Research Internship Report

Yash Bansal
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Internship Details

- **Institution:** Saarland University, Foundations of Artificial Intelligence Group
- **Professor:** Prof. Joerg Hoffmann
- **Supervisor:** Songtuan Lin
- **Duration:** 14th May 2025 – 7th July 2025
- **Topic:** Training Neural Action Policies for Planning Benchmarks in JANI

1 Overview

The primary objective of this internship was to train neural action policies using reinforcement learning for planning benchmarks modeled in JANI. The existing PLAJA framework developed by the group provided a baseline implementation using Deep Q-Learning for smaller feedforward neural networks. My goal was to explore more complex architectures and experiment with larger networks to improve performance.

2 Work Description

2.1 Understanding JANI and PLAJA

- I first familiarised myself with the JANI format and various planning benchmarks.
- I studied the PLAJA codebase and the research paper to understand how neural action policies are trained using various algorithms.
- I learned how Deep Q-Learning is implemented, including policy structures, architectures, and reward functions.

2.2 Initial Experiments with PLAJA

- Initially, I tried to train neural action policies using PLAJA, but I encountered several build and runtime issues. Even after debugging, the issues remained.
- So, I switched to building a new training framework using Stormpy, Gymnasium, and Stable-Baselines3.

2.3 JANI Model Loading and Building using Stormpy

- I parsed the JANI benchmark files using Stormpy and extracted the built model and property specification (goal property in our case).
- I analysed the model structure (automata count, number of states, initial state, action space, and transition matrix) and verified that the goal property is received correctly.
- I inspected the model class given by Stormpy and explored the functions and attributes available to understand how to use the model effectively to build the Gymnasium environment.

2.4 Gymnasium-Compatible Environment

- I implemented a custom Gymnasium environment as required by Stable-Baselines3 for policy training.
- I built the state space, action space, and transition dictionary for proper mapping.
- I implemented the following core environment methods:
  - `reset()`: resets the environment to the initial state.
  - `step(action)`: takes an action and moves to the next state as per the model. Rewards are given as:
    - Invalid action: reward = -1
    - Goal reached: reward = +100
    - Valid step (non-goal): reward = -0.1

    For probabilistic transitions, the next state is chosen according to the given probabilities. Invalid actions do not change the state and give a high penalty. Reaching the goal gives a high reward. The episode ends if the goal is reached or the maximum steps limit is crossed.
  - `render()`: visualizes the current state, goal status, and available actions.
  - `get_valid_actions()`: returns the list of valid actions and handles constraints.

2.5 Reinforcement Learning with Stable-Baselines3

- I used Stable-Baselines3 to train RL policies using Deep Q-Learning on the Gymnasium environment. I also checked compatibility to make sure everything worked correctly.
- The neural network architecture was configurable. I experimented with networks ranging from 2 to 5 layers.
- I performed ablations and tuned the following hyperparameters:
  - Learning rate
  - Learning start threshold (number of steps before training starts)
  - Exploration fraction (fraction of training episodes with high exploration)
  - Final exploration epsilon

  - Total timesteps

- After training, I evaluated the model's goal-reaching ability by running multiple simulations under both deterministic and probabilistic transitions.

3 Experiments and Analysis

3.1 Benchmark Domains

The main experiments were done on two JANI benchmark domains: `blocksworld_5` and `elevators3-3`.

3.2 Neural Network Architecture Analysis

Initial experiments with a 2-layer neural network [64, 64] on `elevators3-3` showed that the model was not learning well. Using a deeper 5-layer architecture gave better results. However, the 2-layer network worked fine for the `blocksworld_5` benchmark.

3.3 Elevators3-3 Domain Analysis

The `elevators3-3` domain has around 1000 states and 27 goal states.

Episode length: In the beginning, training with a max of 2000 steps per episode gave poor results because the agent rarely reached the goal. So it didn't get any positive rewards early on. Increasing the max steps to 3000 helped solve this.

Exploration rate: A low exploration fraction led to fast initial learning (within 50 episodes), but the model didn't explore enough states. So during testing, it would get stuck due to probabilistic transitions. Increasing the exploration fraction from 0.3 to 0.6 increased the training time (around 200 episodes) but improved testing success rate from 20% to 50%.

3.4 Reward Function Analysis

The reward structure had a big impact on learning and policy behavior.

Initial Reward Design: When both invalid actions and non-goal steps had similar negative rewards, the model avoided valid steps and preferred invalid actions. This was not useful for learning.

Optimized Reward Design: The final reward setup (+100 for goal, -1 for invalid, -0.1 for valid step) worked well. The model quickly learned to avoid invalid actions (within 20 episodes) and then focused on reaching the goal. Within 10 more episodes, the agent started consistently reaching the goal during training and testing.

3.5 Blocksworld_5 Domain Analysis

The `blocksworld_5` domain has around 1100 states and only one goal state.

Network Architecture: A simple 2-layer network was enough for learning this environment. This is because the action space is small and the benchmark is simpler.

Training: Due to the single goal state, training required more episodes (around 1000) and higher exploration to find the goal during early training.

Goal-Directed Behavior: After training, the model could reach the goal in 20–30 steps, which is much lower than in `elevators3-3`, where it took around 150–200 steps.

4 Conclusion

This internship gave me hands-on experience with reinforcement learning for planning tasks using JANI models. I developed a custom Gymnasium environment using Stormpy and Stable-Baselines3, experimented with neural network architectures, and fine-tuned training setups. Through this, I learned how model structure, reward design, and exploration strategies affect learning. Overall, the project helped me build a strong foundation in applying RL to formal planning domains.