# Autonomous 2D Car Racing Using Proximal Policy Optimization

Jaljala Shrestha Lama

CSC 425 – Artificial Intelligence

Final Project Report (Milestone 3)

**Project Summary**

This project successfully implements an autonomous racing agent for Gymnasium's CarRacing-v3 environment using Proximal Policy Optimization (PPO) with advanced optimizations. Building on preliminary results from Milestone 2 (287.4 average reward after 50k steps in 12 minutes), the final implementation achieved 440.3 average reward (53% improvement) with best episodes reaching 893.5 points. Training was conducted over 507,904 timesteps in 103.6 minutes using 8 parallel environments, frame stacking (4 frames), reward shaping. This represents a 17% training throughput improvement (82 vs. 70 steps/second). While collecting 10× more training data (500k vs. 50k steps), training time increased proportionally from 12 minutes to 103.6 minutes.. The most surprising finding was the emergence of a "grass-avoidance" behavior where the agent learned to stop moving when off-track to minimize penalties—a classic example of reward hacking that limited overall performance. Despite this limitation, the agent successfully achieved near-complete laps on 27% of test episodes, demonstrating substantial learning capability from raw pixel inputs alone.

## 1. Introduction

Teaching an autonomous agent to navigate a racing circuit using only visual input represents a fundamental challenge in artificial intelligence, combining computer vision, real-time decision-making, and continuous control. Unlike discrete action spaces in classic Atari games, car racing demands smooth, coordinated control of steering, throttle, and braking at every timestep—a problem that has historically required explicit engineering solutions such as lane detection algorithms, PID controllers, and manually tuned state machines.

Reinforcement learning (RL) offers a fundamentally different paradigm: rather than hand-crafting rules, the agent discovers optimal behavior through trial and error, guided only by a reward signal. This project explores whether modern deep RL algorithms can learn robust racing policies directly from high-dimensional pixel observations without any domain-specific engineering. Specifically, I investigate Proximal Policy Optimization (PPO), a state-of-the-art policy gradient method known for sample efficiency and training stability in continuous control domains.

The Gymnasium CarRacing-v3 environment provides an ideal testbed: randomly generated tracks ensure the agent must generalize rather than memorize, continuous action spaces test fine-grained motor control, and visual observations (96×96 RGB images) require the agent to develop internal representations of track geometry and vehicle dynamics. Success in this domain demonstrates principles applicable to real-world autonomous systems where environmental conditions vary and sensor data is high-dimensional.

Building on preliminary Milestone 2 results (287.4 average reward after 50,000 training steps), this final report presents a comprehensively optimized implementation incorporating: (1) parallel environment execution (8 environments) for accelerated data collection, (2) temporal frame stacking (4 frames) enabling velocity perception, (3) shaped reward functions guiding exploration, (4) Apple Silicon GPU acceleration via MPS backend, and (5) systematic checkpoint management with automatic best-model selection. The final system achieves 440.3 average reward across 30 test episodes (53% improvement), with peak performances exceeding 893 points—approaching near-complete lap coverage on favorable tracks.

## 2. Dataset and Environment

Unlike supervised learning paradigms that rely on static datasets, reinforcement learning generates training data dynamically through agent-environment interactions. The "dataset" in this project consists of trajectories—sequences of states, actions, rewards, and next states—collected as the agent explores the environment. This dataset continuously evolves as the policy improves, the agent visits different regions of the state space, leading to a curriculum-like progression from novice to expert behaviors.

