Enhancing Urban Mobility: A Real-Time Anomaly Detection System for the Chicago Train Network

Gaurav Khanal

2025-08-23

1 Introduction

1.1 Objective

To design, build, and deploy an end-to-end MLOps system that monitors live Chicago Transit Authority (CTA) train data, detects anomalies in real-time using machine learning with rigorous confidence bounds, and visualizes the system's status on a public-facing dashboard.

1.2 Impact

This project is impactful because it addresses a critical challenge in urban mobility: ensuring public transportation is reliable and efficient. For millions of commuters, unexpected delays and disruptions can significantly impact their day. While transit agencies report major outages, they often lack the tools to detect subtle, cascading, or emerging issues in real-time.

By building a real-time anomaly detection system, this project delivers tangible impacts:

1. **For Passengers**: It can power next-generation transit apps that provide smarter alerts. Instead of just saying a train is "delayed," it could warn users about an "unusual slowdown" on their line before the official delay is announced, allowing them to reroute and save time.
2. **For Transit Operators**: A live dashboard highlighting anomalous behavior acts as an early warning system. Operators can see a slowdown developing on a map and proactively investigate the cause—be it a signal issue, a mechanical problem, or overcrowding—potentially preventing a minor issue from becoming a major system-wide failure.

3. **For City Planners**: The historical data on anomalies provides a rich dataset for identifying systemic bottlenecks. Planners can use this information to optimize schedules, allocate resources more effectively, and make informed decisions about infrastructure upgrades.

In essence, this project moves beyond simple tracking to provide actionable intelligence, making the transit system more resilient and trustworthy.

1.3 Anomaly Detection

Anomaly detection is a machine learning technique used to identify rare items, events, or observations that deviate significantly from the majority of the data. These deviations are often called anomalies, outliers, exceptions, or contaminants.

At its core, the process involves establishing a clear definition of "normal" behavior based on historical data. The system then monitors new, incoming data and flags any points that do not conform to this established norm.

For example, in our transit system:

- **Normal Behavior**: Trains slow down as they approach a station, travel at a consistent speed between stations during off-peak hours, and experience slightly longer travel times during rush hour.
- **Anomalous Behavior**: A train stopping for an extended period between stations, moving significantly slower than usual for that time of day, or deviating from its designated route.

Anomaly detection is a critical tool in many industries, including cybersecurity (detecting network intrusions), finance (flagging fraudulent transactions), and manufacturing (predicting equipment failures).

2 Related Works

The task of detecting anomalies in transportation systems has been an active area of research, evolving from statistical methods to complex deep learning architectures.

Early research often focused on statistical and classical time-series models. For instance, methods like ARIMA (Autoregressive Integrated Moving Average) were used to forecast passenger flow or travel times, with significant deviations from the forecast being flagged as anomalies (Williams & Hoel, 2003). While effective for capturing regular, seasonal patterns, these models often struggle with the complex, non-linear dynamics of a real-world transit system.

With the rise of machine learning, tree-based models like Isolation Forest and clustering algorithms like DBSCAN became popular. These methods are effective at identifying outliers in large, multi-dimensional datasets without needing to model the temporal sequence explicitly. For example, research has shown success in using these techniques to detect anomalous bus trajectories or unusual passenger ticketing patterns (Liu, Ting, & Zhou, 2008). However, they can sometimes miss context-dependent anomalies that are only apparent when viewed as part of a sequence.

More recently, the field has been dominated by deep learning, particularly Recurrent Neural Networks (RNNs) like LSTMs and GRUs. These models excel at learning patterns from sequential data, making them a natural fit for vehicle GPS traces. A systematic literature review on anomaly detection in connected vehicles highlighted that LSTMs, CNNs, and Autoencoders are the most commonly used deep learning techniques (Saleh, et al., 2024). They are often trained to predict a vehicle's future state (e.g., location or speed), and a large error between the prediction and reality signals an anomaly. This approach effectively captures the spatiotemporal nature of the problem.

Current research is pushing towards even more sophisticated models and addressing key limitations. This includes using Graph Neural Networks (GNNs) to model the entire transit network as a graph and exploring Transformer-based architectures to capture long-range dependencies. Furthermore, there is a growing emphasis on explainability and real-time performance, ensuring that detection systems are not only accurate but also interpretable and fast enough for real-world operational use.

3 Project Timeline

This timeline provides an ambitious but achievable weekly breakdown for completing the project.

- **Week 1: Foundation & Data Collection**
  - Focus: Establishing the project's technical foundation and data pipeline.
  - Process: This week is dedicated to environment setup and building the core data ingestion script. The primary process involves obtaining API credentials, defining a robust database schema in SQLite, and developing a resilient Python script that can run continuously to poll the CTA API. Error handling and logging are critical components. The week concludes with launching the script to begin accumulating a rich historical dataset.
  - Outcome: A stable, automated `fetch_data.py` script and a growing SQLite database with at least 50,000 rows of clean, structured train data.
- **Week 2: Analysis & Baseline**

  - Focus: Deep data understanding and establishing a performance benchmark.
  - Process: We transition from data collection to analysis. The main activities involve a comprehensive exploration of the dataset's temporal and spatial characteristics within a Jupyter Notebook. This analysis will directly inform our first anomaly detection model—a context-aware Z-score baseline. This baseline is critical as it provides the first tangible results and a benchmark against which all future, more complex models will be measured.
  - Outcome: A detailed EDA notebook (`01-EDA.ipynb`) with key visualizations (including GeoPandas maps) and a documented Z-score model that successfully identifies initial anomalies.

