Date of publication September 10, 2025, date of current version October 23, 2025.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

TrafficWiz: A Database-Driven Tool for Analyzing Congestion and Safety in Nashville

HAIDER BAIG[1], MERHAWIT MEHARI[1], BENJAMIN SCHNEIDER[1], ZELALEM TEWELDE[1], DEMARICK WEBB-RIVERA[1]
[1]Middle Tennessee State University, Murfreesboro, TN

Corresponding author: Team TrafficWiz.

This IEEE document is part of the overall semester project for CSCI 4560 Fall 2025.

ABSTRACT Traffic congestion in Nashville and other major cities has become a constant challenge for city planners and commuters, often leading to wasted time and economic losses. There is an increased rate of accidents, resulting in higher stress levels for drivers and passengers alike. There are several existing approaches that try to help manage the problem. However, they often fail to integrate multiple data sources and/or provide dependable insights for both residents and city planners. To address this problem, we came up with the solution, our project, *TrafficWiz*, which aims to develop a database-driven analysis tool that combines real-world traffic data sets from the National Highway Traffic Safety Administration (NHTSA) and the Department of Transportation (DOT). The analysis tool uses MySQL for structured data storage, Python for queries and analysis, and JavaScript visualization libraries to highlight congestion patterns and accident hotspots. Our approach enables the creation of an interactive dashboard that presents traffic insights and predictive models in an accessible format. By integrating the multiple data sets from trusted sources into an intuitive dashboard, *TrafficWiz* aims to help decision makers improve traffic safety, reduce delays for common drivers, and improve traffic congestion to increase urban mobility in Nashville.

INDEX TERMS Traffic analysis, database systems, congestion, accident hotspots, MySQL, Python, visualization, Nashville

I. INTRODUCTION

URBAN traffic congestion in densely populated cities has become a topic of study in transportation research for smart city planning. Studies consistently show that traffic delays and accident frequency increase as road networks become saturated, leading to billions of dollars in lost pro ductivity each year [1]. Data-driven approaches, such as predictive modeling and traffic simulation, have been used to understand congestion trends with the end goal of guiding policy interventions to improve the flow of traffic. Research has shown that integrating real-time traffic conditions with historical accident data improves the accuracy of traffic flow forecasts, enabling more effective transportation management [2]. However, many current systems are fragmented, focusing on safety metrics or congestion analytics, without combining them into a unified user-accessible platform.

There have been several tools and projects that have attempted to provide traffic insights at the city or regional level. For example, applications such as Google Maps and Waze use crowd-sourced data and sensor data from their applications to provide route recommendations, while government agencies such as the Tennessee Department of Transportation (TDOT) release periodic reports on traffic patterns [3]. Other academic projects, such as those that use the Federal Highway Administration datasets, focus on large-scale analysis but often lack interactive dashboards tailored to specific communities [4]. Our project, *TrafficWiz*, builds on these efforts by integrating open data sets from the National Highway Traffic Safety Administration (NHTSA) and the Department of Transportation into a single relational database. Unlike existing solutions, *TrafficWiz* emphasizes displaying transparent facts regarding both safety (accident hotspots) and efficiency (congestion trends) through a dashboard interface, allowing local stakeholders, as well as Nashville residents, to visualize and make decisions using traffic insights from *TrafficWiz* in real time.

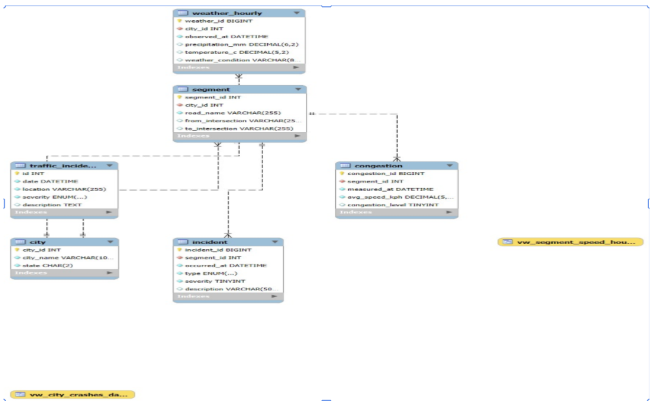

FIGURE 1. Entity–Relationship Diagram (ERD) generated from the TrafficWiz MySQL schema. The schema integrates accidents, vehicles, and traffic flow data through relational links.

II. DATABASE SCHEMA

The database is implemented using MySQL and made up of several different tables, including *Accidents* table, *Vehicles* table, and *Traffic_Flow* table. These tables are interconnected through primary and foreign keys to ensure that the data we retrieve remains consistent and is capable of supporting the complex analytical queries we ahve developed.

