Resilient Forestry

INT-25-1

Jeremy Kawahigashi

Aditya Jain

Benjamin Spitzauer

Version: 0.1.1

1st April 2025

Table of Contents

1. Roster

Aditya (Lead Communicator): ajain1@seattleu.edu

Jeremy (CS Technical Lead): jkawahigashi@seattleu.edu

Benjamin (Lead Project Manager): bspitzauer@seattleu.edu

Emily (Environmental Science Technical Lead): egranken@seattleu.edu

Miranda (Environmental Science Communicator): mgormley@seattleu.edu

Emma (Environmental Science Project Manager): eweaver1@seattleu.edu

Dr. Bae (Advisor): baew@seattleu.edu

Dr. Lauer (Advisor): lauerj@seattleu.edu

Dr. Thompson (Advisor): thompson@seattleu.edu

2. Architecture

Our project will utilize a pipe-and-filter architecture, which is well suited for the data-intensive task of transforming UAV-captured aerial imagery into actionable geospatial insights for Resilient Forestry. This architecture will break down the workflow into a series of independent, modular stages (filters), each dedicated to a specific task, while connecting them via data flows (pipes) to ensure seamless processing. Key filters in our workflow include:

- **Image preprocessing (QA/QC):** Ensuring data quality and consistency.
- **3D Reconstruction:** Applying various algorithms to generate accurate spatial models.
- **Post-processing and data analysis:** Extracting actionable insights from processed imagery.

This architecture provides transparency and control at each stage. This addresses the limitations of the current "black-box" approach, which outputs only the optimal parameter suggestion, while still allowing ecologists and scientists to fine-tune parameters at specific stages. Our architecture ensures flexibility and adaptability to meet the project's requirements.

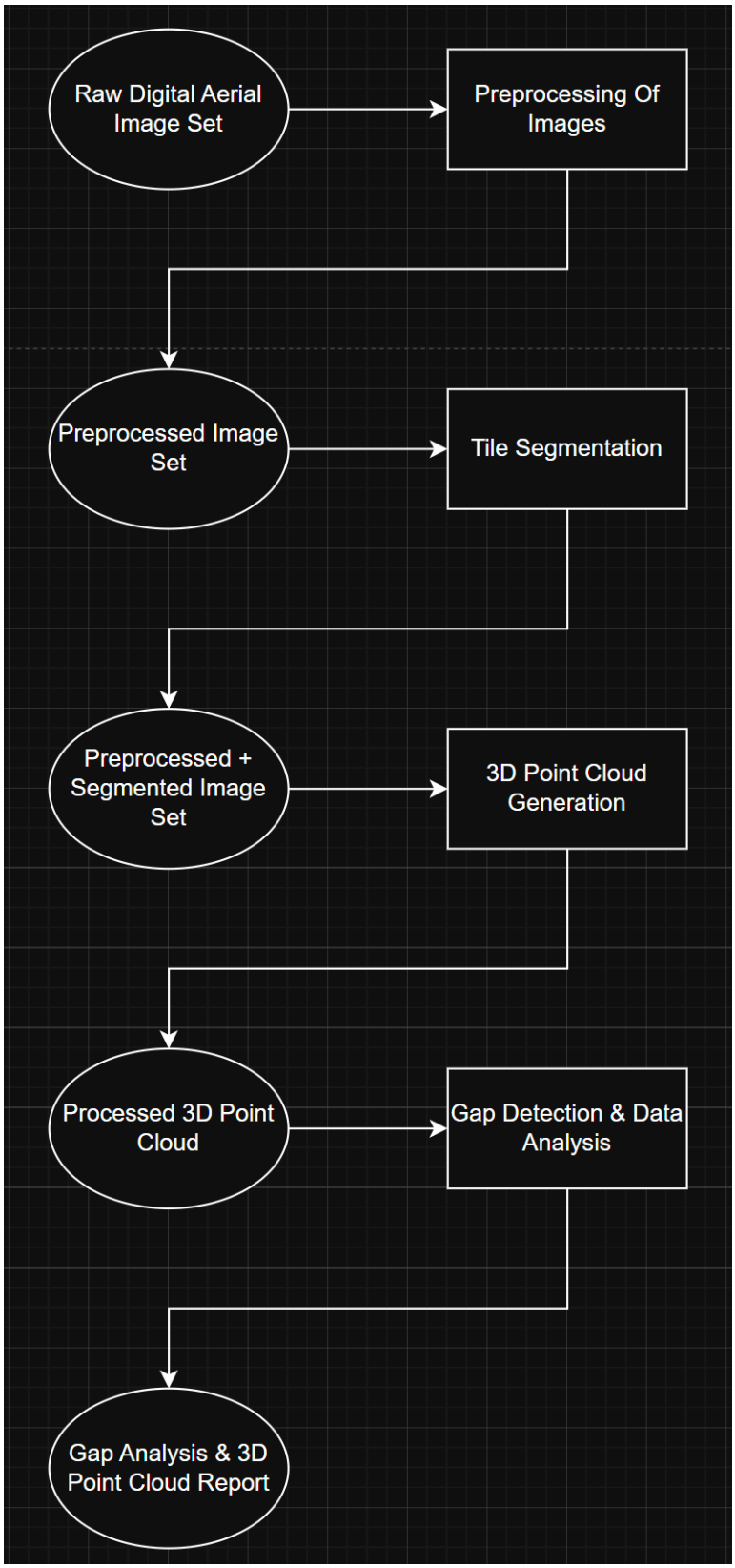
Raw Digital Aerial Image Set
Preprocessing Of Images
Preprocessed Image Set
Tile Segmentation
Preprocessed + Segmented Image Set
3D Point Cloud Generation
Processed 3D Point Cloud
Gap Detection & Data Analysis
Gap Analysis & 3D Point Cloud Report

Fig 1. Architecture Diagram

3. Use Case 1

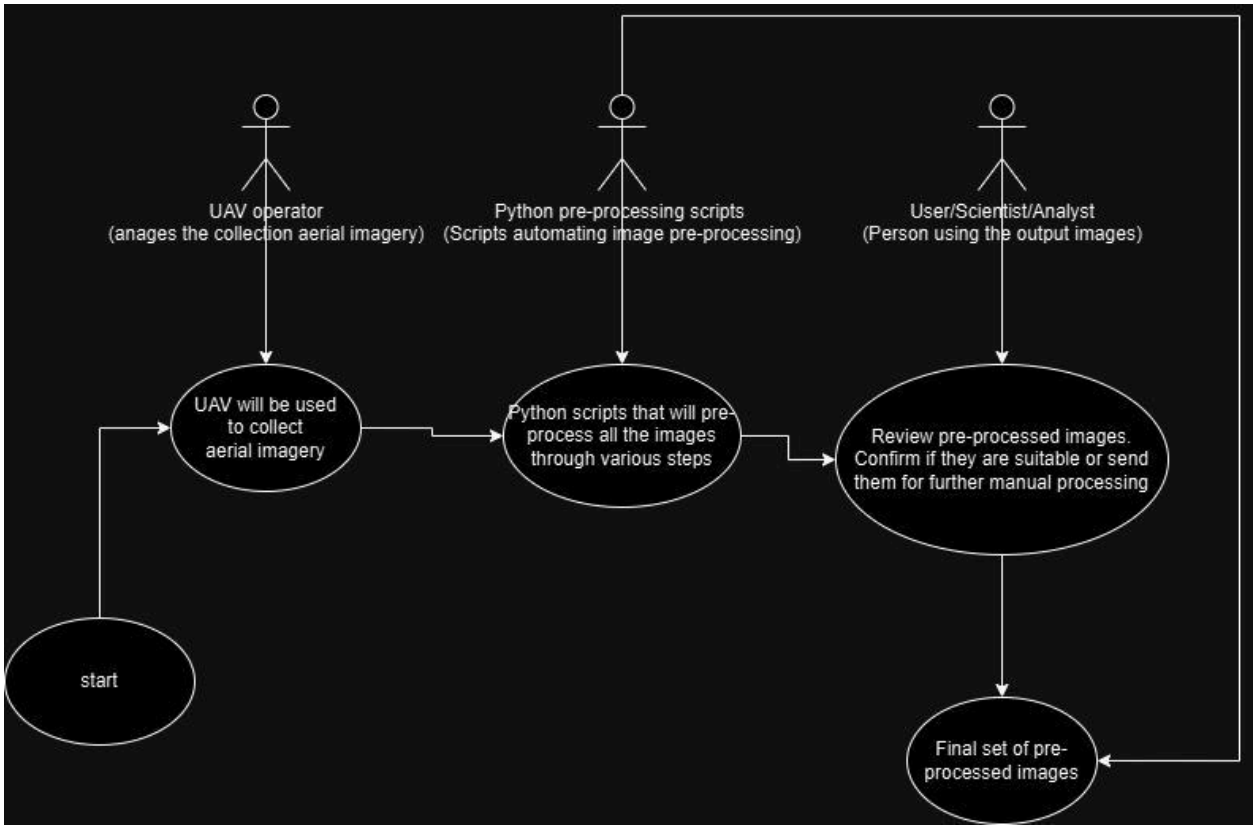

Fig 2. Diagram illustrating Use Case 1

Actors	UAV collected Images, Python scripts to pre-process images, user (scientist/analyst)
Description	A survey and images of an area may be collected. This data will be pre-processed for clarity, filtering out unusable images.
Data	Collected images from the ground and drone
Stimulus	Desired images uploaded to pre-process and filter
Response	Return of best usable images
Comments	The pre-processing software aims to automate as much of the photo enhancement and filtering as possible, minimizing manual intervention. Usable images are forwarded to WebODM, and flagged photos may be manually reviewed if necessary. This process reduces errors that could arise during the 3D modeling phase and increases the likelihood of generating a high-quality point cloud.

During pre-processing the photos in the image set may need processing due to factors including blur or lens distortion, incorrect white balance, lack of lighting or images that are too bright. These images may introduce inaccuracies in the 3D point cloud and negatively impact the result if used. The pre-processing step aims to automatically fix these images in a standardized manner. This will effectively cut down on the time and effort spent filtering through each photo manually. The result is a folder with the preprocessed image set which can be manually reviewed by the ecologist to determine if they are sufficiently processed or not.

3.1 Logical View

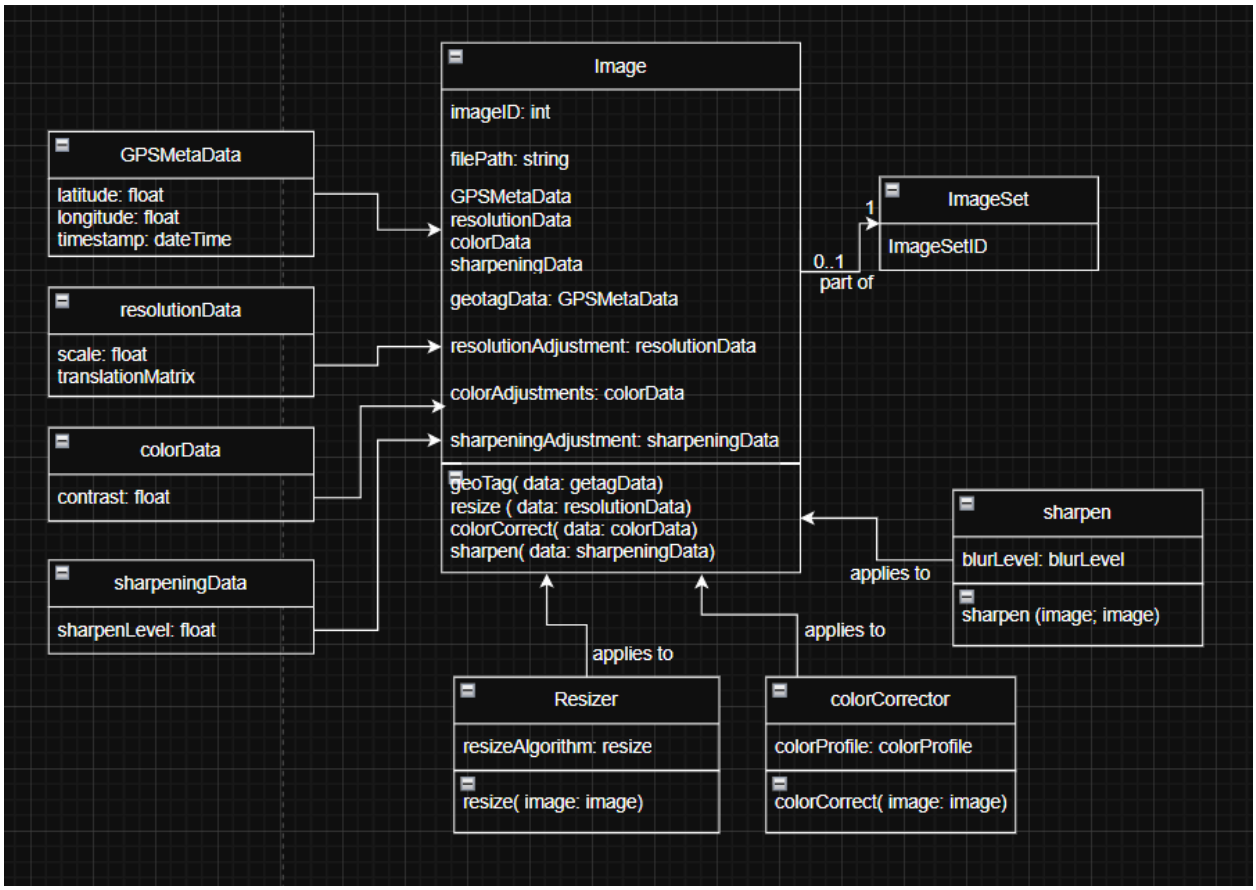

Figure 3. Use Case 1 Logical View Diagram

This logical view diagram represents the image pre-processing classes and their dependencies. Each set of UAV collected images undergoes several modifications so they can be efficiently used for point cloud generation. The entities and relationships in this diagram are:

- **ImageSet**
    - Represents a collection of images
    - Attributes:
        - **ImageSetID:** Unique identifier for an image set
- Image
    - Represents an individual image
    - Attributes:
        - **ImageID:** unique identifier for an image

        - **GPSMetaData:** class within Image to define what GPS Metadata is
        - **resolutionData:** class within image to define what resolution data is
        - **ColorData:** class within image to define what color data is
        - **filePath:** file path where the image is stored
        - **geotagData:** variable that holds the GPS Metadata for the image
        - **SharpeningData:** class within image to define sharpening data
        - **resolutionAdjustment:** holds values for the resolution adjustment made on image
        - **ColorAdjustment:** holds values for color adjustments made on the image
        - **SharpeningAdjustment:** holds values for sharpening adjustments made on image
    - Methods:
        - **GeoTag():** method to geotag images (geotag: add GPS Metadata)
        - **resize():** method to apply resolution resizing adjustments to image
        - **ColorCorrect():** method to apply color correction to image
        - **Sharpen():** method to apply sharpening to image
- GPSMetadata:
    - Class to define what GPS metadata is
        - Holds the latitude, longitude and timestamp
- ResoutionData:
    - Class to define what alignmentData is
        - Holds the scale data and the transformation matrix
- ColorData:
    - Defines what colorData is
        - Holds the contrast profile of image
- SharpeningData:
    - Defines what sharpening data is
        - Holds blur level and sharpening thresholds of images
- Resizer
    - Class responsible for resolution resizing of images
    - Attributes:
        - ResizeAlgorigthm: Resize algorithm and parameters used for adjustments
    - Methods:
        - **Resize():** method to apply resize adjustments on image
- ColorCorrector
    - Class responsible for color correction of images

    - Attributes:
        - **ColorProfile:** color profile and data to be used for color corrections
    - Method:
        - **ColorCorrect():** applies color correction to an image
- sharpen
    - Class responsible for sharpening of images
    - Attributes:
        - **blurLevel:** blurLevel data of image
    - Method:
        - **sharpen():** applies sharpening to an image
-

3.2 Process

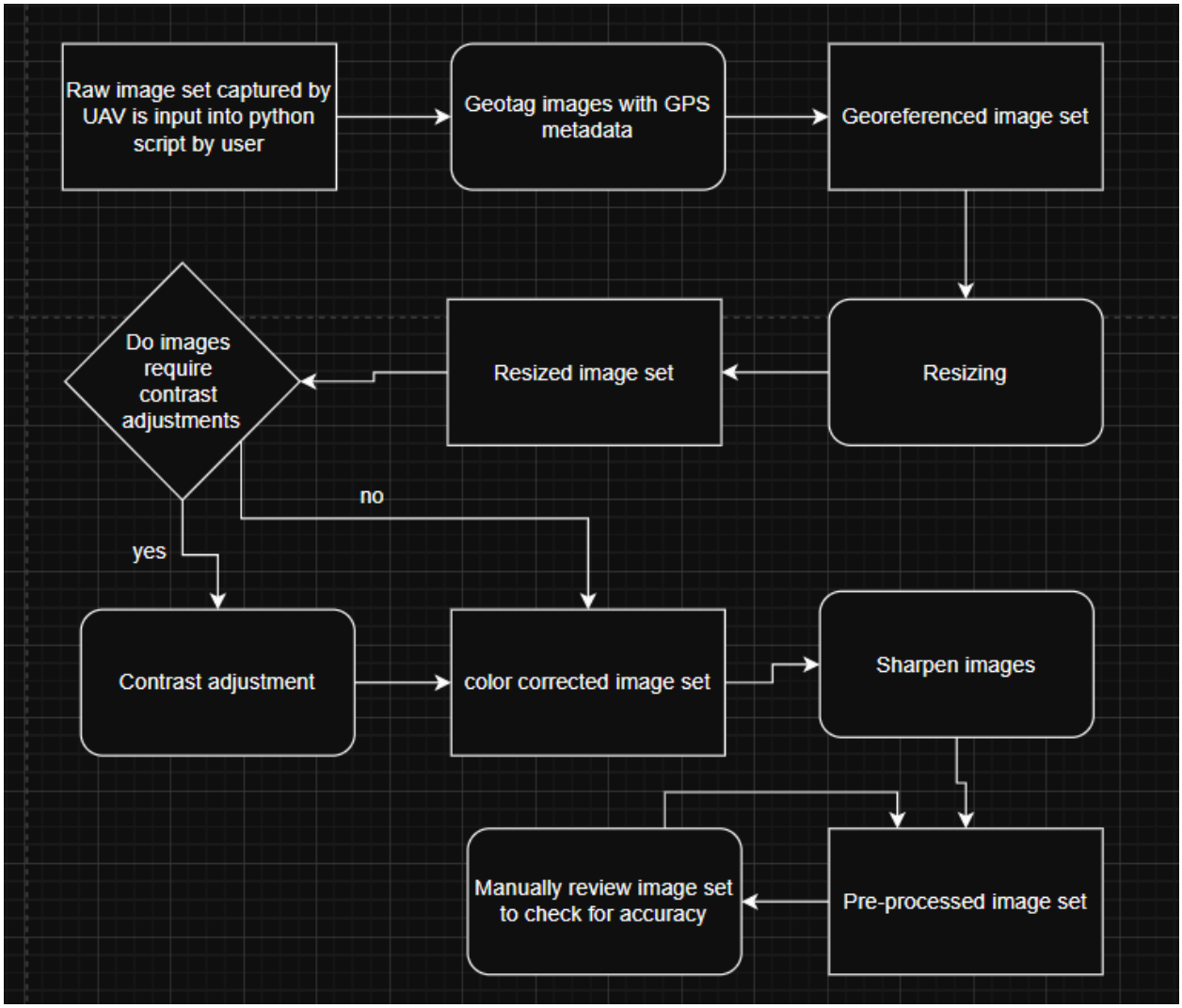

Figure 4. Use Case 1 Process Diagram

This workflow outlines the steps to preprocess UAV captured images for subsequent point cloud generation. The process ensures that the images are properly aligned, georeferenced, and color-corrected to maintain consistency and quality.

1. **Raw Image Set Captured by UAV**:
    a. The process begins with a collection of raw images collected from the UAV. These images are sent down the pipeline for step-by-step processing.
2. **Geotagging Images**:

    a. The first step involves geotagging the images to ensure they are correctly georeferenced with GPS metadata.
        i. We are assuming that GPS metadata is collected during the image collection process as that is usually a part of our sponsor's workflow.
    b. The outcome of this step is an image set where each image is georeferenced with the corresponding GPS data
3. **Image Resizing**:
    a. The georeferenced images are then processed for resizing. All images are resized to a smaller standardized resolution which helps reducing processing times in later stages.
4. **Contrast Consistency Check**:
    a. The aligned images are checked for consistent contrast levels in the image set.
        i. If the contrast levels are consistent, the image is set is sent to the final step
        ii. If the image colors are not consistent (i.e some images are too dark/too light) adjustments are applied across the images.
5. Sharpening
    a. Images are sharpening to reduce blur and improve detail in the images
6. **Pre-Processed Image Set**:
    a. The final pre-processed image set which is ready for subsequent point cloud generation.
    b. Will be checked, once manually at the end by the user to confirm that it is indeed a usable image set.

3.3 Development

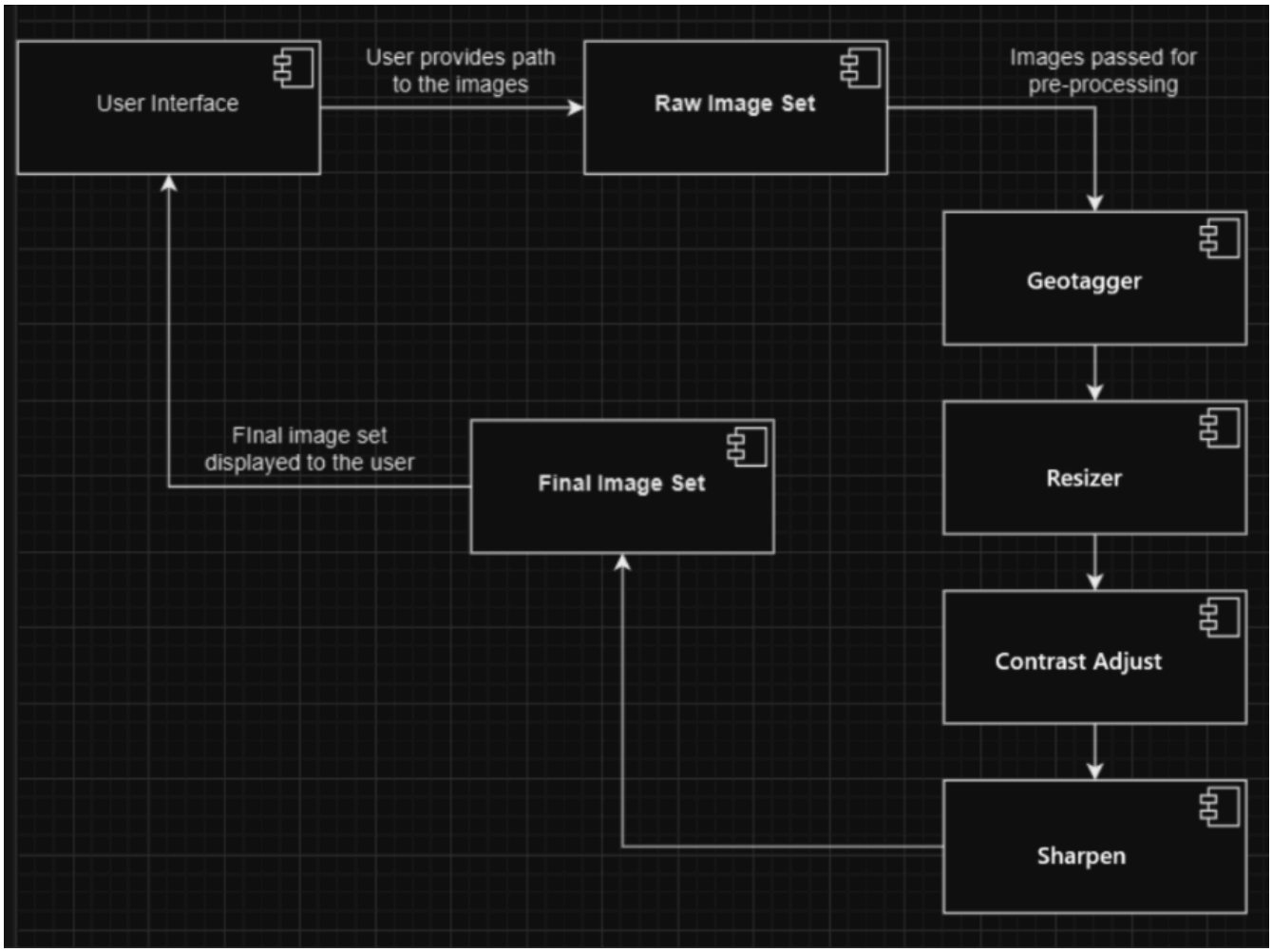

Figure 5. Use Case 1 UML Components Diagram

This diagram shows the different components involved in the image pre-processing workflow. The user will use the UI/Command Line and provide the path to the raw image set. That image set will be used by the UI and will be sent through all the steps of the processing pipeline (geotagging, resizing etc.) and once passed through all those components, a final image set will be generated which will be displayed in the user interface to the user.

