Crash Rate Prediction from Traffic Volume Data using AI

Final Report

Jonathan Le, Travis Chan, Syeda Afia Z. Hossain,
Arham Asam, Devanshu Khadka

CS4624 Multimedia, Hypertext, and Information Access
Instructor: Professor Mohamed Farag
Client: Dr. Mohamed Farag
Virginia Tech, Blacksburg, VA 24061

December 12, 2024

Contents

1 Executive Summary

In today's fast-paced, technology-driven world, we're generating more transportation data than ever before. This data offers opportunities to making roads safer and more efficient, but is often hard to take advantage of. Our client, Dr. Mohamed Farag is a researcher in the Center for Sustainable Mobility (CSM) at the Virginia Tech Transportation Institute, a research institute whose work contributes to the advancement of the transportation industry.

To address this challenge, we have developed a user-friendly web application that harnesses machine learning to predict crash rates based on traffic volume data. The goal is to empower transportation authorities and researchers with actionable insights derived from the vast amounts of data collected, ultimately enhancing road safety.

We have developed a web application that allows users to use machine learning models to predict crash rates for roads. It is comprised of four main components: a frontend interface, a backend server, an API, and offline machine learning model development using Google Colaboratory. Users can easily submit new data for predictions by navigating to the Input Page via the "Add New Prediction" button on the Home Page. The Reports Page provides detailed insights into individual predictions, including the date, model used, input data, and the predicted crash rate.

Administrators have additional privileges, such as managing machine learning models through the Model Management section. They can upload new models, specify model details like names and attributes, and monitor existing models via the Model List page, which displays all models along with their creation dates and statuses. These features help in effectively monitoring and updating the models used for predictions.

We've implemented secure user authentication on the frontend using JWT tokens for login and sign-up processes. The Home Page presents users with a tabular view of past predictions, allowing them to see the date, model used, and results, as well as the option to add new predictions.

Our backend architecture features a Next.js server for the web backend and a FastAPI server for the machine learning backend. This setup ensures the application is efficient, scalable, and user-friendly for both end-users and developers aiming to add new features. The web backend handles user authentication, prediction collections, and model management, while interfacing with the FastAPI ML backend to generate predictions.

To ensure quality and reliability, we've conducted extensive testing and evaluation, including machine learning model testing, model evaluation, and client assessments. So far, we've completed testing for a deep neural network regression model and have received valuable feedback from our client on the web application prototype.

We recognize that further work is needed to finalize the product. This report outlines our plans for the remainder of the semester and proposes ideas for future enhancements beyond the current project scope—all aimed at making our roads safer through data-driven insights.

2 Introduction

2.1 Problem

Highways across Virginia experience frequent crashes, leading to significant concerns regarding public safety. Predicting crash rates based on traffic volume is crucial to enhancing road safety measures and preventing accidents. However, current methods for predicting crash rates lack advanced analytics, making it difficult for transportation authorities to take proactive steps in reducing accident rates. This issue is compounded by the lack of user-friendly tools that allow agencies to upload and analyze their own crash rate data to make informed decisions.

The primary problem we aim to solve with this project is to develop a web-based tool that utilizes machine learning models, such as linear regression and deep learning regression, to predict crash rates based on historical traffic data. By providing a platform that allows users to upload their traffic data, interact with models, and check crash predictions, the web app will empower transportation authorities to implement more effective preventive measures.

2.2 Motivation

Our primary motivation to address shortcomings in traffic safety analytics by providing a tool for relevant stakeholders to assess current and future road developments in Virginia. Data driven solutions are increasingly relevant in the modern day and have the power to save lives. We hope this project will provide the necessary tools to make these solutions accessible to researchers and traffic authorities, which may help positively influence policy change, traffic engineering, and other factors that can help make Virginia roads safer for everyone.

2.3 General Approach

Our approach, as discussed with the client, is to develop a web application to allow the client and stakeholders to utilize machine learning to make data-driven decisions in the context of crash prediction. By interacting with a user interface, users will be able to provide their data and utilized readily available models to generated predictions. There predictions will then be stored in the form of collections for future viewing if desired by the user. On the admin side, all user functionality will be accessible with the addition of functionality to upload models to be used by both the users and admin. Machine learning models are developed offline. Data processing and data cleaning is integrated on the back end to ensure database management integrity.

3 Requirements

3.1 User Functionality

The client requires user functionality to upload/insert data and generate predictions to be stored in a prediction collection.

Uploading and inputting data for prediction will be performed on a page separate from the homepage. The client has requested to integrated two options: one observation and .csv upload. Regarding one observation, to meet this requirement, we plan on integrating conditional rendering based on the model selected to provide the correct number of input fields and corresponding labels for the user to provide data. Regarding the .csv upload, to meet this requirement, we have integrated an upload box in the UI and plan on integrating data processing checks on the back end to ensure (1) a .csv file is uploaded, (2) the number of attribute labels is equal to the number of input fields for the model, and (3) ensure compatible input value types.

Prediction collections will be visible to the user on the home page. From the home page, users can select a specific prediction collection, which will bring them to a separate page detailing prediction information. Prediction information includes input data, output data, the model selected, and the date and time the prediction was generated. Additionally, users will have the option to delete a prediction collection directly from the home page, allowing them to manage their data efficiently and keep their workspace organized.

3.2 Admin Functionality

The client requires admin functionality to be integrated which includes, similar to the user functionality requirements, uploading data and tracking prediction collections.

Additionally, the client has requested to include functionality that will allow admin to upload models to be used by both users and admin. When an admin uploads a model, the user interface will request the admin to provide labels for the input attributes of the model, so that we can dynamically render input fields when making predictions.

3.3 Machine Learning

The client has required the complete development of a deep learning regression model to serve as the default model under the assumption no additional models have been uploaded yet. The model is to be train, tested, and evaluated using a sample dataset and code base provided by the client.

Further requirements can be found at the project page

4 Design

4.1 System Architecture

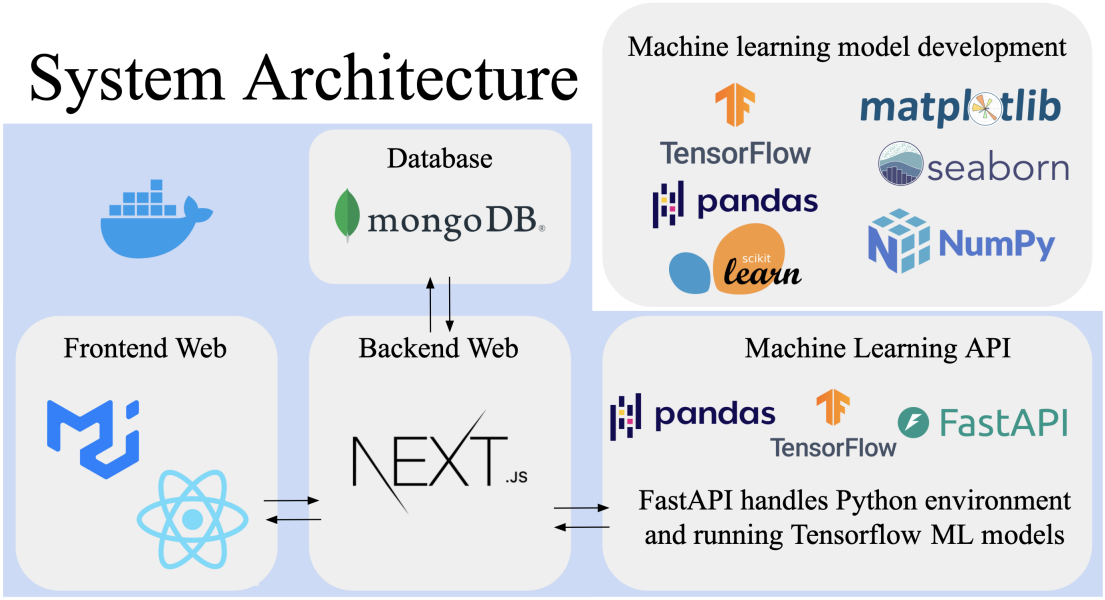

Figure 1: Final System Architecture Design presented in final presentation

The figure above shows our final architecture. The frontend is built using React and Material-UI, which helps create a user-friendly interface with a clean design. The backend is implemented with Next.js, which handles the core application logic, routing, and communication with the other parts of the system. For storing and managing data, we used MongoDB, which provides a reliable and efficient database solution. The machine learning API is developed using FastAPI, allowing us to integrate machine learning models effectively. This API utilizes libraries like TensorFlow, Pandas, and NumPy for training and running the models. We also incorporated Docker for containerization, which made deployment and scalability much easier. During the development of the ML models, we used Matplotlib and Seaborn for data visualization to better understand the data and refine our models. This architecture allowed us to build a modular, scalable, and efficient system.

4.2 Frontend Design

4.2.1 User Pages

The user pages are designed to provide an intuitive experience for general users who interact with the application by making predictions and viewing past transactions. The primary pages include:

Homepage The Homepage allows for users to manage their interactions with the system. It includes the following features:

- Your Collections section displays a table with the following columns for the user's past transactions:
  - Created At: Date and time the collection was created.
  - Collection Name: Name of the transaction collection.
  - Number of Predictions: Total predictions contained in the collection.
  - Actions:
    - **View Button**: Navigates to the *View Reports Page*, which provides detailed insights about the collection.
    - **Delete Button**: Removes the collection.
- Add New Collection button:
  - Navigates to the Add New Collection Page, where users can create a new collection.
- Navigation options available on all pages:
  - A Home Button to navigate back to the Homepage.
  - A User Menu with a Logout option.

The homepage helps streamline the user's workflow, giving direct access to previous predictions and allowing easy navigation to other pages such as adding predictions and viewing reports.

Create Collection Page The Create New Collection Page enables users to create a new transaction collection. This page includes:

- Collection Name Field: A text input field for users to specify the name of the new collection.
- A Create Collection button to create the collection and proceed to the Add Predictions Page.

The Create New Collection Page is simple and focused, making it easy for users to name and initiate a new collection.

Add Prediction Page The Add Prediction Page enables users to submit data for new crash rate predictions. This page includes:

- Model Selection Dropdown: Enables users to select a machine learning model from a predefined list.
- Input Type Selection: Options include Manual Input and CSV File Upload.
  - For Manual Input, users are provided with fields to enter the Average Daily Traffic (ADT) and Length of the road segment.
  - For CSV File Upload, users can upload a file containing the necessary data.
- Add Transform Log Button: Allows users to include additional transformation logs relevant to the predictions.
- Submit Button: Sends the input data to the backend for processing and redirects the user to the View Collection Page to review results.

The Add Predictions Page is designed to be flexible and user-friendly, accommodating various input types and additional transformations.

View Reports Page The View Reports Page allows users to review past predictions and their associated details. This page includes:

- A **"Back to Collections"** button that navigates users to the homepage.
- The name of the collection prominently displayed at the top of the page.
- The **number of predictions** within the collection displayed below the collection name.
- The **model used** for the predictions listed for clarity.
- A table that includes the following columns:
  - **Attributes**, such as Length and Average Daily Traffic (ADT).
  - **Result**, showing the prediction outcome.
  - **Source Type**, indicating the input type (e.g., manual or CSV).
- An **"Add Prediction"** button that allows users to add a new prediction to the same collection.

The View Reports Page is designed to provide users with a detailed and organized view of a specific collection's predictions, enabling them to analyze data and add new predictions conveniently. The addition of navigational elements like the "Back to Collections" button and an "Add Prediction" button enhances user workflow and accessibility.

