521479S Software project (2024) – Hybrid Intelligence Autonomous Driving System

Hybrid Intelligence Autonomous Driving System

Final Report

Draft 1.0

Name	Student number
Arttu Myllyneva	41380514835
Abdurrehman Syed	2308546
An Vu	2409284
Sakari Ronkainen	Y626811
Sampo Ylisiurua	2405192

Contents

1. Introduction

This report presents the development and implementation route planning algorithm models for optimal route selection in a complex, multi-criteria environment. The goal of these algorithms is to learn to make decisions by selecting the best route from multiple alternatives based also on route parameters such as distance, duration, traffic level, and road condition.

The aim of this project was to research and implement Proof-of-Concept (PoC) route planning algorithms as part of a larger ADS system under development in the 6G Visible project. Route planning is crucial in autonomous vehicles and navigation systems, requiring the identification of optimal routes while balancing constraints such as distance, time, safety, and multi-criteria optimization [1]. These algorithms are integral to applications where trade-offs between conflicting objectives, such as speed and safety, are necessary.

Traditional route planning methods often fail in adapting to real-time changes like traffic congestion or adverse weather. Reinforcement Learning (RL) offers a promising alternative by dynamically optimizing decision-making through environmental interaction and cumulative rewards. Models like Deep Q-Networks (DQN) [2] enable agents to learn from complex, dynamic conditions, making them well-suited for real-world route planning challenges.

This report explores two modern approaches to route planning. The first approach uses a DQN agent guided by a dynamic reward function that balances exploration and exploitation to identify optimal routes along with Human Feedback (HF) feature to enhance adaptability and user-centered optimization. The second approach focuses on a Deep Learning (DL) based cost optimization model, leveraging a contrastive loss function to dynamically learn cost trade-offs and emulate user preferences from training data.

The report provides an overview of the project's objectives, methodologies, and findings. It details the development and testing of two distinct DQN workflows and a DL model tailored for user-specific route preferences. Key topics include the integration of diverse data sources for algorithm training, the presentation of various algorithm training methodologies, and the implementation of Human Feedback (HF) mechanisms. Finally, the report concludes with an evaluation of results, challenges encountered, and recommendations for future development.

1.1 Deep-Q-Network Architecture

DQN combines Q-learning with deep neural networks to address high-dimensional problems that traditional Q-tables cannot handle. The workflow and key features of DQN are presented diagrammatically for optimal route selection in Figure 1. The architecture of the model and its various components are presented in more detail below:

1. **Environment**
   The environment represents all possible states, where each state includes features of available routes such as distance, duration, traffic, and road condition. These route features are fed into the Local Q Network, which predicts Q-values for all possible actions.

2. **Local Q Network**
   The local network uses the agent's experience to generate Q-values for each action. The agent employs so-called epsilon-greedy policy to balance exploration (testing new actions) and exploitation (leveraging learned strategies). The training process is iterative, involving continuous updates of the Q-values and the weights of the local network. The workflow integrates Replay Buffer and Local Q Network to stabilize learning. The bi-directional flow between these components ensures that the agent incrementally improves its decision-making while adapting to changes in the environment.

3. **Action and Reward**
   After the agent selects an action (choosing a route) based on the predicted Q-values, a reward is calculated using a reward function. This function evaluates the selected route based on environmental feature values and weights.

4. **Replay Buffer**
   The replay buffer stores the agent's experiences, which include the current state, selected action, received reward, and resulting next state. These experiences are sampled in batches during training, allowing the agent to learn from past experiences and stabilize Q-value updates.

5. **Target Q Network**
   To avoid instability during training, the Target Q Network provides a stable reference for Q-value updates. Its parameters are periodically updated with the local network's weights. This ensures the Q-value targets remain consistent over multiple training steps, reducing oscillations in learning.

6. **DQN Loss Function**
   The DQN loss function measures the difference between the predicted and the target Q-values. The loss is minimized during training by adjusting the local network's weights. Gradients computed during this process guide the updates, improving the network's predictions over time.

