IntelliInspect Hackathon Specification

Title:

IntelliInspect: Real-Time Predictive Quality Control with Kaggle Instrumentation Data

---

Objective

Design and implement a full-stack AI-powered application that enables real-time quality control prediction using Kaggle Production Line sensor data. Participants will build an end-to-end solution with an Angular frontend, .NET backend, and a Python ML service deployed in a Dockerized environment.

Technologies

Layer	Technology
Frontend	Angular (v18+)
Backend API	ASP.NET Core 8
ML Service	Python 3.13 + FastAPI
ML Framework	XGBoost / LightGBM / scikit-learn
Deployment	Docker + docker-compose

System Architecture

User interacts via the Angular UI. Backend services (.NET Core API) coordinate with the Python ML microservice for training and real-time prediction. All components are containerized and orchestrated via Docker Compose.

Dataset

- https://www.kaggle.com/c/bosch-production-line-performance/data
- Dataset does not contain timestamps; each row must be synthetically augmented with a timestamp field
- Synthetic timestamps must follow second-level granularity (one second apart)
- Augmented timestamps must be used to support:
    - Time-based train/test/simulation segmentation
    - Real-time streaming inference loop
- Time-based filtering using synthetic_timestamp column
- Target: Response column (binary classification)

Screen 1: Upload Dataset

Objective

Allow the user to upload the mandatory Kaggle dataset (CSV format), augment it with a synthetic timestamp column if not already present, and extract basic metadata for downstream operations like training, testing, and simulation.

Functional Requirements

- The user must upload the Kaggle CSV dataset using a drag-and-drop interface or file chooser.
- On upload:
    - Validate that the file is in .csv format.
    - Trigger backend API to:
        - Parse the file.
        - Augment dataset with synthetic timestamps (1-second granularity) if not already present.
        - Persist the parsed dataset temporarily.
        - Calculate and return:
            - Total number of records
            - Number of columns

        - Pass rate: % of rows with Response = 1
        - Earliest and latest synthetic timestamp (date range)
- Display extracted metadata back in the UI.
- Proceed to next step only after successful file analysis and confirmation.

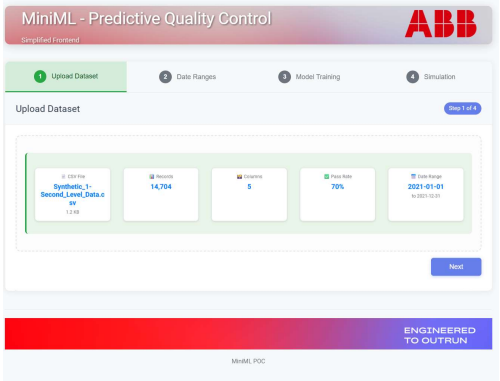
UI Elements

- Step progress indicator (highlighted as “Step 1 of 4”)
- Tab Navigation:
    - Upload Dataset (active)
    - Date Ranges
    - Model Training
    - Simulation
- Upload Card:
    - Drag and drop area with dashed border
    - “Choose File” button (enabled if no file selected)
    - CSV icon placeholder
    - Instruction text: “Click to select a CSV file or drag and drop”
- Post-upload Info Summary Card (appears after file processing):
    - File name (clickable link to original CSV)
    - Total records (e.g., “14,704”)
    - Total columns (e.g., “5”)
    - Pass Rate (e.g., “70%”)
    - Date Range (e.g., “2021-01-01 to 2021-12-31”)
- Next Button:
    - Disabled until file is uploaded and processed
    - Enabled once metadata is shown

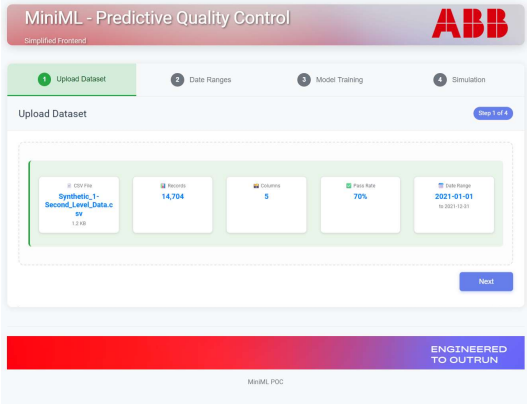
Backend Responsibilities (On File Upload)

- Parse CSV and validate schema (must include Response column)
- Augment rows with synthetic timestamp column if missing
  e.g., start timestamp = 2021-01-01 00:00:00, increment 1 second per row
- Generate metadata summary:
    - Count of rows
    - Count of columns
    - Pass rate (% of Response = 1)
    - First and last synthetic timestamp
- Return metadata to frontend for rendering
- Store preprocessed dataset for next steps (training, simulation)

Initial screen showing the empty upload card with the “Choose File” button.



