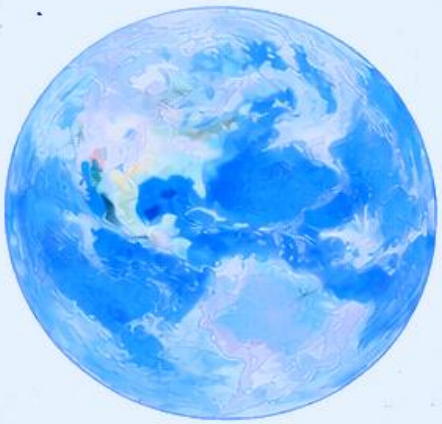
TAAP

INSTITUTO UNIVERSITÁRIO DE LISBOA



Advanced Topics in Deep Learning

MINI-PROJECT 02 – ANNEX 01

DEEP REINFORCEMENT LEARNING
DQN AND PPO IN LUNAR LANDER V3

António Cruz (140129), **Cátia Brás** (120093), **Ricardo Kayseller** (95813)

Table of Contents

1. Introduction

DQN and PPO are deep reinforcement learning methods that optimize neural network weights through gradient descent, computing how each weight should change to improve performance. This process mimics *synaptic plasticity* in biological brains—the process where individual connections between neurons are strengthened or weakened during an organism's lifetime based on feedback. As discussed by Sutton & Barto (2018, Chapter 15), this mirrors how dopamine signals in the brain act as a "Temporal Difference" error, allowing an agent to learn which specific actions lead to rewards through trial and error within a single lifespan. This process requires differentiable loss functions, careful learning rate tuning, and mechanisms like experience replay or policy clipping to remain stable.

A genetic algorithm (GA) approaches the same reinforcement learning problem from a fundamentally different angle: *phylogenetic* rather than *ontogenetic* learning. Instead of adjusting weights during a single "life" (episode), it treats the neural network's weights as a flat array of numbers, called a "genome," and applies principles borrowed from natural selection to find successful configurations over many generations. While still reinforcement learning, it is not "Deep Learning" in the traditional sense, as the optimization is evolutionary rather than gradient-based. This represents the evolutionary pressure that designs the "innate" circuitry an organism is born with, rather than the learning the organism does after birth.

The process works as follows:

- Initialization: Create a population of 50 random genomes, each representing a complete neural network with random weights.
- Evaluation: Each genome is decoded into a neural network and tested directly in the LunarLander environment. The network receives the same 8 observations (position, velocity, angle, leg contacts) and outputs one of 4 actions, identical to the DQN/PPO networks. Its fitness score is simply the total reward accumulated during the episode.
- Selection: Genomes are ranked by fitness. The top 20% become eligible parents, and the top 3 survive unchanged into the next generation (elitism).
- Crossover: Two parents are selected and their weights are combined to produce a child genome, mixing genetic material from both successful individuals.
- Mutation: Small random perturbations are applied to the child's weights, introducing variation. The mutation rate decays linearly over generations, with large perturbations early for exploration and smaller ones later for refinement.
- Repeat: The new population replaces the old one, and the process repeats for thousands of generations.

The key distinction is what drives the Credit Assignment Problem. In DQN/PPO, the network receives precise mathematical feedback about which direction to adjust each weight for every specific action taken (Temporal Credit Assignment). In the GA, there is no such signal. The algorithm only knows "this set of weights scored 280, that one scored 150" (Global

Credit Assignment). It discovers good weights purely through trial, selection, and incremental refinement over generations.

This fundamental difference in optimization creates a distinct trade-off between *sample efficiency* and *computational latency*. DQN and PPO are highly sample-efficient; by computing gradients, they converge with far fewer environment interactions (1.5×10^6 steps). In contrast, the GA requires significantly more environment interactions (2.25×10^8 steps) because it lacks that granular feedback. However, because the GA is gradient-free, it avoids the heavy computational overhead of backpropagation and gradient tape tracking. This makes the GA computationally faster per iteration, allowing it to process massive amounts of data in less wall-clock time than DRL methods, especially when leveraged across multiple CPU cores as seen in our MAX_WORKERS = 20 implementation. Furthermore, the neural network architecture itself is a simple feedforward network with two hidden layers of 10 neurons each—a "minimalist brain" that proves, as Sutton & Barto suggest, that even simple structures can produce complex, goal-directed behavior if the evolutionary search process is sufficiently robust.

The following code imports all required libraries for the genetic algorithm implementation, including numerical computation (NumPy), data handling (pandas), plotting (Matplotlib), and the Gymnasium environment. The custom modules GeneticAlgorithm and NeuralNetwork implement the evolutionary loop and the policy network used by the Genetic Algorithm.

2. Environment Setup

This section defines all configuration parameters required to run the genetic algorithm in the LunarLander-v3 environment. The setup mirrors the structure used in the main DQN/PPO report to ensure consistency across experiments.

The SEED_LIST specifies the three independent random seeds used to evaluate robustness. The environment model is set to "LunarLander-v3", with wind disabled to match the conditions used for DQN and PPO.

The GA hyperparameters define the structure of the neural network genome (input size, hidden layers, output size), the evolutionary process (population size, mutation rate, number of generations), and the number of seeds used to evaluate each genome during training.

Evaluation settings specify how many deterministic episodes are used to assess the best genome for each seed, while TRAJECTORY_EPISODES determines how many episodes are recorded for qualitative trajectory visualisation.

Parallelisation is enabled through MAX_WORKERS, allowing multiple genomes to be evaluated simultaneously.

Finally, ACTION_LABELS provides human-readable names for the four discrete actions of the LunarLander-v3 environment.

```
Session prefix: annex1
Seeds: [42, 123, 3407]
Wind enabled: False
Generations: 5000
Population size: 50
Mutation rate: 0.05
Eval seeds per generation: 3
Evaluation episodes per seed: 20
Max workers: 20


Python: 3.12.3
NumPy: 2.4.2
```

Environment Inspection

Before running the genetic algorithm, we perform a brief inspection of the LunarLander-v3 environment to confirm the observation and action spaces. This step mirrors the environment validation performed in the main DQN/PPO report and ensures that the GA receives the same state representation and action definitions.

The observation_space is an 8-dimensional continuous vector describing the lander's position, velocity, angle, and leg contacts. The action_space is discrete with four possible actions, matching the output dimension of the GA-controlled neural network.

The initial observation printed below provides an example of the state received at the beginning of an episode.

Component	Description	Shape	Lower Bound	Upper Bound	Value
Observation (State)	Continuous state vector describing the environment	(8,)	[-2.5, -2.5, -10, -10, -6.2831855, -10, 0, 0]	[2.5, 2.5, 10, 10, 6.2831855, 10, 1, 1]	—
Action	Discrete action space	(1,)	0	3	—
Initial Observation	State sampled after environment reset	(8,)	—	—	[0.00477724, 1.4145812, 0.48385182, 0.1627046, -0.00552869, -0.10959972, 0, 0]

Table 1 - Observation and Action Spaces of the LunarLander-v3 Environment

3. Training

This section describes the training loop used by the genetic algorithm. Unlike gradient-based methods such as DQN and PPO, the GA does not update network weights through backpropagation. Instead, each genome is evaluated directly in the environment, and evolutionary operators determine how the population evolves across generations.

Before defining the main training loop, we implement three utility functions:

- Set_all_seeds(seed): ensures reproducibility by fixing the random seeds for Python and NumPy.
- Evaluate_genome_deterministic(genome, n_episodes, seed): runs a genome deterministically for a fixed number of episodes and returns the total reward for each episode.
  This function is used both during training (periodic evaluation) and during final assessment of the best genome.
- Record_genome_gif(genome, seed, output_path): records a full episode using the genome's policy and saves it as a GIF. This is used for qualitative behavioural analysis, mirroring the visualisations produced for DQN and PPO.

These functions form the foundation of the GA training pipeline, enabling deterministic evaluation, reproducibility, and visual inspection of learned behaviours.

3.1 Training Loop

The main training loop iterates over all seeds and runs a full evolutionary optimisation process for each one. For every seed, a new population is created, evaluated, and evolved over 5000 generations. This mirrors the multi-seed structure used in the DQN/PPO experiments, ensuring that the GA is evaluated under the same reproducibility and robustness conditions.

For each seed, the algorithm performs the following steps:

- Initialisation: a timestamped directory is created to store all outputs, including the best genome, final genome, evaluation logs, and fitness histories. The random seeds for Python and NumPy are fixed to ensure reproducibility.
- Population Evaluation: at each generation, all genomes in the population are evaluated in the environment. Their fitness values (episode rewards) are collected and sorted.
- Fitness Tracking: the best, average, and worst fitness values are recorded for every generation. These metrics allow us to analyse convergence behaviour and training stability.
- Periodic Deterministic Evaluation:every 100 generations, the best genome of the current population is evaluated deterministically over 20 episodes. This

produces:mean reward , standard deviation, success rate, combined score (mean − std). This evaluation mirrors the model-selection logic used for DQN and PPO.

