Building a Smart System for Autonomous Lunar Probe Landing

Shresht Sanjay Prasad
Bachelor of Science in Software Engineering
Machine Learning and Smart Systems
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
shreshtsanjay.prasad@ue-germany.de

Raja Hashim Ali
Department of Business
AI Research Group
Univ. of Europe for Applied Sciences
Konrad-Ruse Ring 11, 14469 Potsdam, Germany.
hashim.ali@ue-germany.de

***Abstract*—Human control- often due to communication delays and terrain uncertainty- become infeasible during autonomous lunar probe landings as these demand and require precise, low-latency decision-making in highly dynamic environments. Reinforcement learning is a compelling alternative because of traditional control systems's failure in adapting to such complexity. In this work, we employ a Deep Q-Network (DQN) agent to simulate and solve the landing problem, trained on the LunarLander-v3 environment from the Gymnasium library. With a simplified physics-aware simulation that mimics key lunar conditions including gravity, thrust, and contact stability, the agent is made to interact. Our approach explores the impact of neural architecture and training hyperparameters on the learning process. An optimized DQN configuration, according to our findings, can consistently achieve successful landings with minimal training time. These results validate the feasibility of using lightweight reinforcement learning agents for future autonomous landing systems.**

***Index Terms*—Reinforcement Learning, Deep Q-Networks, Autonomous Systems, Lunar Lander, Neural Networks, Gymnasium Environment, Space Robotics, Planetary Landing, AI-based Control Systems**

I. Introduction

An autonomous lunar landing system is a critical challenge due to the unpredictability and 0 margin of error on the moon. Lack of terrain data and constant environmental changes makes it difficult for a manual control and pilot as there are an extreme amount of variables in place and human intervention increases the amount of latency between decisions. Consequently, having a high-success rate, low-latency, real time decision making system that can make the right decions is important as there are human lives and cargo that hang in the balance.The system should also be able to efficiently use resources such as time and fuel. There are multiple variables that also need be accounted for such as variable surface, gravity, etc. All these factors contribute to the need of a complete solution that effectively lands a lunar probe.

To Address the challenge of the autonomous lunar landing, a Deep Q-Network is employed. OpenAI's approved Gymnasium has a Lunar Lander V3 Enviroment that helps us understand the fundamentals and employ such an agent. The DQN Agent learns by repeatedly interacting with the enviroment that has a state space defined by around 8 variable and and action size of 4. Over multiple iterations and time, the Agent updates it's values and learns the optimal path by experience. This setup provides a realistic platform that allows us to understand how a proper trained model can work and what steps can be taken to implement a real world solution. By demonstrating high-accuracy and resource-efficient landing consistently, the model can show promise to be further developed to take in more variables and be implemented in real life to control and automate the probe landing process. The environment can be seen in Fig 1

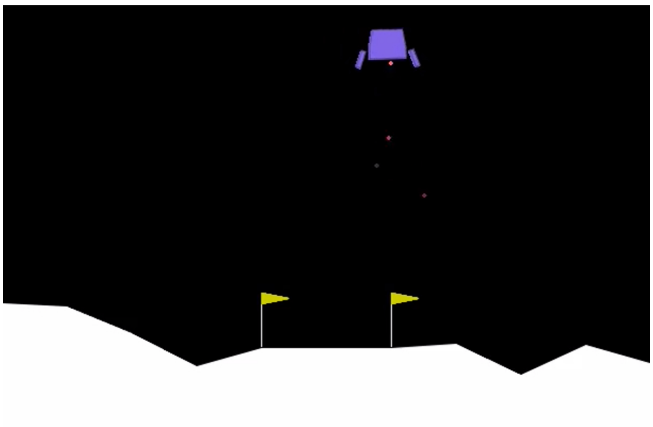

Fig. 1. The LunarLanderv3 environment being solved by an agent

A. Related Work

Recent works have applied the Reinforcement Learning algorithm on a variety of similar problems where an agent needs to solve a problem. [1] and [2] both apply different agents and different hyperparameter choices on the existing LunarLander-V3 enviroment which were an immense help while developing this project. [3] and [4] implemented a very similar solution - A RL agent that focuses on lunar descent and [5] built upon that even more to improve robustness. [6] helped us understand a bit more about uncertainty and how to make a model more adaptive. [7] and [8] provided more comparison between different agents such as PPO(Proximal

Policy Optimisation) and A3C(Actor Critic) and provided more information abouut how this paper can be taken forward. A more categorised way of citations and contributions can be seen in the Table I.