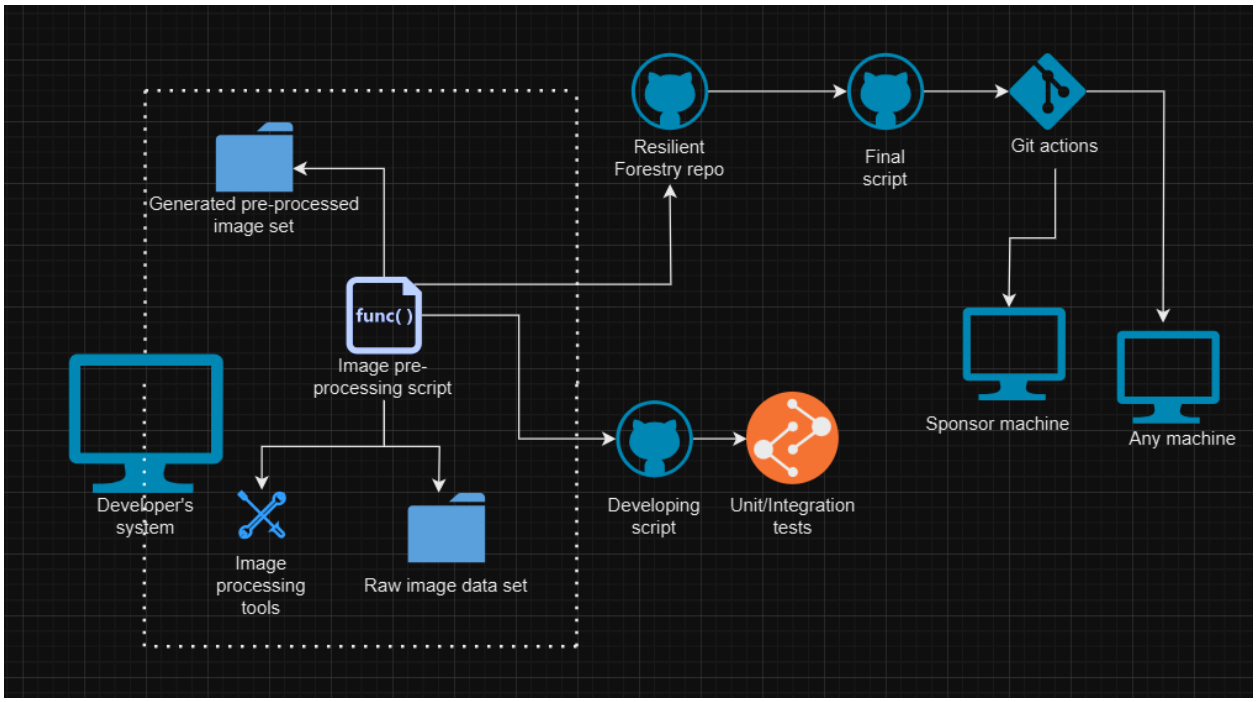

Figure 6. Use Case 1 Dev-Ops Diagram

The pre-processing script and all the corresponding data like the raw image set and image preprocessing tools will be stored locally on the developer's system. Once tested internally with integrated and unit testing, the script will be uploaded to Resilient Forestry's repository from where it will be deployed onto the sponsor's systems or any other machines that the sponsor desires where they can start using the script locally to pre-process images.

3.4 Physical Infrastructure

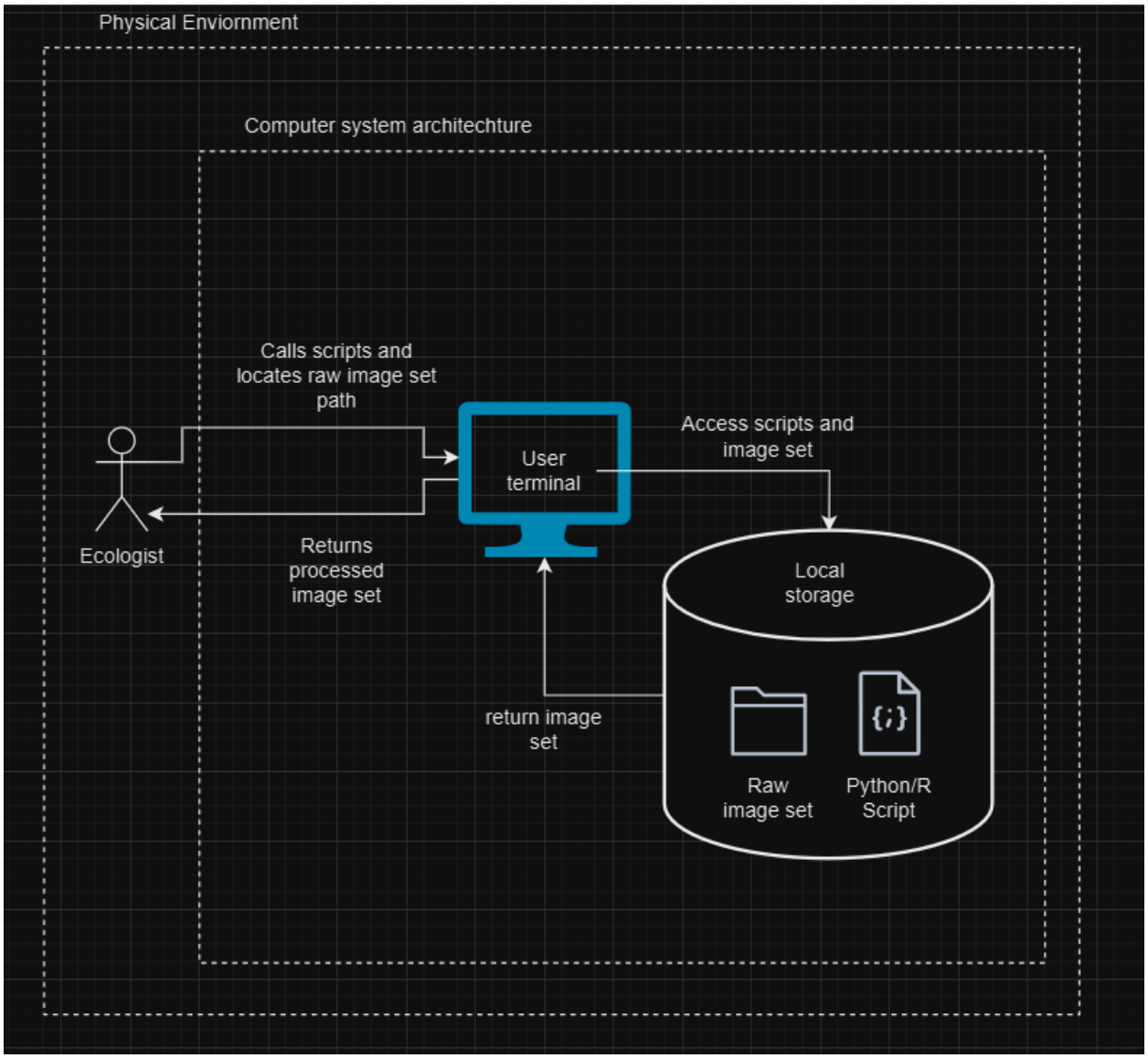

Figure 7. Use Case 1 Physical Diagram

This physical diagram for the preprocessing workflow has 2 main components: the physical environment and the computer system architecture.

The physical environment has the ecologist, who will use their system to run the preprocessing script and also provide the raw image set path to the script. The local storage of their system (HDD or SSD) will store the raw image set and the Python script.

4. Use Case 2

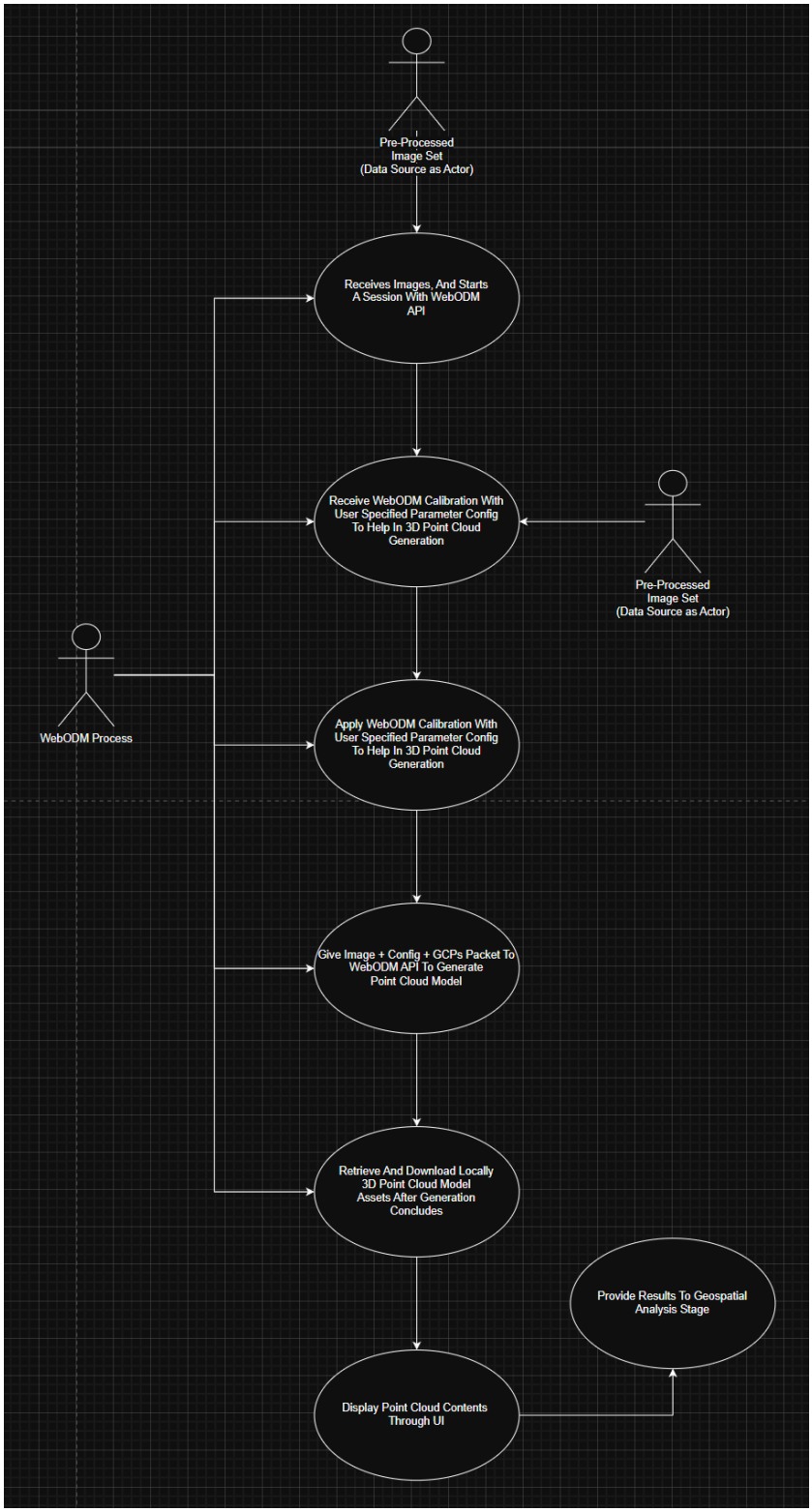
Pre-Processed
Image Set
(Data Source as Actor)
Receives Images, And Starts
A Session With WebODM
API
Receive WebODM Calibration With
User Specified Parameter Config
To Help In 3D Point Cloud
Generation
Pre-Processed
Image Set
(Data Source as Actor)
WebODM Process
Apply WebODM Calibration With
User Specified Parameter Config
To Help In 3D Point Cloud
Generation
Give Image + Config + GCPs Packet To
WebODM API To Generate
Point Cloud Model
Retrieve And Download Locally
3D Point Cloud Model
Assets After Generation
Concludes
Provide Results To Geospatial
Analysis Stage
Display Point Cloud Contents
Through UI

