Software Design Specifications

for

AirCraft Classification

Prepared by:
ritika bachupali(se22ucse225)
Sreeja pamu(se22ucse257)
lakshmi vyshali(se22ucse280)
yellapragada pranav(se22ucse301)
merajuddin mohammed(se22ucse307)
alla sathwika(se22ucse311)
rohan gunda(se22ucse312)
theepireddy karthik reddy(se22ucse324)

Document Information

Title: AirCraft Classifcation	
Project Manager:	**Document Version No:1.0.0**
	Document Version Date:09-02-2025
Prepared By:	**Preparation Date:09-04-2025**

Version History

Ver. No.	**Ver. Date**	**Revised By**	**Description**	**Filename**
1.0.0	09-04-2025	This v1 of SDS	SDS_V1	

Table of Contents

1 Introduction

The Aircraft Classification System Software Design Specifications document outlines the architectural and technical design for a machine learning-based system that automatically classifies aircraft images into four distinct categories: civilian, military, unmanned aerial vehicles (UAVs), and unknown. This document serves as a comprehensive guide for the development team, detailing the system's components, data flows, algorithms, and implementation strategies.

The system leverages neural network techniques to analyze and classify aircraft images from a large dataset of 240,000 images (60,000 per class). The specifications in this document follow standard software engineering practices while incorporating specialized considerations for machine learning applications.

1.1 Purpose

The purpose of this Software Design Specification is to define the structure and behavior of the Aircraft Classification System. It serves as a bridge between the project requirements and the implementation phase by specifying how the system will operate, what modules are involved, and how they interact.

This document is especially useful for:

- Developers implementing and optimizing the neural network.
- Testers developing test plans based on architecture and flow.
- Project managers tracking progress and verifying requirement fulfillment

1.2 Scope

This document covers the design of a machine learning-based aircraft classification system trained on a dataset of 240,000 images (60,000 per class). It includes preprocessing steps like resizing and denoising, followed by model training using neural network-based classification techniques.

The scope includes:

- Image preprocessing and augmentation
- Neural network design and training
- Image classification into four categories: Civilian, Military, UAV, and Unknown
- Evaluation metrics and accuracy analysis

The SDS does not cover video classification, real-time detection, or deployment pipelines (though these can be considered in future extensions).

1.3 Definitions, Acronyms, and Abbreviations

UAV	Unmanned Aerial Vehicle
SDS	Software Design Specification
NN	Neural Network
CNN	Convolutional Neural Network
Dataset	A collection of labeled aircraft images used for training and test
Preprocessing	Image enhancement steps like resizing and noise reduction
Classification	Assigning an input image to one of the four predefined categories

1.4 References

https://www.kaggle.com/datasets/a2015003713/militaryaircraftdetectiondataset

https://www.kaggle.com/datasets/rhammell/planesnet/data

https://github.com/dilsadunsal/HRPlanesv2-Data-Set?utm_source=chatgpt.com

https://www.kaggle.com/datasets/a2015003713/militaryaircraftdetectiondataset

https://www.airliners.net/search?photoCategory=9

unmanned conbat arial bechicles - Google Search

Reference papers

https://www.researchgate.net/publication/280625683_AIRCRAFT_CLASSIFICATION_USING_IMAGE_PROCESSING_TECHNIQUES_AND_ARTIFICIAL_NEURAL_NETWORKS

Remote Sensing Aircraft Classification Harnessing Deep Learning Advancements | IEEE Conference Publication | IEEE Xplore

https://arxiv.org/pdf/1306.5151

https://www.youtube.com/watch?si=ISRkHDKAVa8a5QAd&v=pFiGSrRtaU4&feature=youtu.be&themeRefresh=1

https://github.com/Hamed-Aghapanah/yolo8-based-aircraft-detection

2 Use Case View

The Use Case View for the Aircraft Classification System illustrates how a user interacts with the system to classify images of aircraft into four categories: Civilian, Military, UAV, or Unknown. The diagram defines a simple and linear workflow involving user input and system-driven processing.