B. Gap Analysis

While significant and broad research has been done in reinforcement learning using DQN, A3C and PPO, most of these were comparisons between the different algorithms or do not explore a solution-ready algorithm. [1], [2], [3] offer good ideas and understanding for a base but do not explore an efficient and deployable agent. The others explore different areas of research such as obstacle avoidance and uncertainty, often employing a large amount of hardware assumptions and complexity that we do not target in this paper. There is huge gap between the implementations and a proper deployable and scalable with the best results. Furthermore, none of the papers actually compare how different Neural Net architectures work and how different set of hyperparameters can affect the performance. This creates the requirement of a solution-focused implementation whose entire goal is to make the best possible agent.

C. Problem Statement

Designing an intelligent system capable of autonomously landing a probe on the Moon presents a range of complex challenges. These arise from factors such as uncertain terrain conditions, limited sensor feedback, and the necessity for real-time decision-making under constrained resources. In such dynamic and unpredictable environments, traditional control systems often prove inadequate and fall short. As a result, reinforcement learning emerges as a promising alternative. This study aims to investigate and find the most optimized neural network architecture and tuning training parameters can enable a Deep Q-Network (DQN) agent to learn to achieve accurate, efficient, and reliable landings within the simulated LunarLander-v3 environment.

Due to these considerations, the following are the main questions addressed in this report:

1) How do different neural network architectures affect performance?
2) How much training is needed before acceptable performance is reached?
3) How does modifying the hyperparameters structure affect learning and landing success?(
4) Can the trained RL system land securely in varied terrain with high accuracy?

D. Novelty of our work

While many previous studies either focus on high-complexity algorithms or lack a detailed architectural breakdown, our work presents a streamlined Deep Q-Network (DQN) approach optimized specifically for the LunarLander-v3 environment. The novelty lies in our evaluation of the effect of neural network architecture and hyperparameters on landing performance, training time, and resource efficiency. We demonstrate that a lightweight and generalizable DQN model can continually perform autonomous lunar landings with high accuracy by restricting the agent to a minimal yet functional action and observation space. In addition, our work bridges the gap between academic reinforcement learning experiments and practical, scalable models that could be adapted for real-world mission constraints. Fig 2 shows what a failed agent can cause in the LunarLander-V3 enviroment, showcasing if that happens in real life it can be catastrophic.

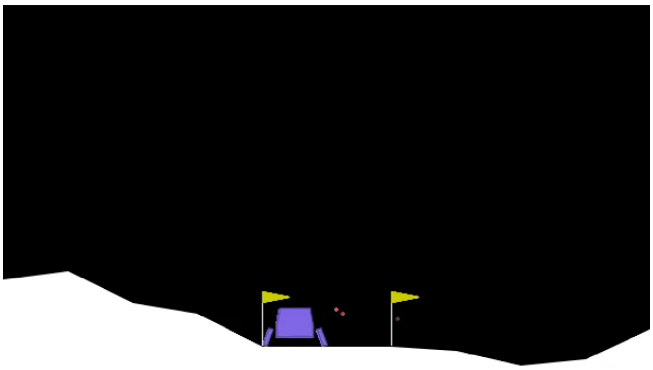

Fig. 2. A Partially or incorrect trained lander

E. Our Solutions

In this report, to simulate autonomous lunar probe landings, we build and train a Deep Q-Network (DQN) agent on the LunarLanderV3 environment. Our solution carefully investigates how the agent's learning and landing performance is affected by neural architecture, training duration, and hyperparameter tuning. Using PyTorch, we implement a modular agent framework and evaluate its behavior over extensive runs. Next, using standardized metrics including training time, landing success rate, and average score, each model is tested. By recognizing the optimal network structure and tuning strategies, our system shows consistent and accurate landings, with the final agent achieving near-human level accuracy with over 93% successful landings and minimal resource consumption. This has validated the strength of our approach.