Fig 8. Diagram illustrating Use Case 2

Actors	Pre-processed Images, Python Calibration Scripts, WebODM, User (Scientist/Analyst)
Description	Pre-processed images are uploaded to WebODM for 3D reconstruction. Python scripts automatically calibrate WebODM by applying predefined modes that adjust critical parameters based on the project's requirements. These modes optimize image stitching and 3D model generation by adjusting algorithms and settings in WebODM.
Data	Pre-processed images with metadata, calibration scripts, and initial point cloud
Stimulus	Pre-processed photos uploaded to WebODM for 3D reconstruction
Response	Point cloud model generated and displayed with parameters adjusted automatically, with the option for manual overrides by the analyst.
Comments	The Python scripts automate the tuning of various parameters such as feature detection (e.g., SIFT, SURF), optimization methods, and image processing settings in WebODM. This automated calibration simplifies the reconstruction process, reducing manual intervention while still allowing the flexibility to fine-tune results manually if necessary.

Once the images are initially uploaded into WebODM, multiple algorithms can be run to match the 3D point cloud to the ground control points taken. During the 3D modeling process certain algorithms will be tested and the best algorithms with the best parameters will be returned. Ideally, the chosen algorithms will do the image-stitching and match similar features across the images better than the other algorithms that weren't chosen. Further algorithms will be applied to find the best parameters (max iterations, convergence, tolerance, damping, etc.) and morphological filters to remove vegetation if needed. This entire process will still allow for manual input and override as the scientist best see fit.

4.1 Logical View

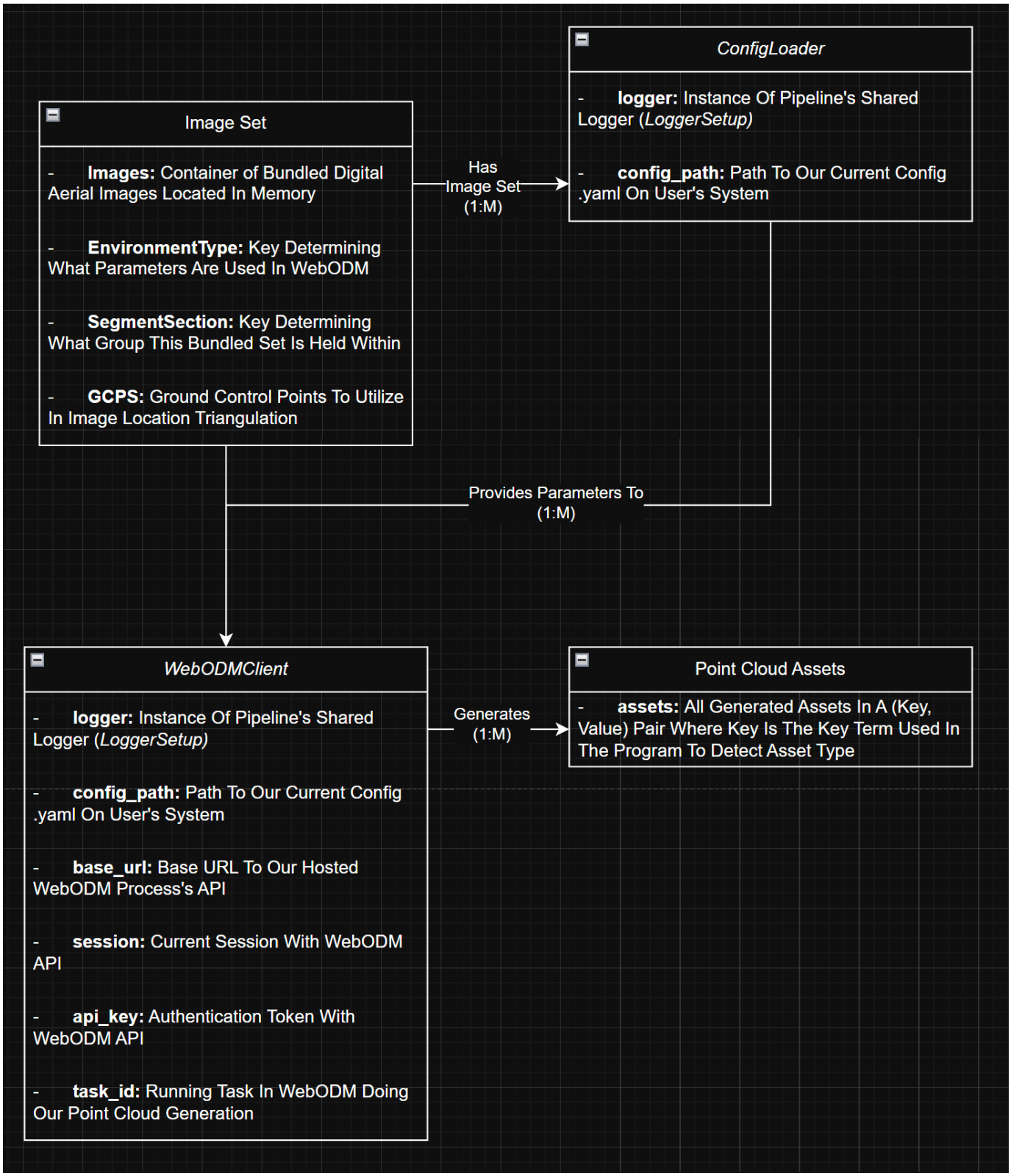

Figure 9. Use Case 2 Logical View Diagram

The Logical View Diagram for the WebODM benchmarking pipeline defines the core classes and their interdependencies, which collectively represent the essential data structure for managing image processing, parameter configuration, and point cloud generation. Central to this design is the Image Set, an abstract object that encapsulates bundled images stored in memory along with an EnvironmentType attribute that determines the specific parameters to be used in WebODM. Rather than being a direct code implementation, the Image Set serves as a conceptual container that initiates the pipeline by linking to a dedicated ConfigLoader. As well as this, this also contains the segment section (SegmentSection) which the image set lies in. Ground Control Points (GCPs) are utilized in cross referencing with the GPS metadata of images to help in increasing the precision of our generated point cloud.

The ConfigLoader class, which is tied to each Image Set in a one-to-one relationship, is responsible for handling the configuration settings needed throughout the pipeline. It maintains a logger—an instance of the shared LoggerSetup—to record activities and errors, and it also holds a config_path pointing to the current configuration file in YAML format on the user's system. This setup ensures that every Image Set has an associated configuration source that feeds necessary parameters to other components.

One such component is the WebODMClient, a class designed to interact with the WebODM API. Each WebODMClient instance inherits the configuration parameters provided by the ConfigLoader, including the logger and config_path, and further specifies a base_url for API access, along with session management, an authentication token (api_key), and a task_id corresponding to the running point cloud generation task. The relationship between Image Set and WebODMClient is one-to-many, allowing a single Image Set to be processed by multiple WebODMClient instances concurrently, while the ConfigLoader supports a one-to-many distribution of configuration parameters to these clients.

Completing the picture is the Point Cloud Assets, an abstract class that captures the output of the processing pipeline. This class manages all generated assets using a flexible key-value pair structure, where each key represents a specific asset type detected by the program. By doing so, it provides a structured yet adaptable means of handling the various assets produced during the 3D reconstruction process.

Overall, this logical view offers a comprehensive structural map that demonstrates how data flows and is organized within the pipeline. It emphasizes modularity by clearly delineating responsibilities— from the abstraction of the Image Set and its environmental context to the configuration managed by ConfigLoader, the API

interactions facilitated by WebODMClient, and finally, the asset management encapsulated in Point Cloud Assets.

4.2 Process

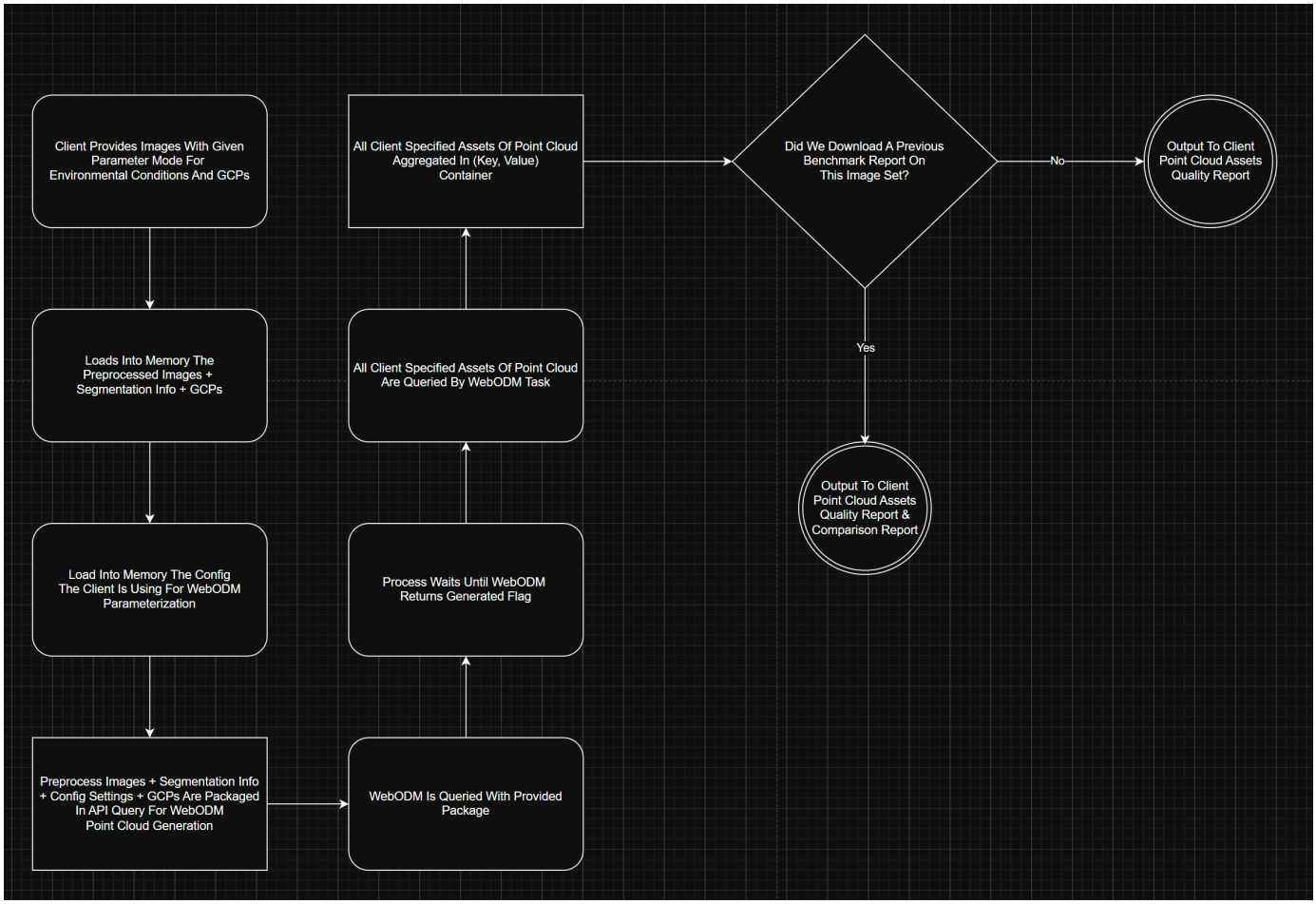

Figure 10. Use Case 2 Process Diagram

This Activity Diagram represents the sequence of steps in WebODM's benchmarking pipeline from the client's initial input to the final delivery of point cloud assets and reports, emphasizing the system's automated and hands-off operation.

