Machine Learning Engineer Assignment

This assessment is divided into a number of steps, designed to build upon each other. They will give you a feel for the type of work we do and help us understand your skills in areas crucial to this role. The project should be completed within one week.

Please submit your work as a link to a single public Git repository containing the solutions for all steps. We expect to see a clear and logical commit history that reflects your progress through the tasks. The code that you will submit should be developed in Python.

Part 1: Project Setup

Your first task is to set up a clean, self-contained project.

1. **Create a Git Repository**: Start by creating a new public GitHub repository for this project.
2. **Set Up a Python Environment**: Create a self-contained Python virtual environment (using `venv` or a similar tool).
3. **Manage Dependencies**: Create a `requirements.txt` file to list all the necessary Python libraries for your project.

Part 2: The Stacking Grid Environment

You will design a custom environment using the gymnasium library. The environment is represented as a 2D grid of size $M \times N$, where a robot must navigate to locate a box placed at a target position (see the figure below).

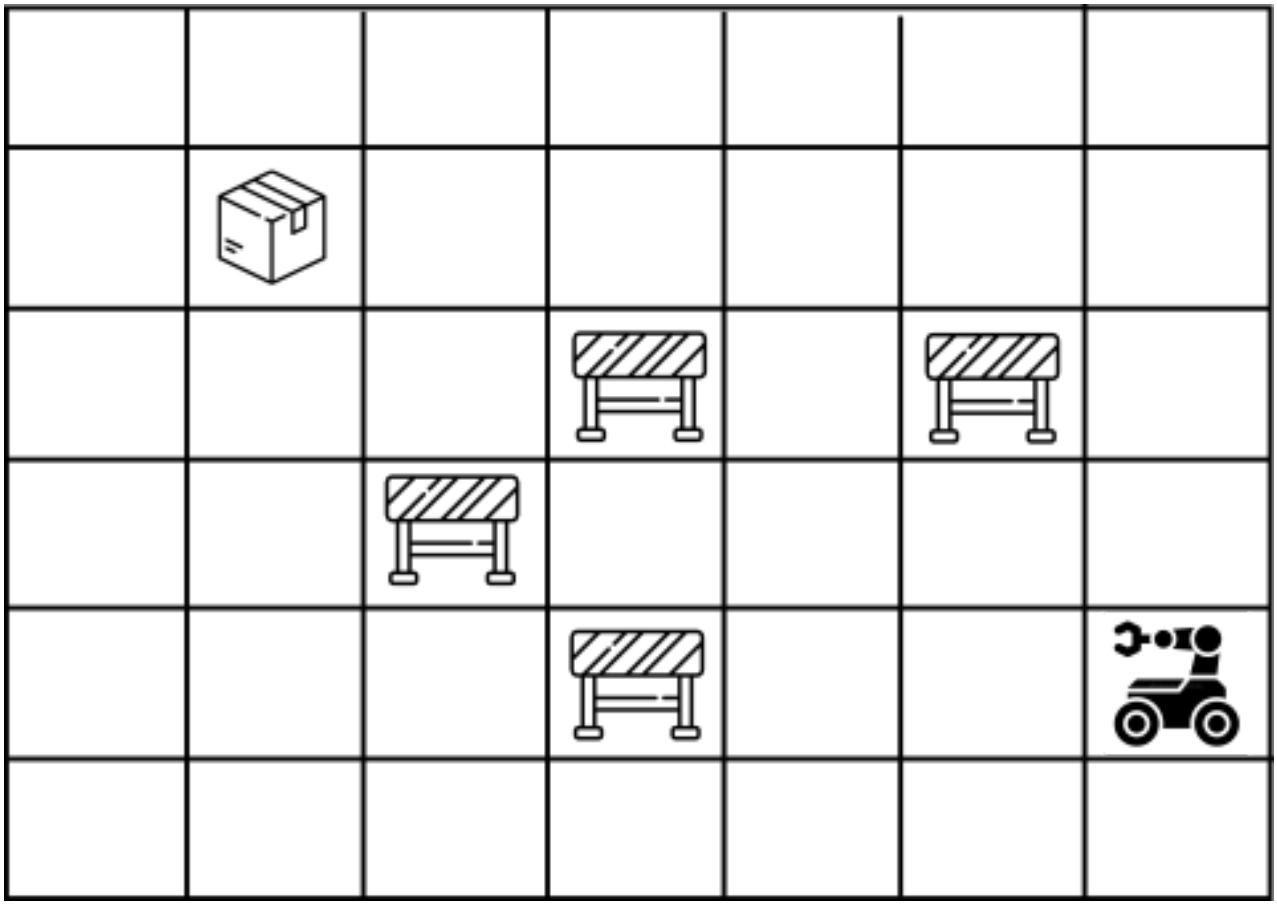
Within the grid, there are K obstacles that the robot must avoid. Both the grid size and the number of obstacles should be configurable. At the start of each episode, the following are randomized:

- The robot's initial position
- The positions of the obstacles
- The position of the goal

The robot can move **up, down, left, or right** (no diagonal movement allowed).

An episode ends if:

- the robot collides with an obstacle,
- the robot moves outside the grid boundaries,
- reaches the target position where the box is located.



Your Task: Implement the `GridEnv` class, inheriting from `gymnasium.Env`. You will need to define the `__init__`, `reset`, `step`, and `render` methods.

Part 3: Agent Training and Evaluation

You will use the `stable-baselines3` library to train and evaluate two types of agents.

Training

- **Choose Algorithms**: Select one on-policy algorithm (e.g., **PPO** or **A2C**) and one off-policy algorithm (**DQN** is recommended).
- **Training Script**: Write a script to train your chosen agents on the `GridEnv`. The training should continue until the agent's performance converges. Save the trained models.
- **Default Parameters**: Use a 6x6 grid with 5 obstacles for training.

Evaluation

- **Evaluation Script**: Create a separate script to load the trained model and evaluate its performance over 100 episodes.
- **Metrics**: The script should print the following metrics: average steps per episode, average total reward per episode, success rate (the percentage of episodes where the robot reaches the target).

Part 4: Submission

Submit a link to your public GitHub repository containing the complete solution.

- **Project Code**: All the code from Parts 1-3.
- **README.md**: A comprehensive `README.md` file. This file should explain:
    - How to set up the project environment.
    - How to train the agent.
    - How to run the evaluation scripts.
- **Design Analysis**: A written analysis in a separate file of your design choices.
    - **Environment Design**: Explain your choices for the state representation and the reward function. How do these choices affect the agent's learning?
    - **Algorithm Comparison**: Compare the performance of the on-policy and off-policy algorithms you used. Which one performed better, and why do you think that is?
    - **Challenges**: Describe any significant challenges you encountered and how you solved them.

Bonus 1: Dockerization.

- **Dockerfile:** Create a `Dockerfile` that uses the `ubuntu:22.04` base image and installs all project dependencies.
- **docker-compose.yml**: Create a `docker-compose.yml` file to build and manage your container.
- **README.md Update**: Add a section to your `README.md` explaining how to build and run the Docker container.

Bonus 2: Expanded Environment

- **Modify `GridEnv`**: Introduce "bonus positions" that the robot must visit before reaching the final goal.
- **Update Logic**: Adjust the state representation, the reward function and termination conditions to reflect this new objective.
- Train an agent of your choice on the expanded environment and report the results.