Low Level Design (LLD)

Mushroom Classification Project

**Contents**

[1 Introduction 4](#_Toc163683206)

[1.1 Why this Low-Level Design Document? 4](#_Toc163683207)

[1.2 Scope 4](#_Toc163683208)

[1.3 Constraints 5](#_Toc163683209)

[1.4 Risks 5](#_Toc163683210)

[1.5 Out of Scope 5](#_Toc163683211)

[2 Technical specifications 5](#_Toc163683212)

[2.1 Dataset 5](#_Toc163683213)

[2.1.1 Mushroom dataset overview 5](#_Toc163683214)

[2.1.2 Input schema 6](#_Toc163683215)

[2.2 Predicting Mushroom Edibility 6](#_Toc163683216)

[2.3 Logging 7](#_Toc163683217)

[2.4 Database 7](#_Toc163683218)

2.5 Deployment 7

[3 Technology stack 8](#_Toc163683219)

[4 Proposed Solution 9](#_Toc163683220)

[5 Model training/validation workflow 10](#_Toc163683221)

[6 User I/O workflow 11](#_Toc163683222)

[7 Test cases 12](#_Toc163683223)

[8 Key performance indicators (KPI) 13](#_Toc163683224)

**Abstract**

The Mushroom Classification Project aims to develop a robust machine learning web application capable of accurately predicting the edibility of mushrooms based on their attributes. Leveraging sophisticated MLOps frameworks such as DVC, MLflow, and Docker, the project integrates seamless data management, model tracking, and deployment processes. Through a user-friendly interface, users can input mushroom characteristics and receive immediate predictions regarding their edibility. The project embodies a comprehensive approach, encompassing data fetching, preprocessing, model training, evaluation, and deployment pipelines. By adopting industry best practices and leveraging cloud infrastructure, the Mushroom Classification Project ensures scalability, reliability, and transparency in its operations. Through continuous improvement and innovation, the project seeks to enhance user experience and contribute to the advancement of machine learning applications in real-world scenarios

# Introduction

## Why this Low-Level Design Document?

The purpose of this Low-Level Design Document is to offer a detailed blueprint of the Mushroom Classification Project's system architecture. It serves as a guide for developers, providing clarity on the implementation of various components, their interactions, and data flow within the system. Additionally, this document aids in ensuring consistency and coherence throughout the development process by defining clear specifications for each component.

The main objective of the Mushroom Classification Project is to develop a sophisticated machine learning-based web application that accurately predicts the edibility of mushrooms based on user-provided characteristics.

Various needs for validating this project are -

* Safety Priority: Accurately differentiate edible and poisonous mushrooms to prevent health risks.
* Complex Identification: Visual similarities among species challenge manual mushroom identification.
* Automation Advantage: Machine learning enhances accuracy and efficiency in mushroom classification.



## Scope

The scope of the Mushroom Classification Project's Low-Level Design (LLD) encompasses defining and detailing the interactions, components, and data flow within the system. It involves breaking down functionalities such as data fetching, transformation, model training, and deployment into specific components.

## Constraints

Resource availability, data quality, model accuracy, and deployment infrastructure limitations.

## Risks

Model inaccuracies, user misinterpretation, reliance on external data, security vulnerabilities, and deployment failures pose risks.

## Out of Scope

Hardware design, field data collection, extensive user training, and regulatory compliance are out of scope.

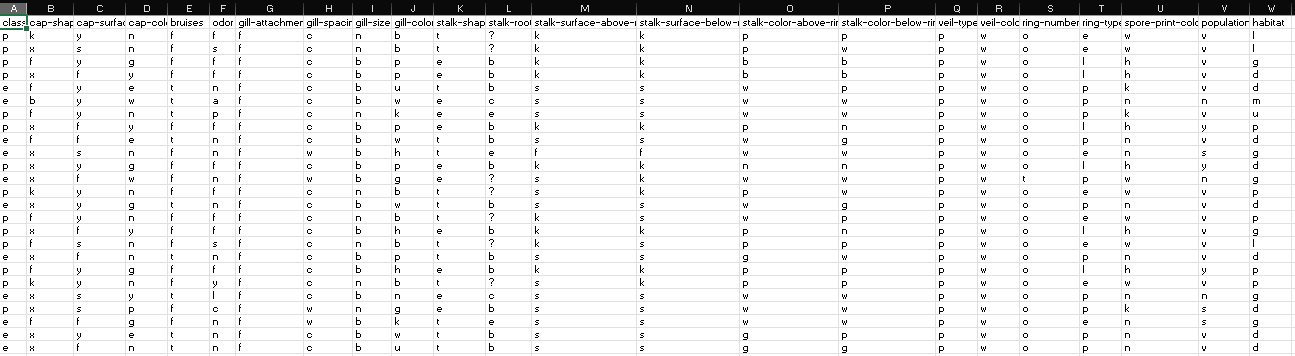
# Technical specifications

## 2.1 Dataset

|  |  |  |
| --- | --- | --- |
| **Mushroom Classification** | **Finalized** | **Source** |
| Mushroom.csv | yes | Kaggle.com/datastax astra |

## 2.1.1 Mushroom dataset overview

The Mushroom Classification dataset from Kaggle contains 8124 records/mushroom samples, each described by various attributes such as cap shape, surface, color, and odor. It includes a binary label indicating edibility ('edible' or 'poisonous'). The dataset is pre-processed to ensure quality, handling missing values and encoding categorical variables. Class imbalance may exist, necessitating robust model development.

