REAL-TIME PUBLIC TRANSIT CONGESTION MONITORING IN TRENTO: BusFlow WebApp

- *Virginia Di Mauro* – virginia.dimauro@studenti.unitn.it
- *Luisa Porzio* – luisa.porzio@studenti.unitn.it
- *Anthony Tricarico* – anthony.tricarico@studenti.unitn.it

Big Data Technologies Class – Year 2025
Professor Daniele Miorandi and Stefano Tavonatti

Submission Date: 12-06-2025

ABSTRACT

BUSFLOW

01 THE PROJECT

The aim of this project was to build a big data architecture able to ingest data relative to public transportation, GPS feeds, events, traffic and weather data, in order to predict the most congested lines, hours and stops. This would help the agency managing the public transport to adjust and redirect where needed the personnel and buses resources.

02 KEY GOALS

- Generate **synthetic data** simulating real-time stream data
- Build an **architecture integrating real-time stream data** from different sources
- Select appropriate technologies (Kafka + spark, dask) to manage **streaming analytics**
- Implement a **predictive model** that forecasts bus congestion rates
- Implement **dashboard** to visualize the forecast and deriving insights

03 RESULTS

- **Comprehensive Pipeline Setup**: A fully functional data pipeline integrating data ingestion, processing, storage, analysis and visualization.
- **Machine Learning Model Implementation**: Development and deployment of machine learning models with tracking capabilities via MLflow.
- **Interactive Visualization**: Creation of an interactive application using Streamlit to visualize data insights and model outputs.
- **Containerization**: Use of Docker and docker-compose for containerizing the application, ensuring consistent deployment environments.

PROBLEM STATEMENT AND MOTIVATION

THE PROBLEM

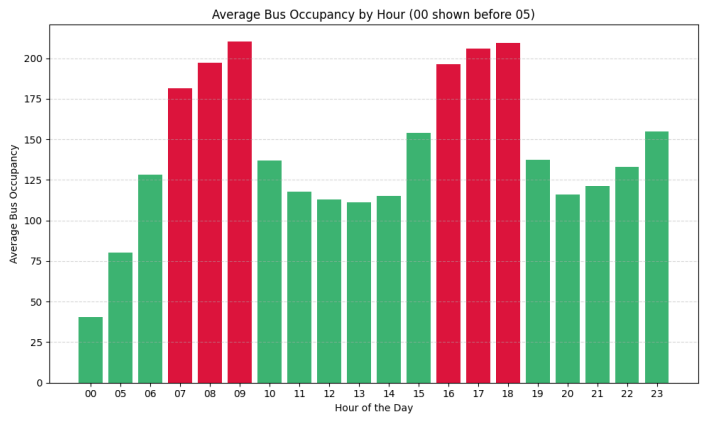
This project has been developed with the main goal of providing support to authorities managing public transport schedules. The need for this system comes from the current absence of a bus congestion monitoring structure in Trento, that takes into account also social and logistic factors. Organization's managers are not able to make informed decisions on the number of buses to allocate to each line at different hours of the day, leading to excessive crowding, wrong allocation of resources and commuters' complaints.

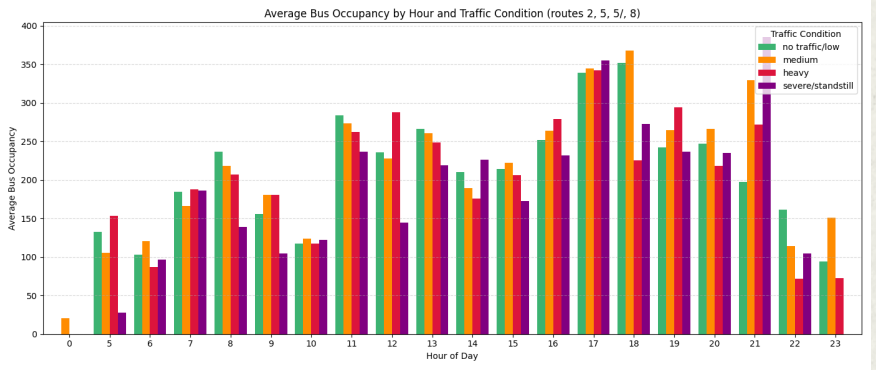
WHY IT MATTERS

Data-driven insights stemming from this big data system can provide easy access to real-time congestion levels of all bus lines in Trento. This can ultimately solve commuters' discontent and decrease their distress, making moving across Trento easy and thus improving customer satisfaction with the service.