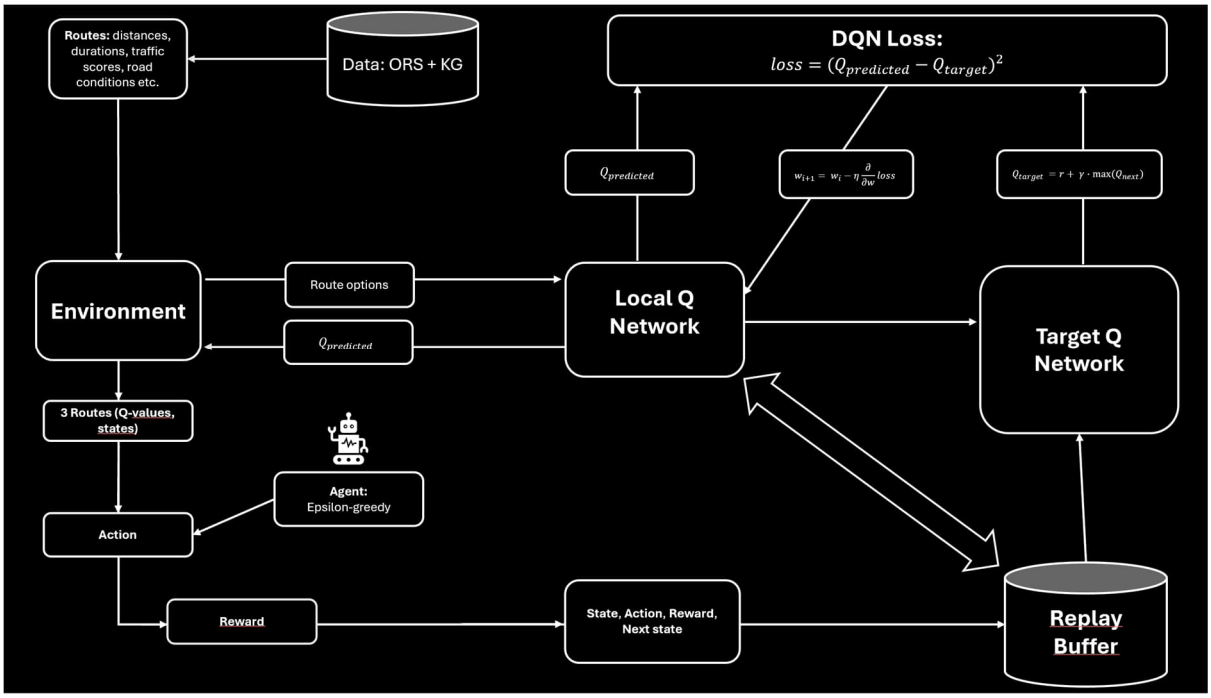

Figure 1. Diagram of the architecture and operational workflow of DQN for optimal route selection.

1.2 DL Architecture

In the deep learning (DL) model, the key idea is to model user preferences using a travel cost function. The travel cost represents how much a user dislikes a particular route based on various attributes such as distance, road conditions, traffic, safety, and other factors. The higher the travel cost of a route, the less favorable it is for the user. The deep learning model aims to predict the travel cost for each route, thereby enabling the system to select the most optimal route that minimizes the user's discontent and maximizes their preference satisfaction.

1. The Environment: Similar to DQN approach, the environment is the data relate to traffics like way types, way steepness or other weather conditions (we didn't incorporate weather data in this project yet)
2. The Model: The model simply suggest routing to user in real time and get feedback, If there is discrepancy between user choice and model choice, the new data point is added into the data set and training live (Figure 3).

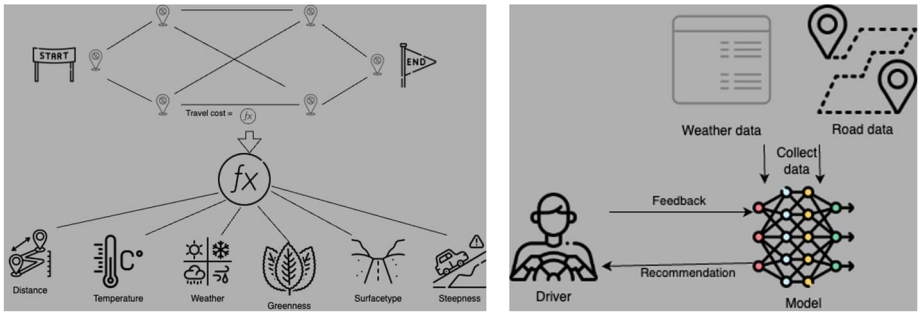

Figure 2. The travel cost function (left) and the data flow diagram (right) in DL model.

2. Customer

Institution: 6G Visible Project, University of Oulu
Name: Nada Sanad & Anna Teern
Email: nada.sanad@oulu.fi & anna.teern@oulu.fi

3. Acronyms and Definitions

DQN: Deep Q-Network

DL: Deep Learning

ORS API: Open Route Service Application Programming Interface

RL: Reinforcement Learning

KG: Knowledge Graph

HF: Human Feedback

PoC: Proof-of-Concept

4. References

[1] M. Reda, A. Onsy, A. Ghanbari, and A. Y. Haikal, "Path planning algorithms in the autonomous driving system: A comprehensive review," *Rob. Auton. Syst.*, vol. 174, p. 104630, Apr. 2024, doi: 10.1016/J.ROBOT.2024.104630.

[2] J. Bernhard, R. Gieselmann, K. Esterle, and A. Knol, "Experience-Based Heuristic Search: Robust Motion Planning with Deep Q-Learning," *IEEE Conf. Intell. Transp. Syst. Proceedings, ITSC*, vol. 2018-November, pp. 3175–3182, Dec. 2018, doi: 10.1109/ITSC.2018.8569436.