4.2.2 Admin Pages

The admin pages are designed to provide an intuitive experience for administrative users who interact with the application by making predictions, adding models, and viewing past transactions. The primary pages include:

Homepage The Homepage allows users to manage their interactions with the system. It includes the following features:

- Your Collections section displays a table with the following columns for the user's past transactions:
  - Created At: Date and time the collection was created.
  - Collection Name: Name of the transaction collection.
  - Number of Predictions: Total predictions contained in the collection.
  - Actions:
    - **View Button**: Navigates to the *View Reports Page*, which provides detailed insights about the collection.
    - **Delete Button**: Removes the collection.
- Add New Collection button:
  - Navigates to the Add New Collection Page, where users can create a new collection.
- Model Management Menu:
  - A menu in the navigation bar with the following options:
    - **Add Model**: Takes users to the Add Model Page to upload new machine learning models.
    - **Model List**: Takes users to the Model List Page where they can view and manage existing models.
- Navigation options available on all pages:
  - A Home Button to navigate back to the Homepage.
  - A User Menu with a Logout option.

The Homepage helps streamline the user's workflow, giving direct access to previous predictions and allowing easy navigation to other pages such as adding predictions, viewing reports, and managing models.

Model Management The Model Management section of the system provides administrative users with tools to manage machine learning models. Under Model Management, there are two primary options in the navigation menu: Add Model and Model List.

Add Model The Add Model page allows administrators to add new machine learning models to the system. This page includes:

- A text field for entering the **Model Name**, where the administrator specifies the name of the new model.
- A text field for **Input Fields (Ordered)**, allowing administrators to define the attributes for the model in an ordered list.

- An **Add Input** button that lets the administrator add additional input fields as needed.
- A text indicating **Upload Model File (only .keras and .h5 files accepted)** to inform administrators of the accepted file formats.
- A **Choose File** button to upload the model file.
- A **Submit** button to finalize the addition of the new model to the system.

The Add Model page is designed to make it simple for administrators to register new machine learning models in the system by providing fields to name the model, define its input fields, and upload the model file.

Model List The Model List page displays a list of all models currently stored in the system. Each entry in the list includes:

- The **Name** of each model.
- The **Date** when the model was added to the system.
- An **Action** column with a delete button (depicted by a trash icon), allowing administrators to remove models from the system.

The Model List page helps administrators keep track of all models, ensuring easy access for management, updates, or removal as needed.

Create Collection Page The Create New Collection Page enables users to create a new transaction collection. This page includes:

- Collection Name Field: A text input field for users to specify the name of the new collection.
- A Create Collection button to create the collection and proceed to the Add Predictions Page.

The Create New Collection Page is simple and focused, making it easy for users to name and initiate a new collection.

Add Prediction Page The Add Prediction Page enables users to submit data for new crash rate predictions. This page includes:

- Model Selection Dropdown: Enables users to select a machine learning model from a predefined list.
- Input Type Selection: Options include Manual Input and CSV File Upload.
  - For Manual Input, users are provided with fields to enter the Average Daily Traffic (ADT) and Length of the road segment.
  - For CSV File Upload, users can upload a file containing the necessary data.
- Add Transform Log Button: Allows users to include additional transformation logs relevant to the predictions.

- Submit Button: Sends the input data to the backend for processing and redirects the user to the View Collection Page to review results.

The Add Predictions Page is designed to be flexible and user-friendly, accommodating various input types and additional transformations.

View Reports Page The View Reports Page allows users to review past predictions and their associated details. This page includes:

- A **"Back to Collections"** button that navigates users to the homepage.
- The name of the collection prominently displayed at the top of the page.
- The **number of predictions** within the collection displayed below the collection name.
- The **model used** for the predictions listed for clarity.
- A table that includes the following columns:
  - **Attributes**, such as Length and Average Daily Traffic (ADT).
  - **Result**, showing the prediction outcome.
  - **Source Type**, indicating the input type (e.g., manual or CSV).
- An **"Add Prediction"** button that allows users to add a new prediction to the same collection.

The View Reports Page is designed to provide users with a detailed and organized view of a specific collection's predictions, enabling them to analyze data and add new predictions conveniently. The addition of navigational elements like the "Back to Collections" button and an "Add Prediction" button enhances user workflow and accessibility.

4.3 Backend Design

4.3.1 Web and Machine Learning Backend

Introduction

The backend architecture of the application is designed to be modular, scalable, and secure, comprising two main components: the web backend and the machine learning (ML) backend. The design follows best practices to ensure efficient handling of user requests, secure data management, and seamless integration between components.

Architecture Overview

The backend architecture consists of:

- **Web Backend**: A Next.js server that handles user authentication, API requests from the frontend, data validation, and interactions with the database.
- **ML Backend**: A FastAPI server that provides machine learning prediction services, dynamically loading models as required.
- **Database**: A MongoDB instance that stores user data, model metadata, and prediction records.

Design Principles

The backend design is guided by the following principles:

- **Separation of Concerns**: Dividing responsibilities between the web backend and the ML backend to enhance maintainability and scalability.
- **Security**: Ensuring that sensitive operations are protected through authentication and authorization mechanisms, and that the ML backend is not directly exposed to external clients.
- **Scalability**: Designing components to handle increasing loads by allowing independent scaling of the web and ML backends.
- **Modularity**: Building components that can be developed, tested, and deployed independently.

Component Descriptions

Web Backend:

The web backend serves as the intermediary between the frontend and backend services. Its responsibilities include:

- **API Endpoints**: Exposing RESTful APIs for user authentication, model management, and prediction collections.
- **Authentication and Authorization**: Managing user sessions, handling login/logout, and enforcing access controls based on user roles.
- **Data Validation**: Validating incoming data from the frontend before processing or storing it.
- **Database Interactions**: Performing CRUD operations on the MongoDB database to manage users, models, and predictions.
- **Integration with ML Backend**: Forwarding prediction requests to the ML backend and returning results to the frontend.

ML Backend:

The ML backend focuses solely on processing machine learning predictions. Its key features are:

- **Dynamic Model Loading**: Loading models from the file system based on the `model_id` provided in prediction requests.
- **Prediction Services**: Providing endpoints to process single and batch prediction requests.
- **Data Processing**: Handling data transformations required by the models (e.g., log transformations).
- **Error Handling**: Returning informative error messages for issues like missing models or invalid input data.

Data Flow and Interaction

1. **User Interaction**:
    - The user interacts with the frontend application, submitting requests for actions such as login, uploading models, or making predictions.
2. **Web Backend Processing**:
    - The web backend receives API requests from the frontend, validates the input, and performs necessary operations.
    - For model uploads, it saves the model file to the shared directory and updates the database with metadata.
    - For predictions, it validates input data and sends a structured request to the ML backend.
3. **ML Backend Processing**:
    - Upon receiving a prediction request, the ML backend loads the specified model (if not already cached) and processes the input data.
    - The ML backend returns the prediction results to the web backend.
4. **Response to User**:
    - The web backend processes the response from the ML backend, updates the database if necessary, and sends the results back to the frontend.

Security Considerations

- **Authentication**: User authentication is managed by the web backend using JWT tokens, ensuring that only authorized users can access protected resources.
- **Role-Based Access Control**: Different roles (e.g., user, admin) are enforced to restrict access to certain functionalities, such as model management.
- **Data Validation**: Both the web backend and ML backend perform data validation to prevent malicious inputs.
- **Internal Services**: The ML backend is not exposed directly to the internet, reducing the attack surface and ensuring that only the web backend can communicate with it.

Scalability and Performance

- **Independent Scaling**: The web backend and ML backend can be scaled independently based on demand, allowing for efficient resource utilization.
- **Caching Mechanisms**: The ML backend caches loaded models to improve prediction response times.
- **Asynchronous Processing**: The architecture can be extended to use asynchronous processing for handling large batches or intensive computations.

Conclusion

The backend design emphasizes a clear separation of responsibilities, security, and scalability. By dividing the system into the web backend and ML backend, each component

can be optimized for its specific tasks. This modular approach facilitates maintenance, testing, and future enhancements, ensuring that the application remains robust and efficient as it evolves.

4.3.2 Machine Learning

We have chosen to complete machine learning model developing using the Python programming language. Machine learning model development is composed of many steps: exploratory data analysis, data processing, model selection, model development, and data visualization. Ideally, working in one environment and using one programming language allows for efficient model development. In discussion with our client, our machine learning design is tailored for regression model development.

Python is a programming language with many libraries to perform all steps of machine learning development. Our machine learning model development utilizes the following Python libraries: Tensorflow, Numpy, Pandas, Matplotlib, Seaborn. Tensorflow is used for building machine learning models, Numpy is used for data processing, Pandas is used for data analysis, and Matplotlib and Seaborn are both used for data visualization. Since machine learning model development is carried out offline, additional technology is required to connect machine learning models to the web backend. We used FastAPI to establish dataflow from model to web backend so that the data can be rendered on the frontend to be visible by users.

5 Implementation

5.1 Frontend

The frontend of our project is implemented using **ReactJS**, a popular JavaScript library for building user interfaces, along with **Material-UI (MUI)** for styling and design components. By leveraging these technologies, we aim to create a responsive and dynamic web application that provides a seamless user experience.

5.1.1 ReactJS

ReactJS enables the development of reusable UI components, which enhances code modularity and maintainability. We organized the project structure in a component-based architecture, making it easier to manage and scale as new features are added. Key features of the frontend implementation include:

- **Component-Based Architecture:** Each section of the web application, such as the Home Page, Add Prediction Page, and Reports Page, is implemented as a separate component. This modular approach improves code reusability and maintainability.
- **State Management:** We used React's built-in `useState` and `useEffect` hooks to manage component states and lifecycle events. For example, the input fields on the Add Prediction Page rely on state variables to store and update user inputs.
- **Dynamic Rendering:** We employed conditional rendering to dynamically display components based on user actions. For instance, when an admin uploads a model, the system renders additional input fields for specifying the model attributes.
- **Routing and Navigation:** To manage navigation between different pages, we utilized the `react-router-dom` library. This enables users to easily switch between the Home Page, Add Prediction Page, Reports Page, and Admin Pages.

5.1.2 Material-UI (MUI)

Material-UI is a React component library that provides pre-designed components and theming options for building responsive and visually appealing user interfaces. By utilizing MUI, we were able to achieve a consistent design language throughout the application. Key aspects of MUI usage include:

- **Responsive Layout:** MUI's `Grid` system and responsive design utilities allowed us to create layouts that adapt to various screen sizes, providing an optimal user experience across devices.
- **Theming and Customization:** We customized MUI's default theme to match our project's design guidelines, ensuring a consistent color scheme and typography across all pages.
- **Pre-Built Components:** We utilized MUI's pre-built components such as `Button`, `Table`, `Dialog`, and `AppBar` to expedite the development process while maintaining a cohesive design.

Overall, the combination of ReactJS and Material-UI has enabled us to develop a frontend that is both user-friendly and visually consistent. By utilizing reusable components and a responsive design, we aim to provide an intuitive interface that facilitates effective interaction with the underlying machine learning models.

5.2 Backend Web

5.2.1 Web Backend

Introduction: The web backend connects the front end interface to backend services through a REST API, handling HTTP requests, processing data, and communicating with both MongoDB and the FastAPI ML backend.

