Graph Neural Network–Driven EV Route Optimization from Live Traffic & Signal APIs

Authors: Aniket Gulab Khedkar
Project Duration: November 2024 – April 2024
Date: April 2025

Abstract

We present an end-to-end pipeline that ingests **live traffic speeds** and **traffic-signal timing** from public/vendor APIs, cleans and fuses them into a **road graph**, and learns **edge-level travel time and stop-probability** with a **Graph Neural Network (GNN)**. The learned costs feed a **resource-constrained route optimizer** that respects **electric-vehicle (EV) battery** limits and can insert **charging stops** when necessary. The system builds a large-scale urban graph (≈114,883 road/intersection nodes; ≈500 EV charging stations) and returns precise, real-time routes. Visual overlays mark EV stations (green), intersections (red), and roads (blue); routes show start (blue), end (red), and traversed path (white).

---

1. Introduction

Conventional shortest-path routing relies on static edge weights (length/speed-limit) or coarse historical averages. Urban mobility, however, is **temporal, signal-controlled, and congested**. Simultaneously, EV routing is **energy-constrained**, requiring **SoC-aware** path planning and **charging detours**. We address both problems by (i) learning **time-varying edge costs** (ETA and signal delay) with a GNN over the road graph, and (ii) solving a **constrained shortest path with resources** (battery, time) to deliver realistic, EV-feasible routes.

---

2. System Overview

Data → Graph → GNN → Route Optimizer → API

1. **Data ingestion (APIs):**
   - Link-level speeds/flow, incidents, and travel-time snapshots (Δt = 1–5 min).
   - Signal programs (cycle length, phase splits, offsets, coordination groups, detector actuations).
   - Static base map (road geometry & attributes), elevation tiles (for grade).
2. **Preprocessing:** schema harmonization, deduplication, spatial snapping, time alignment, and **CSV exports** for reproducibility.
3. **Graph construction (NetworkX/GeoPandas):** directed multi-graph G=(V,E) with intersections & EV stations as nodes; road segments as edges with static and dynamic attributes.
4. **Learning: spatio-temporal GNN** to predict edge-time **ETA** and **signal stop probability**.
5. **Routing: A*** with a fast heuristic on the **learned costs**, extended to a **resource-constrained label-**

settingalgorithm for EV battery and optional charging stops.

---

3. Data & Pre-processing

3.1 Sources & schema

- **Traffic API:** link id, timestamp, speed, travel time, confidence.
- **Signals API:** intersection id, phase plan (C, g, y, r), offsets, coordination, current phase, queue length (if available).
- **Base map:** road centerlines, lanes, speed limit, oneway, functional class; node coords; segment length; curvature; **grade** derived from elevation rasters.
- **EV stations:** location, connector type, power (kW), access hours.

All feeds are normalized to UTC, deduplicated by (id, timestamp), and written to **CSV** staging tables.

3.2 Cleaning & fusion

- **Map-matching:** traffic link ids → base-map edges; signals → nearest node within ϵ meters; resolve many-to-one cases via geometry overlap.
- **Temporal alignment:** resample all feeds to a common grid (e.g., 5-min buckets) with **last-observation-carried-forward** for ≤10-min gaps; mark imputed flags.
- **Outlier handling:** Huber clipping on speeds; discard negative/zero travel times; enforce logical bounds (e.g., $v \le 1.3\times$ speed limit).
- **Feature standardization:** z-score for continuous features; one-hot for categorical (functional class, control type).

3.3 Graph statistics

The resulting city-scale graph contains **≈114,883 intersection nodes** and **≈500 EV charging stations** (marked as special nodes), with roads as directed edges; map renders use green (EV), red (intersections), blue (roads), and route overlays with white path from blue start to red end.

---

4. Graph Representation & Features

4.1 Nodes

- **Intersections:** degree, signal control type (fixed/actuated), offset within coordination band, historical stop rate.
- **EV stations:** charger power P_{kw}, occupancy (if available), dwell-time distribution.