- **Start:** The process is initiated when the client provides images with a specified parameter mode for environmental conditions and GCPs. This input triggers the loading of preprocessed images, segmentation information, and GCPs into memory, while simultaneously loading the configuration that the client is using for WebODM parameterization.

- **Packaging for API Query:** The system then packages the preprocessed images, segmentation info, configuration settings, and GCPs into an API query, which is sent to WebODM to initiate point cloud generation.
- **Process Wait:** The pipeline enters a waiting state until WebODM returns a generated flag, indicating that the point cloud creation process is complete.
- **Asset Aggregation:** Once the flag is received, the system queries all client-specified point cloud assets from the running WebODM task and aggregates these assets into a structured (key, value) container.
- **Comparison with Previous Reports:** At a decision point, the system checks whether a previous benchmark report for the same image set exists. If a previous report is found, both a quality report and a comparison report are generated, providing insights into changes in quality and performance; otherwise, only a quality report is produced.
- **Output and Client Delivery:** The final output—comprising the aggregated point cloud assets, along with the quality report (and comparison report when applicable)—is then packaged and delivered to the client for review.

This activity diagram offers a high-level view of the automated process, clearly illustrating the logical flow from data ingestion and configuration loading through API querying, asset aggregation, decision-based report generation, and final client delivery, thereby ensuring an efficient, adaptive pipeline.

4.3 Development

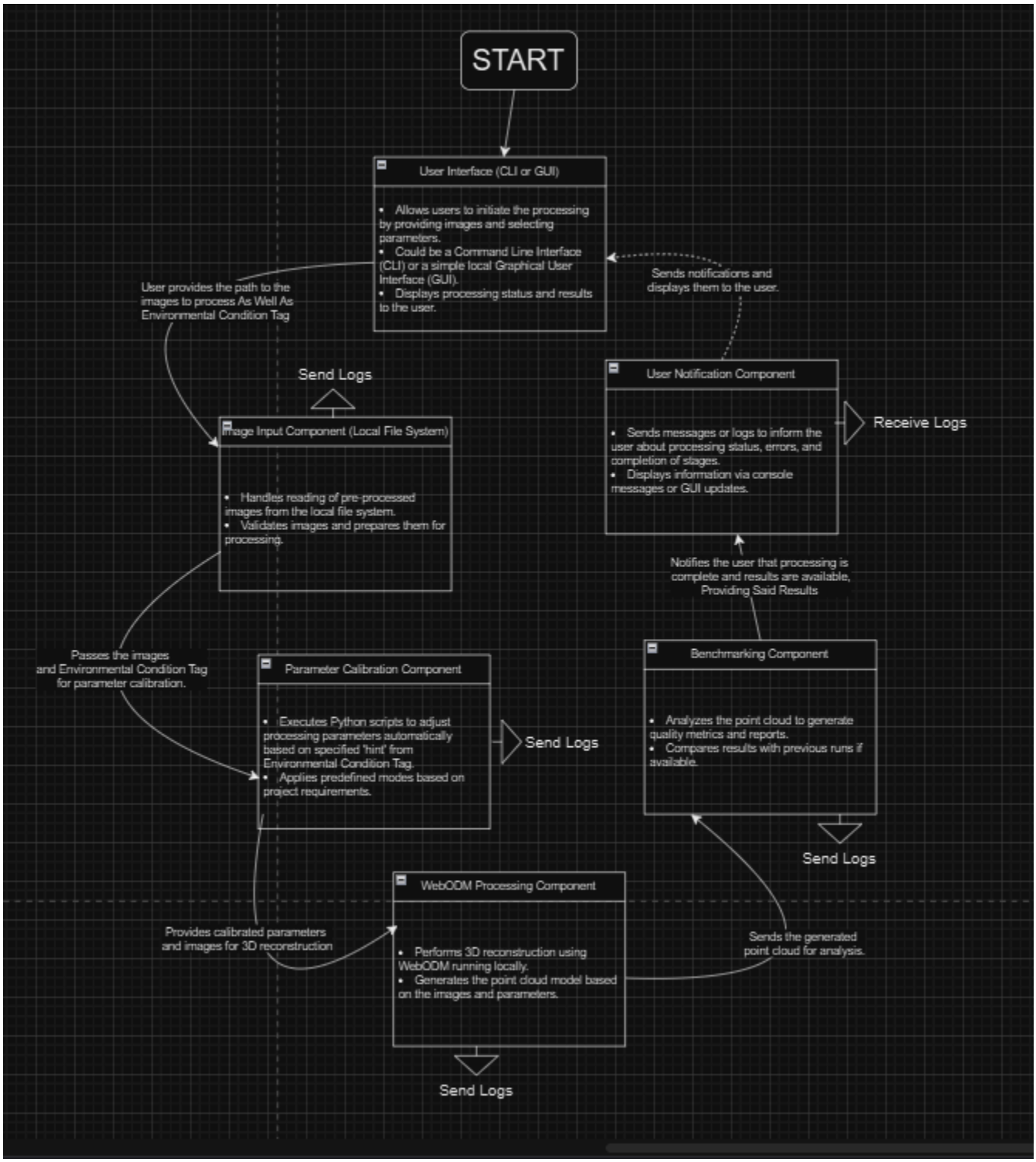

Figure 11. Use Case 2 UML Components Diagram

The process starts at the User Interface Component (CLI or GUI), where users initiate the pipeline by providing paths to images and providing an environmental condition tag (simple string like “fog”, “rainy”) to help parameterize WebODM for proper point cloud generation. The interface component facilitates user input, displays processing status updates, and outputs final results to the user through intuitive means (Terminal GUI, or Framework GUI).

Next, the **Image Input Component** handles image reading from the local file system. It verifies and prepares images for processing (validate as in ensures proper image extension and prepare as in load into memory), sending logs of its activity to the **User Notification Component**. The verified images are then passed to the **Parameter Calibration Component**, where Python scripts adjust WebODM’s parameters based on predefined modes based on the environmental condition tag provied. These modes are tailored for specific project needs, such as environmental conditions, enhancing the accuracy of subsequent 3D reconstruction steps. Calibration logs are also sent to the **User Notification Component** to inform the user of progress.

Once parameters are calibrated, both the images and adjusted parameters are handed off to the **WebODM Processing Component**. This component performs the main 3D reconstruction tasks locally, generating the initial point cloud. During processing, logs are continuously sent to the **User Notification Component** to keep the user updated on the reconstruction’s progress. After the point cloud is created, it is transferred to the **Benchmarking Component**.

The **Benchmarking Component** analyzes the point cloud to produce quality metrics and generates a benchmark report. It compares the current results with previous runs if available, providing insights into performance changes over time. The benchmarking component then sends logs and comparison results back to the **User Notification Component**, which provides the user with final notifications and results.

Throughout the process, each major component (Image Input, Parameter Calibration, WebODM Processing, and Benchmarking) sends logs and status updates to the **User Notification Component**, ensuring that the user is informed at every stage, from initiation through to result generation. The **User Notification Component** is coupled with the **User Interface Component** to ensure we display these logs to the user in a neat and digestible manner. This modular setup enables clear separation of responsibilities across components, allowing for flexibility, easy debugging, and comprehensive monitoring of the pipeline’s execution.

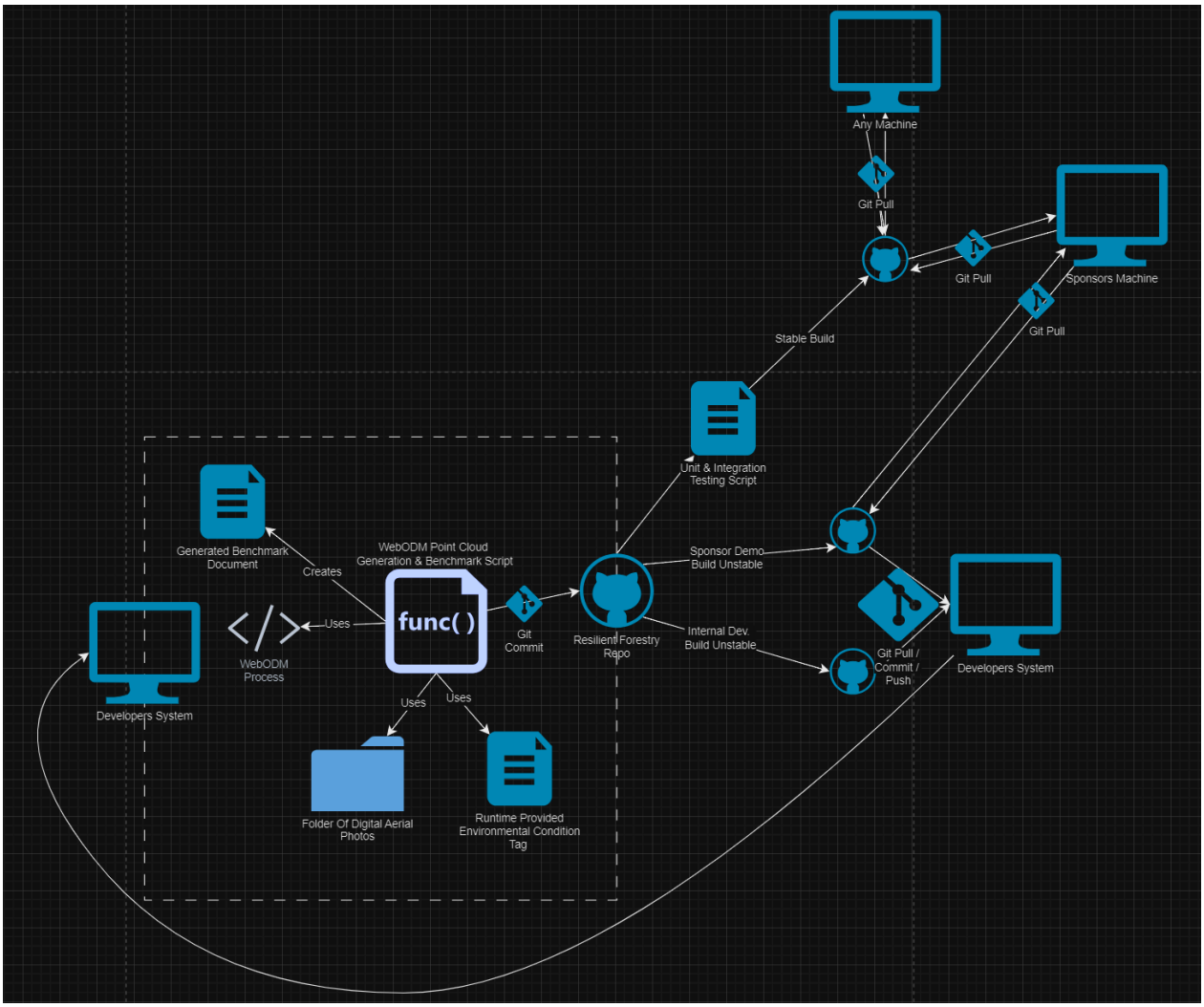

Figure 12. Use Case 2 Dev-Ops Diagram

The diagram starts with the **Developer's System**, where local development occurs using **WebODM** software process and the **WebODM Point Cloud Generating & Benchmark Script**. The script utilizes a **Folder of Digital Aerial Photos** and a **Runtime Provided Environmental Condition Tag** to create a **Generated Benchmark Document**, outputting benchmark data.

Once changes are made to the development script, they are committed to the **Resilient Forestry Git Repository**. The repository hosts both the code and scripts, allowing developers to **Git Commit, Push, and Pull** code updates. After each commit, a **Unit & Integration Testing Script** is triggered to validate the build if pushed to the stable branch. If the build is stable, it can be pulled by various machines, including the **Sponsor's Machine** for stable demo builds they can prematurely utilize in the field, as well as **Any Machine** for general access, and other **Developer Systems** for collaborative development. In cases of unstable or builds still being developed on, they are pushed on **Internal Dev (unstable)** or **Sponsor Demo (unstable)** builds.