**Mushrooms.csv Sample**

## 2.1.2 Input schema

| **Feature Name** | **Data Type** | **Size** | **Null/Required** |
| --- | --- | --- | --- |
| Cap Shape | String | Variable | Required |
| Cap Surface | String | Variable | Required |
| Cap Color | String | Variable | Required |
| Bruises | String | Variable | Required |
| Odor | String | Variable | Required |
| Gill Attachment | String | Variable | Required |
| Gill Spacing | String | Variable | Required |
| Gill Size | String | Variable | Required |
| Gill Color | String | Variable | Required |
| Stalk Shape | String | Variable | Required |
| Stalk Root | String | Variable | Required |
| Stalk Surface Above Ring | String | Variable | Required |
| Stalk Surface Below Ring | String | Variable | Required |
| Stalk Color Above Ring | String | Variable | Required |
| Stalk Color Below Ring | String | Variable | Required |
| Veil Type | String | Variable | Required |
| Veil Color | String | Variable | Required |
| Ring Number | String | Variable | Required |
| Ring Type | String | Variable | Required |
| Spore Print Color | String | Variable | Required |
| Population | String | Variable | Required |
| Habitat | String | Variable | Required |
| Edibility | String | Variable | Required |

## 2.2 Predicting Mushroom Edibility

1. Utilizes machine learning to predict mushroom edibility accurately.
2. Input data transformed and processed for model compatibility.
3. Features selected for optimal prediction performance.
4. Classification model distinguishes between edible and poisonous mushrooms.
5. Predictions aid in mushroom identification and safety assurance.
6. User-friendly interface facilitates seamless input and prediction retrieval.

## 2.3 Logging

We should be able to log every activity done by the user.

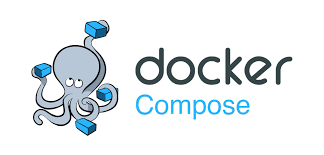
1. **Centralized Logging:** Logger.py initializes centralized logger object for consistent logging.
2. **Standardized Configuration:** BasicConfig ensures consistent logging configuration across project files.
3. **Enhanced Debugging:** Logger integration aids in debugging and troubleshooting efforts.
4. **Error Identification:** Logs assist in identifying and tracing errors during execution.
5. **Real-time Monitoring:** Facilitates real-time monitoring of application behavior and performance.
6. **Issue Resolution:** Log messages aid in identifying and resolving issues promptly.

## 2.4 Database

The project employs DataStax Astra as its database, fetching data exclusively from the main\_db. Once retrieved, collections are transferred to the archived database for preservation. Subsequently, these collections are removed from the main\_db, streamlining data management processes while ensuring efficient storage and retrieval mechanisms.

**2.5 Deployment**

1. AWS 2. Docker-compose



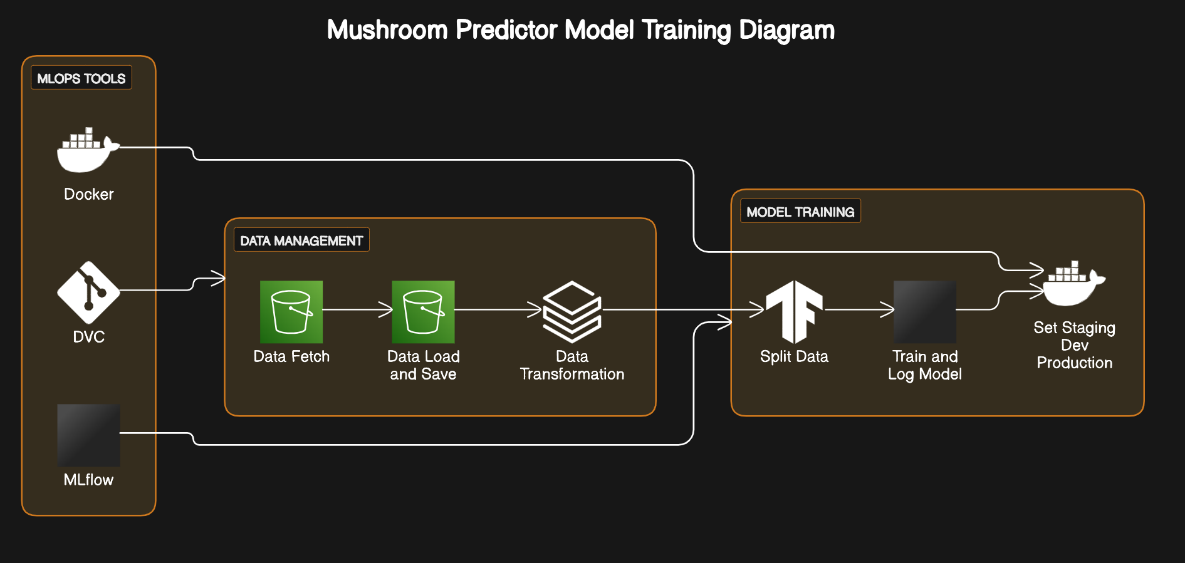
# Technology stack

|  |  |
| --- | --- |
| **Front End** | HTML/CSS/JS |
| **Backend** | Python Flask |
| **Database** | Data Stax Astra (Cassandra) |
| **Deployment** | AWS Docker-compose |
| **MLOps Tools** | Docker DVC Mlflow |
| **Source Control Management** | Git/Github |