4.2 Edges

Each directed edge $e=(u \to v)$ holds:

Static: length L_e, speed limit V_e^{max}, lanes, grade θ_e, curvature κ_e, functional class.
Dynamic (per time slice t): speed $v_{e,t}$, travel time $\tau_{e,t}$, incident flags, **signal delay** $\delta_{e,t}$ (expected red-time contribution), density proxy (e.g., v/V^{max}), weather (optional).

Target variables:

- $\hat{\tau}_{e,t}$ — predicted travel time (regression)
- $\hat{p}_{e,t}^{stop}$ — probability of stopping at downstream signal (classification)

---

5. GNN Model

5.1 Architecture

We adopt a **spatio-temporal GraphSAGE** with a temporal encoder:

1. **Temporal encoder** (per edge): **TCN** (1D dilated conv) or GRU over the last W snapshots of edge features.
2. **Graph encoder** (per time t): **GraphSAGE / GAT** on a **line-graph** L(G) (edges as nodes) to capture turn penalties & signal spillback, or message passing on G with **edge features** using PyTorch Geometric (`NNConv`, `EdgeConv`).
3. **Heads:**
   - Regression head → $\hat{\tau}_{e,t}$ (MSE loss)
   - Classification head → $\hat{p}_{e,t}^{stop}$ (BCE or focal loss)

The joint loss:

$$\mathcal{L}=\lambda_\tau \mathrm{MSE}(\tau_{e,t},\hat{\tau}_{e,t})+\lambda_p \mathrm{BCE}(y_{e,t}^{stop},\hat{p}_{e,t}^{stop})+\lambda_{reg}\|\Theta\|_2^2.$$

5.2 Training protocol

- **Splits:** time-blocked (train: past weeks; val/test: future weeks) to avoid leakage; spatial holdout for robustness.
- **Sampling:** edge-time mini-batches with neighbor sampling k hops.
- **Optimization:** Adam (lr 1e−3), early stopping on **sMAPE** (for ETA) and **AUROC** (for stops).
- **Imbalance handling:** focal loss ($\gamma=2$) and class weights for rare stop events.

---

6. EV Energy & Route Cost Modeling

6.1 Edge energy approximation

For edge e traversed at predicted speed v and grade θ:

$$E_e \approx [c_0+c_1 v+c_2 v^2+c_3 \sin(\theta)]\cdot L_e,$$

where coefficients summarize **aero drag** ($\propto v^2$), **rolling resistance**, and **grade**; regenerative braking is modeled via efficiency η_{regen} when $\sin(\theta)<0$.

6.2 Multi-objective route cost

We minimize a convex combination:

$$C(\pi)=\sum_{e\in\pi}\left(w_\tau \hat{\tau}_{e,t}+w_\delta \hat{p}_{e,t}^{stop}\bar{\delta}+w_E E_e+w_r \mathrm{turnPenalty}_e\right),$$

subject to **battery constraints**:

$$SoC_{k+1}=SoC_k-\frac{E_e}{B_{cap}} \quad \text{and} \quad SoC_k \ge SoC_{min}.$$

Charging at station node s adds dwell time $T_{chg}(s,\Delta SoC)$ and increases SoC accordingly.

---

7. Routing Algorithms

7.1 Learned-cost A*

We run **A*** using a consistent heuristic $h(n)=\mathrm{euclidean}(n,\mathrm{goal})/V_{free}$. Edge weights are the **GNN-predicted** travel time plus expected signal penalty and (optionally) an energy term weighted into equivalent seconds.

7.2 Resource-constrained shortest path (battery)

We solve a **Shortest Path Problem with Resource Constraints (SPPRC)** using a **label-setting** algorithm over states (node, SoC):

```
Initialize label at (source, SoC0)
with cost 0
while queue not empty:
  pop best label (n, soc)
  for each edge e = (n -> m):
    cost' = cost + C_e
    soc'  = soc - E_e/Bcap
    if soc' < SoCmin: continue
    if m is EV station and soc' <
SoC_thresh:
        consider charging options:
soc'' = min(1.0, soc' + Δsoc),
cost'' = cost' + Tchg
        relax (m, soc'') if
Pareto-better
    relax (m, soc') if Pareto-
better
Return best label at (goal, soc >=
SoCmin)
```

Dominance rules prune inferior labels on (cost,SoC); optional **bi-directional** expansion accelerates search.