The main use cases identified from the diagram are:

- **Upload Image**
- **Preprocess Image**
- **Classify Image**
- **Display Output**

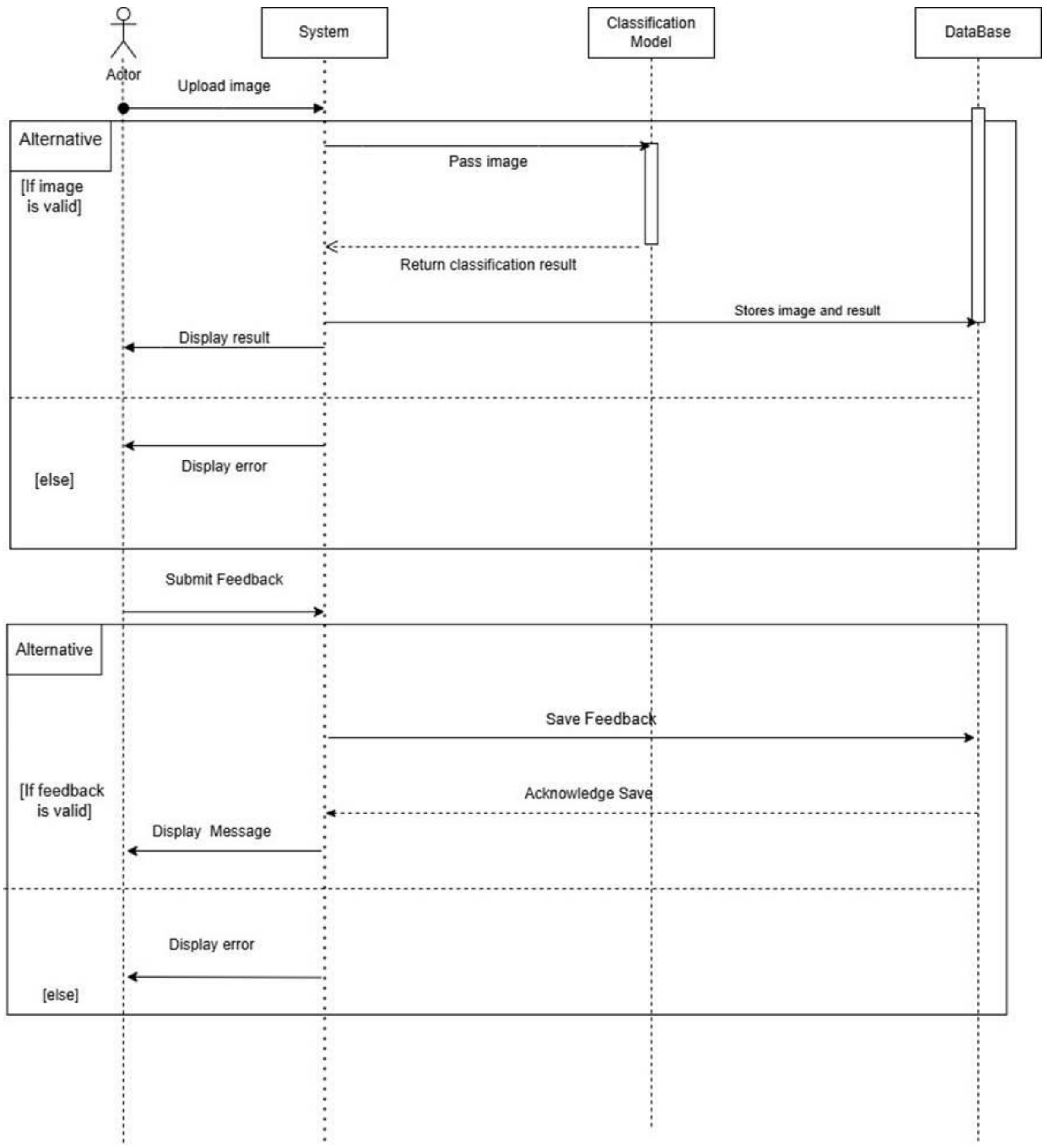
These represent the central functionality of the system, starting from image acquisition and ending in classification display. Each use case directly aligns with a major component in the design and processing pipeline.

