***Dijikstra - GPT***

---

# Technical Design Document (TDD)

## End-to-End ML Classifier Platform with CI/CD

**Author:**Neha Amin
**Audience:** CTO, Principal Engineers, Platform & ML Leadership
**Status:** Work In Progress
**Revision:** v1.0

---

## 1. Executive Summary

This document proposes an upgrade of the existing project into a **production-grade end-to-end machine learning system**:

> **Data ingestion → Model training → Model registry → REST-based inference → Monitoring → CI/CD automation**

The goal is to demonstrate **ML operational readiness**, and **deployment automation**, using **ML infra** while adhering to MLOps best practices.

The system intentionally separates *model development velocity* from *production stability* by decoupling training pipelines from inference deployment.

This system is intentionally designed to be:

- Deterministic and reproducible
- Auditable and testable
- CI/CD-driven
- Cloud-deployable with low operational overhead

---

# 2. Engineering Objectives

## 2.1 Engineering Objectives

- Enforce **separation of concerns** across ML lifecycle
- Apply **DevOps principles to ML workflows (MLOps)**
- Enable **automated testing and deployment**
- Ensure low-latency and observable inference

---

# 3. Scope Definition

## In Scope

- ☐ Supervised classification model
- ☐ Offline training + online inference
- ☐ Model versioning
- ☑ ~~REST API for predictions~~
- ☑ ~~CI/CD automation~~
- ☑ ~~Basic monitoring and metrics~~
- ☑ ~~Dockerized workloads~~
- ☑ ~~Free-tier cloud deployment~~

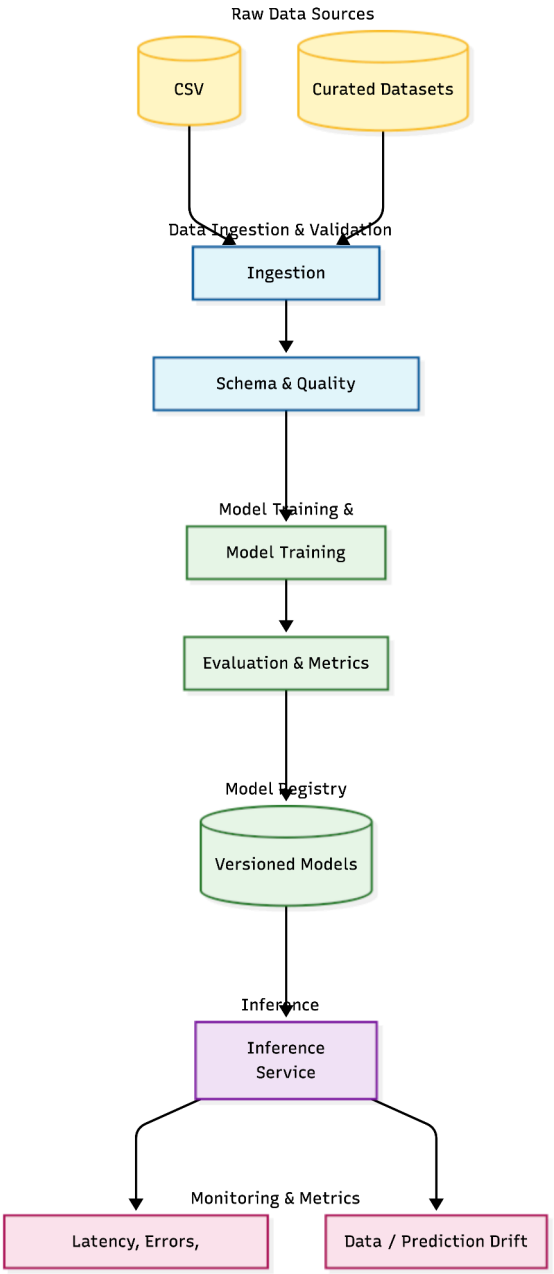
## Out of Scope

- Distributed training
- Real-time feature stores
- Online learning
- Advanced monitoring (e.g., drift detection)
- Enterprise authentication

---

## 4. High-Level Architecture

## 4.1 Logical Architecture




---

### 4.2 Tech - Stack

| Component | Technology |
|---|---|
| Training | Python + scikit-learn |
| Registry | Filesystem / MLflow (local) |
| Inference | FastAPI + Uvicorn |
| Containerization | Docker |
| CI/CD | GitHub Actions |
| Infra | Free-tier Cloud (Render/Fly.io/AWS EC2) |
| Monitoring | Prometheus-style metrics |

---

# 5. Detailed Component Design

---

## 5.1 Data Ingestion Layer

**Purpose:**
Ensure clean, validated, reproducible input data.

**Responsibilities:**

- Load raw datasets
- Validate schema & implies constraints
- Handle missing/invalid labels
- Split data into training and test sets
- Persist processed data artifacts

