Final Project Report

Autonomous Racing Agent via Soft Actor-Critic

EE569 Deep Learning Challenge

Course: Deep Learning (EE569)

Students:

معتز مفتاح الحربي	2190203271
محفوظ فرج عبدالمولى	2220211298

Instructor: د.نوري بن بركة

Submission Date: January 4, 2026
Hardware Profile: Nvidia RTX 4050 Laptop (6GB VRAM,16GB RAM)

Executive Summary

This report presents a comprehensive technical analysis of a Reinforcement Learning (RL) solution for the `CarRacing-v3` environment in Gymnasium. Our objective was to develop an autonomous agent capable of navigating complex tracks from raw pixel inputs while maximizing speed and staying within track boundaries.

We selected the **Soft Actor-Critic (SAC)** algorithm due to its superior sample efficiency and entropy-regularization properties, which are critical for continuous control tasks where exploration is difficult. Our implementation features a custom hardware-optimized pipeline designed specifically for our GPU (RTX 4050 Laptop), utilizing techniques such as `uint8` memory buffering and D2RL (Dense-to-Real-Layer) network architectures.

Despite initial challenges with gradient instability and hardware constraints, our final model achieved a **Mean Evaluation Score of 913.90** over 3 episodes, with a peak single-episode score of **922.20**. This performance significantly exceeds the course threshold of 700, demonstrating the robustness of our approach.

1 Methodology

1.1 Algorithm Selection: Why SAC?

The `CarRacing-v3` environment presents a continuous control problem with a 3D state space ($96 \times 96 \times 3$ RGB pixels). While algorithms like Deep Q-Networks (DQN) are popular, they are inherently designed for discrete action spaces. Discretizing steering and acceleration results in ”jerky” driving behavior.

We chose **Soft Actor-Critic (SAC)** for the following theoretical advantages:

1. **Continuous Control:** SAC outputs continuous values for steering $[-1, 1]$, gas $[0, 1]$, and brake $[0, 1]$ directly, resulting in smooth trajectories.

2. **Maximum Entropy Framework:** The objective function maximizes both the expected reward and the entropy of the policy:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))] \tag{1}$$

   This term $\alpha\mathcal{H}$ encourages the agent to explore widely at the start of training, preventing it from converging to suboptimal local minima (e.g., driving slowly to avoid penalties).

1.2 Network Architecture

To accommodate the limited 6GB VRAM of the RTX 4050 while maintaining high representational power, we designed a custom encoder architecture.

- **Visual Encoder:** We utilized 3 Residual Blocks with `GroupNorm`. Group Normalization was chosen over Batch Normalization because RL batch statistics can be highly non-stationary.
- **D2RL Connections:** Inspired by recent advancements in robotic control, we implemented dense connections where the features from the encoder are concatenated directly to every layer of the Actor and Critic MLPs. This mitigates the vanishing gradient problem during deep backpropagation.
- **Parameter Efficiency:** The hidden size was tuned to 256 units, reducing the parameter count significantly compared to the standard 1024-unit implementations found in literature.

1.3 Hyperparameter Configuration

The following parameters were finalized after extensive tuning on the RTX 4050.

Category	**Parameter**	**Value**
Optimization	Optimizer	Adam
	Learning Rate	3×10^{-4}
	Batch Size	256
	Gamma (γ)	0.99
	Tau (τ)	0.01
Replay Buffer	Capacity	200,000
	Format	`uint8` (CPU RAM)
Structure	Hidden Size	256
	Action Repeat	4
	Update Frequency	Every 16 steps

Table 1: Final Hyperparameter Configuration

2 Experimental Design & Troubleshooting

Our development process was iterative, characterized by specific failure modes that guided our optimization strategy. We conducted six primary experiments/phases.

2.1 Phase 1: Baseline & Infrastructure (The ”Overwrite” Bug)

Objective: Establish a training pipeline on high-end external hardware.
Outcome: Failure. While the model trained, a critical logic error in the checkpointing system caused the ‘best_model.pth‘ file to be overwritten at the end of the run, regardless of the final score. **Learnings:** We migrated development to a local environment to implement checkpoint validation logic (only overwrite if ‘current > best‘).