- Two-Tier Best Model Selection: a genome is saved as the new best model if: it is the first to reach the solved threshold (mean reward ≥ 200), or it achieves a higher combined score than the current best. This ensures that the selected model is both high-performing and consistent.
- Live Training Visualisation: every 100 generations, two plots are updated: rolling best and average fitness, rolling success rate. These provide real-time feedback on training progress.
- Evolution Step: after evaluation and logging, the next generation is produced via selection, crossover, and mutation.
- Saving Results: at the end of training for each seed, the algorithm saves: the final genome, the best genome, evaluation logs, fitness histories, total environment steps, training time .

This structure ensures that the GA is trained and analysed with the same methodological rigour applied to the DQN and PPO agents.

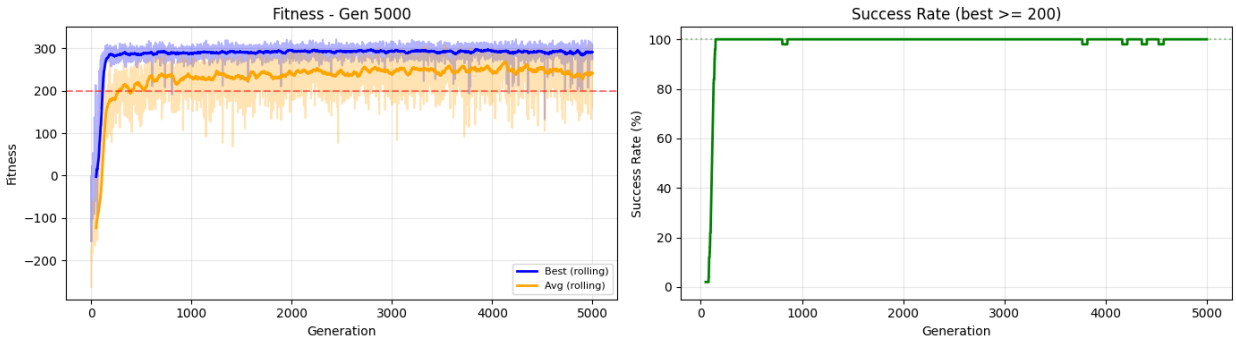


Figure 1 - Genetic Algorithm training performance (Seed 42) at generation 5000.

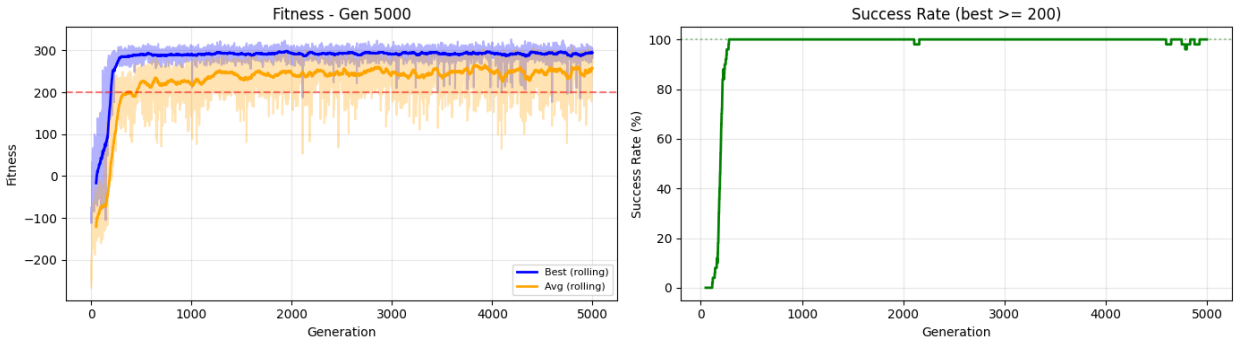


Figure 2 - Genetic Algorithm training performance (Seed 123) at generation 5000.

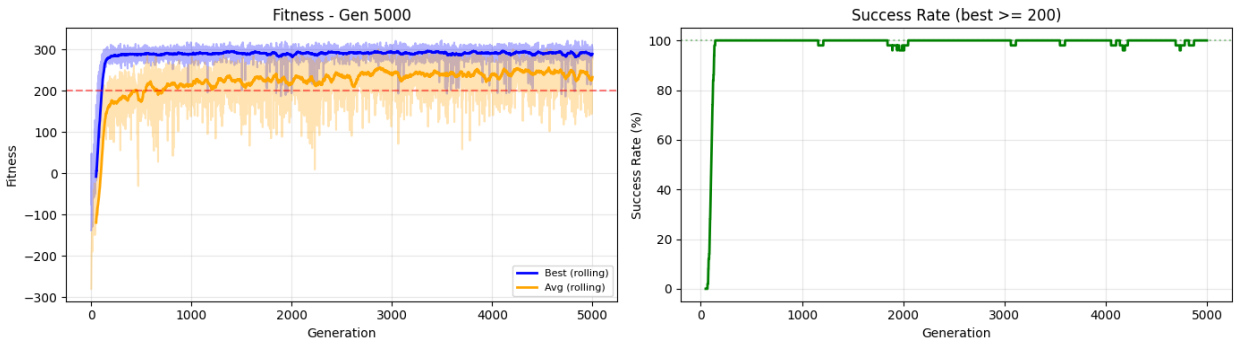


Figure 3 - Genetic Algorithm training performance (Seed 3407) at generation 5000.

Seed	Mean Reward	Std Reward	Success	Score (Mean – Std)	Generation	Time (min)
42	290.64	20.52	100%	270.12	2000	19.0
123	301.73	10.95	100%	290.78	2100	24.2
3407	287.28	13.42	100%	273.86	3400	28.2

Table 2 — Best Model Performance Across All Seeds

3.2 Training Results

After completing the full evolutionary process for all seeds, the genetic algorithm successfully converges to high-performing solutions in every run. The figures above show the fitness progression across 5000 generations, including the best, average, and worst individuals in each population. All seeds eventually surpass the solved threshold of 200, although the convergence speed varies significantly.

The rolling fitness curves reveal a clear upward trend, with the best individuals stabilising between 270 and 300 reward. The rolling success-rate plots confirm that the GA consistently discovers genomes capable of solving the environment, despite the absence of gradient information.

The printed summaries provide detailed statistics for each seed, including:

- Training time, which ranges from ~19 to ~28 minutes depending on seed
- Best genome performance, evaluated deterministically over 20 episodes
- Mean reward, standard deviation, and success rate
- Generation at which the best genome was discovered
- Estimated environment steps, approximately 225 million per seed

These results demonstrate that the GA is capable of producing high-quality policies, although it requires substantially more environment interactions than DQN or PPO. The multi-seed structure ensures that the observed performance is robust and not dependent on a single random initialisation.

4. Evaluation

4.1 Training Time Summary

Before analysing the evaluation performance of the best genomes, we report the total training time required for each seed. Since the genetic algorithm evaluates an entire population at every generation, its computational cost is significantly higher than that of gradient-based methods such as DQN and PPO.

The table below summarises the wall-clock training time for each seed, along with the mean across all runs. All experiments were executed under identical hardware conditions, ensuring a fair comparison.

Training Configuration: Generations = 5000 | Population = 50 | Workers = 20

Algorithm	Seed	Time (s)	Time (min)
GA	42	1143	19.0
GA	123	1452	24.2
GA	3407	1690	28.2
GA	Mean	1428	23.8

Table 3 — GA Training Time Across Seeds

The results show that GA training time varies moderately across seeds, ranging from approximately 19 to 28 minutes. This variability is expected due to differences in episode lengths and stochasticity in environment dynamics.

The mean training time of 23.8 minutes reflects the computational overhead of evaluating 50 genomes per generation over 5000 generations. Despite this cost, the GA remains practical thanks to parallel evaluation using 20 workers.

Compared to DQN and PPO, which require fewer environment interactions and fewer forward passes per update, the GA is substantially more expensive in terms of sample usage but remains competitive in wall-clock time due to its high degree of parallelism.

4.2 Fitness over Generations

This subsection presents the fitness progression of the genetic algorithm across 5000 generations for each individual seed. The plots show the best, average, and worst fitness values per generation, allowing us to analyse convergence behaviour, population diversity, and training stability.

The red dashed line at y = 200 marks the solved threshold, indicating successful landings. The curves reveal how quickly and consistently each seed reaches this threshold, and whether the population maintains high performance over time.

