IntelliInspect
Design and Usage Guide

Team-9

Team Members:
KATHIRAVAN N

PARUPATI ABHINAV

ADITYA KUMAR SINGH

SOMESHWARAN

CONTENTS:

Introduction:

IntelliInspect is a full-stack AI-powered web application designed for real-time predictive quality control using Kaggle production line sensor data. It enables users to upload datasets, define time-based splits for training/testing/simulation, train machine learning models, and simulate real-time predictions. The app is built with an Angular frontend for user interaction, an ASP.NET Core backend for API orchestration, and a Python ML service using FastAPI for model training and inference. All components are containerized with Docker for easy deployment.

This documentation covers the system's architecture, data flows, API contracts, design details, deployment instructions, and usage guide.

Objective: Process Kaggle CSV data (e.g., Bosch Production Line dataset), augment with synthetic timestamps, train binary classification models on the "Response" column, and simulate streaming predictions.

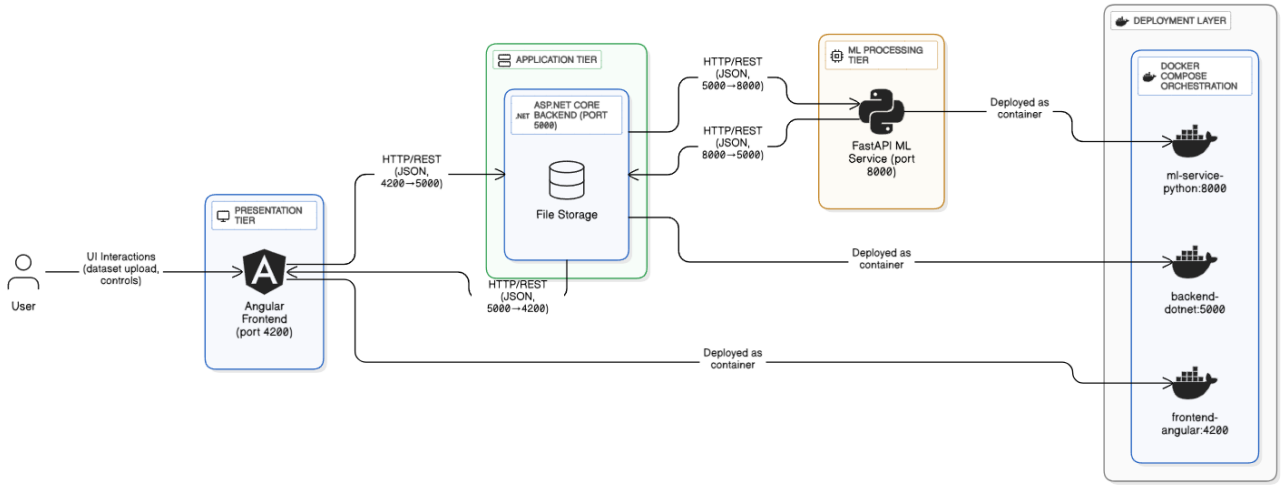
Key Features:

- Dataset upload and metadata extraction.
- Date range validation for training, testing, and simulation.
- Model training with evaluation metrics.
- Real-time simulation with live charts and tables.