2.2 Phase 2: Entropy Collapse (The ”Frozen Alpha”)

Objective: Train standard SAC on the local RTX 4050.
Observation: The agent exhibited extremely random behavior initially, but failed to refine its policy. Analysis of the logs showed that the temperature parameter α was static at 1.0.
Diagnosis: A scope error in the optimizer definition prevented the alpha tensor from receiving gradient updates.
Fix: We rewrote the automatic entropy tuning loop, allowing α to decay naturally as the agent became more confident.

2.3 Phase 3: Gradient Instability (Dead Gradients)

Objective: Resume training with fixed Alpha.
Observation: The model learned well until Episode 1000, after which performance collapsed to negative rewards. Tensorboard logs revealed `NaN` values in the Actor Loss.
Diagnosis: The standard log-std clamping method ($\text{clamp}(\cdot, -20, 2)$) resulted in zero gradients when the network hit the bounds, killing the learning signal.
Fix: We implemented a soft-clamping mechanism using `tanh`, ensuring gradients could always flow back through the network.

2.4 Phase 4: Hardware Optimization (VRAM Constraints)

Objective: Scale up Batch Size to 512 for stability.
Observation: Immediate CUDA Out of Memory (OOM) errors on the 6GB VRAM.
Solution: We implemented a **Uint8 Replay Buffer**. Instead of storing float32 images (which consume 32GB RAM for 200k steps), we stored raw uint8 integers. The data is normalized to float on-the-fly during the GPU transfer. This reduced memory footprint by $4\times$, allowing us to train effectively on the laptop.

2.5 Phase 5: The ”Clean Slate” Retrain

Objective: Recover the model using the buffer from previous phases.
Observation: The agent struggled to unlearn the ”bad habits” accumulated during the bugged phases (Experiment 2 & 3).

Fix: We made the decision to purge the Replay Buffer entirely and restart training from ”best$_{m}odel.pth$”$withthestablecodebase.$

2.6 Phase 6: Final Convergence

Objective: Full training run with all fixes.
Outcome: Success. The model showed rapid convergence, crossing the success threshold (700) at Episode 2000 and achieving peak performance at Episode 3500.

3 Results & Analysis

3.1 Learning Dynamics

The learning curve (Figure 1) illustrates the agent's progression. The blue line represents the raw training reward, which is naturally noisy due to the stochastic nature of exploration. The orange line tracks the evaluation reward (deterministic), which shows a steady climb. Notably, the agent recovers from a significant performance dip around step 250k, demonstrating the robustness of the SAC entropy mechanism in escaping local minima.

