Final report
MLOPS - Road Accidents in France

Aryan Absalan, Tanja Schröder, Jannis Zeelen

1. Context and objectives

The primary objective of this project is to develop an application that provides real-time accident severity estimates to optimize emergency response. The prediction model, based on historical accident data, estimates severity and helps allocate resources more effectively. The application also assesses risk in specific areas, particularly focusing on time-related factors such as the time of day. By offering real-time predictions, the system improves emergency response efficiency and ensures resources are deployed where they are most needed.

Key benefits of the application include improved public safety and more efficient resource management for government agencies, including the Ministry of Interior and Transport. Emergency services such as SAMU and fire departments could reduce response times and optimize deployment. Insurance companies may benefit from cost savings due to more effective accident response, while tech companies and EU research grants could foster advancements in predictive modeling and real-time AI applications.

The application is designed for use by emergency medical services, police, and fire brigades, helping them assess accident severity, allocate resources, and prioritize interventions. Insurance companies and research institutions could also benefit from insights into accident patterns and risk assessments.

The application's administration could be handled by the IT departments of relevant government agencies or, if developed by a private tech company, by the company's technical support team, overseeing infrastructure, security, and updates.

2. Training

The foundation of the training process is built upon a dataset consisting of 3,213,764 entries (accidents) spanning from 2005 to 2016. The dataset includes 57 features and a target variable, *severity*. These features cover a wide range of aspects, including accident details, vehicle characteristics, participant data, environmental factors, as well as time and location information. With such a comprehensive set of variables, this dataset serves as a robust foundation for predicting the severity of accidents in the training process.

2.1. Data Cleaning

The data cleaning process involved several key operations: removal of duplicate records, handling of outliers and inconsistencies, dropping redundant or irrelevant columns, and converting data to the correct types. Missing values were addressed through imputation or removal, depending on the context. Additionally, relevant data sources were merged to create a unified dataset for analysis.

2.2. Feature Extraction

Feature extraction involved several steps to improve the quality and relevance of the dataset for modeling. First, a correlation matrix was used to identify features most correlated with the target variable, severity. Highly correlated features, such as 'Pedestrian_Location' and 'Pedestrian_Activity', were removed to reduce redundancy. New features were engineered, including Driver_Age, calculated from the year of birth and the accident year, and Time_of_Day, categorized into morning, afternoon, evening, and night based on the hour of the accident. The Day_of_Week feature was derived from the accident date, and a Road_Width_Proportion was created by calculating the ratio of the Main_Carriageway_Width to Total_Road_Width. After removing less relevant features, Random Forest and Decision Tree models were trained to assess feature importance, leading to the selection of 15 key features. The final dataset, containing these 15 features, was checked for missing values and duplicates, then saved for further modeling.

2.3. Modelling

The preprocessing of the dataset included splitting the data into training and testing sets, as well as applying transformations to the features. Categorical features were one-hot encoded, while numerical features were scaled using standardization to ensure that all features were on a similar scale for model training.

For modeling, three models were evaluated: Logistic Regression (LR), Decision Tree (DT), and Random Forest (RF). The Decision Tree was primarily used due to its simplicity, interpretability, and computational efficiency. The Decision Tree model was tuned with several hyperparameters, including max_depth=10, min_samples_leaf=5, and min_samples_split=5 to prevent overfitting and ensure that the tree was not too complex while maintaining predictive accuracy.

The model was evaluated using several key metrics: accuracy, F1 score, precision, and recall. The Decision Tree model was selected as the primary model due to its favorable balance between performance and computational efficiency, with execution time and model size considered for deployment in real-world emergency response applications.

2.4. DVC and Pipeline Automation

To ensure consistent version control, reproducibility, and streamlined model management, we integrated Data Version Control (DVC) and MLflow into the project, hosted on DagsHub. This combined setup enables full tracking and comparison of datasets, models, and performance metrics across multiple training iterations. The entire pipeline—from data ingestion, validation, transformation, model training, to model evaluation—is managed through DVC, allowing us to version and track the dataset and the model at each step of development.

To facilitate a reproducible and automated training process, we created a DVC pipeline that orchestrates each stage. The process begins with data ingestion, where new data is added and versioned in DVC. For demonstration purposes, artificial data is generated daily and appended to simulate real-time data ingestion.

Next, data validation and transformation occur, ensuring the data's quality and consistency. Necessary transformations are applied to prepare the data for model training.