Post-upload screen showing parsed file summary with metadata and enabled Next button.



Screen 2: Date Ranges

Objective

Allow the user to define time-based splits on the dataset for training, testing, and simulation. The application must validate that the selected date ranges are sequential, non-overlapping, and within the uploaded dataset's available timestamp window.

Functional Requirements

- The user must define three non-overlapping date ranges:
    1. Training Period
    2. Testing Period
    3. Simulation Period
- All date ranges must be selected using calendar-based date pickers.
- Date validation rules:
    - Each start date must be earlier than or equal to the corresponding end date.
    - The testing period must begin after the training period ends.
    - The simulation period must begin after the testing period ends.
    - All selected dates must fall within the dataset's available synthetic timestamp range.
- Backend validates the ranges and returns:
    - Status: "Valid" or "Invalid"
    - Count of records within each period
    - Breakdown summary to support visualization
- UI reflects the selection with:
    - Range cards showing number of days per period
    - Timeline summary bar chart (color-coded):
        - Green: Training period
        - Orange: Testing period
        - Blue: Simulation period
- On successful validation, the "Next" button becomes active.

UI Elements

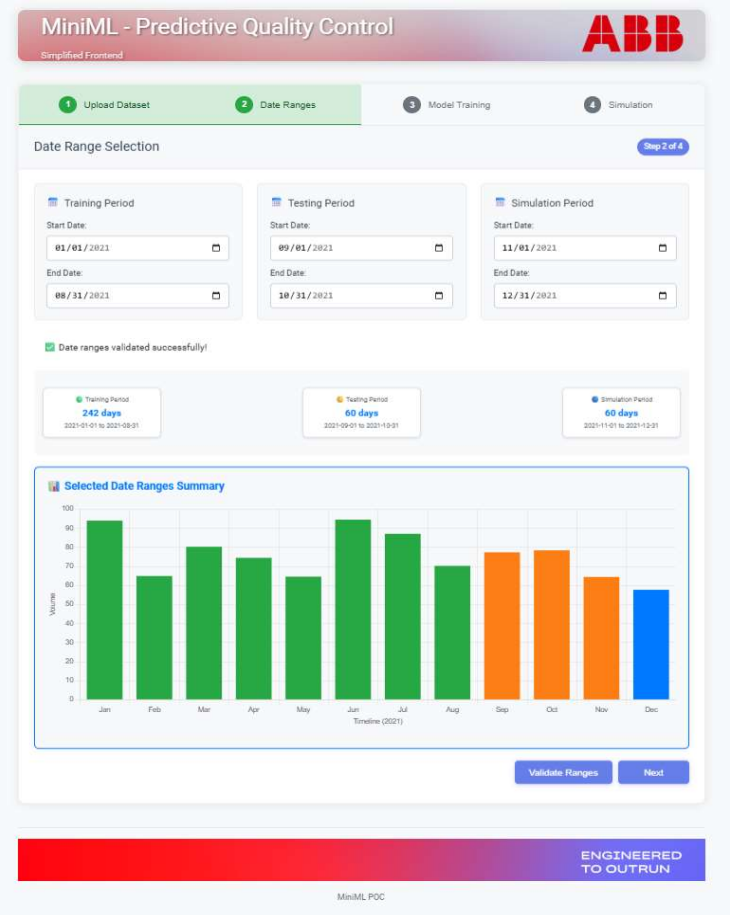
- Step progress indicator (Step 2 of 4)
- Navigation bar:
    - Upload Dataset
    - Date Ranges (active)
    - Model Training
    - Simulation
- Three Period Cards:
    - Each has:
        - Title (Training Period / Testing Period / Simulation Period)

        - Start date input (calendar picker)
        - End date input (calendar picker)
- Validation Message:
    - Green tick icon and success message: “Date ranges validated successfully!” (if valid)
    - Red icon and warning message if invalid
- Range Summary Cards (post-validation):
    - Training Period: duration and range
    - Testing Period: duration and range
    - Simulation Period: duration and range
- Timeline Bar Chart:
    - Bar chart showing data volume per month
    - Bars color-coded according to their associated range (training/testing/simulation)
- Action Buttons:
    - “Validate Ranges” button triggers backend validation
    - “Next” button is disabled until validation passes

Backend Responsibilities

- Accept selected date ranges from frontend
- Validate date logic and integrity against uploaded dataset timestamps
- Count records per date range
- Send:
    - Success/failure status
    - Duration in days per range
    - Monthly record counts for visualization

Date Range Selection Interface with All Periods Defined and Validated



Screen 3: Model Training & Evaluation

Purpose

This screen enables the participant to trigger machine learning model training using the training and testing datasets configured in the previous step. It also displays comprehensive evaluation metrics and performance visualizations post-training.

