STAT 4830

Optimizing Vehicle Routing with Graph-Based and Probabilistic Models

Team: Vishva Gajaraj, Aman Sharma, Darren Mo

The Problem

Problem Statement

- With many routing software the insights one can gain into the eco-friendliness and efficiency of their trips is limited – we aim to expand on this
- Route Planning and Optimization is classified as an NP-Complete Problem
  - Metaheuristics are popular approach (explore problem space and find satisfactory solution)
- Create systemic pipeline that takes into account many trip specific features, geographic features, infrastructural elements, and traffic data to accurately model and predict travel behavior and energy usage
- Calculate concrete figures and statistics benchmarking different models and approach the problem using new techniques

Motivation

- Richer, more detailed insights on vehicle energy consumption compared to what is provided by modern routing applications
- With the growing emphasis on reducing energy consumption in modern society, access to accurate vehicle metrics and efficient routing algorithms can play a crucial role in minimizing overall energy usage
- Explore novel approaches to the NP Complete problem with novel constraints (energy considerations)

Metrics of Success Pt. 1

Our algorithms provide two valuable predictions:

1. A prediction for the optimal route that would minimize time to destination and energy consumption

2. A prediction of how much energy would be consumed by a trip given geographical data, traffic and infrastructural elements, and the start and end coordinates of the trip

Metrics of Success Pt. 2

In analysis of the validity of these predictions:

1. For route prediction, we evaluate our model's performance by comparing its output to the optimal route generated by a baseline model (Dijkstra’s algorithm), and then assess how similar both routes are to historically efficient trips within our dataset

2. To evaluate our energy consumption predictions, we compare the energy used in historical trips with the energy predicted by our model, ensuring that both share the same start and end points.

Technical Approach

Data Sources

- OSMnx
  - Python package for processing street networks (using OpenStreetMap api)
  - Assists in creating Graph Structure

- Extended Vehicle Energy Dataset (eVED)
  - GPS trace records
  - Road Elevation
  - Speed Limits (km/h)
  - Calibration of latitude / longitude
  - Approximation of energy usage for each trip (MAF[g/sec] * duration of trip in sec.)
  - Data from 383 vehicles driving over 370k miles in Ann Arbor, Michigan
    - Large Scale Dataset for Vehicle Energy Consumption Research - (Oh et al., 2020)

- Electric Vehicle Trip Energy Data
  - 1-minute interval driving records of EVs (June 2015 - June 2016)
  - Contains energy consumption in KwH for each trip
  - Avg. latitude and longitude in each trip
  - Current, Voltage, and Battery Temperature data

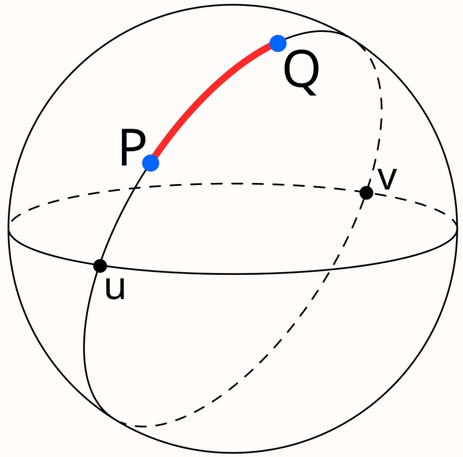
Data Transformations

- Haversine Formula
  - Distance between points given Latitude and Longitude coordinates
- Overlaying Location, Trip, and Vehicle Information
  - Structured with a Graph Structure
  - Can visualize trips using a 2D-Graph

$$a = \sin^2\left(\frac{\Delta \text{lat}}{2}\right) + \cos(\text{lat}_1) \cdot \cos(\text{lat}_2) \cdot \sin^2\left(\frac{\Delta \text{lon}}{2}\right)$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$



Baseline Algorithms

- Dijkstra's Algorithm
  - A simple graph algorithm that finds the shortest path between two points in a weighted graph
  - Intersections and dead ends represent nodes, streets represent edges, and distance of roads are weights of edges
  - Runs in O((V + E)logV) time, where V is the number of nodes and E is the number of edges

- A* Algorithm
  - A more advanced graph algorithm that runs quicker than Dijkstra's and also guarantees to find the shortest path between two nodes
  - To find which node to traverse next, this algorithm uses a formula f(n) = g(n) + h(n), where g(n) is the cost to get to the current node, and h(n) is the predicted cost to get to the destination from the current node.
  - Picks node with lowest f(n) until destination reached

BERT/RL vs Dijkstras/Naive A star

Base Dijkstra's

A* with custom edge weights

Adam

We chose Adam to optimize our models over SGD and other optimizers:

- AdamW adapts learning rates per parameter, which helps with unstable gradients from BERT embeddings and RL loss signals.
- Combines momentum and adaptive scaling for faster, more stable convergence than standard SGD.
- Performs better with high-dimensional, sparse gradients common in transformer-based inputs.
- Decoupled weight decay improves generalization, especially important in noisy RL environments.
- Requires less tuning of learning rate schedules, making training more robust and easier to manage

BERT Model

Literature

Route planning using divide-and-conquer: A GAT enhanced insertion transformer approach (Zhang et al., 2023)

- Used a divide-and-conquer strategy to find optimal routes by identifying high-potential waypoints and recursively splitting the problem into smaller sub-problems.

- Trained an Insertion Transformer to model the sequential relationships and spatial-temporal dependencies between tokenized road segments in historical routes.

- Enhanced learning with attention mechanisms to capture dependencies across longer routes, even between non-adjacent road segments.

Literature

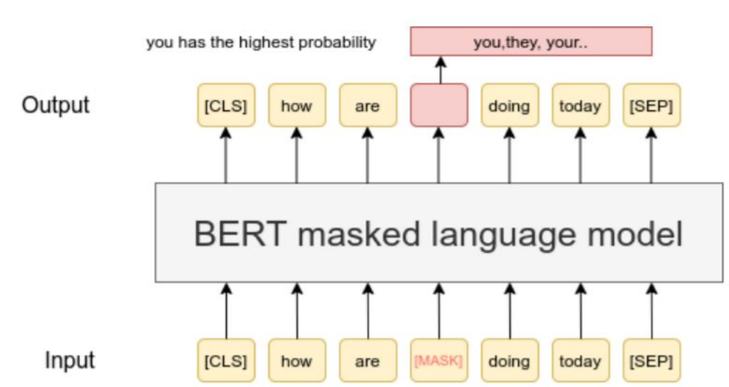
BERT-Trip: Effective and Scalable Trip Representation using Attentive Contrast Learning (Kuo et al., 2023)

- Masks random points of interest (POIs) in trips and predicts them using bidirectional context from surrounding road segments, with fixed position encoding.

- In this study BERT was chosen as the transformer architecture due to its ability to leverage bidirectional context, utilizing both preceding and succeeding tokens to accurately predict masked elements within a sequence.