Finally, model training and evaluation take place on a daily basis, using the latest data. Performance metrics are logged and versioned, and if model performance falls below a set threshold (e.g., 50% accuracy), an automated alert is triggered, and the API container stops to prevent low-performance predictions.

The DVC pipeline runs on a daily schedule, ensuring the model remains up-to-date with the latest data trends and performance levels. By automating this process, we enable the continuous integration and deployment of the model.

MLflow was integrated into the project to enhance model tracking and management. Each model training run is logged as a separate MLflow experiment, capturing important details such as hyperparameters, performance metrics (accuracy, F1 score, precision, and recall), and modifications to data preprocessing. This structured experiment tracking facilitates easy comparison across model versions, making it straightforward to identify the most effective configurations.

3. Database

In this project, we used a MySQL database to store essential information, which was organized into two main tables, one to store user infromation and the other to store predicition results.The users table stores user

4. API

The API for this project is designed to provide various endpoints for user authentication, model prediction, and risk analysis. The API integrates with both the machine learning model and the MySQL database to process user requests, provide predictions, store results, and perform risk-level analyses. Below is a detailed description of the API structure and endpoints.

4.1. API Endpoints

This FastAPI application provides several key endpoints to manage users, authenticate them, and allow for accident severity predictions. Below is an overview of the available endpoints and their functionality:

- GET/ - Returns the home page of the application.
- POST/register/ - Registers a new user.
    - Input: User's username, hashed password, and active status (disabled or not).
    - Process: Checks if the username already exists, creates the user if it doesn't, and stores the data securely (hashed password).
    - Output: The newly created user instance.
- POST/token - Logs in a user and generates a JWT access token.
    - Input: Username and password.
    - Process: Verifies the provided credentials and generates a JWT token for session authentication.
    - Output: The generated access token (Bearer token).
- GET/users/me/ - Retrieves the details of the currently authenticated user.
    - Input: Valid JWT token from the login request.
    - Output: The details of the currently logged-in user.
- POST/predict/ - Generates a prediction for accident severity based on user input and saves it to the database.
    - Input: Accident features.
    - Process: Uses a pre-trained model to predict accident severity based on the provided input. The prediction, along with the input data, is stored in the database.
    - Output: The predicted severity, prediction record ID, and the username of the current user.

- GET/risk-levels/{Department_Code} - Retrieves the risk level for a specific department.
    - Input: Department code.
    - Process: Estimates the risk level of an accident occurring for the specified department.
    - Output: The risk level and normalized risk score for the specified department.
- GET/visualization-risk-levels/ - Retrieves a real-time visualization of risk levels.
    - Process: Provides a real-time visualization of risk levels for all departments, merged with geographical data.
    - Output: The generated visualization image (PNG format) of risk levels across departments.

The API integrates with the database to store user and prediction data using SQLAlchemy ORM (SQLModel). Each prediction request is saved to the PredictionModel table, where the input features and predicted severity are associated with the authenticated user's ID. This ensures that all prediction data is linked to the user making the request. Additionally, logs are generated to monitor user actions and prediction requests. These logs serve as a tool for debugging and auditing, providing insights into the application's operations and helping track activities for security and maintenance purposes.

5. Containerization

To streamline deployment and ensure consistency across environments, we used Docker to containerize the API, the database, and the model. Docker provides a lightweight, portable, and scalable environment for running applications. By encapsulating the entire application stack including the API, database, and model into containers, we ensure that all dependencies are bundled together, and the application runs consistently regardless of the host machine's configuration.

6. Testing & Monitoring

For testing and monitoring, we have implemented two primary tools: Grafana and unit tests. Grafana is used for real-time monitoring of system performance, including tracking key metrics such as server uptime, prediction request counts, and response times. This provides an easy-to-use dashboard to visualize the health and performance of the application.

Unit tests are designed to check the status and functionality of the API endpoints and the new data ingestion process. These tests ensure that the application behaves as expected, verifying that data is correctly processed and that endpoints return accurate results. The unit tests focus particularly on the API's core functions, including user registration, prediction generation, and the ingestion of new data.

Model performance is continuously tracked, with particular attention to accuracy and prediction reliability. If the model's performance falls below 50%, an automated response is triggered, causing the container to shut down and the API to stop functioning. In such cases, an alert is sent by email to notify the team of the performance issue. Additionally, a sample of the most recent data is reviewed to diagnose potential causes of the performance drop and to take corrective action if necessary.

7. Implementation scheme

The implementation scheme offers a detailed overview of the entire project, encompassing all key components such as data preprocessing, model training, deployment, user interaction, monitoring, testing, and containerization, while illustrating how these elements interact and work together to create a seamless end-to-end solution.

