Mushroom Prediction

Revision Number: 1

Last Date of Revision: 04/07/2025

# 

# Document Version Control

| Date Issued | Version | Description | Author |
| --- | --- | --- | --- |
| 04/07/2025 | 1.0 | Initial HLD — Mushroom Classifier | Agam Patel |

# Table of Contents

| 1. Document Version Control | 1 |
| --- | --- |
| 1. Abstract | 3 |
| 1. Introduction | 4 |
| 1. Why This HLD Document | 5 |
| 1. Scope | 6 |
| 1. Definitions / Abbreviations | 7 |
| 1. Proposed System Design | 8 |
| 1. Functional Requirements | 9 |
| 1. Non-Functional Requirements | 9 |
| 1. Software and Tools Used | 10 |
| 1. Architecture and Pipelines Flow | 11 |
| 1. Module-wise Summary | 12 |
| 1. Future Scope | 13 |

# 

# Abstract

The increase in mushroom foraging and consumption has raised concerns regarding the accurate identification of edible vs. poisonous mushrooms. Misidentification can lead to serious health hazards or fatalities. This project presents a machine learning–based solution to classify mushrooms as either edible or poisonous based on various morphological characteristics.

The solution uses a pre-processed dataset from UCI Machine Learning Repository. The model pipeline includes data ingestion, validation, transformation, training with multiple classifiers, evaluation, and prediction — all structured using MLOps principles with MLflow and DVC. This project demonstrates the integration of machine learning with operational best practices to ensure accuracy, reproducibility, and maintainability.

# Introduction

The Mushroom Classification project is a machine learning-based system designed to accurately classify mushrooms as edible or poisonous using their physical and morphological attributes. The solution is implemented using modular Python code and follows MLOps practices to ensure reproducibility, scalability, and ease of deployment.

This HLD (High-Level Design) document outlines the major components of the system, including data pipelines, model training strategies, evaluation techniques, and prediction interfaces. It serves as a blueprint for understanding the architecture, workflows, and design considerations of the project.

# Why This HLD Document?

This document provides a high-level overview of the system to:

* Clearly communicate the architectural components and their roles
* Enable collaboration among data scientists, developers, and reviewers
* Serve as a base reference for LLD (Low-Level Design) and implementation
* Ensure traceability of decisions made during the system’s design

# Scope

The scope of this project includes:

* Building an ML pipeline to classify mushrooms using supervised learning
* Implementing all stages: data ingestion, validation, transformation, training, evaluation, and prediction
* Using MLOps tools like DVC (Data Version Control) and MLflow for experiment tracking and reproducibility
* Supporting a prediction interface that accepts user inputs and returns edible/poisonous classification
* Ensuring all code is modular and scalable for future enhancements

# Definitions / Abbreviations

| **Term** | **Definition** |
| --- | --- |
| HLD | High Level Design |
| LLD | Low Level Design |
| MLOps | Machine Learning Operations |
| MLfllow | Open-source platform for managing ML lifecycle |
| DVC | Data Version Control |
| UI | User Interface |
| ML | Machine Learning |
| CSV | Comma Separated Values |

# 5. Proposed System Design

The Mushroom Classification System is structured into the following major modules:

Data Ingestion: This module is responsible for downloading the mushroom dataset from an online source using the KaggleHub API and saving the raw CSV file in a predefined local directory structure for further processing.

Data Validation: The validation module ensures that the dataset matches the expected schema, contains no null or inconsistent values, and that the class distribution is acceptable. This step helps maintain data quality before any processing or model training begins.

Data Transformation: In this stage, all categorical features are encoded using OrdinalEncoder, while the target column ('class') is encoded using LabelEncoder. The transformed dataset is then split into training and testing sets and stored for use by the model training module. Encoders are saved separately for use during prediction.

Model Training: Multiple machine learning models such as Logistic Regression, Random Forest, and XGBoost are trained using the transformed training data. Hyperparameter tuning is applied to identify the best-performing model. Model performance metrics and models themselves are tracked using MLflow, and the best model is saved in the model artifact directory.

