Progress Report

Background

The Porto Taxi Dataset

The Porto taxi dataset is a real-world GPS trajectory dataset collected from 442 taxis operating in Porto, Portugal, over a period of one year (July 2013–June 2014). It was released as part of the ECML/PKDD 15 Taxi Trip Time Prediction Challenge and has since become a standard benchmark for trajectory prediction and travel-time estimation tasks.

Dataset structure:

- **train.csv** — Contains full trajectories with ground-truth POLYLINE (GPS points); ~1.71 million trips
- **test.csv** — Contains partial trajectories; the goal is to predict the remaining travel time (ETA); 320 trips

CSV columns:

Column	Description
TRIP_ID	Unique identifier for each trip
CALL_TYPE	A = central dispatch, B = taxi stand, C = street hail
ORIGIN_CALL	Call center ID (NA for stand/street)
ORIGIN_STAND	Taxi stand ID (NA for dispatch/street)
TAXI_ID	Unique taxi identifier
TIMESTAMP	Unix timestamp of trip start
DAY_TYPE	A = weekday, B = Saturday, C = Sunday/holiday
MISSING_DATA	Boolean flag indicating if trajectory has missing GPS samples
POLYLINE	List of [lon, lat] points sampled every 15 seconds

The POLYLINE is a JSON-like array of GPS coordinates. Points are recorded at 15-second intervals, so trajectory length directly reflects trip duration.

Train CSV Data Analysis

Metric	Value
Total trajectories	~1,710,670
Test set size	320
CALL_TYPE distribution (sample)	A (dispatch), B (stand), C (street hail) — C is most common
Invalid trips (pre-cleaning)	~98,800 (~5.8% of dataset)

Invalid trips are those where consecutive GPS points exceed 1 km apart (unrealistic jumps indicating data errors). These are logged in `invalid_trips.csv` with segment distances, indices, and statistics.

Goal

Our objective is to **validate the result of our ETA (Estimated Time of Arrival) prediction model using real data from the Porto taxi dataset**. We aim to:

1. Convert real GPS trajectories into SUMO-compatible routes
2. Run simulations and/or model inference on these routes
3. Compare predicted ETAs against ground-truth travel times from the dataset
4. Assess model accuracy and identify failure modes

Challenges

1. Trajectories Have Missing Data

- The dataset includes a **MISSING_DATA** flag. When `True`, the trajectory contains gaps where GPS samples were lost (e.g., tunnel, urban canyon, device issues).
- Missing data breaks the assumption of continuous, evenly sampled trajectories.

2. Trajectories Are Set Manually by User

- Manual trajectory selection and validation is error-prone and does not scale to large datasets.

3. GPS Points Are Inaccurate by Nature

- GPS accuracy varies with conditions (urban canyon, multipath, etc.). Points may be offset from the true road position.

4. Invalid Trajectories with Large Jumps

- We found trajectories where consecutive GPS points are more than 1 km apart—physically implausible for a 15-second sampling interval. These indicate data corruption or device errors.

5. Taxi Movement Differs from General Traffic

The nature of taxi movement is different from most traffic:

- **Pick-ups and drop-offs** — Taxis stop at origins and destinations, introducing idle periods and non-driving segments that do not reflect pure travel time.
- **Detours and search behavior** — Taxis can wander around looking for the destination, leading to a different pattern of movement from common traffic flow.
- **Passenger-induced behavior** — Stops at traffic lights, mid-trip stops, or route changes requested by passengers add variability.
- **Heterogeneous trip purposes** — CALL_TYPE (dispatch, stand, street hail) affects where and how trips start.

As a result, raw taxi trajectories are a mix of actual road-following movement, waiting periods, and short erratic movements (e.g., maneuvering in parking lots). This complicates both route inference and ETA validation.

Stage 1: Enhanced Cleaning

We implemented an enhanced cleaning pipeline that removes or corrects problematic trajectories:

1. **Removed trajectories with MISSING_DATA** — Excluded trips where GPS samples were lost.
2. **Trimmed static start/end points —** Detected repeated or nearly identical first/last GPS points (e.g., taxi waiting at pickup/dropoff). Points within 15 m of each other are considered static; we trim to the first/last point where significant movement begins/ends.
3. **Split at invalid segments** — When consecutive GPS points are more than 1 km apart, we split the trajectory at that segment. Each resulting segment is treated as a separate trip, since the jump indicates a data error or device reset.

Result: Stage 1 enhanced cleaning removed approximately **5%** of the original dataset (trajectories that could not be reliably cleaned or had too many invalid segments).