This setup ensures that all code changes are tested locally and then systematically validated in shared environments, enabling reliable collaboration and testing across multiple machines. The DevOps pipeline helps developers verify functionality before deployment to the sponsor or production, facilitating a streamlined and controlled development process for the WebODM benchmarking pipeline.

4.4 Physical Infrastructure

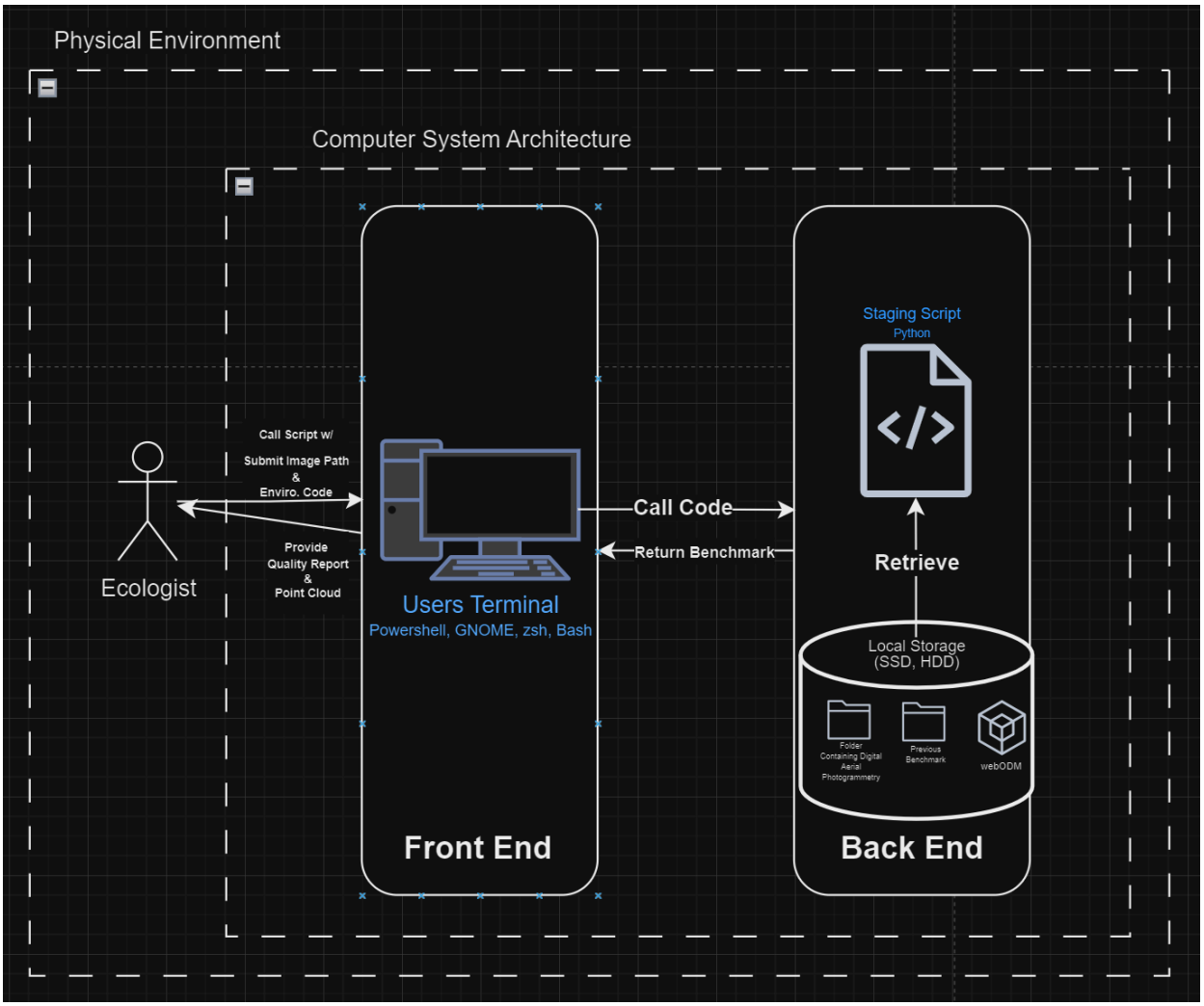

Figure 13. Use Case 2 Physical Diagram

This **Physical Diagram** illustrates the interaction between the physical environment, user terminal, and backend processes for the WebODM benchmarking pipeline within our defined scope. The system enables the **Ecologist** to perform locally executed processing tasks using minimal input while automating the majority of the pipeline.

1. **Start - Physical Environment**:
    - The **Ecologist** begins the process by providing an image path and an environmental condition tag (e.g., "fog," "rainy," or "sunny") through the **User's Terminal**. This simple input helps parameterize the WebODM

pipeline for optimal point cloud generation based on the specific environmental conditions of the dataset.

2. **Front End - User's Terminal**:
    - The **User's Terminal** serves as the primary interaction point, using common shell environments for varying types of OS such as PowerShell, GNOME, Zsh, or Bash.
    - Through the terminal, the ecologist calls a **Staging Script**, which orchestrates the entire pipeline. The terminal provides the user with real-time feedback on the pipeline's progress, error notifications, and final outputs, such as the **Quality Report** and **Point Cloud**.
3. **Back End - Processing and Retrieval**:
    - The **Back End** contains all the processing logic and data retrieval tools required for the pipeline. The **Staging Script**, written in Python, retrieves input from the terminal and interfaces with local storage to access:
        - **Digital Aerial Photogrammetry**: The folder containing the pre-processed aerial images.
        - **Previous Benchmarks**: Historical data used for comparative analysis.
        - **WebODM Software Process**: The core system that executes 3D reconstruction and benchmarking tasks.
    - The **Staging Script** processes the input images using WebODM and applies adjustments based on the environmental condition tag provided by the ecologist. Once processing is complete, it generates benchmark reports and point clouds, which are then returned to the front end.
4. **Interconnection Between Front End and Back End**:
    - The **Call Code** is sent from the front end (User's Terminal) to the back end (Staging Script) to initiate the process.
    - The **Back End** retrieves and processes the data and sends back the **Benchmark Report** and **Point Cloud** to the **User's Terminal**. The results are then displayed to the ecologist for further review.
5. **Output and Feedback**:
    - The system ensures that the ecologist receives all necessary outputs in an efficient, user-friendly manner. Feedback is continuously provided via the terminal to ensure transparency during the process, allowing for easy debugging and verification.

This diagram emphasizes the modular and localized architecture of the system, ensuring seamless interaction between the user and the computational components. By keeping all components within a controlled, local environment, the system achieves a streamlined and efficient workflow tailored for ecologists in the field or lab.

5. Use Case 3

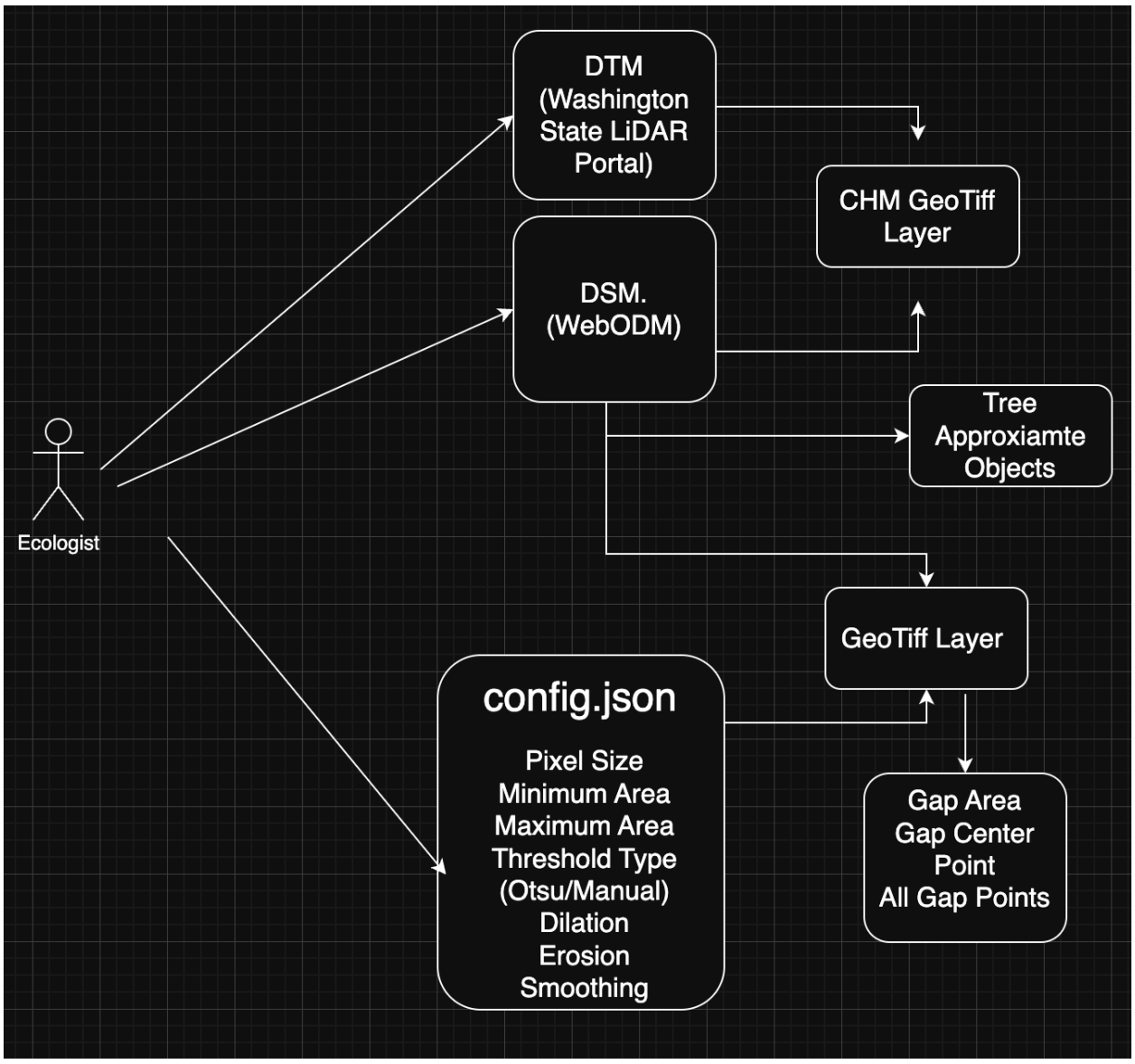

Figure 14. Diagram Illustrating Use Case 3

Actors	3D Point Cloud, Python Automated Scripts, User (Scientist/Analyst)
Description	Once the processed 3D point cloud is generated, Python scripts are used to extract and visualize key environmental data. Analysts can perform specialized geospatial analyses such as measuring tree canopy cover, identifying clump sizes, and delineating land use patterns
Data	Processed 3D point cloud, Washington State LiDAR data

Stimulus	Processed 3D point cloud and additional data run using python scripts and libraries
Response	Geospatial analysis results returned as GIS layers or structured data reports
Comments	**Use Case 3** enables the ecologist to select and configure specific parameters to generate a Canopy Height Model (CHM) and calculate the gap area. The ecologist must define the parameters for their definition of a gap and provide historical LiDAR data from the Washington State LiDAR portal as input.

Use Case 3 leverages data from the Washington State LiDAR Portal alongside the point cloud generated from WebODM to create a Canopy Height Model (CHM) and streamline the calculation of forest gap areas. To further tailor the analysis, a config.json file is employed, allowing users to define thresholds and apply filters for identifying and isolating the desired gaps within the forest canopy. This approach ensures a more efficient and customizable process for analyzing spatial variations and structural patterns in forested environments.

