­­­Image description LLM

Artificial intelligence-based agricultural image description system

|  |  |
| --- | --- |
| AuthorS: | Kacper Sikorski, Konrad Wnuk, Krzysztof Witek, Mateusz Falfus, Justyn Rojkowski, Mikołaj Mieszczok |
| Created: | 30 October 2024 |
| Modified: | 07 January 2025 |
| Recipients: | dr hab. inż., prof. PŚ Jakub Nalepa |
| VerSION: | [5.2] |
| CLASSIFICATION: | Internal |

Content

Image description LLM 1

1. Document history 4

2. Introduction 5

2.1. Background 5

2.2. System Overview 5

3. Requirements 6

3.1. Functional Requirements 6

3.2. Non-Functional Requirements 7

4. Use case diagram 9

4.1. Actors 9

4.2. Use Cases 10

4.3. Relationships 11

4.3.1. Associations 11

4.3.2. Includes 11

4.3.3. Extends 12

5. Use Case Scenarios 13

5.1. Scenario: Generate Descriptions 13

5.2. Scenario: Review Outputs 13

6. Class hierarchy 15

6.1. Model Layer 15

6.2. View Layer 17

6.3. Manager Layer 17

6.4. Class diagram 20

7. Behaviour Modelling 21

7.1. Sequence Diagrams 21

7.2. Communication Diagrams 22

8. Testing the program's operation and behaviour 24

8.1. Image Input and Processing Tests 24

8.1.1. Test Image Upload 24

8.1.2. Test Hyperspectral Image Handling 24

8.1.3. Test Corrupted Files 25

8.2. Batch Processing Tests 25

8.2.1. Test Batch Division 25

8.3. Description Generation Tests 26

8.3.1. Test LLM Output 26

8.3.2. Test Accessibility Descriptions 26

8.4. Export Functionality Tests 26

8.4.1. Test Export Formats 26

8.4.2. Test Metadata Inclusion 27

8.5. GUI Interaction Tests 27

8.5.1. Test Navigation Flow 27

8.5.2. Test Responsiveness 28

8.6. Class-Specific Unit Tests 28

8.6.1. Image Class: 28

8.6.2. Batch Class: 28

8.6.3. DescriptionGenerator Class: 28

8.6.4. Exporters: 29

8.7. Error Handling Tests 29

8.7.1. Test Error Propagation 29

9. Glossary 31

1. Document history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| version | Date | Author | Approved by | Additional Information |
| 0.1 | 2024/10/20 | Kacper Sikorski | Project Team | Initial version of the document |
| 1.0 | 2024/10/25 | Kacper Sikorski | Konrad Wnuk | Text specification |
| 1.1 | 2024/10/27 | Konrad Wnuk | Krzysztof Witek | Functional system requirements |
| 1.3 | 2024/10/30 | Krzysztof Witek | Konrad Wnuk | Glossary of project terms |
| 2.0 | 2024/11/07 | Krzysztof Witek | Konrad Wnuk | Non-functional system requirements |
| 2.1 | 2024/11/10 | Konrad Wnuk | Krzysztof Witek | Defined use case scenarios |
| 2.2 | 2024/11/17 | Mateusz Falfus | Kacper Sikorski | Expanded glossary |
| 3.0 | 2024/11/22 | Konrad Wnuk | Kacper Sikorski | Preparation of class hierarchy |
| 3.1 | 2024/11/23 | Krzysztof Witek | Konrad Wnuk | Preparation of class diagram |
| 3.2 | 2024/11/24 | Mikołaj Mieszczok | Mateusz Falfus | Prototype development |
| 3.3 | 2024/11/25 | Justyn Rojkowski | Mateusz Falfus | Model testing |
| 3.4 | 2024/11/26 | Konrad Wnuk | Kacper Sikorski | Expanded glossary |
| 4.0 | 2024/11/30 | Krzysztof Witek | Konrad Wnuk | Interaction diagrams |
| 4.1 | 2024/12/05 | Konrad Wnuk | Krzysztof Witek | Extend description of the class hierarchy |
| 5.0 | 2024/12/14 | Konrad Wnuk | Kacper Sikorski | Development of tests used in the projection |
| 5.1 | 2024/12/15 | Konrad Wnuk | Kacper Sikorski | Expanded glossary in testing field |
| 5.2 | 2025/01/07 | Konrad Wnuk |  | Correct and design new test descriptions according to recommendations |

1. Introduction

This project document outlines the development of an AI-based **system** that generates descriptive captions for **agricultural images**, leveraging the capabilities of **Large Language Models (LLMs)**. The project aims to support various users, including **farmers**, **agricultural analysts**, and **visually impaired individuals**, by transforming visual data into **accessible**, informative content. By utilizing advanced AI technologies, this system bridges the gap between the complex analysis of agricultural imagery and practical, user-friendly output.

* 1. Background

Advances in artificial intelligence, particularly with **LLMs**, have significantly impacted various industries by enabling automated interpretation and text generation. These models have been instrumental in transforming how we interact with data across sectors like healthcare, environmental science, and automated support services. In agriculture, AI systems capable of processing and describing images offer critical support for tasks such as monitoring crop health, identifying plant species, or detecting pests. Such tools benefit users who may not have technical expertise and create more inclusive access for individuals with visual impairments.