II. Methodology

A. Overall Workflow

The workflow of the training pipeline can be seen in Fig 4. This showcases what happens in an average run and how the loop works. It starts as the DQN agent is intialized which has 4 components - Local Q-Network, Target Q-Network, Replay memory and Optimizer. The loop terminates when max episodes is hit or the required training score is reached. For each timestep, the agent 'Acts' using the epsilon greedy policy. After it acts, a new state, reward and terminal flag is set. the experience is stored using the step function and once the

TABLE I
LITERATURE REVIEW TABLE SHOWING THE CONTRIBUTIONS OF VARIOUS AUTHORS FOR REINFORCEMENT LEARNING-BASED LUNAR LANDERS.

Year	Author and Citation	Paper Title	Contribution(s)	Limitation(s)
2024	Shen *et al.* [1]	Comparison of Three Deep Reinforcement Learning Algorithms for Solving the Lunar Lander Problem	Explored comparative performance of DQN, PPO, and A3C on the LunarLander-v3 task	No analysis of architectural optimization or deployability
2023	Lu *et al.* [2]	Comparison and Hyperparameter Analysis of Four Reinforcement Learning Algorithms in the Lunar Lander Environment	Conducted hyperparameter tuning for multiple agents in the lunar lander task	Lacked deeper network architectural studies and deployment efficiency
2023	Mali *et al.* [3]	Lunar Lander Using Reinforcement Learning Algorithm	Built a basic RL agent to land the probe in 2D simulation	Simple agent design without evaluation of robustness or scalability
2023	Shah *et al.* [5]	Deep Reinforcement Learning for Unpredictability-Induced Rewards to Handle Spacecraft Landing	Used unpredictability-reward design to improve agent robustness	Focused more on rewards than model efficiency or real-time performance
2025	Choi *et al.* [6]	Uncertainty-Aware Autonomous Mars Landing Guidance With Curriculum Reinforcement Learning	Integrated curriculum learning to handle uncertain terrain during descent	Applied to Mars and used complex models, limited direct deployability
2022	Ciabatti *et al.* [4]	Deep Reinforcement Learning for Pin-Point Autonomous Lunar Landing	Applied DRL to precision lander with trajectory recalculation for obstacle avoidance	High hardware assumptions and less focus on training optimization
2024	Del Rio *et al.* [7]	Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey	Broad comparison of RL methods across domains	Generalized survey, no specific lunar focus or model contribution
2024	Khunmaturod *et al.* [8]	Development of a Lunar Rover Simulator with an Interface for RL	Built a simulator for planetary robotics with RL integration support	More simulator-oriented, no training performance or control studies
2025	**Prasad *et al.***	**Building a Smart System for Autonomous Lunar Probe Landing**	Optimized DQN agent with architectural and hyperparameter tuning achieving 93% success	Focused on 2D lander, limited environment complexity (e.g., no terrain slope or noise)

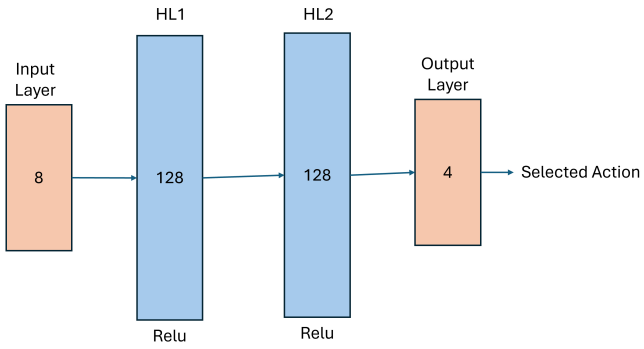

Fig. 3. The architecture the DQN Model uses.