DATA EXPLORATION INSIGHTS

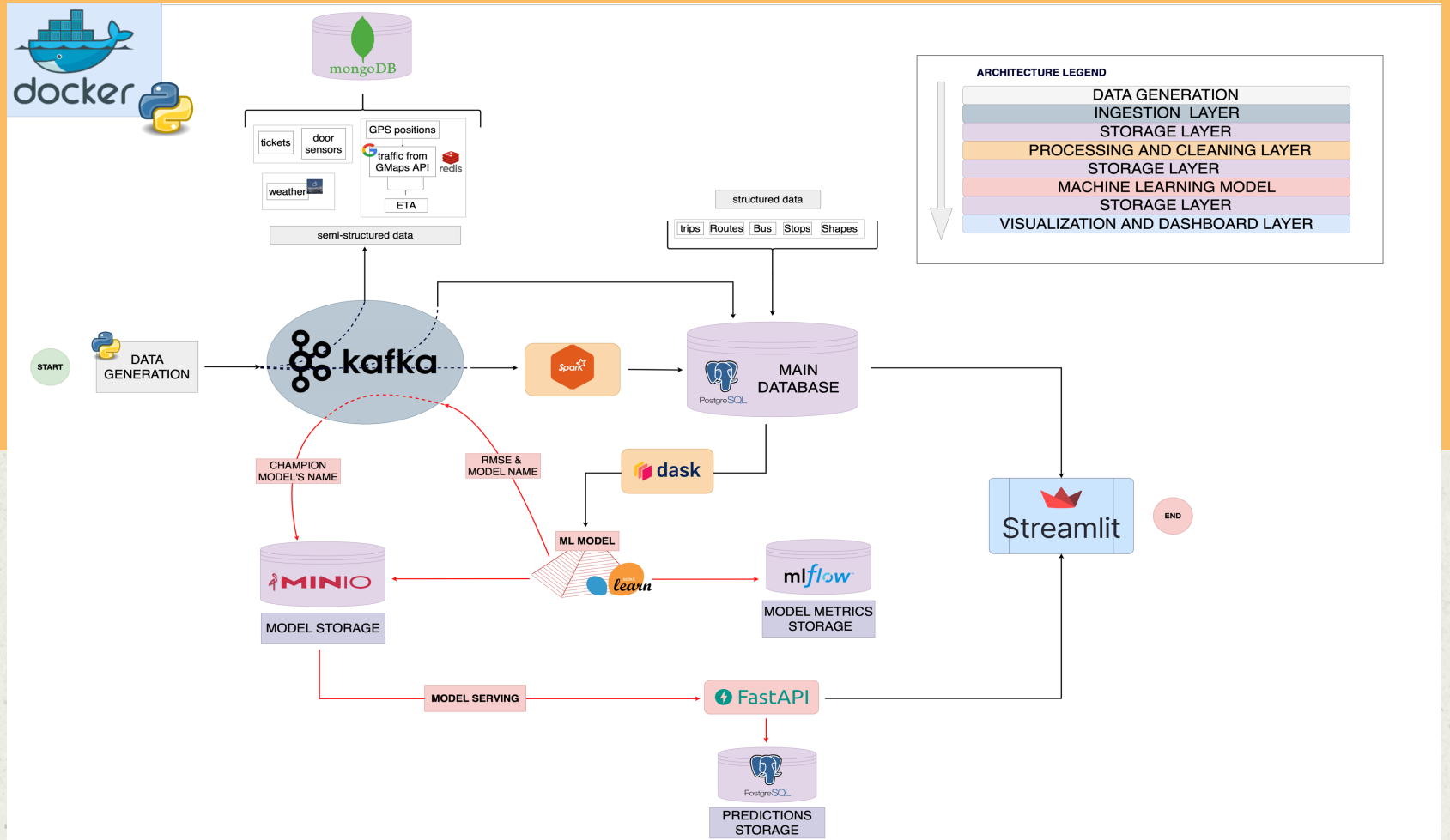
- **Passenger demand peaks during rush hours,** between 7–9 AM and 4–6 PM. This is more prominent on routes near schools and hospitals.
- **Weather conditions affect ridership**: colder and rainy days show increased bus usage compared to sunny days.
- **Traffic severity significantly impacts travel time**, with “severe” traffic doubling the expected time between stops.
- **Synthetic data reflects real-world behavior** thanks to fine-tuned generation logic (e.g., more passengers during events or peak hours).