* 1. System Overview

The system will be built using **Python** as the main programming language and will employ **TensorFlow** for integrating and **fine-tuning pre-trained LLMs** that can generate descriptions from images. System will generate **images descriptions** related to agricultural topics. Within the system, the language model will process images in **RGB** format to generate a general description of the images. These images may depict elements such as agricultural fields, plants or plant problems. The system will be a tool to support visual content analysis, useful for **crop management**. **GUI** built in **Tkinter** for **Python**, which will allow the user to point to a folder containing images for analysis. The process begins with the extraction and analysis of the **RGB channels** of the images – the images will be divided into packets, which will allow efficient management of memory and processing time, avoiding the simultaneous analysis of too many images. Each image packet is processed sequentially, and the generated descriptions will be written in **batches** to the **output file**. **Farmers** can use the system as **assistive tool** to monitor crop health and detect potential issues through generated image descriptions, facilitating quicker and more informed decisions. **Advanced analysts** benefit from the system by conducting more detailed interpretations of many agricultural images. The system's performance can be further enhanced through **model training**, a process that involves teaching the AI model to improve its ability to generate accurate descriptions by exposing it to various datasets.

1. Requirements

This section specifies the functional and non-functional requirements for the project, including clear identifiers and acceptance criteria. The functional requirements (FR) detail what the system must accomplish, while the non-functional requirements (NFR) describe the attributes and constraints for optimal performance and user satisfaction.

* 1. Functional Requirements

The functional requirements specify the key functionalities of the system. Each requirement is identified for easy reference.

**FR.1 – Image Input and Processing** – The system must allow **user** to upload and process input images, including hyperspectral image sets. **User** should be able to select specific **RGB channels** from these hyperspectral bands or have the system automatically detect the appropriate channels.

*Acceptance Criteria:*

* The system provides an option in the GUI to manually select **RGB channels** from an input image.
* The system has an automated detection feature for identifying **RGB channels** in hyperspectral images.
* The system can process **low-resolution images** up to a resolution of 1024x1024 pixels without errors.

**FR.2 – Batch Image Processing** – The system must process images in batches to manage memory and optimize processing time.

*Acceptance Criteria:*

* The system outputs progress logs for **batch processing**.
* The system must handle large batches without encountering errors, except in cases of extremely large, impractical volumes. It should continue processing for longer durations if necessary but remain stable, operational and provide a final output after completion.

**FR.3 – Description Generation** – The system must use a pre-trained LLM to generate concise descriptions of the analysed images, focusing on key visual elements.

*Acceptance Criteria:*

* The generated descriptions cover primary image **nature elements** such as crops, plants, and pest indicators.
* The system provides the user with simple and quick access to **visual analysis** of images by a **factual description**.
* The system achieves an accuracy significantly better than a simple **Markov chain-based text generation model** when compared to benchmark human-generated descriptions.
* The generated descriptions adhere to the principle of **description simplicity**, ensuring that they are clear, easy to understand, and not overly detailed.
  1. Non-Functional Requirements

The non-functional requirements define the qualities the system must possess to be effective. Each requirement is assigned an identifier and includes strict acceptance criteria.

**NFR.1 – Performance** – The system should be able to process images quickly, ensuring efficient use of resources.

*Acceptance Criteria:*

* The system can process a batch of 100 **images** within 10 minutes using a modern mid-range GPU from the NVIDIA GeForce RTX 20 series.
* The application's startup and initial loading time, before it is ready for use, should not exceed 2 minutes.

**NFR.2 – User Interaction via GUI** –The system must provide **GUI** for user interaction, allowing the upload and selection of image folders.

*Acceptance Criteria:*

* Users can navigate the GUI and select image folders within three clicks.
* The GUI is responsive and functional on displays with a resolution of 1920x1080 pixels.
* Every key feature of the program can be performed from the GUI level.
* The GUI adheres to **accessibility** best practices, providing clear labels and descriptive tooltips for key functions to enhance usability for all users, including those with disabilities.

**NFR.3 – Screen Reader Compatibility** –The system must generate output text that is compatible with screen readers to aid **visually impaired users**.

*Acceptance Criteria:*

* **Visually impaired users** can navigate the system's output.
* The text formatting, including new lines and overall structure, must be optimized for **screen reader** usage to ensure smooth navigation and comprehension.

**NFR.4 – Compatibility** – The system should be compatible with standard operating systems such as Windows and macOS.

*Acceptance Criteria:*

* The system runs without errors on Windows 10/11 and Linux Ubuntu 20.04 LTS or newer.
* Every key function operates as expected on both platforms.

**NFR.5 – Modularity for LLM Integration** – The system should allow for easy swapping of the LLM module with another model.

*Acceptance Criteria:*

* The LLM module can be replaced with a different model without impacting on the overall system architecture.
* System administrators with professional technical knowledge should be able to perform the replacement by following the provided documentation.