Use Case Diagram Description

1. The actor is the User, who initiates the workflow.
2. The User interacts with the system to:

- Upload an image of an aircraft.
- Allow the system to preprocess the image (resizing, denoising).
- Use a trained neural network model to classify the image.
- view the final classification result displayed by the system.

.



2.1 Use Case

Use Case Name: Aircraft Image Classification

Brief Description:

This use case represents the sequence of actions performed by the system when a user uploads an aircraft image. The image is processed, analyzed by a trained neural network model, and a predicted class is returned.

Actors:

- **User** – Initiates the process by uploading an image.

Preconditions:

- The model is trained and deployed.
- The image is in an accepted format (e.g., JPEG, PNG).
- The system services are active and functional.

Postconditions:

- The system outputs the predicted aircraft class on the user interface.

Basic Flow:

1. The User uploads an aircraft image to the system.
2. The system performs preprocessing, including resizing and noise reduction.
3. The preprocessed image is passed to the classification model, which predicts the class.
4. The classified result (Civilian, Military, UAV, Unknown) is displayed to the user.

Alternative Flows:

- If the uploaded file is invalid or unsupported, the system notifies the user and requests a re-upload.
- If the model returns low confidence or an unrecognized result, the system may label the image as "Unknown".

3 Design Overview

3.1 Design Goals and Constraints

The primary design goal is to build an efficient and scalable system for classifying aircraft images into civilian, military, drone, or unknown categories. Key constraints include the use of machine learning frameworks like TensorFlow or PyTorch, support for GPU acceleration, and integration with a pre-existing surveillance infrastructure. The design also accommodates limited development time and must align with available open-source tools and a small team of developers. Compatibility with legacy image databases and adherence to performance benchmarks like image processing speed, classification accuracy which also influenced the design strategy.Furthermore, the design had to ensure scalability to accommodate growing datasets and adaptability to improvements in model architecture over time.

3.2 Design Assumptions

The design of the system is based on several key assumptions that influence its architecture and performance:

- **Image Source and Format**:
  All input images are assumed to be captured by fixed, ground-based camera systems and provided in standard image formats like JPEG or PNG, with resolutions compatible with the model's input specifications.

- **Computational Infrastructure**:
  Adequate hardware resources, including GPU-enabled systems, are assumed to be available during both training and inference stages to ensure high-speed processing.

- **Connectivity**:
  The system assumes stable and reliable network connectivity for the transmission of images from external surveillance systems, whether in real-time or scheduled batch uploads.

- **Pre-Labeled Dataset Availability**:
  A sufficiently large and accurately labeled historical image dataset is assumed to exist for effective model training and validation.

- **User Access Management**:
  It is assumed that user roles, permissions, and access controls are managed externally or through an integrated authorization framework, minimizing the need for custom user management within the application.

3.3 Significant Design Packages

The design model is decomposed into multiple functional packages, each addressing a specific aspect of the system workflow. The architecture follows a layered approach to promote separation of concerns, modularity, and maintainability.

1. **Data Ingestion Package**: Handles input from surveillance systems, both real-time streams and batch uploads, and routes the images for processing.

2. **Preprocessing Package**: Normalizes and resizes images, removes noise, and prepares inputs suitable for the machine learning model.

3. **Result Management Package**: Stores predictions, metadata, and confidence scores in a structured database with indexing for quick retrieval.

4. **Monitoring and Alerting Package**: Tracks system performance, manages logs, and raises alerts in case of anomalies or failures.