Model Evaluation: This module evaluates the performance of the trained model on both training and test data. Key metrics such as accuracy, precision, recall, and F1-score are computed. The results are saved in a report file and also logged to MLflow for comparison and reproducibility.

Prediction Pipeline: The prediction pipeline takes user input in the form of feature values (as original category names), encodes them using the previously saved encoders, and passes them to the trained model. The model then returns whether the mushroom is edible or poisonous.

MLOps Tools

DVC (Data Version Control) is used to version the dataset and track pipeline stages, ensuring reproducibility and consistency.

MLflow is used for logging models, metrics, parameters, and artifacts, enabling experiment tracking and model lifecycle management.

# 6. Functional Requirements

The following are the key functional requirements for the Mushroom Classification System:

The system must be able to download the dataset from an online source and store it in the appropriate directory for processing.

The system must validate the dataset structure, including column names, null values, and basic distribution checks, before allowing further processing.

The system must encode all categorical feature columns using Ordinal Encoding and the target column using Label Encoding to prepare the data for model training.

The system must split the transformed data into training and testing sets and save them for consistent and reproducible training and evaluation.

The system must support training of multiple machine learning models and perform hyperparameter tuning to select the best-performing model.

The system must log model metrics such as accuracy, precision, recall, and F1-score for both training and testing sets.

The system must save the best model and related artifacts (like encoders and reports) for future use.

The system must be able to accept new input from users in the form of categorical features and return a prediction indicating whether the mushroom is edible or poisonous.

The system must support MLflow for experiment tracking and model logging, and DVC for data and pipeline versioning.

# 7. Non-Functional Requirements

The non-functional requirements define how the system should behave and set constraints on its development. The following are the key non-functional requirements for the Mushroom Classification System:

Modularity: The system must follow a modular code structure so that each component (data ingestion, transformation, training, etc.) can be independently tested, maintained, and reused.

Scalability: The pipeline should be designed to scale with larger datasets or additional model types without requiring a complete rewrite.

Reproducibility: Using tools like DVC and MLflow, the system must support full reproducibility of results, including datasets, parameters, models, and metrics.

Reliability: All stages of the pipeline should be robust to invalid inputs and failures, with appropriate error handling and logging in place.

Maintainability: The codebase should follow clean coding practices with proper documentation, configuration files, and separation of logic to allow easy updates and collaboration.

Performance: Model training and prediction should be efficient and optimized for reasonable execution time even on mid-level systems.

Usability: The system should provide a user-friendly interface (CLI or optional Streamlit UI) that allows users to input data and interpret predictions easily.

Version Control: All code, data, and models must be version controlled using Git, DVC, and MLflow to ensure traceability and collaborative development.

# 8. Software and Tools Used

The following software, frameworks, and tools are used in the development and deployment of the Mushroom Classification System:

Python: Primary programming language used for all modules, including data processing, model training, and pipeline integration.

Pandas & NumPy: For data manipulation, transformation, and numerical operations.

Scikit-learn: For implementing machine learning models, preprocessing, model evaluation metrics, and hyperparameter tuning.

XGBoost: Used as one of the primary classifiers due to its high performance and flexibility.

Joblib: For serialization and saving of trained models and encoders.

MLflow: Used for experiment tracking, logging of metrics and parameters, and model versioning.

DVC (Data Version Control): For versioning datasets and pipeline stages, ensuring reproducibility and traceability of results.

KaggleHub: To programmatically download datasets from the Kaggle repository.

PyYAML: Used to handle configuration and parameter files (config.yaml, params.yaml) in a structured way.

Logging (Python Standard Library): For consistent runtime logging of events and tracking of errors across modules.