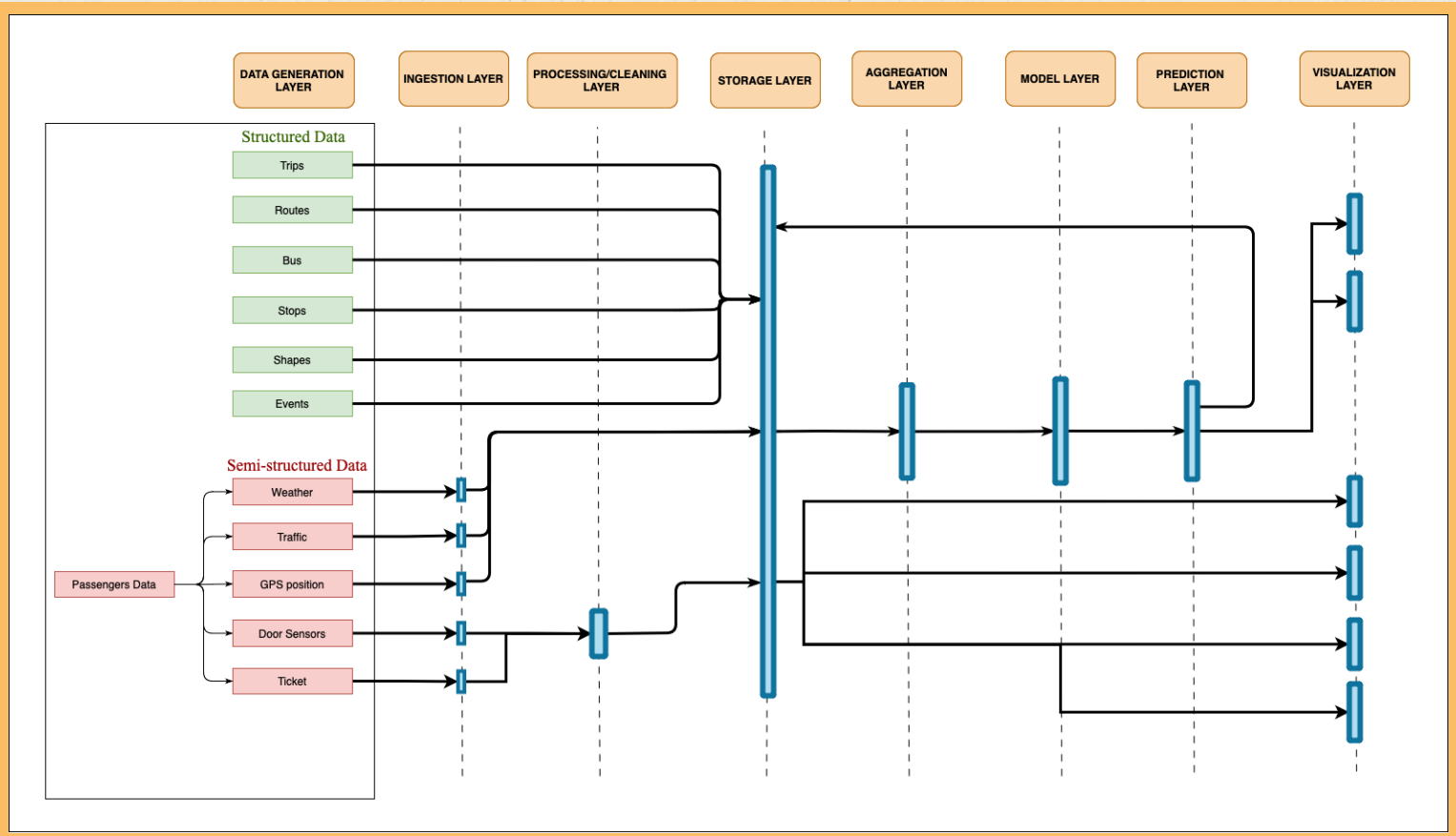


SYSTEM ARCHITECTURE DIAGRAM

BUSFLOW



SYSTEM ARCHITECTURE: how data flows



TECHNOLOGIES AND JUSTIFICATION

BUSFLOW

CORE TECHNOLOGIES

Python

Main programming language

-> Chosen for its flexibility, rich ecosystem and seamless integration with data science tools, it was used throughout the application to develop microservices, implement machine learning models, manage data workflows and build the Streamlit dashboard.

Docker

Containerization of all services

-> Chosen for its ability to ensure portability, isolation and ease of deployment across systems, Docker was used to containerize every component (Kafka, FastAPI, PostgreSQL, etc.) and orchestrate them using docker-compose for consistent local and production setups.

REAL-TIME DATA STREAMING

Redis

Used to control API request rates in memory

-> Selected for its speed and simplicity as a key-value store, Redis was used to limit the rate of API calls (e.g., to Google Maps) and prevent request overloads by caching recent queries and apply usage limits.

Kafka

Distributed message broker

-> Chosen to enable reliable, real-time communication between producers and consumers, Kafka was used to stream data (e.g., GPS, passenger count, traffic, etc.) between services and support scalable, asynchronous data ingestion. It was also chosen for its integration with Apache Spark.

BACKEND & MICROSERVICES

FastAPI

RESTful microservices (e.g., prediction, ML model retraining)

-> Selected for its high performance and asynchronous support, FastAPI was used to develop microservices such as the congestion prediction API, enabling fast responses and seamless integration with frontend and streaming components.

TECHNOLOGIES AND JUSTIFICATION

BUSFLOW

DATA PROCESSING & ANALYTICS

Dask

Lightweight parallel computing for SQL-like transformations

-> Chosen for its ease of use and familiar syntax similar to pandas, Dask was used to perform fast prototyping and aggregations on large datasets during ML pipeline development, replacing PySpark where needed for greater flexibility.