**Output Artifacts:**

- `train.csv`
- `test.csv`

**Key Engineering Decisions:**

- Schema validation prevents silent model degradation
- Deterministic data splits enable reproducibility

---

## 5.2 Model Training & Evaluation

**Purpose:**
Train a deterministic, interpretable classifier with measurable quality.

**Responsibilities:**

- Feature preprocessing
- Model training
- Metric computation (accuracy, precision, recall)
- Model serialization
- Metadata generation

**Model Choice:**

- Logistic Regression / Random Forest
  (Chosen for interpretability and deterministic behavior)

**Artifacts Produced:**

- Serialized model (`.joblib`)
- Metrics summary
- Dataset hash

---

## 5.3 Model Registry

**Purpose:**
Enable versioned and auditable model management.

**Design:**

- File-based registry
- Semantic versioning (`classifier_v1`, `classifier_v2`)
- Metadata tracking

**Metadata Stored:**

- Model version
- Training timestamp
- Dataset checksum
- Performance metrics

**Rationale:**

- Lightweight registry sufficient for small-scale deployment
- MLflow compatible for future expansion

---

## 5.4 Inference Service

**Purpose:**
Expose predictions via a stable REST API.

**Framework:** FastAPI

**Endpoints:**

**Health Check**
GET /health

**Prediction**
POST /predict

**Response Includes:**

- Predicted class
- Confidence score
- Model version

**Non-Functional Requirements:**

- Latency < 100ms (local)
- Input validation
- Stateless service
- Container-ready

---

### 5.5 Monitoring & Observability

**Purpose:**
Provide operational visibility into model behavior.

**Metrics Captured:**

- Request count
- Inference latency
- Error rate

**Endpoint:**

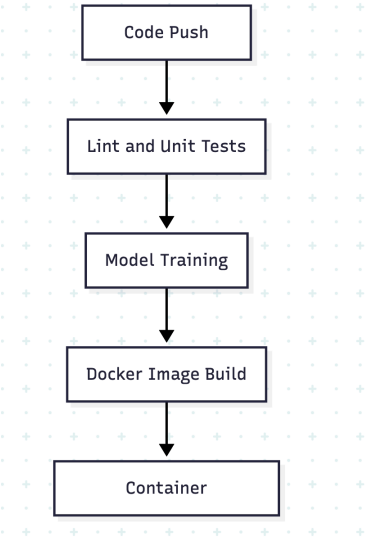
GET /metrics

**Design Philosophy:**

- Lightweight monitoring
- Prometheus-compatible
- Extendable for future drift detection

---

# 6. CI/CD Pipeline Design




---

### 6.2 CI/CD Stages

**Stage 1: Validation**

- Python linting
- Unit tests for training & inference
- Schema checks

**Stage 2: Training**

- Execute training pipeline
- Validate metrics thresholds
- Register new model version

**Stage 3: Build**

- Build Docker image
- Tag with commit SHA

**Stage 4: Deployment**

- Push container to cloud
- Restart inference service

**Automation Tool:** GitHub Actions

---

# 7. Containerization Strategy

## Docker Principles Applied

- Minimal base image
- Deterministic builds
- Immutable artifacts
- Clear runtime separation

## Containers

| Container | Responsibility |
|---|---|
| Training | Offline model training |
| Inference | Online prediction |

---

# 8. Infrastructure (Minimal & Reproducible)

### Infrastructure as Code

- Terraform used for provisioning
- Enables repeatable deployments

### Deployment Targets

- Render / Fly.io / AWS Free Tier
- Stateless inference service

---

# 9. Security & Compliance Considerations

- No secrets committed to source control
- Environment-based configuration
- API input validation
- Minimal exposed surface area

---

# 10. Testing Strategy

### Test Types

- Unit tests (training, inference)
- API contract tests
- Health check validation

### CI Enforcement

- Pipeline fails on test failure
- No manual deployment allowed

---

## 11. Risks & Mitigations

| Risk | Mitigation |
|---|---|
| Model drift | Versioned retraining |
| API misuse | Input validation |
| Cost overrun | Free tier + limits |
| Reproducibility | Deterministic seeds |

---

## 12. Success Metrics

- Fully automated CI/CD
- Sub-100ms inference latency
- Reproducible model builds
- Clean rollback via model versioning
- Deployment without manual steps

---

---

## 14. Future Enhancements (Roadmap)

- Drift detection
- Feature store integration
- Canary deployments
- Auth-protected inference
- Centralized model registry

---

## 15. Conclusion

This upgrade transforms the project from a **demo application** into a **production-ready ML system** aligned with modern **MLOps best practices**.
It demonstrates strong fundamentals in **ML engineering, DevOps, and system design**, making it suitable for real-world deployment and leadership review.