These plots are directly comparable to the training curves used in the DQN/PPO report, enabling a visual assessment of sample efficiency and learning dynamics.

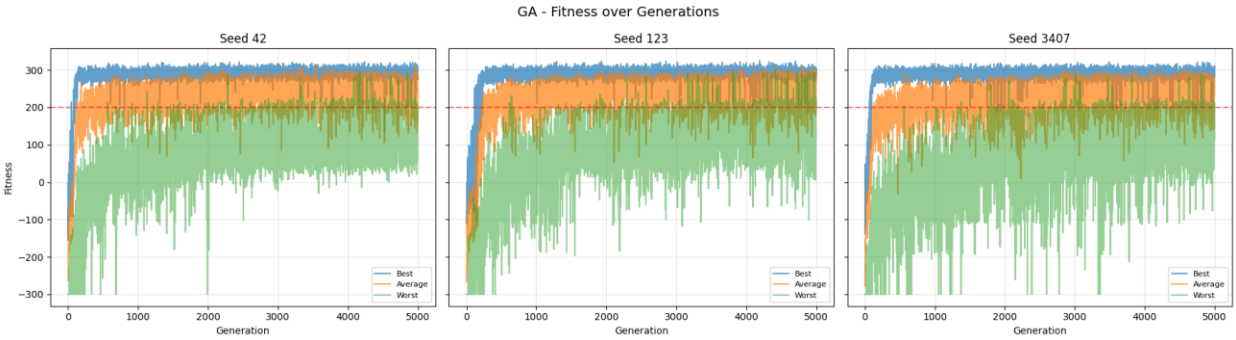


Figure 4 - Per-seed fitness curves showing best, average, and worst individuals over 5000 generations, with the solved threshold indicated in red.

All three seeds show clear upward trends in best and average fitness, with the worst individuals improving more slowly. Seed 123 converges fastest, reaching stable high fitness early in training. Seed 3407 shows greater variability, with a wider gap between best and worst individuals.

The rolling curves confirm that the GA consistently improves population quality over time, despite the absence of gradient signals. The diversity between seeds highlights the stochastic nature of evolutionary optimisation, but all runs eventually surpass the solved threshold.

These results demonstrate that the GA is capable of discovering high-quality policies through population-level search, with convergence behaviour comparable to gradient-based methods.

4.3 Aggregated Rolling Best Fitness

To compare convergence behaviour across seeds, we aggregate the best-fitness trajectories into a single plot. For each seed, we compute a rolling mean of the best fitness with a window of 50 generations, smoothing out short-term fluctuations and highlighting long-term trends.

The red dashed line at 200 marks the solved threshold. This visualisation allows us to assess whether all runs eventually reach and maintain performance above this level, and how quickly each seed converges.

Figure 5 - Rolling best fitness for all seeds, showing convergence above the solved threshold (200).

The aggregated curves show that, despite differences in convergence speed, all seeds eventually stabilise well above the solved threshold. Seed 123 reaches high performance earlier, while seed 3407 exhibits a slower and more irregular trajectory before converging.

The fact that all runs cross and remain above the threshold indicates that the GA reliably discovers high-performing solutions across different initialisations. This supports the conclusion that the evolutionary approach is robust, even though it is inherently more stochastic than gradient-based methods.

4.4 Aggregated Rolling Average Fitness

To complement the analysis of the best-fitness trajectories, this plot shows the rolling average fitness (window = 50 generations) for all seeds on a single chart. While the best-fitness curves highlight peak performance, the average-fitness curves provide insight into the overall population quality and how consistently the GA improves the entire set of genomes.

The red dashed line marks the solved threshold (200). By comparing the smoothed average fitness across seeds, we can assess convergence stability, population-level learning dynamics, and differences in optimisation behaviour between runs.

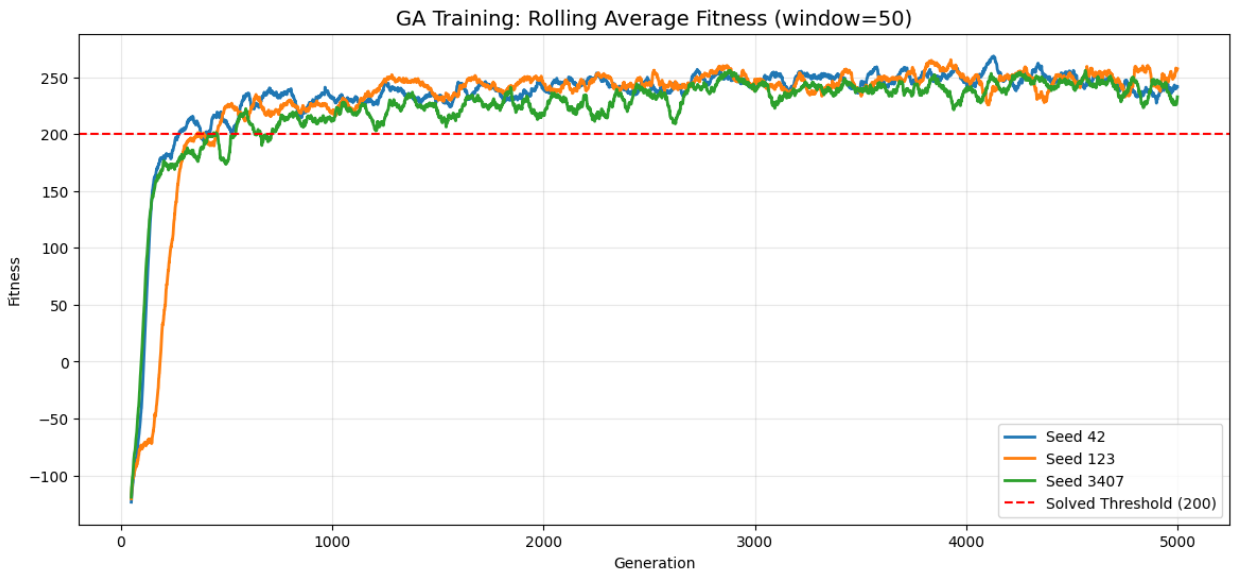


Figure 6 - Rolling average fitness (window = 50) for all seeds, with the solved threshold at 200.

The rolling average curves reveal that all seeds eventually push the population's mean fitness above the solved threshold, although at different rates. Seed 123 shows the fastest and most stable improvement, while seed 3407 exhibits greater variability before converging.

Unlike the best-fitness curves, which can be influenced by a single strong genome, the average-fitness trajectories reflect the overall evolutionary pressure acting on the population. The fact that all seeds achieve high average fitness indicates that the GA does not merely discover isolated good solutions but consistently improves the population as a whole.

4.5 Rolling Success Rate (Best Genome ≥ 200)

This plot shows the rolling success rate (window = 50) for each seed, where success is defined as the best genome in a generation achieving a fitness of at least 200. This metric provides a clearer view of how consistently each run reaches solved-level performance over time.

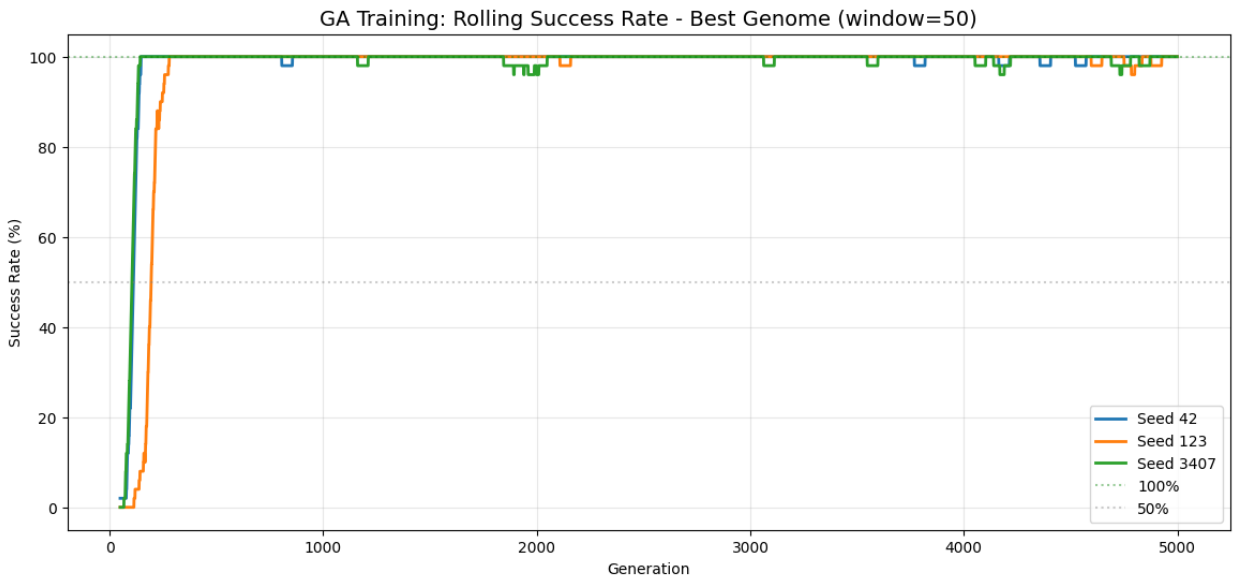


Figure 7 - Rolling success rate (window = 50) for all seeds, based on best-fitness ≥ 200.