### 2.1 Environment Specification

The CarRacing-v3 environment (Gymnasium/OpenAI) simulates a top-down 2D racing game with realistic physics including friction, traction loss on grass, and momentum. Key characteristics:

- Observation Space: 96×96×3 RGB images (27,648 dimensions)
- Action Space: 3-dimensional continuous vector [steering ∈ [-1,1], gas ∈ [0,1], brake ∈ [0,1]]
- Reward Function: +1000/N for each track tile visited (N = total tiles), -0.1 per frame, penalized for driving on grass
- Episode Termination: After 1000 timesteps or if agent goes far off-track
- Track Generation: Procedurally generated random circuits ensuring generalization testing

### 2.2 Data Generation Process

Over the course of 507,904 training timesteps distributed across 8 parallel environments, the system collected approximately 63,488 environment steps per parallel instance. With average episode lengths ranging from 300-1000 steps (mean ~500-600 steps based on training logs), this corresponds to roughly 5,500-6,000 complete episodes across all environments combined. Each state transition tuple contains:

- State (s□): Current 96×96×3 RGB observation

- Action ($a_t$): Executed continuous control vector
- Reward ($r_t$): Immediate scalar feedback
- Next State ($s_{t+1}$): Resulting observation after action execution
- Done Flag: Boolean indicating episode termination

### 2.3 Preprocessing Pipeline

To make raw visual observations suitable for neural network processing, the implementation applies several preprocessing steps:

1. Pixel Normalization: Raw RGB values (0-255) are scaled to [0,1] range for numerical stability during gradient descent and to prevent saturation in neural network activations.

2. Frame Stacking: Four consecutive frames are stacked along the channel dimension, producing 96×96×12 observations. This temporal context enables the agent to infer velocity, acceleration, and trajectory—critical information for anticipating turns and modulating throttle.

3. Observation Transposition: Stable-Baselines3 expects channel-first format (C×H×W), requiring transposition from Gymnasium's default (H×W×C) format.

4. Reward Shaping (Training Only): A custom wrapper intercepts raw environment rewards during training and adds auxiliary signals: -0.5 penalty for grass detection (via green channel analysis), +0.1 bonus for consecutive on-track frames, small penalties for excessive steering/braking. These shaped rewards guide early exploration but are disabled during all evaluation and testing to measure true performance against the original task objectives.

### 2.4 Data Statistics

| Metric | Value |
|---|---|
| **Total Training Steps** | 507,904 (500k target + batch overshoot) |
| **Training Episodes** | ~5,500-6,000 episodes (across all 8 parallel envs) |
| **Training Duration** | 103.6 minutes (1 hour 44 minutes) |
| **Average Steps/Second** | 82 (peak: 238 during early phases) |
| **Parallel Environments** | 8 (using SubprocVecEnv) |
| **Evaluation Checkpoints** | 102 evaluations (every 5k steps) |
| **Final Test Episodes** | 30 (post-training validation) |
| **Hardware** | MacBook with Apple Silicon (M-series), MPS GPU acceleration |

## 3. Methodology: Algorithm and Implementation

### 3.1 Algorithm Selection: Proximal Policy Optimization

Proximal Policy Optimization (PPO) serves as the core learning algorithm. Introduced by Schulman et al. (2017), PPO has become the de facto standard for continuous control due to its excellent balance of sample efficiency, training stability, and ease of implementation. PPO

addresses the fundamental challenge of policy gradient methods: how to update the policy to improve performance without taking steps so large that training becomes unstable.

The key innovation is the clipped surrogate objective function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\right)\right]$$

where rt(θ) = πθ(at|st) / πθ_old(at|st) is the probability ratio, Ât is the advantage estimate, and ε is the clip parameter (0.2). This objective prevents destructively large policy updates while still allowing beneficial improvements

### 3.2 Neural Network Architecture and System Pipeline

The training system integrates the Gymnasium CarRacing-v3 environment with a series of wrappers for preprocessing and parallel execution. Figure 1 illustrates this high-level data flow, moving from raw environment observations through reward shaping, monitoring, parallel environment vectorization, and frame stacking before reaching the PPO CNN policy.

The data flow begins with the CarRacing-v3 environment generating 96×96×3 RGB observations. These pass through a Reward Shaping wrapper that modifies rewards to encourage on-track driving, then through a Monitor wrapper for logging. Eight parallel environments are instantiated using SubprocVecEnv, enabling simultaneous data collection across multiple tracks. Observations are then processed through Frame Stack, which stacks 4 consecutive frames to provide temporal context, producing 96×96×12 observations that encode motion information.