Key Functionalities

- Trigger model training using configured date ranges.
- Visualize the output metrics of training and testing (Accuracy, Precision, Recall, F1-Score).
- View additional insights via training loss/accuracy plots and confusion matrix-style donut chart.

Expected Behavior

- On clicking "Train Model":
    - A REST API call is made to the Python ML service (FastAPI) which:
        - Extracts the training and test data from backend storage (or in-memory object)
        - Trains a classification model (e.g., using XGBoost or LightGBM)
        - Evaluates the model on the test window
        - Stores training metrics and model artifacts
        - Returns the evaluation results and visualizations
- On successful training, display model performance summary with:
    - Accuracy (%)
    - Precision (%)
    - Recall (%)
    - F1-Score (%)
    - Training metrics line chart (Accuracy & Loss)
    - Confusion matrix chart (True/False Positives/Negatives)

UI Elements

Component	Type	Description
Train Model Button	Button	Triggers model training via backend API
Metrics Cards	Summary Tile	Accuracy, Precision, Recall, F1-Score
Training Chart	Line Chart	Training Accuracy vs Training Loss (epoch-wise or batch-wise)
Model Performance Chart	Donut Chart	Confusion matrix visualization: TP, TN, FP, FN
Status Message	Label/Text	Displays training status (e.g., "Model Trained Successfully")
Next Button	Button	Enables once training is completed

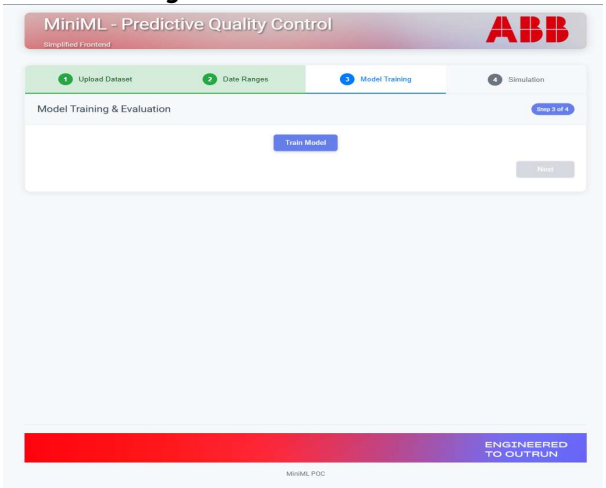
Technical Integration

- API Endpoint: POST /train-model
- Payload: {
  "trainStart": "",
  "trainEnd": "",
  "testStart": "",
  "testEnd": ""
  }
- Backend (.NET Core) forwards request to ML service
- ML Service uses framework (XGBoost/LightGBM/scikit-learn) to train and evaluate
- Response includes metric values and optionally base64-encoded chart data

