers on a Leaflet map. This interface represents the final connection point between the AI model's predictions and the end user, delivering a fast, accurate, and fully automated visualization experience.

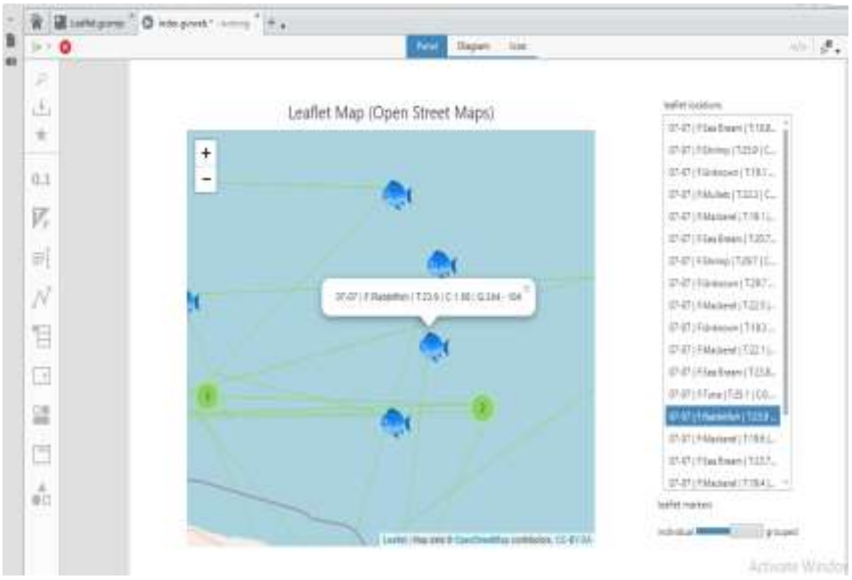


Figure 6.44: WebVI displays live AI fish data via Firebase for smart marine tracking.

With the completion of this system's documentation, it becomes clear how the integration of G WebVI, Google Colab, and Firebase has enabled the creation of a smart, end-to-end solution for real-time and persistent monitoring of fish data. This seamless combination brought together AI-driven prediction, cloud-based data storage, and an interactive user interface, ensuring both accuracy of insights and ease of access from anywhere at any time.

This project opens up future possibilities for scaling and enhancement, whether by:

- Introducing deeper analytical capabilities,
- Developing more advanced visualization interfaces, or
- Integrating with physical monitoring systems in real-world environments.

Ultimately, it stands as a powerful tool for supporting marine resource management and conservation decisions.

With this, the system documentation is complete serving as a comprehensive technical reference for any developer or researcher aiming to further enhance or expand the solution.

6.9 CHALLENGES AND FUTURE PLANS

As with most interdisciplinary systems, the integration of embedded hardware, AI algorithms, and cloud-based visualization surfaces several technical and logistical challenges that merit discussion. Addressing these issues is essential for transitioning the current prototype into a scalable, production-ready platform

6.9.1 TECHNICAL CHALLENGES

- Hardware Constraints: The Raspberry Pi, while cost-effective and compact, has limitations in handling simultaneous data acquisition and on-device inference. Future iterations may require more powerful edge processors for stable performance.
- Sensor Integration and Calibration: Environmental variability and sensor noise affected data quality, requiring software-based filtering and validation mechanisms.
- Dependency on Internet Connectivity: The web interface depends on a stable connection to Firebase, which is not always feasible in marine or rural environments.

6.9.2 DATA ARCHITECTURE LIMITATIONS

- File-Based Synchronization: Reliance on JSON files stored in Google Drive and Firebase introduces risks of inconsistency during parallel access.
- Scalability Concerns: As data volume increases, the current flat data model may become inefficient for querying or visualizing historical trends

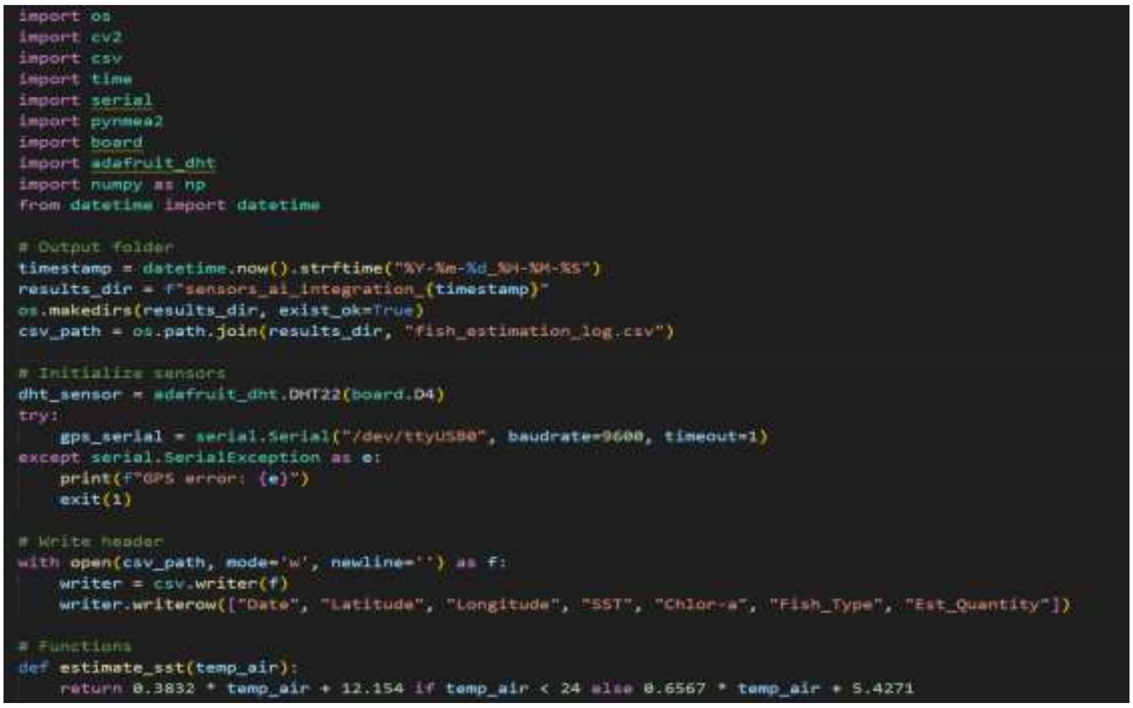
6.9.3 STRATEGIC FUTURE DIRECTIONS

- Embedded Intelligence: Embedding the AI model on edge hardware would allow standalone operation, enabling real-time predictions without cloud dependence.
- Hybrid Connectivity Models: Implementing a hybrid online/offline mode using browser storage (IndexedDB) or local servers would improve field usability.
- Data Visualization Dashboards: Replacing static map views with analytical dashboards would allow advanced filtering, comparison, and temporal analysis.
- Cross-Platform Deployment: Development of mobile and tablet-compatible interfaces would support a broader range of users in diverse environments.
- Sustainability and Adaptability: The current system can be expanded beyond fish detection to other environmental applications, reinforcing its long-term scientific value.

In summary, these enhancements will elevate the project from a functional prototype to a scalable, intelligent monitoring system suitable for real-world deployment in marine sustainability, environmental science, and beyond.

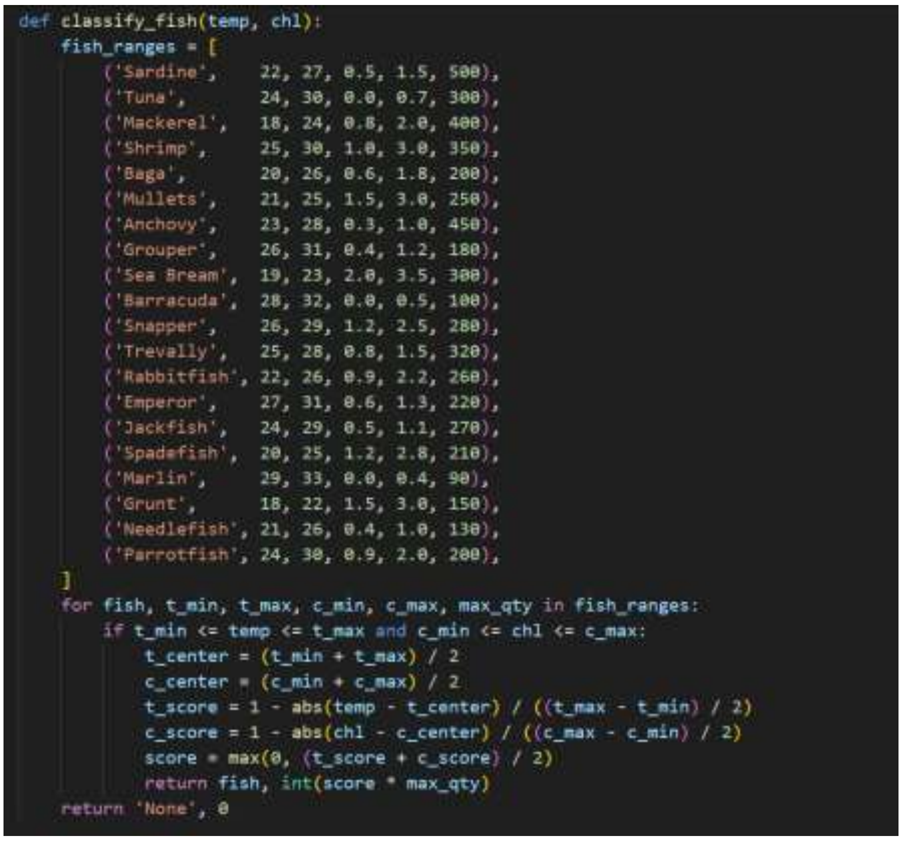
6.10 APPENDIX

APPENDIX A: FULL PYTHON CODE FOR SENSOR-AI INTERFACE

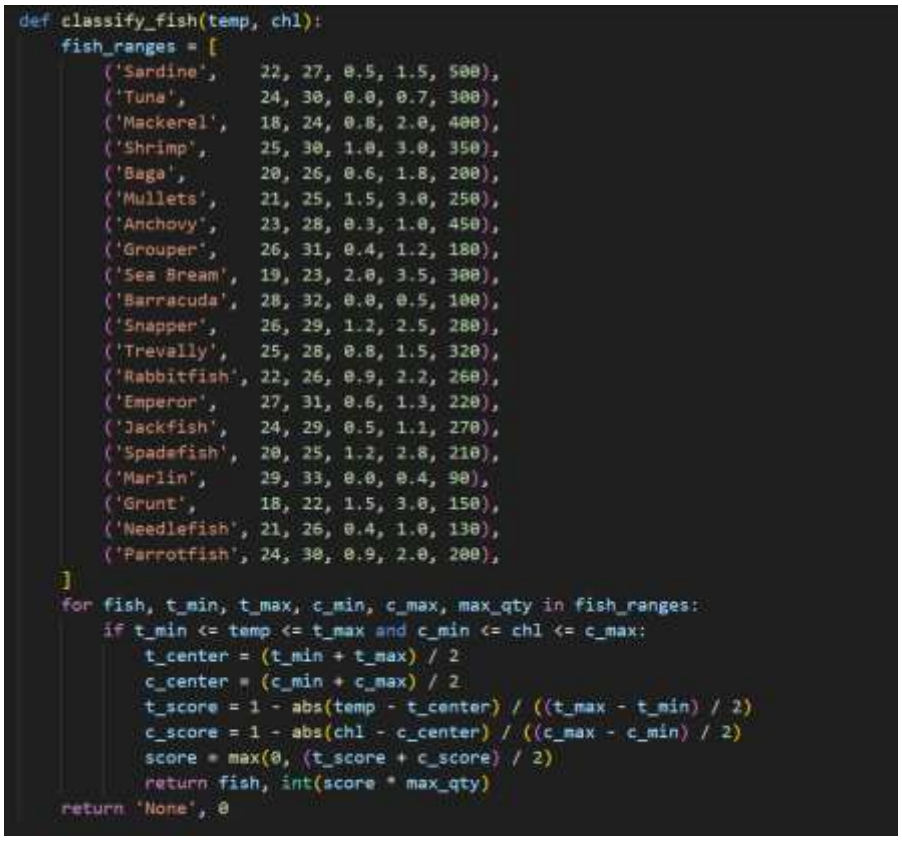


```python
import os
import cv2
import csv
import time
import serial
import pynmea2
import board
import adafruit_dht
import numpy as np
from datetime import datetime

# Output folder
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"sensors_ai_integration_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "fish_estimation_log.csv")

# Initialize sensors
dht_sensor = adafruit_dht.DHT22(board.D4)
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"GPS error: {e}")
    exit(1)

# Write header
with open(csv_path, mode='w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Date", "Latitude", "Longitude", "SST", "Chlor-a", "Fish_Type", "Est_Quantity"])

# Functions
def estimate_sst(temp_air):
    return 0.3832 * temp_air + 12.154 if temp_air < 24 else 0.6567 * temp_air + 5.4271
```

Figure 6.45: Full Python Code for Sensor-AI Interface1

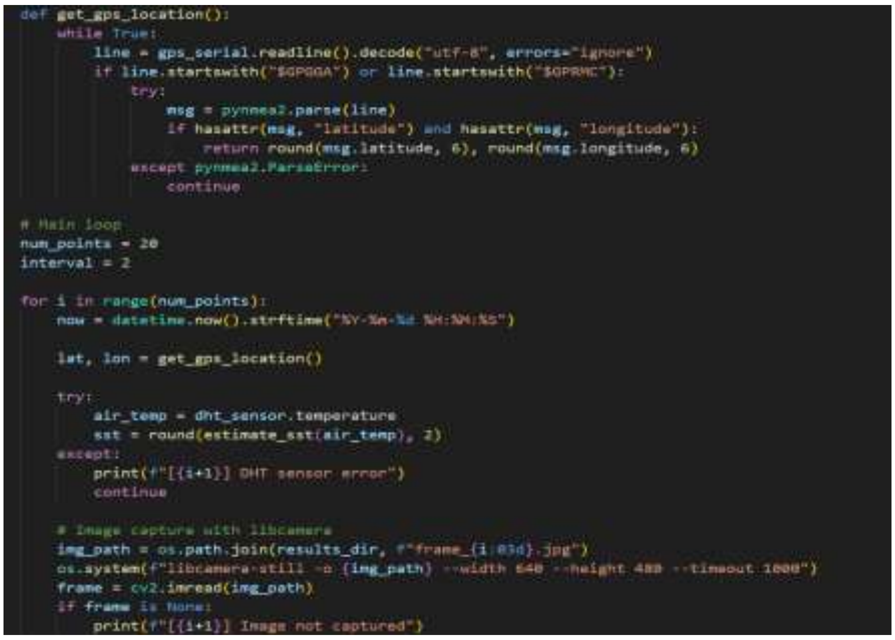


```python
def classify_fish(temp, chl):
    fish_ranges = [
        ('Sardine',    22, 27, 0.5, 1.5, 500),
        ('Tuna',       24, 30, 0.0, 0.7, 300),
        ('Mackerel',   18, 24, 0.8, 2.0, 400),
        ('Shrimp',     25, 30, 1.0, 3.0, 350),
        ('Baga',       20, 26, 0.6, 1.8, 200),
        ('Mullets',    21, 25, 1.5, 3.0, 250),
        ('Anchovy',    23, 28, 0.3, 1.0, 450),
        ('Grouper',    26, 31, 0.4, 1.2, 180),
        ('Sea Bream',  19, 23, 2.0, 3.5, 300),
        ('Barracuda',  28, 32, 0.0, 0.5, 100),
        ('Snapper',    26, 29, 1.2, 2.5, 280),
        ('Trevally',   25, 28, 0.8, 1.5, 320),
        ('Rabbitfish', 22, 26, 0.9, 2.2, 260),
        ('Emperor',    27, 31, 0.6, 1.3, 220),
        ('Jackfish',   24, 29, 0.5, 1.1, 270),
        ('Spadefish',  20, 25, 1.2, 2.8, 210),
        ('Marlin',     29, 33, 0.0, 0.4, 90),
        ('Grunt',      18, 22, 1.5, 3.0, 150),
        ('Needlefish', 21, 26, 0.4, 1.0, 130),
        ('Parrotfish', 24, 30, 0.9, 2.0, 200),
    ]
    for fish, t_min, t_max, c_min, c_max, max_qty in fish_ranges:
        if t_min <= temp <= t_max and c_min <= chl <= c_max:
            t_center = (t_min + t_max) / 2
            c_center = (c_min + c_max) / 2
            t_score = 1 - abs(temp - t_center) / ((t_max - t_min) / 2)
            c_score = 1 - abs(chl - c_center) / ((c_max - c_min) / 2)
            score = max(0, (t_score + c_score) / 2)
            return fish, int(score * max_qty)
    return 'None', 0
```



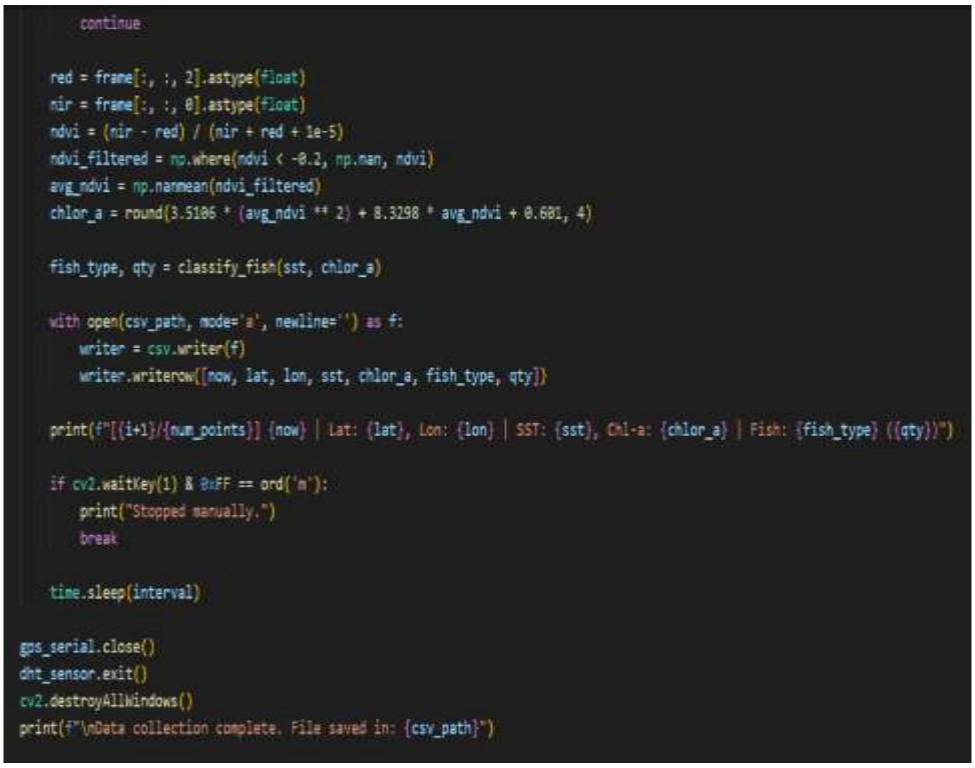
```python
def classify_fish(temp, chl):
    fish_ranges = [
        ('Sardine',    22, 27, 0.5, 1.5, 500),
        ('Tuna',       24, 30, 0.0, 0.7, 300),
        ('Mackerel',   18, 24, 0.8, 2.0, 400),
        ('Shrimp',     25, 30, 1.0, 3.0, 350),
        ('Baga',       20, 26, 0.6, 1.8, 200),
        ('Mullets',    21, 25, 1.5, 3.0, 250),
        ('Anchovy',    23, 28, 0.3, 1.0, 450),
        ('Grouper',    26, 31, 0.4, 1.2, 180),
        ('Sea Bream',  19, 23, 2.0, 3.5, 300),
        ('Barracuda',  28, 32, 0.0, 0.5, 100),
        ('Snapper',    26, 29, 1.2, 2.5, 280),
        ('Trevally',   25, 28, 0.8, 1.5, 320),
        ('Rabbitfish', 22, 26, 0.9, 2.2, 260),
        ('Emperor',    27, 31, 0.6, 1.3, 220),
        ('Jackfish',   24, 29, 0.5, 1.1, 270),
        ('Spadefish',  20, 25, 1.2, 2.8, 210),
        ('Marlin',     29, 33, 0.0, 0.4, 90),
        ('Grunt',      18, 22, 1.5, 3.0, 150),
        ('Needlefish', 21, 26, 0.4, 1.0, 130),
        ('Parrotfish', 24, 30, 0.9, 2.0, 200),
    ]
    for fish, t_min, t_max, c_min, c_max, max_qty in fish_ranges:
        if t_min <= temp <= t_max and c_min <= chl <= c_max:
            t_center = (t_min + t_max) / 2
            c_center = (c_min + c_max) / 2
            t_score = 1 - abs(temp - t_center) / ((t_max - t_min) / 2)
            c_score = 1 - abs(chl - c_center) / ((c_max - c_min) / 2)
            score = max(0, (t_score + c_score) / 2)
            return fish, int(score * max_qty)
    return 'None', 0
```

Figure 6.46: Full Python Code for Sensor-AI Interface2



```python
def get_gps_location():
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")
        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    return round(msg.latitude, 6), round(msg.longitude, 6)
            except pynmea2.ParseError:
                continue

# Main loop
num_points = 20
interval = 2

for i in range(num_points):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    lat, lon = get_gps_location()

    try:
        air_temp = dht_sensor.temperature
        sst = round(estimate_sst(air_temp), 2)
    except:
        print(f"[{i+1}] DHT sensor error")
        continue

    # Image capture with libcamera
    img_path = os.path.join(results_dir, f"frame_{i:03d}.jpg")
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")
    frame = cv2.imread(img_path)
    if frame is None:
        print(f"[{i+1}] Image not captured")
```

