PROJECT STRUCTURE & WORKING DOCUMENTATION

RUN SURE

Intelligent Readiness Assessment for CI Pipeline Execution

What is this project?

The RunSure: Intelligent Readiness Assessment for CI Pipeline Execution is a hybrid validation system that analyzes CI pipeline configuration files (JSON format) before execution.

It combines:

- Rule-based validation (syntax, schema, structural checks)
- Machine learning-based risk prediction
- A REST API backend
- A user-friendly frontend interface

The system outputs a **risk score (0–100)** and a **risk level (LOW / MEDIUM / HIGH)** along with recommended action.

TOOLS AND DEPENDENCIES :

Core Language
- Python 3.13

Machine Learning
- scikit-learn
- NumPy
- Pandas

Backend
- FastAPI
- Uvicorn

Frontend
- Streamlit
- Requests

Validation
- JSON
- JSON Schema

Development Tools
- Kaggle Notebook (Model Training)
- VS Code
- Command Prompt / Terminal

INSTALL & RUN GUIDE (GITHUB USER GUIDE SECTION)

Step 1: Clone Repository

git clone <repo-url>
cd ci-pipeline-intelligent-validator

Step 2: Install Dependencies

pip install -r requirements.txt
Or manually:
pip install fastapi uvicorn streamlit scikit-learn pandas numpy requests

---

Step 3: Start Backend

uvicorn backend:app --reload
Access API docs at:
http://127.0.0.1:8000/docs

Step 4: Start Frontend

In another terminal:
streamlit run frontend.py
Upload a JSON file and analyze.

PROJECT REP0 / DIRECTORY STRUCTURE

```
CI-PIPELINE-INTELLIGENT-VALIDATOR/
|
 ├── backend.py
 ├── frontend.py
 ├── ci_validator_cli.py
 ├── demo.py
 ├── integration_examples.py
 ├── config.json
 ├── pipeline_schema.json
 ├── pipeline.py
 ├── project_summary.json
 ├── README.md
 ├── requirements.txt
|
 ├── models/
|    ├── best_model.pkl
|    ├── scaler.pkl
|   └── pipeline_schema.json
|
 ├── notebooks/
|    ├── brain_core.py
|    ├── ci-pipeline-intelligent-validator.ipynb
|    ├── pipeline_schema.json
|    ├── sample.json
|   └── test_brain.py
|
 ├── sample_bin_dump/
|    ├── lowtest.json
|    ├── medtest.json
|   └── hightest.json
|
 ├── results/
|
 ├── .streamlit/
|   └── config.toml
|
 ├── __init__.py
└── __pycache__/
```

STRUCTURE EXPLANATION (WHAT EACH PART DOES)

Core Application Files (Root Directory)

backend.py

Implements the FastAPI backend service.

- Exposes the REST endpoint:
- POST /analyze
- Receives uploaded JSON files.
- Calls the validation engine from brain_core.py.
- Returns structured risk results as JSON.
- Deletes temporary uploaded files after processing.

This file acts as the bridge between the frontend and the validation engine.

---

frontend.py

Implements the Streamlit user interface.

- Allows users to upload JSON pipeline files.
- Sends uploaded files to the backend using HTTP requests.
- Displays:
    - Risk level (LOW / MEDIUM / HIGH)
    - Risk score
    - Summary of validation results
    - Expandable detailed JSON output
- Includes a progress bar during analysis.

This file provides the user-facing layer of the system.

---

ci_validator_cli.py

Provides command-line interface access to the validator.
Allows usage such as:
python ci_validator_cli.py validate <directory>
This enables automation and CI/CD integration without the UI.

---

demo.py

A console-based demonstration script.
Used to:

- Showcase validation functionality.
- Demonstrate single-file and batch validation.
- Provide example usage for testing.

---

integration_examples.py

Contains examples of how to integrate the validator with CI/CD systems such as:

- GitHub Actions
- GitLab CI
- Jenkins

---

config.json

Stores configurable project settings such as:

- Model directory path
- Reports directory path
- Base project directory

This allows centralized configuration management.