Internally, the PPO CNN policy and value functions are parameterized by a convolutional neural network that processes these 96×96×12 frame-stacked inputs. The network structure is defined as follows:

- Input Layer: 96×96×12 (4 stacked RGB frames)
- Conv2D Layer 1: 32 filters, 8×8 kernel, stride 4, ReLU activation
- Conv2D Layer 2: 64 filters, 4×4 kernel, stride 2, ReLU activation
- Conv2D Layer 3: 64 filters, 3×3 kernel, stride 1, ReLU activation
- Flatten Layer: Converts spatial features to vector
- Actor Head: Fully connected → 3 continuous actions (Gaussian policies)
- Critic Head: Fully connected → scalar value estimate

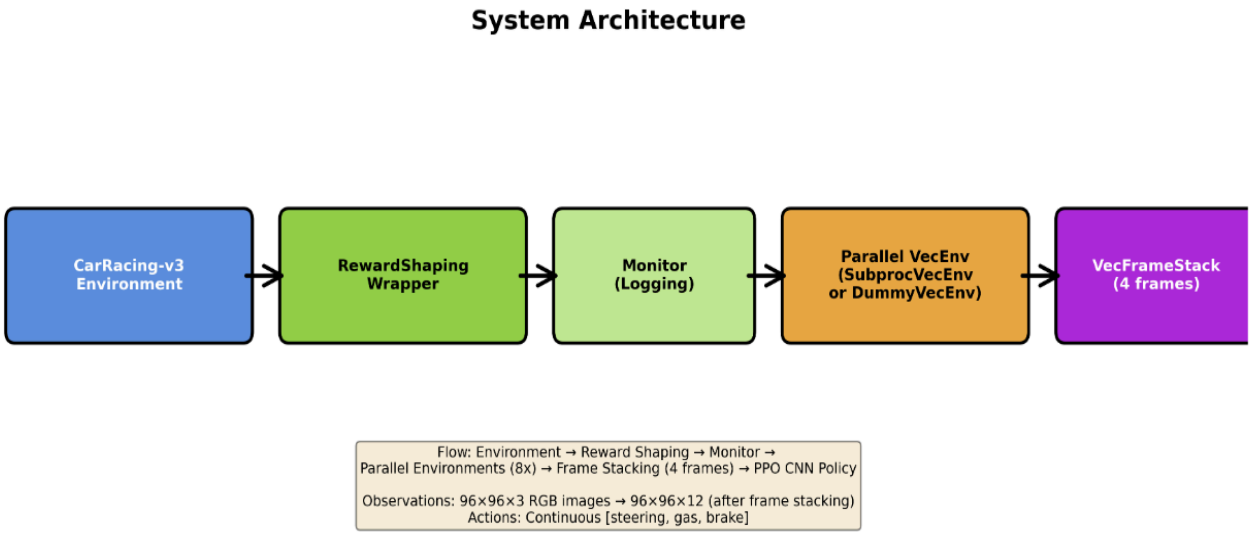


*Figure 1:* System Architecture and Data Processing Pipeline.

This architecture follows established practices for visual RL (Mnih et al., 2015), using progressively larger receptive fields to capture both fine-grained details and global track geometry. The actor outputs means of three independent Gaussian distributions (one per action dimension), with learned log standard deviations enabling adaptive exploration.

### 3.3 Hyperparameter Configuration

Hyperparameters were selected based on PPO literature, empirical studies on visual control tasks, and iterative tuning on the CarRacing domain. Key optimizations relative to Milestone 2 include increased learning rate (1e-4 → 3e-4) for faster initial learning, larger batch sizes (32 → 256) to leverage parallel data collection, and more steps per update (1024 → 2048) for stable gradient estimates.