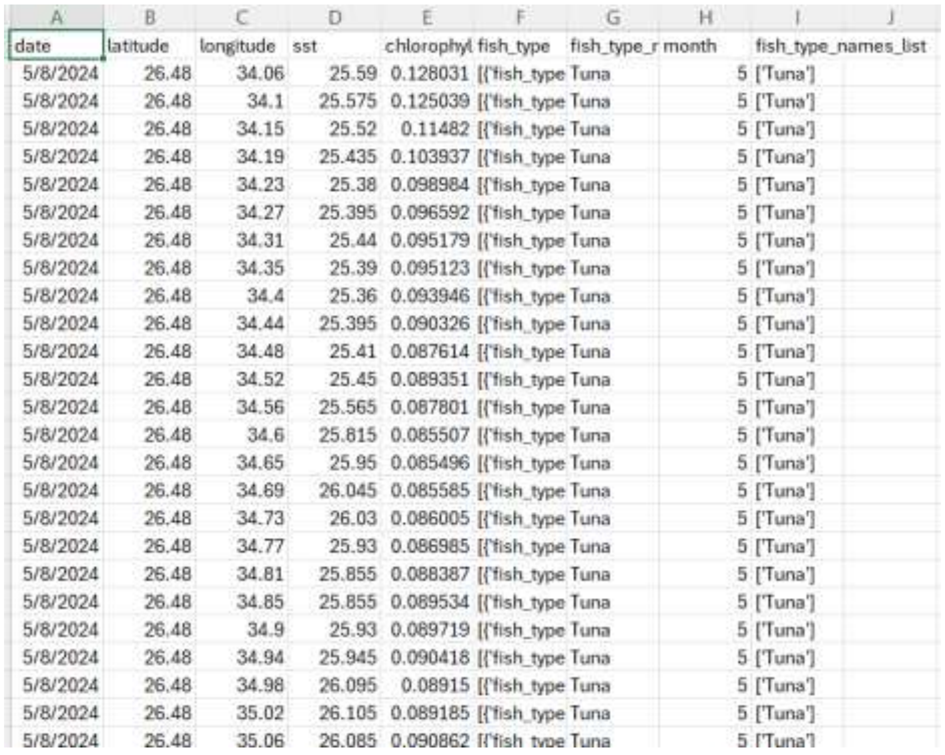
Figure 6.47: Full Python Code for Sensor-AI Interface3



```python
            continue

        red = frame[:, :, 2].astype(float)
        nir = frame[:, :, 0].astype(float)
        ndvi = (nir - red) / (nir + red + 1e-5)
        ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
        avg_ndvi = np.nanmean(ndvi_filtered)
        chlor_a = round(3.5106 * (avg_ndvi ** 2) + 8.3298 * avg_ndvi + 0.601, 4)

        fish_type, qty = classify_fish(sst, chlor_a)

        with open(csv_path, mode='a', newline='') as f:
            writer = csv.writer(f)
            writer.writerow([now, lat, lon, sst, chlor_a, fish_type, qty])

        print(f"[{i+1}/{num_points}] {now} | Lat: {lat}, Lon: {lon} | SST: {sst}, Chl-a: {chlor_a} | Fish: {fish_type} ({qty})")

        if cv2.waitKey(1) & 0xFF == ord('m'):
            print("Stopped manually.")
            break

        time.sleep(interval)

gps_serial.close()
dht_sensor.exit()
cv2.destroyAllWindows()
print(f"\nData collection complete. File saved in: {csv_path}")
```

Figure648: Full Python Code for Sensor-AI Interface4

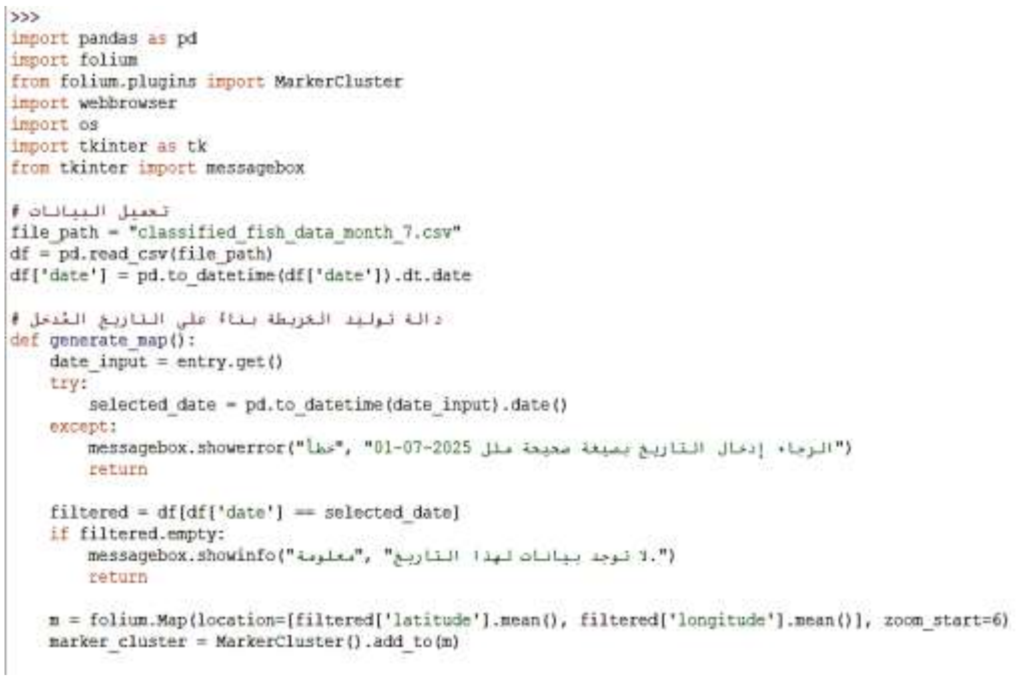
APPENDIX B: SAMPLE OUTPUT CSV FILE



A	B	C	D	E	F	G	H	I	J
date	latitude	longitude	sst	chlorophyl	fish_type	fish_type_r	month	fish_type_names_list	
5/8/2024	26.48	34.06	25.59	0.128031	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.1	25.575	0.125039	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.15	25.52	0.11482	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.19	25.435	0.103937	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.23	25.38	0.098984	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.27	25.395	0.096592	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.31	25.44	0.095179	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.35	25.39	0.095123	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.4	25.36	0.093946	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.44	25.395	0.090326	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.48	25.41	0.087614	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.52	25.45	0.089351	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.56	25.565	0.087801	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.6	25.815	0.085507	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.65	25.95	0.085496	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.69	26.045	0.085585	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.73	26.03	0.086005	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.77	25.93	0.086985	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.81	25.855	0.088387	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.85	25.855	0.089534	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.9	25.93	0.089719	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.94	25.945	0.090418	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.98	26.095	0.08915	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	35.02	26.105	0.089185	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	35.06	26.085	0.090862	[{'fish_type	Tuna	5	['Tuna']	

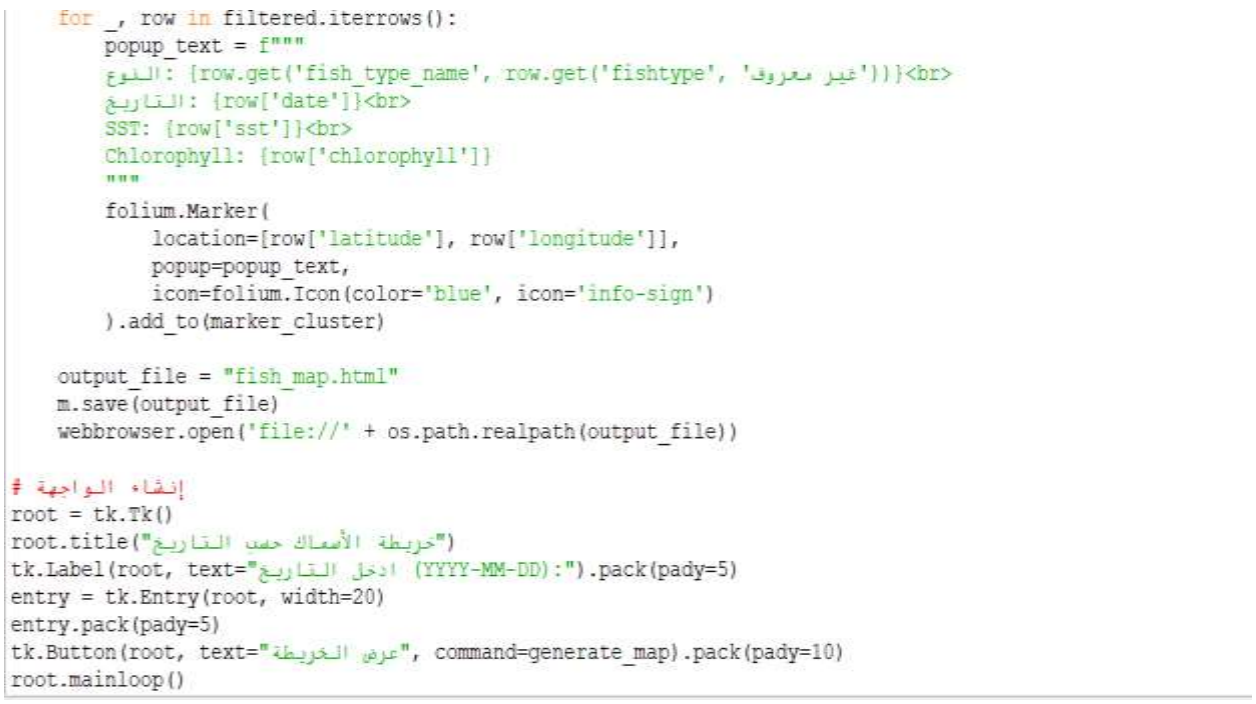
Figure 6.49: Sample Output CSV File

Appendix C: Python Code for GUI Visualization



```
>>>
import pandas as pd
import folium
from folium.plugins import MarkerCluster
import webbrowser
import os
import tkinter as tk
from tkinter import messagebox

# تحميل البيانات
file_path = "classified_fish_data_month_7.csv"
df = pd.read_csv(file_path)
df['date'] = pd.to_datetime(df['date']).dt.date

# دالة توليد الخريطة بناء على التاريخ المدخل
def generate_map():
    date_input = entry.get()
    try:
        selected_date = pd.to_datetime(date_input).date()
    except:
        messagebox.showerror("خطأ", "الرجاء إدخال التاريخ بصيغة صحيحة مثل 01-07-2025")
        return

    filtered = df[df['date'] == selected_date]
    if filtered.empty:
        messagebox.showinfo("معلومة", "لا توجد بيانات لهذا التاريخ.")
        return

    m = folium.Map(location=[filtered['latitude'].mean(), filtered['longitude'].mean()], zoom_start=6)
    marker_cluster = MarkerCluster().add_to(m)
```

Figure 6.50: Python Code for GUI Visualization1



```
    for _, row in filtered.iterrows():
        popup_text = f"""
        النوع: {row.get('fish_type_name', row.get('fishtype', 'غير معروف'))}<br>
        التاريخ: {row['date']}<br>
        SST: {row['sst']}<br>
        Chlorophyll: {row['chlorophyll']}
        """
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=popup_text,
            icon=folium.Icon(color='blue', icon='info-sign')
        ).add_to(marker_cluster)

    output_file = "fish_map.html"
    m.save(output_file)
    webbrowser.open('file://' + os.path.realpath(output_file))

# إنشاء الواجهة
root = tk.Tk()
root.title("خريطة الأسماك حسب التاريخ")
tk.Label(root, text="ادخل التاريخ (YYYY-MM-DD):").pack(pady=5)
entry = tk.Entry(root, width=20)
entry.pack(pady=5)
tk.Button(root, text="عرض الخريطة", command=generate_map).pack(pady=10)
root.mainloop()
```

Figure 6.51: Python Code for GUI Visualization2

Chapter (7): ***Structural Design***

7.1 OVERVIEW

Before designing any CubeSat, we must ask ourselves what its purpose is and what this CubeSat can and cannot bear. The purpose of the CubeSat must be compatible with its shape, size, mass, and with the material from which it is made. There are multiple shapes, different dimensions, and a different arrangement of those dimensions that differ from one CubeSat to another, and the first and last consideration for those dimensions is what this cube will contain in terms of payload, thermal system, and energy, whether it is batteries, a solar system, or something else.

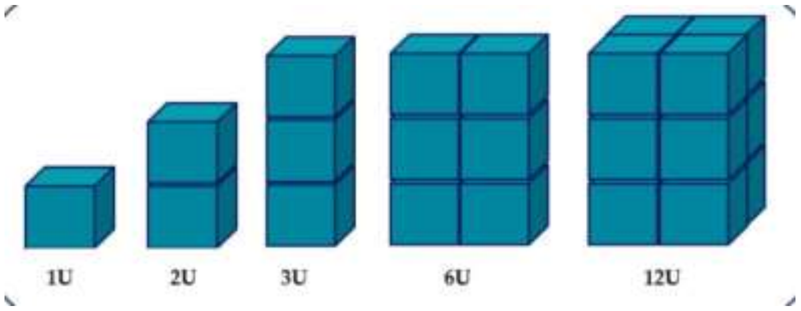


Figure 7.1: Sizes of Cubesats

CubeSat Design and Deployable Cell System

<u>First:</u> The CubeSat

This project designed a 2U CubeSat. This size was chosen to provide sufficient space for the installation of imaging and processing systems, as well as the electrical system and solar cells.

The CubeSat consists of a metal structure made of 6061 aluminum alloy, which has excellent mechanical properties, including light weight, good corrosion resistance, and the ability to withstand launch conditions and the orbital environment.

For the prototype phase, 1050 aluminum alloy was used due to its good optical characteristics. This alloy has a high reflectivity to both light and heat, which helps reduce thermal radiation absorption and protects internal components from overheating. It is also easy to form and manufacture, making it suitable for ground-based testing and prototyping.

General Specifications:

Overall Size: 2U

Dimensions: 227 mm x 100 mm x 100 mm (per unit, total 2U)

Chassis: Made of 6061 aluminum with a design that allows for the installation of deplorable systems Aluminum 1050 was used in the prototype model.

Main Tasks: Sea surface imaging and onboard image processing using artificial intelligence, with the aim of contributing to environmental sustainability by analyzing the distribution of marine organisms.

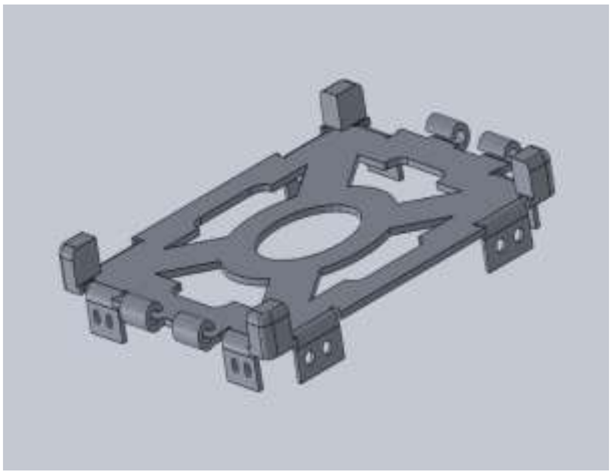


Figure 7.2: Top

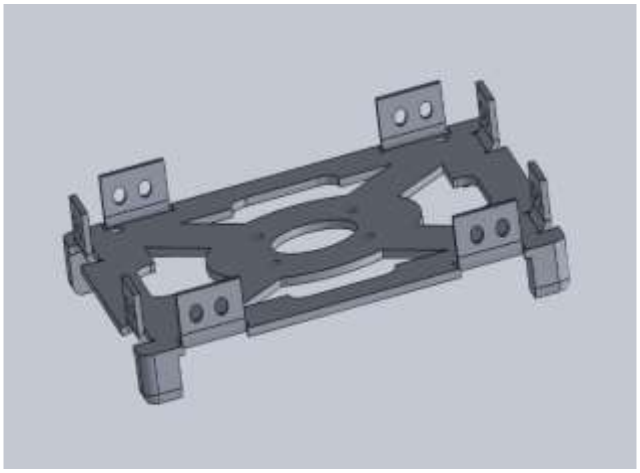


Figure 7.3: Bottom

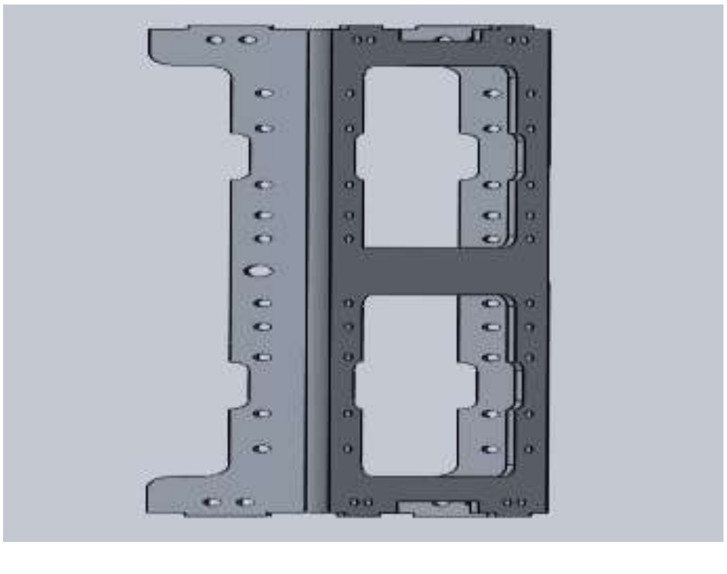


Figure 7.4: Side

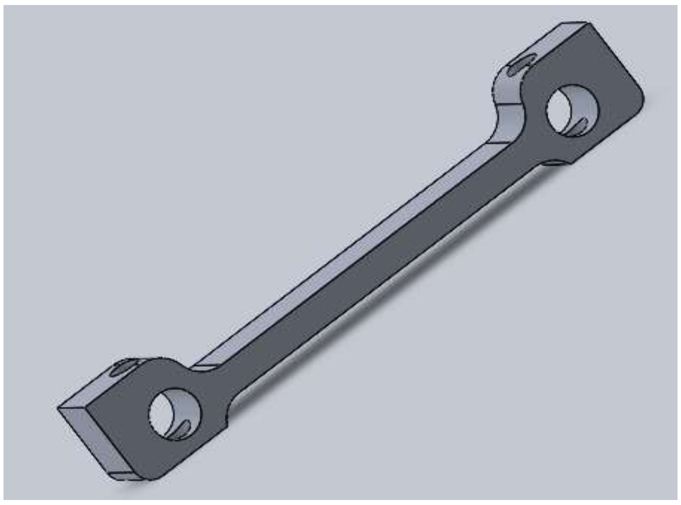


Figure 7.5: Support

INTRODUCTION

Given the limited surface area available on CubeSats, a deployable solar cell system was designed to increase the area available for solar radiation absorption, thereby improving power generation efficiency.

The system relies on a simple and effective mechanical mechanism consisting of torsion springs and a thermal cutout mechanism using an electrical resistor and nylon thread.

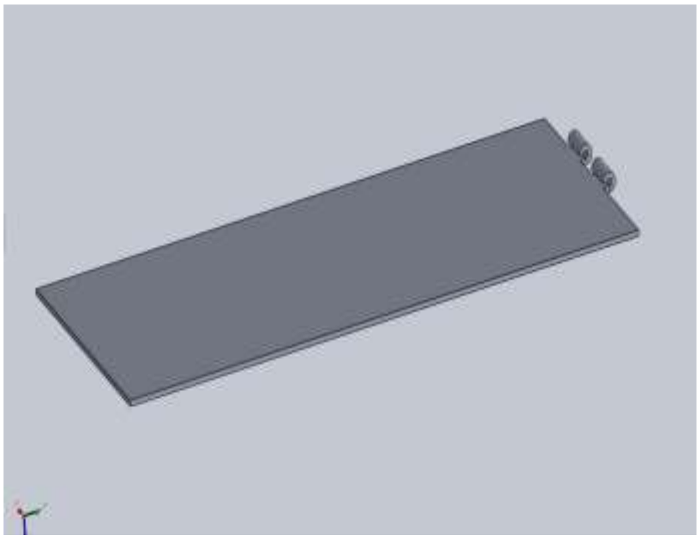


Figure 7.6: Solar panel

7.2 SYSTEM COMPONENTS

Component Description

Spring Type:	Torsion Spring
Number of Springs:	4 (2 for each solar panel)
Number of Panels:	2 openable solar panels

Opening Angle:	90° for each panel
Mounting Method:	Nylon thread (thermally disconnected upon reaching orbit)
Release Mechanism:	Power Resistor to generate heat and disconnect the thread

Spring Selection for the Deployable Solar Panel System in CubeSat

In the design of deployable solar panels in a CubeSat, a spring is used as a passive mechanical mechanism to ensure the automatic deployment of the panels once released. To ensure the effectiveness and reliability of this mechanism in the harsh space environment, the spring material must be carefully selected to withstand extreme conditions.

Suitable Materials for Springs Used in Space Applications

It is essential to use special materials that can withstand the extreme environmental conditions of space, such as very low or high temperatures, vacuum, and radiation. The recommended materials include:

<u>**Recommended Materials:**</u>

1-Beryllium Copper (BeCu)

2-Stainless Steel (Types 302 or 316)

3-Elgiloy Alloy (a high-performance alloy composed of cobalt, chromium, and iron)

<u>**Advantages of These Materials:**</u>

1. Thermal Stability:

These materials retain their mechanical properties across a wide temperature range (–150°C to +200°C), making them ideal for rapid thermal cycling in orbit.

2. Corrosion Resistance:

They are highly resistant to oxidation and corrosion caused by vacuum and solar radiation, ensuring long-term performance

3. Non-Magnetic Properties:

Materials like BeCu and Elgiloy are non-magnetic, protecting magnetic sensors from interference.

4. High Fatigue Life:

These materials can endure repeated stress cycles (compression and release) without losing elasticity or experiencing fracture.

5. Low Outgassing:

In vacuum environments, materials with low outgassing are essential to prevent contamination of optical or sensor components. These materials meet that requirement.