[3] "Neo4j Graph Database & Analytics | Graph Database Management System." Accessed: Dec. 28, 2024. [Online]. Available: https://neo4j.com/

[4] "OpenStreetMap." Accessed: Dec. 28, 2024. [Online]. Available: https://www.openstreetmap.org/#map=15/65.01387/25.46661

[5] "Homepage - Finnish Meteorological Institute." Accessed: Dec. 28, 2024. [Online]. Available: https://en.ilmatieteenlaitos.fi/

[6] "Open data and APIs for traffic | Digitraffic - Fintraffic." Accessed: Dec. 28, 2024. [Online]. Available: https://www.digitraffic.fi/en/

[7] "openrouteservice." Accessed: Dec. 28, 2024. [Online]. Available: https://openrouteservice.org/

[8] "PyTorch." Accessed: Dec. 28, 2024. [Online]. Available: https://pytorch.org/

5. Project Description

A Python framework was designed and implemented to support the integration of multiple data sources and DL models for route planning. This framework enables route data retrieval from KG and external ORS API, along with data preprocessing and the pre- and post-training phases of route planning algorithms. The architecture of the Python framework, along with two DQN training workflows and one conventional DL training workflow, is detailed in this section.

5.1 Python Framework

Key functionalities of the Python framework are represented in Table 1, showcasing how components such as KG, ORS API, and route planning algorithms are interconnected.

Table 1. Python framework key-components and their roles in the route planning workflow.

Component	**Description**	**Python libraries**	**Connection to other components**
Neo4j Knowledge Graph (KG)	Database containing information about routes, weather conditions, and traffic etc	`neo4j`	Geospatial node pairs for ORS API
OpenRouteService (ORS)	API for retrieving routes using coordinate pairs	`requests`, `openrouteservice`	Receives geospatial node pairs from KG, Provides route distances, durations, and geospatial coordinates
Data preprocessing	Prepares data from KG and ORS for algorithm model training	`numpy, pandas`	Ensures data is scaled and augmented (simulations) for training
Route planning algorithm	DQN and conventional DL based model that learns to select the optimal route	`torch, numpy`	Receives preprocessed data (route parameters) as input
Visualization	Displays route options and model recommendations to the user	`matplotlib`	Visualizes model performance
Human Feedback	User provides feedback on the route selection, influencing model learning	-	Provides additional training data for algorithms, Used to align the model with user preferences through post-training

5.2 Knowledge Graph

The neo4j KG [3] utilized in this project was constructed using data collected from the Oulu region during early autumn 2024. It contains a total of 69,565 nodes and provides detailed information about road segments, including e.g. road types, surface materials, and additional attributes such as speed limits and weather conditions. The contents of the KG are summarized in Table 2.

The KG combines information from multiple API's to be used in decision-making. It uses data from

OpenStreetMap [4], Finnish Meteorological Institute [5], City of Oulu plowing data, and Digitraffic API's [6] and links the data to the road segments so that it can be used effectively.

Knowledge about routes is retrieved from the KG using the coordinates that the route consists of to match road segments in the KG. It is possible to return every piece of information related to the road segments, or it is possible to only take specific elements if there is less relevant information.

However, the limited variation of some parameters in the KG posed challenges for route planning algorithm training (See Chapter 9.). The lack of sufficient variation in critical parameters hindered the generalizability of the KG data. As a result, the first DQN training workflow partially relied on simulated data to augment the dataset.

Table 2. Overview of the KG parameters. * After component name indicates the lack of variation in data.

Component	**Description**
OSMnode	Represents the nodes in OpenStreetMap
RoadSegment	Connects two OSMnodes, representing a road segment
PointForecast*	Provides weather forecast data for a specific geospatial location
StationForecast*	Provides broader weather forecast data from weather stations
RoadType	Categorizes roads (e.g., highway, residential)
SurfaceType	Defines the surface material of a road
SpeedLimit	Provides speed limits for road segments
WeatherStation*	Represents the source of weather forecast data for broader areas

5.3 OpenRouteService API

The OpenRouteService (ORS) API [7] serve as the primary source for route data between geospatial node pairs extracted from the KG. The API enabled the generation of multiple route options, including data on distances, travel durations, and turn-by-turn route coordinates. For each node pair queried from the KG, the ORS API provided three alternative routes to ensure diversity in the available options.

5.4 Route Planning Algorithms

Two DQN-based models and one conventional DL model were trained using integrated data from the KG and ORS to develop algorithms for route planning in Oulu area. The workflow of each method, detailing their training processes and integration of KG and ORS data from Oulu area, are described below. More detailed description of training phase of each model is represented in the Chapter 7.

5.4.1 DQN Training Workflow 1: Dynamic Reward System

1. **Random Route Pair Generation**
   The Neo4j KG was utilized to retrieve random geospatial node pairs for routes from Oulu area. These nodes represented geospatial points connected by road segments.

2. **Route Data Collection**
   For each pair of geospatial coordinates from KG, three alternative route options were retrieved from ORS API. These routes were returned as structured responses, including route coordinates, distances, and estimated travel times. Python script was automated for API calls.

3. **Route Data Storage and Preprocessing**
   The retrieved route data was stored in a CSV file, with each entry containing:
   - Start and end node coordinates.

- Route coordinates in GeoJSON format.
- Route distance and travel time.

4. **Data Enrichment with Simulated Traffic and Road Condition**
   The KG data was leveraged only for node pair retrieval because the lack variation in several parameters. To simulate real-world variability and enhance the CSV dataset, additional metrics were generated:
   - Traffic Scores: Simulated random integers between 1 and 5, representing traffic density.
   - Road Condition Scores: Simulated random integers between 1 and 5, representing road quality.