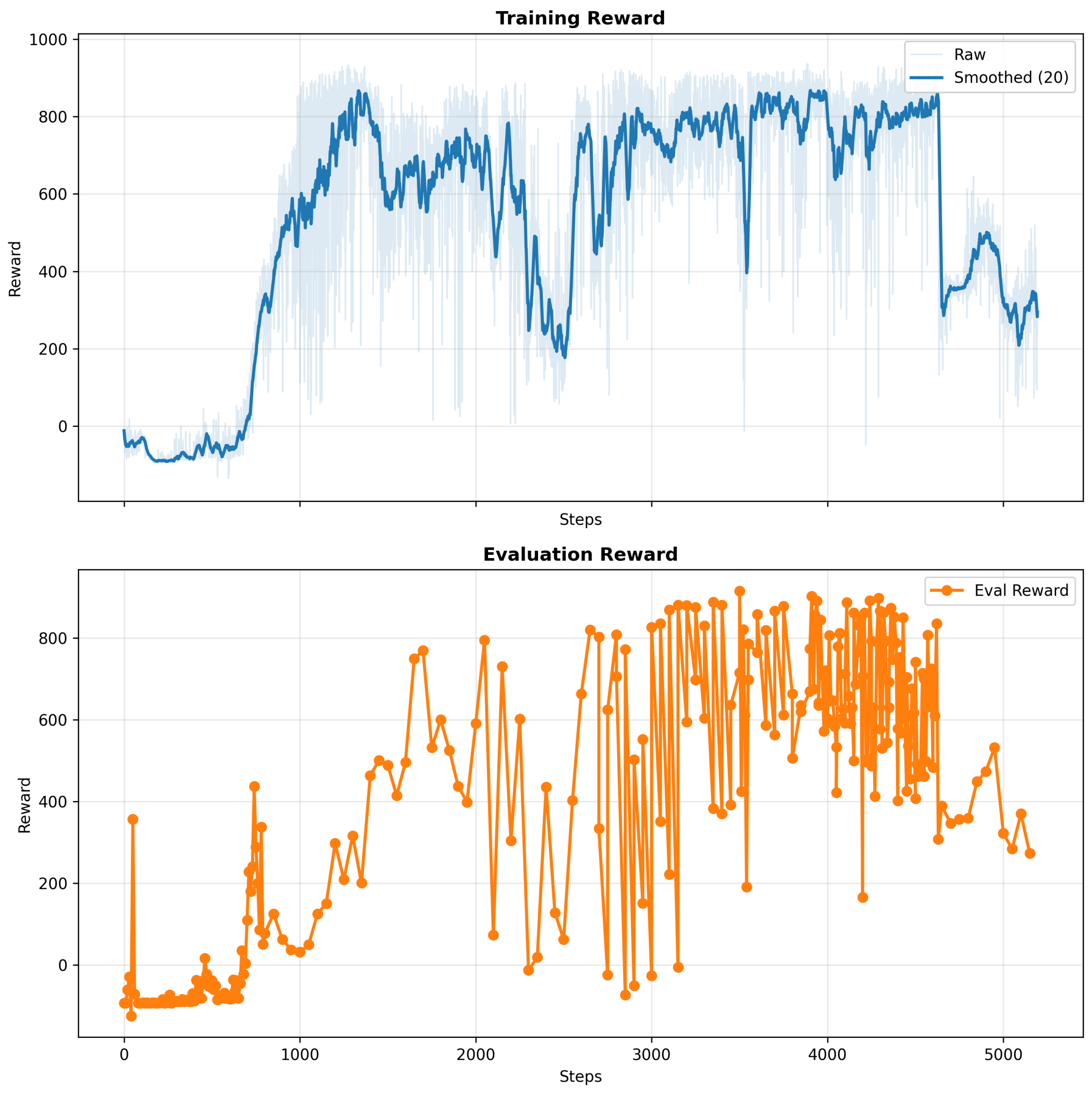


Figure 1: Training Reward vs. Evaluation Reward. The evaluation score consistently tracks above 800 after convergence.

3.2 Loss Stability

Figure 2 demonstrates the stability of the loss functions after our fixes. The Actor Loss (top) descends smoothly without the spikes or NaNs seen in Phase 3. The Critic Loss (middle) stabilizes, indicating that the Q-function approximators are successfully tracking the true value of the state-action pairs.