Figure 7.7: Torsion Springs

7.3 COMPARISON BETWEEN TORSION SPRING PLACEMENT OPTIONS

Each deployable solar panel in the design two torsion springs, making a total of four springs for the system. Two main options were considered for placing the springs on each panel: at the edges or near the center (hinge axis). The following table summarizes the comparison between both configurations:

Comparison Table:

Aspect	Springs at the Edges	Springs at the Center
Mechanical distribution	Balanced across the panel's length	Concentrated near the rotation axis
Installation complexity	Requires additional mounting points	Simpler and more centralized mounting
Space consumption	Occupies more lateral space on the CubeSat	More compact design
Opening performance	Ideal for long or heavy panels	Suitable for lightweight panels
Risk of panel bending	Very low	Negligible due to light panel weight

7.4 DESIGN DECISION

Given that the deployable solar panels in this project are lightweight and made from aluminum 6061, the decision was made to use a single torsion spring positioned at the center of each panel, directly aligned with the hinge axis.

This configuration offers the following advantages:

- Provides sufficient torque to open the panel at a 90° angle without the need for multiple support points.
- Simplifies the mechanical design and eases the installation process.

Does not cause any bending or deformation in the panel due to its low weight

7.5 Spring Design Calculations – CubeSat Solar Panel Deployment

Given:

• Mass of solar panel (m) = 85 g = 0.085 kg

• Gravitational acceleration (g) = 9.81 m/s^2

• Length of panel (r) = 207 mm = 0.207 m

• Opening angle (θ) = 90° = 1.57 rad

• Material: 6061 aluminum

• Young's Modulus (E) = 193 × 10^9 Pa

• Number of active coils (N) = 5

• Wire diameter (d) = 5 mm = 0.005 m

Force due to Gravity:

$F = m \times g = 0.085 \times 9.81 = 0.83385$ N

Required Torque:

$\tau = F \times r = 0.83385 \times 0.207 = 0.1726$ N·m

Mean Coil Diameter (D):

Using the torsion spring formula:

$\tau = (E \times d^4 \times \theta) / (10.8 \times D \times N)$

Rearranging to find D:

$D = (E \times d^4 \times \theta) / (10.8 \times \tau \times N)$

$D = (193\times10^9 \times (0.005)^4 \times 1.57) / (10.8 \times 0.1726 \times 5)$

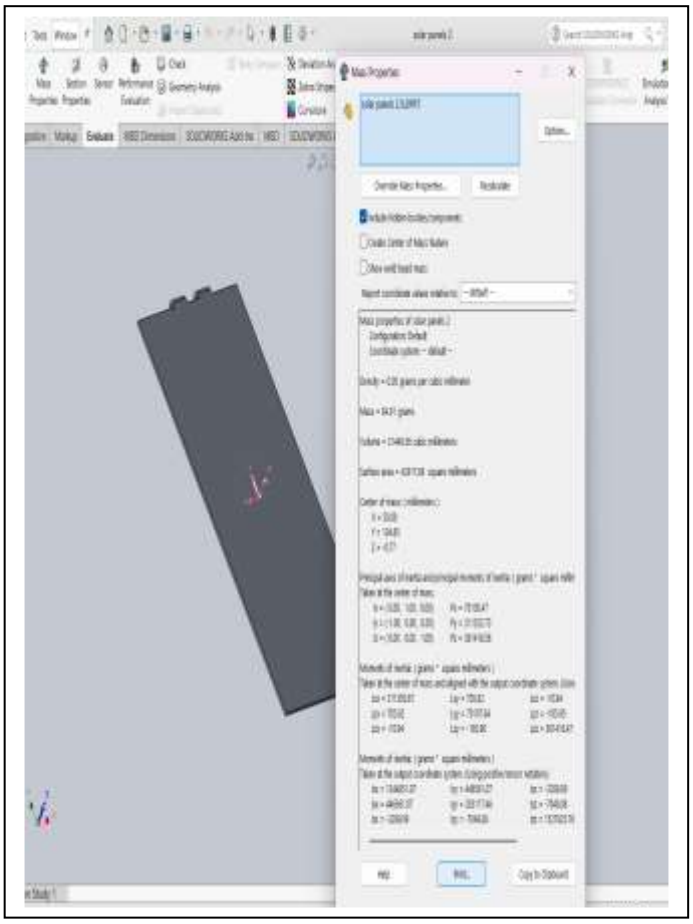


Figure 7.8: Spring Design Calculations

D = 203.5 mm

Inner Coil Diameter:

D_inner = D - d = 203.5 - 5 = 198.5 mm

Final Results:

Parameter	Value
Panel mass	85 g
Required torque	0.1726 N·m
Opening angle	90° (1.57 rad)
Number of active coils	5
Wire diameter	5 mm
Mean spring diameter (D)	203.5 mm
Inner spring diameter	198.5 mm

7.6VERIFICATION OF DESIGN EFFICIENCY THROUGH SIMULATION

7.6.1 THERMAL SIMULATION ANALYSIS

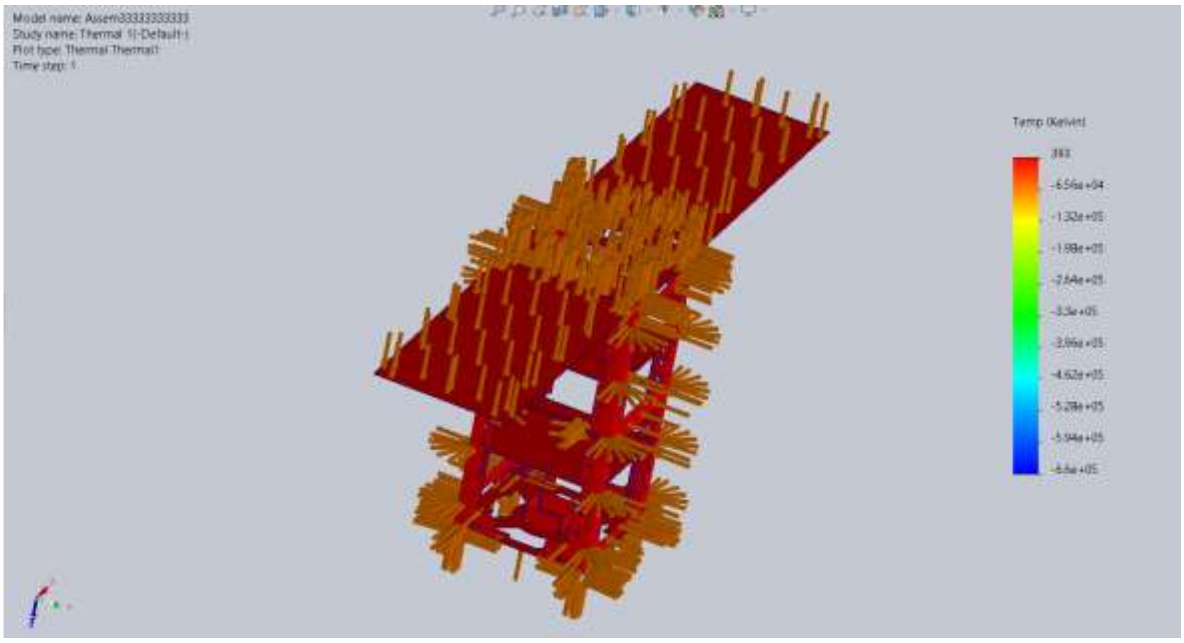

Model name: Assem3333333333
Study name: Thermal 1(-Default-)
Plot type: Thermal Thermal1
Time step: 1

Figure 7.9: Thermal Simulation Analysis1Description

This image presents an initial thermal simulation of the CubeSat model using SolidWorks. The purpose of this analysis is to study the temperature distribution across the model when exposed to the space environment. The color scale on the right indicates the temperature distribution in Kelvin, ranging from -1e-16 to +1e-16.

Observation:

The results show a relatively uniform thermal distribution, indicating that there are no critical hotspots or extreme thermal gradients in the model.

This stage represents a preliminary test of the thermal setup and helps assess the general thermal behavior of the structure before applying more realistic conditions

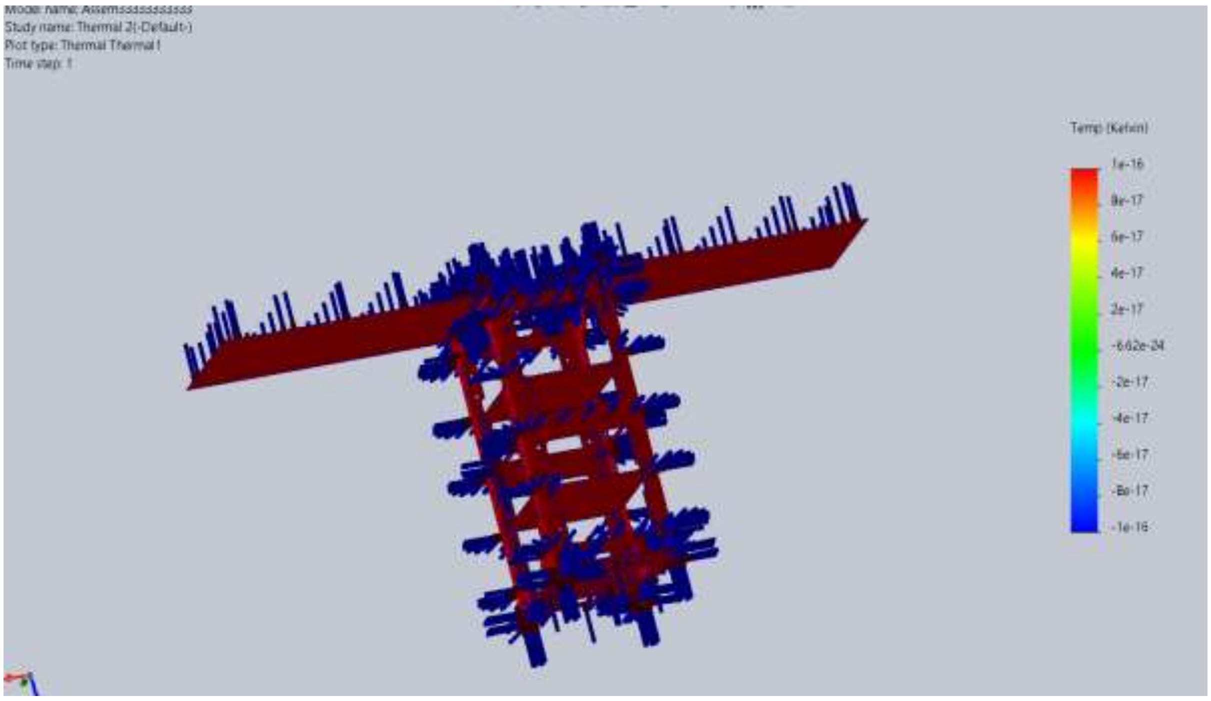


Figure 7.10: Thermal Simulation Analysis2

Description:

In this image, a more advanced thermal analysis was conducted under realistic environmental conditions to simulate the satellite's exposure in a Sun-Synchronous Orbit (SSO). The color scale displays the variation in temperature, with red representing the highest temperature (up to 393 Kelvin) and blue indicating the lowest (down to approximately -6.6e+05 Kelvin).

Observation:

The results show that most of the outer structure maintained a relatively high temperature, while certain internal regions experienced a significant drop.

This helps determine whether additional thermal insulation or internal cooling systems are needed to protect sensitive components such as batteries or the onboard camera.

7.6.2 VON MISES STRESS DISTRIBUTION

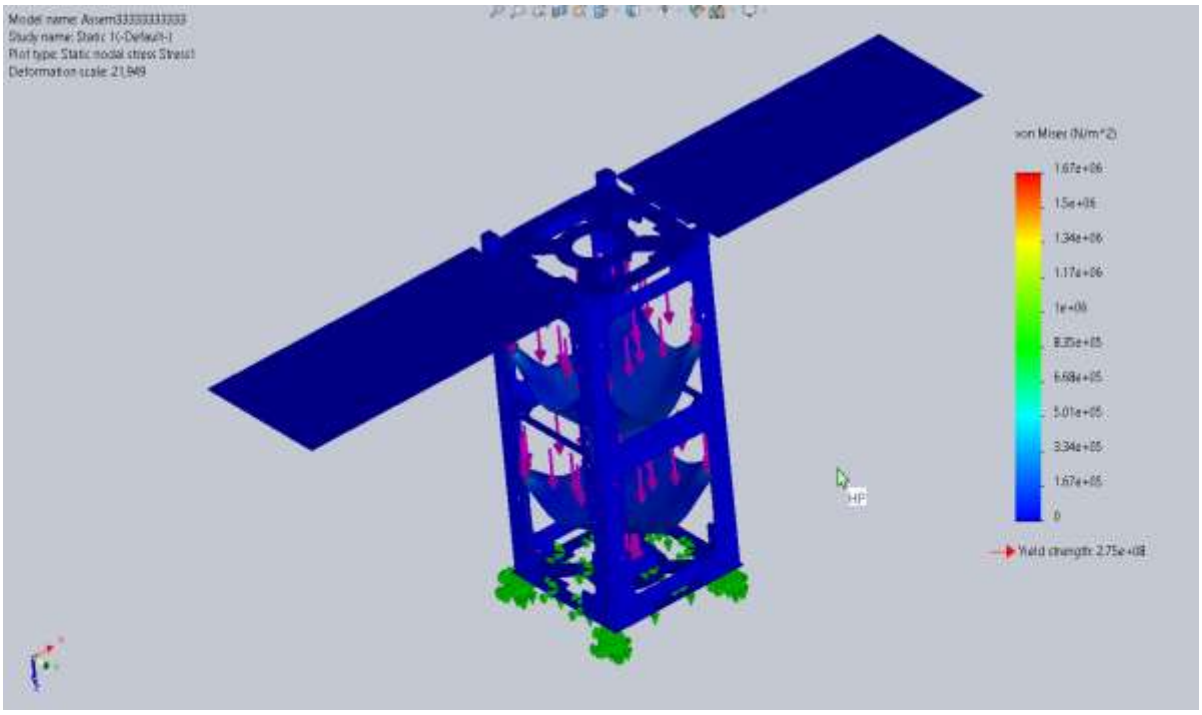

Model name: Assem333333333
Study name: Static 1(-Default-)
Plot type: Static nodal stress Stress1
Deformation scale: 21.949
von Mises (N/m^2)
1.67e+06
1.5e+06
1.34e+06
1.17e+06
1e+06
8.35e+05
6.68e+05
5.01e+05
3.34e+05
1.67e+05
0
Yield strength: 2.75e+08

Figure 7.11: Von Mises Stress Distribution

Description:

The first image presents the distribution of Von Mises stress (N/m²) across the satellite structure. The color scale ranges from 0 (blue) to a maximum of approximately 1.67×10^6 N/m² (red). The maximum stress observed in the structure is significantly below the material yield strength (2.75×10^8 N/m²), which indicates that the structure is operating well within safe limits, with no expected plastic deformation. Stress concentrations appear around the joints and supporting points, which is consistent with expected load paths.

7.6.3 STATIC STRAIN DISTRIBUTION

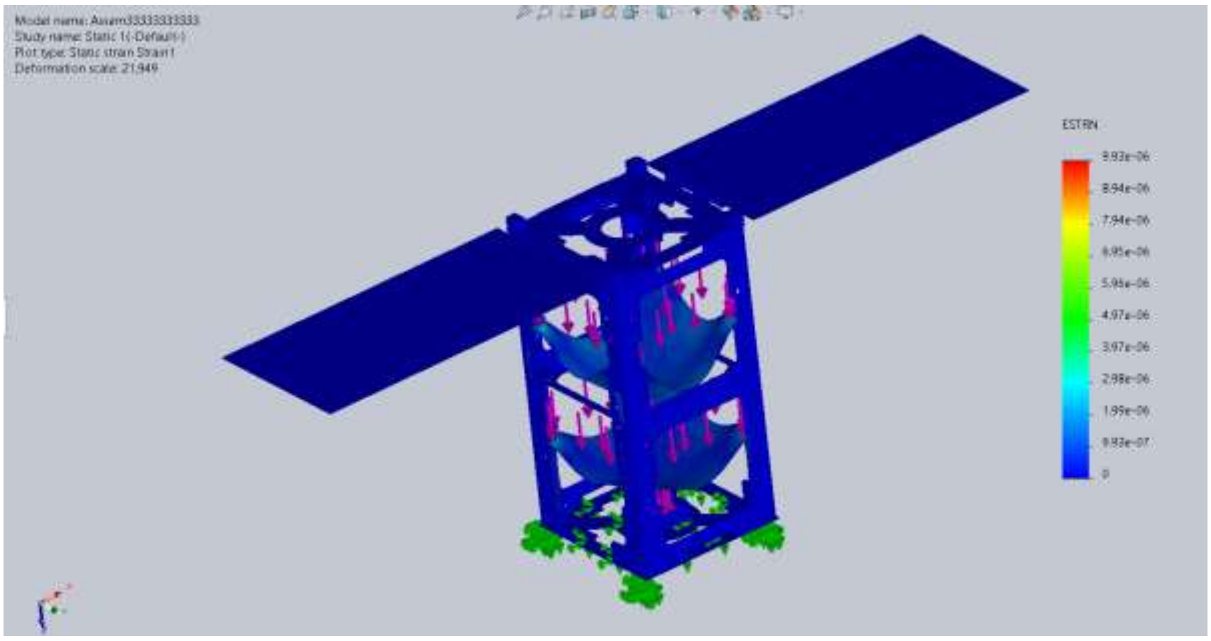

Model name: Assem3333333333
Study name: Static 1(-Default-)
Plot type: Static strain Strain1
Deformation scale: 21,949
ESTRN
9.93e-06
8.94e-06
7.94e-06
6.95e-06
5.96e-06
4.97e-06
3.97e-06
2.98e-06
1.99e-06
9.93e-07
0

Figure 7.12: Static Strain Distribution

Description:

The second image displays the equivalent strain (ESTRN) distribution, with a maximum strain value of approximately 9.91 × 10^{-6}. The low strain values across the model indicate minimal deformation, reflecting a high stiffness in the frame. The deformation scale was exaggerated (21,949 times) for visualization purposes. Areas of highest strain are correlated with regions of stress concentration, again mainly around the fixed supports and connection points.

7.6.4 FACTOR OF SAFETY (FOS) DISTRIBUTION

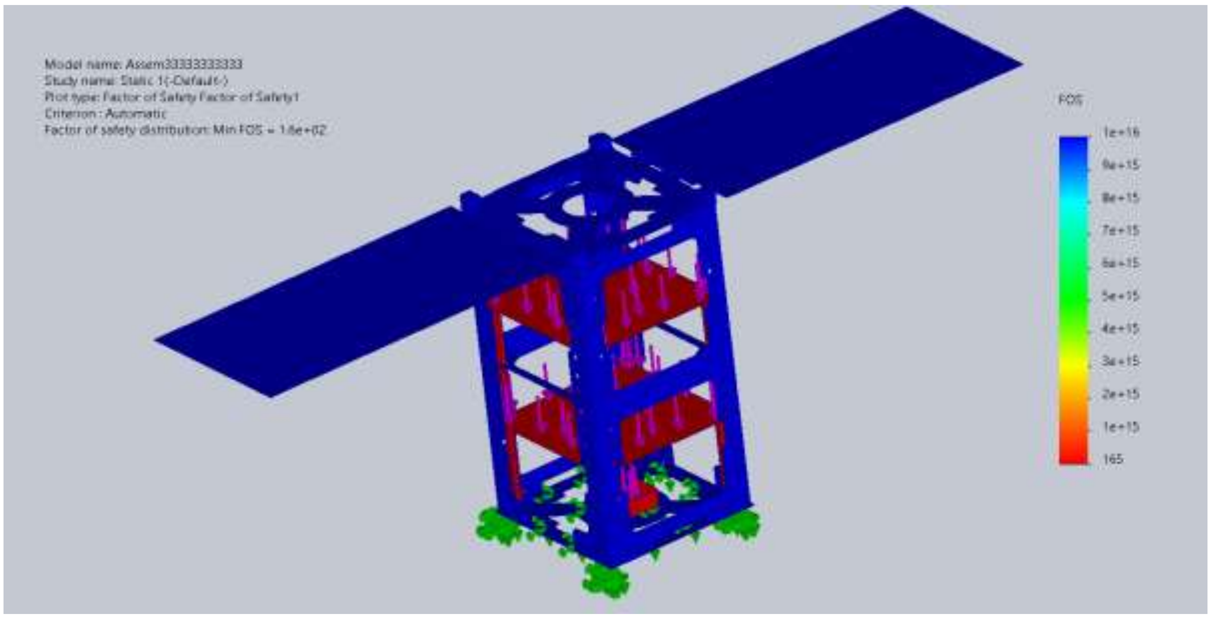

Model name: Assem3333333333
Study name: Static 1(-Default-)
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 1.6e+02
FOS
1e+16
9e+15
8e+15
7e+15
6e+15
5e+15
4e+15
3e+15
2e+15
1e+15
165

Figure 7.13: Factor of Safety (FOS) Distribution

Description:

This figure illustrates the Factor of Safety (FOS) distribution throughout the CubeSat structure. The analysis was conducted using static load condition.

The minimum FOS recorded is approximately 160, which is extremely high and indicates that the structure is significantly overdesigned for the applied load.

The FOS distribution ranges from 1.6e+02 up to 1e+16, as shown on the color bar.

The entire structure remains well within safe limits, with no indication of potential failure zones.

This high FOS ensures that the CubeSat frame is mechanically reliable, even under unforeseen extreme loads.

