Real-Time Traffic-Aware Routing for Manhattan Using OSMnx and HERE Traffic Data

Junhao Qu(jq2434),Boxiong Li(bl3155)

***Abstract*—Urban traffic congestion significantly impacts travel efficiency and commuter experience, particularly in dense metropolitan areas such as Manhattan. This project presents a real-time traffic-aware routing system that integrates an OpenStreetMap-based road network with live traffic data from the HERE Traffic Flow API. The system dynamically updates per-road travel speeds, computes fastest paths using graph-based routing, and exposes a backend API consumed by an interactive web-based frontend. In addition, a machine learning extension using a Random Forest model is explored to predict near-future road speeds, enabling more realistic time-optimal routing. Experimental results demonstrate that the system can process tens of thousands of traffic segments in near real time and provide accurate travel time estimations across Manhattan.**

***Index Terms*—Traffic-aware routing, OSMnx, HERE Traffic API, spatial indexing, shortest path, machine learning, Random Forest**

I. Introduction

Traffic congestion remains a persistent challenge in large cities, where static routing solutions often fail to reflect real-time road conditions. Manhattan, with its dense road network and highly variable traffic patterns, serves as a compelling testbed for intelligent routing systems. Traditional navigation systems rely on proprietary data and closed-source implementations, limiting transparency and extensibility.

This project aims to design and implement an end-to-end, open, and extensible real-time routing system for Manhattan. By combining OpenStreetMap road data with live traffic information from the HERE Traffic Flow API, the system dynamically updates edge weights in a road graph and computes time-optimal routes. A web-based frontend allows users to interactively select start and end points, while a backend routing service computes and returns the fastest route along with distance and estimated travel time.

The primary contributions of this project are:

- A scalable pipeline for integrating live traffic data with an OSMnx road graph.
- Efficient spatial matching between HERE traffic segments and OSMnx edges using an STRtree index.
- A multi-threaded routing backend that safely updates traffic data in real time.
- An exploratory machine learning model for short-term road speed prediction.

II. System Overview

Figure 1 illustrates the overall system architecture. The pipeline begins by downloading and preprocessing the Manhattan road network using OSMnx. Live traffic data is periodically fetched from the HERE Traffic Flow API and matched to graph edges using spatial nearest-neighbor queries. Updated edge weights are then used by the routing backend to compute fastest paths, which are exposed via a REST API and visualized on the frontend.

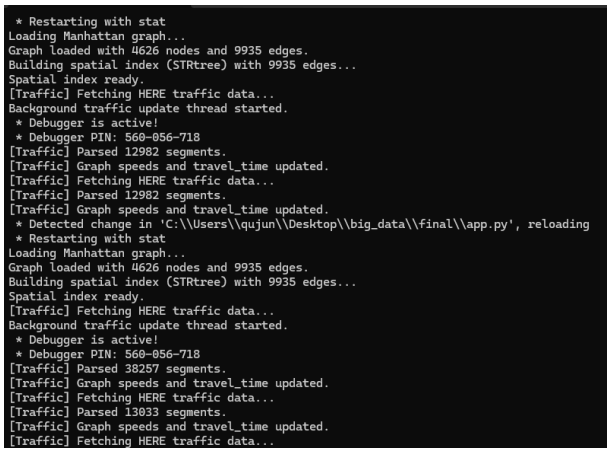

Fig. 1. System flow chart for real-time traffic-aware routing.

III. Map and Dataset

A. Road Network

The Manhattan road network is downloaded using the OSMnx library, which queries OpenStreetMap and constructs a directed graph where nodes represent intersections and edges represent road segments. The graph contains tens of thousands of nodes and edges. Missing or invalid geometries are validated and reconstructed where necessary. For efficient reuse, the processed graph is serialized and stored using Python `gpickle` format.

