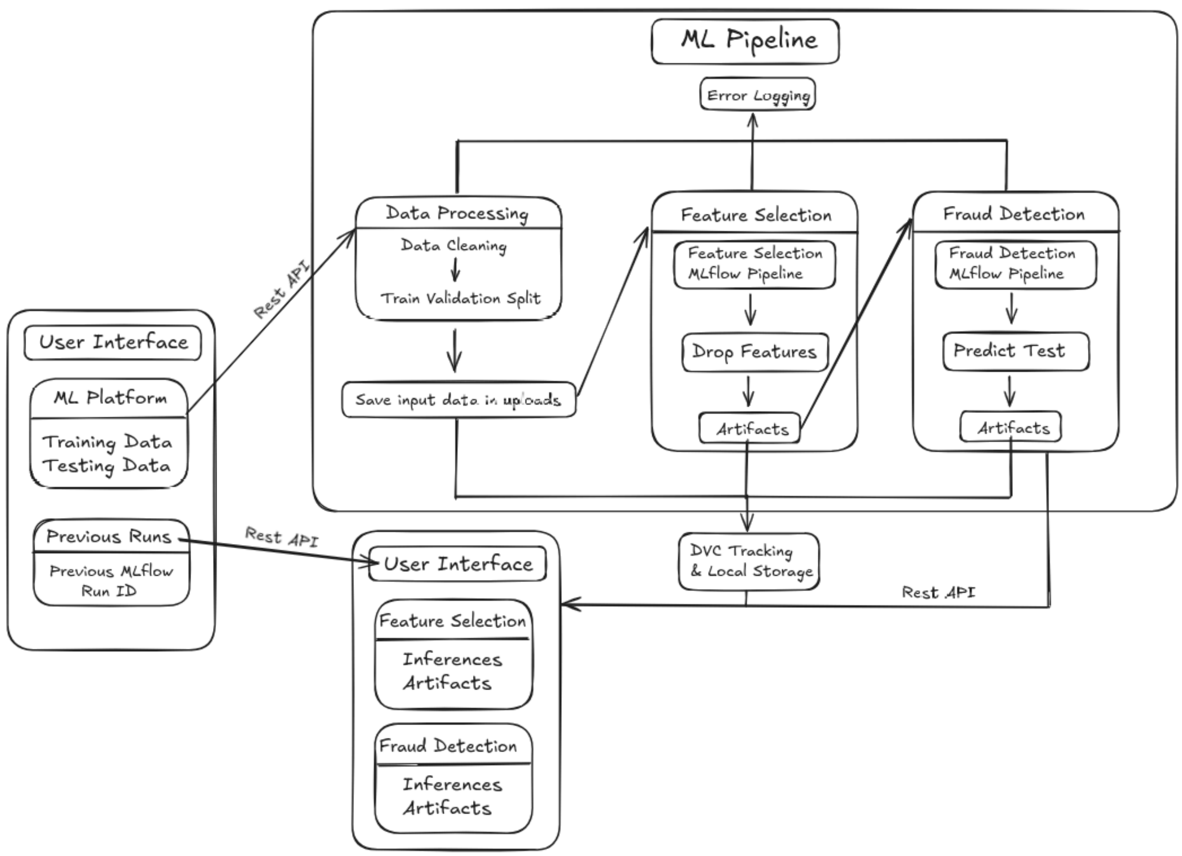
High-Level Design

1. Overview

This MLOps platform enables users to upload datasets, train machine learning models, track experiments, manage artifacts, and view results through a user-friendly web interface. The system is modular, scalable, and designed for reproducibility and collaboration.

2. Flow of the Program



3. Components:

Data Processing:

- Data Cleaning using KNN imputer for Missing Data

- Train Validation Split for Training Fraud Detection Model

Feature Selection Model:

- A Deep Learning Model which selects the most important features and removes the ones which arent.
- Saved the artifacts and inferences

Fraud Detection Model:

- Using Inferences from Feature Selection it runs a model on mlflow for detecting frauds

DVC Tracking:

- Artifacts from every component are saved using DVC.

User Interface:

- UI has 5 Routes namely, ML Platform, Previous Runs, Model Specs, Feature Selection and Test Results
- Each Route runs an api call to the backend for accessing data and displaying results.

4. Design Choices & Rationale

Frontend

- **React** was chosen for its component-based architecture, facilitating the development of complex, dynamic dashboards and forms.
- RESTful API communication keeps the frontend decoupled from backend logic, enabling independent development and scaling.

Backend

- **Node.js** offers non-blocking I/O, making it well-suited for handling concurrent requests and long-running ML jobs.
- **Express** provides a simple, extensible routing layer for API endpoints.

ML Pipelines

- **Python** is the de facto language for machine learning, offering rich libraries (scikit-learn, pandas, TensorFlow).
- Running pipelines as subprocesses or jobs allows for language separation and easy scaling.

Experiment Tracking

- **MLflow** is an industry standard for tracking experiments, storing metrics, and managing models.
- This enables reproducibility, auditability, and comparison of different runs.

Data Versioning

- **DVC** integrates with Git to version control large datasets and models, supporting collaborative workflows and rollback.

Database

- **MongoDB** is chosen for its flexible schema, which accommodates evolving ML metadata and experiment logs.

Infrastructure

- **Docker Compose** ensures all components run in isolated, reproducible environments, simplifying setup for both development and production.

5. Scalability & Extensibility

- **Modular Design**: Each service (frontend, backend, ML, database, MLflow, DVC) can be scaled independently.
- **API-First Approach**: Enables integration with other tools or automation scripts.
- **Artifact and Data Versioning**: Supports collaborative and reproducible research.

6. User Experience

- **Intuitive UI** for non-technical users to upload data, launch runs, and view results.
- **Clear feedback** on job status, errors, and results.
- **Downloadable artifacts** (e.g., confusion matrix, predictions).

7. Summary Table

Component	Technology	Rationale
Frontend	React	Dynamic UI, component-based, REST integration
Backend	Node.js/Express	Async I/O, easy API routing
ML Pipeline	Python	ML ecosystem, subprocess separation
Tracking	MLflow	Experiment tracking, artifact management
Data Version	DVC	Git-like data/model versioning
Database	MongoDB	Flexible schema, easy scaling
Infra	Docker Compose	Reproducibility, easy orchestration