- Unlike the insertion transformer which dynamically generates a full trip node by node from a partial trip by predicting what POIs to use and where to insert them, BERT trip is given positions of masked POIs and learns what those POIs should be from positional context

- Using inspiration from this study's usage of BERT and to accommodate for our computational resource constraints, we design a BERT-based model that encodes source and destination nodes as special tokens to capture bidirectional context, and employs an MLP (Multilayer Perceptron) head to regress the full trip trajectory in a single forward pass



Data Processing/Modeling

- Loaded ~10 GB of CSVs in 100 k-row chunks
- Each row represented a trip of a vehicle at a given latitude and longitude
- By grouping by trip number of vehicle id, a sequence of latitudes and longitudes sorted by timestamps could be derived from the data – a sequence of nodes traversed by a vehicle for a trip
- Calculated a bounding box to normalize lat/lon without losing too many data points
- Padded/truncated to 100 waypoints per route
- Created a custom tokenizer by augmenting the BERT vocabulary with synthetic latitude and longitude tokens, allowing structured coordinate inputs to be processed as text
- This [CLS] token was fed into a MLP (multi-layer perceptron) to output a 100 x 2 dimensional matrix, representing a predicted optimal trip formatted as a sequence of 100 latitude and longitude coordinates

Modeling Choices

- The problem is modeled as supervised regression, predicting a sequence of trip coordinates from an embedding of source and destination locations
- We use a BERT encoder for its ability to generate bidirectional context in its embeddings as well as its strong ability to extract pre-trained features even for synthetic token sequences
- Using this embedding as input to the MLP simplifies the process of regression to a sequence of points: more information to be derived from high-level contextual relationships rather than isolated coordinate information.
- Normalizing coordinates makes the more optimization more stable by focusing on the spatial structure of the trip rather than incredibly small differences in coordinates

Formula

We use mean squared error as the loss function of our model:

$$L = \frac{1}{N}\sum_{i=1}^{N} \|\hat{r}_i - r_i\|^2$$

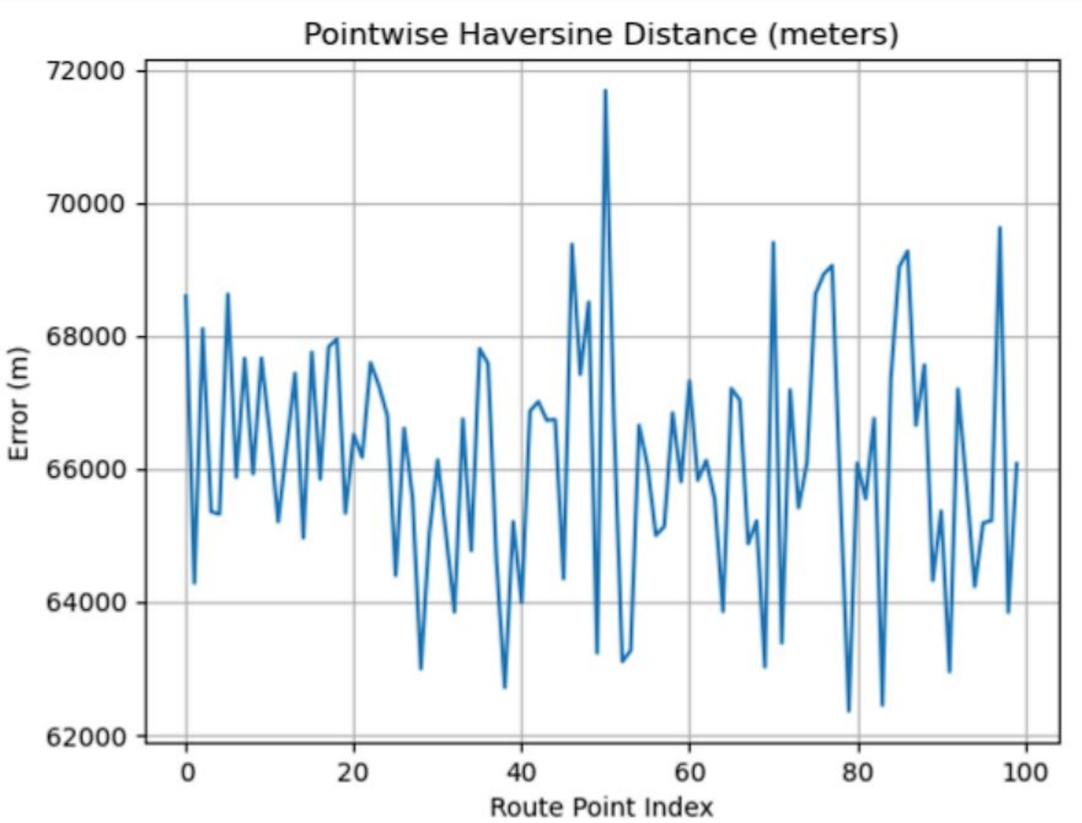
- N is number of points in a trip prediction (100)
- r-hat is the ith coordinate in a sequence of coordinates representing a predicted efficient trip
- r is the coordinate at the ith position of a real efficient trip

BERT Optimisations

- Loaded a Bert embedding generator, filtered to 15,506 vocab tokens; built weight matrix (16,474×200) from random init.
  - LAT_xxx, LONG_xxx, SRC, DEST
- Better init achieved through positional encoding
- Initially freeze embedding layer and pretrained BERT weights for 10 epochs to not entirely lose positional semantics
- Normalisation of Coordinates -> biggest performance boost
- Vanilla Learning Rate decay
- Key HP: Learning Rate, Early Stopping Patience

BERT Results

- Path error curve does converge to ~0.01 MSE
- Average Haversine error ≈ 20 km per point
- Get ~65% training accuracy indicating the model is able to capture the notion of a node in a route
- Takeaway:
  - Loss remains high due to BERT being trained/tuned on text sequences
  - Limited information in embedding
  - Does not inherently recognize the relation between “LAT_01” and “LAT_02”



Reinforcement Learning

Literature

Multi-objective reinforcement learning approach for trip recommendation (Chen et al., 2023)

- Models trip recommendation as a Markov decision process where states represent previously selected points of interest in a trip and actions represent the next POI to append to the trip

- This is solved using a multiobjective deep reinforcement learning framework (Deep Deterministic Policy Gradient, an actor-critic RL method) to execute route planning that optimizes travel time and energy consumption

- The critic in this model is optimized using target q-values that are estimated using the Bellman equation: $Q_{\text{target}}(s,a) = r + \gamma \max_{a'} Q(s',a')$
  - where s is the current state, a is the current action, r is the reward from taking this action a at state s, γ is the discount factor, and maxQ(s', a') is the max reward that can be obtained from any future actions at the next state
  - The target Q-values are computed using a separate target network based on the Bellman equation, and the parameters of this target network are softly updated toward those of the online Q-network to improve training stability

- The critic model is optimized using stochastic gradient descent (SGD), where target q-values from the Bellman equation are compared to predicted q-values while training the model using mean squared error (MSE)