Stage 2: Converting Trajectories to Routes

Model Background: Snapshot-Based Traffic

Our ETA prediction model uses a **snapshot of traffic** at a given time. For each vehicle at snapshot time, we store:

1. **Original route** — The full planned path (sequence of edges)
2. **Current coordinates —** Where the vehicle is at snapshot time
3. **Position relative to route:**
    - The current edge the vehicle is on
    - The current traffic load on that edge
    - The percentage of route remaining

Each edge is updated with:

- Number of vehicles currently on it
- Number of vehicles that have it in their planned route
- Average speed of vehicles on the edge

Hence, **Step 1 requires converting GPS trajectories into SUMO routes**—a sequence of road network edges that the vehicle follows.

Conversion Pipeline

The conversion from trajectory to route is done in the following steps:

1. Validate and Modify the Trajectory

- Apply the same validation and cleaning as Stage 1 (trim static points, split at invalid segments).
- Only segments with at least 3 GPS points are processed.

2. Isolate Relevant Nodes and Edges with a Bounding Box

- Build a bounding box (or rotated minimum bounding rectangle) around the trajectory with padding (e.g., 200 m).
- Filter the road network to only edges whose shape intersects this box.
- This creates a **sub-graph** containing the desired route and dramatically reduces the search space.

3. Assign Weights to Graph Edges (Orange, Green, Other)

For each segment between consecutive GPS points, we match road edges by:

- **Direction match:** The edge's direction (from start to end node) must be within $\sim 20^{\circ}$ of the GPS segment direction. Edges that don't match are discarded.
- **Distance score:** For each candidate edge, we compute the minimum distance from both GPS points to the edge's polyline. A combined score is: `angle_diff × 5 + (d1 + d2) / 2`.

Orange edges — The edge closest to the start point and the edge closest to the end point of each GPS segment. These are the most likely edges the vehicle actually traversed.

Green edges — The top 5 best-matching edges per segment (by score) that are not already orange. These are plausible alternatives along the route.

Other edges — All remaining edges in the sub-graph. These are penalized heavily in the path search.

4. Run Dijkstra + A* to Find the Cheapest Route

- **Edge weights:** Orange = 1, Green = 10, Other = 1000.
- We run a shortest-path search from the start edge to the end edge.
- When `node_positions` and `goal_xy` are available, we use **A*** with an admissible heuristic (Euclidean distance to goal, scaled by minimum cost per meter). Otherwise, plain **Dijkstra** is used.
- Only edges inside the bounding polygon are considered.
- The result is a sequence of edge IDs that matches the shape and characteristics of the original trajectory.

5. Project GPS Points onto the Calculated Route

- For each GPS point, we project it onto the nearest point on the route polyline (the concatenation of edge shapes).
- We compute: timestamp, edge_id, coordinates (projected), distance_from_previous, and speed.
- This yields a **sumo_route_gps** list: each original GPS sample mapped to a point on the route with edge and speed information.

Output: JSON Intermediate Format

All data for a trajectory and its corresponding route(s) is saved to a JSON file. Each record contains:

```
{
```

```
  "trajectory_id": <trip_num>,
  "segments": [
    {
      "starting_timestamp": <int>,
      "duration_seconds": <int>,
      "number_of_gps_points": <int>,
      "gps_points": [[lon, lat], ...],
      "number_of_edges": <int>,
      "route_edges": ["edge1", "edge2", ...],
      "number_of_sumo_route_gps_points": <int>,
      "sumo_route_gps": [
        {
          "timestamp": <int>,
          "edge_id": "<edge>",
          "coordinates": [lon, lat],
          "distance_from_previous": <float>,
          "speed": <float>
        },
        ...
      ]
    }
  ]
}
```

This JSON is an **intermediate, crucial step** in the process of building the final dataset for model validation.

Next Step: Convert Routes to Snapshots

The next stage is to **convert routes to snapshots**. This means capturing the accumulated state of all vehicles traveling at each snapshot time:

- At time T, for each vehicle with a route:
  - Determine its position on the route (current edge, progress along edge)
  - Update edge-level statistics (vehicle count, planned-route count, average speed)
  - Store per-vehicle: current coordinates, current edge, traffic load on edge, percentage of route left

Snapshots will feed into the ETA model for validation against ground-truth travel times from the Porto dataset.

Summary

Stage	Description	Status
Stage 1	Enhanced cleaning (missing data, trim, split)	✓ Complete (~5% removed)
Stage 2	Trajectory → SUMO route conversion	✓ Complete
Stage 3	Routes → Snapshots	⏹ Next