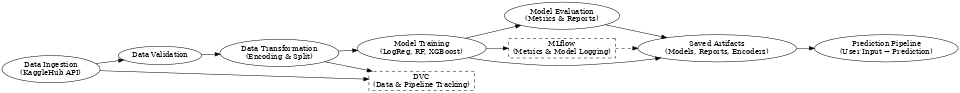
VS Code / Jupyter Notebooks: Used for writing, testing, and experimenting with code during development.

Git & GitHub: Version control and collaborative source code management.

# 9. Architecture and Pipeline Flow

The Mushroom Classification System follows a modular architecture that separates each stage of the ML pipeline into distinct components. The pipeline is orchestrated in the following sequence:

1. Data Ingestion  
    Downloads the raw mushroom dataset using KaggleHub and stores it locally.
2. Data Validation  
    Validates schema, checks for missing or invalid values, and ensures data readiness.
3. Data Transformation  
    Applies encoding to all categorical features and the target label, followed by train-test split. Encoders are saved for future reuse.
4. Model Training  
    Trains multiple models (Logistic Regression, Random Forest, XGBoost), applies hyperparameter tuning, and logs experiments using MLflow.
5. Model Evaluation  
    Evaluates performance using key metrics (accuracy, precision, recall, F1-score) and saves reports.
6. Model & Artifact Storage  
    The best model, encoders, and evaluation reports are saved as artifacts for deployment and reuse.
7. Prediction Pipeline  
    Accepts user input in original string format, encodes it using saved encoders, and returns prediction using the trained model.
8. MLOps Tools  
   * MLflow is used for tracking experiments, models, and parameters.
   * DVC is used for versioning datasets, artifacts, and pipeline stages to ensure reproducibility.



# 10. Module-wise Summary

This section provides a brief overview of each core module developed as part of the Mushroom Classification System:

1. Data Ingestion  
   * Downloads the mushroom dataset from the Kaggle repository using the KaggleHub API.
   * Stores it in a local directory (artifacts/raw/) for further processing.
2. Data Validation  
   * Checks whether the dataset follows the expected schema.
   * Verifies the presence of all required columns, ensures there are no null values, and checks for class distribution balance.
3. Data Transformation  
   * Uses OrdinalEncoder for all categorical input features.
   * Uses LabelEncoder for the target variable (class).
   * Splits the transformed dataset into training and testing sets.
   * Saves encoded data and encoders for prediction reuse.
4. Model Training  
   * Trains Logistic Regression, Random Forest, and XGBoost classifiers.
   * Performs hyperparameter tuning using cross-validation.
   * Logs all experiments, metrics, and artifacts to MLflow.
5. Model Evaluation  
   * Computes evaluation metrics like accuracy, precision, recall, and F1-score on both training and test sets.
   * Saves results as reports.json and logs to MLflow.
6. Model Prediction  
   * Loads the trained model and saved encoders.
   * Accepts user input in original label format.
   * Encodes the input and returns the classification result: edible or poisonous.
7. MLOps Integration  
   * DVC is used for versioning data and tracking pipeline stages.
   * MLflow is used to manage experiments, models, and their metadata.

# 11. Future Scope

The Mushroom Classification System can be significantly enhanced in future iterations. One of the primary additions would be the development of a user-friendly web interface using Streamlit, allowing users to select mushroom characteristics through dropdown menus and receive real-time predictions. Furthermore, the trained model can be containerized using Docker and deployed with FastAPI or Flask, enabling it to serve predictions over REST APIs in production environments.

To streamline the workflow, CI/CD pipelines can be integrated using GitHub Actions or similar tools, automating tasks such as data validation, model retraining, testing, and deployment. Monitoring data drift is another critical enhancement, which can be addressed by integrating tools like EvidentlyAI to detect when incoming data distributions deviate from the training data, thereby preserving model accuracy and reliability.

Lastly, incorporating model explainability frameworks like SHAP or LIME would provide transparent insights into how predictions are made, helping end users and stakeholders understand the impact of individual features. These enhancements would make the system more robust, scalable, interpretable, and ready for real-world deployment.