**NFR.6 – Security** – The system should be capable of operating fully offline to enhance data privacy and prevent unauthorized external access.

*Acceptance Criteria:*

* The system can perform all **image processing**, analysis, and **description generation** without an active internet connection.

1. Use case diagram

This section provides an overview of the system's functionality through a UML use case diagram, highlighting key interactions between actors and the system.

A diagram of a diagram

Description automatically generated

Figure 1 illustrates the UML use case diagram. Selected elements of the diagram are elaborated upon in detail in subsequent sections of this document.

* 1. Actors

Actors are external entities (users or systems) that interact with the system to achieve specific goals.

**ACT.1 – LLM (Large Language Model)**

* AI system responsible for analysing images and generating descriptions. The model is selected and configured by the administrator to adapt to specific user requirements.

**ACT.2 – Farmer**

* A primary user of the system with basic technical knowledge. This actor relies on the system to upload agricultural images and obtain concise descriptions to monitor crop health and detect potential issues such as pest infestations or soil deficiencies.

**ACT.3 – Advanced Analyst**

* A technical expert who uses advanced system features, such as generating detailed descriptions using additional analytical parameters. He can verify results and propose changes to the system configuration.

**ACT.4 – Visually Impaired User**

* A user who accesses the system’s generated descriptions using screen readers. The system is designed to provide accessible text outputs, enabling visually impaired users to gain insights into agricultural imagery.

**ACT.5 – System Administrator**

* An advanced user responsible for system maintenance, configuration, and upgrades. The administrator ensures the system is functioning correctly, manages updates to the language model, and oversees compatibility across different environments.
  1. Use Cases

A use case scenario describes a specific interaction between actors and the system to achieve a particular goal. It focuses on the step-by-step processes involved in fulfilling a user’s objective.

**UC.1 – Upload Images**

* The process of selecting and uploading image files or folders into the system through the GUI. Users can choose individual images or entire directories for processing.

**UC.2 – Process Images in Batches**

* A function to divide the input images into smaller, manageable batches for efficient processing. This step is essential to optimize memory usage and ensure stability during large-scale analyses.

**UC.3 – Generate Descriptions**

* The system’s core functionality, leveraging the LLM to analyse input images and produce concise textual descriptions. These descriptions focus on key elements such as crop conditions, pest indicators, and soil characteristics.

**UC.4 – Generate Advanced Descriptions**

* A function that generates detailed descriptions that include additional analyses, such as identifying more complex crop problems (e.g., analysing trends in crop health) or forecasting changes in crop conditions.

**UC.5 – Generate Accessible Descriptions**

* The process of generating descriptions in simplified language, optimized for users using assistive technologies such as screen readers. These descriptions are more concise and adapted to the needs of the blind or visually impaired.

**UC.6 – Select Language Model**

* The act of configuring the system by the administrator to select the appropriate language model. The administrator can change the model depending on analytical or performance requirements.

**UC.7 – Review Outputs**

* Allows users to view the generated descriptions within the GUI or an output file. This use case supports inspecting individual results or summaries of image analyses.

**UC.8 – Export Results**

* An optional feature enabling users to export the generated descriptions into a preferred format, such as CSV or PDF, for further analysis or record-keeping.
  1. Relationships

Relationships define the connections between use cases and actors or between different use cases. They specify how components of the system interact or depend on each other.

* + 1. Associations

Associations represent direct interactions between actors and use cases. They show which actors perform specific actions within the system.

**ASS.1 – User interacts with Upload Images.**

* This relationship shows that the user initiates the process by uploading images, which serves as the entry point for the description generation workflow.

**ASS.2 – Advanced Analyst interacts with Generate Advanced Descriptions.**

* This relationship shows that advanced analysts utilize the system to generate in-depth and detailed descriptions for specialized use cases.

**ASS.3 – Visually Impaired User interacts with Generate Accessible Descriptions.**

* This relationship emphasizes that visually impaired users rely on the accessible descriptions functionality to understand image content through simplified text outputs.

**ASS.4 – LLM interacts with Generate Descriptions.**

* This relationship reflects that the large language model is the primary component used to analyse images and generate textual descriptions.

**ASS.5 – Administrator interacts with Select Language Model.**

* This relationship highlights that the administrator is responsible for selecting and configuring the language model, ensuring that it is appropriate for the system’s requirements.
  + 1. Includes

The «include» relationship specifies that one use case is a mandatory sub-process of another. The primary use case cannot be completed without the included use case.

**INC.1 – Process Images in Batches is part of Generate Descriptions.**

* This relationship highlights that batch processing is a required sub-process for generating descriptions. The system must first divide the uploaded images into manageable batches to ensure efficient and stable processing.

**INC.2 – Review Outputs is part of Generate Descriptions.**

* This relationship shows that reviewing the outputs is a necessary step after generating descriptions. Users must inspect and validate the descriptions before proceeding to export or use the results.

**INC.3 – Select Language Model is part of Generate Descriptions.**

* This relationship emphasizes that selecting a language model is a prerequisite for generating descriptions. The system depends on an appropriate language model to analyse images and produce meaningful textual outputs.
  + 1. Extends

The «extend» relationship adds optional functionality to a use case. It allows a use case to be extended with additional features under specific conditions.