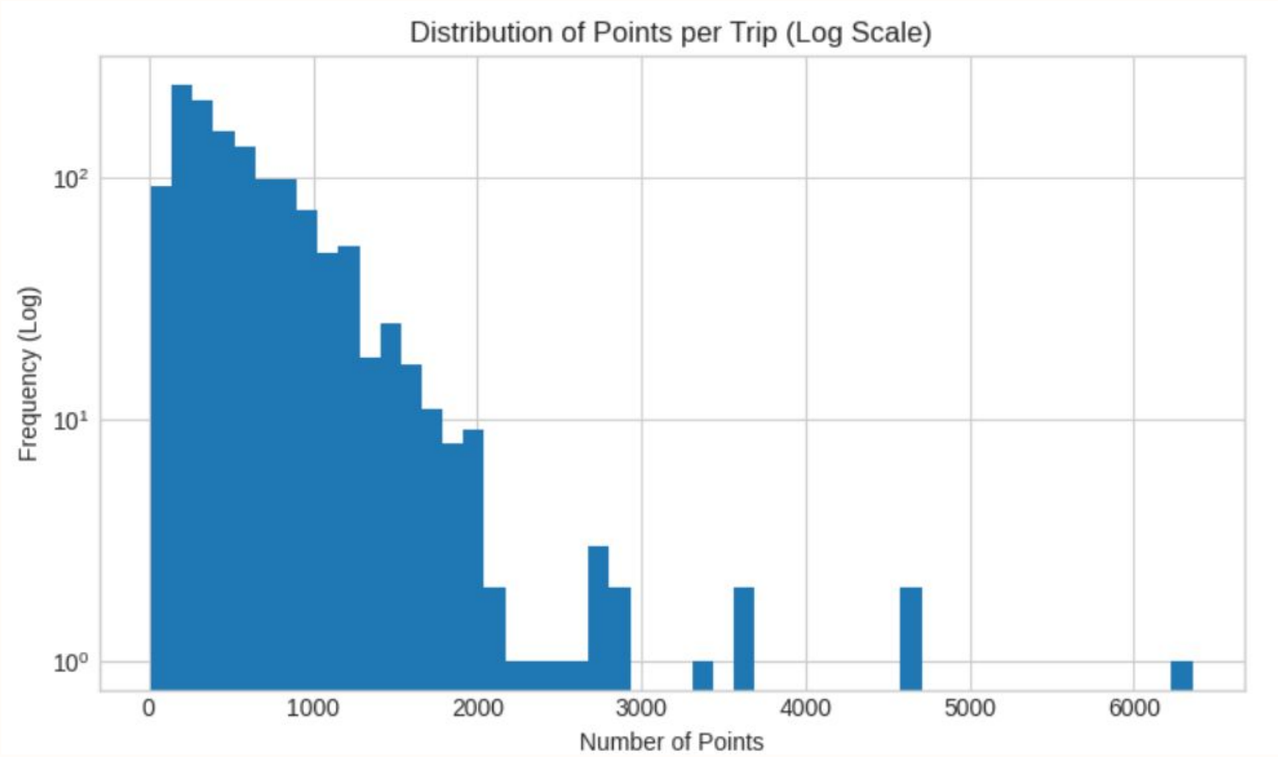
Literature

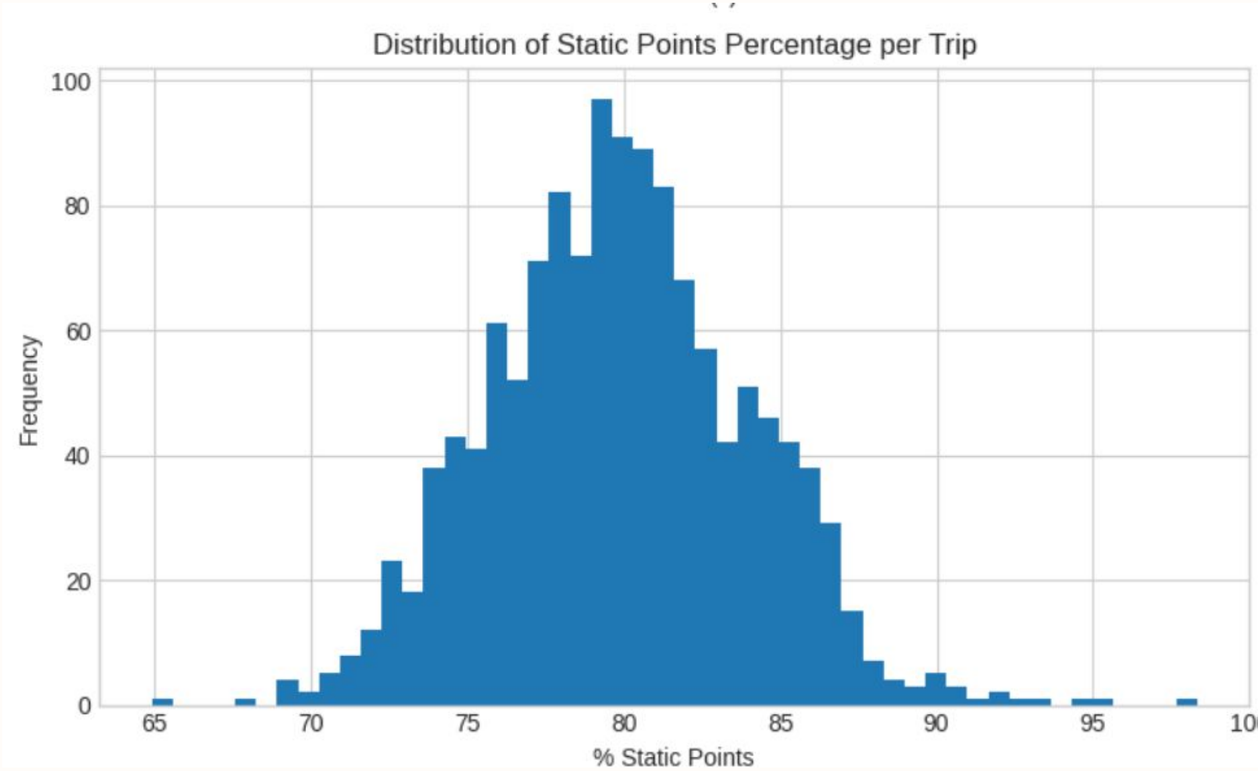
Multi-objective reinforcement learning approach for trip recommendation (Chen et al., 2023)

- The actor model outputs an action given the state that the process is currently on
- This is fed into the critic model that outputs a q-value
- The actor model's goal is to maximize cumulative q-values through every part of the trip, from the source to destination
- Gradient ascent is done on Q(s, a) to update the actor model's weights so it outputs actions at each state that align with what the critic model will think has the highest q-value (maxmizing Q(s,a) by changing the actor's policy)
- At the conclusion of training, the actor and critic models are jointly utilized to generate trips that maximize predicted Q-values, corresponding to minimizing both travel time and energy consumption within a trip