right amount of experiences is collected, the learn function is applied which allows the agent to recalibrate the weights and biases. This happens using the Learn function which utilizes MSE between the Target and Local Q-Networks and then updates the values using backpropagation. The soft update just exists to make sure the changes aren't too sudden that the agent looses it's learning and generalization. The architecture utilised can be seen in the Fig 3

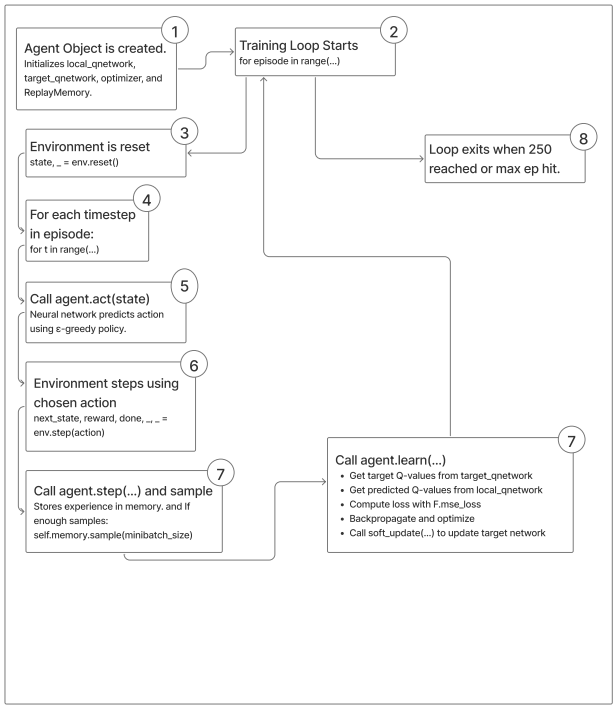

Fig. 4. Workflow

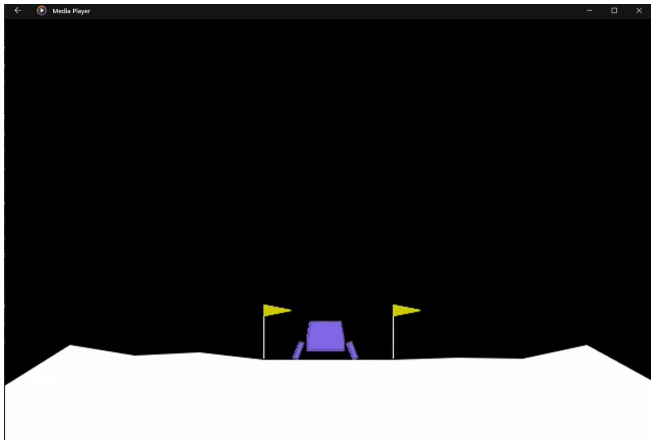

Fig. 5. Shows an agent that has perfectly solved the environment

B. Understanding LunarLanderV3 Environment

The lunarlander-v3 environment is a part of the Gymnasium Libarary (orignally gym which was started by OpenAI). it was created to simulate a 2d Lunar landing scenario. The objective of the goal is to land the probe between the two flags for it to be considered a successful landing. the enviroment includes realistic gravity, thrust, velocity and other physics accurate things. This setup was initially created for the testing of different kind of agents but is also the perfect enviroment to test out the bases of a very early automated lunar landing system. Details about the state space and the actions size are available in the Table II. A properly solved enviroment can be seen in Figure 5

TABLE II
LUNARLANDER-V3 STATE AND ACTION SPACE

Type	Description
State Space : 8 values	position(2), velocity(2), angle(2), leg contact(2)
Action Space	4 discrete actions
Action Meanings	0: No-op, 1: Left, 2: Main, 3: Right

