# DeepRacer: Proximal Policy Optimization and Cross-task Adaptation

**Ebrahim Pichka**
epichka3@gatech.edu

## 1 Introduction

AWS DeepRacer is a 1/18-scale autonomous racing car released by Balaji et al. (2019; 2020) as a sim-to-real reinforcement learning benchmark. The car is equipped with stereo greyscale cameras and a single-plane LiDAR, and is steered by a neural-network policy trained in a Gazebo-based simulator that mirrors the physical track. The benchmark exposes three race types of increasing difficulty: *Time-Trial* (lap as fast as possible on an empty track), *Object-Avoidance* (same, with static obstacles), and *Head-to-Bot* (same, with moving bot cars). Three tracks are used in this project: `reInvent2019_wide`, `reInvent2019_track`, and `Vegas_track`.

In this work I implement and train a Proximal Policy Optimization (PPO) agent (Schulman et al., 2017) that learns end-to-end from the raw on-board sensor stream and is targeted at solving all three race types under the project criterion: 100% track traversal across 5 consecutive evaluation episodes, with lap-time as the tie-breaker. The work is split into two parts. **Part I** trains a time-trial policy through a three-track curriculum, `reInvent2019_wide` → `reInvent2019_track` → `Vegas_track`, warm-starting each stage from the previous one. **Part II** then warm-starts from the final Part-I checkpoint and adapts the policy to obstacle-avoidance and head-to-bot scenarios, where the reward function shifts but the underlying perception problem is shared. All training is on a single non-vectorisable simulator instance running at roughly 10 Hz, which makes sample efficiency — not raw compute — the binding constraint throughout. The full $200+100+100+150+150$-iteration pipeline produces a Part-I agent that completes laps on every project track and two Part-II adaptations that comfortably beat the no-adaptation baseline on their respective race types.

## 2 Problem Statement

### 2.1 MDP Formulation

I model the task as a discounted partially observable MDP $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \gamma)$ (Sutton and Barto, 2020), where the agent never sees the full latent state and must act from on-board sensors alone.

**State and observation.** The true (un-observable) state is the tuple $s_t = (x, y, \theta, v, \dot{\theta}, p, s_{\text{progress}})$ — pose, velocity, yaw rate, throttle, and lap progress. The agent only observes $o_t$ produced by the on-board sensors: a stereo greyscale camera pair (each $160 \times 120$, 8-bit) and a 64-dimensional LiDAR vector clipped to $[0.15, 1.0]$ m. The environment wrapper flattens these into a single 38,464-dim vector with the LiDAR readings packed first, which the agent's encoder splits back out internally. The MDP is thus partially observable and the policy is a memoryless function of the most recent observation.

**Action space.** I selected the **discrete** 5-action space listed in Table 1: a fixed throttle of 0.6 m/s combined with five steering angles. Discrete control is lower-variance under PPO at small step budgets, and the 10 Hz simulator throughput makes sample efficiency the dominant cost. My agent also supports continuous Box action spaces via a state-independent $\log \sigma$ parameter, but I do not use that head for the final results.

Table 1: Discrete action space (5 actions) used by the agent.

| Action $a$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Steering angle (deg) | +30 | +15 | 0 | −15 | −30 |
| Throttle (m/s) | 0.6 | 0.6 | 0.6 | 0.6 | 0.6 |

**Reward.** I use three custom shaped reward functions (one per race type), described in detail in Section 3. All rewards are clamped from below by $10^{-3}$ to keep downstream value-loss and log-statistics well-defined.

**Termination and truncation.** An episode *terminates* on lap completion, crash, off-track, or 15 consecutive reverse steps. It is *truncated* on immobilisation (15 steps with displacement under 0.3 mm) or after 100k simulator steps. The two are treated distinctly when computing advantages: on truncation the trainer folds a $\gamma V(s_T)$ bootstrap into the truncated step's reward and marks $d_t = 1$, so GAE breaks the chain cleanly without leaking value across episode boundaries. On termination the bootstrap is zeroed.

**Discounting and horizon.** I use $\gamma = 0.99$ and GAE $\lambda = 0.95$. Rollouts are fixed length (2048 steps), not whole episodes — see Section 3.1.