Data Processing

- Data is processed the same as it was for BERT model up until after the normalization of latitude and longitude
- We choose to keep data between the 10th and 90th percentiles of longitude and latitude to filter out any data points that are nonsensical in the context of a trip (over 6000 km between consecutive points in a trip)
- Dropped any static points (where vehicle did not move)
- Originally tried to build a custom graph from the trip data:
  - Tried discretized positions with 10 bins and velocity into 100 bins via linear scaling
- Ended up snapping coordinates to nearest OSMnx points





Reduced from 6000 trips to ~1200

Modeling choices

- Like the paper, this process is modeled as a Markov Decision Process and our reinforcement learning model learns an action-value function (predicts q-values)

- This function implements rewards shaped by time and energy proxies such as MAF (airflow into engine during combustion), step time, and RPM – as well as distance from destination

- Rather than using a Actor-Critic framework which is preferable for continuous action spaces, we use a deep q-network (DQN), which is useful and easier to train for smaller discrete action spaces like picking outgoing edges from a graph node

- We start at the source “state” in our GraphRouteEnvironment and traverse the environment either using the action with the maximum q-value (intialized randomly) or a randomized action (for exploration). All experiences are stored in a replay buffer

- After a few experiences, we sample a batch from this replay buffer and try to predict the q-values for a state and action in an experience in this batch

Modeling choices

- Similar to the critic model training regiment used in the paper, a target network, based off the Bellman equation, is used for outputting target q-values to compare to predicted q-values from the online (training) network
  - Separation of networks for prediction q-values and target-values is done to increase the stability of training, rapid changes in the online network's predictions would constantly shift the target values, preventing the model from converging

- Based on differences between the outputs of the target network for q-values and the online network, we adjust weights and biases in the online network

- Every couple steps soft updates are made to the target network parameters, slightly tuning its parameters to lean more towards the online network parameters
  - Done to prevent the agent from learning toward outdated or incorrect targets, eventually stalling learning and preventing convergence to the optimal policy

- Eventually target and predicted q-values converge and an optimal trip is predicted from selecting the action with the max q-value at each state, overall we want the maximum accurate q-value for a trip: $\max_{a_0,\ldots,a_T} \sum_{t=0}^{T} \gamma^t Q(s_t, a_t)$

Formulae

The model is trained to minimize the Bellman error:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - \left(r + \gamma \max_{a'} Q_{\theta^-}(s',a')\right)\right)^2\right]$$

- Where Θ represents online model weights
- s represents the current state
- a represents the chosen action
- $Q_\Theta(s, a)$ represents the online network's predicted q-value for action a at state s
- s' represents the next state from action a from state s
- a' represents any action at s'
- r is the immediate reward for taking a
- γ is the discount factor (how much future rewards matter relative to immediate ones)
- $r + \gamma \max_{a'} Q_{\Theta^-}(s', a')$ represents the target networks predicted q-value for action a at state Θ

Formulae

Additionally soft updates are applied to the target network using the function :

$$\theta_{\text{target}} \leftarrow \tau\,\theta_{\text{online}} + (1-\tau)\,\theta_{\text{target}}$$

- Where Θ_{target} and Θ_{online} represent target model and online model weights respectively
- τ controls how drastically the target model weights shifts towards those of the online model (small in this model)

RL Optimisation

- Changed from a simple Q learning Agent to a Rainbow DQN agent
  - (NoisyNet + Double + Dueling + Prioritized + N-step)

- Reward shaping:
  - Δ shortest-path distance to goal
  - Goal bonus (+10), step penalty (–0.1), invalid action (–1.0), timeout penalty (–10)
  - Clipped final reward to [–5, +5] to stabilize learning

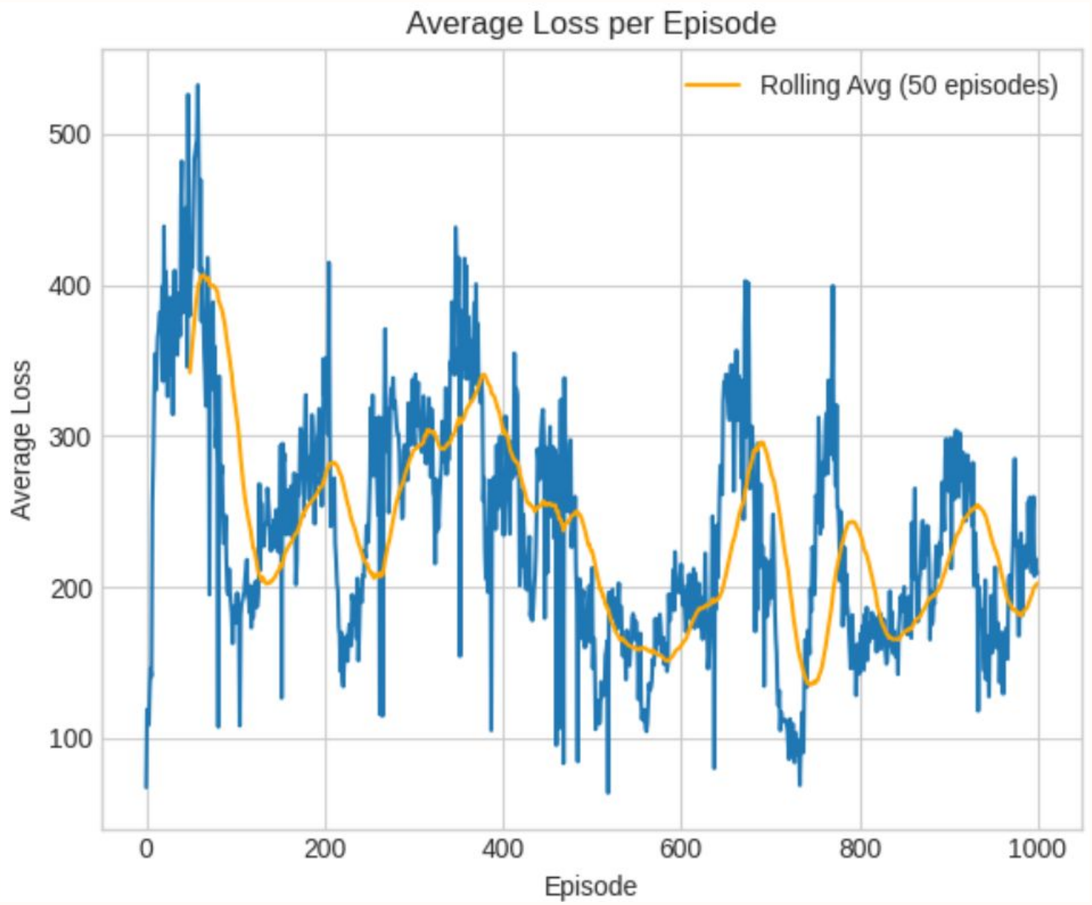
- Deferred learning until sufficient buffer fill

- Residual connections to prevent exploding gradients

- Key HP: max step count, scaler

RL Results

- Consistent 1500 steps → agent wasn't reaching goal initially
- After graph node alignment and shaping, reward improved from –300+ → ~–80
- Still very noisy, but showing learning via improved recent average reward
- Takeaway:
  - Model struggled to adapt to the diverse routes
  - Found some set of routes very efficiently



Result Analysis

- Agent needed more steps to appropriately balance exploration + reward seeking
- Resource bottleneck, model took too long to train to observe reasonable improvements given the massive state space
- Training data had more routes “energy efficient” towards the north making learning imbalance.
- Insufficient reward shaping, energy metric was minimal and should have taken more factors.