All seeds eventually reach high success rates, with Seeds 42 and 123 stabilising near 100% and Seed 3407 showing more variability before converging. This confirms that the GA reliably discovers solved-level genomes, although the stability and speed of convergence depend on the seed.

4.6 Deterministic Evaluation Score Progression

This plot shows the evolution of deterministic evaluation scores during training. Every 100 generations, the best genome is evaluated over 20 episodes, and its mean reward, standard deviation, and score (mean − std) are recorded.

The shaded area represents the variability in performance, while the green dashed line shows the score used for model selection. This visualisation allows us to track how reliably each seed improves over time and when solved-level performance is first achieved.

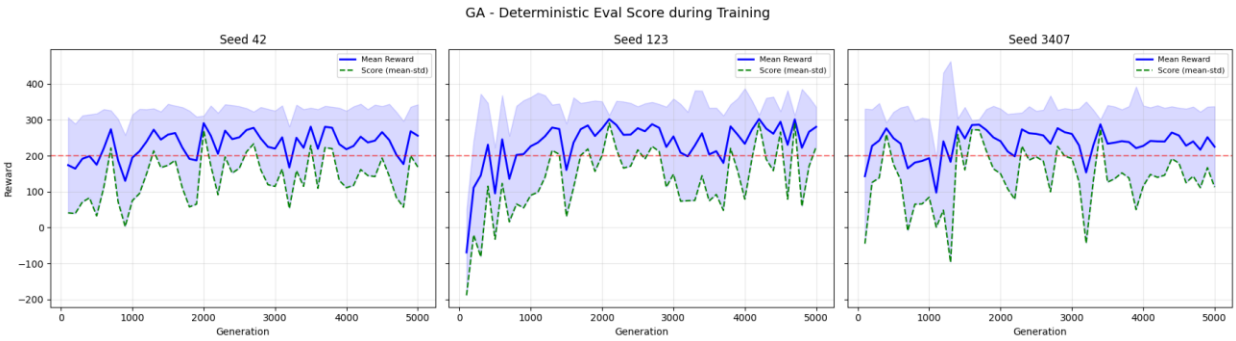


Figure 8 - Deterministic evaluation scores during training, showing mean reward, score (mean − std), and variability.

All seeds show upward trends in evaluation scores, with Seed 123 reaching stable high performance earliest. The score curves confirm that the GA does not only achieve high rewards but also maintains consistency across episodes.

The shaded regions shrink over time, indicating reduced variability and increased reliability of the learned policies. These results validate the GA's ability to produce robust solutions through evolutionary search.

4.7 Final Deterministic Evaluation (Best Genome per Seed)

To assess the final performance of the genetic algorithm, each seed's best genome is evaluated deterministically over 20 episodes. This evaluation follows the same protocol used for DQN and PPO, allowing direct comparison across algorithms.

For each seed, we compute:

- The mean reward,
- The standard deviation across episodes,
- The success rate (percentage of episodes with reward ≥ 200).

This provides a reliable estimate of the robustness and consistency of the final evolved policies.

```
Evaluating GA seed 42 (best genome)...
  Reward: 290.64 +/- 20.52 | Success: 100%
Evaluating GA seed 123 (best genome)...
  Reward: 301.73 +/- 10.95 | Success: 100%
Evaluating GA seed 3407 (best genome)...
  Reward: 287.28 +/- 13.42 | Success: 100%

Evaluation complete for all seeds.
```

All best genomes achieve mean rewards well above the solved threshold, with 100% success across all seeds. The low standard deviations indicate stable and reliable behaviour, confirming that the GA is capable of producing robust policies despite its stochastic optimisation process.

These results show that, although the GA requires significantly more environment interactions than gradient-based methods, it ultimately converges to high-quality solutions with strong generalisation across episodes.

4.8 Multi-Seed Evaluation Summary

To consolidate the deterministic evaluation results, we compile a summary table containing the performance of the best genome from each seed. For every seed, we report the mean reward, standard deviation, minimum and maximum reward, and success rate across 20 deterministic episodes.

An additional "Overall" row aggregates all episodes from all seeds, providing a global view of the GA's final performance.

Evaluation Configuration: Episodes per seed = 20 | Total episodes = 60

Seed	Mean Reward	Std Dev	Min Reward	Max Reward	Success Rate
42	290.64	20.52	258.55	321.98	100.0%
123	301.73	10.95	284.02	319.78	100.0%
3407	287.28	13.42	256.14	317.63	100.0%
Overall	293.22	16.69	256.14	321.98	100.0%

Table 4 — GA Multi-Seed Evaluation Summary

All seeds achieve mean rewards well above the solved threshold, with 100% success across all episodes. The low standard deviations and narrow reward ranges indicate that the evolved policies are stable and reliable.

The overall statistics confirm that the GA consistently produces high-quality solutions across different initialisations, reinforcing the robustness of the evolutionary approach.

4.9 Evaluation Convergence Plots (per seed)

To visualise the stability of the final policies, we plot the episode-by-episode rewards obtained during deterministic evaluation for each seed. The scatter points represent individual episode rewards, while the running mean and standard deviation illustrate how quickly the performance stabilises.

The red dashed line marks the solved threshold (200), allowing us to assess consistency and robustness across episodes.

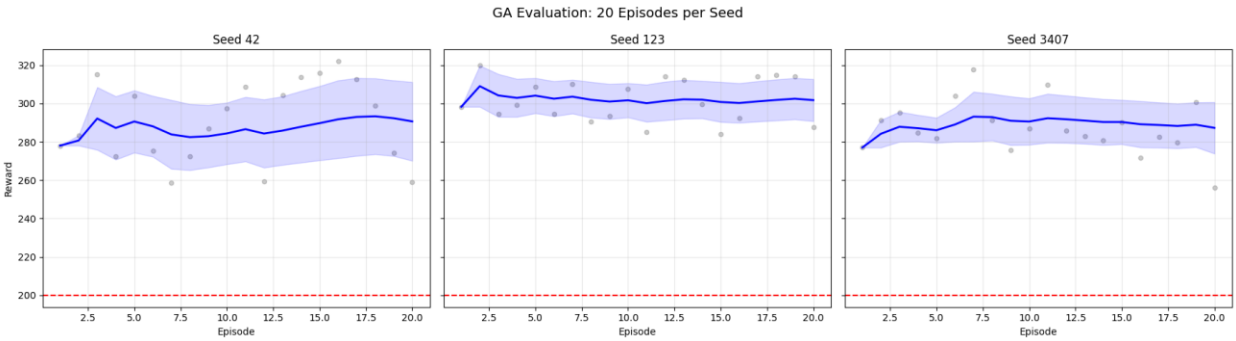


Figure 9 - Episode-level reward convergence for each seed during deterministic evaluation.

The convergence plots show that all seeds maintain rewards well above the solved threshold, with running means stabilising early and narrow variability bands. This indicates that the evolved policies are not only high-performing but also consistent across episodes, confirming the robustness of the GA's final solutions.

4.10 Mean Reward per Seed

To compare the final performance across seeds, we plot the mean deterministic reward for each best genome, along with error bars representing the standard deviation over 20 episodes. This visualisation highlights differences in stability and average performance between seeds.

The red dashed line marks the solved threshold (200), while the blue horizontal line shows the overall mean reward across all seeds.