The *Accidents* table holds some of the applications most used infor- mation, such as accident ID, date, time, location coordinates, and severity level. The *Vehicles* table stores incident specific vehicle details, including make, model, and then associates them with the acci- dent table using a foreign key. The *Traffic_Flow* table captures congestion data from the Department of Transportation (DOT), including average speed, lane occupancy, and traffic density per time interval.

The relationships between these tables allow for an in-depth analysis using SQL queries. For example, a link between accidents and *Traffic_Flow* can reveal correlations between congestion levels and the frequency of accidents in certain zones or time periods.

III. SYSTEM ARCHITECTURE

The overall design of the system follows a three-layer architecture: the *Database Layer*, *Application Layer*, and *Presentation Layer*, as shown in Fig. 2.

- **Database Layer:** Contains the structured MySQL relational database, which stores all accident, vehicle, and congestion data. This layer ensures data integrity and supports concurrent access.
- **Application Layer:** Written mainly in Python, this layer performs SQL searches, cleans data, and aggregates data. It also serves as a binder, helping link the database and the front-end.
- **Presentation Layer:** Implements JavaScript and Web-based visualization frameworks to display traffic metrics in real time. The dashboard provides interactive maps and charts that are useful for city officials, traffic engineers, and residents alike.

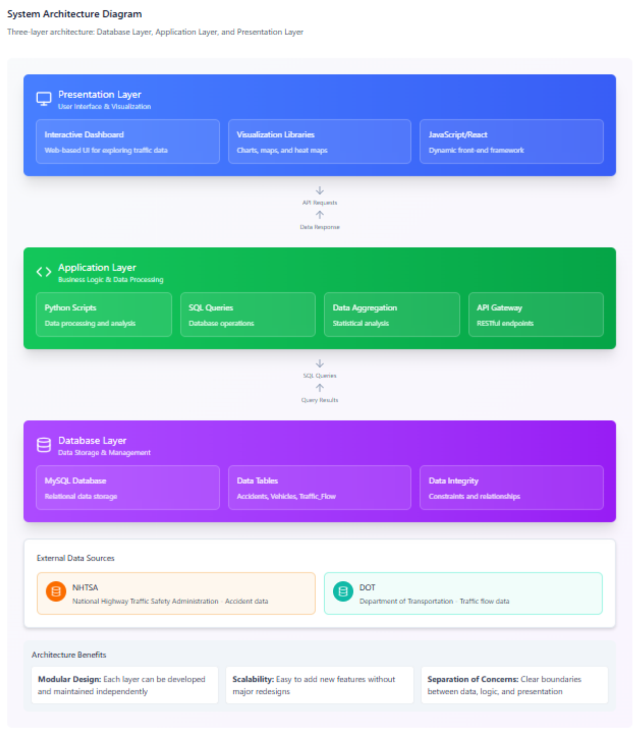
System Architecture Diagram
Three-layer architecture: Database Layer, Application Layer, and Presentation Layer
Presentation Layer
User Interface & Visualization
Interactive Dashboard
Web-based UI for exploring traffic data
Visualization Libraries
Charts, maps, and heat maps
JavaScript/React
Dynamic front-end framework
API Requests
Data Response
Application Layer
Business Logic & Data Processing
Python Scripts
Data processing and analysis
SQL Queries
Database operations
Data Aggregation
Statistical analysis
API Gateway
RESTful endpoints
SQL Queries
Query Results
Database Layer
Data Storage & Management
MySQL Database
Relational data storage
Data Tables
Accidents, Vehicles, Traffic_Flow
Data Integrity
Constraints and relationships
External Data Sources
NHTSA
National Highway Traffic Safety Administration · Accident data
DOT
Department of Transportation · Traffic flow data
Architecture Benefits
Modular Design: Each layer can be developed and maintained independently
Scalability: Easy to add new features without major redesigns
Separation of Concerns: Clear boundaries between data, logic, and presentation

FIGURE 2. System architecture diagram showing database, application, and presentation layers of TrafficWiz.

The architecture allows for modular development—future versions can integrate APIs for real-time traffic data or predictive AI modules without major redesigns.

IV. SYSTEM WORKFLOW

The system workflow begins with importing traffic and accident datasets from NHTSA and DOT sources. These files, often in CSV or JSON format, are pre-processed and normalized prior to being inserted into the MySQL database. Python scripts are used to clean null values, remove duplicates, and structure fields for consistency.