3.4 Dependent External Interfaces

The classification system relies on several external modules and applications to function effectively. These dependencies ensure interoperability with surveillance hardware, alert systems, and administrative platforms. The following table summarizes key external interfaces:

The table below lists the public interfaces this design requires from other modules or applications.

External Application and Interface Name	**Module Using the Interface**	**Functionality/ Description**
Surveillance System	Data Ingestion Module with interface name : image reciever	This interface accepts images streamed or batch uploaded from surveillance devices. It ensures images are passed into the processing pipeline in real time.
Alert Management System	Monitoring Module	This interface is responsible for raising alerts based on detected anomalies or specific classification outcomes, such as identifying a military aircraft in a civilian zone.
Admin Dashboard	Monitoring/Result Module	publishes system health stats and classification summaries to the dashboard used by the analytics and administrators.

3.5 Implemented Application External Interfaces (and SOA web services)

In addition to consuming external services, the application provides multiple interfaces for external systems to retrieve results and monitor operations. These interfaces are built using RESTful APIs and can be consumed by third-party applications or internal tools.

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	**Module Implementing the Interface**	**Functionality/ Description**
ClassificationAPI	Result Management Module	This API provides access to classification results, including predicted class, confidence scores, and timestamp for each processed image. It supports querying by image ID or date range.
SystemHealthAPI	Monitoring Module	Offers real-time metrics about system uptime, response latency, queue lengths, and resource usage. Useful for integration with monitoring tools like Grafana or Prometheus.
ImageSubmission API	Data Ingestion Module	Allows external systems to upload images for classification. Accepts metadata and initiates automated processing.

4 Logical View

The logical view provides a structured framework for understanding how the system functions at a conceptual level. It outlines the main components and their interactions, focusing on the step-by-step logic required to process aircraft images, classify them into categories such as commercial or military, and produce annotated outputs. By presenting the flow of data through well-defined modules—data ingestion, processing, post-processing, visualization, and output—the logical view establishes a clear path from raw inputs to actionable results.

4.1 Design Model

This section breaks the system into individual modules and their respective classes, showing how they work together to achieve the intended behavior. The design model represents the system's structure and describes each class's responsibilities, attributes, and methods.

- Modules and Responsibilities:
  - Input Module:

    ImageLoader: Handles image retrieval, validation, and format consistency. This class ensures that only properly formatted images proceed to the processing stage.

  - Processing Module:

    YOLOv8Runner: Runs YOLOv8 inference on input images. It loads the trained model, applies it to detect aircraft, and returns bounding boxes and classifications.

  - Post-Processing Module:

    ResultProcessor: Refines YOLOv8 output by adjusting bounding box coordinates, applying labels, and removing low-confidence detections. This ensures that the results are as accurate and meaningful as possible.

  - Visualization Module:

    ResultRenderer: Overlays bounding boxes and class labels on the original images, creating visually interpretable outputs.

  - Output Module:

    ResultSaver: Stores the annotated images and their associated metadata in a structured output directory, making them available for downstream use or analysis.

- Class Relationships:
  Each module contains one or more classes that interact to implement specific functionalities. For example, the ImageLoader class provides validated inputs to YOLOv8Runner, whose output is then processed by ResultProcessor. The ResultRenderer takes this refined data to

create annotated images, which are finally saved by ResultSaver. The class diagrams illustrate these relationships, showing how data flows between modules and how each component fits into the larger design.