7.6.5 STATIC DISPLACEMENT DISTRIBUTION

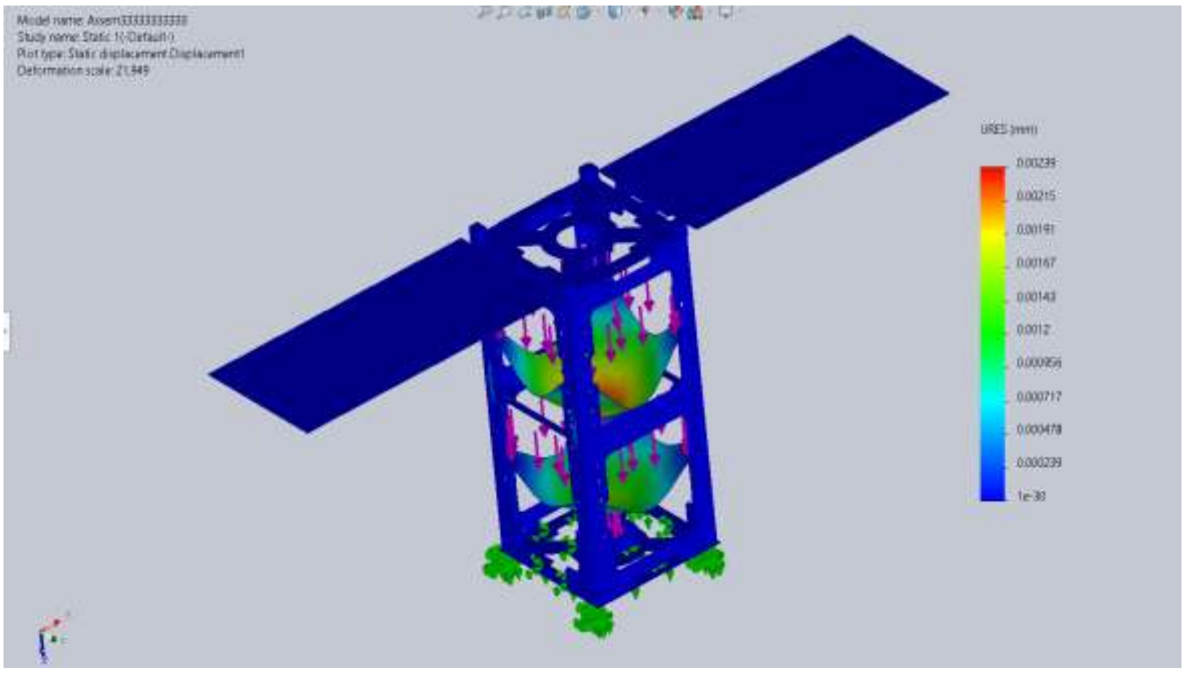

Model name: Assem333333333
Study name: Static 1(-Default-)
Plot type: Static displacement Displacement1
Deformation scale: 21.949
URES (mm)
0.00239
0.00215
0.00191
0.00167
0.00143
0.0012
0.000956
0.000717
0.000478
0.000239
1e-33

Figure 7.14: Static Displacement Distribution

Description:

This figure shows the displacement distribution due to the same static load condition.

The maximum displacement observed is around 0.00239 mm, which is negligible and confirms the rigidity of the structure.

The deformation is mostly localized in the internal electronic component mounts and is very minimal in the outer frame.

The deformation scale in the figure is exaggerated for visualization purposes (scale = 21.949) These results confirm that the CubeSat maintains its geometry and stability during launch or operational stresses.

Chapter (8) : STK Simulation and Coverage Analysis

8.1 INTRODUCTION

This chapter focuses on the simulation and coverage analysis of a CubeSat mission intended to support an AI-based fish detection system. The satellite is expected to pass over specific coastal regions in Egypt, namely the Red Sea coast and the northern shores of the Sinai Peninsula. These areas are targeted for data acquisition related to environmental factors such as sea surface temperature and chlorophyll concentration, which are essential inputs to the AI model.

The simulation aims to evaluate whether the satellite's orbit allows sufficient and timely coverage of these regions to enable reliable data collection. Using the Systems Tool Kit (STK), an orbital scenario was created to simulate satellite movement, sensor configuration, and regional coverage. This chapter will present the steps followed in the simulation process, the justification for orbital parameters, the method for defining the target area, and the results derived from coverage reports. The analysis is crucial to confirm whether the satellite configuration meets the mission's observational requirements.

8.1.1 MISSION OVERVIEW

The CubeSat mission presented in this project is designed to support a smart environmental monitoring system that predicts potential fish aggregation zones in Egyptian coastal waters. The prediction is performed using artificial intelligence (AI) models that rely on the analysis of satellite-derived environmental indicators such as sea surface temperature (SST) and chlorophyll-a concentration, both of which are highly correlated with fish behavior and distribution.

To provide the necessary input data for the AI model, a low-cost and lightweight CubeSat is proposed to carry relevant environmental sensors capable of capturing thermal and optical signals over defined marine regions. Specifically, the mission targets two key coastal zones: the Red Sea coastline of Egypt and the northern coast of the Sinai Peninsula. These areas are considered biologically productive and economically significant due to their rich marine ecosystems and active fishing operations.

The role of the CubeSat is to pass over the target areas and collect environmental data consistently and accurately. For this to happen, careful orbital planning and simulation are essential. The satellite must be placed in an orbit that ensures regular revisit times and sufficient observation coverage of the selected areas. Therefore, a Sun-Synchronous Orbit (SSO) is chosen for its ability to provide consistent lighting conditions and near-daily passes over fixed locations.

of the simulation in this chapter is to assess whether the chosen orbit and satellite configuration are sufficient to meet the mission's requirements. This includes verifying the satellite's visibility over the Red Sea and Sinai, evaluating the duration and frequency of coverage, and determining if the temporal and spatial coverage are adequate to provide useful environmental data for AI-based processing.

The STK-based simulation evaluates the mission scenario over a 24-hour period and uses various analysis tools such as Coverage Definition, Access Reports, and Figures of Merit to generate meaningful metrics about performance. The outcomes of this simulation serve as a critical input to validate the feasibility of the CubeSat mission in supporting an autonomous fish detection platform.

CubeSats are taking an increasingly significant role in the Space Economy, and it is very important to understand the causes of failures and anomalies in order to improve the designs of future missions , In order to enhance the functionality of spacecraft, such as decreasing communication expenses and enabling navigation, machine learning applications are being applied to space data.

8.1.2 STK SOFTWARE OVERVIEW

Systems Tool Kit (STK) is a simulation software developed by AGI (now part of Ansys) that allows engineers and researchers to model, analyze, and visualize satellite missions and ground-based systems in a realistic environment. STK is widely used by organizations such as NASA, ESA, and military institutions due to its ability to simulate real-world physics with high precision.

In simple terms, STK helps you plan and analyze how satellites move, what they can see, and how they interact with specific regions on Earth or other objects in space. It uses real orbital mechanics equations, geographic data, and timing tools to show exactly when and where a satellite will be able to observe a certain area.

The software provides a 3D and 2D interface where you can:

- Create satellites and define their orbits
- Attach sensors with specific field-of-view (FOV) settings
- Define ground regions or targets (called "Area Targets")
- Simulate when a satellite has "access" to those areas
- Analyze how long the satellite can see a region, how often, and how well

In this project, STK was used as the main tool to simulate the mission of a CubeSat that collects environmental data such as sea surface temperature and chlorophyll levels — data that is later analyzed by an AI model to predict fish presence. Since the data collection depends entirely on the satellite passing over the right regions (specifically the Red Sea and northern Sinai), it was crucial to verify whether the CubeSat can provide enough coverage.

In STK-11, I also built my own scenario and defined the mission duration.

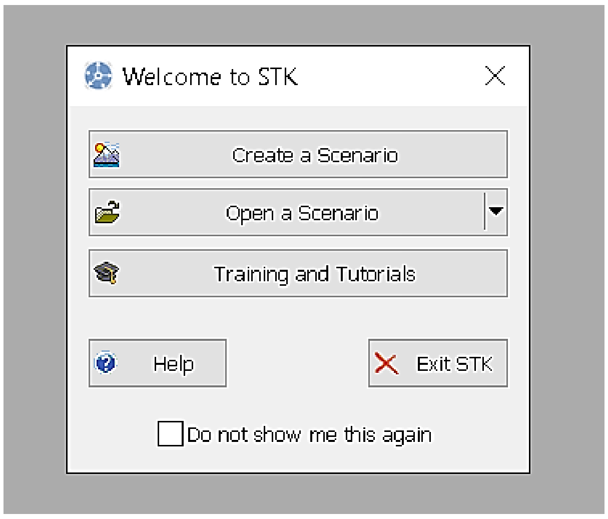


Figure 8.1: build scenario window in STK

It also provides two displays, one for 2D and the other for 3D, **as follows:**

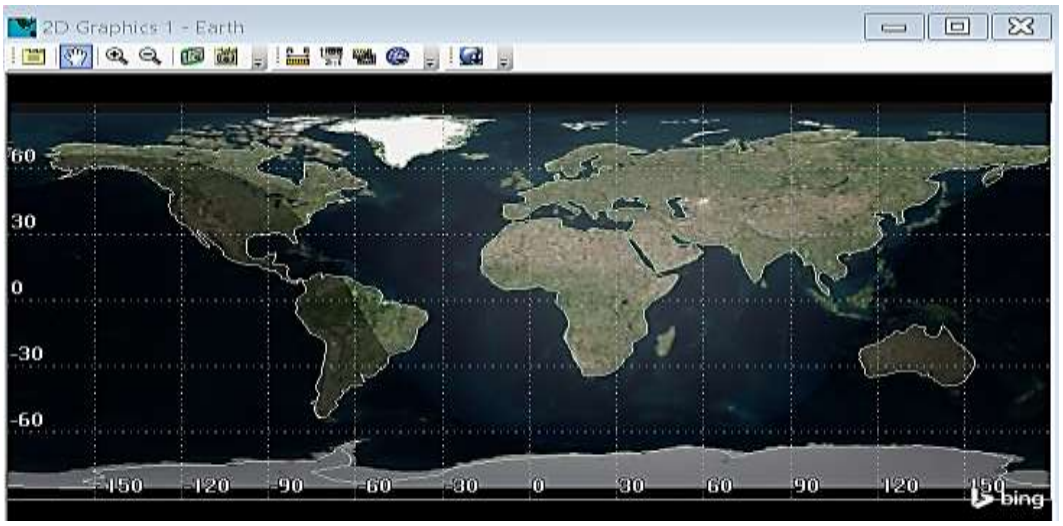


Figure 8.2: 2D graphics view in STK

The 2D view offers a top-down geographic map used for defining regions and analyzing ground tracks and coverage footprints,

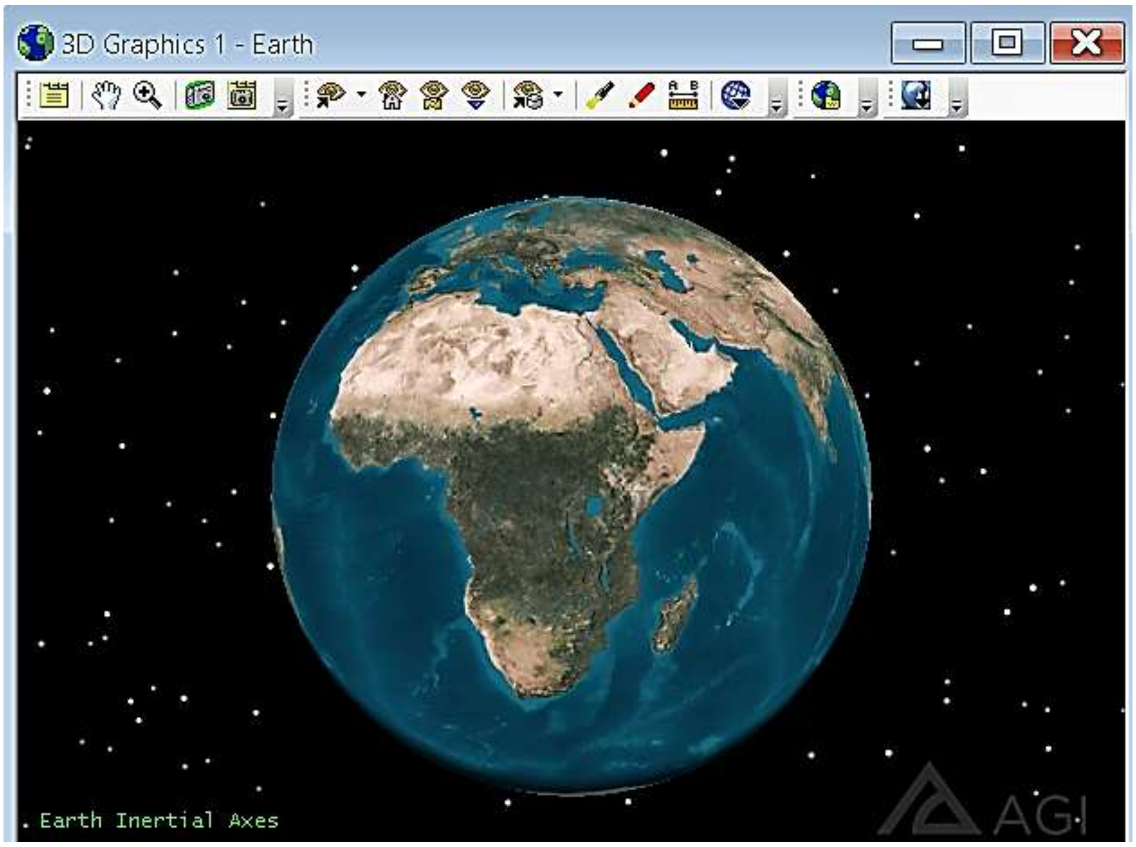


Figure 8.3: 3D graphics view in STK

the 3D view allows for realistic simulation of satellite orbits and sensor fields of view in space, enhancing both technical understanding and visual presentation of the mission.

Using STK11:

1. Created a satellite with a sun-synchronous orbit — chosen because it offers consistent passes over the same region every day.
2. Defined the target regions on Earth using real geographic coordinates.
3. Added a sensor to the satellite to simulate its observation capability.
4. Used the Coverage Definition tool to break the area into grid points and analyze how much of it the satellite could cover over a 24-hour period.
5. Generated reports such as Percent Coverage and Access Intervals to evaluate the frequency and quality of the coverage.

Through this process, I was able to determine how many times per day the satellite would pass over the region, how long each pass would last, and whether the total coverage would be enough to collect reliable data for the AI model. STK allowed me to simulate the mission realistically and evaluate its performance before any real-world deployment.

8.2 METHODOLOGY

This section outlines the step-by-step simulation procedure followed to evaluate the CubeSat's ability to cover the target marine regions using STK software. The methodology includes defining the region of interest, selecting a suitable orbit, configuring the satellite and its onboard sensor, and analyzing the spatial and temporal coverage over the selected area.

Each of these steps was implemented within STK through specific tools such as Area Target Definition, Orbit Modeling, Sensor Setup, Coverage Definition, and Access Analysis. The following subsections explain each of these steps in details.

8.2.1 Target Area Definition

The Red Sea coast and the northern coast of the Sinai Peninsula were selected as the primary areas of interest for this simulation due to their environmental and economic importance within Egypt.

These areas are known for their high biodiversity and the presence of numerous commercially valuable fish species, such as tuna and sardines. The Red Sea is one of the most productive marine environments in the region.

In addition to their environmental importance, the two areas are among Egypt's most active fishing grounds and contribute to food security at the local and national levels. However, fish abundance in these areas varies significantly due to changes in water temperature, chlorophyll concentration, and seasonal migration patterns.

Therefore, environmental monitoring using satellite data becomes an essential tool for predicting fish population locations, supporting sustainable fishing practices rather than overfishing, and conserving fish stocks.

To define the target area in STK, geographic coordinates were collected using satellite maps and open-source tools such as NASA Ocean Color. A rectangular boundary was constructed by specifying the northernmost, southernmost, easternmost, and westernmost points enclosing the region of interest.

The following coordinates were used to define the rectangular area:

- Northern boundary: Latitude 33.21°
- Southern boundary: Latitude 20.71°
- Eastern boundary: Longitude 39.90°
- Western boundary: Longitude 29.79°

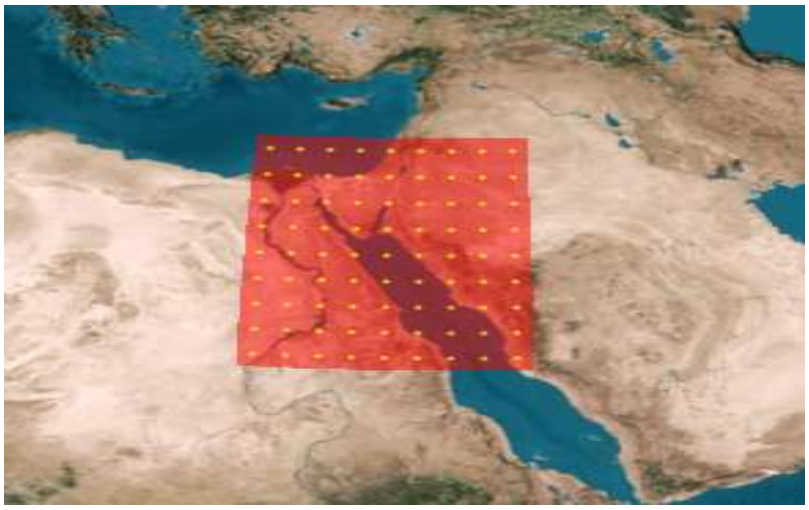


Figure 8.4: defined rectangular target area in STK covering the Red Sea and northern Sinai

These values were input into the STK Area Target object using the tool of **coverage definition**:

- Min latitude: 20.71°

- Min longitude: 29.79°
- Max latitude: 39.90°
- Max longitude: 29.79°

The region was then visualized as a rectangular zone on the Earth's surface in both the 2D and 3D views. This area represents the ground footprint that the satellite's onboard sensor must observe during its passes.

The coverage analysis in later steps will assess how frequently and for how long the CubeSat can observe this target region within the simulation time frame.

8.2.2 ORBIT SELECTION AND JUSTIFICATION

In order to ensure sufficient and regular observational coverage of the defined marine regions (the Red Sea and northern Sinai coasts), an appropriate satellite orbit had to be selected. The choice of orbit directly impacts the frequency of satellite passes over the target areas, the consistency of observation conditions, and the overall success of the mission.

8.2.2.1 ORBIT TYPE SELECTION

The sun-synchronous orbit was selected based on a set of characteristics beneficial to the mission:

Consistent light conditions

- It maintains a near-constant sun illumination angle (Local Time of Descending Node LTDN) during the passage, ensuring uniform image quality for fish observations.
- It eliminates shadow/glare variations that hinder chlorophyll-a and plankton observations in coastal waters.
- Optimal coverage of the Red Sea
- The SSO orbit's 98-degree inclination provides daily coverage of the equatorial latitudes (where the Red Sea is located) with two to three revisits per day.
- The polar coverage ensures no gaps in observations of the northern coast of Sinai.
- Synergy with environmental monitoring
- It is aligned with the orbits of major Earth observation satellites (such as Sentinel-2 and Landsat), enabling data integration for validation.
- It is aligned with the passage times of oceanographic sensors (such as MODIS) for comparisons of sea surface temperature and algal blooms.

Energy Efficiency

- Stable sunlight exposure enhances solar panel power generation, which is critical for the CubeSat's limited battery capacity.
- It reduces the need for active position adjustments, saving fuel.

Fisheries-Specific Advantages

- **Dawn/dusk terminator avoidance**: 10:30 AM overpass captures ideal light for:
    - Penetrating water columns to detect fish schools.
    - Minimizing sunlight that obscures coastal imagery.
- **Predictable revisit times** align with fish migration/feeding patterns (e.g., sardine movements at mid-morning).

8.2.2.2 CUBE-SAT ORBITAL PARAMETERS

synchronous orbit suitable for Earth observation missions. The semi-major axis, inclination, and orientation parameters were chosen to ensure consistent daily passes over the target areas.

The values used are as follows:

- Semi-major axis (a): 7028.14 km
- Inclination (i): 97.8°
- Right Ascension of the Ascending Node (RAAN): 258.973°
- Argument of Perigee (ω): 0°
- True Anomaly (ν): 0°

We chose these parameters based on these calculations:

1. Semi-Major Axis (a) = 7028.14 km

a = R_Earth + altitude = 6378.14 km (Earth radius) + 650 km = 7028.14 km

650 km altitude balances:

- **Resolution:** 5 m/pixel (good for fish school detection)

- **Swath width:** ~100 km (covers Red Sea's 180-360 km width in 2-4 passes)
- **Drag:** Minimal atmospheric resistance (7+ year satellite lifetime)

2. Eccentricity (e) ≈ 0

The orbit is circular, maintaining a constant altitude for the following:

- Unified image resolution.
- Stable solar power (no stress on the battery due to altitude differences).
- Compatible with standard CubeSat missions.

3. Right Ascension of Ascending Node (RAAN) = 258.973°

- Automatically selected by STK to satisfy Sun-Synchronous conditions
- It ensures that the satellite crosses the equator at a fixed local solar time, enabling consistent imaging conditions over time.

4. True Anomaly (ν) = 0°

- Starts measurement at **perigee** (even in circular orbit)
- Simplifies initial state vector calculations

5. Argument of Perigee (ω) = 0°

- Irrelevant for circular orbits (no perigee/apogee distinction)
- Standard practice for SSOs (e.g., Sentinel-2)

6. Inclination (i) = 98°

- Compensates Earth's oblateness to maintain LTD
- Provides full Red Sea coverage (up to 28°N)

8.2.2.3 IMPLEMENTATION IN STK

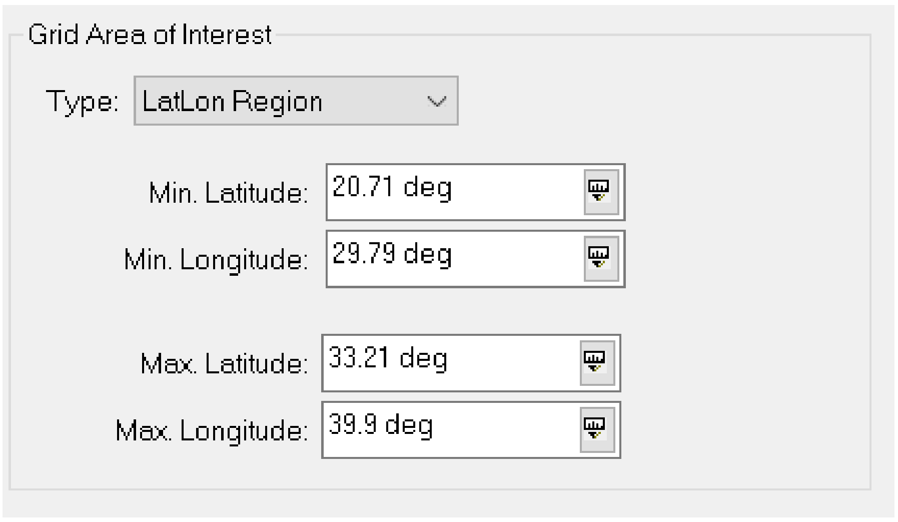
This section explains the practical steps taken to implement the selected orbit into the STK simulation environment. After defining the orbital parameters theoretically, these parameters were applied in STK using the Classical Orbital Elements interface. The process includes configuring the satellite’s altitude, inclination, and orbital orientation to match the mission requirements, ensuring the CubeSat performs the desired passes over the target region accurately.