Technologies Used:

- **Framework:** Next.js 14.2.13
- **Language:** JavaScript (ES6+)
- **Libraries and Middleware:**
  - **jsonwebtoken:** For JWT authentication
  - **bcrypt:** For password hashing
  - **Axios:** For making HTTP requests to the FastAPI ML backend
  - **Mongoose:** For MongoDB object modeling
  - **cookies-next:** For handling cookies in Next.js

Implementation Details: The backend utilizes Next.js, a React framework that supports server-side rendering and API routes, making it suitable for building scalable web applications without the need for an external server

API Routes Next.js provides a built-in API routing system, allowing the creation of API endpoints within the app/api directory. This project uses the more recent NextJS app routing model. The web backend exposes several RESTful API routes

Endpoint	Method	Description
`/api/auth/register`	POST	Registers a new user
`/api/auth/login`	POST	Authenticates user credentials and issues JWT
`/api/auth/logout`	POST	Logs out the user by clearing the authentication cookie
`/api/predict/single`	POST	Generates a prediction using manual user input
`/api/predict/csv`	POST	Generates predictions using CSV file input
`/api/predict/add`	POST	Creates a new prediction collection
`/api/predictions`	GET	Retrieves all prediction collections for the authenticated user
`/api/predictions/{id}`	GET	Retrieves a specific prediction collection
`/api/predictions/{id}`	PUT	Updates a specific prediction collection
`/api/predictions/{id}`	DELETE	Deletes a specific prediction collection
`/api/models`	GET	Retrieves available ML models
`/api/models`	POST	Allows admin to upload new models
`/api/models/{id}`	PUT	Updates a specific model (admin only)
`/api/models/{id}`	DELETE	Allows admin to delete a model
`/api/users/{id}`	GET	Retrieves user profile information
`/api/users/{id}`	PUT	Updates user profile information
`/api/users/{id}`	DELETE	Deletes a user account

Table 1: API Routes in the Web Backend

The REST API allows for CRUD operations for both prediction collections and models, allowing users to easily generate and manage predictions. Note that currently login with JWT authentication, and basic prediction routes have been implemented. The remaining routes have been designated/designed and will be implemented as detailed in section 10.1

Routing and Middleware:

- **API Routes:** Defined within the app/api directory using file-based routing.
- **Middleware:** Custom middleware is implemented to handle JWT authentication and error handling.

Authentication and Authorization:

- **JWT Authentication:** Upon successful login, the server issues a JWT, which is stored in an HTTP-only cookie using the `cookies-next` library.
- **Protected Routes:** Middleware functions check for a valid JWT before granting access to certain API routes.
- **Role-Based Access Control:** Admin routes are protected and only accessible to users with the admin role.

Session Management

- **HTTP-Only Cookies:** Tokens are stored in HTTP-only cookies to enhance security and prevent XSS attacks.

- **Token Expiration:** JWTs are configured to expire after 24 hours, requiring users to re-authenticate periodically.

Error Handling and Logging

- **Error Handling:** Standardized error responses are sent from API routes, and client-side error handling provides user feedback.
- **Logging:** Server-side logging is implemented using the built-in Next.js logging capabilities.

Security Measures

- **Password Hashing:** User passwords are hashed using `bcryptjs` before being stored in the database.
- **Input Validation:** Data received from clients is validated using custom validation logic.

Communication with Frontend

- **Data Exchange Format:** JSON is used for data exchange between the frontend and backend.
- **API Consumption:** Frontend components call the API routes using `fetch` or `Axios` for data fetching.

Interaction with MongoDB

- **Database Connection:** A utility function `mongodb` is used to establish a connection to MongoDB using `mongoose`.
- **Data Models:** Defined using Mongoose schemas and used across API routes for database operations.

Integration with FastAPI ML Backend

- **HTTP Requests:** The web backend communicates with the FastAPI ML backend via HTTP requests using `Axios`.
- **Data Flow:** When a prediction is requested:
  1. The API route receives the input data.
  2. It makes a POST request to the ML backend's prediction endpoint with the input data.
  3. Receives the prediction result from the ML backend.
  4. Stores the prediction in MongoDB under the Predictions collection.
  5. Returns the result to the frontend.
- **Internal Service Model:** The ML backend acts as an internal service and is only accessible through the web backend, minimizing security concerns and bugs.

Conclusion: The web backend effectively manages user authentication, session management, data validation, and serves as the coordinator between the frontend and backend services. By leveraging robust frameworks and adhering to best security practices, it ensures a secure and seamless user experience.

5.2.2 MongoDB Schema Design

Introduction: MongoDB is employed as the database solution due to its flexibility in handling JSON-like documents and ease of integration with Node.js applications. The schema design focuses on efficiently storing user information, prediction records, and machine learning models while ensuring data integrity and performance. [8]

Collections Overview

1. **Users Collection**
2. **Models Collection**
3. **Predictions Collection**

Users Collection

Field	Type	Description
`_id`	ObjectId	Unique identifier for the user
`username`	String	Username of the user (unique identifier)
`password`	String	Hashed password of the user
`role`	String (Enum: `user`, `admin`)	Role-based access control (default is `user`)
`createdAt`	Date	Timestamp when the user was created

Table 2: Users Collection Schema

Design Considerations:

- **Unique Username:** The `username` field is unique and required, serving as the primary identifier for users during authentication.
- **Password Security:** Passwords are stored as hashed strings to enhance security.
- **Role-Based Access Control:** The `role` field determines the user's permissions within the application, allowing for differentiated access between regular users and admins.
- **Timestamps:** The `createdAt` field records when the user account was created, useful for auditing and account management.

Models Collection

Field	Type	Description
`_id`	ObjectId	Unique identifier for the model
`name`	String	Name of the model
`filePath`	String	Path to where the model is stored
`inputFields`	Array of Strings	Ordered list of input field names required by the model
`createdAt`	Date	Timestamp when the model was uploaded
`createdBy`	ObjectId (Reference to User)	Reference to the admin who uploaded the model

Table 3: Models Collection Schema

Design Considerations:

- **Model Metadata:** The `name`, `filePath`, and `inputFields` provide essential information about the model, facilitating its selection and use during predictions.
- **Input Fields Ordering:** Maintaining the order of `inputFields` ensures that input data aligns correctly with the model's expectations.
- **Model Ownership:** The `createdBy` field links the model to the admin user who uploaded it, aiding in accountability and management.
- **Timestamps:** The `createdAt` field records when the model was added to the system.

Predictions Collection

Field	Type	Description
`_id`	ObjectId	Unique identifier for the prediction collection
`userId`	ObjectId (Reference to User)	Reference to the user who owns the collection
`modelId`	ObjectId (Reference to Model)	Reference to the ML model used for predictions
`predictions`	Array of Prediction Objects	Array of predictions in the collection
`createdAt`	Date	Timestamp when the collection was created

Table 4: PredictionCollections Collection Schema

Prediction Object (Embedded in `predictions` Array):

Field	Type	Description
`inputs`	Array of Key-Value Pairs	Ordered input field names and their values
`result`	Number	The predicted result for each input
`sourceType`	String (Enum: `manual`, `csv`)	Source type of the prediction
`createdAt`	Date	Timestamp when the prediction was added

Table 5: Prediction Object in Predictions Array

Design Considerations:

- **User and Model References:** The `userId` and `modelId` fields establish relationships with the Users and Models collections, linking each prediction collection to its owner and the model used.
- **Embedded Predictions:** Storing predictions as an embedded array within a collection allows for efficient retrieval and management of a user's predictions.
- **Prediction Details:** Each prediction object contains detailed information about the inputs, result, source type, and timestamp.
- **Source Type Tracking:** The `sourceType` field differentiates between predictions added manually and those imported via CSV, aiding in data analysis and management.

Schema Design Rationale: The schema design aims to balance flexibility and data integrity:

- **Flexibility:** MongoDB's document-oriented approach allows for the embedding of predictions within collections, facilitating complex data structures which makes managing predictions and relevant metadata easier.
- **Data Integrity:** Using required fields, unique constraints, and enumerations ensures that the data stored adheres to the application's rules and prevents invalid data entry.

Conclusion: The MongoDB schema design effectively balances flexibility and structure, accommodating the dynamic nature of machine learning models while ensuring data integrity. It supports efficient data retrieval and scalability, aligning with the project's current needs and potential future growth.

5.2.3 FastAPI ML Backend

Introduction

The FastAPI ML backend serves as the core processing engine for machine learning predictions. It handles prediction requests from the web backend, processes input data using trained models, and returns prediction results. The ML backend is designed to be an internal service, only accessible through the web backend. This separation of concerns ensures data is processed and validated correctly.

Technologies and Libraries Used

- **Framework:** FastAPI
- **Language:** Python 3.8+
- **Machine Learning Libraries:**
  - **TensorFlow:** For loading and running ML models in the .h5 and .keras format.
- **Data Processing Libraries:**
  - **NumPy:** For numerical computations.
  - **Pandas:** For data manipulation and CSV handling, especially in batch predictions.
- **Others:**
  - **Uvicorn:** ASGI server for running FastAPI applications.

Architecture Overview

The FastAPI ML backend is an internal service that loads machine learning models from the file system and exposes API endpoints for predictions. It is not directly accessible by the frontend and communicates solely with the web backend. This design ensures security and encapsulates the ML functionalities within a dedicated service.

Model Loading

The ML backend has access to a designated models directory (e.g., `models/`) and loads available models upon a request from the web backend. Each model is identified by a unique name corresponding to its filename or a specified identifier.

API Endpoints The ML backend exposes two primary endpoints:

Endpoint	Method	Description
`/predict/single`	POST	Generates a prediction for a single input
`/predict/batch`	POST	Generates predictions for multiple inputs

Table 6: API Endpoints in the FastAPI ML Backend

Model Loading and Caching

- **Dynamic Loading:** Models are loaded from the file system when a prediction request is received.
- **Caching:** Caching may be implemented for ML models to improve performance of the application as it scales.

The web backend handles:

- **Model uploads and deletions**, saving model files to the shared models directory.
- **Storing model metadata** in the MongoDB database.

- **Validating model availability** before sending prediction requests.
- **Parsing CSV files** and converting data to JSON format for batch predictions.
- **Sending structured prediction requests** to the ML backend's endpoints.

The ML backend focuses on:

- **Loading models dynamically** based on the `model_id` provided.
- **Serving prediction requests** using the loaded models.
- **Handling errors gracefully** if models cannot be loaded or predictions fail.

Prediction Pipeline

1. **Data Reception:**
   - ML backend accepts JSON payloads and/or CSV files containing validated input data.
2. **Data Validation:**
   - Validate input data formats again, ensuring .
3. **Preprocessing:**
   - Applies necessary transformations (e.g., log transformation) before prediction.
4. **Model Inference:**
   - Runs the data through the loaded model to generate predictions.
5. **Post-processing:**
   - Converts predictions into a user-friendly format.
6. **Response:**
   - Sends back the prediction results in JSON format.

Concurrency and Performance

- **Asynchronous Processing:**
  - FastAPI supports asynchronous endpoints using `async def`, allowing for non-blocking I/O operations.
- **Uvicorn Server:**
  - Runs the application using an ASGI server optimized for high performance.
- **Batch Predictions:**
  - Supports bulk predictions when CSV files are uploaded, processing data efficiently.

Conclusion

The FastAPI ML backend, designed to dynamically load models based on prediction requests, provides a flexible and efficient service for machine learning predictions. By

focusing solely on prediction processing and leveraging caching mechanisms, the backend achieves a balance between performance and scalability. This architecture, with model management handled by the Next.js web backend, ensures a clean separation of concerns and simplifies the overall system design.