**EXT.1 – Generate Advanced Descriptions extends Generate Descriptions.**

* This relationship denotes that advanced descriptions are an optional extension of the core description generation functionality. Users with advanced needs can request additional analytical insights if required.

**EXT.2 – Generate Accessible Descriptions extends Generate Descriptions.**

* This relationship signifies that accessible descriptions are an optional extension of the standard description generation process. This functionality ensures the output is suitable for visually impaired users and compatible with assistive technologies.

**EXT.3 – Export Results extends Review Outputs.**

* This relationship indicates that exporting results is an optional action following the review of outputs. Users may choose to save the reviewed descriptions in formats such as CSV or PDF only if they find the results satisfactory.

1. Use Case Scenarios

This section details specific use case scenarios, describing step-by-step interactions between actors and the system to achieve defined goals.

* 1. Scenario: Generate Descriptions

**Main Path:**

1. The user opens the system and navigates to the GUI's upload section.
2. The user selects a folder containing one or more images for processing. The GUI provides a confirmation of the selected directory, and a summary of the files detected.
3. The system initializes batch processing, organizing the images into manageable groups based on preconfigured memory and performance parameters. A progress bar is displayed to keep the user informed about the ongoing process.
4. For each batch, the system processes the images by analysing their RGB channels and passing the data to the pre-trained Large Language Model (LLM). The LLM generates concise textual descriptions that summarize key visual elements, such as crop health, pest indicators, or soil conditions.
5. After processing all batches, the system consolidates the results and saves the generated descriptions in an output file located in a user-specified directory.
6. Upon completion, the system provides a notification to the user, indicating that the descriptions are ready. The notification includes the total number of successfully processed images and any skipped files.

**Alternative Path:**

* If an image file is corrupted, unreadable, or in an unsupported format, the system logs the error and skips the problematic file. The log is appended to an error report, which can be reviewed by the user for troubleshooting or file correction. The system continues processing the remaining files without interruption.
  1. Scenario: Review Outputs

**Main Path:**

1. The user navigates to the "Review Outputs" section of the GUI.
2. The user selects the output file containing the generated descriptions. The GUI loads the file and displays its contents in a structured, user-friendly format. Each description is accompanied by metadata, such as the file name, timestamp of processing, and any relevant tags (e.g., "crop health issue detected").
3. The user reviews the descriptions, using sorting or filtering options (e.g., filter by keyword, sort by processing order) to focus on specific outputs of interest.
4. If desired, the user exports the reviewed results to a preferred file format, such as CSV or PDF, for further analysis or reporting. The exported file retains all metadata and includes a summary of the processing session.

**Alternative Path:**

* If the output file is missing or corrupted, the system alerts the user and provides options to either regenerate the file from a backup (if available) or reprocess the original images.

1. Class hierarchy

The class hierarchy outlines the organization of the system's components, dividing them into logical layers to ensure clarity, modularity, and scalability. Each class is designed with specific responsibilities to maintain separation of concerns and promote maintainable software architecture.

* 1. Model Layer

The Model Layer is responsible for the core data and logic of the system. It defines the entities and their attributes, focusing on the fundamental operations and data structures.

**MOD.1 – Image**

Description: Represents an individual image file, holding data related to the image.

Responsibilities:

Holds image data, file path, and processing state.

Attributes:

* file\_path: String — Path to the image file.
* is\_processed: Boolean — Status indicating whether the image has been processed.

Relationships:

* Aggregates Batch (1..\*:1) — An image is part of a batch.

**MOD.2 – Batch**

Description: Groups multiple Image objects into manageable sets for efficient processing.

Responsibilities:

Manages a group of images and their processing status.

Attributes:

* status: *Status* — Processing status (e.g., "Processing", "Completed").
* *images[\*]: Image* — List of images in the batch.

Methods:

* add\_image(Image) — Adds an image to the batch.

Relationships:

* Composition to status (0..\*:1) — The batch has a specific status.
* Aggregates Image (1..\*:1) — Contains one or more images.

**MOD.3 – Description**

Description: Represents a textual description generated for an image or a batch of images.

Responsibilities:

Stores the content and metadata of a description.

Attributes:

* *content: String* — Text content of the description.
* *metadata[\*]: String* — Metadata associated with the description.

Relationships:

* Used by DescriptionGenerator.

**MOD.4 – LLModel**

Description: Represents a machine learning model for language generation.

Responsibilities:

Stores information about the language model and interacts with the configuration.

Attributes:

* language: language — The language used in the model.
* model\_path: String — Path to the model file.

Relationships:

* Aggregates language (0..\*) — Multiple languages may be supported.
* Composition to configuration (0..\*:1) — Tied to configurations.

**MOD.5 – Settings**

Description: Stores configuration settings for the system.

Responsibilities:

Manages available models and sets the current model.

Attributes:

* available\_models[\*]: LLModel — A list of available language models.

Methods:

* add\_new\_model(): void — Adds a new model to the settings.
* set\_current\_model(): void — Sets the current model.