---

8. Evaluation

8.1 Offline predictive metrics

- **ETA:** MAE / RMSE / sMAPE over edges & paths; calibration plots of predicted vs. observed times.
- **Stops:** AUROC / AUPRC and **Expected Red Time** accuracy at signalized approaches.

8.2 Routing quality metrics

- **Optimality gap:** (Cours−Coracle)/Coracle vs. a hindsight oracle (using realized times).
- **On-time arrival rate** under arrival deadlines.
- **Energy feasibility:** % routes meeting SoC constraints without emergency detours.
- **Charging efficiency:** added minutes vs. fixed-stop heuristics; station congestion sensitivity.

Baselines:

- Static Dijkstra (free-flow speeds)
- Historical-mean weights
- Commercial-style ETA without signal model (if available)

8.3 Ablations

- **No-traffic / no-signal** ablations to quantify each source's lift.
- **GNN variants:** GraphSAGE vs. GAT vs. line-graph message passing.
- **Temporal window W** and resampling interval sensitivity.
- **Cost weights** wτ,wδ,wE trade-off curves (Pareto front).

---

9. Implementation Details

- **Graph & geo:** Python, **NetworkX**, **GeoPandas**, Shapely; spatial index (rtree); optional **PostGIS** for persistence.
- **Learning: PyTorch Geometric** / DGL; mixed precision; neighbor sampling; experiment tracking with MLflow/W&B.
- **Pipelines:** `pydantic` configs; **Airflow** or **Prefect** for scheduled data fetch & model refresh.
- **Serving:** FastAPI endpoint `POST /route` accepting `{source,dest, SoC0,SoCmin,timestamp}`; GNN scoring cached per time slice; **A*** in Cython/NumPy for speed.
- **Visualization:** Folium/Kepler.gl map layers (EV stations, intersections, edges, path).

---

10. Limitations & Risks

- **Sensor/API drift:** provider calibrations change; monitor residuals and retrain triggers.
- **Actuated signals:** stochastic phase times; we approximate with expected red; queue spillback may violate independence assumptions.
- **Energy model:** simplified coefficients; vehicle-specific parameters (mass, CdA, tires) should be profiled for accuracy.
- **Data bias:** coverage gaps in speed feeds; EV-station availability may be stale—expose uncertainty.

---

11. Reproducibility

- Version-pin dependencies; save **graph snapshots** (Parquet/Feather) per time slice.
- Log seeds, model state-dict, and train/val/test indices.
- Provide scripts: `fetch_apis.py` → `fuse_to_csv.py` → `build_graph.py` → `train_gnn.py` → `serve_route.py`.
- Ship a **synthetic toy city** for unit tests of SPPRC and charging insertion logic.

---

12. Conclusion

By fusing **traffic** and **signal timing** into a **graph learning** framework, then routing with **resource constraints**, we obtain routes that are **faster**, **more predictable**, and **EV-feasible**. The approach is modular—additional layers (incidents, weather, curb regulations) can be appended—and deployable in real time through cached GNN scores and accelerated search.

Appendix A — Key Hyperparameters (example)

- Temporal window W=6 (past 30 min at 5-min resolution), GraphSAGE depth=2, hidden=128, dropout=0.2
- Loss weights: $\lambda_\tau=1.0, \lambda_p=0.5$; cost weights: $w_\tau=1, w_\delta=0.5, w_E=0.2, w_r=0.05$
- A* heuristic speed V_{free}=60 km/h; SoC_{min} = 10%; charge threshold 20%

Appendix B — Data Schemas (simplified)

traffic.csv: `link_id,timestamp,speed,travel_time,conf`
signals.csv: `int_id,timestamp,cycle,offset,phase,split,control_type`
stations.csv: `node_id,lat,lon,kw,connectors`
graph_edges.csv: `u,v,length,grade,lanes,func_class,speed_limit,oneway`