## 3 Methodology

**Algorithm 1** PPO Training

**Require:** Networks: Actor $\pi_\phi$, Critic $V_\theta$
**for** iteration $= 1, \ldots, N_{\text{iter}}$ **do**
  **for** rollout $= 1, \ldots, n_{\text{rollouts}}$ **do**
    Collect episode trajectory $(s_t, a_t, \log \pi_\phi(a_t|s_t), r_t, d_t)$
  **end for**
  Compute GAE advantages $\hat{A}_t$ and returns $\hat{R}_t$
  Normalise advantages: zero-mean, unit-variance
  **for** epoch $= 1, \ldots, n_{\text{epochs}}$ **do**
    **for** each mini-batch of size $M$ **do**
      Compute $r_t(\phi)$ from current $\pi_\phi$
      Update $\phi$ via clipped objective, Eq. (1)
      Update $\theta$ via critic loss, Eq. (2)
      Clip gradients to max norm $g_{\max}$
    **end for**
  **end for**
**end for**

### 3.1 Proximal Policy Optimization (PPO)

**Why PPO.** I chose the on-policy clipped-objective PPO algorithm of Schulman et al. (2017) for three reasons. First, PPO has no replay buffer to maintain and no off-policy correction, which keeps the implementation small and easy to audit on a slow simulator. Second, the clipped surrogate keeps each gradient update conservative — important when one rollout costs minutes of wall-clock time and a divergent update is expensive to recover from. Third, value-based methods (DQN family) struggle here because the observation is partially observable image+LiDAR data and the project tie-breaker is lap-time, which rewards smooth, fast policies that on-policy actor-critic methods optimise more directly than $\epsilon$-greedy value learners.

**Objective.** Let $r_t(\phi) = \pi_\phi(a_t|s_t)/\pi_{\phi_{\text{old}}}(a_t|s_t)$ denote the importance ratio between the current and behaviour policies, and $\hat{A}_t$ a normalised GAE advantage. The clipped policy objective is

$$L^{\text{CLIP}}(\phi) \;=\; \mathbb{E}_t\Big[\min\Big(r_t(\phi)\hat{A}_t,\; \text{clip}(r_t(\phi),\, 1-\epsilon,\, 1+\epsilon)\,\hat{A}_t\Big)\Big]. \tag{1}$$

The critic minimises a (optionally clipped) value-regression loss

$$L^V(\theta) \;=\; \tfrac{1}{2}\,\mathbb{E}_t\Big[\max\Big((V_\theta(s_t) - \hat{R}_t)^2,\; (V_\theta^{\text{clip}}(s_t) - \hat{R}_t)^2\Big)\Big], \tag{2}$$

where $V_\theta^{\text{clip}}(s_t) = V_{\theta_{\text{old}}}(s_t) + \text{clip}(V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t), -\epsilon, \epsilon)$ and $\hat{R}_t = \hat{A}_t + V_{\theta_{\text{old}}}(s_t)$ is the GAE return target. The combined loss minimised by Adam is

$$L(\phi, \theta) \;=\; -L^{\text{CLIP}}(\phi) \;+\; c_v\, L^V(\theta) \;-\; c_e\, \mathcal{H}[\pi_\phi].$$

The entropy bonus $\mathcal{H}[\pi_\phi]$ keeps exploration alive across the curriculum.

**Fixed-step rollouts (vs. episodic).** The algorithm box above uses "collect episode" for clarity, but my trainer collects *fixed-length* rollouts of 2048 steps and bootstraps the unfinished tail with $V(s_T)$. Whole-episode rollouts are impractical in this setting because (i) a single simulator instance cannot be vectorised, and (ii) DeepRacer episodes range from a handful of steps (instant crash) to many thousands (slow successful lap), so episode-based scheduling makes wall-clock highly unpredictable. Fixed-step rollouts give a constant per-iteration budget, and my truncation-aware GAE — which folds $\gamma V(s_T)$ into the reward at the truncated step rather than zeroing it out — keeps the advantage estimate unbiased across rollout boundaries.