5.3 Machine Learning Model Development

5.3.1 The Client Data

The client has provided us with a sample data set to train our default machine learning model. The data includes the following attributes: crashes, average daily traffic (ADT), and length. Crashes are the number of crashes associated with a given observation. ADT is the average daily traffic recorded for a given observation. Length is the length of the road being observed. Since we have been tasked to predict crashes, ADT and length will be used as the predictors for our model and crashes will be used as the predicted values.

5.3.2 Exploratory Data Analysis

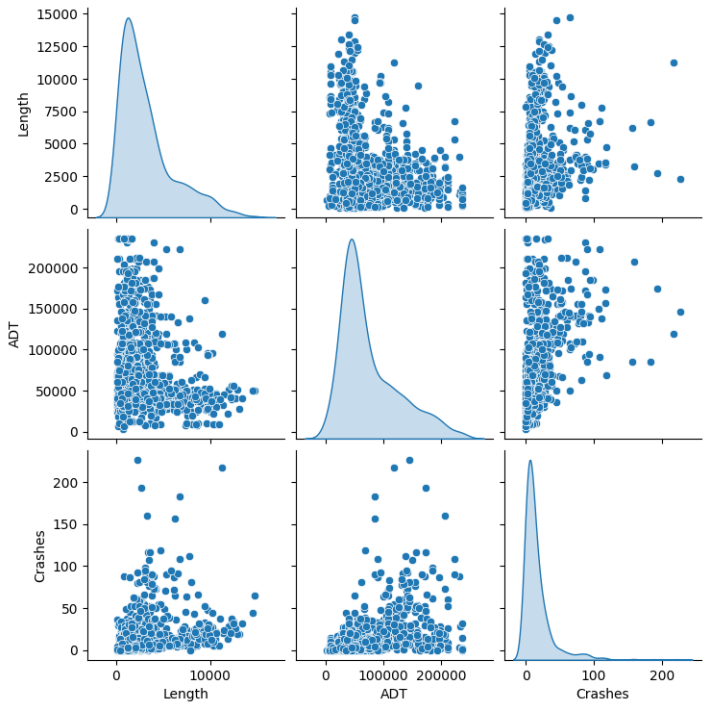

Figure 2: Raw data pair plots using Seaborn on training data from random 80-20 training-testing data split for exploratory data analysis

To begin machine learning model development, we have opted to conduct exploratory data analysis. The purpose of performing exploratory data analysis is to better understand the type of data we are working with. Furthermore, while our model selection was determined by the client, we thought it best to conduct exploratory data analysis to identify correlations among the data as correlation is important to the development of an effective regression model.

	Length	**ADT**	**Crashes**
Length	1.000	-0.251	0.251
ADT	-0.251	1.000	0.429
Crashes	0.251	0.429	1.000

Table 7: Raw data correlation matrix

As seen when plotting the raw data, there were minimal trends. Additionally, when viewing the correlation matrix for the raw data, we notice correlation values that would be categorized as weak correlations. Thus, we deemed data transformation to be a critical step in our machine learning model development.

Upon our realization of the need for data transformation, we had spent some time researching various data transformation. In discussion with the client, and for the purposes of this project, we have opted to perform a log transformation on the predictors of the dataset provided by the client.

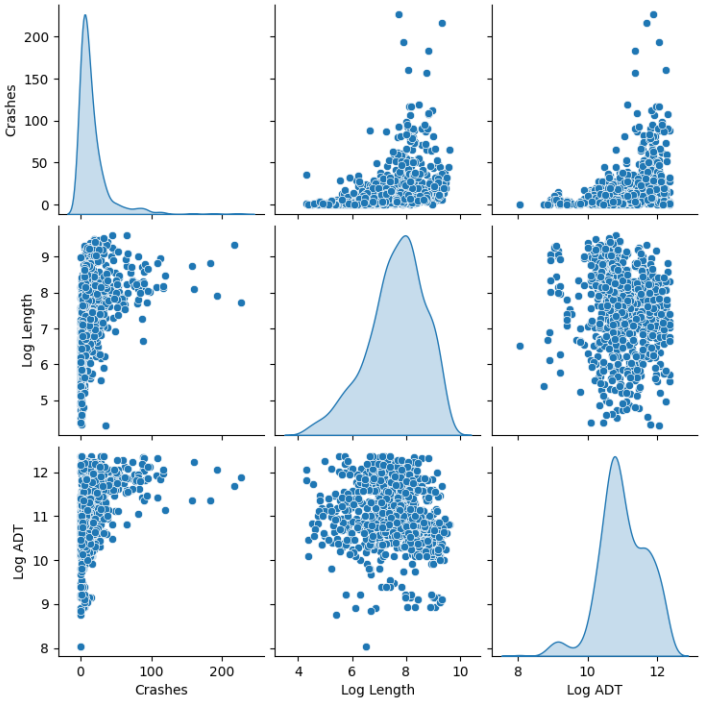

Figure 3: Log transformed data pair plots using Seaborn on training data from random 80-20 training testing data split for exploratory data analysis

	Log Length	**Log ADT**	**Crashes**
Log Length	1.000	0.301	0.412
Log ADT	0.301	1.000	-0.162
Crashes	0.412	-0.162	1.000

Table. 8: Log transformed data correlation matrix

As seen when plotting the log transformed data, there were noticeably more visible trends. Additionally, when viewing the correlation matrix for the log transformed data, we notice correlation values that remained in the category of weak correlations.

5.3.3 K-Fold Cross Validation

Now that the exploratory data analysis is complete, the next step was to train the model. First, we need to define splits the transformed dataset into a training, validation, and testing set. Splitting the data into training, validation, and testing contributes to the effective development of the regression model considering training is used to train the model itself, validation is used to tune the hyper parameters of the model, and testing is used to evaluate the model once trained.

We split the current dataset into 80% for training and the remaining 20% for testing. Of the 80% assigned for training, an additional split was performed such that 80% of the training data has been assigned for training and the remaining 20% of the training data has been assigned for validation.

K-fold cross validation is a cross validation approach utilized to improve model accuracy [2]. k is the number of splits in the original dataset. Setting k=5, the model was trained on 5 different 80-20 training-testing data splits.

5.3.4 Training Deep Neural Network Regression Model

For the development of the deep neural network model, the client has provided starter code to support the development of our default machine learning model [1]. The training labels and training data (log transformed) have been used to train the model. Mean absolute error (MAE) is used as loss function for model training given MAE is robust to outliers [6].

Specifically, the deep neural network regression model that has been trained utilizes four layers: one normalization layer, two 64 unit layers utilizing the Rectified Linear Unit (ReLU) activation function, and one 1 unit layer utilizing the ReLU activation function for output. The normalization layer is included as a data pre-processing measure to ensure model accuracy [3]. The two 64 unit ReLU layers serve as hidden layers to enhance robustness of the model. ReLU is used for the output layer to ensure positive output given the context of the data [5]. The model is iteratively trained over 9 epochs to efficiently increase the robustness of the model while avoiding over fitting utilizing the validation set [4]. The generated loss plot for the first of five data splits is as follows:

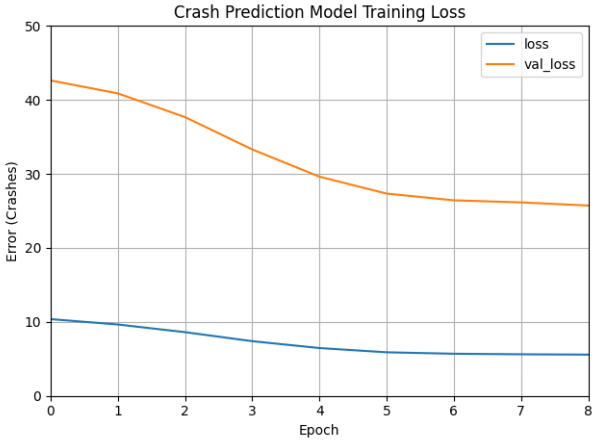

Figure 4: Training loss plot for the first fold.

Per client request, we have utilized mean squared error (MSE) and MAE for evaluating

the model. MSE and MAE are provided for each iteration of training completed. For future plans, please see the Future Works section.

Fold	MSE	MAE
1	274.941	8.090
2	259.603	7.608
3	542.765	9.550
4	570.756	10.117
5	362.758	9.715

Table. 9: MSE and MAE for each k-fold training iteration

6 Testing, Evaluation, & Assessment

Testing and evaluation plays a critical role in our project development to ensure our work meets the quality expectations of the client. Specifically, the following elucidates the extensive evaluation we have conducted for our machine learning model.

6.1 Machine Learning Model Evaluation

Evaluating Deep Neural Network Regression Model

In machine learning, evaluation of models must be complete before shipping the model for use because we must ensure the robustness of the integrated model. There are a handful of evaluation methods provided by the client that we have performed for our implemented model. Now that the model has been trained, we must test the model using the testing dataset that we had created prior to machine learning model development.

Ultimately, the model we have integrated will be used by the end users to make predictions. So, we must generate predictions using the testing labels and testing features to ensure an acceptable level of accuracy. To visualize the accuracy of our predictions, we have opted to generate a true value vs. predicted value plot:

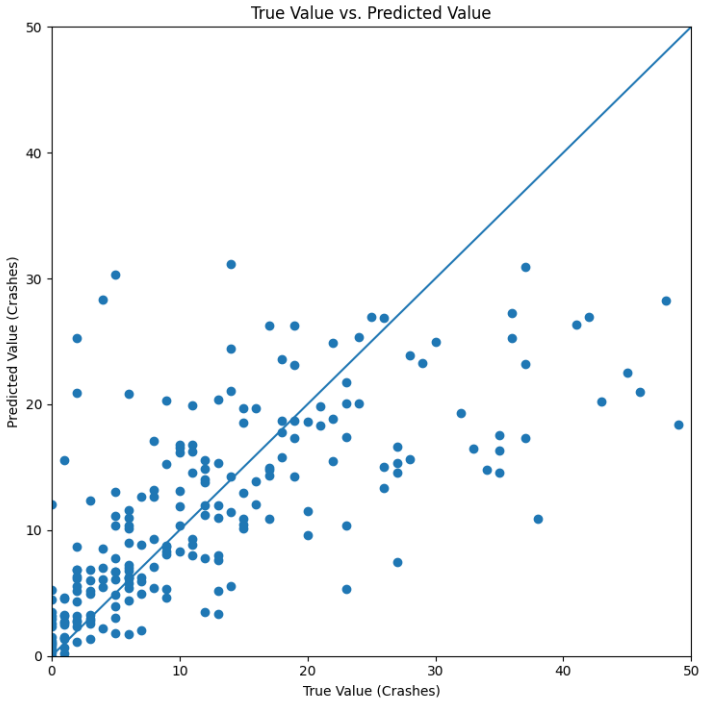

Figure 5: True Value (Crashes) vs. Predicted Value (Crashes) Plot

Ideally, all data points in the true value vs. predicted value plot will be will be on the y=x line. Evidently, our model is able to predict a portion of test values fairly accurately. However, many of the points are not plotted near the line suggesting the model can be improved. See the *Future Work* section for further details regarding model improvement. While the true value vs. predicted value plot did not present output we were hoping for, we have considered an additional evaluation visualization: prediction error distribution.

Generating a prediction error distribution will provide additional visualization for evalu-

ating our model. The prediction error distribution is a bar chart that displays the spread of error derived from generating prediction values from the test labels and features.