Spark

Distributed processing of streaming data

-> Chosen for its scalability and Kafka integration, Spark was used to consume real-time streams, apply schemas to raw data and write structured outputs to PostgreSQL.

STORAGE & DATABASES

PostgreSQL

Relational database for processed/aggregated data

-> Chosen for its reliability and support for complex queries, PostgreSQL served as the main structured database, storing cleaned and aggregated data for use in dashboards, analytics and ML model training.

MongoDB

NoSQL database for raw sensor data

-> Selected for its flexible, schema-free design, MongoDB was used to store data acting as a persistence layer, allowing ingestion of heterogeneous formats without upfront schema definitions.

MACHINE LEARNING

MLflow

Tracking model metrics + ML utilities + model training

-> **MLflow**: Chosen to support MLOps practices and used to track training cycles.

SK-Learn

-> **SK-Learn**: Chosen to compute the model metrics and for model training utility functions.

XGBoost

-> **XGBoost**: Used to train boosted regression model due to its more memory efficient DMatrix data structure used which allows for faster retraining

VISUALIZATION

Streamlit

Web dashboard development

-> Chosen for its ability to quickly build data-driven web apps with a Python backend, Streamlit was used to develop a user-friendly dashboard for transit authorities to monitor congestion, predictions and analytics in real time.

IMPLEMENTATION & CODE REPOSITORY

HOW TO RUN THE APP LOCALLY (QUICK INSTALLATION)

1. Clone the repo from GitHub: `git clone https://github.com/tricarico672/BDTProject.git`

2. Once inside the BDTProject directory run:
   `docker compose up --build -d`
   This will start up all containers

3. Check out the streamlit dashboard by connecting to http://localhost:8501 using Google Chrome as a browser (recommended to avoid visual glitches)

Check out the full README.md document on GitHub to get more info on additional ways you could run the project