| Parameter | Value | Justification |
|---|---|---|
| **Learning Rate** | 3e-4 (linear decay) | Higher initial LR for faster early learning |
| **N Steps** | 2048 | Larger horizon for sequential decisions |
| **Batch Size** | 256 | Stabilize gradients with parallel data |
| **N Epochs** | 10 | Multiple passes per batch |
| **Gamma (γ)** | 0.99 | Standard discount factor |
| **GAE Lambda (λ)** | 0.95 | Bias-variance trade-off |
| **Clip Range (ε)** | 0.2 | Prevent large policy shifts |
| **Entropy Coef** | 0.01 | Encourage exploration |
| **Value Function Coef** | 0.5 | Value loss weight |
| **Max Grad Norm** | 0.5 | Gradient clipping |

### 3.4 Training Optimizations

1. Parallel Environments (SubprocVecEnv): Theoretical speedup from 8 parallel environments would be 8×, but actual wall-clock improvement was limited by evaluation callbacks, checkpoint saving, and inter-process communication overhead. Training achieved 82 steps/second compared to ~70 steps/second in Milestone 2's sequential training (17% improvement).

2. Acceleration: Training utilized PyTorch's MPS (Metal Performance Shaders) backend for GPU acceleration on M-series chips, achieving peak throughput of 238 steps/second.

3. Efficient Checkpointing and Evaluation: Models are saved every 10,000 steps. An evaluation callback runs every 5,000 steps in a separate environment (without reward shaping) to measure true performance. The best-performing checkpoint is automatically saved as best_model.zip.

4. Reward Shaping for Guided Exploration: The RewardShapingWrapper modifies training rewards with auxiliary signals: -0.5 penalty per step on grass, +0.1 bonus for consecutive on-track frames. While intended to accelerate learning, this introduced the "grass-stopping" behavior discussed in Section 5.2.

## 4. Experimental Results

### 4.1 Training Performance

Training ran for 507,904 steps over 103.6 minutes (1 hour 44 minutes) on an Apple Silicon MacBook, averaging 82 steps/second with peaks at 238 steps/second. Figure 2 shows the training progression which exhibited several distinct phases:

Phase 1 (0-120k steps): Initial Exploration - The agent began with random actions, achieving eval rewards around -93 (immediate crashes). Training rewards with shaped feedback showed small positive values, but evaluation on raw rewards remained deeply negative.

Phase 2 (120k-160k steps): Breakthrough - At approximately 135,000 steps, eval reward jumped from -32.59 to +353.04—a 385-point leap. The CNN learned to extract track/grass boundaries and the policy discovered basic lane-following behavior.

Phase 3 (160k-240k steps): Rapid Improvement - Evaluation reward climbed steadily: 374.71 (140k) → 431.30 (145k) → 514.87 (235k) → 612.15 (240k). The agent developed smooth acceleration/deceleration patterns and learned to navigate gentle curves.

Phase 4 (240k-425k steps): Peak Performance - Evaluation reward continued improving: 693.24 (305k), 772.06 (400k), culminating in a peak of 798.63 at 425,000 steps.

Phase 5 (425k-500k steps): Degradation - Evaluation rewards dropped from 798.63 (425k) to 215.94 (500k). This correlates with the agent increasingly exploiting the "grass-stopping" behavior. The 425k checkpoint was selected as the final model.