After processing, SQL queries and Python analytics scripts aggregate the data into usable insights, such as:

- Average accident severity per highway.
- Peak congestion hours by location.
- Correlation between vehicle types and accident occurrence.

The final results are visualized through an interactive dashboard that allows users to filter by date range, location, or vehicle type. Users can explore heat maps of accident hotspots, visualize traffic flow trends, and view predictive congestion forecasts.

V. MACHINE LEARNING INTEGRATION

To help push *TrafficWiz* beyond simply reporting the past, we wanted to add analytical features to our application. We decided to teach our application how to make traffic predictions, by integrating a Machine learning module. This

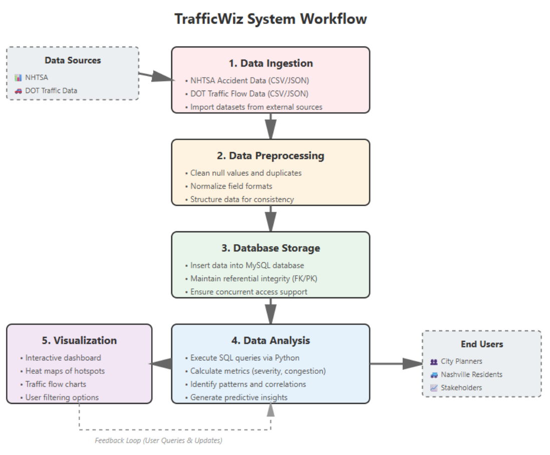
TrafficWiz System Workflow
Data Sources
NHTSA
DOT Traffic Data
1. Data Ingestion
• NHTSA Accident Data (CSV/JSON)
• DOT Traffic Flow Data (CSV/JSON)
• Import datasets from external sources
2. Data Preprocessing
• Clean null values and duplicates
• Normalize field formats
• Structure data for consistency
3. Database Storage
• Insert data into MySQL database
• Maintain referential integrity (FK/PK)
• Ensure concurrent access support
4. Data Analysis
• Execute SQL queries via Python
• Calculate metrics (severity, congestion)
• Identify patterns and correlations
• Generate predictive insights
5. Visualization
• Interactive dashboard
• Heat maps of hotspots
• Traffic flow charts
• User filtering options
End Users
City Planners
Nashville Residents
Stakeholders
Feedback Loop (User Queries & Updates)

FIGURE 3. Workflow diagram for TrafficWiz, showing the data ingestion, preprocessing, storage, analysis, and visualization stages.

model will allow us to not only show where current traffic incidents and risks are but also pinpoint and predict when and where they may occur in the future using the seed data and APIs.

Our model was developed primarily in Python using the *scikit-learn* library. The model uses the SQL files that store the NHTSA and DOT datasets to serve as a baseline for training the model. It uses time and location based filters such as hour of day, day of week, road segment, and average speed to estimate severity of future incidents and how often they happen. After trying multiple models, a Random Forest classifier model was chosen based off of its ability to handle the complex real world relationships and non-linear patterns that are featured in our datasets.

The model was trained using 2,600 past traffic incidents from the Nashville area. Over several iterations, the model learned to recognize the key predictors of accident severity and to identify high-risk road segments. The results reinforced common intuition: the combination of driving time, location, and congestion intensity are the strongest predictors of road safety risk.

To make this functionality accessible to users, the trained model was deployed using Flask and connected directly to the dashboard interface. This allowed real-time predictions to be displayed as a dynamic heat map, showing areas of elevated risk across the city. As a result, the machine learning module transformed *TrafficWiz* from a reactive data viewer into a proactive decision-support tool—helping commuters and city planners visualize and mitigate future congestion and accident risks across Nashville's roads.

VI. RELATED WORK

Research on traffic analysis, urban congestion, and roadway safety has expanded significantly in recent years. Many studies investigate traffic behavior using loop detectors, crowd-sourced mobility data, sensor networks, or GPS tracking from mobile devices. These approaches have enabled deeper understand- ing of congestion patterns, commuter habits, and bottleneck formation within metropolitan regions.

Commercial navigation applications such as Google Maps and Waze rely on real-time user data and proprietary algorithms to estimate travel times and recommend optimal routes. While highly effective for daily navigation, these systems do not generally provide transparent safety analytics or open dashboards that allow communities to interpret trends in their local environments. Their focus is route optimization rather than understanding accident patterns or long-term structural issues in traffic flow.

