Assignment Report

Reinforcement Learning for Testing a Software System

Mohammed Aitezazuddin Ahmed

100829388

Abstract

This report describes the process of building and training a reinforcement learning (RL) agent to test a simplified "WebFlow" software environment. I used Proximal Policy Optimization (PPO) to train an agent that can complete a multi–step workflow while avoiding invalid actions. The final model performed consistently with an error rate of 0.00 across all evaluation episodes. The report also includes my own reflections on the challenges I faced, especially with environment imports, Python versions, and figuring out how to structure the project.

1. Introduction

The purpose of this assignment was to apply reinforcement learning to the problem of software testing. Instead of manually designing test cases, the idea is to let an RL agent interact with a software-like environment and learn valid (and invalid) behaviours through trial and error.

For my project, I built a Gymnasium-compatible environment called `WebFlowEnv`. It simulates a small multi-step workflow similar to simple online forms or navigation systems. The goal for the RL agent is to start at the first page and navigate to the final "Done" page using a sequence of valid actions.

This assignment required:

- Creating custom RL environments,
- Defining reward functions,
- Writing configuration files (YAML),
- Training a PPO model for 50,000 timesteps,
- Evaluating the trained model,
- Producing plots and analysis using a notebook.

Even though the task was not extremely complex, getting all the technical pieces to work together

taught me a lot about RL workflows and debugging.

2. Environment Design

2.1 WebFlow Environment

The WebFlow environment is a deterministic state machine with the following states:

- Home
- Form
- Confirm
- Submit
- Done

The agent interacts using discrete integer actions. For every state, only certain actions are valid. The episode ends when the agent reaches "Done" or when it exceeds 20 steps.

2.2 Reward Function

The reward design was kept simple:

- $+1.0$ for reaching the final "Done" page,
- $+0.1$ for valid forward transitions,
- -0.2 for invalid actions.

I intentionally avoided complicated reward shaping so the agent learns a very clear objective.

2.3 FlappyGame Environment

I also included a small `FlappyGameEnv` for completeness, as the assignment required multiple environments. However, I didn't train on it because the WebFlow environment was the main focus of my work.

3. Algorithm: Proximal Policy Optimization (PPO)

I chose PPO because:

- It is stable and widely used,
- Works well with discrete actions,
- Has good support in Stable-Baselines3.

Key hyperparameters from my YAML file included:

- Policy: MLP,
- Learning rate: 0.0003,
- Steps per rollout: 2048,
- Discount factor: 0.99,
- Batch size: 64.

Overall, PPO worked well for the environment.

4. Training Process

Training was done using the command:

```
python src/train.py --algo ppo --app web --persona explorer --timesteps 500
```

The "explorer" persona added more exploration noise early in training. The agent started performing well quite early (around 8–10k timesteps).

Throughout training, I monitored:

- KL divergence,
- Entropy,
- Policy gradient updates,
- Value function loss.

By the end of 50k timesteps, the agent's behaviour was very stable.

5. Evaluation

Evaluation was performed using:

```
python src/evaluate.py --model_path models/web_ppo_explorer_seed7.zip
```

This produced a CSV file in the logs directory. I then loaded this file inside my `analysis_web.ipynb` notebook to generate:

- Reward curves,
- Steps per episode,
- Error rate,
- State visitation graphs,
- Moving averages.

5.1 Key Results

- Error rate: **0.00** (all episodes completed)
- Rewards ranged from 10 to 16
- All episodes reached the final state

The model behaved exactly as expected.

6. Discussion and Reflection

The PPO agent learned the WebFlow task very reliably. Since the environment is deterministic, the final behaviour of the model is very consistent.

From my side, the most difficult parts of the assignment had nothing to do with PPO itself. The hardest challenges were:

- Fixing module import paths,
- Dealing with Python version problems,
- Setting up the virtual environment,
- Adding `__init__.py` files properly,
- Figuring out how to push the project to GitHub.

Even though these issues were annoying, they helped me understand how real RL projects are structured and how important the project environment is.

7. Conclusion

This project successfully demonstrated how reinforcement learning can be used to test a simple software-like system. The PPO agent learned to complete the WebFlow workflow with perfect reliability.

If I continue this project, I would like to:

- Add randomness to make the tests more realistic,
- Include branching paths in the workflow,
- Compare PPO against A2C or DQN.

Overall, the assignment gave me hands-on experience with RL pipelines and improved my understanding of training, evaluation, and debugging.

References

- Schulman et al., "Proximal Policy Optimization Algorithms" (2017)
- Stable-Baselines3 documentation: `https://stable-baselines3.readthedocs.io/`
- Gymnasium documentation: `https://gymnasium.farama.org/`