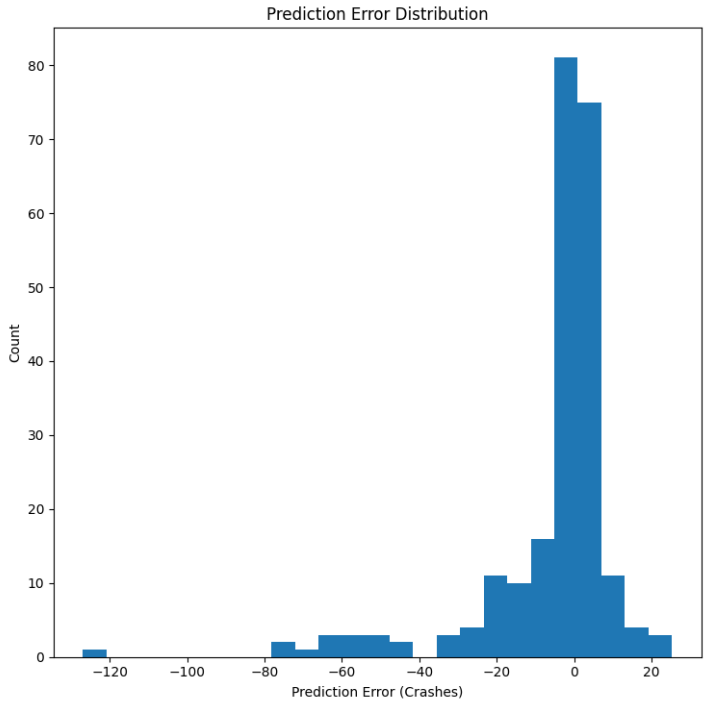

Figure 6: Prediction Error Distribution

In the case that all predicted values equals the the test values, the prediction error distribution will display a single bar at zero. Evidently, this is not the case. As seen in Figure 7, the amount of prediction error that was most frequent occurred near zero which is satisfactory. However, the prediction error distribution also suggests there is room for improvement. For example, prediction error generated using our model ranges from values less than -100 to values greater than 20. Assessing the MAE given Figure 5, the prediction error distribution is slightly left-skewed.

The prediction error distribution along with the MAE suggest that while the magnitude of the error produced by our model on average is relatively low (<11), our model does not predict a good portion of values properly.

6.1.1 Web Backend API Testing

The web backend's RESTful APIs were thoroughly tested to validate their functionality. An example of such testing is the user login endpoint, which authenticates users and issues JWT tokens.

Example Test: User Login Endpoint **Endpoint:** `POST /api/auth/login`

Description: Authenticates user credentials and issues a JWT token upon successful login.

Test Procedure:

1. Opened Postman and set up a POST request to `http://localhost:3000/api/auth/login`.

2. In the request body, selected the `raw` option and set the format to `JSON`.
3. Entered valid user credentials in the JSON payload
4. Sent the request and observed the response.
5. Verified that the response status code was `200 OK`.
6. Confirmed that the response body contained a success message and a JWT token
7. Tested with invalid credentials to ensure the endpoint returned appropriate error messages and status codes.

ML Backend API Testing

The ML backend's prediction endpoints were tested to ensure they provide accurate predictions and handle errors gracefully. An example test involves the single prediction endpoint, which processes input data and returns a prediction using a specified model.

Example Test: Single Prediction Endpoint Endpoint: `POST /predict/single`

Description: Generates a prediction for a single input using the specified machine learning model.

Test Procedure:

1. Opened Postman and set up a POST request to `http://localhost:8000/predict/single`.
2. In the request body, selected the `raw` option and set the format to `JSON`.
3. Entered the `model_id` and input data in the JSON payload.
4. Sent the request and observed the response.
5. Verified that the response status code was `200 OK`.
6. Confirmed that the response body contained the predicted crash rate.
7. Tested with an invalid `model_id` to ensure the endpoint returned an appropriate error message.
8. Tested with invalid input data (e.g., missing required fields) to verify error handling and response messages.

Conclusion

API testing using Postman allowed for comprehensive validation of both the web backend and ML backend services. By simulating various scenarios, including successful requests and error conditions, the robustness and reliability of the APIs were ensured. This testing process is essential for delivering a dependable application and provides confidence that the backend services perform as intended under different conditions.

6.2 Client Assessment

Given the timeline that has been approved by the client, our first major assessment was of the skeleton.

In our discussion of the skeleton during Presentation 1, we had discussed the implementation of the initial front end design, establishment of full stack data flow, and dummy machine learning model integration on the back end. Functionality and user experience includes model selection, data input, and rendering of prediction value.

The primary point made during the assessment is separation of parts. In our skeleton, all of the functionality was grouped on the home page. The client had suggested separating functionality such as having a home page, add prediction page, and collection details page for the user.

The assessment of our skeleton provided additional structure to the project development, supporting the development of our current web application design and integration.

7 User Manual

This section provides a guide for users on how to navigate and use the Traffic AI Crash Rate Predictor web application.

7.1 How to Use the Application

Starting the Application:

- **Install Dependencies**: Make sure to follow the **Developer Manual (8)** instructions for setting up the project. Specifically, after cloning the repository, run the following command in the project root directory to install all necessary dependencies:

```
npm install
```

- **Start the Application**: Once the dependencies are installed, start the application by running:

```
npm run dev
```

- **Access the Application**: Open your web browser and navigate to:

```
http://localhost:3000
```

- You should now be able to see the login page and begin interacting with the system.

Login/Signup Page

- **Access the Application:** Navigate to the login page by entering the application URL into your web browser.
- **Login:** Enter your email and password, then click the "Login" button to access the application.
- **Signup:** If you don't have an account, click on the "Sign Up" link and fill out the boxes with your username and password, and submit to create an account.

Homepage

- **Accessing Collections:**
  - After logging in, you will land on the homepage, which displays a table of your existing prediction collections.
  - For each collection, the table provides:
    * **Collection Name:** The name of the collection.
    * **Date Created:** The date and time the collection was created.
    * **Number of Predictions:** The total number of predictions in the collection.
    * **Actions:**
      · **View Button:** Click this to view the details of a collection, including its predictions.
      · **Delete Button:** Click this to delete the collection permanently.
- **Creating a New Collection:**
  - To create a new collection, click the 'Add New Collection' button.
  - You will be redirected to the 'New Collection' page where you can specify the collection's name.

7.1.1 New Collection Page

- **Creating a Collection:**
  - Enter a name for your collection in the text field provided.
  - Click the 'Create Collection' button to save the collection.
  - After creating the collection, you will be redirected to the Add Prediction Page to add predictions to the new collection.

Add Prediction Page

- **Adding Predictions:**
  - Select a machine learning model from the dropdown menu.
  - Choose how you want to input data:
    * **Manual Input:** Fill in the fields for Average Daily Traffic (ADT) and Length of the road segment.
    * **CSV Upload:** Use the file upload option to upload a CSV file containing batch predictions.
  - Check the "Apply Log Transform" checkbox to apply a logarithmic transformation to your input data.

  - Click the "Submit" button to process the data and generate predictions. The predictions will be saved to the associated collection.

View Prediction Collection

- **Viewing Collection Details:**
  - On the homepage, click the "View" button for any collection to access its details.
  - The collection page displays:
    * **Attributes:** Input data, such as Length and ADT.
    * **Result:** The predicted crash rate.
    * **Source Type:** Indicates whether the input was manual or from a CSV file.
- **Adding More Predictions:**
  - Click the "Add Prediction" button to return to the Add Prediction Page and add more predictions to the same collection.
- **Returning to the Homepage:**
  - Click the "Back to Collections" button to return to the homepage and view all collections.

Logout

- To log out of the application, click the "Logout" button located in the top-right corner of any page. This will log you out and redirect you back to the login page.

7.1.2 Login Page

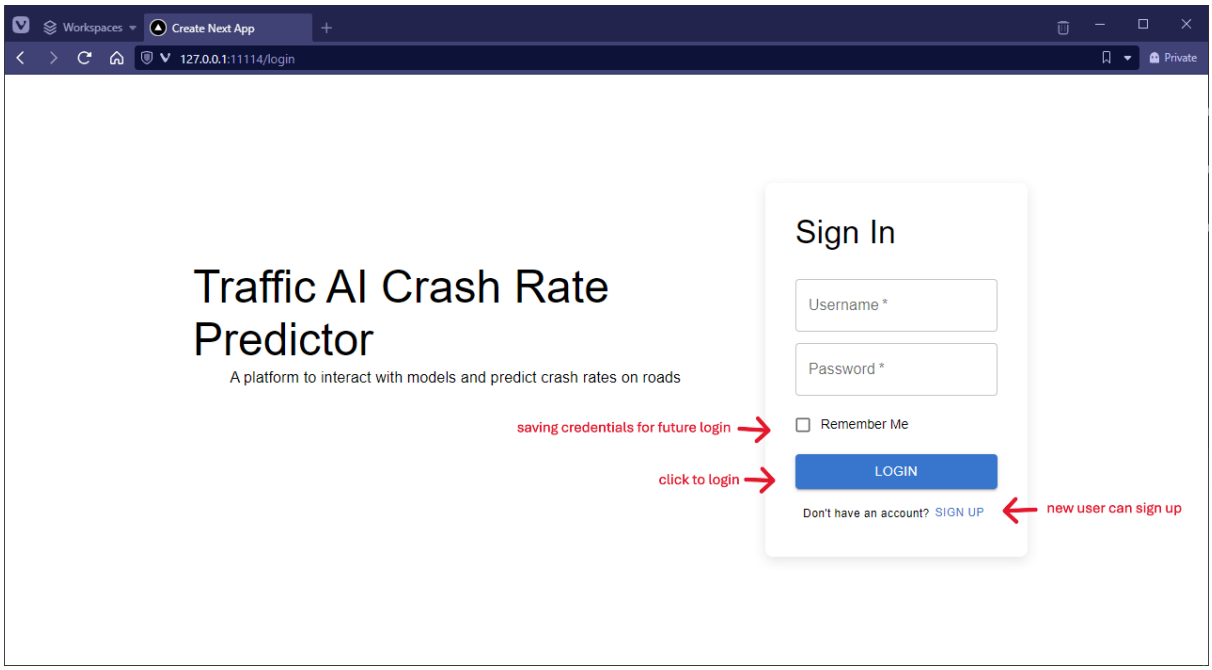

Figure 1: First login using your username and password or if you haven't already, sign up using the link.