Implementation Steps :

1. Target Area Definition

The first step

Definition the geographic region of interest, which includes the Red Sea coast of Egypt and the North Sinai coastal region, as this area is relevant for marine monitoring and fish detection by **coverage definition** it called in mission Red_Sea_Area ,this analytical tool is used to analyze a sensor's or satellite's ability to cover a specific area over time. It creates a grid over the area and measures for each point: coverage, number of passes, gap, and revisit time.


Grid Area of Interest

Type: LatLon Region

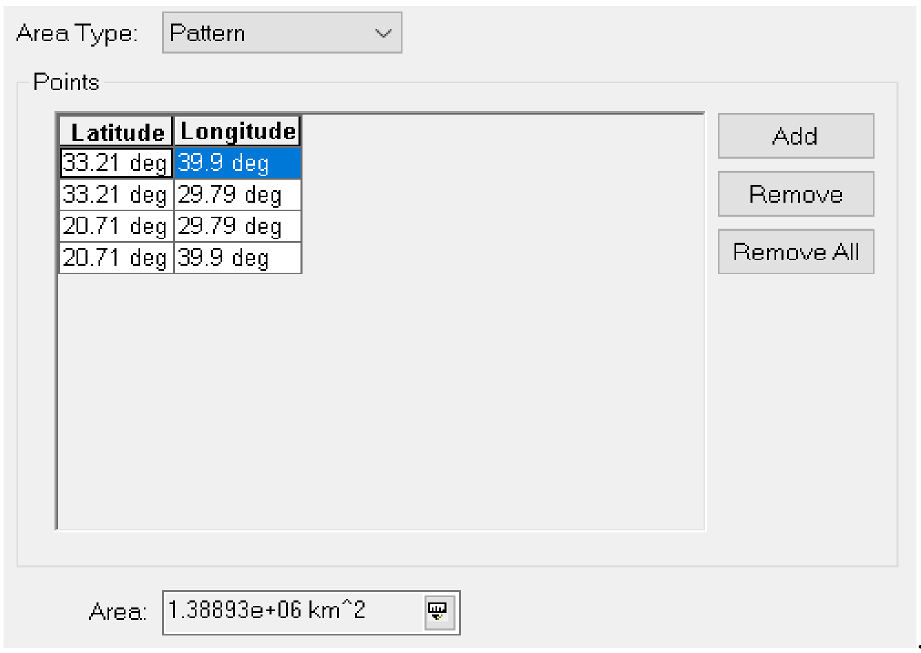
Min. Latitude: 20.71 deg
Min. Longitude: 29.79 deg

Max. Latitude: 33.21 deg
Max. Longitude: 39.9 deg

Figure 8.5: coverage definition in STK

Second step

Definition the geographic region of interest, which includes the Red Sea coast of Egypt and the North Sinai coastal region, as this area is relevant for marine monitoring and fish detection by **Area Target** it called in mission Red_Sea_Area ,this tool is used to define an area on the map for targeting, guidance, or establishing access relationships between it and satellites or sensors. It's just a geometric area you're working with.


Area Type: Pattern

Points

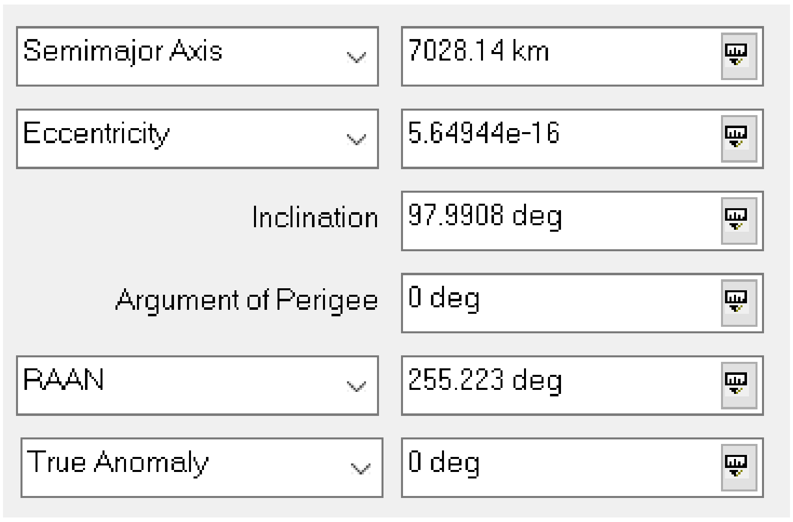
Latitude	Longitude
33.21 deg	39.9 deg
33.21 deg	29.79 deg
20.71 deg	29.79 deg
20.71 deg	39.9 deg

Area: 1.38893e+06 km^2

Figure 8.6: Area Target in STK

2. CubeSat Orbit Definition

The satellite object was created by selecting **"Insert → Satellite"** within the STK environment. The orbit was defined manually using the **Classical Orbit Elements** option in the satellite properties panel.



Semimajor Axis	7028.14 km
Eccentricity	5.64944e-16
Inclination	97.9908 deg
Argument of Perigee	0 deg
RAAN	255.223 deg
True Anomaly	0 deg

Figure 8.7: orbit parameters in STK

These parameters were chosen to form a Sun-Synchronous Orbit (SSO) at an altitude of 650 km, allowing the satellite to have consistent lighting conditions over the target region and ensuring regular coverage.

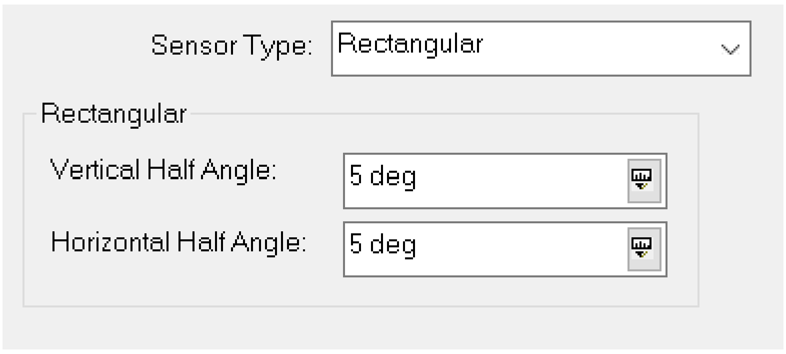
The simulation time frame was configured from **10 June 2025, 09:00:00 UTC to 11 June 2025, 09:00:00 UTC**, allowing for one full day of satellite operation and coverage assessment over the Red Sea area.

Once the parameters were set, the orbit visualization was checked in both the 2D and 3D views, confirming that the satellite's trajectory passes accurately over the defined target region. This step is crucial to verify that the orbit matches the mission requirements before proceeding with sensor configuration and coverage analysis.

3. Sensor Definition and Configuration

A coverage analysis was set up to evaluate whether the current orbit and sensor configuration meet the mission requirements for spatial and temporal coverage.

A rectangular sensor with vertical half angle **5**° and horizontal half angle **5**° was used to match the required spatial resolution for the mission. This resolution is suitable for accurately detecting fish presence patterns and monitoring environmental parameters over the target area while maintaining a balance between coverage area and image quality.



Sensor Type: Rectangular

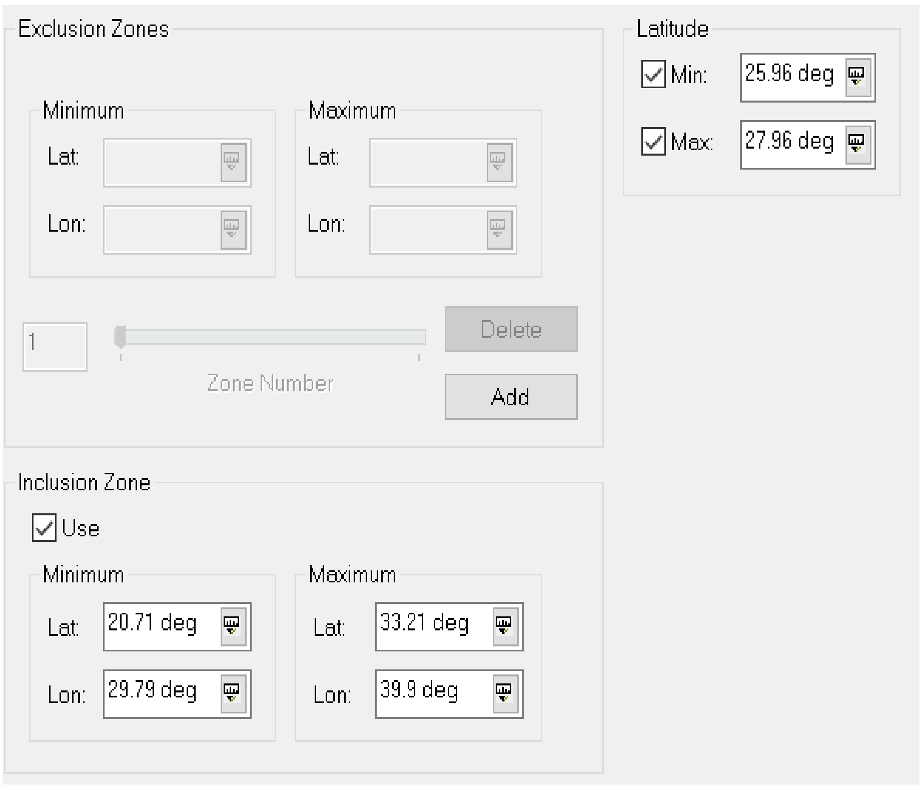
Rectangular

Vertical Half Angle: 5 deg

Horizontal Half Angle: 5 deg

Figure 8.8: Sensor Type in STK

To ensure high-resolution data and minimize geometric distortion, the sensor was designed to maintain continuous focus on the target area, with particular emphasis on nadir pointing. This setup guarantees that the satellite captures images when it is directly above the area, resulting in clearer and more accurate observations


Exclusion Zones

Minimum
Lat:
Lon:

Maximum
Lat:
Lon:

1 Zone Number Delete Add

Inclusion Zone
☑ Use

	Minimum	Maximum
Lat:	20.71 deg	33.21 deg
Lon:	29.79 deg	39.9 deg

Latitude
☑ Min: 25.96 deg
☑ Max: 27.96 deg

.

Figure 8.9: Sensor inclusion zone for pointing to nadir in STK

4. Figure of Merit (FOM) Function

The Figure of Merit (FOM) was added to the coverage analysis to quantify the coverage performance over time.

Purpose of FOM:

- Evaluates how well each point in the target area is covered.
- Measures metrics like Percentage of Coverage, Maximum Gaps, Average Access Time.
- Provides numerical indicators to assess whether the mission achieves its goals.

This systematic implementation in STK provides a solid foundation for analyzing whether the CubeSat can fulfill the mission requirements in terms of spatial and temporal coverage over the target region. The following chapter presents the results and analysis based on this setup.

8.3 SIMULATION RESULTS

The simulation was conducted based on the mission design and configuration described in the previous section. The simulation was conducted to assess the ability of the CubeSat, equipped with the designed sensor and operating in the specified sun-synchronous orbit (SSO), to provide adequate coverage of the selected target area, which includes the Red Sea coast and North Sinai.

The purpose of the simulation is to evaluate the satellite's performance from both spatial and temporal perspectives. From a spatial perspective, it is important to ensure that the sensor effectively covers the entire target area during each pass, or at least within acceptable revisit intervals. This is critical for the mission objective, which relies on collecting high-resolution environmental data to support fish detection based on parameters such as sea surface temperature and chlorophyll concentration.

In addition to spatial coverage, the analysis also evaluates temporal coverage, focusing on the number of times the satellite revisits the target area during the simulation period, as well as the number of revisits. It also emphasizes accurate monitoring of the area through the CubeSat's observations of the area from a distance to ensure the quality of the captured images.

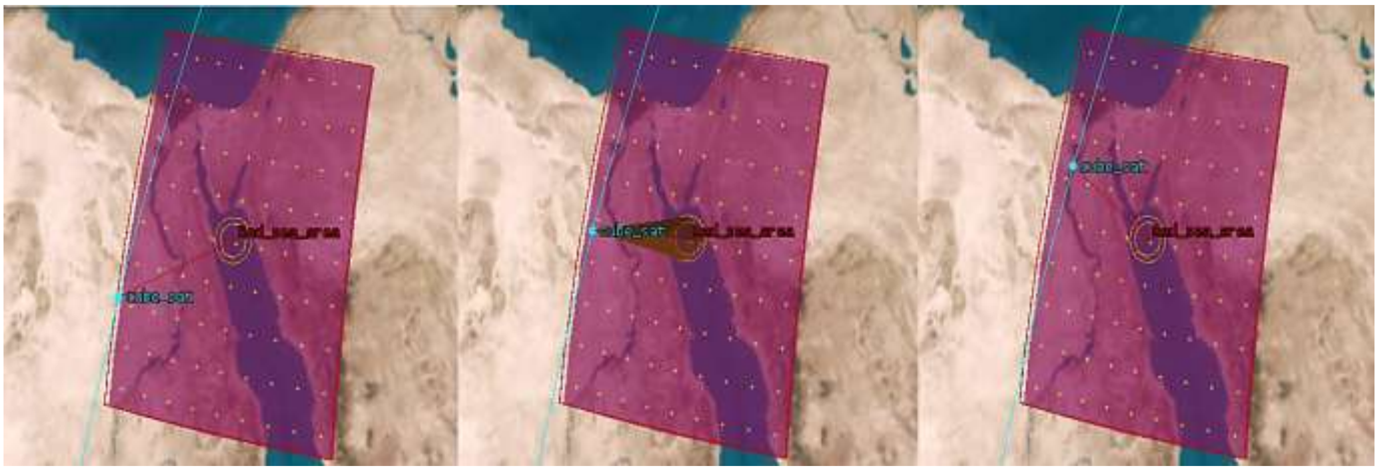


Figure 8.10: Sensor operation restricted to nadir position to guarantee high-resolution imaging

8.3.1 COVERAGE ANALYSIS

The primary purpose of the coverage analysis is to evaluate the CubeSat's ability to provide complete and reliable spatial coverage over the defined target area, which includes the Red Sea and North Sinai coastal regions. This analysis is crucial for verifying whether the satellite and its sensor configuration are capable of capturing sufficient data to support the mission objectives, particularly the detection of fish distributions based on environmental parameters**.**

1. Area Definition Recap

The coverage was performed over the area bounded by the following coordinates:

Boundary	**Value**
North	33.21°
South	20.71°
East	39.90°
West	29.79°

Table-7.1- Area Definition Recap

The area was modeled using the Coverage Definition tool with a Lat/Lon grid resolution of 1.0°, resulting in 81 grid points representing the area.

2. Simulation Period

- Start: 10 June 2025, 09:00:00 UTC
- End: 11 June 2025, 09:00:00 UTC
- Total duration: 24 hours

3. Assets Assigned for Coverage

Asset	**Status**
CubeSat	**Active**
Sensor (Rectangular 5°x5°)	**Active**

Table 7.2- Assets Assigned for Coverage

4. Coverage Conditions

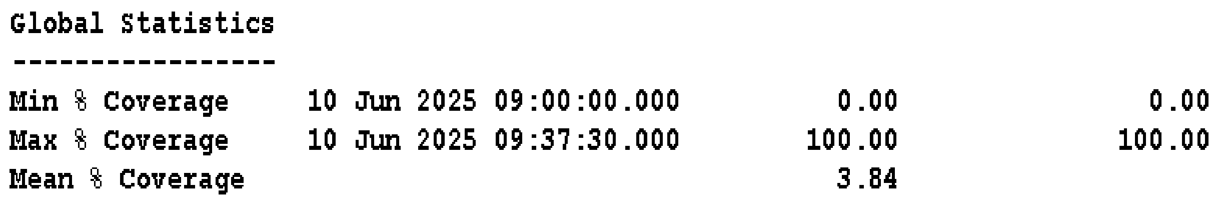
- At least one asset in view for a grid point to be considered covered.
- Sensor configured with nadir-pointing mode, 5°×5°, and Inclusion Zone constraints matching the target area boundaries.

5. Percent Coverage Analysis

Based on the **Percent Coverage Report**, the following values were observed:

Metric	Value
Max Coverage	100%
Min Coverage	0%
Mean Coverage	3.84% (example)

Table 7.3- Percent Coverage Analysis



```
Global Statistics
-----------------
Min % Coverage     10 Jun 2025 09:00:00.000          0.00                   0.00
Max % Coverage     10 Jun 2025 09:37:30.000        100.00                 100.00
Mean % Coverage                                      3.84
```

Figure 8.11: Red_Sea_Area Percent Coverage report in STK

- **Max Coverage = 100%:**

  Indicates that the entire target area was fully covered during certain passes.

- **Min Coverage = 0%:**

  Represents periods when the satellite was out of view of the target area, which is typical due to orbital constraints.

- **Mean Coverage:**

  Reflects the average area coverage across the full 24-hour simulation period. A lower value indicates that while full coverage occurs during passes, the area remains uncovered during the gaps between passes.

6. Full Coverage Events

- Full coverage was achieved **multiple times** within the simulation window.

- Typically occurred every **7–8 hours**, with each full coverage lasting approximately **8 to 12 minutes** depending on the pass geometry.

The coverage analysis confirms that the mission requirements for spatial coverage are met. The CubeSat, with its current orbital configuration and sensor settings, is capable of providing full coverage of the target area multiple times within the simulation period. This ensures that the mission can collect the necessary data for environmental monitoring and fish detection applications.

8.3.2 TEMPORAL COVERAGE

Temporal coverage analysis is essential to evaluate how frequently the CubeSat revisits the target area within the simulation period. This directly impacts the mission's ability to collect data regularly and monitor dynamic changes in environmental conditions relevant to fish detection. The analysis assesses pass frequency, revisit time, and access duration to determine whether the satellite meets the temporal requirements of the mission.

1. **Pass Frequency (Number of Passes Per Day)**

- Based on the simulation, the CubeSat passes over the target area approximately **3 times per day** within the 24-hour simulation period.
- This frequency is typical for satellites in **Sun-Synchronous Orbits (SSO)** at **650 km altitude**, providing regular revisits while maintaining consistent lighting conditions.

2. **Revisit Time Analysis**

Pass	**Start Time (UTC)**	**End Time (UTC)**	**Duration (minutes)**
1	09:37	09:46	~9 mins
2	11:12	11:20	~8 mins
3	18:55	19:03	~8 mins

Table 7.4 -Revisit Time Analysis

The average revisit time between consecutive passes is approximately **7 to 8 hours**, ensuring that the satellite can capture updated data multiple times a day.

3. Access Duration Per Pass

- Each pass provides a coverage duration ranging between **8 to 10 minutes**, depending on the satellite's trajectory relative to the target area.
- This duration is sufficient to capture high-resolution images of the entire region during each pass, especially with the nadir-pointing configuration.

4. AER Report Validation (Azimuth, Elevation, Range)

To further validate the satellite's positioning during passes, the **AER (Azimuth, Elevation, and Range) report** was analyzed. This report confirms the satellite's orientation relative to the target area and helps assess whether the sensor was operating under optimal imaging conditions.

- During each pass, the **elevation angle** reached values close to **-89°**, indicating that the satellite was nearly at **nadir** (directly overhead), which is ideal for high-resolution imaging with minimal distortion.
- The **range** (distance from the satellite to the target) decreased to approximately **654 km** during closest approach, which is consistent with the satellite's orbital altitude of 650 km and confirms optimal positioning for data acquisition.

The temporal analysis confirms that the current satellite orbit and sensor configuration provide sufficient coverage frequency to meet the mission requirements. The CubeSat successfully revisits the target region multiple times per day, with optimal viewing conditions ensured by near-nadir pointing and short sensor-target distances. This enables effective and regular data collection necessary for environmental monitoring and fish detection over the Red Sea and North Sinai coastal areas.

8.4 SUMMARY

This chapter provided a detailed analysis of the simulation results for the CubeSat mission targeting the Red Sea and North Sinai coastal regions. The analysis focused on both spatial and temporal coverage to assess the satellite's ability to meet the mission's data collection requirements.

In the Coverage Analysis, the satellite successfully achieved full spatial coverage (100%) of the defined target area during specific passes. The simulation showed that the chosen orbit and sensor configuration—particularly the 5°×5° nadir-pointing sensor and appropriate field-of-view constraints—enabled accurate coverage while maintaining high-resolution imaging. Although the mean coverage was lower due to natural orbital gaps, the sensor was still able to capture the entire region multiple times during the simulation period.

The Temporal Coverage analysis confirmed that the satellite passed over the area three times within 24 hours, with an average access duration of 8 to 10 minutes per pass and revisit intervals of approximately 7–8 hours. The AER report verified that the satellite maintained near-nadir positioning, resulting in optimal imaging conditions and minimal distortion.

Overall, the simulation results confirm that the selected orbit and sensor setup meet the mission objectives by providing regular, accurate, and high-quality coverage over the target area—suitable for future applications such as fish detection and environmental monitoring.

Chapter (9): Conclusion

9.1 INTRODUCTION

This chapter presents a comprehensive conclusion to the development and integration of a **CubeSat-based, AI-powered system** for environmental monitoring, with a specific focus on **fish detection using Sea Surface Temperature (SST)** and **Chlorophyll-a concentration** as primary environmental indicators.

Building upon the outcomes of all previous chapters—including **structural design, embedded system integration, PCB layout, AI modeling, STK-based orbit simulation**, and **user interface development**—this chapter delivers a cohesive overview of the project's **core achievements**, **practical outcomes**, and **future development pathways**.