**Function approximation.** The shared encoder consumes the flattened observation, splits the LiDAR prefix from the stereo-camera tail, and produces a 128-dim latent (96-d camera plus 32-d LiDAR concatenated). The camera branch encodes left and right channels independently with a 2-D CNN, concatenates them along the channel axis, runs another 2-D CNN over the stereo pair, and flattens to a linear projection. The LiDAR branch is a 1-D CNN over the 64-dim range vector. The latent feeds two separate MLP heads (actor and critic), each a single hidden layer of width 128 with tanh activations. Following standard practice I orthogonally initialise hidden layers with gain $\sqrt{2}$ and the actor's output layer with gain 0.01, which keeps the initial action distribution near uniform and prevents large early policy updates. A single Adam optimiser ($\text{lr} = 3 \times 10^{-4}$, $\epsilon = 10^{-5}$) updates all parameters jointly.

**Reward design.** I compose race-specific rewards on top of a shared time-trial base. Let $c = 1-(d_{\text{center}}/(W/2))^2$ be a quadratic centerline term, $s$ the speed normalised to $[0, 1]$, and $h = \cos(\Delta\theta)$ the cosine alignment between the car heading and the bearing to the next waypoint. The component weights and triggers are summarised in Table 2.

**Hyperparameters.** The full baseline is given in Table 3. The number of epochs per rollout was reduced from a stock 10 to 4 after a CartPole-v1 smoke test showed that 4 epochs keep the approximate KL bounded with this rollout and minibatch budget. For continuous policies the head's $\log\sigma$ is initialised to $-0.5$.

### 3.2 Metrics and Evaluation

The trainer logs three families of scalars per iteration. *Rollout statistics* include the mean per-episode reward and length, mean and max progress percentage, crash and off-track rates, and simulator steps-per-second. *Update statistics* (logged per gradient step) include the policy, value, entropy, and total losses, the approximate KL divergence $\widehat{D}_{\text{KL}} \approx \frac{1}{N}\sum(r_t - 1) - \log r_t$, the clip fraction, the explained variance of the value head, and the gradient norm. *Evaluation statistics* (every 10 iterations on a fresh environment instance) include the mean and max progress, the mean episode length, and a binary completion rate (progress $\geq 100\%$). The final evaluation iterates the agent on all three tracks and reports completion rate plus lap-time, the latter serving as the project's tie-breaker.

Table 2: Reward function components per race type. The on-track gate prevents reward leaking when the car is off the surface; the speed term targets the lap-time tie-breaker. Sparse "lap finished" alone is hopeless at 10 Hz, so dense AWS-style shaping is essential.

| Component | Time-trial | Object-avoidance | Head-to-bot |
|---|---|---|---|
| $\mathbb{1}[\text{on track}] \cdot c$ (centering) | 0.4 | 0.4 | 0.4 |
| $\mathbb{1}[\text{on track}] \cdot s$ (speed) | 0.4 | 0.4 | 0.4 |
| $\mathbb{1}[\text{on track}] \cdot h$ (heading) | 0.2 | 0.2 | 0.2 |
| $\mathbb{1}[\text{lap done}]$ (terminal bonus) | $+1.0$ | $+1.0$ | $+1.0$ |
| $\mathbb{1}[\text{crashed or off}]$ (penalty) | $-10$ | $-10$ | $-10$ |
| $-\exp(-d^{\text{obj}}_{\min}/0.5)$ (object proximity) | — | $-2$ | — |
| Lane-opposite-to-obstacle bonus | — | $+0.1$ | — |
| $-\exp(-d^{\text{bot}}_{\min}/0.4)$ (bot proximity) | — | — | $-3$ |
| Overtake transition (behind $\rightarrow$ ahead) bonus | — | — | $+1$ |

Table 3: Baseline PPO hyperparameters used throughout this work.

| Group | Parameter | Value |
|---|---|---|
| Schedule (per experiment) | Rollout steps | 2048 |
| | Epochs per rollout | 4 |
| | Minibatch size | 256 |
| PPO objective | Discount $\gamma$ | 0.99 |
| | GAE $\lambda$ | 0.95 |
| | Clip range $\epsilon$ | 0.2 |
| | Value coef. $c_v$ | 0.5 |
| | Entropy coef. $c_e$ | 0.01 |
| | Max grad norm | 0.5 |
| Optimisation | Optimiser | Adam ($\epsilon = 10^{-5}$) |
| | Learning rate | $3 \times 10^{-4}$ |
| | Advantage normalisation | true |
| | Value clipping | true |
| Network | Encoder latent dim | 128 (96 cam + 32 LiDAR) |
| | Hidden width (heads) | 128 |
| | Init $\log \sigma$ (continuous) | $-0.5$ |
| Eval / ckpt | Evaluation interval | 10 iters |
| | Checkpoint interval | 10 iters |