7.1.3 Home Page

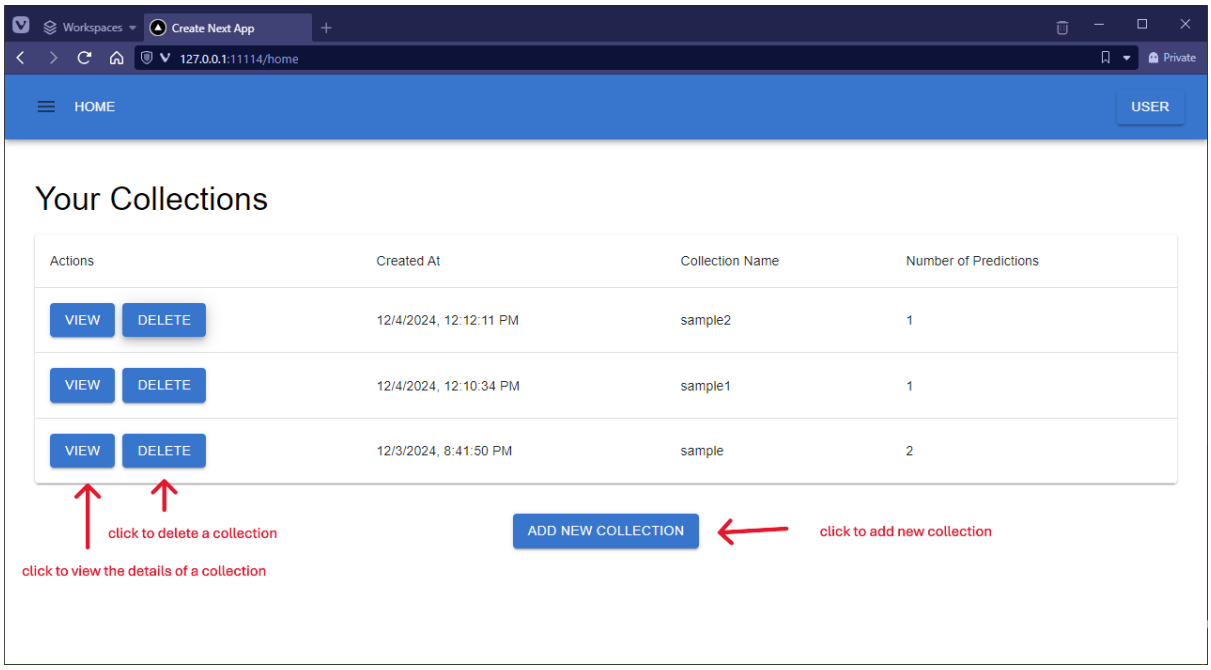

Figure 2: This is the Homepage after login showing past collections and options to add new collection. The user can also log out using the logout button

7.1.4 Create New Collection Page

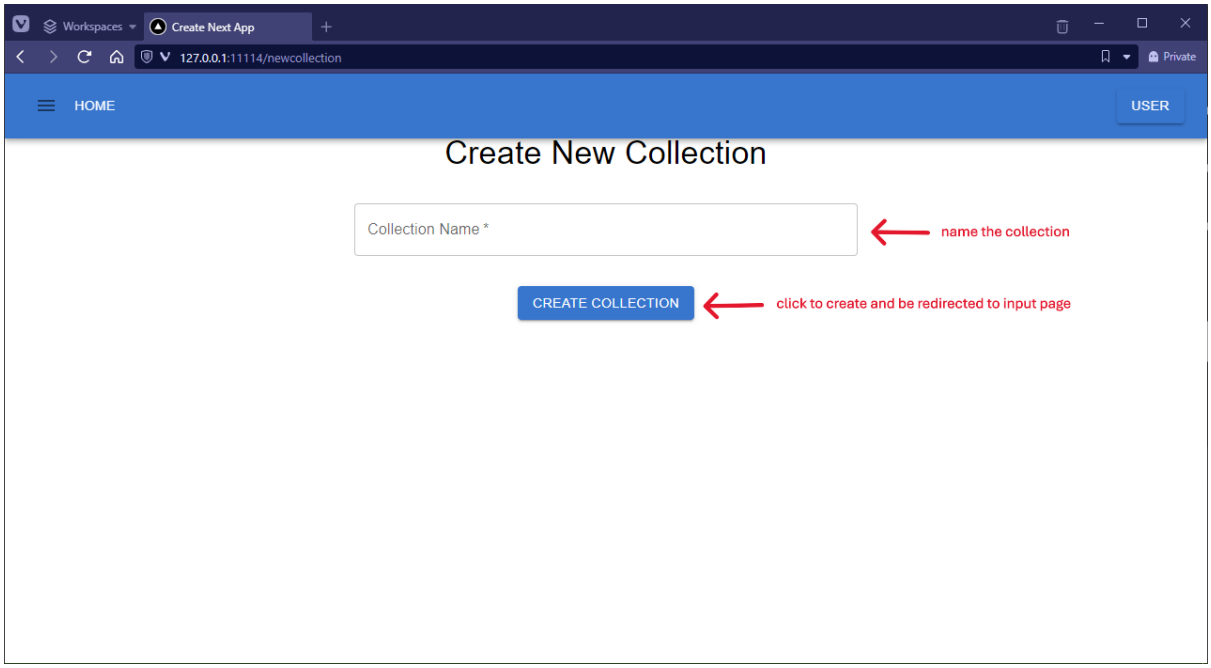

Figure 3: This is the Create New Collection - add text here

7.1.5 Input Pages

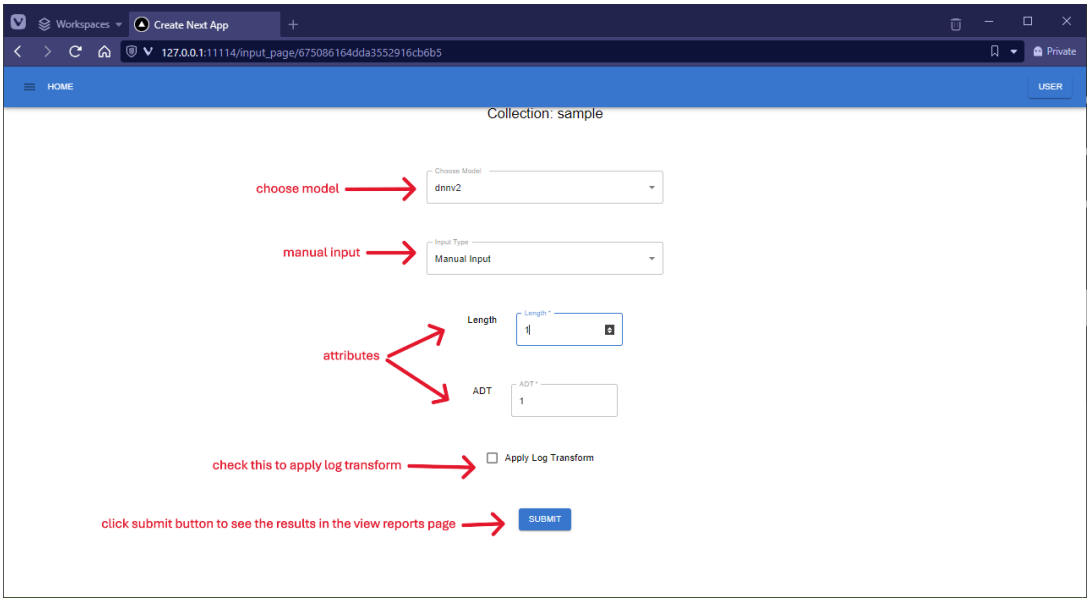

Figure 4: The user is then taken to the Add Prediction Page where they can enter the ADT and Length manually for a prediction.

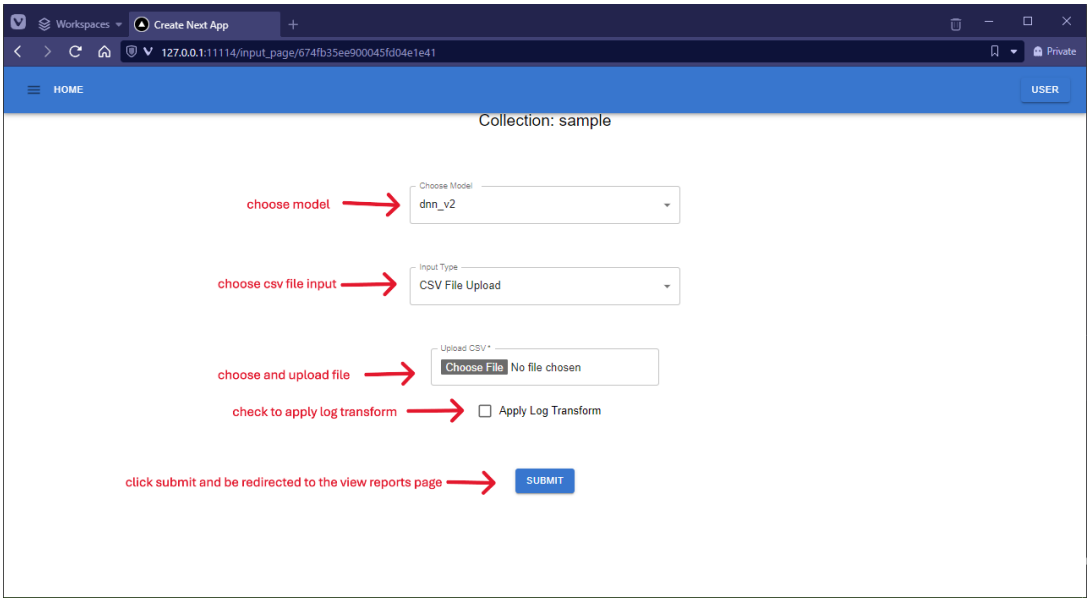

Figure 5: The user can also upload a CSV file with multiple entries for prediction in the input page. The User can also direct to the home page using the home button

7.1.6 Prediction Reports Page

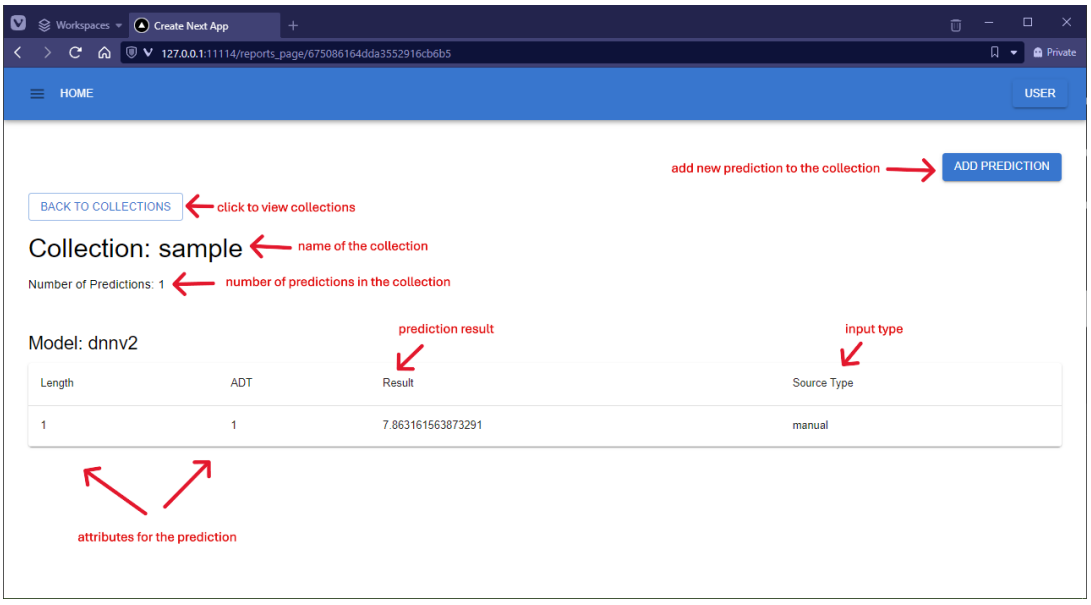

Figure 6: Through the home page the user can go to the View Reports Page to view detailed prediction results including the input, model, and predicted output.

7.1.7 Admin Privileges

When logged in as an administrator, users gain additional privileges to manage machine learning models and oversee system data efficiently.