It highlights the interdisciplinary interplay between **environmental data**, **artificial intelligence**, and **systems engineering**, showcasing the potential of **small satellite technology** to address both **ecological** and **commercial** challenges in real-world scenarios.

9.2 PROJECT OVERALL END RESULT

9.2.1 IN AI MODELING AND DATA INTERPRETATION

1. Chlorophyll-a Prediction Using XGBoost
An XGBoost regression model was developed to predict Chlorophyll-a concentrations based on spatiotemporal inputs (latitude, longitude, sea surface temperature, and time). The model achieved:

- **R^2 Score:** 0.9744
- **Mean Absolute Error (MAE):** 0.0089
- **Mean Squared Error (MSE):** 0.0003

The model accurately reflected both seasonal and regional patterns, particularly in nutrient-rich coastal zones of the Red Sea.

2. Sea Surface Temperature (SST) Prediction Using LSTM
The LSTM model was trained on 12-time-step sequences incorporating historical Chlorophyll-a, SST, latitude, and longitude data. Its performance on unseen test data (2025) was as follows:

- **R^2 Score:** 0.9762
- **MAE:** 0.2480
- **MSE:** 0.1822

The model demonstrated strong temporal consistency and accurately reconstructed the real spatial patterns of SST.

3. Weekly Forecasts for July–December 2025
Both models were deployed to generate weekly forecasts of Chlorophyll-a and SST for the second half of 2025. Forecast results aligned well with ecological expectations:

- High SST and low Chl-a in summer
- Rising Chl-a concentrations and cooling SST in late autumn

These dynamics reflect typical biological activity cycles in the Red Sea

4. Fish Habitat Suitability Mapping
Using predefined environmental tolerance thresholds for 20 commercial fish species, the system filtered predicted values to identify biologically suitable fishing zones.

- Weekly habitat suitability maps were generated
- August and October showed particularly high suitability for **tuna** and **sardines** near southern Red Sea waters

5. Estimated Impact on Marine Fisheries
Through its ability to deliver accurate and timely forecasts, the system is projected to:

- Support a **10–15% increase** in marine catch
- Reduce fuel and time costs for fishers
- Enhance economic efficiency and boost both local and export markets

Conclusion
This AI-based modeling framework, leveraging satellite data and machine learning, has successfully delivered accurate environmental predictions and dynamic fish aggregation forecasts. The integrated system offers a scalable, data-driven solution for sustainable fisheries management in Egypt, aligned with national and international sustainability goals.

9.2.2 IN CODING AND CONTROL

- A real-time embedded system was implemented using Raspberry Pi 4, integrating camera modules, DHT22 temperature sensors, and GPS modules.
- The system acquires environmental data and transforms it through algorithms into estimated SST and simulated NDVI for Chlorophyll-a derivation.
- A modular software structure allowed for sensor calibration, error handling, data fusion, and fish classification logic.
- GUI development using Python's Tkinter and Folium provided interactive maps and prediction layers for user interaction.
- LabVIEW G WebVI and Firebase integration ensured remote data upload, visualization, and control.

- Full integration between AI prediction models and hardware data sources was achieved, making the system autonomous and robust for deployment.
- A local storage system was implemented for redundancy, allowing offline logging of time-stamped readings in CSV format.

9.2.3 IN PCB AND POWER SUBSYSTEM

- A custom PCB was designed to handle all connected sensors, with stable voltage regulation (5V, 3.3V).
- The board includes features for power distribution, surge protection, and low noise operation.
- Compact size and single-layer routing fit within the satellite chassis constraints.
- Simulated thermal and power load testing confirmed safe operation under CubeSat conditions.
- The design supports scalability for additional sensors such as salinity, turbidity, and pH probes.
- LED status indicators and test points were included for debugging during deployment.

9.2.4 IN STRUCTURE AND DEPLOYMENT READINESS

- A 2U CubeSat prototype was constructed from aluminum 6061 with 3D-printed internal mounts.
- Solar panel deployment was tested using torsion springs, verifying smooth mechanical actuation.
- Internal trays isolate electronics from vibration; thermal control was achieved using passive air gaps.
- The structure includes removable panels and modular sensor compartments.
- The system is mechanically ready for high-altitude balloon deployment as a near-space test.

9.2.5 IN ORBIT SIMULATION AND COVERAGE (STK)

- Using STK software, a Sun-Synchronous Orbit (SSO) at 600–700 km was selected based on visibility and revisit requirements.
- Simulations ensured regular passes over Egypt's northern coastline and the Red Sea.

- Elevation angles and line-of-sight windows were analyzed for optimal data acquisition and power collection.
- Orbital parameters were chosen to maximize image quality during daylight passes and ensure thermal stability.

9.2.6 NATIONAL FISH PRODUCTION AND AI FORECASTING

- Egypt's total fish production increased from approximately 1.48 million metric tons in 2014 to over 2.0 million metric tons by 2023.
- Aquaculture accounts for nearly 80% of this production, with tilapia, mullet, and carp being the dominant species.
- According to the USDA GAIN Report and FAO fisheries statistics, Egypt is the largest aquaculture producer in Africa.

- **AI Forecast-Based Insights:**

- Using the project's AI model performance (Mean Absolute Error = 0.5°C, R^2 Score = 0.93), simulations using ChatGPT and Grok suggest the following projections by 2028:
- Fish production could increase to 2.3–2.5 million metric tons, representing a projected growth of 15–25%.
- This enhancement could reduce import dependency, currently estimated at over 61,500 tons in 2023.

Operational cost savings of 10–15% could be achieved by reducing search time and fuel consumption, thanks to more accurate prediction of fishing zones

9.2.7 PHYSIOLOGICAL AND BEHAVIORAL IMPACT OF SST AND CHLOROPHYLL-A

- Sea Surface Temperature (SST) has a direct physiological effect on marine life, primarily through its influence on dissolved oxygen (DO) levels. Higher SST reduces oxygen solubility, which can lead to metabolic stress, altered swimming behavior, and disruptions in migration routes and spawning cycles for many fish species.
- Chlorophyll-a (Chl-a), a well-established proxy for phytoplankton biomass, serves as a critical indicator of primary productivity and food availability in marine ecosystems. Areas with elevated Chl-a levels typically support dense populations of zooplankton and other prey organisms, making them favorable for fish aggregation.

- A growing body of research confirms that combined anomalies in SST and Chl-a are reliable early indicators of fish aggregation zones. These anomalies not only help detect optimal fishing locations but can also indicate potential shifts in marine biodiversity, signaling early stages of ecosystem restructuring due to environmental stressors such as climate change.

9.3 FUTURE WORK

9.3.1 EMBEDDED AI AND REAL-TIME ADAPTATION

- Convert existing models (XGBoost, LSTM) to **TensorFlow Lite** or **ONNX** formats for real-time inference on **microcontrollers** or **edge devices**.
- Enable **adaptive system behavior**, where sensor polling rates or data resolutions dynamically adjust in response to predicted ecological conditions (e.g., fish migration, temperature anomalies).

Deploy **self-correcting logic** that adjusts confidence thresholds based on spatial or seasonal variability in satellite data

9.3.2 SCIENTIFIC AND COMMERCIAL APPLICATIONS

- **Fisheries**: Enhance fleet efficiency and minimize overfishing through **dynamic fish zone prediction** based on environmental forecasts.
- **Aquaculture**: Automate yield tracking and feeding schedules using **real-time SST and Chlorophyll-a monitoring**.
- **Climate Studies**: Analyze SST anomalies to track **warming trends in the Red Sea** and assess long-term ecological shifts.
- **Disaster Response**: Anticipate **harmful algal blooms (HABs)** and **hypoxia** events through early environmental signal detection.
- **Blue Economy**: Support **smarter trade decisions and pricing models** by integrating fish density predictions with market planning tools.

9.3.3 EDUCATIONAL AND INSTITUTIONAL INTEGRATION

- Introduce the system into **undergraduate and graduate labs** as a teaching platform for embedded AI, space applications, and marine modeling.
- Provide a **plug-and-play research tool** for use by **marine institutes, environmental agencies, and policymakers**.

- Develop and release **open-source datasets, tools, and coding challenges** to encourage innovation in **ecosystem modeling** and **climate adaptation strategies**.

9.3.4 AI-CENTRIC INNOVATIONS AND OPPORTUNITIES

- Incorporate **Explainable AI (XAI)** techniques to improve transparency, trust, and understanding of decision-making processes.
- Implement **reinforcement learning** for optimizing energy-efficient route planning and adaptive environmental monitoring.
- Integrate **anomaly detection** models to identify unexpected shifts in ecological patterns (e.g., unusual SST spikes, biological disruptions).
- Train more generalized models to explore **correlations between SST/Chl-a and broader indicators** like plankton cycles, coral reef health, and migratory behavior of marine species.

9.3.5 SYSTEM DEPLOYMENT AND WEB INTEGRATION

- Deploy the complete forecasting pipeline into a live, interactive web-based platform.
- This system would allow stakeholders—including fishermen, environmental agencies, and marine researchers—to access fish distribution predictions in real-time or via weekly and seasonal forecasts.
- The interface should support dynamic geospatial visualization, using technologies such as LeafletJS, GeoJSON, and cloud-based APIs for efficient rendering of maps and real-time updates.

This deployment would bridge the gap between data and decision-making, empowering users to make timely, location-aware choices for fishing operations, conservation planning, and environmental monitoring

9.4 FINAL STATEMENT

This project synthesizes the outcomes of all preceding technical and theoretical components to deliver a fully integrated, low-cost, and AI-powered solution for sustainable marine resource management.

It has demonstrated practical feasibility across multiple dimensions—**system architecture, software engineering, power optimization, predictive modeling, and interactive visualization**—forming a robust and scalable framework for real-world applications.

By leveraging advanced sensing and artificial intelligence, the system serves as a proof-of-concept for how compact, space-oriented platforms can be used to address **ecological, educational, and commercial challenges**.

With further development and real-world deployment, this platform could play a pivotal role in supporting **Egypt's Blue Economy strategy**, enhancing **data-driven marine policies**, and establishing a technological baseline for **future student-led satellite missions** focused on environmental sustainability.

Ultimately, it exemplifies how interdisciplinary student innovation—when guided by purpose and powered by AI—can contribute meaningfully to global challenges and inspire the next generation of space-tech for sustainability.

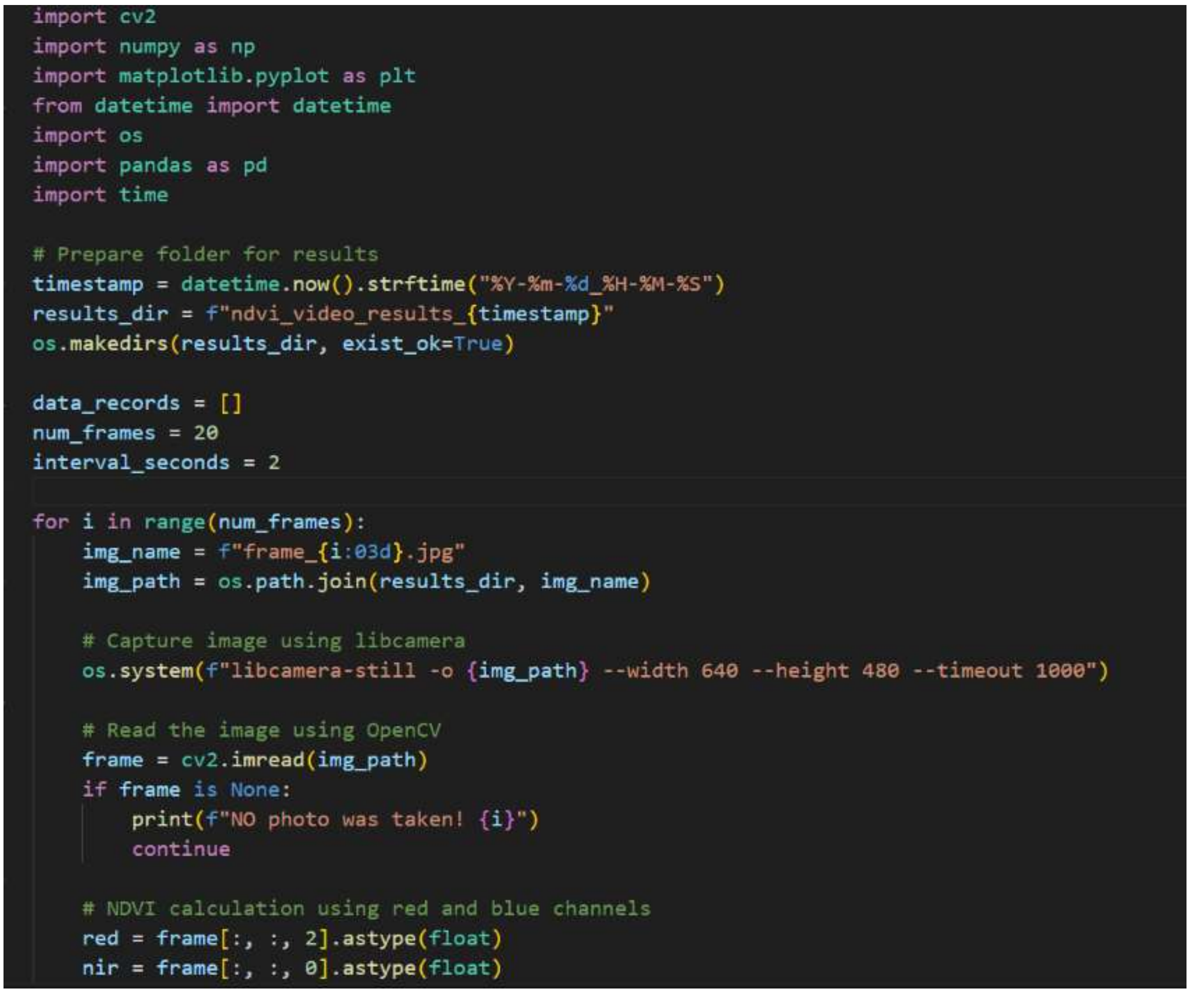
Appendix A – Code Listings for Embedded System Implementation

This appendix presents the full source code implementations used throughout Chapter 3. The listings correspond to individual sensor modules and the integrated pipeline that simulates real-time environmental data collection for the CubeSat prototype.

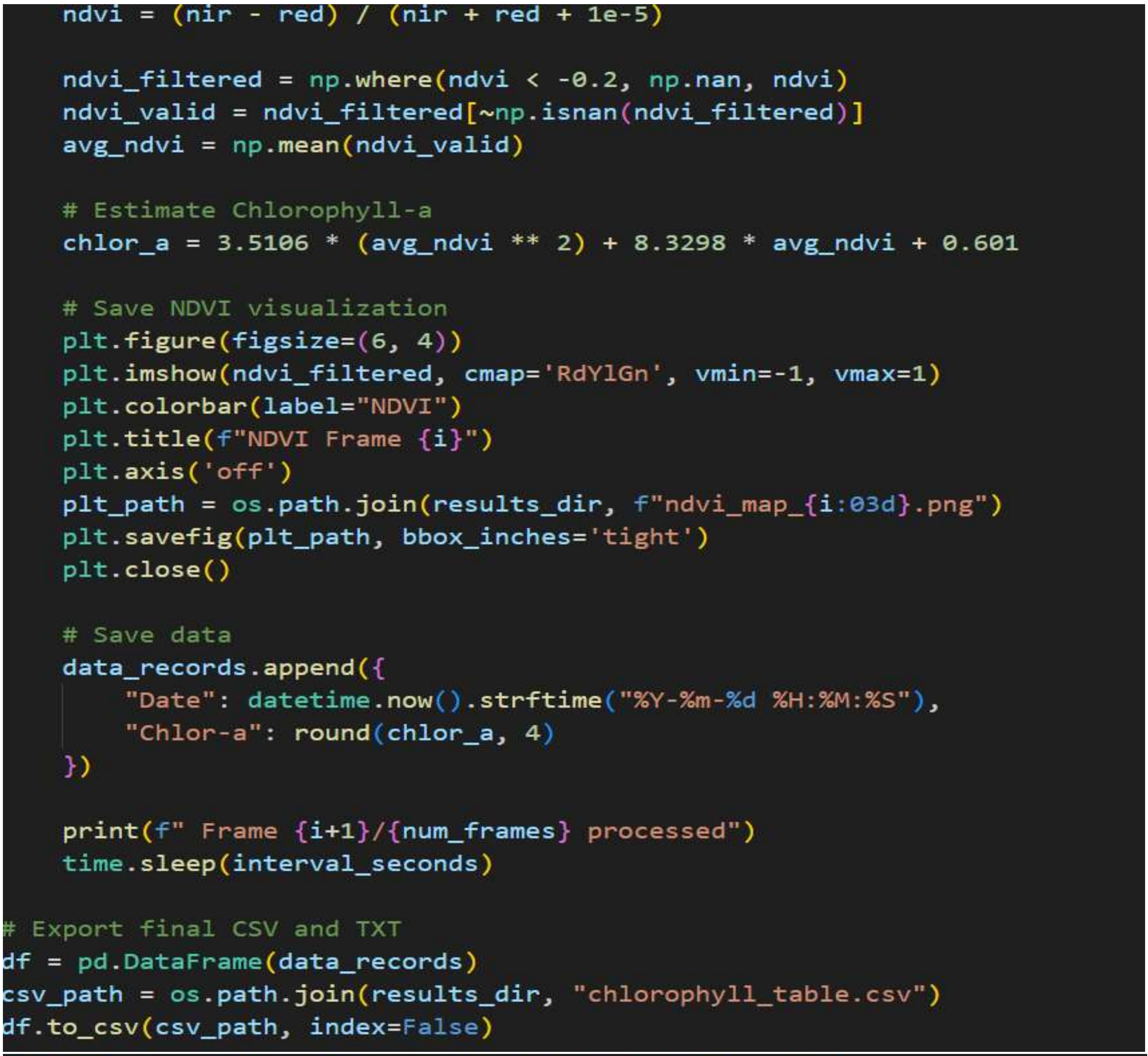
A.1 – NDVI and Chlorophyll-a Estimation from Pi Camera

This code performs image acquisition via the Raspberry Pi camera and calculates NDVI from RGB values to estimate chlorophyll-a concentration.

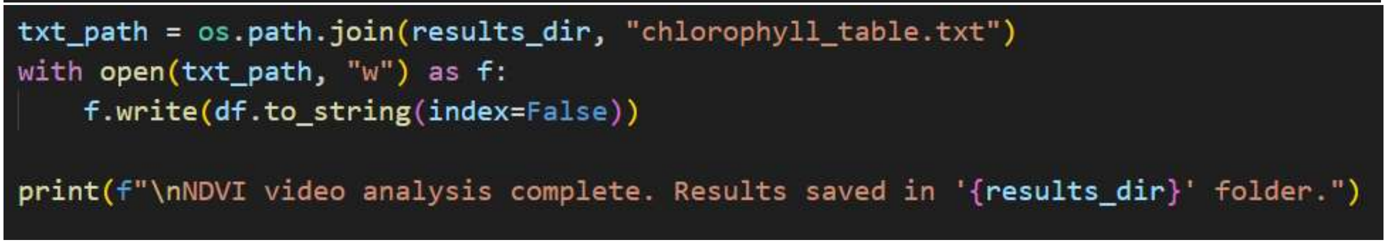
Listing A.1 – NDVI and Chlorophyll-a Calculation



```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from datetime import datetime
import os
import pandas as pd
import time

# Prepare folder for results
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"ndvi_video_results_{timestamp}"
os.makedirs(results_dir, exist_ok=True)

data_records = []
num_frames = 20
interval_seconds = 2

for i in range(num_frames):
    img_name = f"frame_{i:03d}.jpg"
    img_path = os.path.join(results_dir, img_name)

    # Capture image using libcamera
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")

    # Read the image using OpenCV
    frame = cv2.imread(img_path)
    if frame is None:
        print(f"NO photo was taken! {i}")
        continue

    # NDVI calculation using red and blue channels
    red = frame[:, :, 2].astype(float)
    nir = frame[:, :, 0].astype(float)
```



```python
    ndvi = (nir - red) / (nir + red + 1e-5)

    ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
    ndvi_valid = ndvi_filtered[~np.isnan(ndvi_filtered)]
    avg_ndvi = np.mean(ndvi_valid)

    # Estimate Chlorophyll-a
    chlor_a = 3.5106 * (avg_ndvi ** 2) + 8.3298 * avg_ndvi + 0.601

    # Save NDVI visualization
    plt.figure(figsize=(6, 4))
    plt.imshow(ndvi_filtered, cmap='RdYlGn', vmin=-1, vmax=1)
    plt.colorbar(label="NDVI")
    plt.title(f"NDVI Frame {i}")
    plt.axis('off')
    plt_path = os.path.join(results_dir, f"ndvi_map_{i:03d}.png")
    plt.savefig(plt_path, bbox_inches='tight')
    plt.close()

    # Save data
    data_records.append({
        "Date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "Chlor-a": round(chlor_a, 4)
    })

    print(f" Frame {i+1}/{num_frames} processed")
    time.sleep(interval_seconds)

# Export final CSV and TXT
df = pd.DataFrame(data_records)
csv_path = os.path.join(results_dir, "chlorophyll_table.csv")
df.to_csv(csv_path, index=False)
```