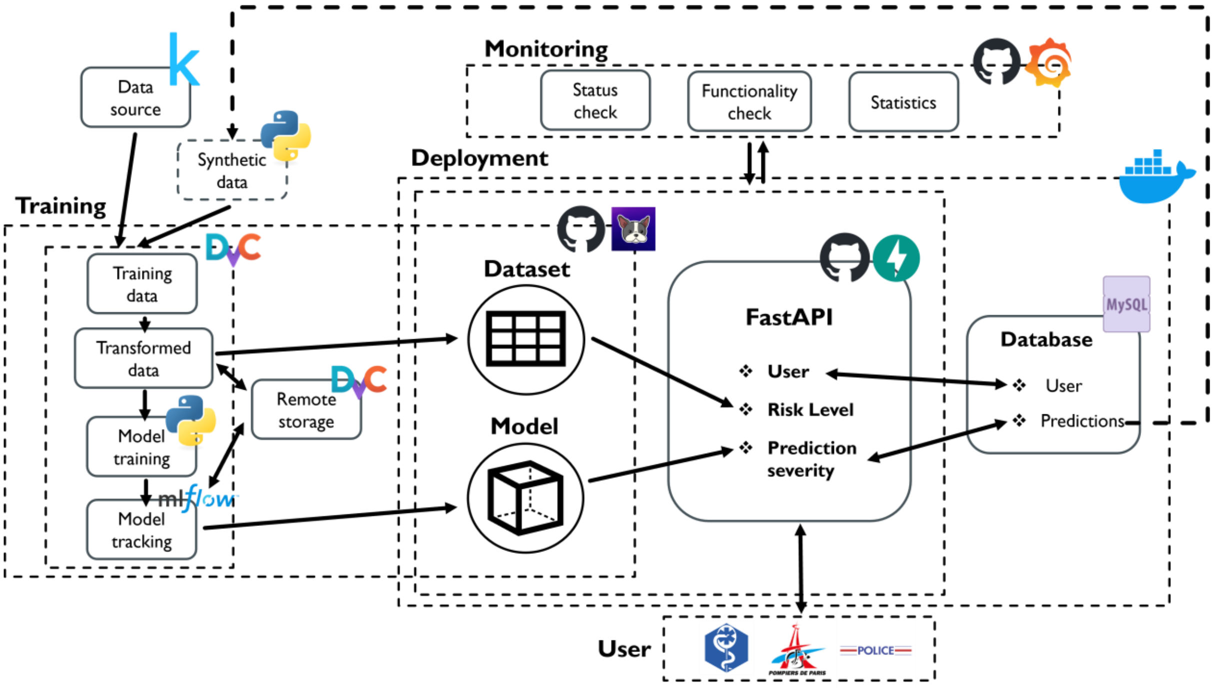

Figure 1: Implementation scheme

8. Project Summary and Future Enhancements

The project aimed to develop a real-time accident severity prediction application to support emergency services, police, and other public safety organizations in prioritizing and allocating resources effectively. By analyzing historical accident data, the model predicts accident severity, enabling responders to quickly assess the level of intervention required. The project followed a comprehensive MLOps framework that included data preparation, model training, and deployment.

The dataset, containing over 3 million entries, was cleaned, processed, and transformed, with key features extracted and engineered to predict accident severity. A Decision Tree model was chosen for its balance of accuracy and computational efficiency. Automated pipelines for data ingestion and model training were set up using DVC and MLflow, ensuring version control and continuous integration of daily data updates. MLflow tracked performance metrics and experiments throughout the process.

An API was developed to handle user authentication, severity predictions, and risk assessments for various departments, with data managed in a MySQL database. Docker was used to containerize the application, ensuring consistent deployment across different environments, while Grafana monitored system performance. Unit tests were implemented to ensure the accuracy and functionality of the API.

Looking ahead, there are several opportunities to enhance the system. One key improvement is implementing Kubernetes for better orchestration and scaling, ensuring efficient resource management and high availability. Enhancing the monitoring setup with advanced Grafana dashboards would provide deeper insights into system performance for faster issue resolution.

Model performance could be improved by experimenting with more complex models, such as ensemble methods or deep learning architectures. Expanding the testing framework with additional unit tests would ensure robustness as the system scales.
User experience could be improved by developing a more intuitive front-end, including a mobile app for emergency responders in the field. Integrating external data sources like traffic, weather, and GPS could enhance prediction accuracy.