4.2 Use Case Realization

Use Case: Annotating Aircraft Images

- Actors: System administrators who run the process and reviewers who interpret the results.
- Description: The system identifies and classifies flying aircraft in a set of input images. It then generates annotated images with labeled bounding boxes and stores these results for review or further analysis.
- Process Steps:
    1. ImageLoader retrieves and validates images from the input directory.
    2. YOLOv8Runner processes each image, detecting aircraft and classifying them into predefined categories.
    3. ResultProcessor refines these initial classifications, ensuring accuracy by adjusting bounding boxes and applying labels.
    4. ResultRenderer generates annotated images by overlaying the refined classifications and bounding boxes on the original files.
    5. ResultSaver organizes and saves these annotated images, along with any relevant metadata, into the output directory.
- Detailed Interaction:
  Each class within the modules contributes to the realization of this use case. The ImageLoader passes its output to YOLOv8Runner, which generates preliminary results that are improved by ResultProcessor. Once the annotations are finalized, ResultRenderer and ResultSaver ensure the data is presented and stored correctly. This logical sequence ensures that each step builds on the previous one, resulting in a complete, end-to-end solution.

5. Data View

This section details how data flows through the system, particularly in relation to the YOLOv8 object detection model. It includes the conceptual entities, relationships, and persistent storage schemas for handling image data, detection results, and classification metadata. We'll also be going through the detailed connection between the individual layers and how all of them come together to make the model work.

5.1 Domain Model

YOLOv8 processes visual data and outputs bounding boxes, class IDs, and confidence scores. The domain model reflects these core components.

Key Entities:

- **ImageFrame**

  - Input frame/image provided to YOLOv8.
  - Attributes: `frame_id`, `file_path`, `timestamp`, `sensor_id`
- **Detection**
  - YOLOv8 output per object instance in a frame.
  - Attributes: `detection_id`, `frame_id`, `class_id`, `confidence`, `bounding_box`
- **ClassLabel**
  - Mapping from YOLOv8 class ID to human-readable aircraft type.
  - Attributes: `class_id`, `class_name`, `category`
- **Sensor**
  - Device or system capturing frames.(for this project sake, we'll be uploading the pre sourced images using forward-feed methods)
  - Attributes: `sensor_id`, `type`, `location`, `resolution`

5.2 Data Model (persistent data view)

Example Database Collections / Tables:

image_frames

```
frame_id UUID PRIMARY KEY,
file_path TEXT,
timestamp TIMESTAMP,
sensor_id UUID REFERENCES sensors(sensor_id)
```

detections

```
detection_id UUID PRIMARY KEY,
frame_id UUID REFERENCES image_frames(frame_id),
class_id INTEGER REFERENCES class_labels(class_id),
confidence FLOAT,
bounding_box JSON -- {x_center, y_center, width, height, format: 'xywh'}
```

class_labels

```
class_id INTEGER PRIMARY KEY,
class_name TEXT,        -- e.g., 'Fighter Jet', 'Drone'
category TEXT           -- e.g., 'Military', 'Civilian'
```

sensors

```
sensor_id UUID PRIMARY KEY,
type TEXT,              -- 'Camera', 'Drone', etc.
```

```
    location GEOPOINT,
    resolution TEXT         -- e.g., '1920x1080'
```

5.2.1 Data Dictionary

FIELD NAME	DESCRIPTION	TYPE	EXAMPLE
file_path	**Path to stored image/frame**	**TEXT**	**/images/frame_2025_04_07.jpg**
timestamp	**Capture time of the frame**	**TIMESTAMP**	**2025-04-07 10:20:15**
bounding_box	**YOLOv8 box output in `x_center`, `y_center`, `w`, `h`**	**JSON**	**{x:512, y:320, w:120, h:90}**
confidence	**Detection confidence score**	**FLOAT**	**0.88**
class_id	**Integer mapped to a class in the YOLOv8 model**	**INTEGER**	**2**
class_name	**Mapped class label (via lookup)**	**TEXT**	**Commercial Jet**
sensor_id	**Refers to sensor that captured the image**	**UUID**	**d123ef45**

5.3 YOLOv8 Model Architecture Overview

YOLOv8 is a single-stage object detection model designed for high performance and real-time inference. The model is composed of three main components:

1. Backbone

The backbone is responsible for extracting rich feature representations from the input image.