---

pipeline_schema.json (Root Level)

Defines the required structure of a valid CI pipeline JSON file.
Includes:

- Required fields
- Data types
- Allowed values
- Structural constraints

Models Directory

```
models/
├── best_model.pkl
├── scaler.pkl
└── pipeline_schema.json
```

This directory stores all machine learning artifacts.

best_model.pkl
Pre-trained Random Forest model trained in Kaggle.
Used for:

- Predicting probability of high-risk pipeline execution.

scaler.pkl
StandardScaler object used during model training.
Ensures feature normalization during inference.

pipeline_schema.json (Model-Level)
Schema version used during validation.

notebooks Directory

notebooks/
├── brain_core.py
├── ci-pipeline-intelligent-validator.ipynb
├── sample.json
├── test_brain.py
This directory contains the core validation logic and development artifacts.

---

brain_core.py (MOST IMPORTANT FILE)

This file implements the entire validation engine.
It includes:

- ValidationResult class
- JSONSyntaxValidator
- JSONSchemaValidator
- JSONContentAnalyzer
- FeatureExtractor
- MLRiskPredictor
- RiskScoringEngine
- CIPipelineValidator

This is the "brain" of the project.
All intelligent processing happens here.

---

ci-pipeline-intelligent-validator.ipynb

Original Kaggle notebook used for:

- Dataset creation
- Feature engineering
- ML training
- Model evaluation

Used only during development phase.

---

test_brain.py

Used for local testing of the validation engine without the frontend.

---

sample.json

Sample input file used for testing.

sample_bin_dump Directory

sample_bin_dump/
├── lowtest.json
├── medtest.json
└── hightest.json
Contains predefined test cases:

- lowtest.json → Valid configuration
- medtest.json → Medium risk configuration
- hightest.json → Invalid / high-risk configuration

Used during demonstration and UI testing.

---

results Directory

Stores generated validation reports if report generation is enabled.
Can contain:

- JSON reports
- Text summaries

---

streamlit Directory

.streamlit/
└── config.toml
Used to configure Streamlit server settings.
Includes:

- Maximum upload size
- Maximum message size

Example:
[server]
maxUploadSize = 250
maxMessageSize = 500

FUNTIONING / WORKING (INTERNAL LOGIC FLOW)

This section explains how internal components operate.

Validation Engine Workflow (brain_core.py)

When validate_file() is called, the following steps occur:

1. Syntax Validation
    - File existence check
    - JSON parsing
    - Encoding validation
2. Schema Validation
    - Required fields check
    - Field type validation
    - Structure verification
3. Content Analysis
    - Structural depth
    - Null value count
    - Suspicious pattern detection
4. Feature Extraction
    - File size
    - Key counts
    - Structural metrics
    - Numerical feature vector creation
5. ML Risk Prediction
    - Load model
    - Apply scaler
    - Predict probability of high risk
6. Risk Scoring Engine
    - Combine rule-based score + ML probability
    - Generate final risk score
    - Assign risk level
7. Final Output
    - validation_passed
    - risk_score
    - risk_level
    - recommended_action

FLOW OF COMPONENTS :

Complete flow:
User
↓
Streamlit Frontend (frontend.py)
↓ HTTP POST
FastAPI Backend (backend.py)
↓
CIPipelineValidator (brain_core.py)
↓
Risk Score Calculation
↓
JSON Result
↓
Frontend Display
Each component has a single responsibility:

- Frontend → Presentation
- Backend → Routing
- Brain → Logic
- Models → Prediction
- Config → Settings

This follows clean modular architecture principles.

RESULT PROCESS AND FLOW :

Final result is computed by combining:
Rule-Based Score:

- Syntax
- Schema
- Content checks

ML Score:

- Probability of high-risk classification

Final Formula (simplified):
Final Score =
(Syntax × weight) +
(Schema × weight) +
(Content × weight) +
(ML Probability × weight)
Risk Level Assignment:

- 0–30 → LOW
- 31–60 → MEDIUM
- 61–100 → HIGH

CONCLUSION

The project implements a hybrid validation architecture combining deterministic rule-based verification with probabilistic machine learning inference. The modular separation of frontend, backend, and validation engine ensures scalability, maintainability, and extensibility.
The system prevents high-risk CI configurations from reaching execution, thereby reducing CI/CD failures and improving deployment reliability.