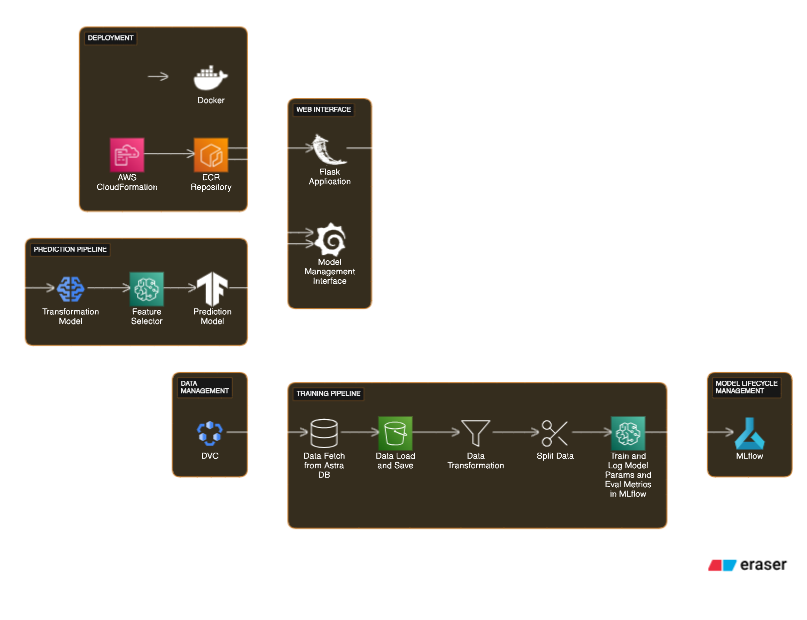
# Proposed Solution

Developing a comprehensive web application leveraging machine learning algorithms to accurately classify mushrooms based on their attributes. The solution integrates advanced MLOps frameworks such as DVC for data versioning, MLflow for experiment tracking, and Docker for containerization. Users can easily input mushroom characteristics via a user-friendly web interface, receiving immediate predictions on their edibility. The system ensures reliability and scalability through AWS CloudFormation for automated infrastructure provisioning. Continuous monitoring and maintenance ensure optimal performance, with updates deployed seamlessly using Docker containers. The solution mitigates health risks associated with misidentification of mushrooms, providing a reliable and accessible tool for users.

# Model training/validation workflow



# User I/O workflow



# Test cases

| **Test Case** | **Steps to Perform** | **Module** | **Pass/Fail** |
| --- | --- | --- | --- |
| 1. Predict Mushroom Edibility | Enter mushroom attributes in the web interface. | Prediction Interface | Pass (if correct prediction), Fail (if incorrect prediction) |
| 2. Validate Data Transformation | Verify transformed data matches expected format. | Data Transformation | Pass (if transformation correct), Fail (if incorrect transformation) |
| 3. Split Data for Training and Testing | Check if data split into training and testing sets. | Data Splitting | Pass (if data split correctly), Fail (if incorrect splitting) |
| 4. Train Classification Model | Train model and log parameters with MLflow. | Model Training | Pass (if model trained successfully), Fail (if training fails) |
| 5. Predict Mushroom Edibility (Integration Test) | Submit sample data through the Flask app. | Flask Application | Pass (if correct prediction), Fail (if incorrect prediction) |
| 6. Model Version Control | Verify proper tracking of model versions. | Model Management | Pass (if versions tracked correctly), Fail (if tracking fails) |
| 7. Data Quality Enhancement | Ensure noise reduction and missing value handling. | Data Preprocessing | Pass (if data quality improves), Fail (if quality remains poor) |
| 8. Deployment Efficiency | Deploy application on AWS with Docker. | Deployment Pipeline | Pass (if deployment successful), Fail (if deployment fails) |
| 9. Security Compliance | Verify adherence to data security standards. | Security Measures | Pass (if compliant), Fail (if non-compliant) |
| 10. User Interaction | Assess user engagement with the prediction interface. | Prediction Interface | Pass (if high engagement), Fail (if low engagement) |

# Key performance indicators (KPI)

1. **Response Time:** Assess promptness in providing prediction results.
2. **False Positive Rate:** Evaluate rate of misclassifying edible mushrooms.
3. **Data Quality Enhancement:** Improve dataset quality for better model performance.
4. **Deployment Efficiency:** Monitor effectiveness of deployment processes.
5. **Model Version Control:** Track changes and versions in classification model.
6. **Security Compliance:** Ensure adherence to data security standards.