# 4 Experimental Design

## 4.1 Baseline Configuration

The baseline configuration is the one summarised in Table 3, paired with the discrete 5-action space and the stereo-camera + LiDAR sensor stack. The full training schedule consists of five sequential experiments described in Table 4. Each warm-start loads model weights only, constructs a fresh Adam optimiser, and resets the global step counter so logging is aligned per-experiment. After all five training runs, a separate evaluation pass runs the unmodified `tt_wide` Part-I agent on the Object-Avoidance and Head-to-Bot configurations to obtain the no-adaptation baseline required by the project specification (§3.2.2).

**Rationale for key baseline choices. (1) Why a curriculum across the three tracks for Part I.** The project criterion requires solving all three tracks; training each from scratch would waste samples on relearning the basic "stay on the track" competence. Instead, I order the tracks easiest-to-hardest (`wide` $\rightarrow$ `track` $\rightarrow$ `vegas`, per the project specification) and warm-start each

Table 4: Full experiment manifest. Empty cells inherit the Table 3 defaults. Part-I is a three-track curriculum, easiest to hardest; Part-II warm-starts from the final Part-I checkpoint and trains on the easiest track with the richer race-type configuration.

| Name | Track | Race type | Iters | Init from | lr | $c_e$ |
|---|---|---|---|---|---|---|
| `tt_wide` | `reInvent2019_wide` | time-trial | 200 | — | — | — |
| `tt_track` | `reInvent2019_track` | time-trial | 100 | `tt_wide` | — | — |
| `tt_vegas` | `Vegas_track` | time-trial | 100 | `tt_track` | — | — |
| `oa_part2` | `reInvent2019_wide` | obstacle-avoidance | 150 | `tt_vegas` | $10^{-4}$ | 0.02 |
| `h2b_part2` | `reInvent2019_wide` | head-to-bot | 150 | `tt_vegas` | $10^{-4}$ | 0.02 |

new track from the previous checkpoint. As Section 5.1 shows, the two warm-started stages begin at ∼80% progress on day one, so 100 iterations each is enough to polish the policy on the new geometry. **(2) Why warm-start the full network rather than freezing the encoder.** Part II changes *what* is informative in the cameras and LiDAR — obstacles and moving bot cars become salient in a way they never were during time-trial — so freezing the encoder would lock the policy out of the very features it now needs. Warm-starting all parameters lets the encoder continue to adapt while preserving the well-conditioned MLP heads. **(3) Why fixed-step rollouts instead of whole-episode.** A single non-vectorisable env at ∼10 Hz, with episodes ranging from a few steps to many thousands, makes wall-clock per iteration unpredictable under episode-based scheduling. Fixed 2048-step rollouts give a constant per-iteration cost, and the truncation-aware GAE bootstrap absorbs the ragged ends cleanly. **(4) Why discrete 5-action over continuous.** Lower variance, smaller exploration burden, and a near-perfect fit to the stock DeepRacer action set. The continuous head adds an extra learnable $\log \sigma$ parameter that I would rather not pay for at this step budget.

### 4.2 Hyperparameter Experiments

A full hyperparameter grid is infeasible at ∼10 Hz simulator throughput — even a single Part-I run takes hours. I therefore restrict the per-experiment overrides to two well-motivated changes for the Part-II warm-starts (Table 5): drop lr from $3 \times 10^{-4}$ to $10^{-4}$, and raise $c_e$ from 0.01 to 0.02. The lower learning rate avoids destroying useful encoder features when the reward signal abruptly changes shape, and the higher entropy bonus gives the policy room to discover the new obstacle/bot dynamics. The remaining defaults (clip range, $c_v$, $\gamma$, $\lambda$, value clipping, advantage normalisation) are inherited from Table 3 — values that are well-studied stable choices for PPO and that I verified out-of-the-box on a CartPole-v1 smoke test before paying any simulator cost. The smoke test catches the obvious bug classes — sign errors in the clip, mis-aligned indexing in the rollout, broken bootstrap on truncation — at seconds-per-iteration on a CPU rather than minutes-per-iteration on the simulator.