- **Input Layer**
    - Accepts images (e.g., 640×640 RGB) normalized to [0,1].
- **C2f Modules** *(Cross Stage Partial Fusion)*
    - Efficiently extract hierarchical features.
- **Conv + BatchNorm + SiLU Activation**
    - Standard building blocks used throughout for stability and performance.

Output: Multi-scale feature maps.

---

2. Neck

The neck enhances feature maps using a feature pyramid to help detect objects at different scales.

- **FPN (Feature Pyramid Network)**
- **PAN (Path Aggregation Network)**
    - Combines low-level and high-level features to help detect both small and large aircraft.
- **Concat and Upsample Operations**
    - For fusing features across scales.

Output: Refined feature maps at different resolutions.

---

3. Head

The head performs the actual prediction of bounding boxes, class probabilities, and objectness scores.

- **Detection Layers (usually at 3 scales)**
    - For each cell in the feature map, the model predicts:
        - **Bounding Box** → `x_center`, `y_center`, `width`, `height`
        - **Objectness Score** → Confidence of object presence
        - **Class Probabilities** → Aircraft type (e.g., Drone, Fighter Jet, etc.)

Output Shape:
For each scale, output tensor shape is:
`[batch_size, num_anchors × (num_classes + 5), height, width]`
The `+5` corresponds to: `x`, `y`, `w`, `h`, `objectness`.

Example YOLOv8-S (Small) Configuration

COMPONENT	Layers	Output Shape (for 640×640 input)	Notes
Backbone	**C2f, Conv, Downsampling**	**80×80, 40×40, 20×20**	**Multi-scale feature maps**
Neck	**FPN + PAN Layers**	**80×80, 40×40, 20×20**	**Enhanced features**
Head	**Detection heads**	**3 prediction heads**	**one per scale**

Model Flow Diagram (Text Version)

```
[Input Image 640x640x3]
        ↓
      [Backbone]
        ↓
[Multi-scale Feature Maps]
        ↓
        [Neck (FPN + PAN)]
        ↓
   [Refined Features at 3 Scales]
        ↓
       [Head]
        ↓
```

```
[Predictions: bbox + class + objectness]
```

This model structure allows YOLOv8 to achieve fast and accurate detection of aircraft across various sizes and angles — crucial for real-world aerial or surveillance applications.

6 Exception Handling

- **Data Loading Exceptions:**
  Custom exceptions like `ImageLoadException` handle missing, corrupted, or unsupported images during ingestion. Such images are skipped, and warnings are logged with file details and timestamps for future review and replacement.

- **Preprocessing Exceptions:**
  Issues during resizing, normalization, or format conversion raise `PreprocessingException`. Faulty images are excluded, and logs capture image ID and error step. Logs guide whether to fix preprocessing logic or filter such data.

- **Model Training Exceptions:**
  Exceptions due to invalid input shapes, type mismatches, or GPU memory errors are logged with traceback, batch info, and system state. Training halts, and we adjust data, batch size, or model settings before resuming.

- **Inference Exceptions:**
  Unexpected inputs during prediction trigger `InferenceException`. The error is logged with input details, and the affected sample is labeled "Unknown," allowing inference to continue without interruption.

- **Unknown Class Assignment:**
  Low-confidence predictions across all classes default to the "Unknown" category. This is logged with confidence scores, and frequent occurrences prompt threshold tuning or additional training.

- **System-Level and I/O Exceptions:**
  Errors like missing directories, file I/O failures, or memory crashes use standard handlers. Depending on severity, the system retries or exits safely. Logs include error type and system state for manual fixes.

- **Logging and Recovery Strategy:**
  We use Python's logging module to record exceptions with timestamps, severity, and context. In case of failure, checkpointing supports recovery from the last stable state, ensuring minimal disruption.

7 Configurable Parameters

This table describes the simple configurable parameters names (value pairs) used in the application. It also indicates which parameters can be updated at runtime without restarting the application.