Admin Home Page The Admin Home Page serves as the main dashboard for administrators. It includes the following features:

- **Navigation Menu:** The menu bar at the top of the page provides options to access:
  - *Model Management*: Navigate to add or view machine learning models.
  - *Input*: Add predictions to collections, similar to regular user functionality.
- **Collection Overview:** Similar to the user homepage, admins can view a table listing existing prediction collections. Admins can:
  - **View Collections:** Click the "View" button to see detailed predictions within a collection.
  - **Delete Collections:** Remove a collection and its associated data by clicking the "Delete" button.

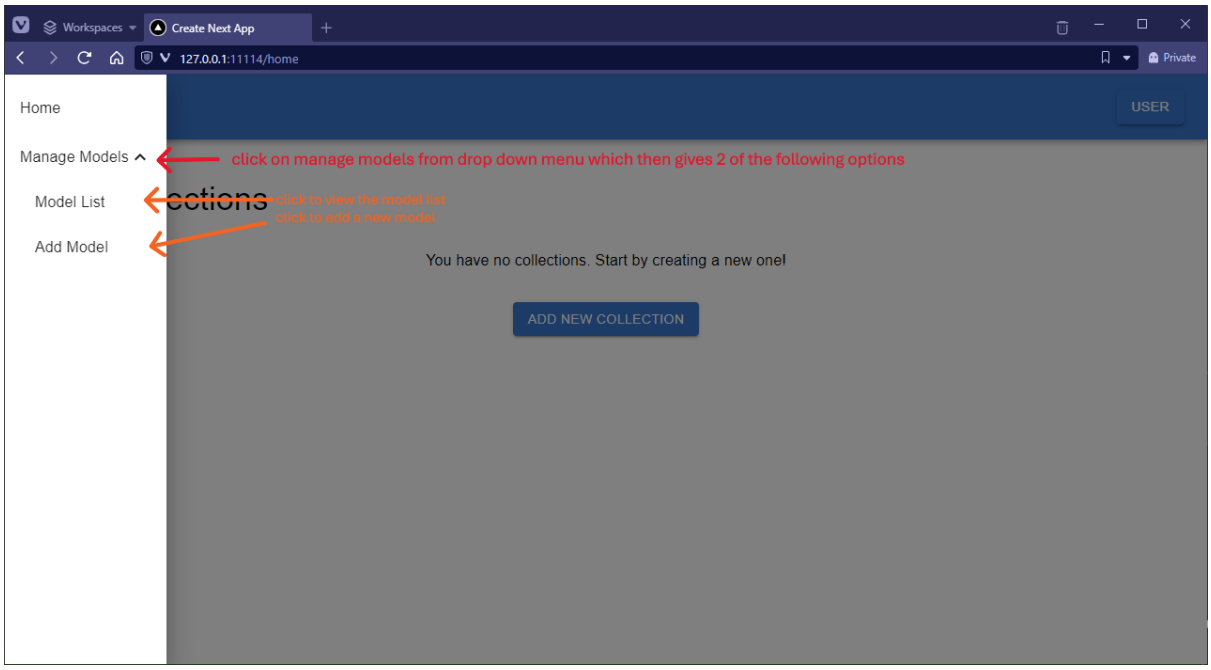

Figure 7: Admin Home Page showing the menu bar with options for Model Management and Input.

Model Management The Model Management section allows administrators to upload, configure, and manage machine learning models. It is divided into two primary functionalities:

Add Model To add a new predictive model:

- Navigate to the "Add Model" page from the navigation menu.
- Fill in the following fields:
  - **Model Name:** Enter a descriptive name for the model.
  - **Input Fields (Ordered):** Specify the required input attributes for the model in the correct order. Use the "Add Input" button to add additional fields.

  - **Upload Model File:** Use the file upload option to upload the model file in the accepted formats (.keras or .h5).

- Click the "Submit" button to save the model. Once added, the model will be available for users and administrators to use for predictions.

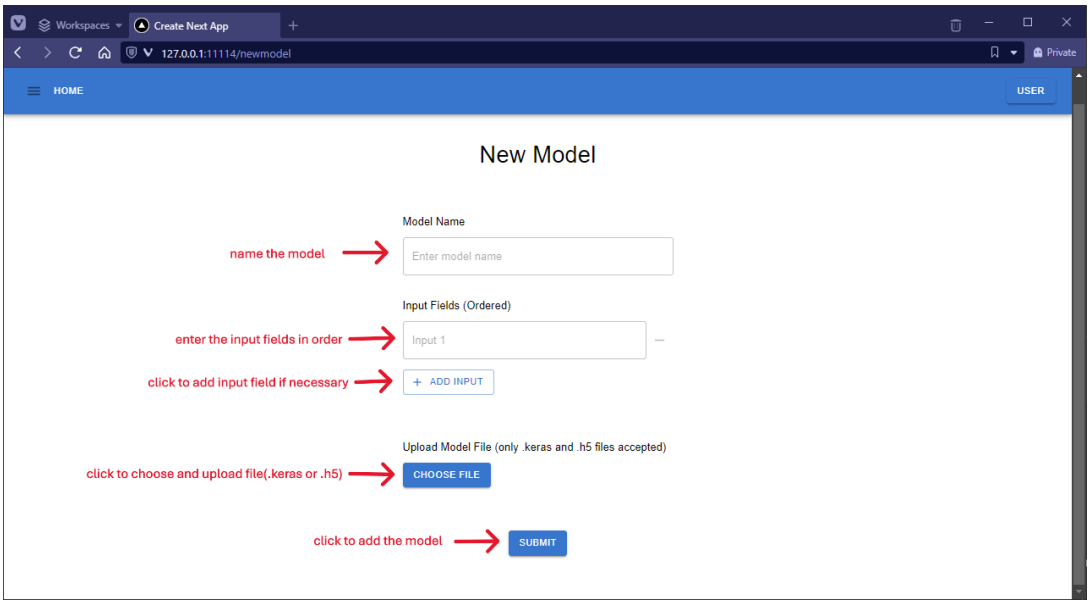

Figure 8: The Add Model page, where admins can upload and configure new predictive models.

Model List The Model List page provides an overview of all existing models in the system. Administrators can:

- View a table containing the following information for each model:
  - **Name:** The name of the model.
  - **Date Added:** The date and time the model was uploaded.
- Use the "Delete" button to remove outdated or unused models from the system.

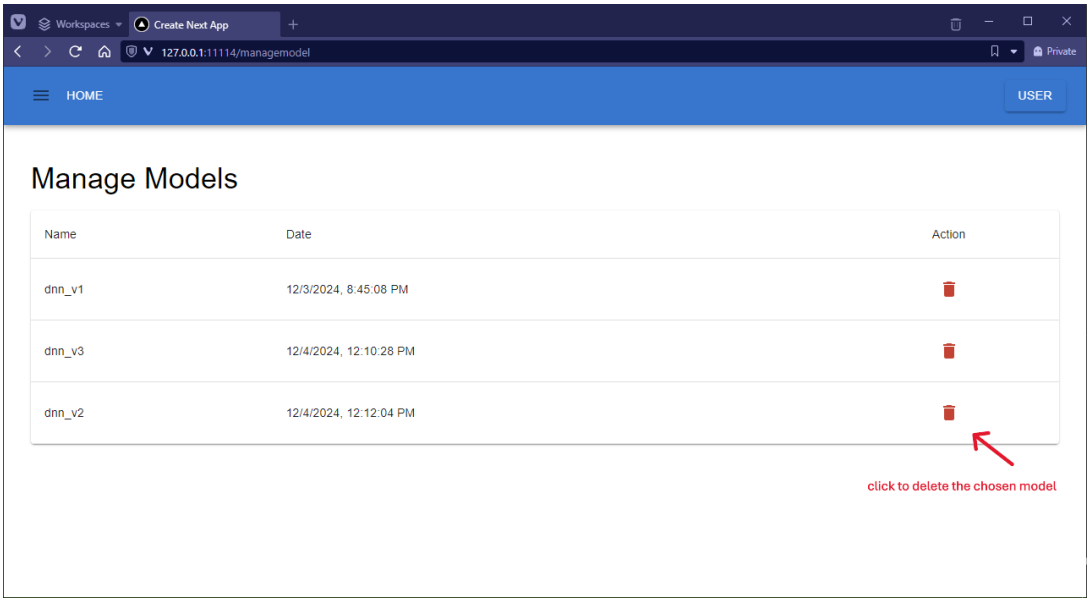

Figure 9: The Model List page, where admins can view and manage the system's available models.

8 Developer Manual

8.1 Developer Environment Installation

In this section, we will walk through installing the required software for a working developer environment.

Required software:

1. **Git**
2. **Docker**
3. **Python**
4. **Node.js**

8.1.1 Installing Git

Git is required to clone the project code to your local machine.

1. Windows / MacOS

   To install Git on Windows, download and install Git from git-scm

2. Linux

   Git can be found on most package managers through: `git`

8.1.2 Installing Docker

Docker is used to host a containerized version of MongoDB.

1. Windows / MacOS

   To install Docker, download and install Docker Desktop from the docker website

2. Linux

   Install Docker through these packages:

   ```
   docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
   ```

   If these packages are not listed on your package manager, please check the installation guide for more information.

8.1.3 Installing Python

Python is used to run the FastAPI framework.

Traffic-AI can be run with Python 2 or Python 3, however Python 3 is recommended.

1. Windows / MacOS

   Install Python through the python website. Latest version is recommended.

2. Linux

   Python 3 can be found through most package managers with: `python3`

8.1.4 Installing Node.js

1. Window / MacOS

   To install Node.js, download and install Node.js from the node.js website

2. Linux

   Node.js can be found through most package managers with: `nodejs`

8.2 Developer Environment Setup

This section will walk through downloading and running the project code.

8.2.1 Cloning the Repository

Clone the Traffic-AI repository onto your local machine.

```
git clone git@git.cs.vt.edu:jonathanle/Traffic_AI.git
```

The general project structure and respective software is as follows:

```
traffic-ai
├── FastAPI (Python)
├── MongoDB (Docker)
└── src (Node.js)
```

We will go through each of these to setup the required libraries to run them.

8.2.2 Setting up MongoDB

MongoDB is the database that the project uses for user login, user records, and model data storage. We are using a containerized version of MongoDB as part of our project. As such, the provided compose.yaml file under the MongoDB directory will set up and run everything required for MongoDB.

In other words, there is no setup required for MongoDB as long as Docker is installed on your system.

(More on running MongoDB later)

8.2.3 Setting up FastAPI

FastAPI is the python environment that runs the machine learning models. To set it up we need to install the required python libraries.

Note that if using Python 2, the following python commands will be `python`, `pip` instead of `python3`, `pip3`

First, change directory into FastAPI:

```
cd FastAPI
```

Create the Python virtual environment:

```
python3 -m venv ./venv
```

Ensure the Python virtual environment is activated on the terminal:

1. Windows (using powershell)

   ```
   ./venv/Scripts/Activate.ps1
   ```

2. Linux / MacOS

   ```
   source ./venv/bin/activate
   ```

Install the python packages for FastAPI:

```
pip3 -r requirements.txt install
```

8.2.4 Setting up Node.js

Node.js is the framework that runs all frontend website code. To set it up, we need to install the required Node.js packages.

Ensure you are in the traffic-ai root directory.

(If you are still in the FastAPI directory you can change directory to the traffic-ai directory with `cd ../`)

In the traffic-ai root directory, install the required npm packages:

```
npm install
```

8.2.5 Verifying proper setup

After installing everything, the project directory should be as shown.

New directories are marked with *

```
traffic-ai
├── FastAPI
│   └── venv*
├── MongoDB
├── node_modules*
└── src
```

8.3 Running the Developer Environment

To run the project, we need to run the three frameworks that we setup earlier.