Public-sector tools provide more structured but less interac- tive information. For example, the Tennessee Department of Transportation (TDOT) publishes periodic traffic and safety re- ports summarizing high-level statistics for state highways [3]. These reports are informative for researchers and planners but lack fine-grained tools for dynamic exploration or location- specific queries. The National Highway Traffic Safety Ad- ministration (NHTSA) and Federal Highway Administration (FHWA) publish rich datasets used widely in academic studies, yet they require considerable preprocessing and integration before becoming actionable for local stakeholders.

A growing body of academic work applies machine learning to model accident risk or predict congestion under varying conditions. Approaches include random forests, gradient boosting, spatiotemporal neural networks, and probabilistic graph models. While many studies report strong performance, they are often implemented as isolated experiments and do not integrate into accessible dashboards for the public.

TrafficWiz positions itself between these existing streams of work: it uses authoritative government datasets similar to FHWA and NHTSA, but packages them inside a practical dashboard tailored specifically to Nashville. It also incorporates machine learning predictions into a transparent interface, aiming not only to analyze past events but to support proactive decision-making.

VII. DATA PIPELINES AND ETL METHODOLOGY

Real-world traffic data rarely arrives in a clean or uniform format. Agency datasets differ in naming conventions, timestamp formats, coordinate precision, and missing-value behavior. TrafficWiz uses a multi-stage extract-transform-load (ETL) pipeline designed to consolidate heterogeneous sources into a structured MySQL database.

A. SQL-BASED ETL FOUNDATION

The foundation of the system is an SQL schema defined through schema.sql, which creates all core tables, primary keys, foreign keys, and constraints. A corresponding seed_data.sql file populates tables with sample accident records, congestion readings, and road metadata. Procedures and triggers enforce internal consistency, such as automatically computing incident durations or verifying that every accident references a valid road segment.

This SQL-first approach ensures that analytic queries behave predictably before real-time ingestion is introduced. It also supports repeatable experimentation by providing a determin- istic baseline dataset.

B. PYTHON-BASED ETL FOR LIVE AND EXTERNAL DATA

Python scripts extend the system to ingest ongoing or external data sources. The here_data_collector.py script retrieves congestion measurements from the HERE Traffic API, including average speed, jam factor, and traffic density for specific Nashville road segments. Other scripts ingest seed data or simulate conditions for demonstration purposes.

These scripts apply transformations that include:

- renaming inconsistent fields,
- converting timestamps into uniform time zones,
- resolving coordinate mismatches,
- mapping external road IDs to internal database keys,
- trimming invalid or duplicate records.

After transformation, cleaned rows are inserted into the database using prepared SQL statements. Over time, this pipeline can be extended to support continuous ingestion or scheduled collection via Windows Task Scheduler.

C. DATA CLEANING CHALLENGES

During development, several non-trivial data issues were encountered. Some datasets used different latitude/longitude precision formats, while others occasionally reported missing severity levels or placeholder numeric codes. In some cases, congestion speeds were represented as strings, and weather fields included mixed categorical and numeric descriptors.

To address these issues, we relied on Pandas for type coercion, missing-value imputation, and consistency checks. Outliers were trimmed using simple interquartile filters, and timestamps were normalized to hourly resolution to align with congestion and weather datasets. These steps significantly improved model quality and ensured uniformity throughout the analytical pipeline.

D. WEATHER AND TRAFFIC FUSION

Weather has a well-documented influence on accident likelihood. As part of the expanded ETL pipeline, TrafficWiz plans to integrate hourly weather readings from the National Weather Service (NWS) API or Weather.gov endpoints. Weather fields such as precipitation intensity, visibility, temperature, and wind speed will be merged with road segments and time intervals to allow queries such as:

> "How do crash frequencies and severities vary be- tween clear and rainy conditions across major inter- states?"

Integrating weather into the pipeline enhances both descriptive analytics and predictive modeling, supporting richer downstream insights.

VIII. SYSTEM IMPLEMENTATION DETAILS

Although the high-level architecture of TrafficWiz consists of three main layers, several practical implementation choices strongly influenced system usability, performance, and maintainability.

A. BACKEND API

The backend is implemented in Flask and exposes REST endpoints for data retrieval and model inference. Endpoints such as /incidents, /traffic, /risk, and /weather provide JSON responses that the frontend can visualize. Each endpoint performs:

- input validation to ensure query parameters are safe and well-formed,
- SQL execution against the MySQL database,
- optional interaction with the trained Random Forest model, and
- formatting of responses into lightweight JSON structures.

The backend also handles environment variables and API keys through separate configuration files, preventing accidental leakage of credentials.