   These scores were added to the dataset, enabling the DQN model to consider more complex state representations during training. The enriched dataset provided a comprehensive basis for training and testing the DQN algorithm for route planning. Each parameter in CSV data was normalized to ensure compatibility across different scales.

5. **DQN Agent Pre-Training Setup**
   Agent pre-trained with the enriched training data with PyTorch framework [8]. Each state in the training represented route characteristics (distance, duration, simulated traffic, and simulated road condition) while actions corresponded to the agent selection of one of the three route options. Rewards were assigned dynamically based on the alignment of the agent's decision with the optimal route selection.

6. **DQN Agent Post-Training with HF**
   After the initial training phase, the DQN agent's performance was further adjusted using a Human Feedback (HF) mechanism. This approach allowed for additional learning based on feedback of the agent's decisions by human users.

5.4.2 DQN Training Workflow 2: Node level analysis

This workflow tests DQN training by concentrating node-level features extracted from a Neo4j graph database for route conditions. This helps capture a more granular view of route conditions, thereby enhancing the quality of route selection beyond just a traditional path-level view.

1. **Node-Level Feature Extraction and Preprocessing**

   At first, possible routes are determined by using ORS, which returns the optimum routes along with a collection of intermediate nodes. These nodes are subsequently used to query a Neo4j graph database for several of their features including:

   - Surface Type
   - Speed Limit
   - Road Type
   - Severity of road condition
   - Temperature

   On the contrary to considering the whole path only, this well-structured set of node-level features offers a finer-grained perspective of the route.

2. **Advanced Preprocessing for Node-Level Features**

   A preprocessing step is subsequently applied to the data in order to train the DQN model. Continuous variables, e.g., speed limits and distances, are normalized, and scales are made comparable. Categorical features, such as road type and surface type, are encoded into numerical representations suitable for the model. Such aggregating methods (e.g., calculating averages, modes, medians) are used to summarise the node-level features and to describe the final path. We use these methods to get an overall idea rather than training the model on each individual road segment's features which would be computationally expensive. For instance, weighted average is used for speed limit as longer road segments will impact on the travel time, mean is used for temperature to get a general idea of the whole path. For road severity we pick the maximum value to see how worse the conditions can get.

3. **Integration with Dynamic Reward System**

   The node-level information gets incorporated into the reward function, and this one pays maximum

attention to:

- Not going over severely poor roads past certain limits.
- Penalizing routes with unsafe speed limits, such as above 80 km/h.
- Accounting for environmental elements, such as bad weather.

This dynamic reward system ensures that the DQN agent learns to balance optimality with safety, thereby enhancing its decision-making capabilities.

4. **User Feedback and Adaptive Learning**

The system has a user feedback loop, which adapts to user preferences over time:

1. The route selected along with the features associated with it is shown to the user for feedback.
2. The weights that are assigned to different features of the reward function are updated based on feedback given about the selected path.
3. This adaptive mechanism allows the system to iteratively refine route propositions and, therefore, continuously seek improvement in accordance with a user's needs.

5.4.3 DL Training Workflow

Due to the lack of real user data, we simulate user data for training purposes.

1. Simulate User data: The data is created by randomizing 100 points within Oulu area. From these 100 points we generate 1000 routes by using ORS to simulate the user preference we create some function to simulate the user preference.
2. Train on population data: To enable the model to generalize more quickly to new users, we first train it on population-level data. By learning from large, diverse dataset, the model develops an understanding of common factors that affect decision making process (We skip this step in PoC due to lack of data).
3. Train on personal data: Then to further learn the user preferences the model could be then learn on personal data to provided better personal experiences.

6. Training and Evaluation of Route Planning Algorithms

The training process involved the implementation of three distinct route planning algorithm approaches:

1. **DQN with Dynamic Reward System**: Utilizes a flexible reward mechanism to adapt to dynamic environmental conditions such as traffic, weather, and urgency.
2. **DQN with Node-Level Analysis**: Enhances decision-making by incorporating granular, node-level features like road conditions, speed limits, and environmental factors.
3. **Conventional DL Approach**: User-oriented approach that learns behavior and preferences while optimizing routes based on simulated user data.

6.1 DQN Training Workflow 1: Dynamic Reward System

In DQN, the reward system is the primary mechanism for guiding the agent's learning. For route selection, a dynamic reward system serves as a flexible way to train a DQN agent, especially in scenarios where the environment is complex and constantly changing. The key features of dynamic reward system are presented below.

1. **Dynamic Adjustments**
   Dynamic reward mechanism ensures that the agent learns to prioritize routes that optimize a combination of different parameters.

2. **Reward Calculation**
   The reward for each DQN agent's action (route selection) is calculated using a dynamically adaptive reward function that evaluates key route parameters. In the DQN training workflow 1, the data included distance, travel time, simulated traffic, and simulated road conditions for each route. The reward function is expressed as a weighted sum of parameter-specific functions, ensuring a balanced trade-off between various factors:

$$R = w_1 \cdot f_D + w_2 \cdot f_T + w_3 \cdot f_{Tr} + w_4 \cdot f_{RD}$$

In reward function, dynamic weighting enables the agent to prioritize different factors depending on the scenario, such as emphasizing traffic reduction during congested conditions or prioritizing road safety under poor road conditions.