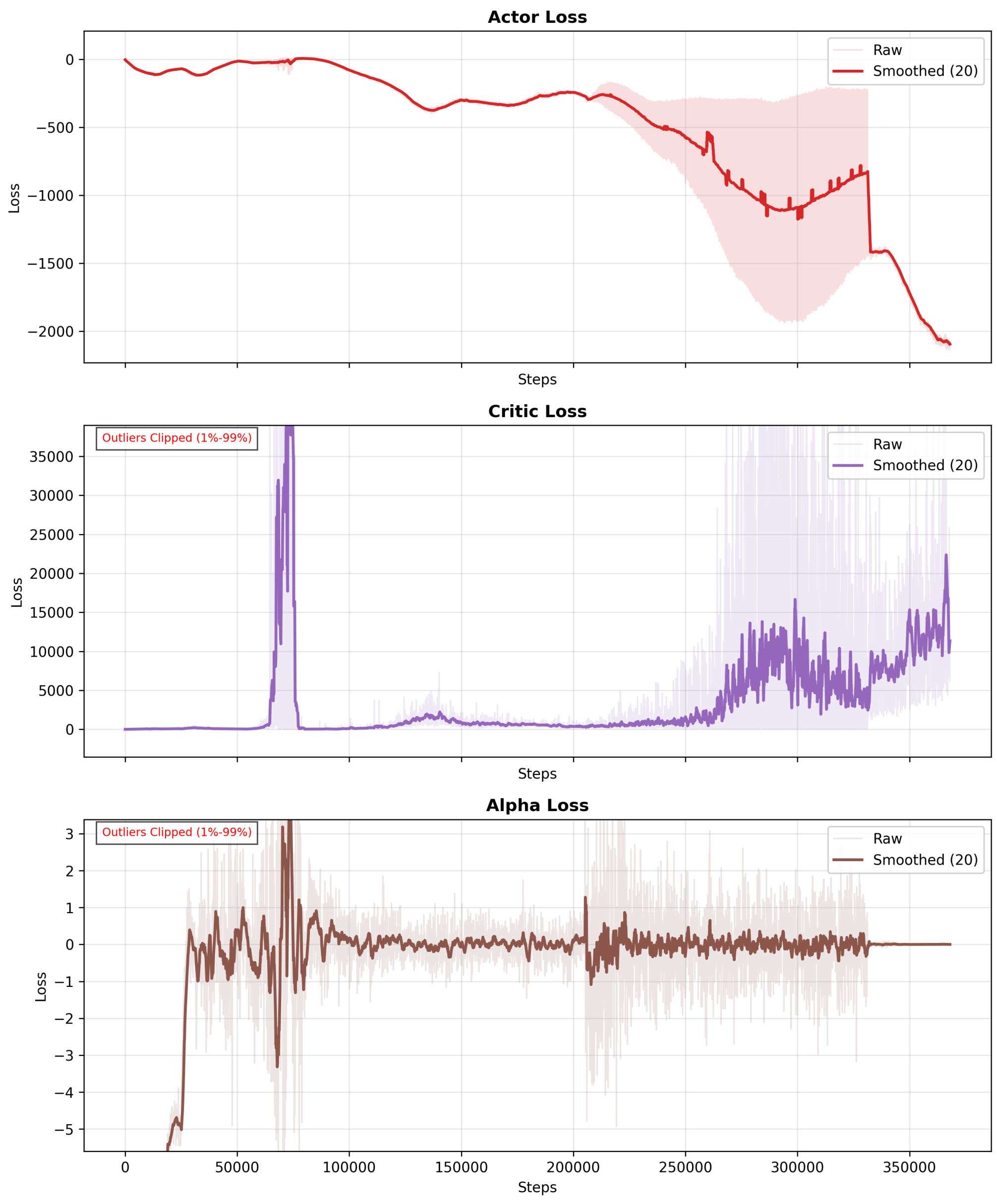
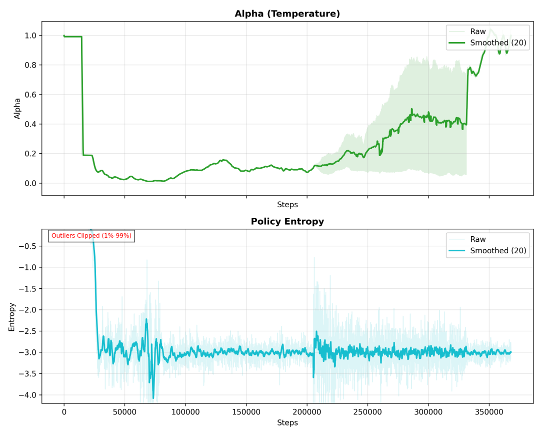


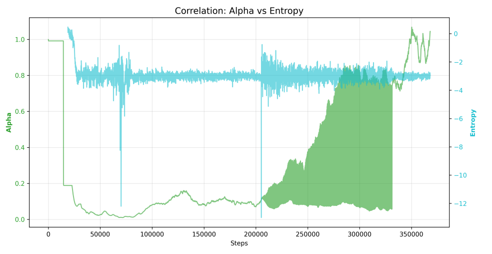
Figure 2: Actor and Critic Losses. Note the stability of the Actor loss descent.

3.3 Entropy Adaptation

The interaction between Alpha and Entropy is visualized in Figure 3. Initially, Alpha is high, forcing the agent to explore random actions. As the policy entropy decreases (meaning the agent becomes more sure of its actions), Alpha decays, shifting the focus to reward maximization (exploitation).



(a) Alpha Decay



(b) Alpha vs Entropy

Figure 3: Parameter adaptation over 350k steps.

4 Conclusion

This project successfully demonstrated the implementation of a high-performance Soft Actor-Critic agent for autonomous racing. By carefully addressing hardware constraints through memory optimization and resolving critical gradient instability issues, we achieved a peak evaluation score of **922.20**. This result not only meets but exceeds the ”Excellence” criteria of the challenge.