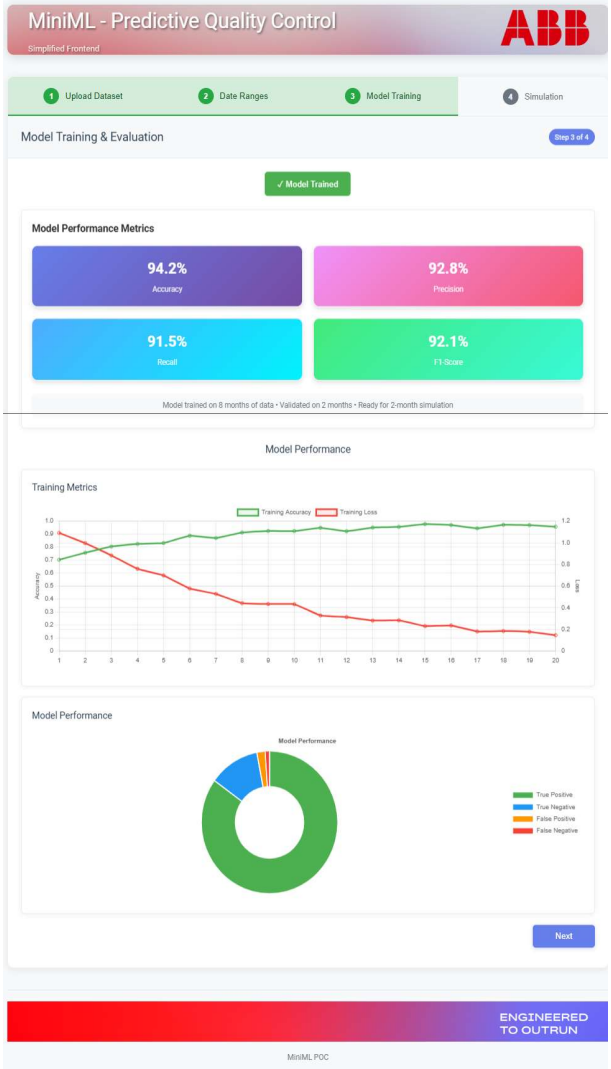
---

Model Training Screen (Before & After Training Execution)

Before Training



After Training



Screen 4: Real-Time Prediction Simulation

Objective

Enable the user to simulate model inference on historical test data at second-level granularity, mimicking a real-time production environment. The simulation must emit one row per second and reflect live quality predictions and confidence in visual and tabular formats.

Functional Requirements

- The user initiates the simulation by clicking “Start Simulation”.
- Records from the selected Simulation Period (from Screen 2) are streamed row-by-row.
- Each row is inferred by the trained model and the result is displayed in real-time.
- The following components must update live:
    - Line chart for Quality Scores over time.
    - Donut chart for prediction confidence breakdown.
    - Counters for total/pass/fail/average confidence.
    - Live table for each sample’s timestamp, ID, prediction, and parameters.
- Once all records are processed:
    - The simulation auto-completes.
    - The button changes to “Restart Simulation”.
    - A success message “✓ Simulation completed” is shown.

UI Elements

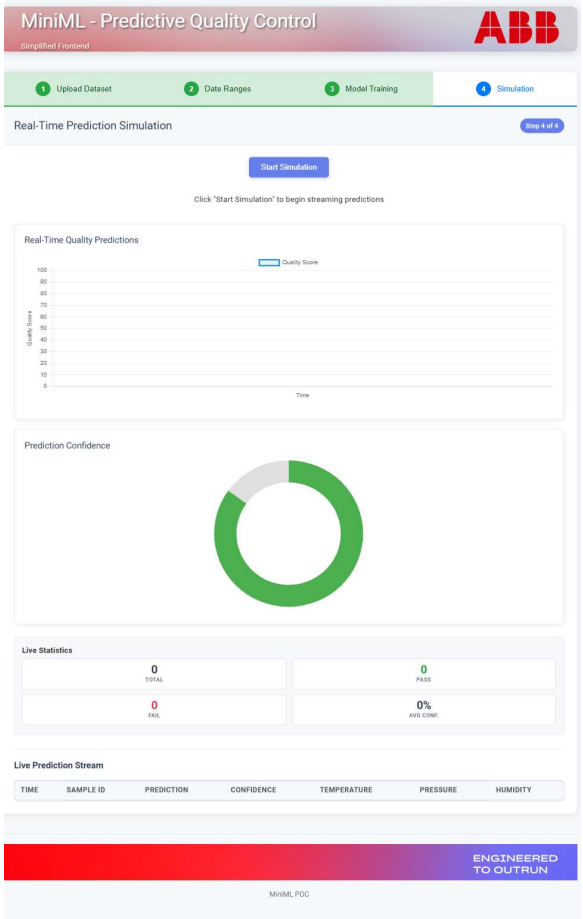
- Step progress indicator (Step 4 of 4)
- Navigation bar:
    - Upload Dataset
    - Date Ranges
    - Model Training
    - Simulation (active)
- Simulation Control:
    - “Start Simulation” (initial state)
    - “Restart Simulation” (post-completion)
- Simulation Status:
    - Message: “✓ Simulation completed” (shown when simulation ends)
- Charts Section:
    - Line Chart: Real-Time Quality Predictions
        - X-axis: Time (auto-updating)
        - Y-axis: Quality Score (0–100)
    - Donut Chart: Prediction Confidence Distribution
        - Green = Pass
        - Red = Fail
- Statistics Panel:
    - Total Predictions
    - Pass Count
    - Fail Count
    - Average Confidence (%)
- Live Prediction Stream Table:
    - Columns:
        - Time
        - Sample ID
        - Prediction (Pass/Fail)
        - Confidence (%)
        - Temperature (°C)
        - Pressure (hPa)
        - Humidity (%)
    - Rows appear 1/sec