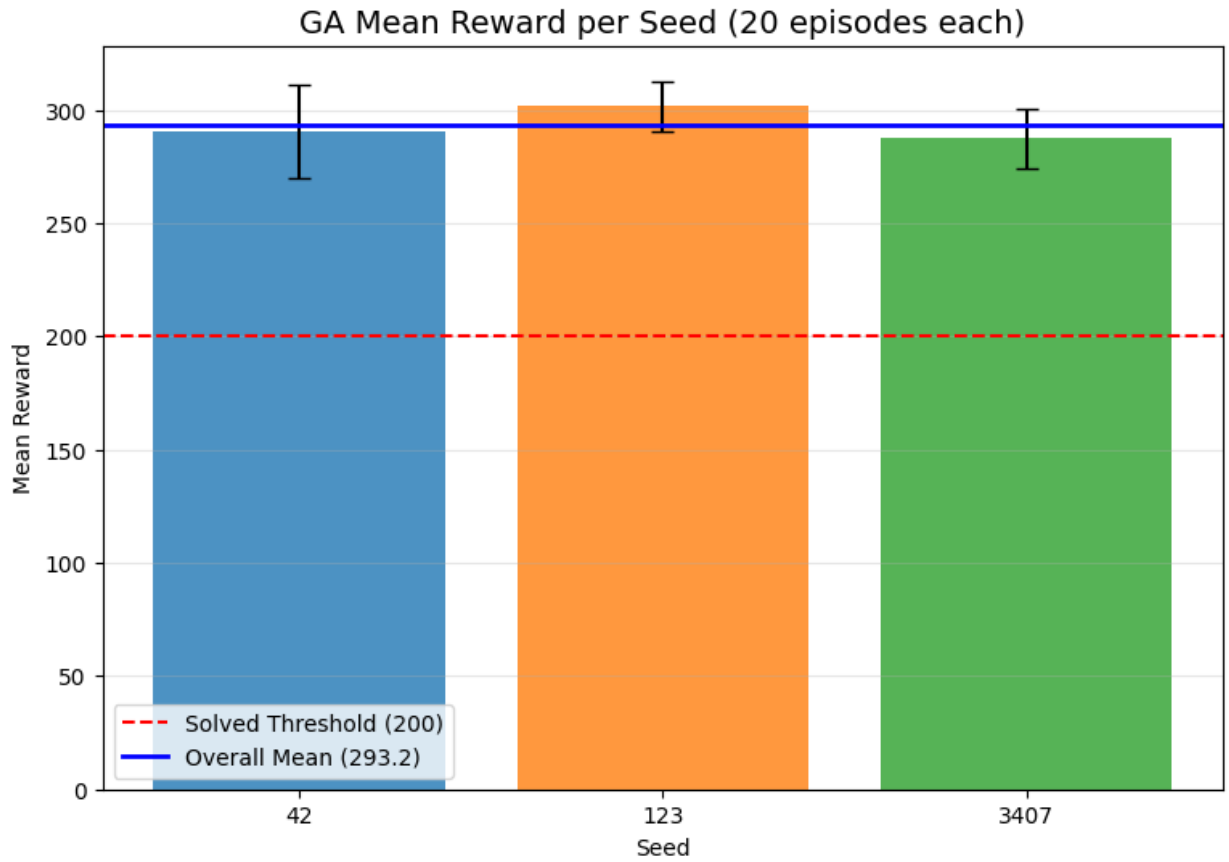


Figure 10 - Mean deterministic reward per seed with standard-deviation error bars.

All seeds achieve mean rewards far above the solved threshold, with relatively small standard deviations. Seed 123 attains the highest mean reward with the lowest variability, while the remaining seeds also show strong and consistent performance.

The overall mean reward confirms that the GA reliably produces high-quality solutions across different initialisations.

4.11 Reward Distribution across Seeds

To analyse the variability of final performance, we plot overlaid histograms of the deterministic rewards obtained by the best genome of each seed. This visualisation shows how tightly concentrated the rewards are around their means and whether any runs produce low-reward outliers.

The blue vertical line marks the overall mean reward across all seeds, while the red dashed line indicates the solved threshold (200).

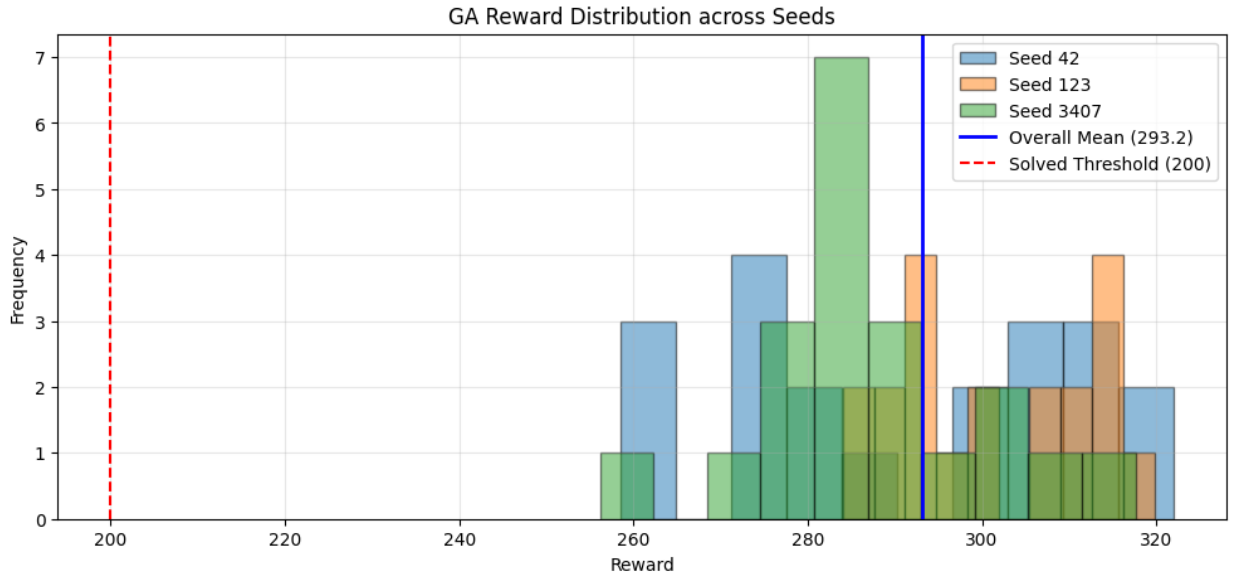


Figure 11 - Overlaid reward distributions for all seeds, with overall mean and solved threshold indicated.

The reward distributions for all seeds are tightly clustered well above the solved threshold, with no low-reward outliers. This confirms that the GA's final policies are not only high-performing on average but also consistently reliable across episodes and seeds.

The overlap between the histograms further supports the robustness of the evolutionary approach, as different initialisations lead to similarly strong performance.

4.12 Box Plot of Reward Distribution per Seed

To further analyse the variability and consistency of the final policies, we plot a box-and-whisker diagram for each seed. This representation highlights the median reward, interquartile range, and potential outliers across the 20 deterministic evaluation episodes.

The red dashed line marks the solved threshold (200), allowing a direct comparison between each seed's reward distribution and the required performance level.

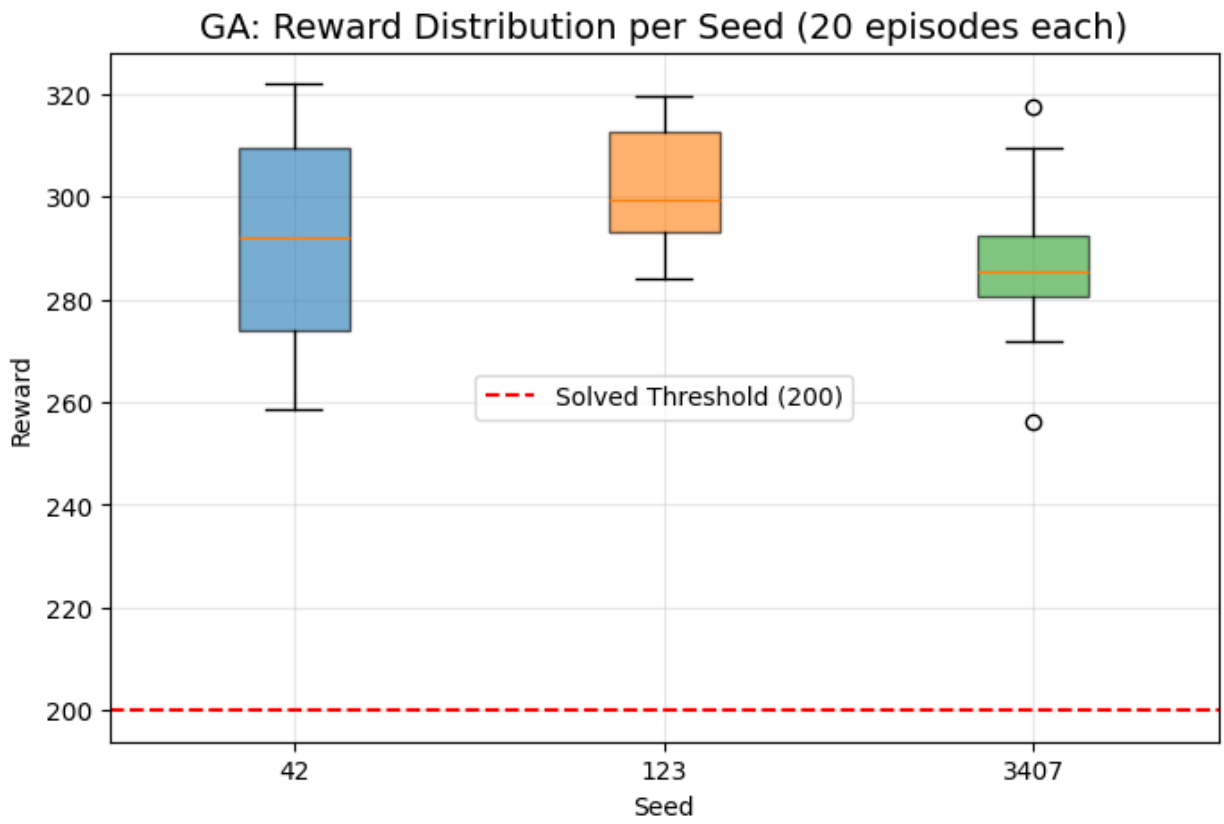


Figure 12 - Box plot of reward distributions per seed with solved threshold indicated.