Technology	Role/Purpose
Python	Primary programming language used across all components
FastAPI	REST API development for various microservices (prediction service, sensors consumer, etc.)
Streamlit	Frontend dashboard development and data visualization
Apache Kafka	Message broker for real-time data streaming between components
Apache Spark	Distributed data processing and streaming analytics
PostgreSQL	Primary database for storing processed data and analytics
MongoDB	NoSQL database for storing raw data
MinIO	Object storage for ML models
XGBoost	Machine learning model implementation for congestion prediction
Docker	Containerization and orchestration of all services
Altair	Interactive data visualization in the dashboard
PyDeck	Geospatial data visualization
SQLAlchemy	SQL toolkit and ORM for database operations
MLflow	ML model lifecycle management and metrics storage
Scikit-learn	Machine learning utilities and preprocessing
Uvicorn	ASGI server for FastAPI applications
Dask	Parallel computing and large dataset handling
Boto3	AWS SDK for Python, used with MinIO
Redis	In-memory data store for API rate limiting in traffic data collection

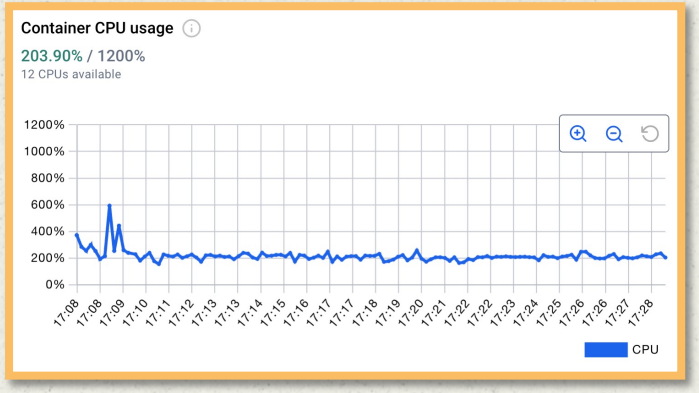
```
./
├── data/                    # Data storage directory
├── kafka-components/        # Kafka producers and consumers
│   ├── kafka-consumer-gps/
│   ├── kafka-consumer-model/
│   ├── kafka-consumer-passengers/
│   ├── kafka-consumer-sensors/
│   ├── kafka-consumer-tickets/
│   ├── kafka-consumer-traffic/
│   ├── kafka-consumer-weather/
│   ├── kafka-producer-gps/
│   ├── kafka-producer-passengers/
│   ├── kafka-producer-sensors/
│   ├── kafka-producer-tickets/
│   ├── kafka-producer-traffic/
│   ├── kafka-producer-weather/
│   └── utils/               # Shared Kafka utilities
├── ml-model/                # Machine learning model files
├── mlflow/                  # MLflow tracking server
├── mong/                    # MongoDB configuration
├── notebooks/               # Development and testing notebooks
│   └── utils/              # Notebook utilities
├── postgresql/              # PostgreSQL configuration
├── prediction-service/      # Real-time prediction API
├── spark-components/        # Spark processing modules
├── src/                     # Source code utilities
├── streamlit/              # Dashboard application
│   ├── .streamlit/          # Streamlit configuration
│   ├── images/              # Dashboard images
│   ├── modules/             # Dashboard modules
│   ├── main.py             # Main dashboard entry
│   ├── page1.py            # Dashboard pages
│   ├── page2.py
│   ├── page3.py
│   ├── page4.py
│   ├── page5.py
│   ├── page6.py
│   └── manager.py          # Dashboard state management
├── docker-compose.yml       # Container orchestration
├── .gitignore              # Git ignore rules
└── README.md               # Project documentation
```