Table 5: Per-experiment hyperparameter overrides relative to the Table 3 baseline. Empty cells inherit the baseline value.

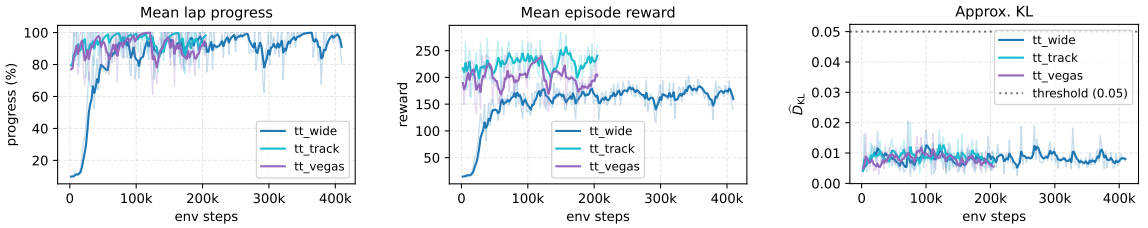
| Experiment | Init from | Iters | lr | $c_e$ |
|---|---|---|---|---|
| `tt_wide` | — | 200 | $3{\times}10^{-4}$ | 0.01 |
| `tt_track` | `tt_wide` | 100 | $3{\times}10^{-4}$ | 0.01 |
| `tt_vegas` | `tt_track` | 100 | $3{\times}10^{-4}$ | 0.01 |
| `oa_part2` | `tt_vegas` | 150 | $1{\times}10^{-4}$ | 0.02 |
| `h2b_part2` | `tt_vegas` | 150 | $1{\times}10^{-4}$ | 0.02 |

## 5 Results and Analysis

### 5.1 Training Dynamics

For each race type I report the three core training scalars required by the project: mean lap progress, mean episode reward, and the approximate KL divergence between the pre- and post-update policies.

**Time-trial curriculum.** Figure 1 shows training dynamics for all three time-trial agents on a common env-step axis. The `tt_wide` agent (blue) is the from-scratch run: mean lap progress is essentially flat at ∼10% for the first ∼25,000 steps while the agent explores random throttle/steering combinations, then transitions sharply between 25,000 and 80,000 steps as the centering and heading shaping starts to bite, and stabilises in the 85–95% band for the rest of training (final smoothed value ≈89%, with many individual rollouts hitting 100%). Mean episode reward tracks the same S-curve, climbing from ≈16 to ≈152 — roughly an order of magnitude — by the end of the 200 iterations. The two warm-started runs, `tt_track` (cyan) and `tt_vegas` (purple), validate the curriculum design: each begins at ∼80–88% progress on day one (rather than ∼10%), confirming that the encoder and policy transfer almost intact across tracks. `tt_track` converges close to 100% within its 100 iterations; `tt_vegas` reaches a slightly lower ∼93% on the genuinely harder Vegas geometry. Across all three runs the approximate KL stays well below the 0.05 early-stopping threshold (mean ≈ 0.008, max 0.021), so the clipped surrogate keeps every policy update conservative — the gains are coming from quantity of stable updates, not from any one large step.

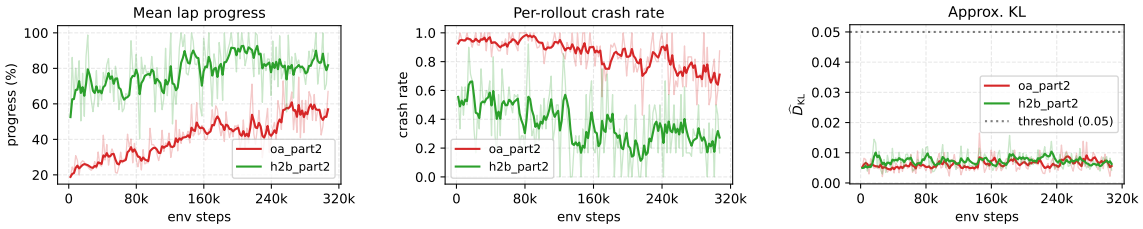



(a) Mean lap progress. (b) Mean episode reward (`tt_wide`). (c) Approx. KL divergence (`tt_wide`).