Relationships:

* Aggregates LLModel (1) — Holds references to available models.
  1. View Layer

The View Layer handles the graphical user interface (GUI) and interaction with the user. It displays data and collects inputs for further processing.

**VIEW.1 – GUI**

Description: Manages the graphical user interface of the system.

Responsibilities:

Displays information to the user and interacts with the model.

Relationships:

* Composition to *ProgressBar* (1:0..\*) — Displays a progress bar.
* Composition to *ExportManager* (1:1) — Handles export functionality.
* Composition to *ImportManager* (1:1) — Handles import functionality.
* Related to *DescriptionGenerator* (1..\*:1) — Uses the description generator to display information.

**VIEW.2 – ProgressBar**

Description: Represents a progress bar in the user interface.

Responsibilities:

Displays the progress of long-running tasks like image processing or exporting.

Attributes:

* percentage\_progress: int — The current percentage of progress (0 to 100).
  1. Manager Layer

The Manager Layer orchestrates interactions between the Model and View. It acts as a mediator, ensuring proper communication and functionality.

**MAN.1 – DescriptionGenerator**

Description: Responsible for generating descriptions for images. Bridge design pattern.

Responsibilities:

Defines the method to generate descriptions, which is implemented by specific generators.

Methods:

* generate\_description(Image): Description — Generates a description for an image.

Relationships:

* Uses Image (1..\*) — Generates descriptions for images.
* Calls LLModel — Interacts with the language model to generate descriptions.
* Related to GUI (1..\*:1) — Used by the GUI to display descriptions.

**MAN.2 – VerboseDescriptionGenerator**

Description: A specialized version of DescriptionGenerator that generates more detailed descriptions.

Responsibilities:

Adds verbosity to the generated descriptions.

Relationships:

* Inherits from DescriptionGenerator.

**MAN.3 – ExportMenager**

Description: Manages the export functionality for the system, including file paths and exporting data.

Responsibilities:

Handles the export process and interacts with exporters.

Attributes:

* output\_path: String — Path where the exported data will be saved.

Relationships:

* Depends on Exporter (1:0..\*) — Uses various exporters (PDFExporter, CSVExporter, TXTExporter).

**MAN.4 – Exporter (Interface)**

Description: Defines the contract for all export formats. Strategy design pattern.

Responsibilities:

Declares a method for exporting data.

Methods:

* export(Image): void — Exports an image in the required format.

Relationships:

* Implemented by PDFExporter, CSVExporter, and TXTExporter.

**MAN.5 – PDFExporter**

Description: Handles the export of images to PDF format.

Responsibilities:

Implements the export() method for PDF export.

Relationships:

* Implements Exporter.

**MAN.6 – CSVExporter**

Description: Handles the export of images to CSV format.

Responsibilities:

Implements the export() method for CSV export.

Relationships:

* Implements Exporter.

**MAN.7 – TXTExporter**

Description: Handles the export of images to plain text format.

Responsibilities:

Implements the export() method for TXT export.

Relationships:

* Implements Exporter.

**MAN.8 – ImportManager**

Description: Manages the import functionality for images.

Responsibilities:

Imports image files and processes them.

Relationships:

* Creates Image (1:0..\*) — Responsible for creating image objects.
* Uses ImageProcessor (1:0..\*) — Uses processors to handle image transformations.

**MAN.9 – ImageProcessor (Interface)**

Description: Defines a contract for image processing operations.

Responsibilities:

Declares a method for processing images.

Methods:

* processImage(Integer\*, Integer): void — Processes an image with specified parameters.

Relationships:

* Refined by RGBImageProcessor (1..\*:1).

**MAN.10 – RGBImageProcessor**

Description: A specific implementation of ImageProcessor for RGB image processing.

Responsibilities:

Implements the processImage() method specifically for RGB image processing.

Relationships:

* Implements ImageProcessor.

**MAN.11 – User**

Description: Represents a general user in the system.

Responsibilities:

Holds common user data and functionality.

**MAN.12 – Administrator**

Description: Represents an administrator of the system

Responsibilities:

Manages system configurations, including user management.

Relationships:

* Inserts from User.
* Aggregates configuration (1..\*) — Has access to system configuration.
  1. Class diagram

A diagram of a computer

Description automatically generated

Figure 2 Class diagram representing the project's structure

1. Behaviour Modelling

Ineteraction diagrams focus on understanding how different components or objects within   
a system collaborates to achieve specific functionalities. They allow developers to visualize the flow of data, messages, and control across the system, offering insights into both structure and behaviour.

* 1. Sequence Diagrams

The sequence diagrams are a crucial tool for visualizing the chronological sequence of interactions between objects or components in a system. It emphasizes the timing and order of messages exchanged, making it particularly effective for representing scenarios like use cases or system processes. By illustrating how objects interact over time, sequence diagram helps in identifying potential bottlenecks, dependencies, and opportunities for optimization.