B. Traffic Data

Traffic data is obtained from the HERE Traffic Flow API (v7), which provides segment-level speed information represented as polyline geometries. Traffic snapshots are collected periodically (every two minutes) and stored in Apache Parquet format for efficient downstream processing and potential offline analysis. Each snapshot contains approximately 38,000 traffic segments covering Manhattan.

IV. Mapping HERE Traffic Data to OSMnx Graph

A. Representation Mismatch

A key challenge is the mismatch between HERE traffic data and the OSMnx graph representation. HERE represents roads

as LineStrings with associated speed measurements, while OSMnx models roads as graph edges whose segmentation and geometry may differ.

B. Spatial Matching Strategy

To resolve this mismatch, a spatial nearest-neighbor matching approach is employed. An STRtree spatial index is built over all OSMnx edge geometries. For each HERE traffic LineString, nearby candidate edges are queried, and the closest edge is selected based on geometric distance. This approach enables efficient matching at scale and allows per-edge speed updates in the road graph.

V. Backend Routing Engine

The routing backend is implemented in Python using NetworkX and OSMnx. Each edge weight represents travel time, computed as edge length divided by current speed. The backend supports a REST endpoint of the form:

```
/route?start=lat,lonend=lat,lon
```

Upon receiving a request, the backend computes the fastest path using Dijkstra's algorithm and returns the GPS coordinates of the route, total distance, and estimated travel time.

To ensure robustness, traffic updates are performed in a background thread, while routing queries operate on a thread-safe snapshot of the graph. This design prevents crashes or inconsistent states during updates.

VI. Frontend Visualization

The frontend is built using Leaflet.js and provides an interactive Manhattan map. Users select start and end points by clicking on the map. The fastest route is displayed as a polyline, along with real-time distance and ETA information. The map automatically recenters to fit the computed route, providing a user-friendly navigation experience.

VII. Machine Learning Extension

To extend the baseline navigation system with predictive capabilities, we introduce a machine learning module that estimates future road speeds on Manhattan Island. This extension enables time-aware routing by forecasting traffic conditions at a specified time offset and integrating the predictions directly into the road network graph.

A. Historical Traffic Data Acquisition

Historical traffic data were collected using the HERE Traffic API. All API queries were spatially constrained to the geographic bounding box of Manhattan Island to ensure consistency with the target routing graph generated by `osmnx`. Traffic conditions were sampled once per minute over a continuous 30-minute period, resulting in 30 temporal snapshots of real-time traffic data.

Each snapshot contains speed-related information associated with HERE road segments. Across all collection intervals, approximately 110,000 raw traffic records were obtained.

B. Data Cleaning and Preprocessing

The raw traffic data contained noise and invalid measurements that could negatively affect model training. A two-stage preprocessing pipeline was applied. First, records with physically implausible values (e.g., negative speeds or extreme outliers) were removed. Second, incomplete or malformed entries returned by the API were discarded.

This process produced a clean and reliable dataset suitable for spatial alignment and supervised learning.

C. Spatial Alignment with OSMnx Graph

A major challenge in integrating HERE traffic data with the routing system lies in the incompatibility between HERE road segment identifiers and OpenStreetMap edge identifiers. To address this issue, a spatial nearest-neighbor matching strategy was employed.

Each HERE road segment was represented by its geometric location and matched to the closest edge in the Manhattan road graph generated using `osmnx`. To maintain consistency, a one-to-one correspondence was enforced by retaining only the nearest spatial match. This procedure resulted in an ID mapping table linking HERE segments to OSM-based graph edges.

After alignment, approximately 4,000 road segments were retained for model training and inference.

D. Training Dataset Construction

Using the established ID mapping table, historical traffic observations were reorganized into a road-centric training dataset. Each training instance corresponds to a single road segment at a given timestamp and includes the current observed speed, temporal features derived from the sampling time, and a time offset indicating the prediction horizon. The target variable is the observed speed at the corresponding future time.

To improve data reliability, road segments with fewer than 10 observations were excluded, and remaining outliers were removed. The final dataset consisted of approximately 110,000 samples spanning nearly 4,000 road segments.