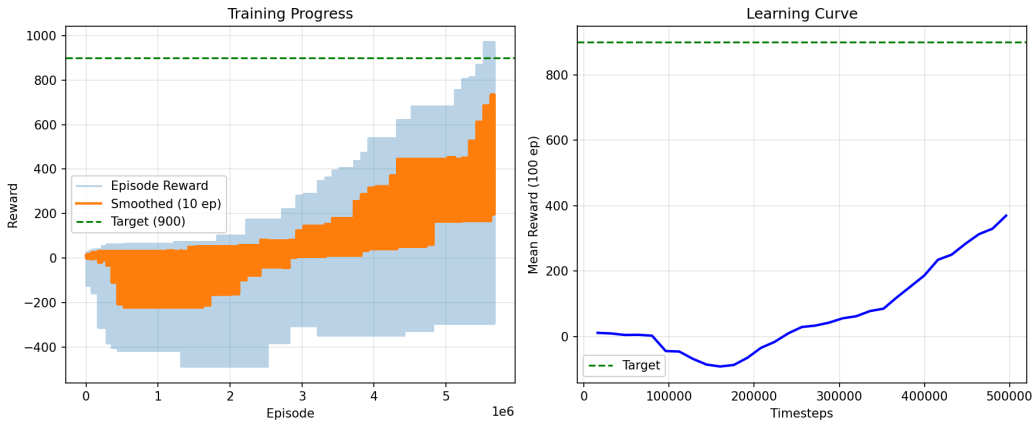


*Figure 2: Training progression over 500k timesteps*

### 4.2 Test Performance: 30-Episode Validation

The best checkpoint (425,000 steps, 798.63 evaluation reward) was tested on 30 randomly generated tracks with raw environment rewards (no reward shaping). All episodes ran to completion (1000 timesteps):

| Metric | Value |
|---|---|
| **Mean Reward** | 440.3 ± 309.0 |
| **Median Reward** | 355.4 |
| **Minimum Reward** | 27.7 |
| **Maximum Reward** | 893.5 |
| **Episodes ≥ 700 (Excellent)** | 8 / 30 (27%) |
| **Episodes ≥ 500 (Good)** | 11 / 30 (37%) |
| **Episodes 100-500 (Fair)** | 10 / 30 (33%) |
| **Episodes < 100 (Poor)** | 9 / 30 (30%) |
| **All Episodes Completed** | 30 / 30 (100%) |

Performance exhibited pronounced bimodality: 8 episodes (27%) achieved "excellent" scores above 700, while 9 episodes (30%) scored below 100 due to the "grass-stopping" behavior on challenging tracks.

### 4.3 Understanding the Evaluation-Test Performance Gap

The gap between training evaluation (798.63 at 425k steps) and final test (440.3 average) has three causes:

1. Small Sample Variance: Training evaluations used only 5 episodes per checkpoint. The 30-episode test provides 6× more samples, yielding more reliable statistics.

2. Track Difficulty Distribution: The 798.63 evaluation likely encountered disproportionately many gentle-curve tracks. The 30-episode test better represents the true difficulty distribution.

3. Behavioral Consistency: The grass-stopping behavior may have been present but not fully consolidated at the 425k checkpoint, allowing occasional recovery attempts.