- **Week 3: Preprocessing & First Advanced Model**
  - Focus: Building a machine learning-ready dataset and training a powerful baseline model.
  - Process: This week is about formalizing the data preparation process. We will build a reusable preprocessing pipeline that handles missing values, encodes categorical variables, and scales numerical features. This pipeline will then be used to prepare data for our first advanced model, LightGBM. We will set up MLflow to rigorously track this experiment, logging parameters and performance metrics.
  - Outcome: A clean, well-documented modeling notebook (`02-Modeling.ipynb`), a trained LightGBM model artifact, and the initial MLflow experiment logs.

- **Week 4: API Development & Containerization**
  - Focus: Transforming the trained model into a portable, production-ready service.
  - Process: The focus shifts from data science to software engineering. We will build a FastAPI application to serve our trained LightGBM model. This involves creating a `/predict` endpoint that encapsulates the entire prediction logic. The entire application, including all dependencies and the model artifact, will then be containerized using Docker. The week ends with thorough local testing of the Docker container.
  - Outcome: A fully functional, containerized FastAPI service that can serve predictions locally.

- **Week 5: Initial Deployment & Dashboard (Vertical Slice)**
  - Focus: Achieving a full end-to-end, publicly accessible version of the system.

  - Process: This is a critical integration week. The Docker image will be pushed to a container registry (Docker Hub) and deployed to Google Cloud Run, making our API live. Concurrently, we will develop a Streamlit dashboard that consumes this live API. The goal is to create a simple but functional user interface that displays live train data on a map and highlights anomalies.
  - Outcome: A live API endpoint and a deployed Streamlit dashboard, representing the first complete, demonstrable version of the project.

- **Week 6: Advanced Modeling & Conformal Prediction**
  - Focus: Improving predictive accuracy and adding statistical rigor.
  - Process: With the core infrastructure in place, we return to modeling. We will train a more complex, sequence-aware model like an LSTM and compare its performance against the LightGBM baseline in MLflow. The superior model will be selected. The second half of the week involves upgrading the FastAPI service to integrate this new model and implement the conformal prediction logic, providing statistically valid p-values for each anomaly.
  - Outcome: An updated API running the best-performing model, now with the added capability of providing confidence scores for its predictions.

- **Week 7: Automation (CI/CD)**
  - Focus: Automating the testing and deployment workflow.
  - Process: This week is dedicated to implementing MLOps best practices. We will write unit tests for our data processing and API logic. Then, we will build a GitHub Actions workflow that automatically runs these tests, builds the Docker image, and deploys it to Google Cloud Run whenever changes are pushed to the main branch. This eliminates manual deployment steps and ensures reliability.
  - Outcome: A CI/CD pipeline that automates the entire deployment process.

- **Week 8: Monitoring & Final Polish**
  - Focus: Ensuring long-term reliability and preparing the project for presentation.
  - Process: The final week involves setting up a monitoring stack. We will instrument the API with Prometheus metrics and use Grafana to build two dashboards: one for service health (latency, errors) and another for model performance (conformal prediction coverage, false alarm rate). The remaining time will be spent on writing comprehensive documentation, including a system architecture diagram in the `README.md`.
  - Outcome: A fully monitored system and a professional, well-documented GitHub repository ready to be shared.

4 Technical Specifications & Dependencies

4.1 Environment Configuration

- Environment Manager: `Conda`
- Environment Name: `transit_anomaly`
- Python Version: `3.10`

4.2 Core Dependencies (`requirements.txt`)

```
# --- Core Data Science & EDA ---
pandas
numpy
jupyterlab
matplotlib
seaborn
scikit-learn

# --- Geospatial Analysis ---
geopandas
shapely

# --- Data Acquisition & Storage ---
requests
python-dotenv
sqlalchemy
# Note: sqlite3 is built into Python

# --- ML Modeling & Experiment Tracking ---
tensorflow
mlflow
lightgbm
statsmodels # For ARIMA baseline
# transformers # Optional, for advanced modeling

# --- Advanced ML ---
# For Conformal Prediction
mapie

# --- Backend API Server ---
```

```
fastapi
uvicorn[standard]

# --- Frontend Dashboard ---
streamlit
folium

# --- Infrastructure & Deployment ---
# These are installed separately, not via pip
# Docker
# Docker Compose
```

4.3 External Services & APIs

- Data Source: Chicago Transit Authority (CTA) Train Tracker API
- Cloud Deployment: Google Cloud Run (Primary) or AWS App Runner (Alternative)
- Dashboard Hosting: Streamlit Cloud (Free)
- Container Registry: Docker Hub

5 Project Phases