Example of dynamic weighting:

- During peak traffic hours, the weight for traffic w_3 is raised to prioritize less congested routes.
- In high-urgency situations, the weight for shorter durations w_1 is increased to reflect the need for faster travel times.

3. **Advantages of the Dynamic Reward System**

The agent can adapt to changing conditions such as traffic congestion, weather, or urgency.

4. Training example

The dynamic DQN reward model was trained on 476 routes over 250 episodes, with start and end nodes selected randomly from the KG. Unlike the static weighting approach, this model introduced dynamically adjusted weights to reflect changing traffic and route conditions.

The agent evaluated three alternative routes per decision step, considering key parameters: distance, duration, simulated traffic, and simulated road conditions. The reward system assigned weights dynamically based on the average traffic and road condition scores:

- Default Weights: w_1 = 0.1 (distance), w_2 = 0.1 (duration), w_3 = 0.7 (traffic), w_4 = 0.1 (road condition).
- Traffic Emphasis**:** When traffic was high, the weight for traffic (w_3) increased further, while road condition (w_4) became a secondary consideration.
- Road Condition Priority: In scenarios with low road condition scores, the weight for road conditions (w_4) increased, ensuring the agent prioritized safer routes.

The agent began training with an epsilon-greedy strategy (ε=1.0), ensuring exploration in early episodes. Epsilon decayed gradually over time (ε_{min}=0.1), transitioning the agent toward exploiting learned strategies (Figure 4). During training, the model dynamically adjusted weights to prioritize route traffic reduction and road condition (safety). The rewards were tracked over episodes to evaluate the agent's learning progress.

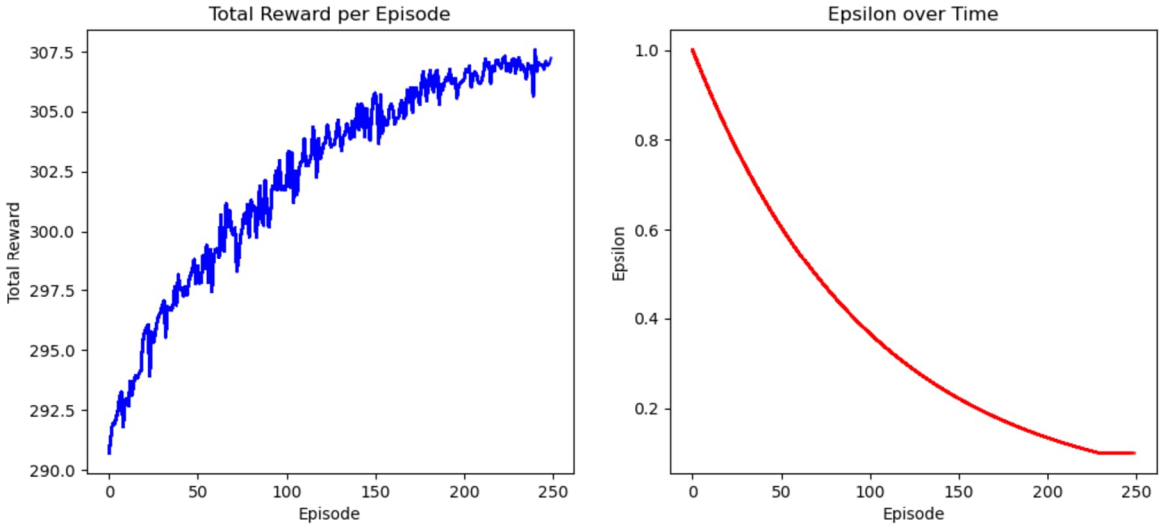

Figure 4. Illustration of the agent's total reward per episode (left) and epsilon decay over time (right) during training with the dynamic weight adjustment model.

5. Training Results

After the initial training phase, the DQN agent's performance was further tested using a HF mechanism. The test data included 20 node pairs, each containing 3 alternative routes. Comparing the trained agent to human choices (prefer less traffic and safer roads) showed that the agent was able to choose 75-90% of the time same route as user.

6.2 DQN Training Workflow 2: Node-Level Analysis

Building on the core DQN framework, this method expands the ability of the agent by bringing in node-level features. The intermediate nodes along those routes, gathered using the ORS and improved through a Neo4j graph database with data, give further details over road conditions, speed limits, and environmental conditions. Integrating these node-level observations into the dynamic reward is expected to enhance decision-making.

1. Safety and Environmental Considerations in Reward Design

In this version, the reward is implemented to encourage choice of routes which are not only optimal but safe for the users, following factors are accounted:

- Severe road conditions up to a certain extent after which the reward would be penalized.
- Unsafe speed limits (e.g., 80km/h) after which the reward would be penalized.
- Account for environmental factors (e.g., by considering weather conditions).

2. State Representation and Route Evaluation

The state vector for the agent includes features such as route length, weighted average speed limit, mean temperature, and maximum road severity for both the shortest and recommended routes. This comprehensive representation allows the DQN to evaluate routes effectively.

