Supplementary Material: Code Explanation

Author: Ujwal Sai Alla

Date: October 5, 2025

1. Overview

This document provides a detailed explanation of the Jupyter Notebook (`dqn-think-ai.ipynb`), which contains the complete Python implementation for the "Learning-Based Path Planning for Optimal Terrain Traversal using Deep Reinforcement Learning" paper.

The code implements a custom 2D grid environment with variable terrain features and trains a Deep Q-Network (DQN) agent to find qualitatively superior paths. The agent's performance is then benchmarked against three classical pathfinding algorithms: **A***, **Breadth-First Search (BFS)**, and **Depth-First Search (DFS)**.

---

2. System Requirements & Dependencies

To run this code, you will need a Python environment with the following libraries installed. The versions listed are recommended for reproducibility.

- **Python**: 3.8 or newer
- **PyTorch**: 1.8.0 or newer
- **NumPy**: 1.20.0 or newer
- **Matplotlib**: 3.4.0 or newer

You can install the required packages using pip:

```bash
Bash

pip install torch numpy matplotlib jupyter
```

---

3. Code Structure and Key Components

The entire implementation is contained within the `dqn-think-ai.ipynb` Jupyter Notebook. The key components are organized into classes and functions:

- **`GridEnvironment` Class**: This class defines the simulation environment. It procedurally generates a 2D grid with features like high-smoothness "highways," obstacles, and speed breakers. It handles the state transitions and reward calculations based on the agent's actions.

- **`DQNAgent` and `QNetwork` Classes**: This is the core of our learning-based planner. The `QNetwork` defines the dual-input neural network architecture, which processes both local perception and a broader surroundings map. The `DQNAgent` class orchestrates the agent's behavior, including the epsilon-greedy action selection, learning from experience, and updating the network weights.
- **`PrioritizedReplayMemory` Class**: This class implements **Prioritized Experience Replay (PER)**. Instead of uniform sampling, it prioritizes transitions with a high Temporal-Difference (TD) error, allowing the agent to learn more efficiently from "surprising" or important events.
- **Classical Algorithm Classes (`AStar`, `BFS`, `DFS`)**: These classes contain standard implementations of the baseline algorithms used for performance comparison. The A* algorithm is also used as a "terrain-aware expert" to generate demonstrations.
- **`generate_expert_demonstrations()` Function**: This function runs the terrain-aware A* expert to generate high-quality paths. The experiences from these paths are used to pre-fill the replay memory, bootstrapping the agent's learning process and ensuring it starts with a competent baseline policy.
- **`run_simulation_and_compare()` Function**: This is the main executable function in the notebook. It initializes the environment and all algorithms, runs the DQN training loop, and then evaluates the final trained agent against the classical methods on a new test map. Finally, it calculates and prints the five key performance metrics: Path Length, Execution Time, Average Speed, Jerkiness, and Average Terrain Smoothness.

---

4. How to Run the Code

1. **Launch Jupyter**: Open a terminal, navigate to the project directory, and run the command `jupyter notebook`.
2. **Open the Notebook**: In the Jupyter interface that opens in your browser, click on `dqn-think-ai.ipynb`.
3. **Execute the Code**: The simplest way to run the entire experiment is to click **"Cell" -> "Run All"** from the top menu. This will execute all the code blocks sequentially.
4. **View Results**: The training progress will be printed in the output of the training cell. The final output of the notebook will be a printed table comparing the metrics for all four algorithms and a bar chart visualizing these results.

You can modify key **hyperparameters** such as `GRID_SIZE`, `NUM_EPISODES`, and learning rates in the "Hyperparameters and Setup" section of the notebook to conduct further experiments.