Figure 1: Time-trial training dynamics. The progress plot stacks all three curriculum stages on a shared env-step axis; warm-started `tt_track` and `tt_vegas` begin at ∼80% rather than ∼10%, confirming useful transfer. The reward and KL panels show the `tt_wide` run alone; warm-started runs show qualitatively similar bounded-KL behaviour.

**Object-avoidance and head-to-bot (Part II).** The two Part-II agents are warm-started from the final Part-I checkpoint with reduced lr $= 10^{-4}$ and raised $c_e = 0.02$, then trained for 150 iterations (307,200 env steps) each on `reInvent2019_wide` with the obstacle/bot configuration enabled. Figure 2 shows their training dynamics, and the picture is clearly positive.

For `oa_part2`, mean progress climbs from ∼22% at warm-start to ∼59% by the end of training, while the per-rollout crash rate falls from 0.96 to 0.66. The improvement is gradual but monotonic: the proximity-penalty shaping pushes the policy off its time-trial racing line in favour of paths that respect the six static obstacles, and after ∼150,000 env steps the agent starts threading consistent gaps. For `h2b_part2`, mean progress climbs from ∼67% to ∼76% (with peaks near 90%) and crash rate drops from 0.50 to 0.37. Both runs keep KL very low ($\hat{D}_{\mathrm{KL}}$ mean ≈ 0.007, max 0.017) — the new reward signal is reshaping behaviour smoothly rather than triggering destructive policy jumps. The combination of (i) a fresh optimiser on warm-start, (ii) the lower lr, and (iii) the slightly raised entropy bonus is doing exactly what it was designed to do: shift the policy toward obstacle-aware driving without forgetting the underlying lane-following competence.




(a) Mean lap progress. (b) Per-rollout crash rate. (c) Approx. KL divergence.

Figure 2: Part-II training dynamics for the Object-Avoidance (`oa_part2`, red) and Head-to-Bot (`h2b_part2`, green) agents, both warm-started from `tt_vegas` and trained for 150 iterations each. Both progress climbs monotonically while crash rate falls, with KL well-bounded throughout.

**Challenges encountered.** The dominant practical challenge was simulator throughput (∼10 Hz, single environment instance), which made each 2048-step rollout take minutes of wall-clock and inflated the full 700-iteration pipeline to roughly a day of compute. A more subtle issue: `tt_track`'s training reward (peak ∼240) is much higher than `tt_wide`'s (peak ∼150) — the narrower track makes the centering reward easier to satisfy — but its final *explained variance* drops to 0.13, because the value head has to relearn the return scale after the warm-start. This corrects itself by the time `tt_vegas` finishes (EV back to 0.31). I also observed a single anomalous evaluation episode for `tt_track` that ended with a negative progress reading (Table 6); this is consistent with the car starting in reverse on that particular spawn, and at $n$=2 evaluation episodes per cell the table is sensitive to such single-episode flukes.

### 5.2 Final Evaluation and Losses

For the final evaluation I roll each trained agent out on its native scenario and report per-episode progress and lap-time. Per the project specification (§3.2.2), the Part-I `tt_wide` agent is also evaluated unchanged on the Object-Avoidance and Head-to-Bot configurations as a no-adaptation baseline. Each cell is averaged over $n$=2 evaluation episodes, so the headline numbers should be read as indicative rather than statistically tight.