3. Training Results

The model is trained upon 476 routes with intermediate nodes for 1000 episodes, where the start and end nodes were picked randomly from the Neo4j database. Similar to the simulated data approach, we also used the same greedy epsilon strategy to explore and exploit the system.

The figure – 5 below shows how the rewards increase exponentially before reaching to its optimum point (blue graph) which highlights that the model successfully learned the safety mechanism and route planning we wanted it to learn. Whereas the second graph (red) shows the decrease in epsilon over the period of 1000 episodes, highlighting that at first the model explores all possible options and then sticks with the optimal ones, hence reducing the epsilon value.

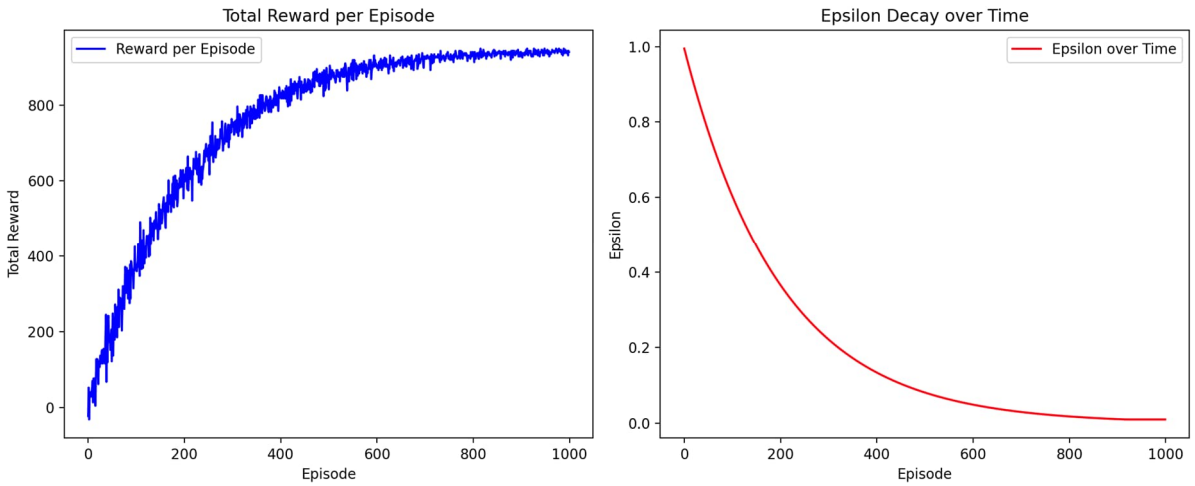

Figure 5. Training Results Workflow2

4. Human Feedback Results and Model Testing

The figure – 6 below shows all the available routes from Point A to B based on the ORS provided results. Additionally, it also provides the different parameters and their values as to show what these routes offer and from them the DQN model picks the optimum route. Afterwards, it then provides its reasoning for choosing that

route and then asks the human for feedback.

The simple approach taken here is that if the user is satisfied with this explanation, they can choose not to change the model weights, otherwise, they can give their own custom weights so that the model updates itself according to the human needs.

```
--- Available Routes ---
Route 1:
  - Distance: 30597.2 meters
  - Duration: 1847.9 seconds
  - Weighted Avg Speed Limit: 0.0 km/h
  - Mean Temperature: 0.0 °C
  - Max Road Condition Severity: 0
  - Route Length: 3 meters

Route 2:
  - Distance: 33286.1 meters
  - Duration: 2171.7 seconds
  - Weighted Avg Speed Limit: 0.0 km/h
  - Mean Temperature: 0.0 °C
  - Max Road Condition Severity: 0
  - Route Length: 3 meters

Route 3:
  - Distance: 33348.4 meters
  - Duration: 2194.9 seconds
  - Weighted Avg Speed Limit: 0.0 km/h
  - Mean Temperature: 0.0 °C
  - Max Road Condition Severity: 0
  - Route Length: 3 meters

Chosen Route: Route 1

--- Reasoning ---
The selected route has the following characteristics:
  - Distance: 30597.2 meters
  - Duration: 1847.9 seconds
  - Weighted Avg Speed Limit: 0.0 km/h
  - Mean Temperature: 0.0 °C
  - Max Road Condition Severity: 0
  - Route Length: 3 meters

Feedback mechanism active.
Was the chosen route satisfactory? (yes/no): no
Adjusting weights based on your feedback.
Enter adjustment for distance (current weight: 0.4): 0.8
Enter adjustment for speed_limit (current weight: 0.3): 0.1
Enter adjustment for temperature (current weight: 0.3): 0.1
Updated weights: {'distance': 0.5555555555555556, 'speed_limit': 0.22222222222222227, 'temperature': 0.22222222222222227}
```

Figure 6. Human Feedback.

6.3 DL Training Workflow

This method is more user-oriented, designed to learn user behaviour and preferences while adapting to new routes and environments. It aims to align with the user's preferences and continuously improve its adaptability.