GITHUB REPO LINK: https://github.com/tricarico672/BDTProject

RESULTS AND PERFORMANCE

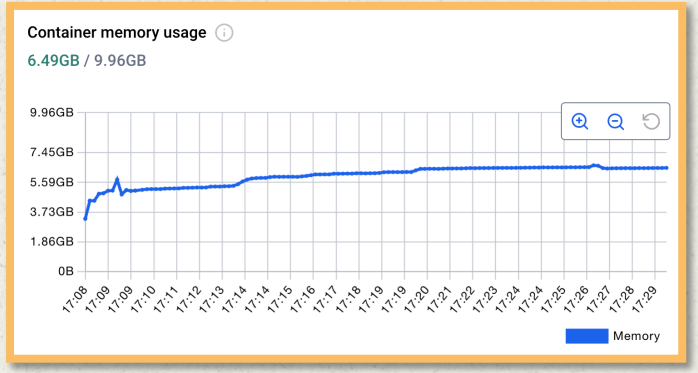
1 CPU usage

In our testing we noticed that at times, due to the speed at which data is generated and shared across components, the load on the CPU would get too high leading some processes to quit unexpectedly and containers to crash at times. To stabilize this problem, we introduced an environment variable called `SLEEP` which makes sure that the data is generated after waiting a constant number of seconds (or fraction thereof) thus artificially slowing down the data generation process.



2 Memory usage

During our testing phase, we monitored container memory usage over time to ensure the system's stability under load. Memory usage remained within acceptable limits, showing that the application can handle high data throughput without exhausting system resources.



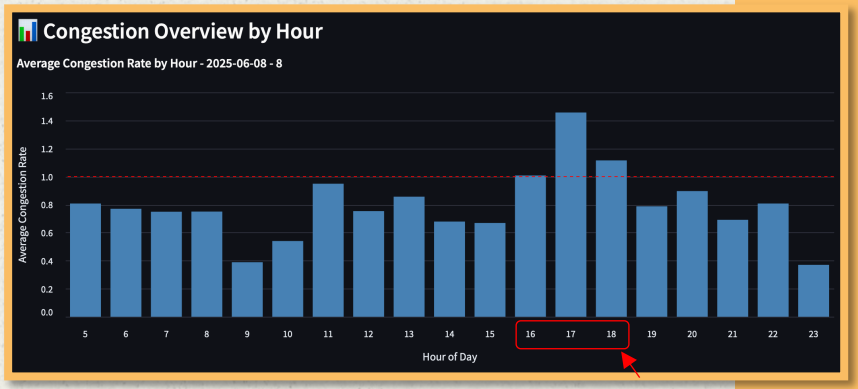
RESULTS AND PERFORMANCE

3 Real-Time Congestion Tracker

This page allows to visualize the results from real-time recorded congestion on buses and guides authorities in the allocation of buses on different routes.

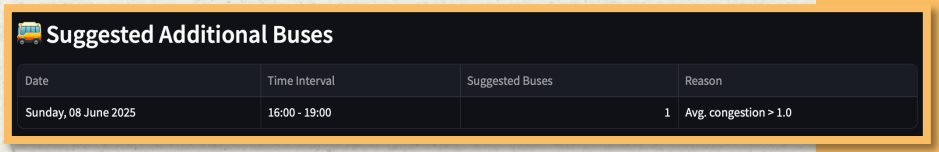
Example: Route 8, 08 June 2025

- Data is collected in **real-time**, filtered by route and date.
- Time interval affected: **16:00–19:00.**
- Peak congestion occurred at **17:00.**
- Result: Congestion level exceeded the defined threshold (avg. > 1.0).
- System triggered an alert and recommended **1 additional bus.**



Key Features

- Dynamic congestion bar chart
- Automatic resource suggestions
- Threshold-driven decision support
- Fully filterable by route and date



RESULTS AND PERFORMANCE

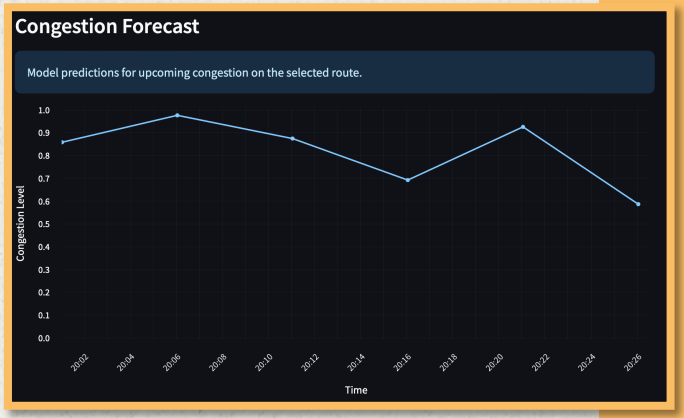
BUSFLOW

4 Forecast & Prediction

Forecast Simulation

This page allows to visualize forecasts for a specific route as different parameters are varied. For instance, it is possible to set a different temperature, change the probability of precipitation or change the weather code representing the weather conditions.