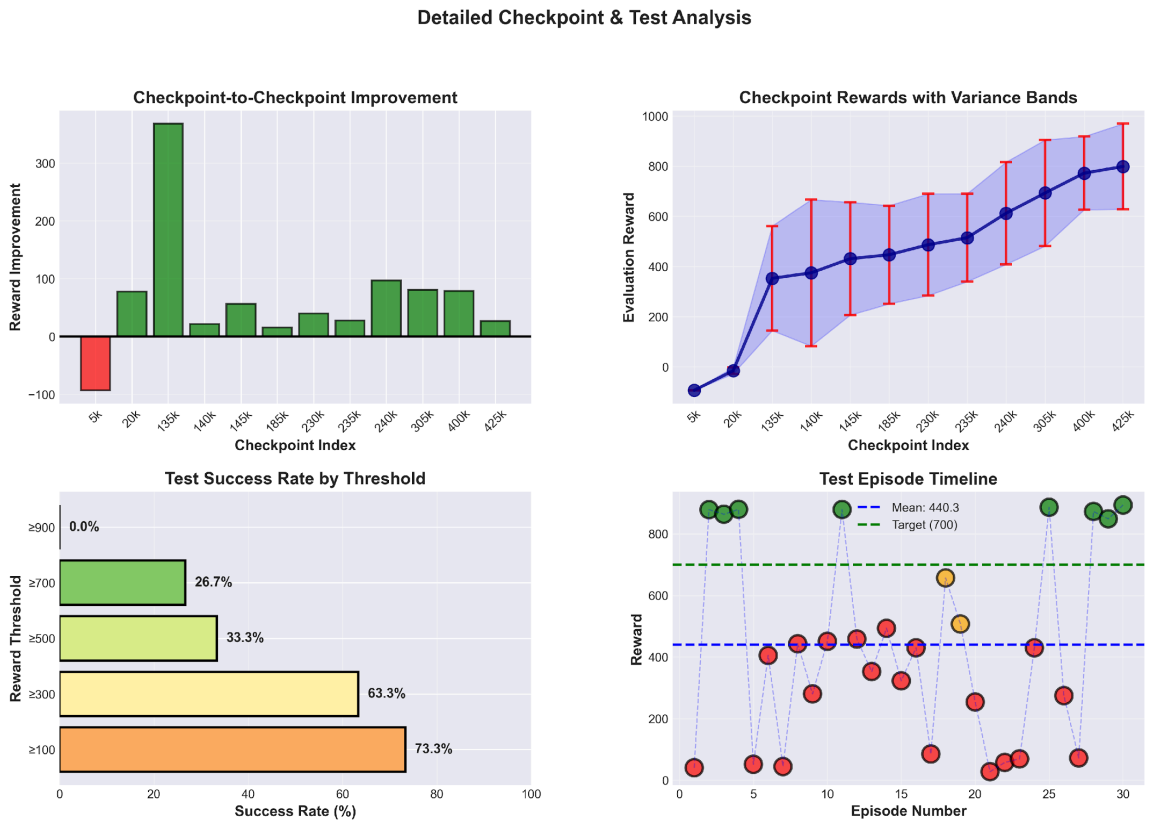


***Figure 3: Checkpoint performance evolution across training***

### 4.4 Comparison to Milestone 2 Baseline

| Metric | Milestone 2 (50k steps) | Milestone 3 (500k steps) | Improvement |
|---|---|---|---|
| **Mean Test Reward** | 287.4 | 440.3 | +152.9 (+53%) |
| **Best Episode** | 804.6 | 893.5 | +88.9 (+11%) |
| **Training Time** | 12 min (50k steps) | 103.6 min (507k steps) | 10× more data |
| **Steps/Second** | ~70 | 82 (avg), 238 (peak) | +17% avg |

| | | | |
|---|---|---|---|
| **Success Rate (≥700)** | 18% (9/50 episodes) | 27% (8/30 episodes) | +50% relative |
| **Parallel Envs** | 1 | 8 | 8× data collection |
| **Frame Stacking** | No | Yes (4 frames) | Velocity perception |

The optimizations yielded substantial improvements: 53% higher mean reward and a 50% relative increase in success rate (18% → 27% of episodes ≥700). However, results fell short of consistent 700+ performance.