1. Cost Function as the Basis for Optimization

The idea is still costs based, each road segments will be assigned with a cost function represent the "cost" or "inconvenience" that user have to experience when travel on the segment. The problem is then transformed into an optimization task: finding the route that minimizes the cost function.

In this approach, we assume the user have an internal preference function qualify how much they dislike the given segment, the model will try to best match that preference function (deep learning).

The cost function of a segment s_i:

$$f(s_i) = d \cdot c(x_i)$$

Where d is the distance of segment and x_i is the attributes of segment i.

Then we have the cost function of the whole route:

$$C(R) = \sum_{i=1}^{n} f(s_i)$$

Let's say at every segment of the road we have 3 Route choice i = 1,2, And the user choice **u** is different from

the model choices.

2. Designing the Loss Function

Then we design the loss function as following:

$$m = Arg\min_i\left(C(R_i)\right)$$

$$L = C(R_m) - C(R_u)$$

In practice, we are using contrastive loss to force the model to increase the contrast of the better route and avoid the cost function converse to 0.

To test this method, we devise some cost function to emulate the user preferences, and this method can reach 85% similar to user choice.
Further improvements could be made by integrating user choice into the model and freezing some of the model parameters, this would help reduce the search space and the model better match the user preference.

3. Training Results

To evaluate whether the model effectively learns user preferences, we conducted tests using two different simulated user functions. One function favours highways, while the other does not:

$F_{prefer\_highway}$ = distance_highway

$F_{not\_prefer\_highway}$ = - distance_highway

Using these two simulated user functions, we generated two separate datasets and trained two distinct models.

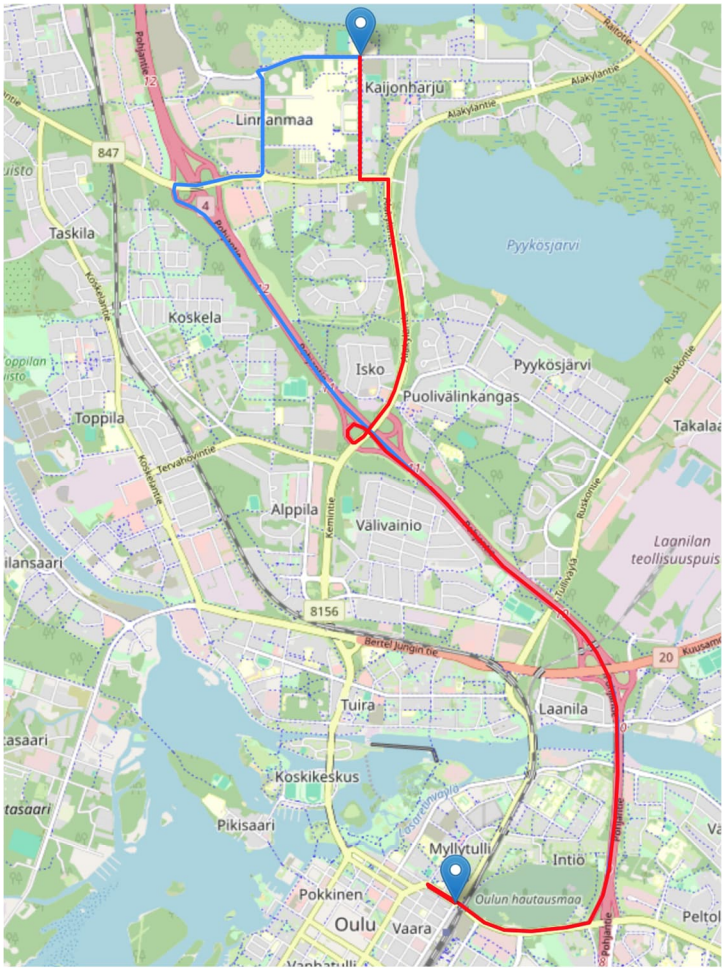

Figure 7. The two different routes choose by different model.

Between the two routes, the red route has slightly more “main_way” segments, and, unsurprisingly, the model that dislikes highways chooses to avoid it.

7. Traffic Announcement Data Collection

The system uses the KG as a knowledge source for decision making. The customer requested the addition of traffic announcement data to the KG from the Digitraffic API. The original system uses TypeScript, so it was used to develop this. The specifics of the data were researched and discussed and relevant datapoints were decided to be used in the KG.

The addition of this traffic announcement data required a component that makes a request to the API and saves the received JSON data in a typed form for further processing. The component returns an array of traffic announcement objects.

The traffic announcement object array is then used to feed the data into the Neo4j graph database that houses the KG. The coordinate data of the traffic announcement is used to link it to existing road segments in the KG so the announcement can be retrieved conveniently when a route is being checked against the KG contents.

The KG area is a specific bounding box around the Oulu area. The traffic announcements are received for the whole country, but only announcements located in municipalities within the bounding box are kept and saved into the KG.

The announcements from Digitraffic do not always have all of the relevant fields filled, so it was important to make sure that the fields can be empty without causing the program to error from either non existing elements in the received JSON, or null elements when creating the typed objects.

This data was not used in the training of the route planning algorithms. A problem with traffic announcements in relation to training was the fact that there were very few announcements for the Oulu area during the development.