5.1 Logical View

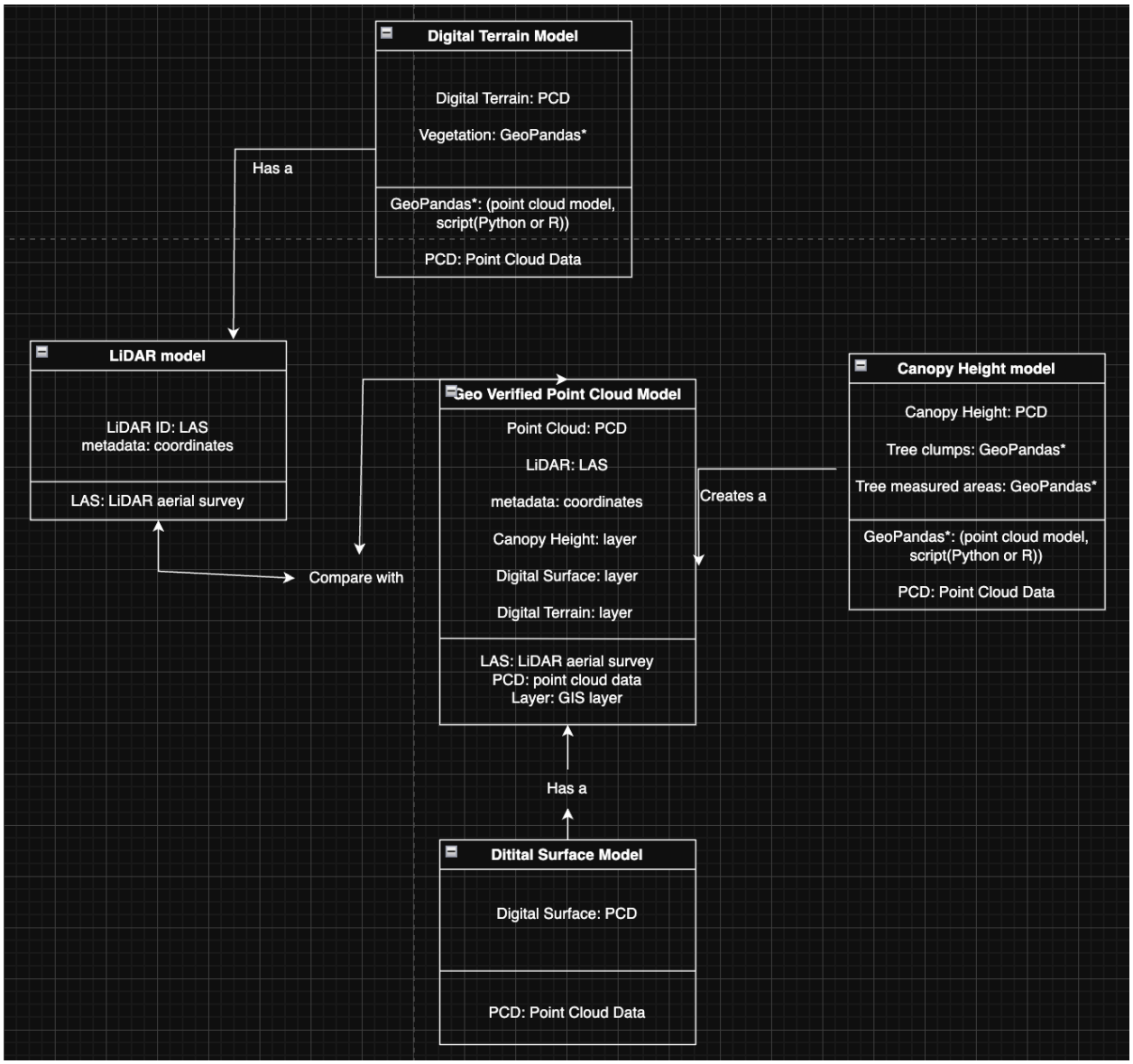

Figure 15. Use Case 3 Logical View Diagram

This logical view diagram represents the point cloud data during post-processing classes and their dependencies. Each point cloud model is compared with LiDAR data if available and then calculations are done resulting in GIS layers. The entities and relationships in this diagram are:

- **LiDAR Model**
    - Represents any previously collected LiDAR data
    - Attributes:

        - **LiDAR ID:** Unique identifier for the LiDAR data
        - Metadata: Coordinates of points on the model
    - Methods:
        - LAS: LiDAR datatype for files
- Geo-Verified Point Cloud Model
    - Represents an individual image
    - Attributes:
        - Point Cloud: unique identifier for the model
        - LiDAR: The LiDAR data the point cloud was compared against
        - MetaData: holds the GPS Metadata for the image
        - Canopy Height: Holds the calculations of the canopy height model
        - Digital Surface: Holds the calculations of the digital surface model
        - Digital Terrain: Holds the calculations of the digital terrain model
    - Methods:
        - Model: Raster model
        - Layer: GIS (graphical information system) layer
        - PCD: point cloud data
- Canopy Height Model
    - Class responsible for the canopy height model, size of tree clumps, and the tree measured area
    - Attributes:
        - Canopy Height: Holds the calculations of the canopy height model
        - Tree clumps: Holds the calculations of each tree clump in the model
        - Tree measured area: Holds the calculations of the measured area of each tree
    - Methods:
        - GeoPandas: Script run using the GeoPandas library and the geo-verified point cloud model.
        - PCD: point cloud data
- Digital Surface Model
    - Class responsible for digital surface model
    - Attributes:
        - Digital Surface: Holds the digital surface model
    - Methods:
        - GeoPandas: Script run using the GeoPandas library and the geo-verified point cloud model.
        - PCD: point cloud data
- Digital Terrain Model

- Class responsible for digital terrain model and the amount of vegetation
- Attributes:
  - Digital Terrain: Holds the calculations of the digital surface model
  - Vegetation: Holds the calculations of the measured area of vegetation
- Method:
  - GeoPandas: Script run using the GeoPandas library and the geo-verified point cloud model.
  - PCD: point cloud data

5.2 Process

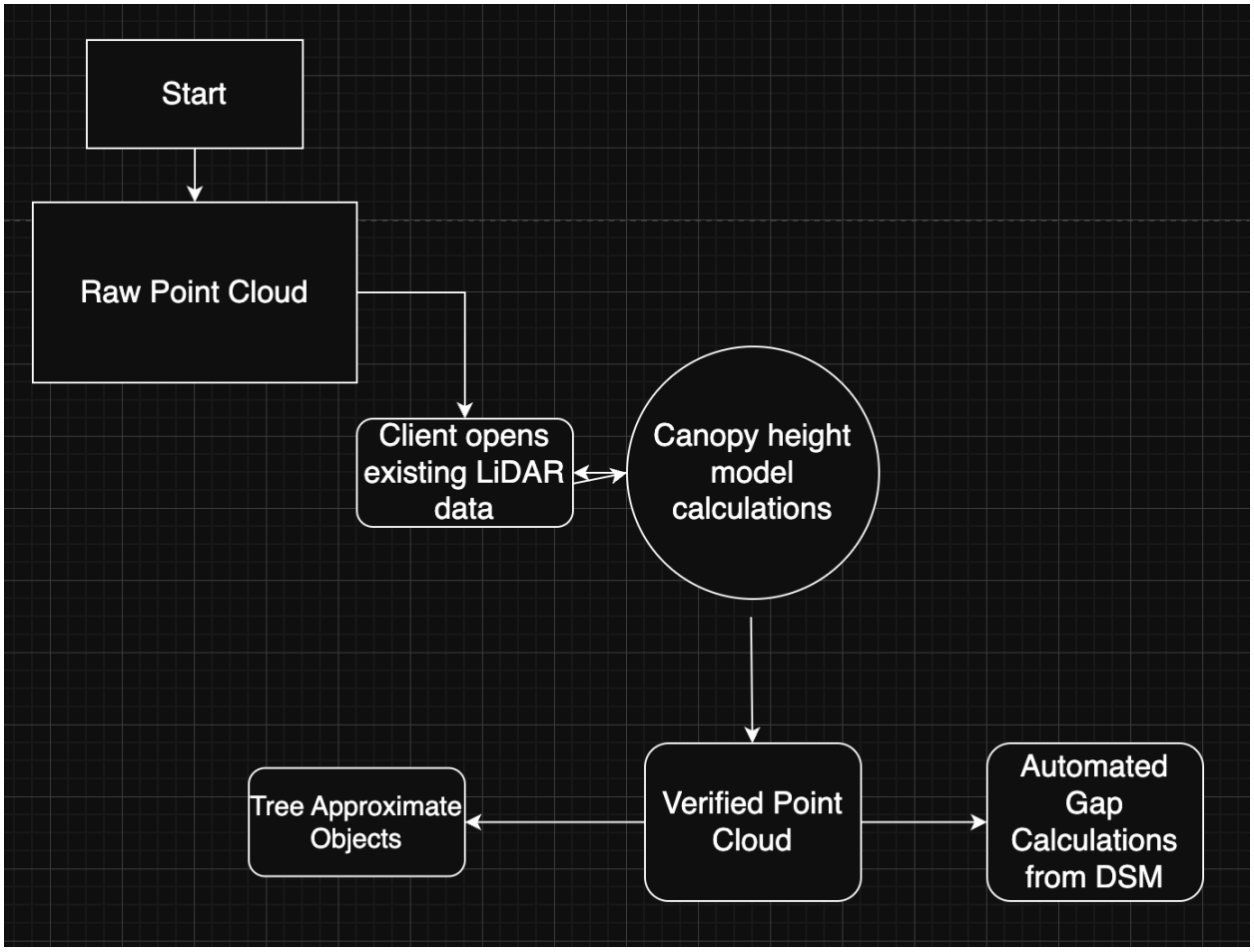

Figure 16. Use Case 3 Process Diagram

The process diagram shows each step in the workflow and where decisions must be made.

1. **Raw_Point_Cloud**: The raw point cloud is the point cloud passed in
2. **LiDAR Data:** LiDAR Data obtained from the Washington State LiDAR potal is used to gather the DSM
3. **Canopy Height Model:** The canopy height model is created using the DTM from the Washington State LiDAR and the raw point cloud DSM
4. **Automated Gaps**: The gaps are automatically found using the verified DSM.
5. **Tree Approximate Objects:** The tree approximate objects are found using the verified point cloud

5.3 Development

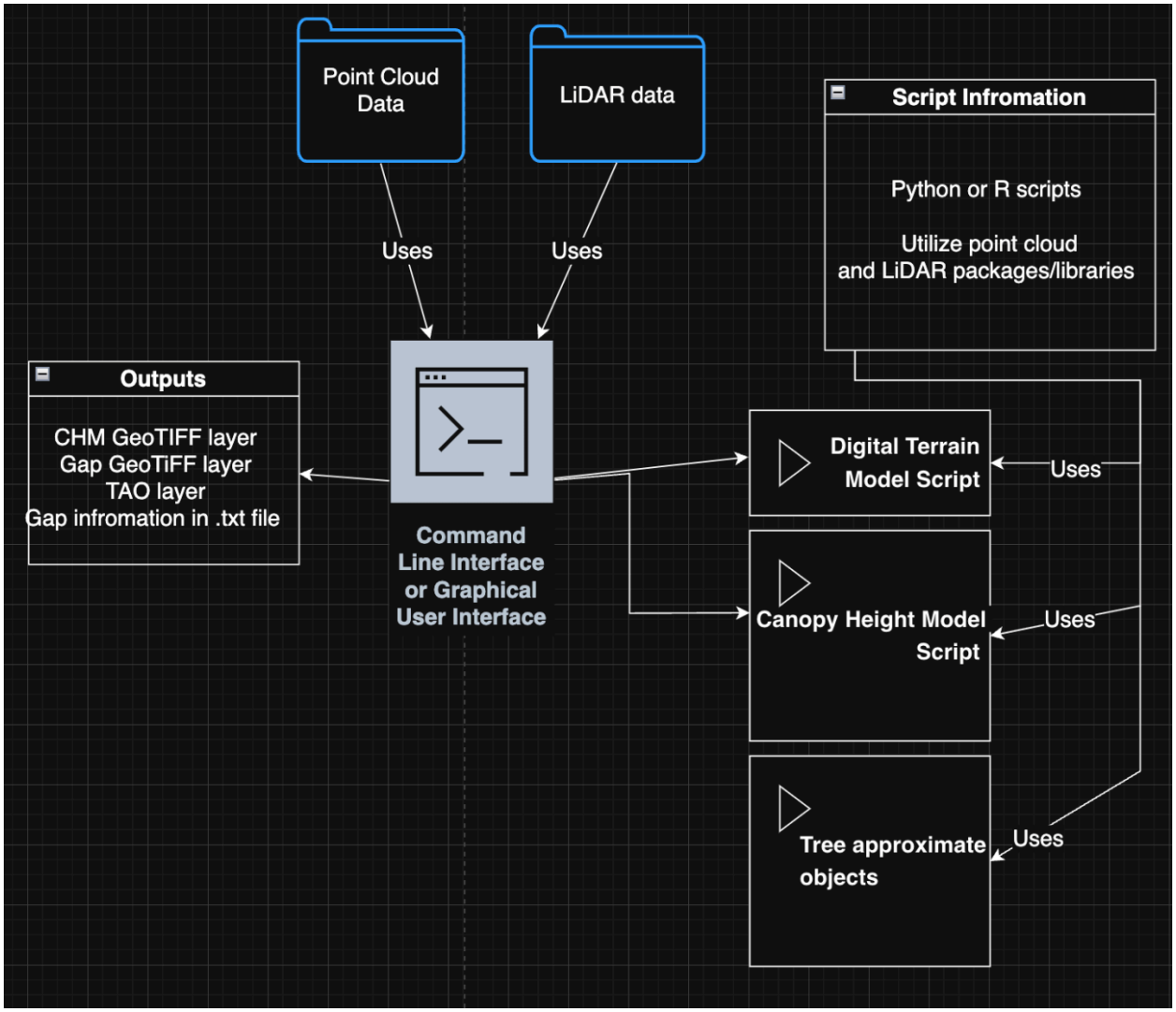

Figure 17. Use Case 3 UML Components Diagram

CLI/GUI

1. The user interface will be in the command line or a simple graphical user interface. The inputs will be the point cloud model, optional LiDAR data, and additional data like ground control points or data to verify the models created