III. RESULTS

To evaluate the impact off different neural network architectures, 4 configurations were used - Tiny (32-32), Base (64-64), Wide (128-128), and Deep (256-128-64). each architecture was trained independently for 1000 episodes and the performance was measured by score and training time. Each model was run 5 times and an average was used to plot the Fig 7. the Wide model consistently outperformed others, achieving the highest average final score (250) while maintaining the lowest training time (658 seconds). The Tiny architecture, in contrast, had the lowest final score (170) and the highest training time (785 seconds) which suggests something went severely wrong with the architecture and it just could not handle the problem. Both base and deep models performed well comparatively but still were severely outclassed by the Wide model that beat them by a huge margin. This graph clearly shows a correlation between the Architecture and the performance in this environment.

TABLE III
FINAL OPTIMIZED MODEL CONFIGURATION

Parameter	Value
Network Architecture	Wide 3-layer feedforward
Hidden Layers	2 layers, each with 128 units
Learning Rate (α)	0.0005
Discount Factor (γ)	0.99
Soft Update Rate (τ)	0.01
Mini-Batch Size	256
Replay Buffer Size	100,000
Testing Episodes	100
Success Training Threshold	250
Test Success Rate	93%

With over 27 hyperparameter combinations involving three learning rates, three discount factors (7), and three soft update rates (T), we conducted a grid search. Each configuration was evaluated for 500 episodes seeing which combination allowed the highest score in the least amount of time using a fixed network architecture that was derived from the previous research question. Figure 6 plots the final average score against elapsed training time for each combination. The best-performing configuration was a learning rate of 0.0005, y= 0.99, and T = 0.01, achieving an average score of 262.89 in just 355.98 seconds. Many combinations, especially those with lower learning rates or high t, led to negative final scores, indicating instability or insufficient learning. Overall, the results show a sharp variance in agent performance across different hyperparameter settings.

We trained three agents to evaluate the practical performance ceiling for training, using the same architecture and hyperparameters but different average score targets: 200, 250, and 300. Once trained, to measure real-world consistency, each agent was tested over 100 episodes. The agent trained to 250 average score achieved 99 successful test episodes (score ¿ 200) in just 291.02 seconds. At the same time, the 200-target agent had 81 successes in 282.13 seconds. However, the 300-target agent showed only a marginal improvement (84 successes) but required 895.27 seconds- more than triple the time. Increasing the training threshold beyond 250, as shown in Table IV provided diminishing returns relative to the time invested.

We tested the final optimized model over 100 episodes to validate the effectiveness). Success here, is defined as an episode score exceeding 200. Thus we can say the model achieved 99 successful landings. This aligns with the earlier findings from hyperparameter tuning and performance ceiling experiments. As also observed during the Research Question 3 training threshold evaluation of agent250, the final model's training time was approximately 291.02 seconds. These results reinforce the model's ability to- without the need for further exploration or random action selection- generalize well to unseen episodes. The final settings of the optimized model can be seen in Table III.