Configuration Parameter Name	Definition and Usage	Dynamic?
`image_input_size`	Dimensions to which all input images are resized (e.g., 224x224).	Yes
`batch_size`	Number of images processed together during training/inference. Impacts speed and memory	Yes
`confidence_threshold`	Minimum prediction confidence required to assign a label other than "Unknown".	Yes
`data_augmentation`	Enables augmentation techniques like flipping, rotation during training	Yes
`model_checkpoint_path`	File path to save or load model weights for training or inference	No
`logging_level`	Controls verbosity of logs (INFO, DEBUG, ERROR, etc.).	Yes
`num_workers`	Number of parallel processes used for data loading	No
`max_retry_on_failure`	Number of allowed retries in case of transient I/O or system errors	Yes

8 Quality of Service

This section outlines the critical quality-of-service aspects of the aircraft classification system. It focuses on system availability, security, performance, and monitoring and control mechanisms, ensuring the solution remains reliable, secure, and scalable when deployed in real-world environments such as surveillance, defense, and air traffic monitoring.

8.1 Availability

The availability of the aircraft classifier system is vital, especially if integrated into critical defense or security infrastructure where continuous operation is required. The application is designed to meet high availability standards by incorporating several architectural and operational strategies:

- **Modular independent Design**: System components such as data ingestion, preprocessing, model inference, and result storage operate independently. This ensures that the failure of one component does not cascade and affect the entire system.
- **Redundancy and Failover**: Critical services can be hosted on redundant servers to ensure continued operation in the event of hardware or software failure.
- **Auto-Restart Mechanisms**: Background services and daemons are monitored, and auto-restart policies are configured in case of unexpected termination.
- **Maintenance and Housekeeping**: All housekeeping tasks such as model retraining, log cleanup, and database indexing are scheduled during off-peak hours and do not require downtime.
- **Data Loading**: Bulk uploads or data ingestion operations are designed to run asynchronously, ensuring the main classification service remains responsive during such operations.

8.2 Security and Authorization

Given the sensitive nature of the data (which may include military or surveillance images), the application adheres to stringent security and authorization protocols:

- **Role-Based Access Control (RBAC)**: Users are assigned roles such as Administrator, Annotator, and Viewer, with permissions tailored to each role. For example, only Administrators can trigger model retraining or access logs.
- **Authentication and Authorization**: Secure login mechanisms, including multi-factor authentication (MFA), are supported to prevent unauthorized access.
- **Data Encryption**: Sensitive data, including uploaded images and classification outputs, are encrypted both in transit (TLS) and at rest (AES-256).

8.3 Load and Performance Implications

- **Load Projections:**
  The system is capable of handling up to 60,000 annotated images. It supports batch processing at a rate of up to 500 images per minute and enables continuous inference for real-time surveillance data.
- **Design Implications:**
  To optimize performance, the system uses multi-threading for image preprocessing and GPU acceleration for model inference through frameworks like TensorFlow or PyTorch. The database is optimized with indexing, batch inserts, and data archiving. Caching is also implemented to reduce redundant model usage.
- **Performance Metrics Tracked:**
  The system tracks key metrics such as inference latency, CPU/GPU/memory usage, database growth, and queue backlogs for batch tasks to ensure smooth operations.

8.4 Monitoring and Control

- **Real-Time Monitoring:**
  Monitoring capabilities are exposed through RESTful APIs or integrated dashboards. The system supports integration with tools like Prometheus, Grafana, or the ELK Stack, enabling full observability of critical metrics.
- **Measurable Parameters:**
  Key parameters tracked include image processing rates, failed job counts, resource usage, and service uptime. These metrics provide valuable insights into system performance and stability.
- **Alerting Mechanisms:**
  Alerts are configured to notify administrators of critical issues like model failures, high latency, or storage limits. Notifications are sent via email or SMS for rapid response and issue resolution.
- **Health Checks and Logs:**
  The application provides self-health-check endpoints and detailed logs at each stage of processing. This supports proactive troubleshooting and accountability across system components.