Graph Neural Network

Literature

Electric vehicle energy consumption modelling and prediction based on road information (Wang et al., 2015)

- Developed a physics based model that predicted how much energy a road segment would take to traverse given its attributes such as speed limit, road slope, elevation, road surface type

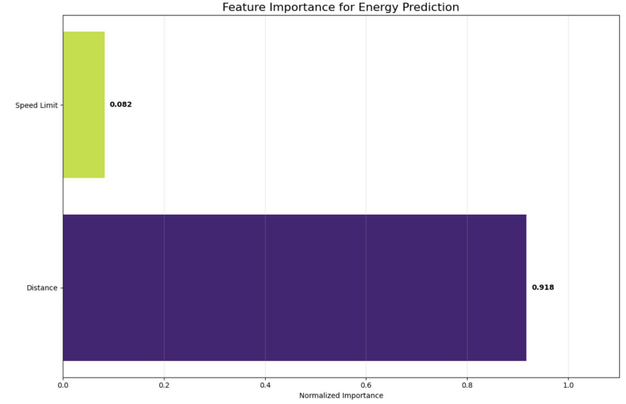
- Used OSM (Open Street Map) and the Shuttle Radar Topography Mission to obtain data on these attributes for each road segment

- Based on the different road segments that were traversed by trips, the predicted energy consumption of these segments were summed to estimate the energy consumption of a trip

- Rather than using a physics based model, our model uses semantic and geographic data from OSMnx to predict energy consumption of a trip represented by a sequence of to and from nodes and the edge connecting them

- Our model assumes that a trip between two points will be the shortest trip in distance as calculated by the OSMnx library functions, represented as a sequence of road segments

Energy Consumption Model



- Used same dataset as BERT and RL model, with OSMNX graph data in relevant areas to provide data to create node and edge embeddings

- We learn a function $f_\theta(G, T_i)$ that maps a trip represented as a sequence of edges (e1,e2,…,ek)in a directed road network graph G=(V,E) with node features $x_v \in R^2$ and edge features $e_{uv} \in R^2$ to a predicted log-transformed energy value $\hat{y}_i$, minimizing the loss $\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(f_\theta(G,T_i) - \log(1+y_i))^2$

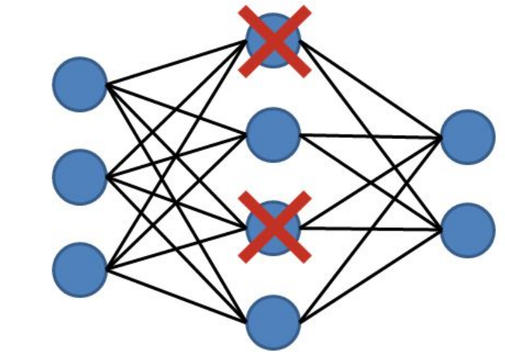
- The graph neural network learns representations of road networks, modeling nodes as intersections and edges as roads with features like length and speed limit

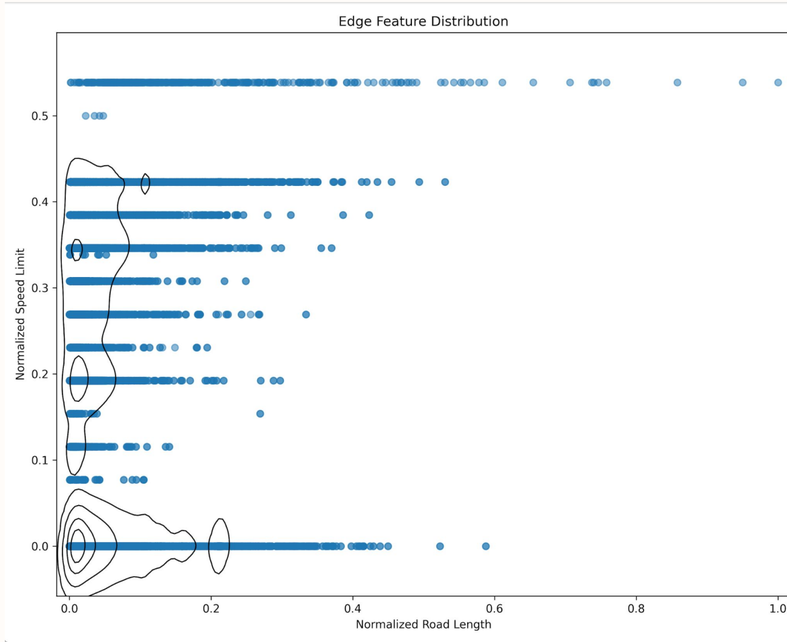
- A start and end coordinate are fed into the model, the model calculates the shortest path (a sequence of nodes) between these coordinates, and based on the embeddings of these nodes and their sequence, predicts the energy consumption of a hypothetical, efficient trip between these points