A diagram of a project

Description automatically generated

Figure 3 Sequence diagram depicting interactions during parallel images import

A diagram of a project

Description automatically generated

Figure 4 Sequence diagram on parallel descriptions generation and export

* 1. Communication Diagrams

While the sequence diagram focuses on the temporal aspect of interactions, the communication diagram highlights the structural relationships between objects or components and the messages they exchange. This diagram provides a more network-like view of the interactions, emphasizing the connections and communication patterns rather than the timing of events. The communication diagram presents the same information as the sequence diagram, but in a different form.

A diagram of a process

Description automatically generated

Figure 5 Communication diagram depicting interactions during parallel images import

A diagram of a software flowchart

Description automatically generated

Figure 6 Communication diagram on parallel descriptions generation and export

1. Testing the program's operation and behaviour

The following unit tests aim to ensure that every aspect of the system is thoroughly tested, adhering to the functional and non-functional requirements described in the document. These tests are built upon the scenarios, flow diagrams, and class hierarchy to maintain a systematic and precise testing methodology.

These unit tests cover every critical aspect of the system, from individual class functionality to integrated workflows. Testing will ensure the system meets functional and non-functional requirements, offering a robust and user-friendly solution for agricultural image description.

* 1. Image Input and Processing Tests
     1. Test Image Upload

**Description:**

This test verifies that the system can correctly accept and load image files from a specified directory. The test includes handling both valid and invalid file paths.

**Input data:**

* Valid folder path containing hyperspectral images.
* Example invalid path.

**Steps:**

* Provide a valid folder containing multiple image files.
* Ensure the system correctly identifies and lists all images.
* Provide an invalid or inaccessible path and check for error handling.

**Expected Outcome:**

Valid images are loaded, while invalid paths return appropriate error messages without halting the system.

* + 1. Test Hyperspectral Image Handling

**Description:**

This test ensures that the system can process hyperspectral images, either manually or automatically detecting RGB channels.

**Input data:**

* Hyperspectral image file.
* Example RGB channel selection: Red (Band 20), Green (Band 34), Blue (Band 45).

**Steps:**

* Load a hyperspectral image.
* Use the GUI to manually select RGB channels.
* Test the automated RGB detection functionality.

**Expected Outcome:**

The system successfully processes hyperspectral images using both manual and automated methods, displaying selected channels.

* + 1. Test Corrupted Files

**Description:**

This test checks the system’s ability to skip corrupted or unsupported files, logging errors for each occurrence.

**Input data:**

* Valid image files.
* Corrupted files or unsupported format like.

**Steps:**

* Include corrupted and unsupported image files in the batch.
* Monitor processing to ensure these files are skipped without stopping the system.

**Expected Outcome:**

The system logs errors for problematic files and continues processing the remaining images without interruptions.

* 1. Batch Processing Tests
     1. Test Batch Division

**Description:**

This test validates that images are grouped into batches based on memory constraints defined in the settings.

**Input Data:**

* Folder containing 500 image files.
* Batch size setting: 100 images per batch.

**Steps:**

* Upload a large number of images.
* Configure the batch size to a specific number.
* Verify that the system divides the images into the correct number of batches.

**Expected Outcome:**

Images are grouped as per the specified batch size without exceeding memory limits.

* 1. Description Generation Tests
     1. Test LLM Output

**Description:**

This test evaluates the descriptions generated by the LLM, ensuring they meet the standards of clarity, accuracy, and simplicity.

**Input Data:**

* Sample images showing crop health issues: crop\_health1, crop\_health2.
* Predefined benchmarks for description accuracy, such as:
  + "Image shows signs of pest infestation in central region."
  + "Healthy crop with no visible damage."

**Steps:**

* Provide sample images with known attributes.
* Compare the generated descriptions against pre-defined benchmarks.

**Expected Outcome:**

The descriptions accurately reflect the key elements of the images (e.g., crop health, pest indicators) and are concise.

* + 1. Test Accessibility Descriptions

**Description:**

This test ensures that descriptions generated for visually impaired users adhere to accessibility standards, such as simplicity and compatibility with screen readers.

**Input Data:**

* Sample images: image\_accessibility\_test1, image\_accessibility\_test2.
* Screen reader software to validate accessibility descriptions.

**Steps:**

* Process images with the accessibility mode enabled.
* Test output using a screen reader to confirm it reads smoothly and logically.

**Expected Outcome:**

Accessible descriptions are clear, concise, and fully compatible with screen readers.

* 1. Export Functionality Tests
     1. Test Export Formats

**Description:**

This test confirms the system's ability to export results in all supported formats (CSV, PDF, TXT).

**Input Data:**

* Batch of processed descriptions with metadata:
  + Description example: "Crop shows signs of overwatering."
  + Metadata: Image name, processing time.

**Steps:**

* Generate descriptions for a batch of images.
* Export the results in each format.
* Open the exported files to verify integrity and structure.

**Expected Outcome:**

Files are exported correctly in the desired formats, with no loss of data.

* + 1. Test Metadata Inclusion

**Description:**

This test checks that exported files include all relevant metadata (e.g., file name, processing time).

**Input Data:**

* Metadata example - Image, Processing time.

**Steps:**

* Generate descriptions for a batch of images.
* Export the results.
* Inspect the exported files to confirm the presence of metadata.