- Scrollable if overflow

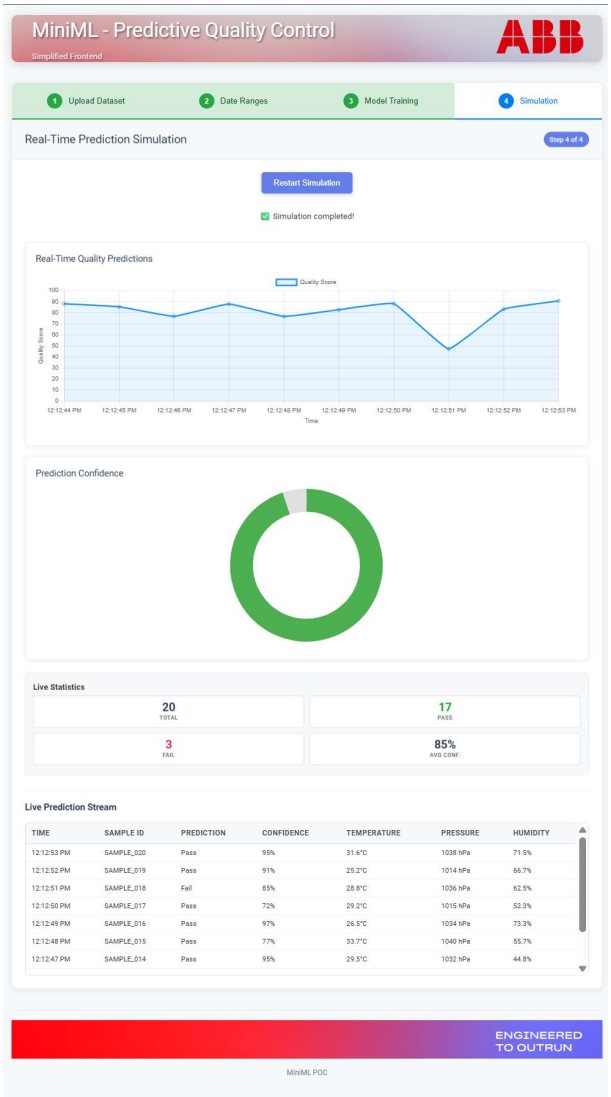
Backend Responsibilities

- Accept Simulation Period range and trigger row-by-row streaming.
- For each row:
  - Perform model inference
  - Return:
    - Timestamp
    - Sample ID
    - Prediction label
    - Confidence score
    - Parameter values (temperature, pressure, humidity)
- Maintain simulation state and stop on completion.
- Return final statistics:
  - Total rows processed
  - Pass/Fail breakdown
  - Average confidence

Real-Time Prediction Simulation (Before Start)



Real-Time Prediction Simulation (Simulation Complete)



Dockerized Deployment (Mandatory)

- Entire application must be runnable via docker-compose up
- Containers:
    - frontend-angular
    - backend-dotnet
    - ml-service-python

Submission Checklist

✅ **Must Have**

- Design document (PDF or Markdown) including:
    - System architecture diagram
    - Data flow diagram
    - API contract and payload structure
- GitHub repository containing:
    - Angular frontend code

    - .NET backend code
    - Python ML service code
- README with:
    - Setup and deployment instructions (Docker-based)
    - Usage guide
- docker-compose.yaml for full system boot-up
- 3-minute voice-over demo video showing:
    - Dataset upload
    - Configuration of training, testing, simulation windows
    - Model training results
    - Streaming real-time predictions in the UI

✳️ **Good to Have**

- Feature importance visualization (SHAP or model-native)
- Live chart for streaming predictions (e.g., confidence over time)
- Retry/Resume simulation logic
- Model versioning and metadata display

🌼 **Nice to Have**

- Authenticated access (basic login)
- Upload history and dataset summary screen
- Admin dashboard with usage stats
- Performance optimization for large CSVs (chunking)
- Theme toggle (light/dark mode)
- Export logs of predictions in CSV or JSON format

Evaluation Criteria

Area	**Description**
Functionality	End-to-end flow works correctly with full dataset
Code Quality	Modular, clean, with error handling
UI/UX	Intuitive, responsive, professional feel
Real-Time Sim	Streaming predictions accurately & progressively
Deployment	Fully dockerized, zero local setup
Documentation	Clear, complete, runnable by third party
Demo Video	Explains each part of the flow within 3 minutes