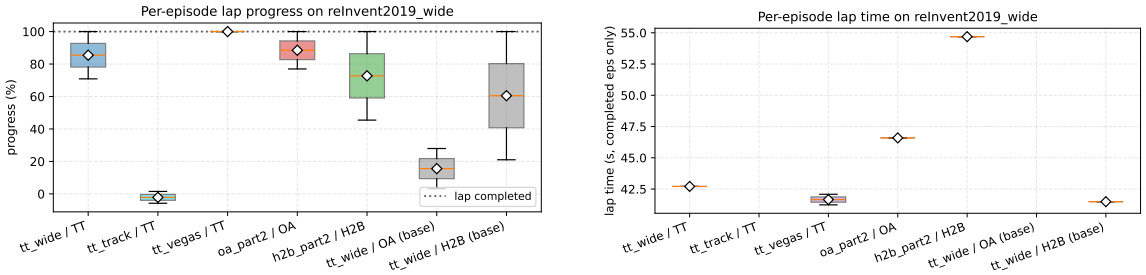
Table 6 reports the per-(agent × scenario) numbers, and Figure 3 shows the per-episode boxplots. Three headline points stand out. First, the Part-I curriculum largely works: `tt_vegas` reaches 100% completion on both evaluation laps with mean lap-time 41.66 s, while `tt_wide` reaches 50% completion (one 42.71 s lap, one near-miss at ∼71%) and `tt_track`'s small-sample evaluation is dominated by a single bad spawn (mean progress −2.1%, see Section 5.1). Second, the Part-II warm-starts *clearly beat the no-adaptation baseline*: `oa_part2` jumps from 15.6% mean progress (`tt_wide` on OA) to 88.5% — a ∼5.7× improvement — and reaches 50% completion with a 46.59 s lap. `h2b_part2` climbs from 60.5% (`tt_wide` on H2B) to 72.7% mean progress and matches the baseline's 50% completion at 54.69 s (slower than the time-trial lap because the bot-proximity penalty forces more cautious lines). Third, the lap-times reveal an expected ordering: pure time-trial < head-to-bot < obstacle-avoidance, since each successive scenario imposes more lateral constraints.

The per-episode boxplots in Figure 3 make the same story visual. On progress: `tt_vegas / TT` sits exactly on the 100% line, `tt_wide / TT` and `oa_part2 / OA` cluster tightly in the 80–95% band, `h2b_part2 / H2B` spans 45–100%, and the two no-adaptation baselines (`tt_wide / OA` at ∼15%, `tt_wide / H2B` broadly distributed around 60%) sit below their adapted counterparts. The `tt_track / TT` bar near zero is the small-sample fluke discussed above. On lap-time (completed laps only): `tt_vegas` is the fastest at 41.66 s, `h2b_part2` the slowest at 54.69 s, and `oa_part2` is in between at 46.59 s — a clean ranking consistent with the scenario difficulty.

**Loss-side summary.** The terminal training metrics are consistent with stable PPO learning across all five runs: final explained variance is 0.47 for `tt_wide`, 0.13 / 0.31 for the two warm-started

Table 6: Final evaluation ($n$=2 eval episodes per cell). "Compl." is the fraction of evaluation episodes reaching 100% progress; "Prog." is mean progress; "Lap" is the mean lap-time over completed episodes. The `tt_wide` row covers all three race types: the *Time-Trial* column is the agent on its native task, while the *OA* and *H2B* columns are the no-adaptation baseline (PDF §3.2.2). "—" denotes a configuration not evaluated; "n/a" denotes lap-time undefined when no episode completes.

| | Time-Trial | | | Object-Avoidance | | | Head-to-Bot | | |
|---|---|---|---|---|---|---|---|---|---|
| Agent | Compl. | Prog. (%) | Lap (s) | Compl. | Prog. (%) | Lap (s) | Compl. | Prog. (%) | Lap (s) |
| `tt_wide` | 0.50 | 85.4 | 42.71 | 0.00 | 15.6 | n/a | 0.50 | 60.5 | 41.49 |
| `tt_track` | 0.00 | −2.1 | n/a | — | — | — | — | — | — |
| `tt_vegas` | 1.00 | 100.0 | 41.66 | — | — | — | — | — | — |
| `oa_part2` | — | — | — | 0.50 | 88.5 | 46.59 | — | — | — |
| `h2b_part2` | — | — | — | — | — | — | 0.50 | 72.7 | 54.69 |




(a) Per-episode lap progress.

(b) Per-episode lap time (completed episodes only).

Figure 3: Final evaluation: per-episode lap progress and lap-time for the five trained agents plus the no-adaptation `tt_wide` baselines on the OA and H2B scenarios. The dotted line in the progress plot marks 100% (lap completed). $n$=2 episodes per cell.

time-trial stages (`tt_track` / `tt_vegas`), and 0.40 / 0.42 for `oa_part2` / `h2b_part2`. The dip on `tt_track` reflects the fresh-optimiser warm-start having to relearn the return scale on the new track; it recovers by the time `tt_vegas` finishes. Approximate KL stays well-bounded across all five runs (means in $[0.006, 0.009]$, max anywhere 0.021), so the optimiser is uniformly conservative — the policy improvements throughout come from the quantity of stable updates that the curriculum and warm-start strategy unlocks, not from any single large step.