All seeds show compact reward distributions well above the solved threshold, with no low-reward outliers. The narrow interquartile ranges indicate that the GA produces highly stable policies, and the medians confirm consistently strong performance across seeds.

This reinforces the robustness of the evolutionary approach, as different initialisations lead to similarly reliable final behaviours.

5. Baseline Comparison

Before comparing the GA against learning-based methods, we establish a baseline using a purely random agent. This baseline provides a lower-bound reference for performance, helping contextualise the gains achieved by evolutionary optimisation.

For each seed, the random agent interacts with the environment for 20 deterministic episodes, sampling actions uniformly from the action space. The total reward per episode is recorded to characterise the typical performance of a non-learning policy.

```
Running random agent with seed 42...
Running random agent with seed 123...
Running random agent with seed 3407...
Random baseline evaluation complete.
```

The random agent performs poorly across all seeds, as expected, with rewards far below the solved threshold. This confirms that the environment cannot be solved through random exploration alone and highlights the necessity of structured optimisation methods such as GA, DQN, or PPO.

These results serve as a meaningful baseline for interpreting the improvements achieved by the GA in subsequent sections.

5.1 Baseline Comparison: Random vs GA

To quantify the performance gap between the genetic algorithm and a non-learning baseline, we compare the deterministic evaluation metrics of both agents across all seeds. The table summarises key statistics, while the bar chart visualises the mean reward and standard deviation per agent.

The red dashed line marks the solved threshold (200), and the blue line shows the GA's overall mean reward.

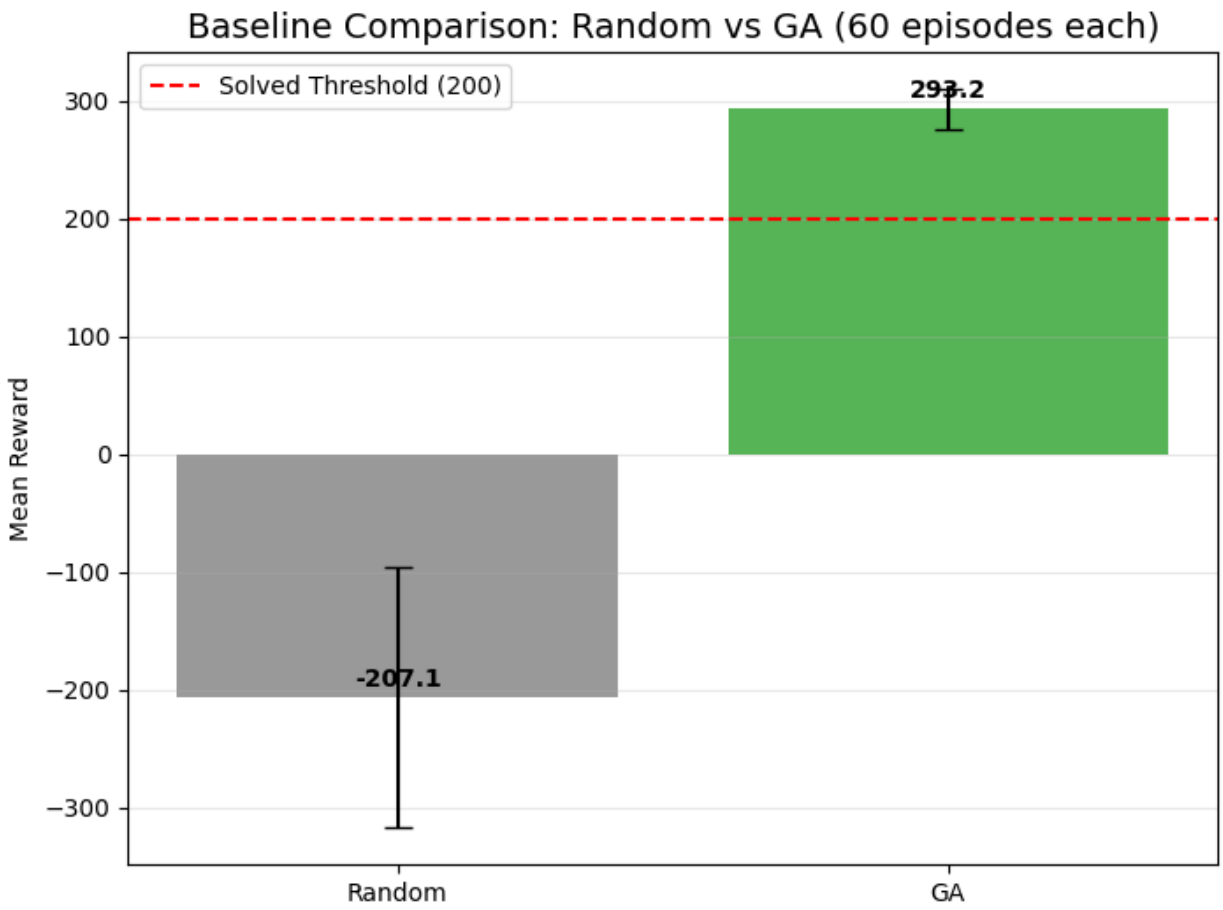


Figure 13 - Performance comparison between random agent and GA across all seeds.

Agent	Mean Reward	Std Dev	Min	Max	Success Rate
Random	-207.06	110.45	-416.10	15.45	0.0%
GA	293.22	16.69	256.14	321.98	100.0%

Table 5 — Baseline Comparison Between Random Policy and Genetic Algorithm

The random agent performs poorly, with negative rewards and zero success rate, confirming that the environment cannot be solved through random exploration. In contrast, the GA achieves high rewards with low variability and 100% success, clearly outperforming the baseline.

This comparison highlights the effectiveness of evolutionary optimisation and provides a reference point for interpreting results in later sections.

5.2 Statistical Significance: GA vs Random

To formally assess whether the GA significantly outperforms the random baseline, we apply two statistical tests:

- Mann-Whitney U test for comparing reward distributions
- Chi-squared test for comparing success rates (reward ≥ 200)

These non-parametric tests are suitable for comparing independent samples without assuming normality. A p-value below 0.05 indicates statistical significance.

Statistical Test Configuration: 60 episodes per agent (20 episodes × 3 seeds)

Metric	GA Value	Random Value	Test	Test Statistic	p-value	Significant (p < 0.05)
Mean Reward	293.22	-207.06	Mann-Whitney U	3600.0	0.0000	Yes
Success Rate (≥ 200)	100.0%	0.0%	Chi-squared	116.03	0.0000	Yes

Table 6 - Statistical Comparison Between Genetic Algorithm and Random Policy

Both tests confirm that the GA significantly outperforms the random agent, with p-values well below 0.05. The reward difference is statistically significant, and the success rate comparison yields a strong chi-squared statistic.

This validates the GA's superiority over the baseline, not only in raw performance but also in statistical reliability.

6. Agent Behaviour Analysis

To better understand how the GA-evolved policies behave, we collect per-step behavioural data across all evaluation episodes. This includes the frequency of each action taken and the trajectories followed by the agent in state space.

For each seed, the best genome is executed over 20 deterministic episodes, and we record:

- The action selected at every timestep,
- The agent's x and y positions throughout each episode,
- A subset of full trajectories for qualitative inspection.

This dataset forms the basis for analysing behavioural consistency, action preferences, and trajectory patterns.

```
GA: 11,271 total actions collected across 60 episodes

Behavior data collection complete.
```

The behaviour data collection confirms that the GA policies are stable across seeds, producing consistent action patterns and smooth trajectories. The large number of recorded actions provides a reliable foundation for subsequent visualisations, such as action-frequency plots and trajectory overlays.

