📘 US Visa Classification ML System High-Level Design Document – v1.0  
Author: Pankaj Kumar Pramanik  
Date: October 2025

## Document Version Control

| Version | Date | Description | Author |
| --- | --- | --- | --- |
| 1.0 | 17-Oct-2025 | Initial Version | Pankaj Kumar Pramanik |

## List of Figures

1. ML Pipeline Architecture
2. Data Ingestion Flow
3. Data Validation Checks
4. Data Transformation Process
5. Model Training Architecture
6. Evaluation and Deployment Logic

## Abstract

This document presents the high-level design of a machine learning pipeline for predicting US visa approvals. The solution involves data ingestion from MongoDB, data preprocessing, model training, evaluation, and deployment. The architecture is modular, scalable, and designed for automation and easy maintenance.

## 1. Introduction

### 1.1 Why this High-Level Design Document?

This HLD document outlines the architecture, components, and data flow of a predictive ML pipeline for US visa applications. It serves as a reference for developers, data engineers, and stakeholders.

### 1.2 Background

#### 1.2.1 Project Overview

The system aims to automate US visa classification (approval/rejection) based on historical application data using machine learning techniques.

#### 1.2.2 Data Source

The dataset is obtained from Kaggle and stored in MongoDB. It includes fields such as education level, visa status, job titles, and years of experience.

#### 1.2.3 Pipeline Architecture

The architecture includes modular stages: ingestion → validation → transformation → training → evaluation → pusher → prediction.

#### 1.2.4 ML Objective

Predict binary outcomes (Approved / Rejected) using optimized models based on F1-score.

### 1.3 Problem Statement

#### 1.3.1 Use Cases

* Automating visa status predictions.
* Integrating model output into a decision-support system.
* Building reusable ML infrastructure for similar classification tasks.

#### 1.3.2 Application Flow

1. Fetch data from MongoDB
2. Validate schema and detect drift
3. Preprocess, encode, scale, and handle imbalance
4. Train multiple models and select best
5. Evaluate vs. production model
6. Deploy best model

### 1.4 Definitions

| Term | Definition |
| --- | --- |
| Drift | Change in data distribution over time |
| SMOTEENN | Oversampling and undersampling technique for class imbalance |
| Artifact | Serialized object (model, preprocessor) or metadata file |
| Model Factory | Framework to evaluate multiple models for selection |

## 2. General Description

### 2.1 Product Perspective

The product functions as a backend service that can be plugged into any frontend or reporting dashboard to generate predictions for US visa applications.

### 2.2 Technical Requirements

* Python 3.8+
* MongoDB instance for raw data
* GitHub for version control
* Docker for containerization
* Conda environment for package isolation

### 2.3 Data Requirements

#### 2.3.1 US Visa Dataset

Sourced from Kaggle: Easy Visa Dataset

#### 2.3.2 Feature Columns

* Education Level
* Job Title
* Company Name
* Experience
* Country of Origin
* Visa Type

#### 2.3.3 Target Variable

* Visa Status (Approved = 1, Rejected = 0)

### 2.4 Tools Used

#### 2.4.1 Python & Libraries

* Pandas, NumPy, Scikit-learn, Imbalanced-learn, PyYAML

#### 2.4.2 MongoDB

* Used as the initial raw data store.

#### 2.4.3 Scikit-learn

* For ML model development, pipeline creation, and evaluation.

#### 2.4.4 Docker & GitHub

* Docker for packaging and GitHub for code collaboration and CI.

### 2.5 Constraints

* Class imbalance in dataset
* Missing values in raw data
* Computational limits for training on large datasets

### 2.6 Assumptions

* The structure of input data remains unchanged over time.
* The training dataset is representative of future data.
* Internet access is available for S3 operations if used.

### 2.7 Objective

To design a robust, modular, and automated ML system that can accurately classify US visa applications and seamlessly transition from development to deployment.

## 3. Design Details

### 3.1 Process Flow

1. Data pulled from MongoDB
2. Stored into feature store
3. Validated for schema and drift
4. Preprocessing applied (encoding, scaling)
5. SMOTEENN handles class imbalance
6. Model Factory evaluates candidates
7. Best model is saved and deployed

### 3.2 Component Interaction

* **DataIngestion** interacts with MongoDB and produces CSVs
* **DataValidation** ensures data quality
* **DataTransformation** outputs train.npy, test.npy, preprocessor.pkl
* **ModelTrainer** uses config to load/train/evaluate models
* **ModelPusher** stores model locally/S3 for deployment

### 3.3 Error Handling

* Structured logs maintained via logger module
* Try/except blocks around each critical operation
* Invalid/missing files raise custom exceptions

### 3.4 Performance

* **Evaluation metric:** F1-score
* Cross-validation enabled
* Tracks:
  + Training Time
  + Model Size
  + Score Comparison (New vs. Old Model)