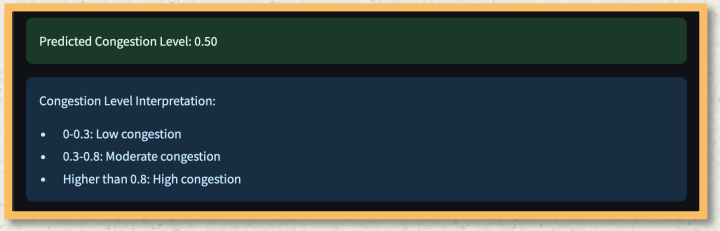
- Example (Route 2):
  - Average congestion = **0.82**
  - It triggers recommendation: **add buses** or **reroute traffic**

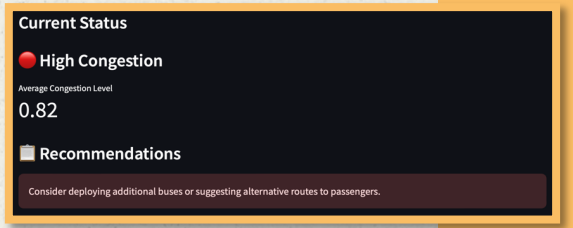


Prediction

This page allows to exploit the trained model to see its outputs as different parameters are varied. It offers more flexibility compared to the forecasts page as it also allows for the arbitrary selection of any time of the day.

- Example (Route 2):
  - **8 June 2025, 10:00**
  - Predicted congestion = **0.50**
  - **Moderate** congestion





RESULTS AND PERFORMANCE

BUSFLOW

5 Anomalies

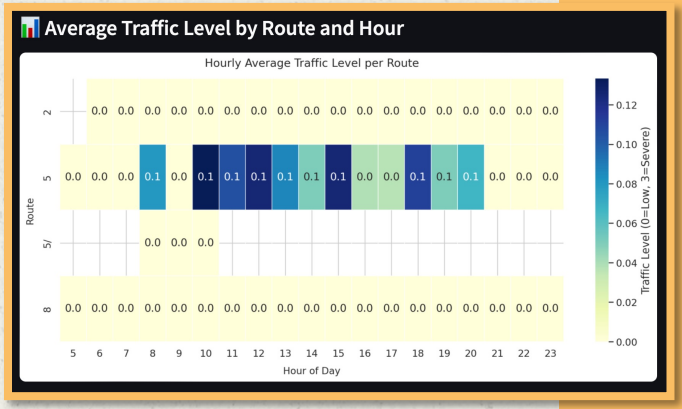
This page shows the main bottlenecks caused by traffic on each different route by combining heatmap analysis and time-series trends.

Heatmap Insights

- The heatmap shows **average traffic intensity** by route and hour.
- Example
  - Route 5 shows increased traffic levels between **08:00 and 14:00**, especially around 10:00.

Trend Line Insights

- The **line plot** tracks traffic **trends** over time.
- Example
  - The line chart confirms **no anomalies**, but Route 5 consistently shows higher average traffic.



LESSONS LEARNED

BUSFLOW

MAIN CHALLENGES

- Data Generation Process (Logic and Limitations of LLMs for data generation)
- Spark implementation (considering time constraints)
- Maintaining an idea of the big picture while working on single components

DEVELOPMENT INSIGHTS

- Make sure to finalize fully one component as it might be hard to come back to fix it later down the road as more dependencies among components are created as development progresses

WHAT WORKED WELL

The proposed system, despite its limitations, has the ability to scale well overall and with future updates can become even more resilient. The main components that would scale well include:

- MongoDB, Kafka, Spark, Dask, Redis, MINio (if scaling to S3 Bucket on AWS).

WHAT DID NOT WORK

The main drawback of the current implementation is represented by the absence of Spark for data aggregation. This did not allow us to introduce a proper WH architecture and might put too much pressure on a single PostgreSQL DB which might need to scale vertically at some point, making it a more expensive choice in the long-run.