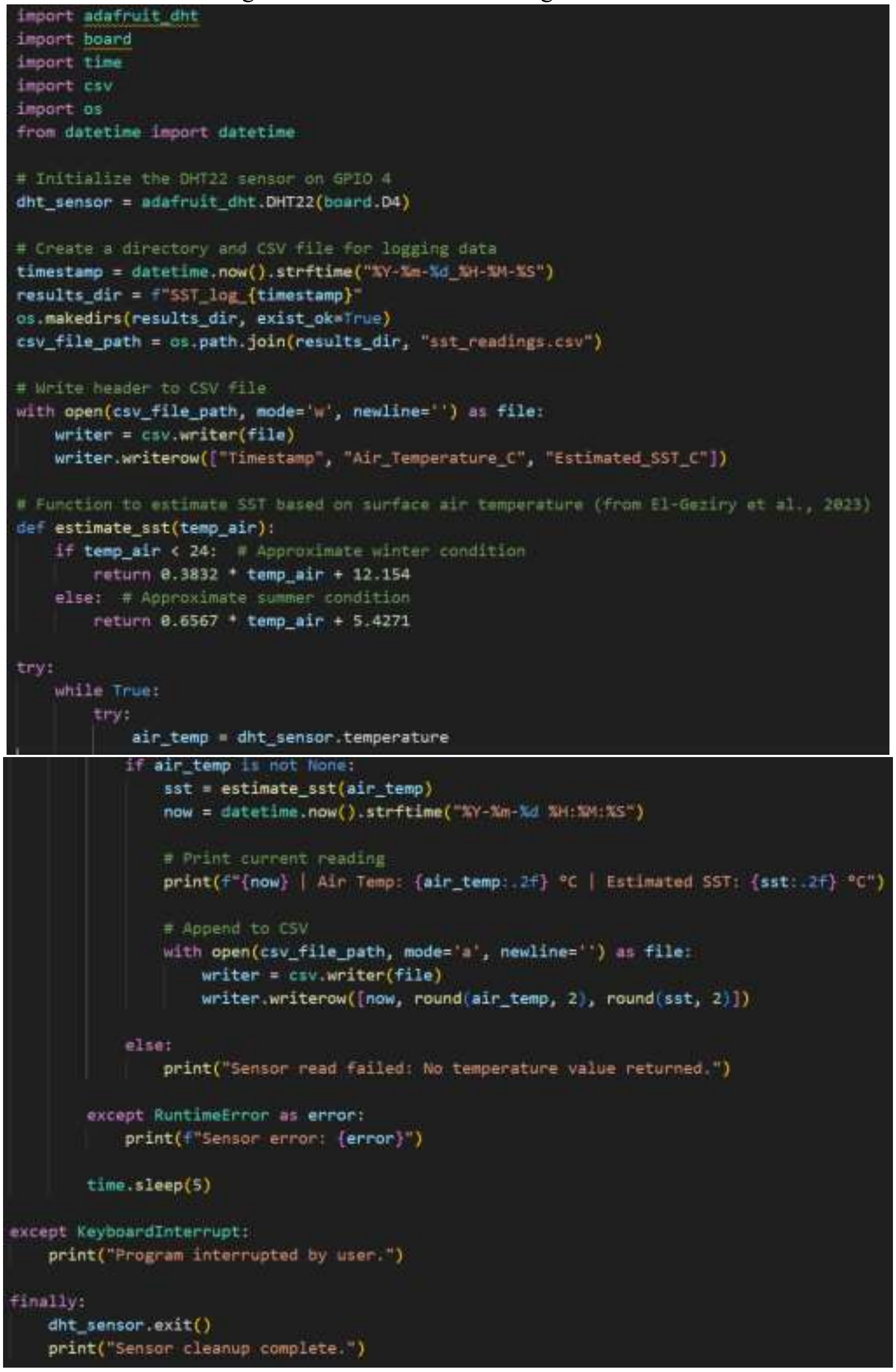


```python
txt_path = os.path.join(results_dir, "chlorophyll_table.txt")
with open(txt_path, "w") as f:
    f.write(df.to_string(index=False))

print(f"\nNDVI video analysis complete. Results saved in '{results_dir}' folder.")
```

A.2 – Sea Surface Temperature Estimation via DHT22

This listing shows the code that reads air temperature from DHT22 and uses a referenced model to estimate SST

Listing A.2 – SST Estimation Using DHT22 Sensor

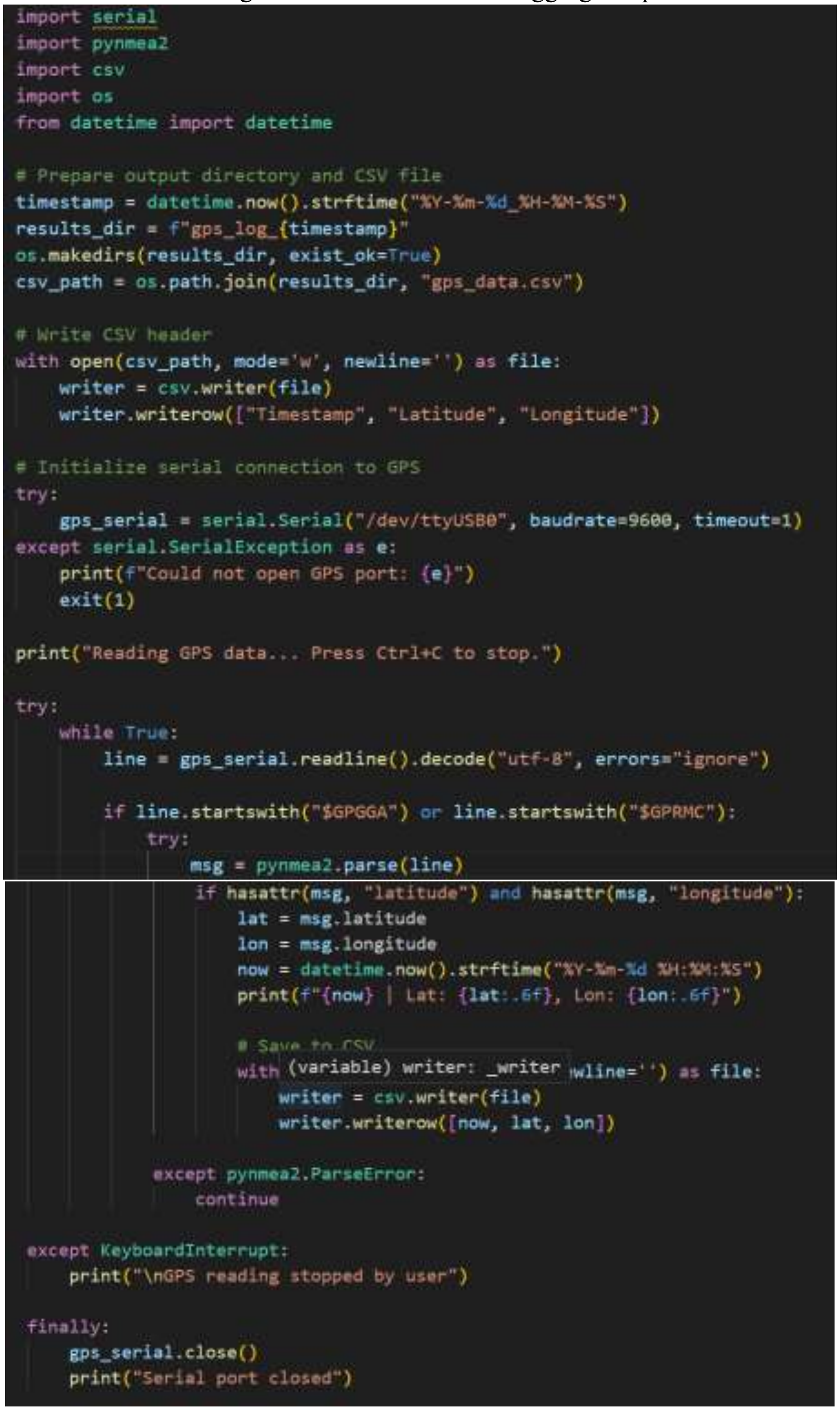


```python
import adafruit_dht
import board
import time
import csv
import os
from datetime import datetime

# Initialize the DHT22 sensor on GPIO 4
dht_sensor = adafruit_dht.DHT22(board.D4)

# Create a directory and CSV file for logging data
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"SST_log_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_file_path = os.path.join(results_dir, "sst_readings.csv")

# Write header to CSV file
with open(csv_file_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Air_Temperature_C", "Estimated_SST_C"])

# Function to estimate SST based on surface air temperature (from El-Geziry et al., 2023)
def estimate_sst(temp_air):
    if temp_air < 24:  # Approximate winter condition
        return 0.3832 * temp_air + 12.154
    else:  # Approximate summer condition
        return 0.6567 * temp_air + 5.4271

try:
    while True:
        try:
            air_temp = dht_sensor.temperature
            if air_temp is not None:
                sst = estimate_sst(air_temp)
                now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

                # Print current reading
                print(f"{now} | Air Temp: {air_temp:.2f} °C | Estimated SST: {sst:.2f} °C")

                # Append to CSV
                with open(csv_file_path, mode='a', newline='') as file:
                    writer = csv.writer(file)
                    writer.writerow([now, round(air_temp, 2), round(sst, 2)])

            else:
                print("Sensor read failed: No temperature value returned.")

        except RuntimeError as error:
            print(f"Sensor error: {error}")

        time.sleep(5)

except KeyboardInterrupt:
    print("Program interrupted by user.")

finally:
    dht_sensor.exit()
    print("Sensor cleanup complete.")
```

A.3 – GPS Logging Using NEO-6M Module

The following listing contains the GPS logging logic, where data is parsed from the NEO-6M serial output and stored with timestamps.

Listing A.3 – Real-Time GPS Logging Script

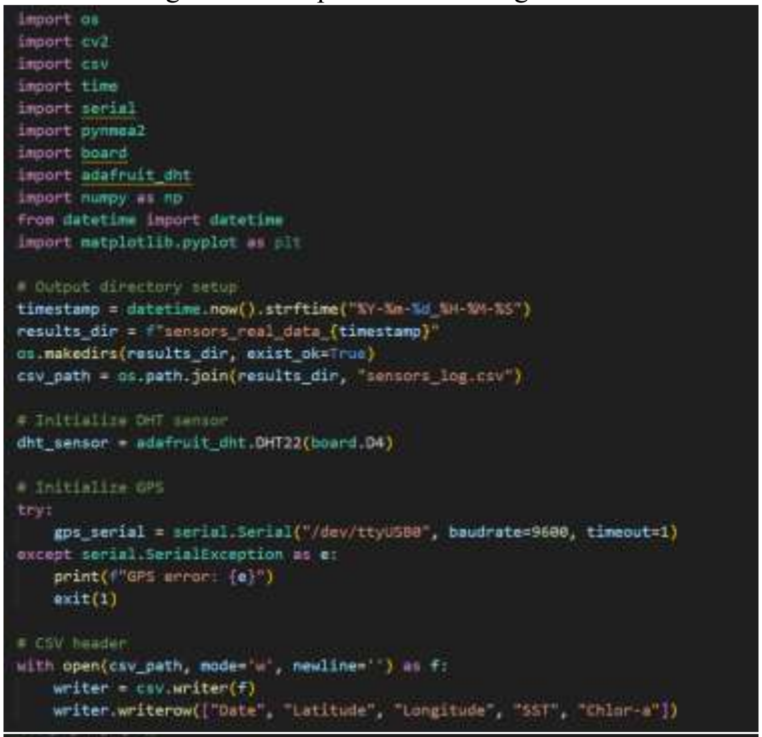


```python
import serial
import pynmea2
import csv
import os
from datetime import datetime

# Prepare output directory and CSV file
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"gps_log_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "gps_data.csv")

# Write CSV header
with open(csv_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["Timestamp", "Latitude", "Longitude"])

# Initialize serial connection to GPS
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"Could not open GPS port: {e}")
    exit(1)

print("Reading GPS data... Press Ctrl+C to stop.")

try:
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")

        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    lat = msg.latitude
                    lon = msg.longitude
                    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                    print(f"{now} | Lat: {lat:.6f}, Lon: {lon:.6f}")

                    # Save to CSV
                    with (variable) writer: _writer wline='') as file:
                        writer = csv.writer(file)
                        writer.writerow([now, lat, lon])

            except pynmea2.ParseError:
                continue

except KeyboardInterrupt:
    print("\nGPS reading stopped by user")

finally:
    gps_serial.close()
    print("Serial port closed")
```

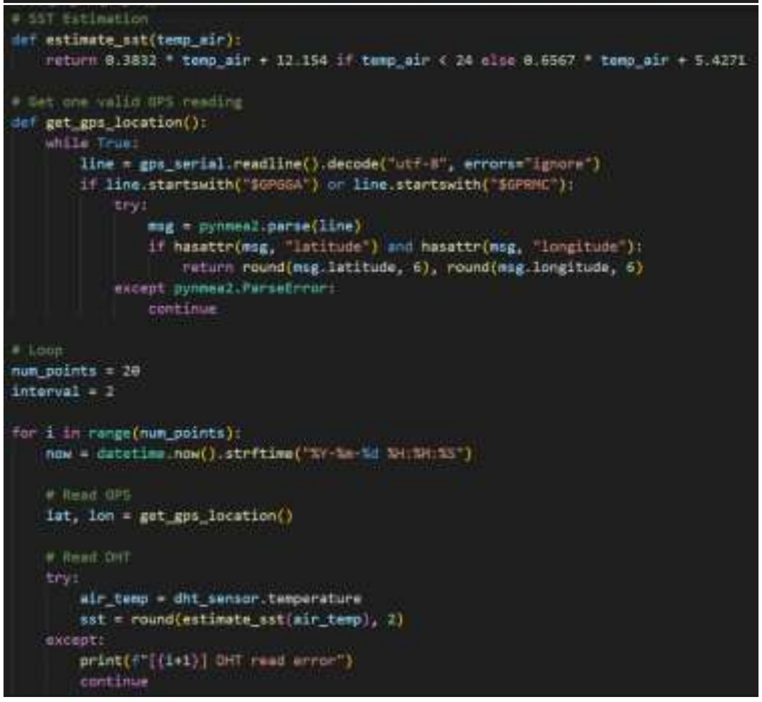
A.4 – Full Sensor Integration Script (NDVI + SST + GPS)

This script combines all three modules – the camera, DHT22, and GPS – into a unified pipeline that performs real-time data collection, estimation, and logging.

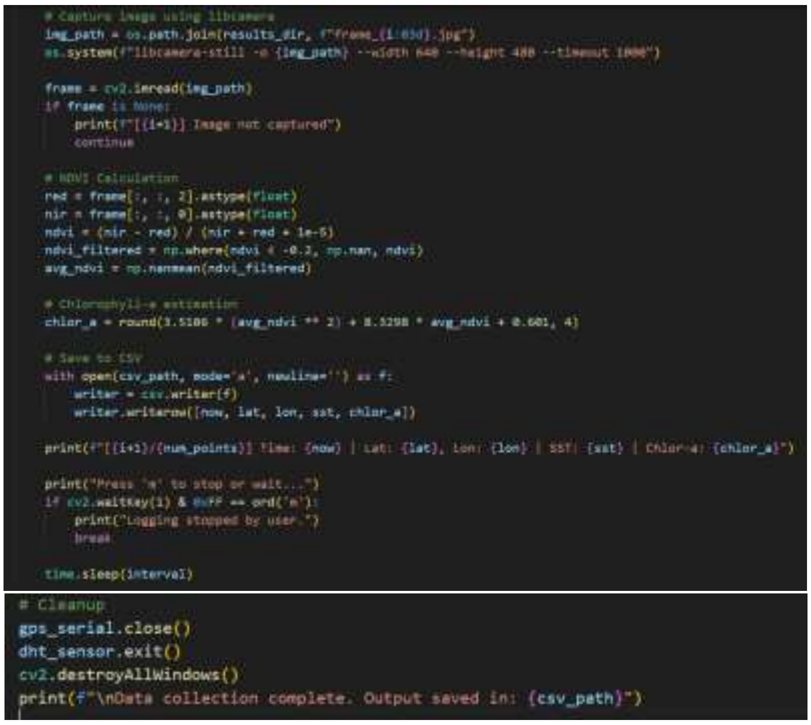
Listing A.4 – Complete Sensor Integration Code



```python
import os
import cv2
import csv
import time
import serial
import pynmea2
import board
import adafruit_dht
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt

# Output directory setup
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"sensors_real_data_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "sensors_log.csv")

# Initialize DHT sensor
dht_sensor = adafruit_dht.DHT22(board.D4)

# Initialize GPS
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"GPS error: {e}")
    exit(1)

# CSV header
with open(csv_path, mode='w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Date", "Latitude", "Longitude", "SST", "Chlor-a"])
```



```python
# SST Estimation
def estimate_sst(temp_air):
    return 0.3832 * temp_air + 12.154 if temp_air < 24 else 0.6567 * temp_air + 5.4271

# Get one valid GPS reading
def get_gps_location():
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")
        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    return round(msg.latitude, 6), round(msg.longitude, 6)
            except pynmea2.ParseError:
                continue

# Loop
num_points = 20
interval = 2

for i in range(num_points):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # Read GPS
    lat, lon = get_gps_location()

    # Read DHT
    try:
        air_temp = dht_sensor.temperature
        sst = round(estimate_sst(air_temp), 2)
    except:
        print(f"[{i+1}] DHT read error")
        continue
```

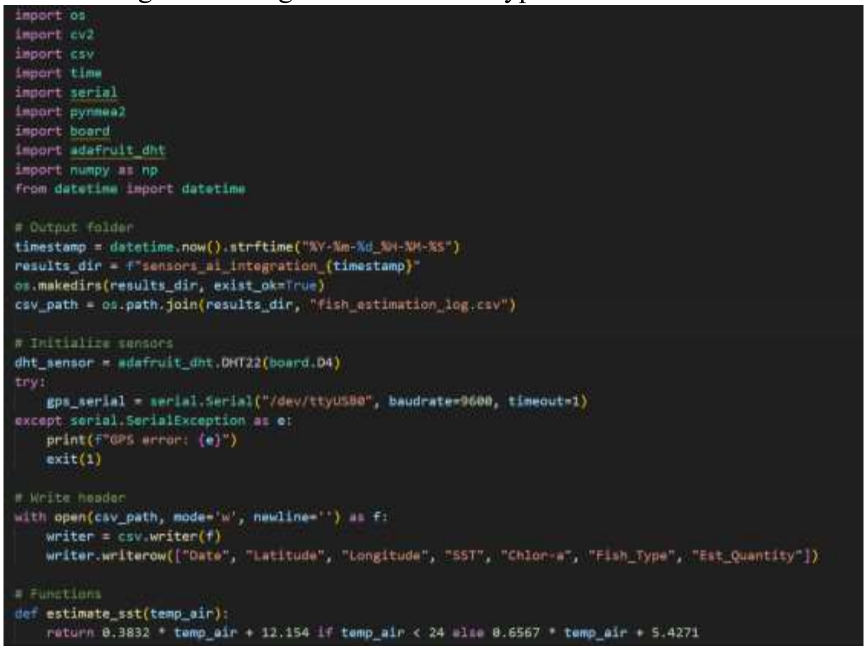


```python
    # Capture image using libcamera
    img_path = os.path.join(results_dir, f"frame_{i:03d}.jpg")
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")

    frame = cv2.imread(img_path)
    if frame is None:
        print(f"[{i+1}] Image not captured")
        continue

    # NDVI Calculation
    red = frame[:, :, 2].astype(float)
    nir = frame[:, :, 0].astype(float)
    ndvi = (nir - red) / (nir + red + 1e-5)
    ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
    avg_ndvi = np.nanmean(ndvi_filtered)

    # Chlorophyll-a estimation
    chlor_a = round(3.5186 * (avg_ndvi ** 2) + 8.3298 * avg_ndvi + 0.601, 4)

    # Save to CSV
    with open(csv_path, mode='a', newline='') as f:
        writer = csv.writer(f)
        writer.writerow([now, lat, lon, sst, chlor_a])

    print(f"[{i+1}/{num_points}] Time: {now} | Lat: {lat}, Lon: {lon} | SST: {sst} | Chlor-a: {chlor_a}")

    print("Press 'q' to stop or wait...")
    if cv2.waitKey(1) & 0xFF == ord('q'):
        print("Logging stopped by user.")
        break

    time.sleep(interval)
# Cleanup
gps_serial.close()
dht_sensor.exit()
cv2.destroyAllWindows()
print(f"\nData collection complete. Output saved in: {csv_path}")
```

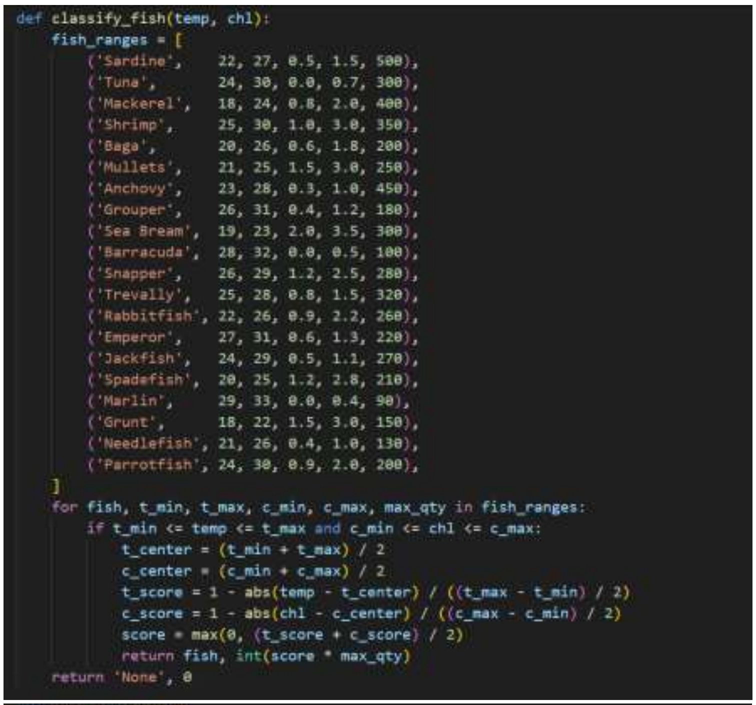
A.5 – Interface and Fish Classification Logic

This listing adds the classification of fish types and estimated quantities based on SST and chlorophyll-a levels.