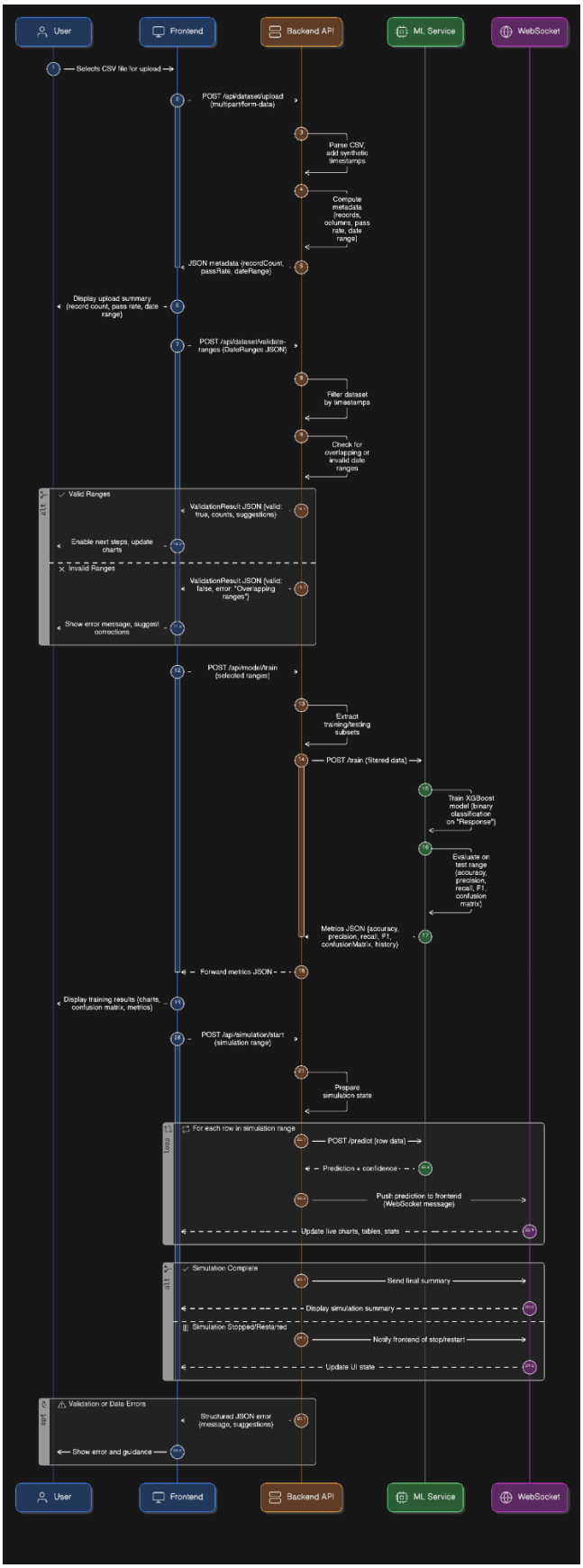
Technologies:

· Frontend: Angular 18+.
· Backend: ASP.NET Core 8.
· ML Service: Python 3.13 + FastAPI, with XGBoost
· Deployment: Docker + docker-compose.

System Architecture:



Dataflow Diagram:



API Contract and Payload Structure:

The backend exposes RESTful endpoints that handle dataset operations, date range validation, model training, and simulation. The ML service endpoints are called internally by the backend. All APIs use JSON payloads and follow standard HTTP status codes for responses.

1. Upload Dataset Endpoint

- Method: POST
- Path: /api/dataset/upload
- Description: Handles CSV upload, augmentation with synthetic timestamps, and metadata extraction.
- Request Payload (multipart/form-data):
    - file: The CSV file (binary).
- Response Payload (JSON on success, HTTP 200):

text

{

"fileName": "example.csv",

"totalRecords": 14704,

"totalColumns": 5,

"passRate": 70.0,

"StartTimestamp": "2021-01-01T00:00:00Z",

"EndTimestamp": "2021-12-31T23:59:59Z"

}

- Error Response (e.g., HTTP 400): { "error": "Invalid CSV format" }

2. Validate Date Ranges Endpoint

- Method: POST
- Path: /api/dataset/validate-ranges
- Description: Validates non-overlapping date ranges for training, testing, and simulation against the dataset.
- Request Payload (JSON):

text

{

"training": { "start": "2021-01-01T00:00:00Z", "end": "2021-06-30T23:59:59Z" },

"testing": { "start": "2021-07-01T00:00:00Z", "end": "2021-09-30T23:59:59Z" },

```
  "simulation": { "start": "2021-10-01T00:00:00Z", "end": "2021-12-31T23:59:59Z" }
}
```