We discussed the traffic announcement processing with the customer, and did not have time to come to a conclusion of how to interpret the announcement contents so that correct decisions can be made. This was because the announcements give an affected area, but they tell what has happened in a freeform written format, so it would have to be parsed and interpreted so accurate decisions can be made. The announcement could be that the road speed limit is lowered due to a reason, which would still allow drivers to drive through the area. The announcement could also be that a road is completely closed, requiring an alternative route to be selected. This issue has to be explored further if the system is going to be used in a production environment.

8. Special issue: Lack of variation in KG data

The primary challenge in training the route planning algorithms stems from the limited variation in the available KG data. Since the KG data was collected over a short timeframe in autumn 2024, it failed to capture the diversity of key environmental parameters, such as seasonal weather variations and road conditions. This lack of variation poses an issue for the algorithm’s generalizability. Furthermore, traffic data, identified as one of the most critical parameters for route selection, was entirely absent from the KG. Consequently, the model’s training partially relied on simulated data to compensate for missing traffic information and road condition variability. While this approach allowed for the creation of a route planning PoC model, it also introduced a dependency on artificially generated data, which does not fully reflect real-world scenarios. To address this limitation, future development should focus on collecting a comprehensive and diverse dataset for different KG parameters.

9. Used software development method

The project followed an iterative development approach, where the model and its components were developed, tested, and refined in successive iterations. The methodology was chosen to allow for flexibility in addressing challenges that arose during research and implementation, such as limitations in data variation and the need to simulate missing parameters.

10. Testing / acceptance of the software

Correctness of frontend DQN code was checked by performing several steps. The first step was done by running minimal unit tests on DQN model classes. The purpose at this point was to check that the most critical DQN related code runs without errors. Unit tests used random arguments to functions, so this step did not give any accurate information on how well DQN model was performing. The second step was training of DQN model with training data collected from Open Route Service. The trained model was saved to disk to use it with remaining steps. The third step was to test how well DQN model code was working in theory. The trained model was loaded from disk and testing was done with randomly created test data set of 100000 steps. Code performed correctly after checking plotted data. Main thing here was that amount of nonoptimal values of plotted data started to converge more and more to low value by every 10000 steps. That did indicate that the learning aspect of code was working correctly. The final fourth step was to do a similar test on the same trained model, but with using real routes from Open Route Service as a test data set and limiting tests to 3000 steps. This resulted in there being less optimal values in plot compared to third step plot. If tests were to run without trained model loaded, data in plot had always only one third of optimal values. As a conclusion code was working correctly in all tests according to plots.

Traffic announcement fetching was tested manually, and unit tests were created for checking if the API is available. Saving the data into the knowledge graph was tested continuously tested manually during the development. Results were presented to the customer, and they were accepted based on the presentation.

11. Conclusion

In this project, we successfully developed and evaluated PoC level route planning algorithms by using DQN and DL models. The integration of KG and ORS API enabled the generation of enriched datasets, which formed the foundation for training both DQN-based and conventional DL models. These PoC implementations explored dynamic and adaptive solutions to address the complex challenges of route optimization under varying environmental and user-specific conditions.

The key contributions of the project include:

1. The development of two distinct DQN workflows, which demonstrated the potential to adapt to dynamic conditions.
2. The creation of a conventional DL model aligned with simulated user preferences, providing a basis for future user-oriented route planning solutions.
3. The integration of HF mechanisms, which allowed the DQN models to evolve and align more closely with individual user needs over time.

Performance evaluations showed that the DQN models, particularly those utilizing dynamic reward systems, achieved notable success in balancing multiple factors, such as traffic, distance, and road conditions. The node-level analysis further enhanced decision-making by incorporating granular features, leading to safer and more contextually aware route recommendations. However, it is important to note that these models were developed as PoC implementations and require further refinement and validation before deployment in real-world applications.

While the PoC algorithms demonstrated significant promise, the project faced challenges related to the limited variability of real-world data in KG. This limitation necessitated the use of simulated metrics to augment training scenarios, highlighting the need for more diverse real-world inputs to enhance model generalizability.

In conclusion, this project demonstrated the feasibility of combining data-driven insights with advanced learning algorithms to develop intelligent and adaptive route planning systems. The PoC algorithms provide a strong

foundation for future advancements, serving as a stepping stone toward scalable, real-world applications.

12. Future Work

Based on the results of this project, future efforts could focus on the following:

- Enhanced Data Integration: Incorporating real-time traffic, weather, and road condition updates into the KG to improve accuracy and adaptability.
- Scalability: Refining the models to handle larger and more complex datasets, including multi-regional KGs.
- Real-World Testing: Deploying the system in practical environments to validate its performance and adaptability under real-world constraints.

13. Project timeline

Phase	**Duration**
Requirements Analysis	4 weeks
Data Collection and Integration	6 weeks
Model Development	8 weeks
Model Testing	4 weeks
Documentation	4 weeks

14. Feedback to the course organizers

There could have been more involvement from the course side to help organize things easier. The amount of freedom given to the students is nice and teaches a lot about individual responsibility in project work, but it can also be harmful for the advancement of the project if miscommunication happens within the project team.