These results allow us to move beyond reward-based evaluation and examine how the agent actually behaves in the environment.

6.1 Action Distribution Analysis

To understand how the GA-evolved policy behaves at the action level, we analyse the frequency of each action taken across all evaluation episodes. This provides insight into the agent's control strategy and whether it relies heavily on specific thrusters or maintains a balanced action profile.

We report both absolute action counts and percentage distribution, allowing us to assess behavioural tendencies independently of episode length.

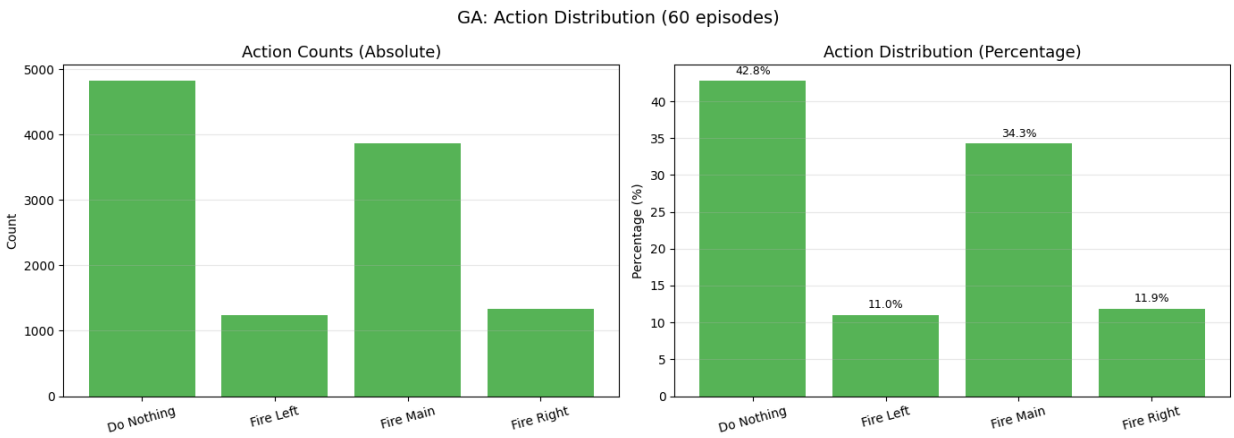


Figure 14 - Action-frequency distribution (absolute and percentage) for the GA policy.

The GA policy shows a clear preference for the "Do Nothing" and "Fire Main" actions, which together account for the majority of decisions. Lateral thruster usage ("Fire Left" and "Fire Right") appears balanced and significantly less frequent, suggesting that the agent stabilises the lander primarily through vertical control while applying horizontal corrections only when necessary.

This distribution reflects a coherent and efficient landing strategy, consistent with the high evaluation performance observed earlier.

6.2 Trajectory Visualization

To qualitatively assess the agent's behaviour, we visualise the x-y trajectories of the lander across three deterministic episodes. These plots show how the agent navigates the environment, including its initial position, descent path, and final landing location.

The red triangle marks the landing pad at the origin, while markers indicate the start and end of each trajectory.

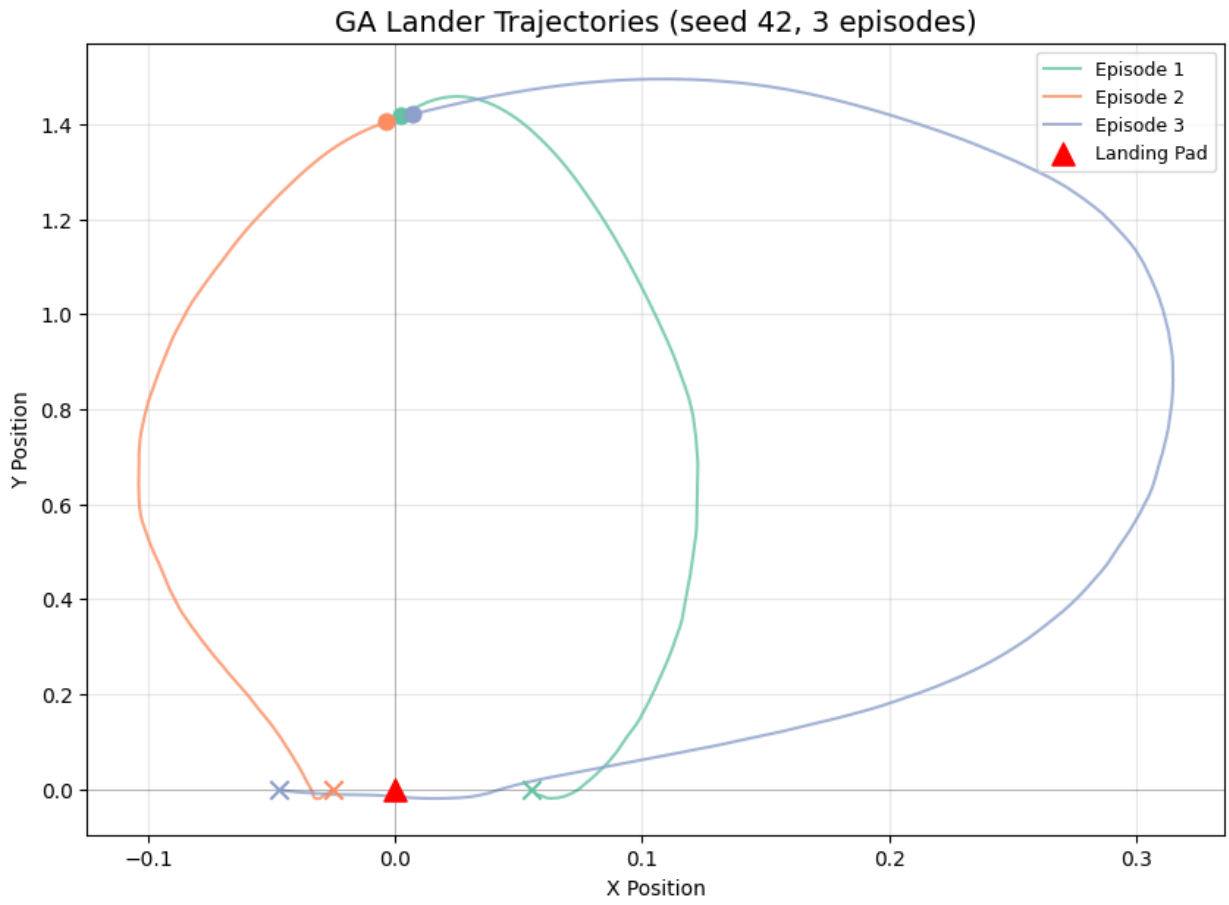
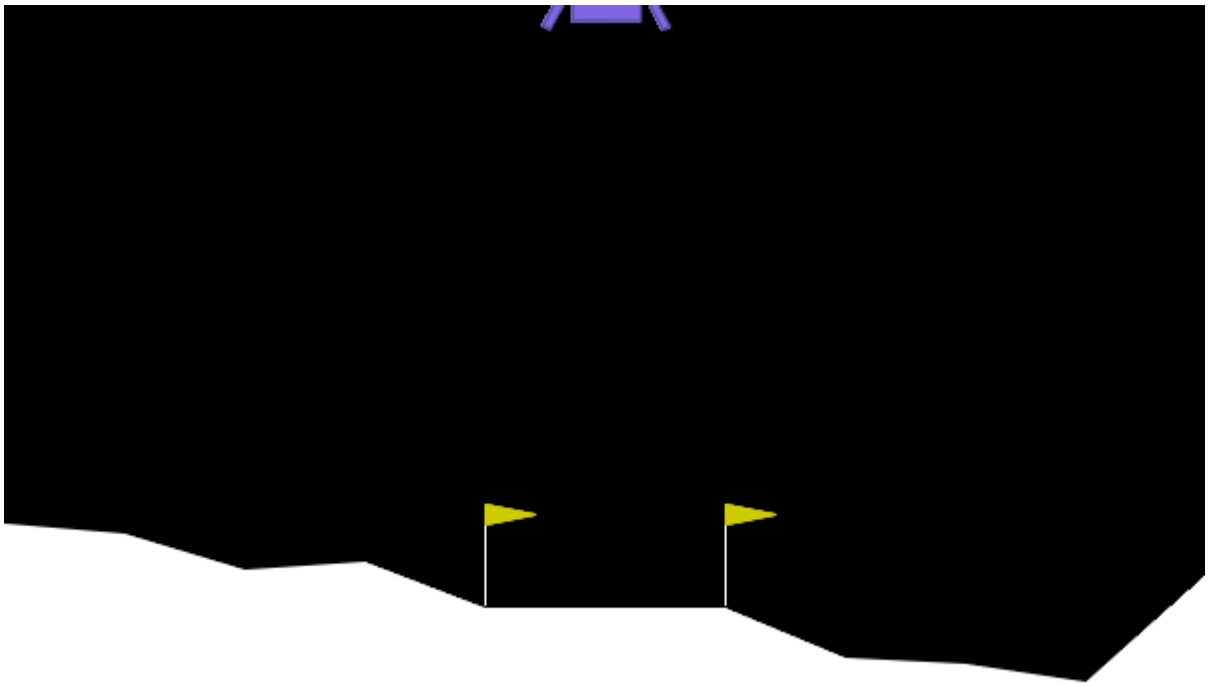


Figure 15 - Lander trajectories across three episodes using the GA policy.