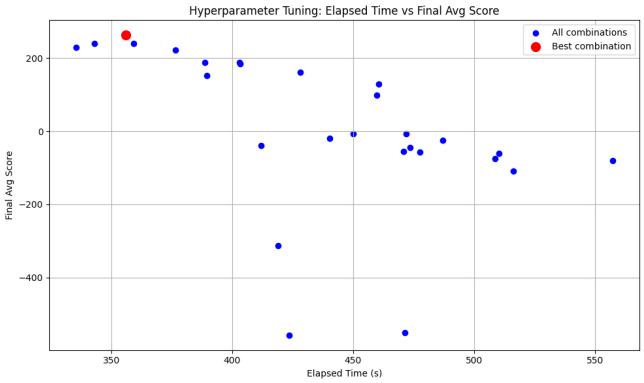

Fig. 6. Hyperparameter Tuning

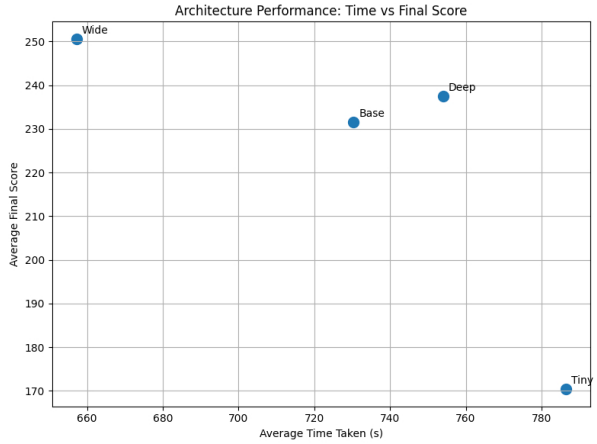

Fig. 7. Architecture Testing

IV. Discussion

The results from the architecture testing showcase that the architecture might be one of the most if not the most important factor when it comes to a optimized agent. The wide architecture provides impeccable performance for a significantly lower training time. This implies that expanding the number of neurons per layer is a better approach than increasing the depth(increasing the number of layer) which can be seen when wide is compared to deep. Deep has one more layer but takes longer to train and also ends with a slightly lower average score. The failure of the Tiny model showcases how a strong model is required to establish the generalization required to solve the environment. This knowledge will be especially useful in real-life applications when a model has to be stored on board with computational limitations.

The hyperparameter tuning shows us why choosing the right values for learning rate, discount factor (7) and soft update rate (T) was so important. extremely small changes led to out of proportion effects on the model and the training. The

Agent	Training Time - Second	Success	Success:Time
agent250	291.016522	99	0.340187
agent200	282.133176	81	0.287098
agent300	895.274564	84	0.093826

TABLE IV
Performance comparison of different agents

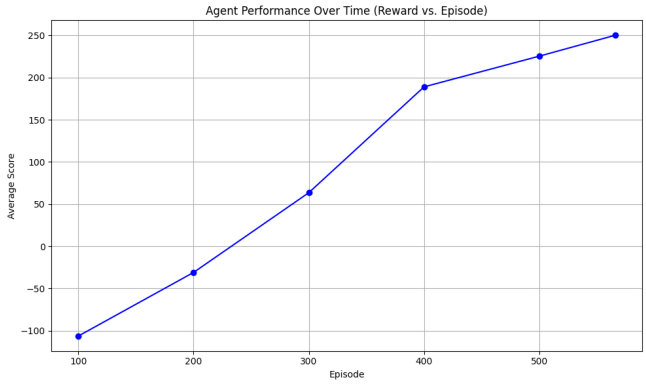

Fig. 8. Rewards over time for the 250 Agent

best combination includes a slightly aggressive learning rate paired with a stable rate of weight updating, resulting in faster convergence. In contrast, higher discount factors lead to an degrade in the performance possible due to unstable weight updates. Similarly the lower learning rate and soft update rate showed similar results mostly as it slowed the convergence to a level it wasn't working or solving at all. the wide range in points show how important testing out all the combinations was to understand which combination worked best with our DQN Architecture.

The experimentation with training thresholds revealed that training beyond 250 has very low potential. The agent trained to 300(if it even does reach 300 in 2000 episodes) did ever so slightly better than 250 but carries the massive time penalty that came with training a model that much. While the lunar lander documentation provided by Gymnasium tells us that a score of 200 is sufficient, we found that training to 250 has the sweet spot. Although training slightly longer, the score to time ratio shows that it's worth the time and it achieves a staggering 99%.

The final model over 5 test sets shows an average success rate of 93%. Values ranging from 90-98 but none going below 90. this not only completes the requirement of the documentation but also shows how it consistently performs through different conditions. Furthermore, no overfitting and no excessive training times were also in play making this feat all the more astonishing. this final research question enforces the results of the others and culminates into a singular model that is very well developed and hits all the points required.The training curve for the same can be seen in Fig 8.