General Optimization Choices on GNN

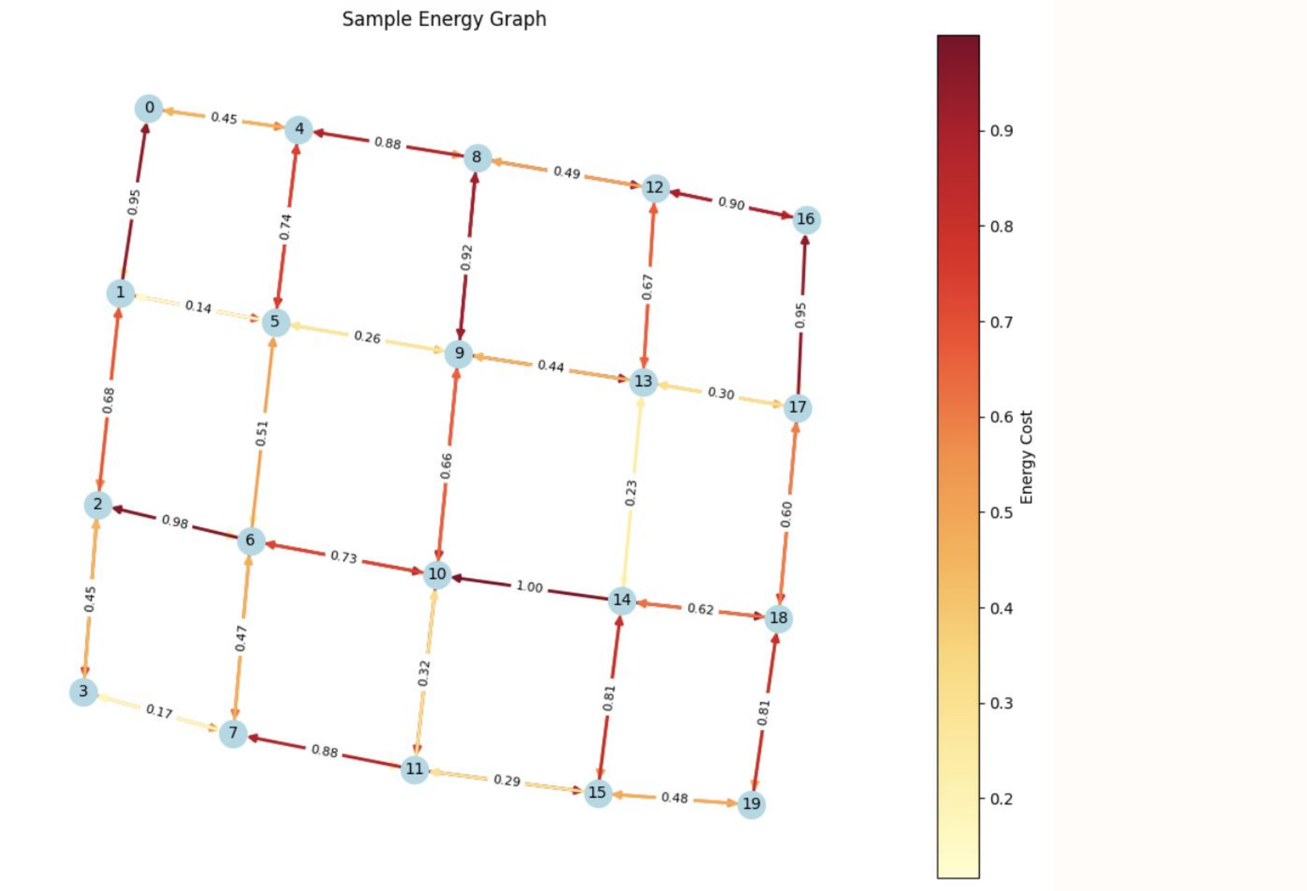
$$\theta^{(t)} = \theta^{(t-1)} - \frac{\alpha^{(t)}\hat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}} + \epsilon}$$

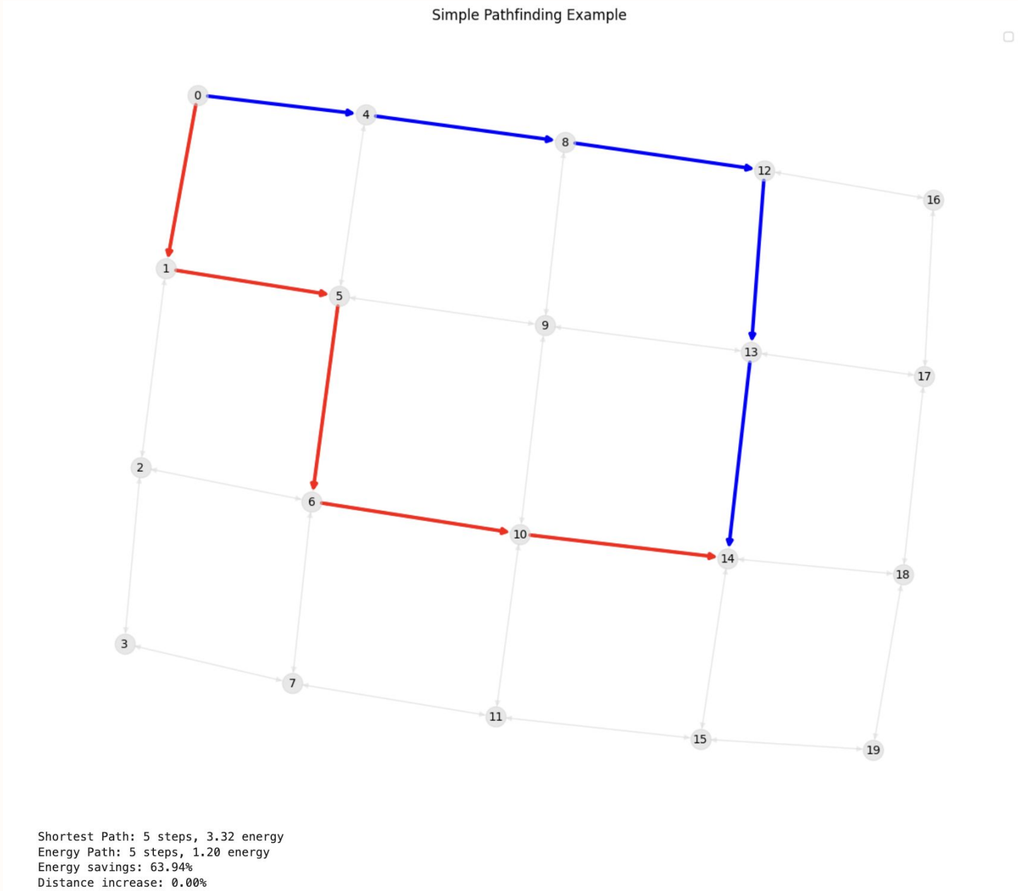
- Normalization of coordinates, road lengths, and speed limits for stabilized training
  - Data is often heavily skewed due to vehicle idling
- Hierarchical Information Processing
  - First Level: GNN layers learn node and edge embeddings to capture the spatial relationships and road properties
  - Second Level: The GRU (recurrent neural network) to processes the sequence of embeddings along a path, learning how energy accumulates over a route (like time-series)
- Utilized standard regression loss and ADAM optimizer
- Dropout (30%) → randomly set 30% of neurons to 0 during training to prevent overfitting (learn more robustly)