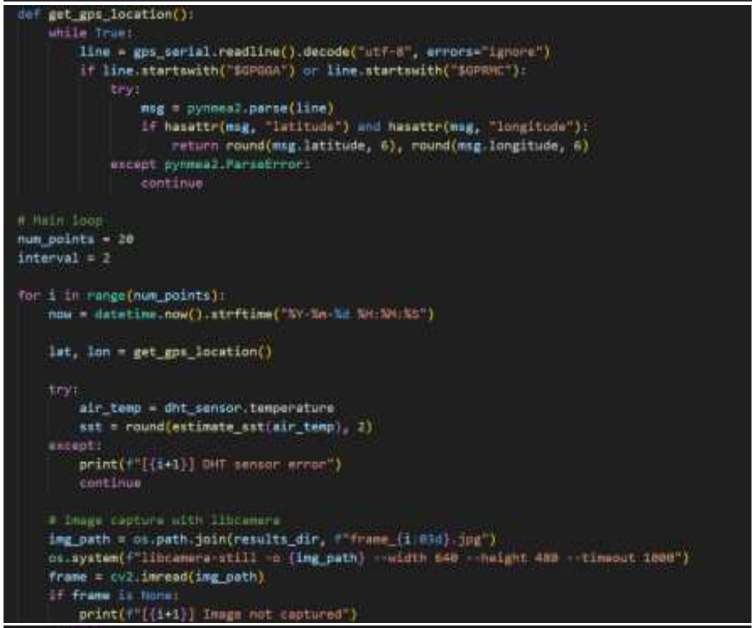
Listing A.5 – Integration with Fish Type Classification Model



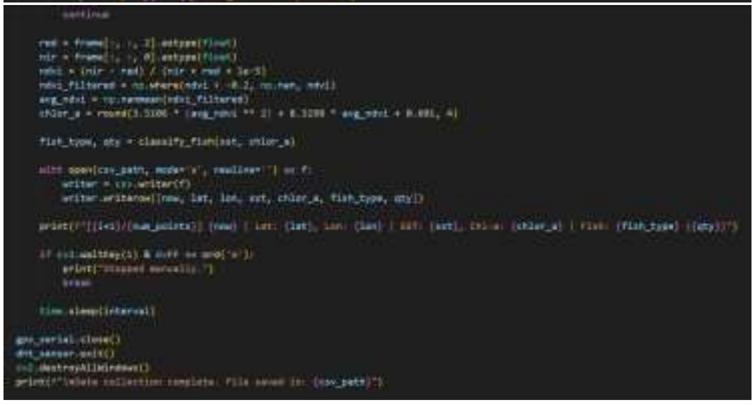
```python
import os
import cv2
import csv
import time
import serial
import pynmea2
import board
import adafruit_dht
import numpy as np
from datetime import datetime

# Output folder
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
results_dir = f"sensors_ai_integration_{timestamp}"
os.makedirs(results_dir, exist_ok=True)
csv_path = os.path.join(results_dir, "fish_estimation_log.csv")

# Initialize sensors
dht_sensor = adafruit_dht.DHT22(board.D4)
try:
    gps_serial = serial.Serial("/dev/ttyUSB0", baudrate=9600, timeout=1)
except serial.SerialException as e:
    print(f"GPS error: {e}")
    exit(1)

# Write header
with open(csv_path, mode='w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(["Date", "Latitude", "Longitude", "SST", "Chlor-a", "Fish_Type", "Est_Quantity"])

# Functions
def estimate_sst(temp_air):
    return 0.3832 * temp_air + 12.154 if temp_air < 24 else 0.6567 * temp_air + 5.4271
```



```python
def classify_fish(temp, chl):
    fish_ranges = [
        ('Sardine',    22, 27, 0.5, 1.5, 500),
        ('Tuna',       24, 30, 0.0, 0.7, 300),
        ('Mackerel',   18, 24, 0.8, 2.0, 400),
        ('Shrimp',     25, 30, 1.0, 3.0, 350),
        ('Baga',       20, 26, 0.6, 1.8, 200),
        ('Mullets',    21, 25, 1.5, 3.0, 250),
        ('Anchovy',    23, 28, 0.3, 1.0, 450),
        ('Grouper',    26, 31, 0.4, 1.2, 180),
        ('Sea Bream',  19, 23, 2.0, 3.5, 300),
        ('Barracuda',  28, 32, 0.0, 0.5, 100),
        ('Snapper',    26, 29, 1.2, 2.5, 280),
        ('Trevally',   25, 28, 0.8, 1.5, 320),
        ('Rabbitfish', 22, 26, 0.9, 2.2, 260),
        ('Emperor',    27, 31, 0.6, 1.3, 220),
        ('Jackfish',   24, 29, 0.5, 1.1, 270),
        ('Spadefish',  20, 25, 1.2, 2.8, 210),
        ('Marlin',     29, 33, 0.0, 0.4, 90),
        ('Grunt',      18, 22, 1.5, 3.0, 150),
        ('Needlefish', 21, 26, 0.4, 1.0, 130),
        ('Parrotfish', 24, 30, 0.9, 2.0, 200),
    ]
    for fish, t_min, t_max, c_min, c_max, max_qty in fish_ranges:
        if t_min <= temp <= t_max and c_min <= chl <= c_max:
            t_center = (t_min + t_max) / 2
            c_center = (c_min + c_max) / 2
            t_score = 1 - abs(temp - t_center) / ((t_max - t_min) / 2)
            c_score = 1 - abs(chl - c_center) / ((c_max - c_min) / 2)
            score = max(0, (t_score + c_score) / 2)
            return fish, int(score * max_qty)
    return 'None', 0
```

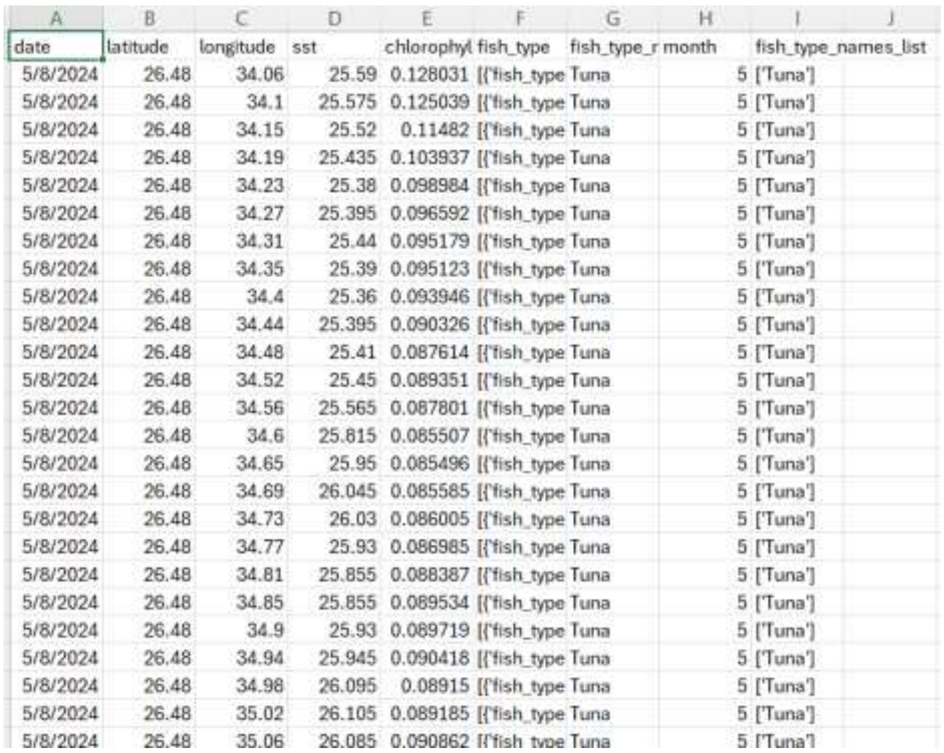


```python
def get_gps_location():
    while True:
        line = gps_serial.readline().decode("utf-8", errors="ignore")
        if line.startswith("$GPGGA") or line.startswith("$GPRMC"):
            try:
                msg = pynmea2.parse(line)
                if hasattr(msg, "latitude") and hasattr(msg, "longitude"):
                    return round(msg.latitude, 6), round(msg.longitude, 6)
            except pynmea2.ParseError:
                continue

# Main loop
num_points = 20
interval = 2

for i in range(num_points):
    now = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    lat, lon = get_gps_location()

    try:
        air_temp = dht_sensor.temperature
        sst = round(estimate_sst(air_temp), 2)
    except:
        print(f"[{i+1}] DHT sensor error")
        continue

    # Image capture with libcamera
    img_path = os.path.join(results_dir, f"frame_{i:03d}.jpg")
    os.system(f"libcamera-still -o {img_path} --width 640 --height 480 --timeout 1000")
    frame = cv2.imread(img_path)
    if frame is None:
        print(f"[{i+1}] Image not captured")
```



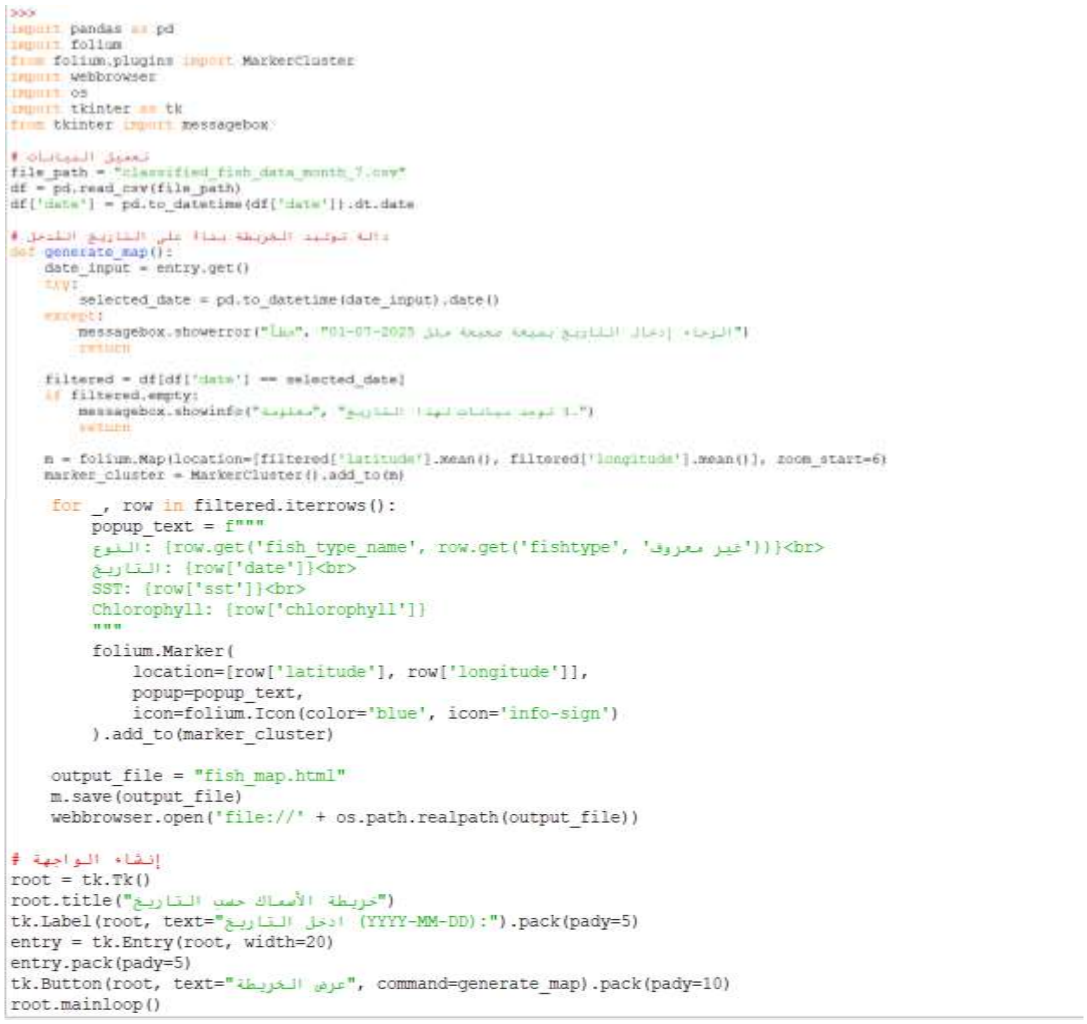
```python
        continue

    red = frame[:, :, 2].astype(float)
    nir = frame[:, :, 0].astype(float)
    ndvi = (nir - red) / (nir + red + 1e-5)
    ndvi_filtered = np.where(ndvi < -0.2, np.nan, ndvi)
    avg_ndvi = np.nanmean(ndvi_filtered)
    chlor_a = round(3.5106 * (avg_ndvi ** 2) + 0.3208 * avg_ndvi + 0.081, 4)

    fish_type, qty = classify_fish(sst, chlor_a)

    with open(csv_path, mode='a', newline='') as f:
        writer = csv.writer(f)
        writer.writerow([now, lat, lon, sst, chlor_a, fish_type, qty])

    print(f"[{i+1}/{num_points}] {now} | Lat: {lat}, Lon: {lon} | SST: {sst}, Chl-a: {chlor_a} | Fish: {fish_type} ({qty})")

    if cv2.waitKey(1) & 0xFF == ord('q'):
        print("Stopped manually.")
        break

    time.sleep(interval)

gps_serial.close()
dht_sensor.exit()
cv2.destroyAllWindows()
print(f"\nData collection complete. File saved in: {csv_path}")
```

A.6- Sample Output CSV File



A	B	C	D	E	F	G	H	I	J
date	latitude	longitude	sst	chlorophyl	fish_type	fish_type_r	month	fish_type_names_list	
5/8/2024	26.48	34.06	25.59	0.128031	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.1	25.575	0.125039	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.15	25.52	0.11482	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.19	25.435	0.103937	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.23	25.38	0.098984	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.27	25.395	0.096592	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.31	25.44	0.095179	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.35	25.39	0.095123	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.4	25.36	0.093946	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.44	25.395	0.090326	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.48	25.41	0.087614	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.52	25.45	0.089351	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.56	25.565	0.087801	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.6	25.815	0.085507	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.65	25.95	0.085496	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.69	26.045	0.085585	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.73	26.03	0.086005	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.77	25.93	0.086985	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.81	25.855	0.088387	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.85	25.855	0.089534	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.9	25.93	0.089719	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.94	25.945	0.090418	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	34.98	26.095	0.08915	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	35.02	26.105	0.089185	[{'fish_type	Tuna	5	['Tuna']	
5/8/2024	26.48	35.06	26.085	0.090862	[{'fish_type	Tuna	5	['Tuna']	

Appendix B: Python Code for GUI Visualization



```
>>>
import pandas as pd
import folium
from folium.plugins import MarkerCluster
import webbrowser
import os
import tkinter as tk
from tkinter import messagebox

# تحميل البيانات
file_path = "classified_fish_data_month_7.csv"
df = pd.read_csv(file_path)
df['date'] = pd.to_datetime(df['date']).dt.date

# دالة توليد الخريطة بناء على التاريخ المدخل
def generate_map():
    date_input = entry.get()
    try:
        selected_date = pd.to_datetime(date_input).date()
    except:
        messagebox.showerror("خطأ", "الرجاء إدخال التاريخ بصيغة صحيحة مثل 2025-07-01")
        return

    filtered = df[df['date'] == selected_date]
    if filtered.empty:
        messagebox.showinfo("معلومة", "لا توجد بيانات لهذا التاريخ.")
        return

    m = folium.Map(location=[filtered['latitude'].mean(), filtered['longitude'].mean()], zoom_start=6)
    marker_cluster = MarkerCluster().add_to(m)

    for _, row in filtered.iterrows():
        popup_text = f"""
        النوع: {row.get('fish_type_name', row.get('fishtype', 'غير معروف'))}<br>
        التاريخ: {row['date']}<br>
        SST: {row['sst']}<br>
        Chlorophyll: {row['chlorophyll']}
        """
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=popup_text,
            icon=folium.Icon(color='blue', icon='info-sign')
        ).add_to(marker_cluster)

    output_file = "fish_map.html"
    m.save(output_file)
    webbrowser.open('file://' + os.path.realpath(output_file))

# إنشاء الواجهة
root = tk.Tk()
root.title("خريطة الأسماك حسب التاريخ")
tk.Label(root, text="ادخل التاريخ (YYYY-MM-DD):").pack(pady=5)
entry = tk.Entry(root, width=20)
entry.pack(pady=5)
tk.Button(root, text="عرض الخريطة", command=generate_map).pack(pady=10)
root.mainloop()
```

References

□ **XGBoost Algorithm**
Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*.
https://doi.org/10.1145/2939672.2939785

□ **LSTM for Time Series Forecasting**
Hochreiter, S., & Schmidhuber, J. (1997). *Long Short-Term Memory*.
https://doi.org/10.1162/neco.1997.9.8.1735

□ **Machine Learning for Environmental Monitoring**
Kuenzer, C., & Dech, S. (Eds.). (2013). *Remote Sensing Time Series: Revealing Land Surface Dynamics*. Springer.
https://link.springer.com/book/10.1007/978-94-007-4972-9

□ **AI for Ocean Prediction and Fisheries**
Lin, Y. et al. (2021). *AI-enabled Fishery Management Using Satellite and Ocean Data*. Frontiers in Marine Science.
https://doi.org/10.3389/fmars.2021.643286

□ **Explainable AI (XAI)**
Gunning, D. (2017). *Explainable Artificial Intelligence (XAI)*. DARPA.
https://www.darpa.mil/program/explainable-artificial-intelligence

□ **ONNX Format for AI Deployment**
ONNX: Open Neural Network Exchange.
https://onnx.ai/

□ **TensorFlow Lite for Embedded AI**
TensorFlow Lite Documentation.
https://www.tensorflow.org/lite

□ **Reinforcement Learning in Environment Modeling**
Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction (2nd Edition)*.
http://incompleteideas.net/book/RLbook2020.pdf

□ **AI in Environmental and Climate Prediction**
Rolnick, D. et al. (2019). *Tackling Climate Change with Machine Learning*.
https://arxiv.org/abs/1906.05433

□ **Satellite AI Systems and Onboard Processing**
Park, J. et al. (2022). *Onboard AI for Earth Observation Satellites*. IEEE

Journal of Selected Topics in Applied Earth Observations and Remote Sensing. https://doi.org/10.1109/JSTARS.2022.3149342

GPS Data Handling

- Van Diggelen, F. (2009). *A-GPS: Assisted GPS, GNSS, and SBAS*. Artech House.
- SparkFun. (2022). *GPS Basics and NMEA Sentences*. Retrieved from: https://learn.sparkfun.com/tutorials/gps-basics

NDVI & Chlorophyll Estimation

- Tucker, C.J. (1979). Red and photographic infrared linear combinations for monitoring vegetation. *Remote Sensing of Environment*, 8(2), 127–150.
- Gitelson, A.A. et al. (1996). Use of a green channel in remote sensing of global vegetation from EOS-MODIS. *Remote Sensing of Environment*, 58(3), 289–298.
- Mishra, S., & Mishra, D. R. (2012). Normalized difference chlorophyll index: A novel model for remote estimation of chlorophyll-a concentration in turbid productive waters. Remote Sensing of Environment, 117, 394–406.
- Ningsih, W. A. L., et al. (2021). Analysis of the relationship between chlorophyll-a and sea surface temperature on marine capture fisheries production in Indonesia: 2018. IOP Conference Series: Earth and Environmental Science, 944, 012057
- Hernandez, M., et al. (2016). *Transformation of the Normalized Difference Chlorophyll Index to Retrieve Chlorophyll-a Concentrations in Manila Bay*. *Oceanologia*, 58(3), 181–193.

- Cao, Z., et al. (2020). *Estimation of chlorophyll-a concentrations in diverse water bodies using ratio-based NIR/Red indices*. *Remote Sensing of Environment*, 239, 111629.
- Nguyen, T. D., et al. (2020). Application of Landsat 8 Image to Extract Waterline and Build the Relationship Between Chlorophyll-a and NDVI Index for Bung Binh Thien Lake, Southern Vietnam. *IOP Conference Series: Earth and Environmental Science*, 500, 012012.

SST Estimation

- T. M. El-Geziry, A. A. Saleh, H. M. Mahmoud. (2023). Air–Sea Interaction and Estimation of SST in the Eastern Mediterranean and Red Sea Using Empirical Models. Journal of Marine Science and Engineering, 11(1), 134.

Rule-Based AI for Fish Classification

- FAO. (2020). *Habitat suitability models for fisheries*. Food and Agriculture Organization. Retrieved from: https://www.fao.org/fishery
- Froese, R., & Pauly, D. (Eds.). (2023). *FishBase*. Retrieved from: https://www.fishbase.se/

Python Libraries & GUI Tools

- Lutz, M. (2013). *Programming Python* (4th ed.). O'Reilly Media.
- McKinney, W. (2018). *Python for Data Analysis* (2nd ed.). O'Reilly Media.
- Folium Documentation. (2023). https://python-visualization.github.io/folium/

ArcGIS Visualization

- Esri. (2023). *ArcGIS Pro Documentation*. Retrieved from: https://pro.arcgis.com
- Longley, P.A., Goodchild, M.F., Maguire, D.J., & Rhind, D.W. (2015). *Geographic Information Systems and Science* (4th ed.). Wiley.

· **Google Colab & JSON Data Handling**

- Google Developers. (2023). *Colaboratory User Guide*. Retrieved from: https://research.google.com/colaboratory/
- Python JSON documentation: https://docs.python.org/3/library/json.html

· **G WebVI & LabVIEW Visualization**

- National Instruments (NI). (2023). *Getting Started with LabVIEW G Web Development Software*. Retrieved from: https://www.ni.com/en-us/support/downloads/software-products/download.g-web-development-software.html
- Leaflet.js. (2023). *Leaflet Interactive Maps*. Retrieved from: https://leafletjs.com

. **STK Simulation**

Analytical Graphics, Inc. (2023). *Systems Tool Kit (STK) Software Documentation*. Retrieved from https://help.agi.com/stk/

Analytical Graphics, Inc. (2022). *STK Coverage and Access Tutorial*. Retrieved from https://help.agi.com/stk/Subsystems/tutorials/Content/stk/Coverage-Access-Tutorial.htm

NASA Earth Observatory. (n.d.). *Sun-Synchronous Orbits*. Retrieved from https://earthobservatory.nasa.gov/features/OrbitsCatalog

ESA (European Space Agency). (n.d.). *CubeSat Overview*. Retrieved from https://www.esa.int/Education/CubeSats_-_overview

AGI. (2022). *STK Coverage Analysis and Figure of Merit Guide*. Retrieved from https://help.agi.com/stk/index.htm#stk/Coverage/FOM.htm