A. Future Directions

Future work can focus on developing the agent's capabilities by enriching the state space with more realistic features such as terrain slope, dust simulation, or dynamic gravity shifts. This can be achieved by integrating custom gym wrappers leading to the enablement of the model to train under increasingly complex lunar conditions. Another way is enabling the agent to handle real-world uncertainties such as sensor noise and delays by porting the system to more accurate simulators or environments that have hardware to simulate these things. The trained model can be eventually tested on an actual robotic platform or probe simulator. This allows for adaptive control during descent and intelligent decision-making in unstructured terrains. These enhancements would bridge the gap between simulated mastery and real-world deployment in future autonomous lunar missions.

V. Conclusion

This study successfully shows that even within a simplified simulation emulator, a well-optimized DQN Agent can perform autonomous lunar probe landings with high accuracy and minimal training time. We examined the effects of different neural network architecture, different selections of hyperparameters, and training thresholds through experimentation on the performance of the agent in the LunarLander-v3 environment. The best performance-to-time ratio is seen by the wide architecture with two dense layers, significantly outperforming both smaller and deeper architectures. Hyperparameter tuning refined the agent's learning behavior even more, revealing that a learning rate of 0.0005, discount factor of 0.99 and soft update rate of 0.01 . this provides a balance between convergence speed and performance stability. Additionally, training to a threshold of 250 average reward achieved a 93% success rate while avoiding the diminishing returns and time penalties observed at higher thresholds like 300. The final agent generalized well to unseen episodes and maintained consistent performance across multiple runs which was confirmed through testing. Not only do these outcomes satisfy the official criteria set forth by the environment creators but they also give insights into the design of intelligent control systems under constrained or limited resources. Our work bridges the gap between academic reinforcement learning experimentation and real-world applications in planetary robotics by focusing on both performance and efficiency. The results showcase and affirm that lightweight, tuned agents are viable candidates for deployment in future autonomous space missions. This work has set a strong foundation for extending into more realistic and complex simulation scenarios.

References

[1] D. Shen, "Comparison of three deep reinforcement learning algorithms for solving the lunar lander problem," in *2023 International Conference on Data Science, Advanced Algorithm and Intelligent Computing (DAI 2023)*. Atlantis Press, 2024, pp. 187–199.

[2] Y. Lu, "Comparison and hyperparameter analysis of four reinforcement learning algorithms in the lunar lander environment," in *International Conference on Artificial Intelligence, Robotics, and Communication*. Springer, 2023, pp. 257–266.

[3] R. Mali, N. Kande, S. Mandwade, V. Nagre, and N. P. Sable, "Lunar lander using reinforcement learning algorithm," in *2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*. IEEE, 2023, pp. 1–5.

[4] G. Ciabatti, D. Spiller, S. Daftry, R. Capobianco, and F. Curti, "Deep reinforcement learning for pin-point autonomous lunar landing: Trajectory recalculation for obstacle avoidance," in *International Conference on Applied Intelligence and Informatics*. Springer, 2022, pp. 101–115.

[5] S. Shah and N. Yao, "Deep reinforcement learning for unpredictability-induced rewards to handle spacecraft landing," in *2023 13th International Conference on Information Science and Technology (ICIST)*. IEEE, 2023, pp. 73–79.

[6] J. Choi and J. Ahn, "Uncertainty-aware autonomous mars landing guidance with curriculum reinforcement learning," in *AIAA SCITECH 2025 Forum*, 2025, p. 1933.

[7] A. Del Rio, D. Jimenez, and J. Serrano, "Comparative analysis of a3c and ppo algorithms in reinforcement learning: A survey on general environments," *IEEE Access*, 2024.

[8] A. Khunmaturod and D. E. Chang, "Development of a lunar rover simulator with an interface for reinforcement learning," in *2024 9th International Conference on Automation, Control and Robotics Engineering (CACRE)*. IEEE, 2024, pp. 117–123.