LiDAR Data

1. Holds the LiDAR data used from Washington State LiDAR Portal

Point Cloud Data

1. Holds the point cloud data

Gap Analysis Script

1. Scripts in python using information from the point cloud to automatically identify gaps using the DSM
2. Data will be output in a .txt file alongside the Geo TIFF layer

Canopy Height Model Script

1. Scripts in python to combine all Geo Tiff files into one DTM from the Washington State LiDAR Portal, the DTM from the DSM generated from the point cloud data.
2. CHM is generated from subtracting the DTM from the DSM.
3. Data will be output in a Geo TIFF layer.

Tree Approximate Objects Script

1. Script in R to find the tree approximate objects within the forest
2. Data will be output in a GeoTiFF layer.

Script Infromation:

1. Python scripts will be used.
2. Existing packages/libraries are utilized, specifically designed for point cloud data or LiDAR data.

Output

1. The output will be GeoTIFF layers of CHM and TAO/gaps within the forest.

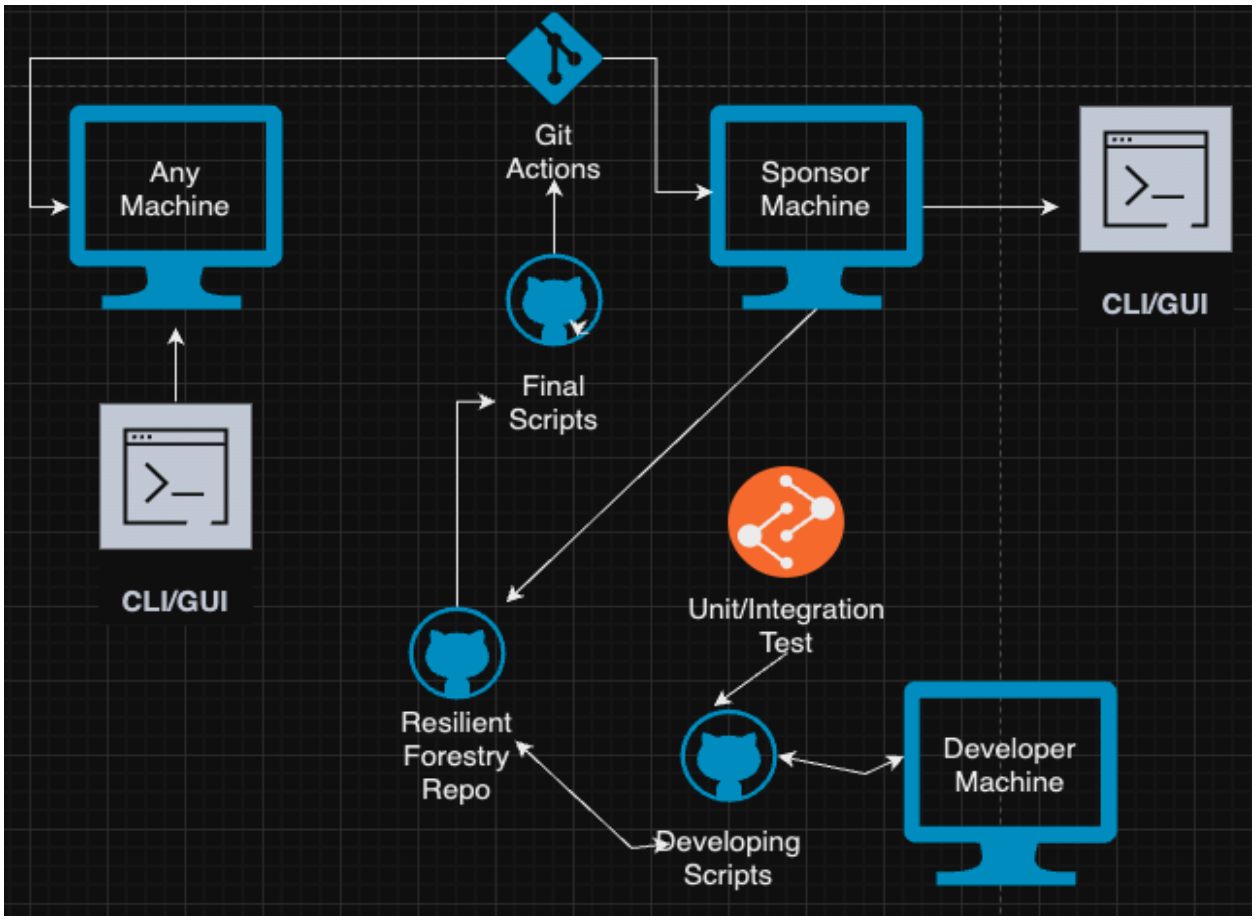

Figure 18. Use Case 3 Dev-Ops Diagram

Resilient Forestry Repo

1. **Developing Scripts**
    a. Where the developing scripts will be placed. The sponsor may have access to these scripts though the Repo as well.
    b. Unit/Intergration tests will be fun on the Dev build to ensure functionality.
2. **Final Scripts**
    a. This will be the working build of our system. Overtime as the internal dev build is adding additional features, the features will then be applied to the sponsor build as long as the tests pass.
3. **Git actions**
    a. Used for continuous integration and delivery of our system. This will allow automation of build, test, and deployment of updates and versions before being sent into production for our sponsors and other customers of our sponsor.

b. All scripts will run through GIT actions if any changes have been made to ensure the script still runs and no accidental mistakes have been made.
c. Git Actions may or may not be needed due to the architecture of our project.

Machines

1. **Developer Machine**
    a. Aim to use an integrated development environment such as Visual Studio Code, languages including Python/R, and additional libraries/packages such as GeoPandas and PYPL LiDAR.
2. **Sponsor Machine**
    a. Our final sponsor build will be able to run on the sponsor's machine through the command line interface or graphical user interface.
3. **Any Machine**
    a. As Resilient Forestry is a consulting company, other machines and their clients should also be able run our system to view the point cloud and the analysis of the data.

CLI/GUI

1. The system will run on the command line interface or a simple graphical user interface.

5.4 Physical Infrastructure

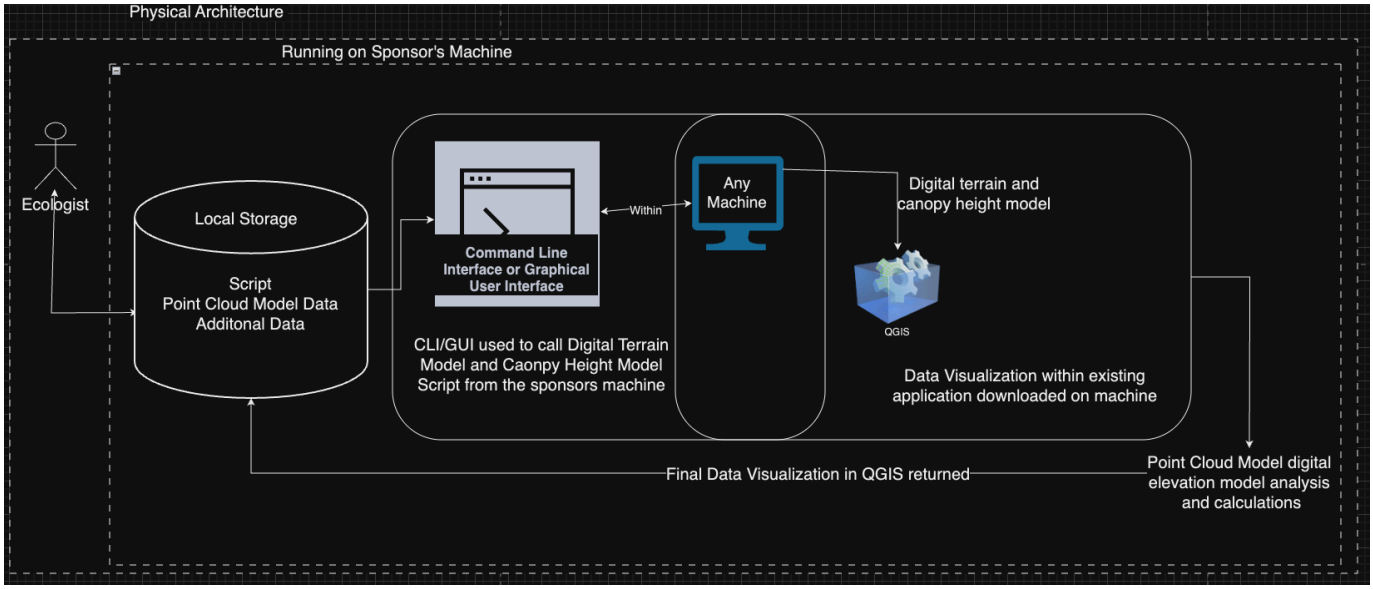

Figure 19. Use Case 3 Physical Diagram

Post-Processing will have two parts, the physical architecture and the computer architecture.

Ecologist

1. The ecologist will pass in point cloud model data and any additional data needed for verification of digital terrain or the canopy height model.

Running on Sponsor's Machine

1. Local Storage
    a. The point cloud model data, scripts to find the digital terrain and canopy height model, and data to verify the digital terrain and canopy height model.
    b. Local storage allows the input to access the local storage, gather required information, and return the output within the local storage.
2. Command Line Interface/Graphical User Interface
    a. The scripts will be directly applied on the sponsor's machine on the command line interface or a simple graphical user interface.
    b. The CLI/GUI will run the digital terrain model and the canopy height model script to generate an analysis of the forest. Which will be passed on to QGIS, confirming analysis has been completed through the CLI/GUI.

3. Data Visualization
    a. The incoming will be in QGIS-acceptable formats and be displayed using QGIS (geographical information system) an open-source application that can be downloaded on any computer for data visualization. Allowing the analysis to be shared across Resilient Forestry and its clients.

6. Version Control:

Date	Version	Author	Change
10/25/24	0.0.1	AJ, Jeremy, Ben	Initial Draft
11/1/24	0.0.2	AJ, Jeremy, Ben	Logical View and revision
11/6/24	0.0.3	AJ, Jeremy, Ben	Process View and revision
11/13/24	0.0.4	AJ, Jeremy, Ben	Development View and revision
11/20/24	0.0.5	AJ, Jeremy, Ben	Physical and revision
11/25/24	0.0.6	AJ, Jeremy, Ben	Final draft 1
4/1/25	0.1.1	AJ, Jeremy, Ben	Updated after Winter Quarter