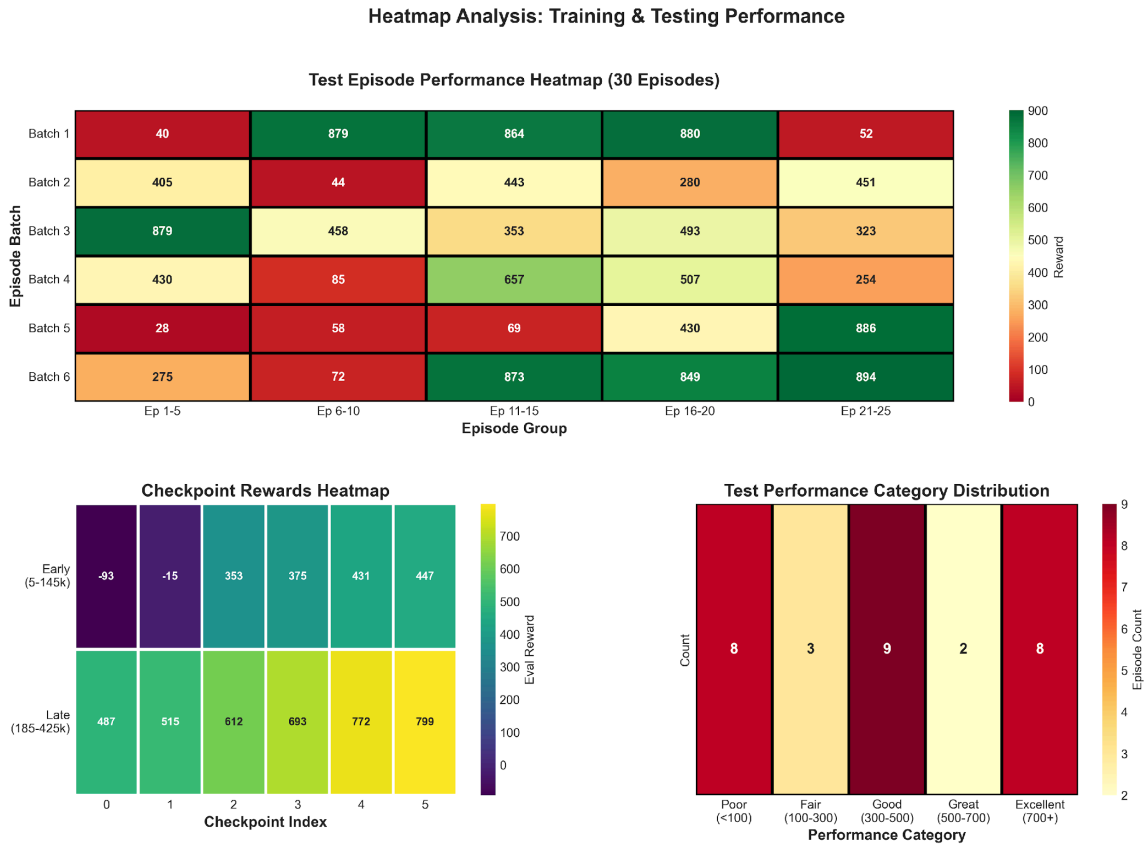


*Figure 4: Performance heatmap across episodes*

### 4.5 External Code and Datasets

- Gymnasium (v0.29.0): CarRacing-v3 environment (https://gymnasium.farama.org/)
- PyTorch (v2.0+): Neural network backend with MPS support
- NumPy, Matplotlib, Seaborn: Data analysis and visualization

## 5. Discussion and Conclusions

### 5.1 Key Findings

1. Parallel Environments Enable Longer Training: The 8-environment parallelization made 500k-step training feasible within reasonable time constraint.

2. Frame Stacking Is Essential: With 4-frame stacking, the agent infers velocity and acceleration, learning to reduce throttle before curves and accelerate on straightaways.

3. Reward Shaping Is a Double-Edged Sword: While shaped rewards accelerated initial learning, they introduced the unintended "grass-stopping" behavior. The agent optimized the shaped signal rather than the true objective.

4. Visual RL Requires Extended Training: At 500k steps, the agent showed clear learning but had not fully converged (±309 variance). Literature reports 1-2M timesteps for consistent expert performance.

5. Bimodal Performance: Test results show two distinct modes: "success" episodes (700-893 points) and "failure" episodes (<100 points) where a single error triggers grass-stopping.

### 5.2 The "Grass-Stopping" Phenomenon: A Case Study in Reward Hacking

The most significant limitation discovered was a "grass-stopping" behavior: when the agent went off-track onto grass, it would cease all forward movement rather than attempting to steer back. This represents a textbook case of reward hacking:

• The reward shaping wrapper applied -0.5 penalty per timestep on grass
• Stopping immediately minimizes timesteps on grass, thereby minimizing cumulative penalties
• From the agent's perspective, "give up when on grass" became locally optimal
• This behavior was reinforced over 500k steps, becoming the default response

This explains the test bimodality: on easy tracks, the agent rarely went off-track and scored 700-900 (27% of episodes). On challenging tracks with sharp turns, mistakes triggered grass-stopping and minimal reward accumulation (30% of episodes <100 points)