B. FRONTEND INTERFACE

The user-facing dashboard is implemented in React and styled using TailwindCSS. Visualizations use Recharts for charts and mapping libraries for geographic overlays. The interface includes:

- Dashboard: summary of recent incidents by severity,
- Incidents Page: searchable, sortable listings of accidents,
- Risk Page: recommended travel times and risk estimations by road,
- Live Traffic: real-time congestion overlays using HERE API data.

React components communicate with Flask via a proxy defined in the Vite configuration, simplifying development by routing requests through /api paths without requiring hardcoded URLs.

C. AUTOMATION AND STARTUP TOOLS

To streamline the workflow, a Windows batch script (start.bat) automates environment setup. The script ver- ifies the presence of a Python virtual environment, installs dependencies if needed, creates a scheduled task for periodic data collection, starts the Flask server, and launches the frontend. This reduces friction for demonstrations and ensures that the entire stack runs consistently.

IX. CASE STUDY AND SAMPLE ANALYSIS

To demonstrate the capabilities of TrafficWiz, we conducted a preliminary case study on major Nashville roadways. Using seeded and externally ingested datasets, we analyzed how congestion and accidents vary across time and locations.

Morning peak congestion typically occurred between 7:00 and 9:00 AM, particularly around interstates leading into downtown Nashville. Evening congestion peaked around 4:00 to 6:00 PM. These findings align with expected commuter patterns but also highlight specific road segments where speed drops sharply under moderate volume, indicating structural bottlenecks.

We also compared weather conditions with incident data. Early results showed that rainy and foggy conditions were associated with higher frequencies of medium-severity incidents on certain roadways. When merged with congestion data, these patterns produced meaningful predictive signals for the Random Forest model.

The machine learning model produced intuitive predictions: segments with high congestion levels, complex merge pat- terns, or historical incident density tended to show higher predicted risk. For example, certain portions of I-24 displayed consistently elevated risk scores during morning rain events. These insights demonstrate how TrafficWiz supports scenario- focused questions such as:

> "Which road segments are most vulnerable under Monday morning rain conditions?"

X. FUTURE WORK

TrafficWiz serves as a solid foundation for integrating open datasets, predictive analytics, and interactive visualization into a unified traffic analysis platform. Several opportunities exist for future improvements.

A. REAL-TIME DATA INTEGRATION

A major next step is completing full real-time integration of weather, congestion, and incident reports. This would allow the system to update predictions continuously and support live dashboards or alerts.

B. ADVANCED PREDICTIVE MODELS

Future versions may explore gradient boosting, recurrent neural networks (RNNs), or spatiotemporal graph neural networks (GNNs) to better capture dependencies across time and space.

C. CLOUD AND PUBLIC DEPLOYMENT

Deploying TrafficWiz to a cloud platform such as AWS or Azure would enable city planners and residents to access the dashboard remotely. Cloud deployment would also support automated scaling and scheduled ETL tasks.

D. MOBILE AND BROWSER EXTENSIONS

A mobile app or browser extension could surface predictions or safety insights directly within navigation tools such as Google Maps, enhancing accessibility for everyday drivers.

E. USER FEEDBACK AND CONTINUOUS LEARNING

Incorporating user-submitted feedback—such as reports of construction zones, unusual congestion, or temporary closures could create opportunities for hybrid human–ML workflows that refine predictions over time.

Overall, TrafficWiz aims to evolve from a classroom project into a genuine decision-support tool for the Nashville community, combining data, visualization, and prediction to improve urban mobility and roadway safe

REFERENCES

[1] Texas A&M Transportation Institute, *2021 Urban Mobility Report*. College Station, TX, USA, 2021.

[2] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban Computing: Concepts, Methodologies, and Applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 3, pp. 1–55, 2014.

[3] Tennessee Department of Transportation, "Traffic and Safety Reports," 2023. [Online]. Available: https://www.tn.gov/tdot

[4] Federal Highway Administration, "Highway Statistics Series," 2023. [Online]. Available: https://www.fhwa.dot.gov/policyinformation/statistics.cfm

[5] D. Helbing, "Traffic and Related Self-Driven Many-Particle Systems," *Reviews of Modern Physics*, vol. 73, no. 4, pp. 1067–1141, 2001.

[6] A. Doshi and M. Trivedi, "Attention Operators for Driver Assistance Systems: Issues, Review, and Future Directions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 2, pp. 596–614, 2011.

[7] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] S. Qiu and J. Ning, "Investigating the Impact of Adverse Weather on Traffic Safety," *Journal of Advanced Transportation*, vol. 2018, pp. 1–12, 2018.

[9] HERE Technologies, "HERE Traffic API Documentation," 2024. [Online]. Available: https://developer.here.com

[10] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.