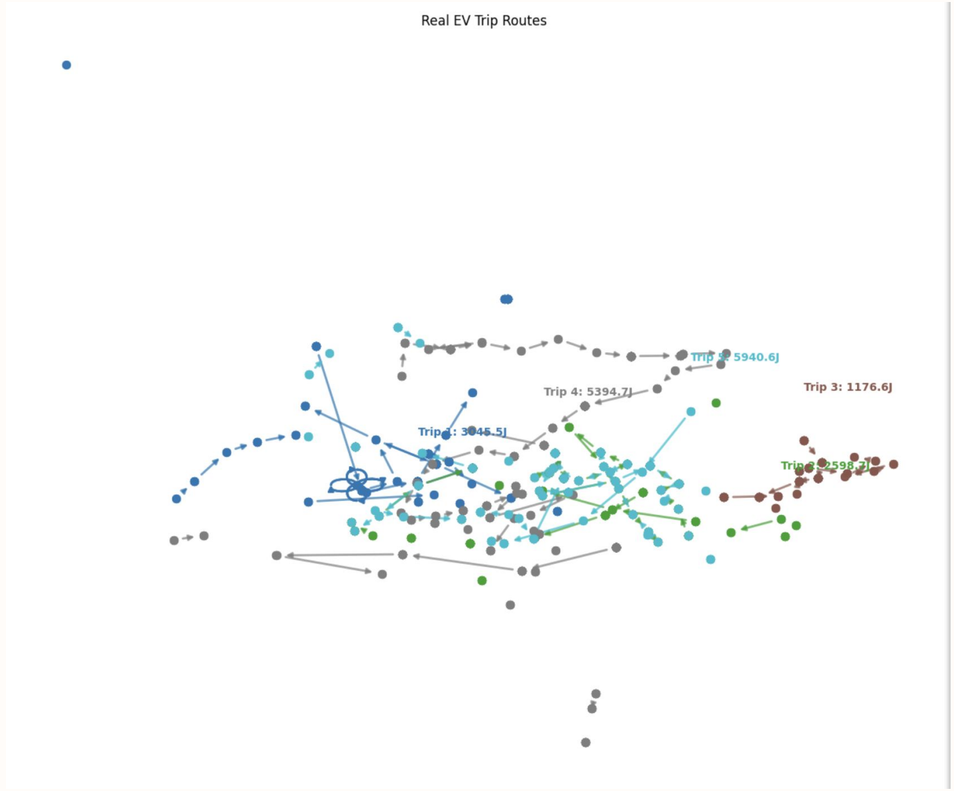


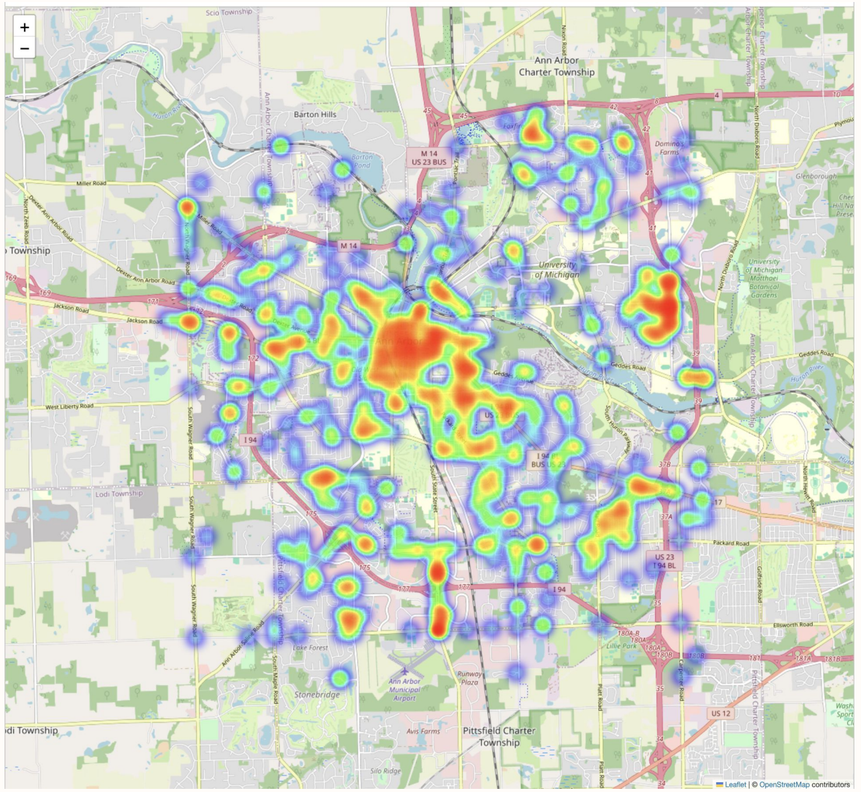
Proof of Concept with Synthetic Data





Graphical Visualizations of Vehicles Traveling in Michigan



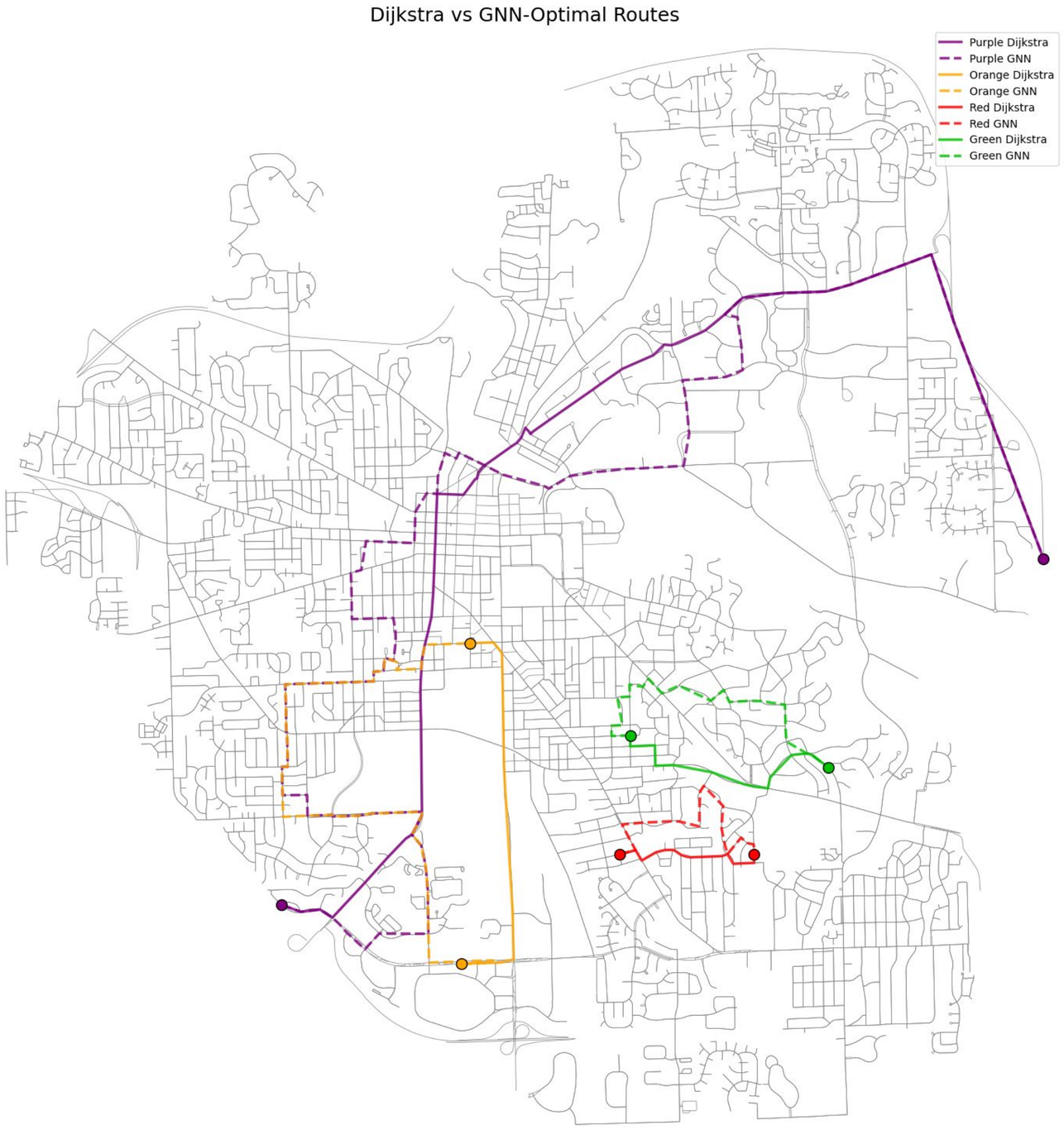


GNN Results

- Key Results:
  - Predicted energy efficient route
- Example Route Prediction
  - Purple Path Traversal for Dijkstras: 14037.92 meters, 10.414 avg grams of airflow in engine/sec
  - Purple Path Traversal for GNN: 20,346 meters, 10.3 avg grams of airflow in engine/sec

- Obtained Routes from previous graphic
  - Energy significantly down on average from locations on original graph (shown to the right not fully meaningful)

- Average loss for energy consumption prediction of about .49



Project Reflection

Challenges

- Data Integration
  - Problem with trying to locate and integrate sources that were limited and unrelated
    - Tried using mathematical transformations on coordinates (partly worked)
  - Obtaining free, accessible EV behavior data
    - We created models that are ready to work on more specialized data if provided
- Project Evolutions
  - With our successive self-critiques, we slowly realized that our biggest problem was our data and that the original problem we had proposed a solution to simply did not have feasible resources to solve
  - Pivoted to a problem of energy consumption in general for vehicles

Trip Energy C	Vehicle ID	Trip Distance	Time of Day	Day of the W	Longitude	Latitude	Speed	Current	Total Voltage	Maximum Ce	Minimum Ce	Trip Time Length
0.672	1	6	10.3333333	4	121.497948	31.2815742	246	2.58334828	308.283333	31	30.8333333	13
0.896	1	6	16	4	121.587564	31.2560699	393.714286	2.98572904	304.485714	29	28	18
1.344	1	7	16.0909091	2	121.576968	31.2620344	192	2.35456036	308.463636	31.2727273	30	21
1.344	1	8	19	5	121.549709	31.2577959	369.24	1.54001495	308.06	30	30	16
0.896	1	6	14.1666667	6	121.58228	31.2150302	413.450617	9.65989154	304.473457	28	28	129
1.344	1	6	18.5483871	6	121.58998	31.2842383	175.83871	-1.9128883	304.648387	29	29	18

VehId	Trip	Timestamp(n	Latitude[deg]	Matched Lon	Vehicle Spee	Speed Limit[k	Engine RPM[r	MAF[g/sec]	Absolute Loa	HV Battery S	HV Battery C	HV Battery V	OAT[DegC]	source_file	step_id	Power[kW]	Acceleration[km/h/s]
2	685	22500	42.3025192	-83.706528	3	56	723	5.01000023	36.8627472	35.2941284	0	311	16	eVED_17110	0	0	0
2	685	107900	42.2992892	-83.716389	66	56	1696	22.3899994	74.1176453	35.2941284	0	311	16	eVED_17110	1	0	0.7377049180327868
2	685	139100	42.2979717	-83.720074	0	56	658	4.42999983	33.7254906	35.2941284	0	311	16	eVED_17110	2	0	-2.1153846
2	694	54800	42.3084372	-83.73494	62	48	1453	3.82999992	12.9411764	35.2941284	0	311	16	eVED_17110	3	0	0
2	694	63700	42.3073878	-83.734965	58	48	1355	3.6099999	13.333334	35.2941284	0	311	16	eVED_17110	4	0	-0.4494382
2	694	100000	42.3020047	-83.735255	55	40	1290	3.3499999	12.9411764	35.2941284	0	311	16	eVED_17110	5	0	-0.0826446
2	694	122100	42.3007661	-83.735714	0	40	655	3.86999989	29.8039227	35.2941284	0	311	16	eVED_17110	6	0	-2.4886878
2	694	143600	42.3007139	-83.735733	0	40	1608	17.8400002	54.5098038	35.2941284	0	311	16	eVED_17110	7	0	0
2	694	152600	42.3001561	-83.735955	45	40	1613	16.5	70.5882339	35.2941284	0	311	16	eVED_17110	8	0	5

What Worked Well