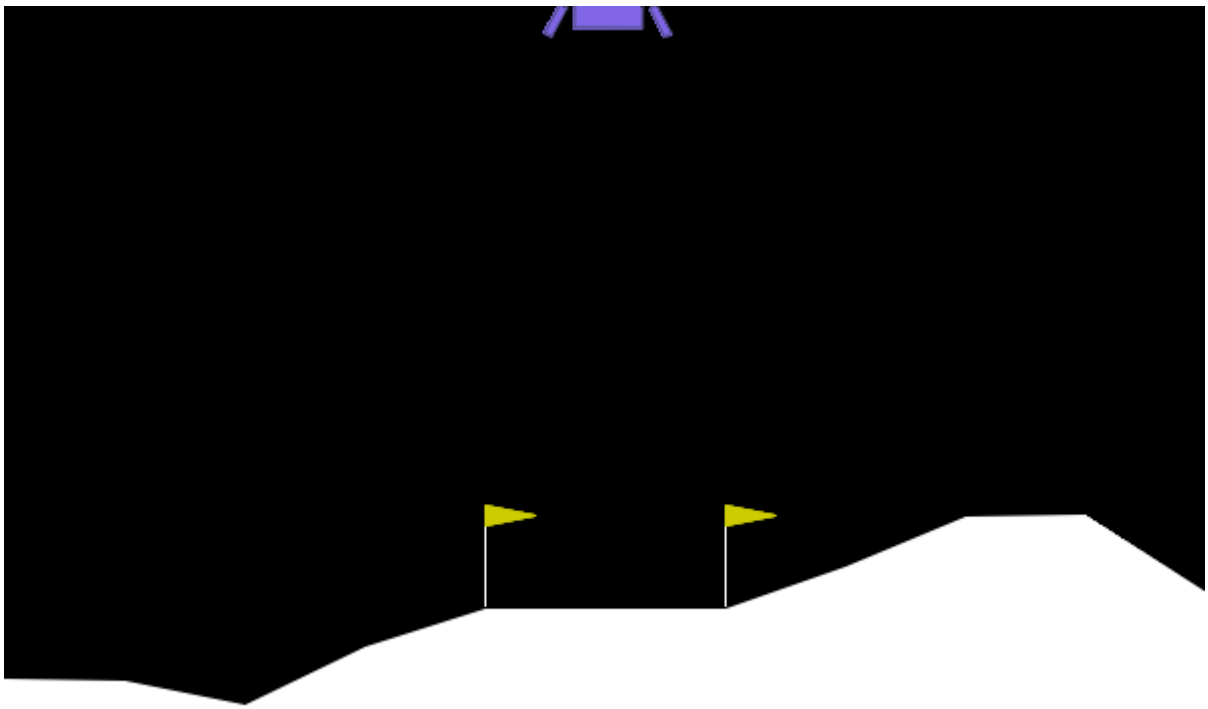
The trajectories show smooth and controlled descent paths, with consistent convergence toward the landing pad. The agent avoids erratic movements and maintains stable altitude and horizontal positioning throughout each episode.

This confirms that the GA-evolved policy produces coherent and repeatable behaviour, aligned with the task objective.

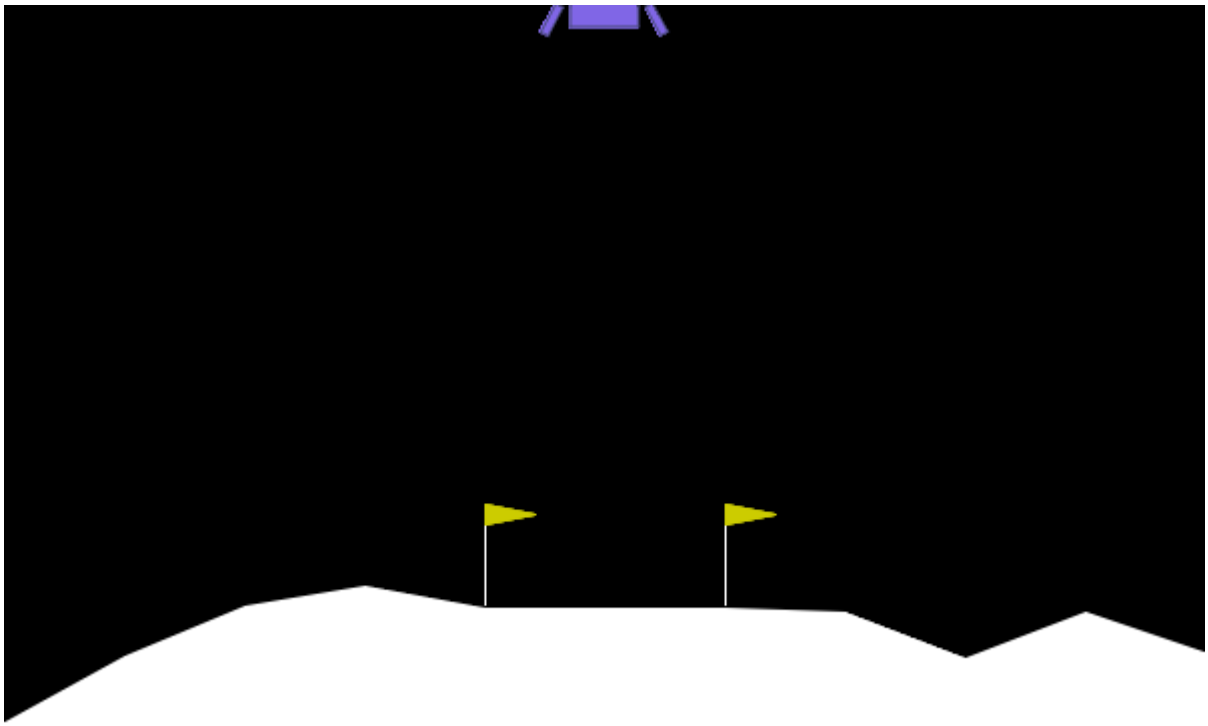
7. Lunar Lander Visualizations



Gif 1 – GA Landind (Seed 42)



Gif 2 – GA Landind (Seed 123)



Gif 3 – GA Landing (Seed 3407)

Parameter	Value
input_size	8
hidden1_size	10
hidden2_size	10
output_size	4
population_size	50
mutation_rate	0.05
generations	5000
eval_seeds_per_generation	3
elitism	3
parent_selection	top 20%
crossover	uniform gene-wise
mutation	Gaussian (clipped ±0.1)
activation	tanh (hidden), linear (output)
max_workers	20

Table 7 — Genetic Algorithm Hyperparameters

Appendix 1 - Experimental Setup

```
Python: 3.12.3
Gymnasium: 1.2.3
NumPy: 2.4.2
Network: 8 -> 10 -> 10 -> 4
Activation: tanh (hidden), linear (output)
```

Property	Value
Environment	LunarLander-v3 (Gymnasium)
Observation Space	Box(8,) — continuous 8-dimensional vector
Action Space	Discrete(4) — do nothing, fire left, fire main, fire right
Solved Threshold	Mean reward >= 200 over 100 consecutive episodes
Wind	Disabled (enable_wind=False)

Table 8 - Environment Details

- Observation vector: [x, y, vx, vy, angle, angular_velocity, left_leg_contact, right_leg_contact]
- Reward structure:
  - Moving toward the landing pad: positive
  - Moving away: negative
  - Crash: -100
  - Successful landing: +100
  - Each leg ground contact: +10
  - Firing main engine: -0.3 per frame
  - Firing side engine: -0.03 per frame
- Termination rules:
  - Terminated (success): The lander comes to rest on the ground with both legs in contact, near-zero velocity
  - Terminated (crash): The lander body contacts the ground, or the lander moves outside the viewport boundaries
  - Truncated (timeout): The episode exceeds 1000 timesteps without termination