**Expected Outcome:**

Metadata is included in all exported files, formatted as specified.

* 1. GUI Interaction Tests
     1. Test Navigation Flow

**Description:**

This test simulates user interactions to ensure smooth navigation through the GUI.

**Input Data:**

* Test scenarios: Navigate through GUI sections in sequence:

"Image Upload" → "Batch Processing" → "Description Review".

**Steps:**

* Open the GUI and navigate through each section (e.g., image upload, description review).
* Test transitions between sections.

**Expected Outcome:**

Users can navigate through the GUI effortlessly without encountering any bugs or delays.

* + 1. Test Responsiveness

**Description:**

This test validates GUI responsiveness across different screen resolutions.

**Input Data:**

* Screen resolutions: 1920x1080, 1366x768, 1280x720.
* GUI layouts with various scaling options.

**Steps:**

* Test the GUI on resolutions such as 1920x1080, 1366x768, and 1280x720.
* Check for proper scaling and functionality.

**Expected Outcome:** The GUI scales properly and remains functional across all tested resolutions.

* 1. Class-Specific Unit Tests
     1. Image Class:

Test initialization, attribute modification (e.g., file path, processing status), and relationships (e.g., batch association).

**Input Data:**

* Test attributes: File path, status.
  + 1. Batch Class:

Verify image grouping, status transitions, and operations like adding/removing images.

**Input Data:**

* Images for grouping.
  + 1. DescriptionGenerator Class:

Mock LLM interactions to ensure descriptions align with input data.

**Input Data:**

* Mock LLM input: Crop images with health conditions.
* Expected output: Concise descriptions, e.g., "Crop shows signs of pest infestation."
  + 1. Exporters:

Unit test each exporter (PDF, CSV, TXT) for proper data handling and output generation.

**Input Data:**

* Processed data.
* Formats: PDF, CSV, TXT.
  1. Error Handling Tests
     1. Test Error Propagation
        1. System Tests

**Description:**

This test ensures that the logging system accurately captures errors in various scenarios, providing detailed information for troubleshooting.

**Input Data:**

* Missing file path.
* Unsupported file format.
* Corrupted file.

**Steps:**

1. Simulate system-level errors:

* Attempt to process a missing file path.
* Include an unsupported file format in the upload batch.
* Include a corrupted file in the upload batch.

1. Verify system logs for error details.

**Expected Outcome:**

Logs include the following details for each error:

* Error type (e.g., "File Not Found").
* File name or path where applicable.
* Timestamp of the error occurrence.

System continues functioning without interruption.

* + - 1. Implementation Tests

**Description:**

This test validates the implementation of the logging mechanism within individual modules, ensuring accurate and detailed logging at the code level.

**Input Data:**

* Missing file path: Passed to the file handler module as "".
* Unsupported format: Passed to the file parser.
* Mock error: Deliberately triggered exception in the batch processor.

**Steps:**

1. Unit test the file handler:

* Pass an empty string ("") as the file path.
* Check the generated log entry.

1. Unit test the file parser:

* Attempt to parse an unsupported format.
* Validate that the error log includes specific details about the unsupported file.

1. Mock an exception in the batch processor:

* Trigger an artificial error in the batch processing logic.
* Confirm that the log entry captures the exception message and stack trace.

**Expected Outcome:**

Each module produces log entries with:

* Module name and context (e.g., "File Handler").
* Precise error description (e.g., "Unsupported file format: .xyz").
* Timestamp of occurrence.
* Any additional debug information if available.