### 5.3 Strengths of the Learned Policy

- Lane Positioning: Maintains center of track on straightaways with minimal drift
- Smooth Steering: Avoids jerky corrections, demonstrating refined motor control
- Throttle Modulation: Reduces speed before curves, accelerates on straights
- Near-Perfect Episodes: 8/30 episodes (27%) achieved 700-893 points
- Zero Early Terminations: All 30 test episodes completed full 1000 timesteps

### 5.4 Remaining Limitations

- Hairpin Turns: Sharp 180° turns frequently trigger off-track incidents
- S-Curves: Rapid direction changes cause overcorrection
- Recovery Failure: Grass-stopping prevents learning recovery behaviors
- High Variance: ±309 standard deviation indicates inconsistent performance

### 5.5 Lessons for Future Work

1. Redesign Reward Shaping: Penalize lack of forward velocity rather than grass presence. Add recovery bonuses for returning from grass to track.

2. Extended Training: Increase to 1-2M timesteps for full convergence.

3. Curriculum Learning: Start on tracks with gentle curves, gradually increasing difficulty.

4. Temporal Modeling: Augment CNN with LSTM layers for better long-range dependencies.

### 5.6 Conclusions

This project demonstrates that deep RL can train an autonomous racing agent from pixels, achieving 440.3 average reward (53% improvement over Milestone 2) with occasional near-perfect performance (893.5 points). The implementation represents a substantial achievement: parallel environments, GPU acceleration, frame stacking, and comprehensive evaluation infrastructure.

However, the "grass-stopping" phenomenon illustrates fundamental challenges in reward engineering. The agent successfully minimized the shaped penalty but failed to learn recovery behavior—a critical lesson for real-world AI systems where misspecified rewards can lead to unintended optimization.

Despite not achieving consistent 700+ performance, this project validates that modern RL can discover complex visuomotor control without domain-specific engineering. With refined rewards and extended training, expert-level performance appears achievable.

## 6. Ethical Considerations

While this project focuses on a simulated environment, the techniques have applications to real-world autonomous systems, requiring consideration of ethical implications.

Societal Impact: Similar systems deployed in autonomous vehicles could reduce accidents (94% caused by human error) but may disrupt transportation employment and raise liability questions.

Bias and Fairness: The procedurally generated environment avoids demographic bias, but real-world training data could inherit and amplify biases (e.g., poor performance on underrepresented road types).

Safety and Robustness: The "grass-stopping" failure demonstrates that RL agents can develop unexpected behaviors when rewards are misspecified. In safety-critical domains, this underscores the necessity of extensive testing, formal verification, fail-safe mechanisms, and human oversight.

Privacy: This project uses only synthetic data. Real-world applications processing camera feeds must implement strict data governance, anonymization, and privacy regulation compliance.

Transparency: The learned policy operates as a "black box." Real-world deployment requires explainability techniques, decision logging, and clear accountability chains.

Limitations: This system is trained in simulation without pedestrians, other vehicles, or traffic rules. Deployment without extensive real-world validation would be irresponsible and dangerous.

## 7. Statement of Individual Contribution

This project was completed independently by me. All work—including the literature review, implementation, training, testing, analysis, visualization, and report writing—was performed solely by me. External libraries (Stable-Baselines, Gymnasium, and PyTorch) were used as documented; however, all integration, configuration, and experimental procedures represent my original work. AI-assisted tools were used only for minor grammar refinement and debugging support, and not for generating core ideas, algorithms, or experimental results.

## 8. References


1. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

2. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

3. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

4. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1–8.

5. Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. L., Deleu, T., ... & Willems, L. (2023). Gymnasium (Version 0.29.0) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.8127026

6. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

7. Farama Foundation. (n.d.). *Gymnasium CarRacing-v3 documentation*. Retrieved December 7, 2023, from https://gymnasium.farama.org/environments/box2d/car_racing/

8. PyTorch. (n.d.). *MPS backend*. PyTorch documentation. Retrieved December 7, 2023, from https://pytorch.org/docs/stable/notes/mps.htm