E. Predictive Models

1) Random Forest Regression: A Random Forest regression model was trained using the processed dataset. To control computational complexity and facilitate deployment, the model architecture was constrained to a maximum of 80 trees with a maximum depth of 10. The trained model occupies approximately 11.9 MB of memory and was serialized for integration into the navigation system.

Random Forest was selected due to its robustness to noise, ability to model nonlinear relationships, and compatibility with lightweight deployment environments.

2) Chronos-Bolt-Tiny (Exploratory): In addition to Random Forest, a lightweight time-series forecasting model, Chronos-Bolt-Tiny, was explored due to its significantly faster inference speed and suitability for temporal prediction tasks.

The model was successfully trained and evaluated using the same dataset in a cloud-based environment.

However, persistent dependency and version compatibility issues prevented reliable local deployment of the Chronos model. Despite extensive troubleshooting and attempts to resolve these issues, the model could not be integrated into the final system and was therefore excluded from the deployed pipeline.

F. Graph-Level Future Speed Prediction

To incorporate predicted traffic conditions into routing decisions, a graph-level prediction function was implemented. The function takes as input the current road network graph G, enriched with real-time traffic attributes from `osmnx` and the HERE API, as well as a user-specified time offset.

The function iterates over all edges in the graph and performs the following steps:

1) Check whether the edge exists in the HERE–OSM ID mapping table.
2) If present, use the trained model to predict the road speed at the specified future time offset.
3) Update the edge attributes in the graph with the predicted speed.
4) If the edge is not included in the mapping table, retain its current speed value.

The output is a modified graph representing predicted traffic conditions at the future time horizon, which is subsequently used by the routing algorithm.

G. Performance Considerations and Limitations

Although the Random Forest model is relatively compact, graph-wide inference is computationally expensive. Experimental evaluation on Google Colab shows that a single full-graph update requires over seven minutes to complete. This limitation arises from per-edge inference across a large number of road segments.

While Chronos-Bolt-Tiny demonstrated superior inference speed during testing, deployment constraints ultimately necessitated the use of Random Forest. Future work may explore parallelization, model pruning, or GPU-accelerated inference to address this performance bottleneck.

H. Functional Validation of the Random Forest Inference Pipeline

To verify the functional correctness of the trained Random Forest model and its integration into the navigation system, a functional validation test was conducted. The purpose of this test was not to assess predictive accuracy, but to confirm that the model inference pipeline executes successfully and produces valid outputs when applied to real traffic data.

Figure 2 shows an example output generated by the trained Random Forest model during testing. The successful generation of predicted road speed values confirms that the model loading, feature construction, inference process, and result visualization are correctly implemented. This functional validation demonstrates that the trained model can be reliably executed within the system and provides the necessary foundation for future quantitative evaluation and performance optimization.

Fig. 2. Output visualization demonstrating successful execution of the Random Forest inference pipeline

VIII. Experimental Results

The system successfully processes over 38,000 traffic segments per update cycle and updates graph edge weights in near real time. Routing queries typically return results within milliseconds. A demonstration shows realistic routes across Manhattan with ETAs consistent with observed traffic conditions. The Random Forest model provides reasonable short-term speed predictions, though performance is limited by feature simplicity.

IX. Limitations and Future Work

While effective, the system has several limitations. Spatial matching errors may occur in dense road regions. The machine learning model uses limited features and does not capture long-term temporal dependencies. Future work includes incorporating temporal models such as LSTMs, expanding feature sets, and evaluating predictive routing against real-time routing.

X. Conclusion

This project demonstrates a complete, real-time traffic-aware routing system for Manhattan built using open-source tools and live traffic data. By integrating spatial indexing, dynamic graph updates, and interactive visualization, the system provides accurate and responsive routing. The exploratory machine learning extension highlights the potential for predictive traffic-aware navigation in future work.

References

[1] G. Boeing, "OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, 2017.

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

[3] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," 2008.