Note that these frameworks need to be simultaneously running on their own separate terminals.

8.3.1 Running MongoDB

First, ensure you are in the MongoDB directory:

```
cd MongoDB
```

Then, start it up by running this command:

```
docker compose up
```

8.3.2 Running FastAPI

First, ensure that you are in the FastAPI directory:

```
cd FastAPI
```

Next, ensure that you have activated the python virtual environment:

1. Windows (using powershell)

   ```
   ./venv/Scripts/Activate.ps1
   ```

2. Linux / MacOS

   ```
   source ./venv/bin/activate
   ```

Finally, start FastAPI by running this command:

```
uvicorn app:app --reload
```

8.3.3 Running Next.js

First, ensure you are in the traffic-ai root directory, then run this command to start the frontend website:

```
npm run dev
```

8.4 Accessing the Website

After ensuring that all 3 frameworks are running, the website can be accessed through http://localhost:3000.

8.4.1 Changing Ports

The Traffic-AI project uses the 3 following ports for communication:

1. FastAPI - 8000
2. MongoDB - 27017
3. Node.js - 3000

If any of these ports are already taken by an existing service, we will have to change the port to an available one.

8.4.2 Changing the FastAPI Port

The FastAPI port can be changed by running the FastAPI framework with a new switch:

```
uvicorn app:app --reload --port PORT_NUM
```

8.4.3 Changing the MongoDB Port

The MongoDB port can be changed by modifying the docker compose file. This can be found in:

```
traffic-ai
├── FastAPI
├── MongoDB
│   └── compose.yaml*
└── src
```

Modify the compose.yaml file in this way. Change this line of code from:

```
ports:

  - 27017:27017
```

To:

```
ports:

  - PORT_NUM:27017
```

8.4.4 Changing the Node.js Port

First, open the **package.json** file.

This can be found in:

```
traffic-ai
├── FastAPI
├── MongoDB
├── src
└── package.json*
```

Next, modify the scripts section of the json file.

Change this line of code from:

```
"dev":  "next dev"
```

To:

```
"dev":  "next dev -p PORT_NUM"
```
(With PORT_NUM being your new port)

Then run Node.js:

```
npm run dev
```

The website should then be open on that PORT_NUM

8.5 Deploying a Production Build

To deploy a production build, we have set up a Dockerfile and a Compose file that will build a Docker image and run an unattended version of the project.

The files are located as shown:

```
traffic-ai
├── Dockerfile
└── compose.yaml
```

We recommend you become familiar with Docker since these deployments will most likely be on shared server space, but to help get started we have listed a couple of general commands:

1. List Images

   ```
   docker image ls
   ```

2. List Containers

   ```
   docker image ls
   ```

3. Build a New Image

   ```
   docker build -t traffic-ai .
   ```

   You must be in the same directory as the Dockerfile file so that docker recognizes what to run

4. Run the Latest Image

   ```
   docker compose up
   ```

   You must be in the same directory as the compose.yaml file so that docker recognizes what to run

   Again, the -d switch can be used to run an unattended version

NOTE: While you can technically build images and run them as a means for a developer environment, we highly recommend **NOT** to do this! If you change any part of code, you will have to rebuild the image for the changes to reflect on the Docker image. Docker is better used for deploying a final product to the server. Instead please use the **Developer Manual (8)** to setup a native developer environment.

8.5.1 Port Forwarding

To be able to get a response from the deployed docker container, you will have to open a port so that NextJS can respond to the request.

At the time, our reserved ports for the server were 11114, 41211, and 14121. However, depending on your situation, this may need to be changed.

1. **Expose the Docker Container's Port to the Server**

   Modify this line in the compose.yaml file:

   ```
   - 11114:3000 # NodeJS
   ```

   To:

```
- DESIRED_PORT:3000 # NodeJS
```

2. **Expose the Port to the Internet**

   Opening the port to the internet will depend on the Linux Distribution being used on the server. To find out what distribution is being used, run this commmand:

   ```
   cat /etc/os-release
   ```

   Some linux distributions may not use this file exactly but will have a file of similar name that will give you the distro name

   The linux distribution will be used to determine firewall implementation and the firewall commands that are used to open the port. At the time that we wrote this documentation, our server was running CentOS. So the command to interact with the firewall is: 'firewall-cmd'.

   At this point we assume your server is running CentOS.

   Before you open a port you must determine what the default zone is. This may be different so please verify before opening the port.

   You can list the default zone by running this command:

   ```
   sudo firewall-cmd --list-zones
   ```

   Then open the port, ensuring you specify the zone with the zone switch:

   ```
   sudo firewall-cmd --permanent --zone=public --add-port=DESIRED_PORT/tcp
   ```

   Generally the default zone is 'public', but for reference, please see Redhat Firewalld Guide

9 Known Bugs

This section lists bugs that we have discovered, but have not been able to resolve over the semester.

1. **Manual Input Prediction has Misaligned State**

   When the website only has one model uploaded, the manual input fields will not display when adding a prediction. We have found that this has something to do with the frontend state not updating properly on the page. Currently, the workaround is ensuring you have two models in the database, and switching back and forth between models during selection.

2. **Cookie Not Forwarding to User on Publicly Deployed Project**

   On a publicly deployed version of the project, the user's cookie does not get forwarded upon successful login, preventing access to the website. However this does not seem to be an issue when running the docker image locally, or ssh tunneling to the server that the project is deployed on.

3. **Deprecated Code Preventing Production Build**

   During creation of the Dockerfile, we encountered an issue preventing npm from creating a production build of our code. This is due to deprecated code being used in api/models/route.js. As of now, the Dockerfile gets around this by building the

image using 'npm run dev'. However, it would be preferential in the future to use a production build for the docker image.

10 Lessons Learned

10.1 Timeline

Week 2	Setting goals
Week 3	First client meeting, review ML notebook provided by client, start web dev front end design (UI). Finalize full stack design, decide on stack/tools/database.
Week 4	Start backend integration. Integrate ML model. Keep working on frontend.
Week 5	Complete the skeleton by adding login, signup and the initial home page. Add support features between backend and frontend. Keep working on ML backend.
Week 6	Continue working on Exploratory data analysis for client data Complete login page for the frontend and connect it to the database in the backend.
Week 7	Integrate linear/deep learning regression model using client data
Week 8	Work on user transactions log on the frontend and backend.
Week 9	Develop user collections log. Test and evaluate models. Add admin support.
Week 10	Default ML model saved and integrated into web backend Complete the extra frontend web.
Week 11	Finalize and integrate ML model and frontend enhancements.
Week 12	Complete full stack integration. Draft presentation materials.
Week 13	Complete presentation material and prepare for the presentation. Test and finalize the tool.
Week 14	Finalize presentation preparation
Week 15	Final presentations.

Table. 10: Timeline

10.2 Problems and Solutions

10.2.1 Data Transformation for Exploratory Data Analysis

Upon deciding to perform a log transformation on the raw data provided by the client, we had ran into some complications with applying the log transformation to the data.

Initially, we had only log transformed the predictor variables: Crashes, Log ADT, and Log Length. When exploring the data to confirm the effectiveness of the log transformation, the pair plots had shown promise as it had revealed new patterns in the data. However, when generating the corresponding correlation matrix, we had notices the correlation values remained weak.

Continuing to employ log transformation, our next intuition was to transform all of the data set attributes: Log Crashes, Log ADT, and Log Length. By performing such a data transformation, another problem arose when generating summary statistics for the dataset: the mean and minimum value for the Log Crashes attribute were both negative infinity. To understand why Log Crashes include negative infinity values, we must understand the nature of the log function: as x approaches zero from the positive x direction, y approaches negative infinity.

Upon further discussion with the client and for the purposes of the project, the solution we have integrated is to proceed with the original log transformation on the predictor attributes: Crashes, Log ADT, and Log Length.

10.2.2 Backend Machine Learning Architecture

Initially the group decided to use Next JS as the sole backend to avoid having to deal with multiple REST APIs. The initial implementation involved using the NodeJS child-process library to run a python script each time a user made a prediction request to the backend. However, we immediately ran into numerous issues, including dependency management and concurrency issues.

Although it seemed like a simpler approach initially, it was clearly more complex and difficult to scale, which led to the decision to add a FastAPI ML backend. This was implemented as an internal service to prevent adding complexity, and kept the ML backend separate from the frontend, solving the previous issues.

11 Future Enhancements

While the Traffic AI Crash Rate Predictor is fully functional in its current form, there are several opportunities to enhance its capabilities, usability, and scalability. These improvements aim to create a better user experience and make the system more robust and adaptable to real-world applications.

- **Improved User Interface (UI):**
  - **Enhanced Aesthetics:** Redesign the UI with a modern, visually appealing layout using advanced frameworks or libraries to ensure a polished, professional look.
  - **Interactive Navigation:** Add better navigation elements, such as tooltips, collapsible menus, and interactive buttons, to make the application more intuitive.
  - **Real-Time Feedback:** Implement real-time validation and feedback on input fields to improve user interaction and reduce errors.
- **Integration of Real-Time Traffic Data:** Incorporate APIs or IoT devices to fetch real-time traffic data from transportation authorities or monitoring systems. This feature would enable dynamic predictions, ensuring more timely and accurate insights for users.
- **Advanced Data Visualization:**
  - **Interactive Charts:** Introduce interactive visualizations for exploring data trends and understanding prediction outcomes.
  - **Geographical Mapping:** Use maps to display crash predictions,giving better insights for location-based analysis.
  - **Performance Metrics:** Add plots for comparing true values and predicted outcomes, helping users evaluate the model's accuracy.
  - **Improved Machine Learning Models:** Explore advanced ML techniques to improve prediction accuracy.
  - **Better Scalability:** Optimize the system to handle larger datasets and higher user volumes by:
    * Using efficient database queries and indexing for faster data retrieval.
    * Implementing caching mechanisms for commonly accessed data.

12 Acknowledgements

Thank you, Dr. Mohamed Farag, for supporting the development of this project. [7]

Client contact information: mmagdy@vt.edu (email)

13 References

References

[1] The TensorFlow Authors. *Regresssion Python Notebook*. URL: https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/regression.ipynb#scrollTo=GGn-ukwxSPtx. (accessed: 12.12.2024).

[2] GeeksforGeeks. *Cross Validation in Machine Learning*. URL: https://www.geeksforgeeks.org/cross-validation-machine-learning/. (accessed: 12.12.2024).

[3] GeeksforGeeks. *Data Normalization Machine Learning*. URL: https://www.geeksforgeeks.org/what-is-data-normalization/. (accessed: 12.12.2024).

[4] GeeksforGeeks. *Epoch in Machine Learning*. URL: https://www.geeksforgeeks.org/epoch-in-machine-learning/. (accessed: 12.12.2024).

[5] GeeksforGeeks. *ReLU Activation Function in Deep Learning*. URL: https://www.geeksforgeeks.org/relu-activation-function-in-deep-learning/. (accessed: 12.12.2024).

[6] IBM. *What is a loss function?* URL: https://www.ibm.com/think/topics/loss-function. (accessed: 12.12.2024).

[7] Virginia Tech Transportation Institute. *Mohamed Farag, Research Associate*. URL: https://www.vtti.vt.edu/staffdir/bio.php?pn=04208. (accessed: 12.12.2024).

[8] Microsoft. *Non-relational data and NoSQL*. URL: https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data. (accessed: 12.12.2024).