- Response Payload (JSON on success, HTTP 200):

```
text
{
  "status": "Valid",
  "ranges": {
    "training": { "recordCount": 5000, "durationDays": 181 },
    "testing": { "recordCount": 3000, "durationDays": 92 },
    "simulation": { "recordCount": 6704, "durationDays": 92 }
  },
  "monthlyCounts": [ /* Array of monthly record counts for visualization */ ]
}
```

- Error Response (e.g., HTTP 400): { "status": "Invalid", "error": "Overlapping ranges" }

3. Train Model Endpoint

- Method: POST
- Path: /api/model/train
- Description: Triggers model training via the ML service using specified ranges.
- Request Payload (JSON):

```
text
{
  "trainStart": "2021-01-01T00:00:00Z",
  "trainEnd": "2021-06-30T23:59:59Z",
  "testStart": "2021-07-01T00:00:00Z",
  "testEnd": "2021-09-30T23:59:59Z"
}
```

- Response Payload (JSON on success, HTTP 200):

```
text
{
  "accuracy": 92.5,
  "precision": 91.0,
```

"recall": 93.0,

"f1Score": 92.0,

"trainingChartData": { /* Base64-encoded or data points for line chart */ },

"confusionMatrix": { "tp": 1000, "tn": 1500, "fp": 200, "fn": 100 }

}

- Error Response (e.g., HTTP 500): { "error": "Training failed" }

4. Start Simulation Endpoint

- Method: POST (for initiation; subsequent streaming via WebSocket or polling)
- Path: /api/simulation/start
- Description: Initiates row-by-row streaming inference for the simulation period.
- Request Payload (JSON):

text

{

"simulationStart": "2021-10-01T00:00:00Z",

"simulationEnd": "2021-12-31T23:59:59Z"

}

- Response: Streams predictions (e.g., via WebSocket messages):

text

{

"timestamp": "2021-10-01T00:00:00Z",

"sample_id": "ID123",

"prediction": "Pass",

"confidence": 95.5,

"sensor_data": {feature1,feature2,..}

}

- Final Response (on completion): { "totalPredictions": 6704, "passCount": 5000, "failCount": 1704, "averageConfidence": 88.2 }
- Error Response (e.g., HTTP 400): { "error": "Invalid simulation range" }

ML Service Internal Endpoints (Called by Backend)

- Train Model: POST /train (payload mirrors backend's train request; returns metrics).
- Predict: POST /predict (single row input; returns prediction and confidence).

Deployment Instructions:

1. Clone the GitHub repository.
2. Run **docker-compose up –-build** to start containers.
3. Access frontend at http://localhost:4200.
4. Ensure ports 4200, 5000, 8000 are available.

IntelliInspect Usage Guide :

This guide provides step-by-step instructions for utilizing the IntelliInspect application. Follow the sequence below to ensure a smooth workflow from dataset upload to real-time simulation.

1. Uploading the Dataset

- Navigate to the "Upload Dataset" screen.
- Use the drag-and-drop interface or the file chooser to select and upload a CSV file from the Kaggle dataset.
- Upon successful upload and processing, review the displayed metadata, including total records, columns, pass rate, and date range.

2. Configuring Date Ranges

- Proceed to the "Date Ranges" screen.
- Select non-overlapping time periods for training, testing, and simulation using the calendar-based date pickers.
- Click "Validate Ranges" to confirm the selections are sequential, non-overlapping, and within the dataset's timestamp bounds.
- Once validated, the system will display record counts and a timeline visualization for confirmation.

3. Performing Model Training

- Advance to the "Model Training" screen.
- Initiate the training process by clicking the "Train Model" button.
- After completion, examine the evaluation metrics (such as accuracy, precision, recall, and F1-score) along with associated charts, including training history and confusion matrix visualizations.

4. Running the Simulation

- Move to the "Simulation" screen.

- Begin the real-time simulation by clicking "Start Simulation."
- Monitor live updates, including prediction charts, statistics (e.g., pass/fail counts and average confidence), and a streaming table of results.
- If needed, stop the simulation at any time and restart as required. The simulation will auto-complete upon processing all records in the designated period.

Login Details:
Email : abb@gmail.com

Password: Abb@1234