1. Glossary

|  |  |
| --- | --- |
| **Term** | Definition |
| **Accessibility** | A design principle that ensures the system can be used by people with various abilities, particularly those who are visually impaired. |
| **Actor** | Any entity interacting with the system, such as users or external systems. |
| **Adaptive Thresholding** | A method in image processing that dynamically adjusts the threshold value for different regions of the image based on local pixel intensity variations. |
| **Advanced Analyst** | A user with expertise in agricultural analysis who uses the system for more detailed and in-depth interpretation of agricultural aerial images. |
| **Agriculture** | The field in which the system will be applied, supporting crop monitoring and the analysis of agricultural images. |
| **Assistive Tool** | Software which is designed to help users analyze visual data more effectively and make quicker decisions. It supports the decision-making process by providing insights and recommendations but does not make the final decision for the user, ensuring that the user remains an integral part of the process and is not rendered obsolete. |
| **Batch Processing** | The method of processing images in groups (or batches) to optimize memory and processing time, preventing the system from being overwhelmed by large volumes of data. |
| **Batch Size** | The number of images processed simultaneously by the system during batch processing. This parameter affects memory usage and system performance. |
| **Conciseness in Description** | A specific system feature ensuring that descriptions are short and focused, meeting user needs without overloading with unnecessary details. |
| **Crop Health** | The condition of agricultural crops, assessed via image analysis. |
| **Crop Management** | The process of monitoring and controlling the health of agricultural crops, which the system can support by providing image descriptions. |
| **Description Simplicity** | A project principle that ensures generated descriptions are clear, easy to understand, and not overly detailed. |
| **Dynamic Batching** | Adjusting batch size dynamically during processing to optimize memory and performance. |
| **Error Log** | A file or system output that records issues encountered during processing, such as corrupted images or unsupported formats. |
| **Error Propagation** | The impact of one error (e.g., a corrupted file) on subsequent processing tasks, and how the system isolates and mitigates these effects. |
| **Explainability** | The ability of an AI model to provide transparent and understandable reasoning behind its outputs. |
| **Factual Description** | A type of description that focuses solely on presenting the facts and visual elements without any additional interpretation. |
| **Farmer** | A user of the system with basic technical knowledge who relies on generated image descriptions to monitor crop health and detect potential issues such as pests or diseases. |
| **Factual Description** | Objective representation of visual elements without speculative interpretation. |
| **Filter Options** | A feature in the system that allows users to sort or filter generated descriptions based on metadata, such as keywords or timestamps |
| **Graphical User Interface (GUI)** | Graphical User Interface that allows users to interact with the system through graphical elements like buttons and input fields. |
| **Hyperspectral Imaging** | A type of imaging that captures data across multiple wavelengths of light, beyond the visible spectrum. It is used in agriculture for detailed analysis, such as nutrient levels or crop stress detection. |
| **Image** | A graphic file in RGB format that may depict agricultural subjects, such as fields, plants, pests, or other nature elements. Aerial images provide a broader view and may include unique features such as crop patterns, soil variation, and large-scale pest infestations. |
| **Image Description** | Text generated by the language model that provides a summary of the primary visual elements in the image, making it easier to understand. |
| **Image Packet** | A subset of images grouped together for batch processing, ensuring efficient resource utilization. |
| **Image Processing** | The analysis and extraction of information from an image in RGB format to prepare data for the language model. |
| **Large Language Model (LLM)** | Large Language Model, an advanced AI system based on machine learning that processes text and generates descriptions of images based on input data. |
| **Low-Resolution Image** | A graphic file with limited resolution and size, allowing the system to process it more quickly without significant strain on system resources. Low-resolution image can be up to a resolution of 1024x1024 pixels. |
| **Markov Chain-Based Text Generation Model** | A basic type of text generation model that uses probability chains to predict the next element in a sequence based on the current state. It is often used as a baseline for evaluating the performance of more advanced AI models, such as LLMs, which should achieve significantly better accuracy compared to Markov chain models when generating descriptions. |
| **Memory Optimization** | Techniques used to manage the system's memory consumption during image analysis, particularly when processing large datasets. |
| **Metadata** | Supplemental information attached to image descriptions, such as timestamps, image resolution, processing time, and tags. |
| **Model Training** | The process of teaching the AI model to improve its ability to generate accurate descriptions through exposure to various datasets. |
| **Nature Elements** | Basic components found in agricultural images, such as plants, soil, crops, and pests. |
| **Output File** | A file generated by the system that contains the descriptions of the analyzed images, which users can review and use. |
| **Pest Detection** | Identifying signs of pests in agricultural images to assist in crop management. |
| **Pre-trained Model** | An AI model trained on a large dataset before integration into the system, enabling efficient and accurate task-specific performance. |
| **Processing Log** | A detailed record of the system's operations during batch processing, including successful tasks, errors, and skipped files. |
| **Python** | A high-level, interpreted programming language known for its readability and versatility. Used as the primary programming language for building the system due to its extensive libraries and support for machine learning and AI development. |
| **RGB Channel** | A component of a digital image that contains color information for red (R), green (G), and blue (B). These channels are processed by the system to generate descriptions. |
| **Scalability** | The system's capability to adapt to increased workload, such as processing more images or integrating advanced analytical features, without degradation in performance. |
| **Scenario** | A detailed narrative outlining how a user interacts with the system to complete a specific use case. |
| **Screen Reader** | Software that enables visually impaired or blind individuals to use computers and applications by reading the text displayed on the screen aloud. |
| **System** | An application that integrates the language model, user interface, and image processing mechanisms, enabling the automatic generation of image descriptions. |
| **System Administrator** | An advanced user responsible for maintaining and managing the system, ensuring its optimal operation, updating language models. |
| **TensorFlow** | An open-source machine learning framework developed by Google, used to build, train, and deploy AI models, including those used for image analysis and language processing. |
| **Tkinter** | A standard GUI (Graphical User Interface) library for Python, used to create simple and intuitive user interfaces. In this project, Tkinter is employed to build the interface that allows users to upload and manage images for analysis. |
| **Use Case** | A description of how a user interacts with the system to achieve a specific goal, such as uploading images or reviewing outputs. |
| **User** | Any individual using the system, including farmers, agricultural specialists, and visually impaired people looking for an overview of image content. |
| **Visual Analysis** | The process of understanding and interpreting the fundamental elements of an image, carried out by the language model to generate a description. |
| **Visually Impaired User** | A user who accesses the system’s generated descriptions through screen readers, allowing them to interpret visual data related to agriculture. Visually impaired users have partial or complete loss of vision, impacting their ability to perceive visual information. The system is designed to generate screen reader-compatible descriptions to assist users with visual impairments. |