- LLM Models for Ideation, Preliminary Coding, Debugging
  - We used models like Claude and ChatGPT to help with our original idea generation as well as basic structures of our training models.
  - Debugging some of the errors that came with running and training our models, as well as the data preprocessing steps were solved with LLM assistance (ex. premature casting to a type before safely loading and processing datasets)
- Data Preprocessing
  - Once data was obtained and the objective of our project pivoted in a different direction, the process of preparing the data in the datasets as well as readily provided data in OSMNX did not prove to be too difficult
  - Additionally, some data, such as node pagerank, could be calculated on our own and could still provide valuable information for our models
- Literature Review for Informed Model Development
  - We used Google Scholar to find academic sources to inspire our model development
  - Was a particularly easy process, as there were many different approaches to route prediction all of which had accurate results returned by their models

Acknowledgements

- Professor Davis
- Zhang, S., Fatih, D., Abdulqadir, F., Schwarz, T., & Ma, X. (2022). Extended vehicle energy dataset (eVED): an enhanced large-scale dataset for deep learning on vehicle trip energy consumption. *arXiv preprint arXiv:2203.08630*.
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 2019.

Individual Contributions: Darren

Most surprising result: Our models still being able to generate paths that were relatively efficient compared to the dijkstras paths (as observed by validating with our dataset) even if our predictions for energy consumption were very underestimated

Specific lecture topic that helped the most: Going over Reinforcement Learning briefly in class for part of a lecture was very helpful as it helped us decide which type of RL model to use in our project (actor-critic vs DQN). The lectures about SGD with momentum and Adam also helped, as it helped us decide which optimizer was best suited for the complexity of our models, our modeling choices, and our data

Perspective Change: Before taking this class, I was very unsure about how the optimization process worked with models in general, my only experience was learning the SGD approach that LeCun used for his CNN. So learning about a plethora of other methods, such as different Adam optimizers and the use of momentum, broadened my understanding of optimizers and ML as a whole. Additionally I also have a better low level understanding about the differences between each optimizer and even differences in model types typically in different ML based problems (Transformers vs. markov decision processes with RL).

With two more weeks: I would like to try an insertion transformer model to predict a sequence of paths rather than just use BERT in the manner we did, although i am sure this would take a lot more computation resources than we have

If i could restart my project: I would choose an entirely different topic in itself that had more publicly available data. This way we would be able to create an accurate model that could be trained sufficiently with the large amount of data we had even with the restricted amount of compute we have.