LIMITATIONS AND FUTURE WORK

BUSFLOW

01 LIMITATIONS

- Dask to manage SQL aggregations
- Model retraining and comparison to current champion
- Lack of dimensional modeling and optimization for faster queries in the “Data Warehouse”

02 POTENTIAL IMPROVEMENTS

- Add PySpark support for data aggregations and SQL writes of cleaned data
- Add representative test set used to compute RMSE of all models
- Retrigger model retraining based on performance over time
- Implement a proper DW with support for fast complex queries

03 WHAT WOULD BREAK FIRST WHEN THE SYSTEM SCALES, WHY?

Some aggregations steps were carried out using conversions to pandas DataFrames and this might cause troubles when data that is too large is loaded into memory, thus ML Model retraining could potentially break.

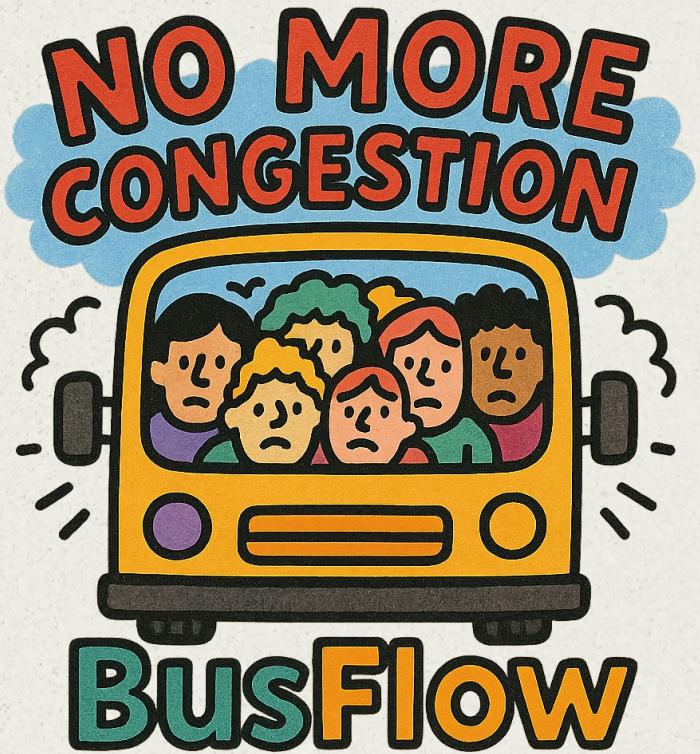
REFERENCES AND ACKNOWLEDGMENTS

ACADEMIC REFERENCES

- [1] K. M. Nissen et al., "How does weather affect the use of public transport in Berlin?" Environ. Res. Lett., vol. 15, no. 8, p. 085001, Jul. 2020, doi: 10.1088/1748-9326/ab8ec3.
- [2] M. Wei, J. Corcoran, T. Sigler, and Y. Liu, "Modeling the Influence of Weather on Transit Ridership: A Case Study from Brisbane, Australia," Transportation Research Record, vol. 2672, no. 8, pp. 505–510, Dec. 2018, doi: 10.1177/0361198118777078.
- [3] J. Feng, X. Li, B. Mao, Q. Xu, and Y. Bai, "Weighted Complex Network Analysis of the Different Patterns of Metro Traffic Flows on Weekday and Weekend," Discrete Dynamics in Nature and Society, vol. 2016, no. 1, p. 9865230, 2016, doi: 10.1155/2016/9865230.

ADDITIONAL REFERENCES

- Events in Trentino
- MongoDB vs Redis
- Explanation for GMaps API behavior
- Data Visualization Ideas for Traffic Data
- Dask vs Spark Benchmarks
- Data from Trentino Trasporti
- Graphic Map in Streamlit



THANK YOU FOR YOUR TIME AND ATTENTION!

BUSFLOW

For further qustions or clarifications you can contact us at:

- *Virginia Di Mauro* – virginia.dimauro@studenti.unitn.it
- *Luisa Porzio* – luisa.porzio@studenti.unitn.it
- *Anthony Tricarico* – anthony.tricarico@studenti.unitn.it