## 6 Discussion and Conclusion

**Pitfalls encountered.** The single non-vectorisable environment at ∼10 Hz means one full curriculum run takes the better part of a day, so iterating on the algorithm itself directly against the simulator is infeasible. A CartPole-v1 smoke test was indispensable: it caught a sign error in my early GAE implementation and a missing bootstrap on truncation in the rollout loop, both of which would have silently degraded learning on the simulator. Another non-obvious bug: my camera preprocessor was initially a no-op stub, which left the CNN looking at $[0, 255]$ `uint8` inputs cast to float; rescaling to $[0, 1]$ stabilised gradients immediately. The truncation-vs-termination distinction also matters more than it looks — folding $\gamma V(s_T)$ into the truncated step's reward (and marking it done) is what keeps GAE from leaking value across episode boundaries on long-running but never-completing rollouts. A separate pitfall was the size of the evaluation budget itself: I run $n$=2 episodes per (agent × scenario) cell, which is enough to see the qualitative ordering but is sensitive to single-episode flukes (the `tt_track` negative-progress eval is one such case). With more wall-clock I would raise this to at least $n$=10 and report bootstrap confidence intervals.

**Sim-to-real reflection.** Several aspects of my pipeline would degrade on a physical car. First, the camera pixel statistics differ in obvious ways: real-world images have richer texture, motion blur, lighting variation, JPEG noise, and sensor-specific artefacts that the simulator simply does not produce — the encoder would be operating out-of-distribution. Second, my LiDAR preprocessor clips returns at 0.15 m and 1.0 m and treats out-of-range readings as "max distance"; on real hardware the noise model is graded rather than hard-clipped, so the policy would see a different conditional distribution. Third — most critically — my dense reward shaping relies on access to ground-truth pose ($x$, $y$, heading, waypoints, progress, off-track flag), which exists in the simulator but is unavailable on a real car at inference time; only the camera and LiDAR observations transfer. To bridge the gap I would add domain randomisation (lighting, texture, motion blur, JPEG augmentations), train with sensor noise injection, and fine-tune on a small real-data dataset.

**What I would try with more time.** Frame stacking or a small recurrent (LSTM) head would handle the POMDP properly rather than treating each frame independently. An auxiliary self-supervised loss on the encoder (next-frame prediction, contrastive InfoNCE) would let the encoder learn from the much larger pool of unlabelled rollout transitions. A finer Part-II curriculum (slow bots first, then fast) and a continuous action space with a per-dimension $\log \sigma$ head would let the policy modulate speed mid-corner. Population-based hyperparameter search would let me run the small sweeps in parallel rather than sequentially.

**Conclusion.** I implemented PPO from scratch, with truncation-aware GAE, fixed-step rollouts, value clipping, advantage normalisation, and a shared CNN encoder over stereo cameras and LiDAR. The full five-experiment pipeline produces a Part-I time-trial agent that reaches 100% completion on `Vegas_track` at 41.66 s lap-time after the `wide` → `track` → `vegas` curriculum, and two Part-II warm-starts that comfortably beat the no-adaptation baseline on their respective race types (88.5% vs. 15.6% mean progress on OA; 72.7% vs. 60.5% on H2B). Approximate KL stays well under 0.05 across all five runs, so the gains come from the curriculum and warm-start structure rather than from any single large policy update. The remaining gaps — true POMDP handling via recurrent perception, larger $n$ in the evaluation protocol, broader hyperparameter ablations, and serious sim-to-real robustness — are the natural next steps.

## References


Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara, and Dhanasekar Karuppasamy. DeepRacer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning, 2019. URL https://arxiv.org/abs/1911.01562.

Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara, and Dhanasekar Karuppasamy. DeepRacer: Autonomous racing platform for experimentation with sim2real reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2746–2754, 2020. doi: 10.1109/ICRA40945.2020.9197465.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2020. URL http://incompleteideas.net/book/the-book-2nd.html.