Pontifícia Universidade Católica
do Rio de Janeiro

Gustavo Arcary Passos

Análise Comparativa de Algoritmos de Aprendizado por Reforço Profundo em Ambiente Local de Simulação DeepRacer-for-Cloud

Projeto Final de Curso

Projeto apresentado como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação, Informática da PUC-Rio .

Orientador: Prof. Augusto Cesar Espíndola Baffa

Rio de Janeiro
Dezembro de 2025

Pontifícia Universidade Católica
do Rio de Janeiro

Gustavo Arcary Passos

Análise Comparativa de Algoritmos de Aprendizado por Reforço Profundo em Ambiente Local de Simulação DeepRacer-for-Cloud

Projeto apresentado como requisito parcial para obtenção do grau de Bacharel em Engenharia da Computação da PUC-Rio . Aprovada pela Comissão Examinadora abaixo:

Prof. Augusto Cesar Espíndola Baffa
Orientador
Departamento de Informática – PUC-Rio

Rio de Janeiro, 10 de Dezembro de 2025

Gustavo Arcary Passos

Ficha Catalográfica

Arcary Passos, Gustavo

Análise Comparativa de Algoritmos de Aprendizado por Reforço Profundo em Ambiente Local de Simulação DeepRacer-for-Cloud / Gustavo Arcary Passos; orientador: Augusto Cesar Espíndola Baffa. – 2025.

80 f: il. color. ; 30 cm

Projeto (graduação) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2025.

Inclui bibliografia

1. Engenharia da Computação – Teses. 2. DeepRacer. 3. AWS. 4. Algoritmos Evolutivos. 5. aprendizado por reforço profundo. I. Espíndola Baffa, Augusto Cesar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Agradeço à minha família pelo apoio incondicional ao longo de toda a minha formação. Aos meus amigos da faculdade, pela parceria, incentivo e pelas inúmeras conversas que tornaram esta jornada mais leve. Aos professores do Departamento de Informática da PUC-Rio, em especial ao meu orientador, pelo apoio, orientação e dedicação na construção deste trabalho. Também agradeço a todos que, direta ou indiretamente, contribuíram para o meu crescimento acadêmico e pessoal durante este período.

Resumo

Arcary Passos, Gustavo; Espíndola Baffa, Augusto Cesar. **Análise Comparativa de Algoritmos de Aprendizado por Reforço Profundo em Ambiente Local de Simulação DeepRacer-for-Cloud**. Rio de Janeiro, 2025. 80p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho tem como objetivo analisar o desempenho de algoritmos de aprendizado por reforço profundo aplicados à tarefa de condução autônoma em ambientes de simulação. A pesquisa utiliza a plataforma AWS DeepRacer e sua versão de código aberto, o DeepRacer-for-Cloud (DRfC), para realizar treinamentos e experimentos locais. Foram empregados os algoritmos *Proximal Policy Optimization* (PPO) e *Soft Actor-Critic* (SAC), comparando-se seus resultados em diferentes pistas e configurações de treinamento. A análise busca compreender como variações nos hiperparâmetros e na função de recompensa impactam o desempenho dos agentes, bem como discutir as vantagens e limitações do uso de uma infraestrutura local em relação ao serviço em nuvem da AWS.

Palavras-chave

DeepRacer; AWS; Algoritmos Evolutivos; aprendizado por reforço profundo.

Abstract

Arcary Passos, Gustavo; Espíndola Baffa, Augusto Cesar (Advisor). **Comparative Analysis between Evolutionary Algorithms and Deep Reinforcement Learning in the AWS DeepRacer Environment**. Rio de Janeiro, 2025. 80p. Projeto Final – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work aims to analyze the performance of deep reinforcement learning algorithms applied to autonomous driving tasks in simulation environments. The study uses the AWS DeepRacer platform and its open-source version, DeepRacer-for-Cloud (DRfC), to perform local training and experiments. The algorithms *Proximal Policy Optimization* (PPO) and *Soft Actor-Critic* (SAC) were employed, comparing their results across different tracks and training configurations. The analysis seeks to understand how variations in hyperparameters and reward functions affect agent performance, as well as to discuss the advantages and limitations of using a local infrastructure compared to the AWS cloud service.

Keywords

DeepRacer; AWS; Evolutionary Algorithms; Deep Reinforcement Learning.

Sumário

Lista de figuras

Lista de tabelas

Lista de Códigos

Lista de Abreviaturas

AWS – *Amazon Web Services*

DRfC – *DeepRacer-for-Cloud*

PPO – *Proximal Policy Optimization*

RL – *Reinforcement Learning*

DRL – *Deep Reinforcement Learning*

SAC – *Soft Actor-Critic*

TRPO – *Trust Region Policy Optimization*

WSL2 – *Windows Subsystem for Linux 2*

Follow your bliss.

Joseph Campbell, *The Power of Myth.*

1
Introdução

O avanço das técnicas de aprendizado por reforço tem impulsionado o desenvolvimento de veículos autônomos, permitindo que agentes artificiais aprendam a realizar tarefas complexas por meio da interação com o ambiente e da maximização de uma função de recompensa (KAMIL; ABDULAZEEZ, 2024). Entre as plataformas educacionais e experimentais voltadas a essa área, destaca-se a *AWS DeepRacer*, um ecossistema desenvolvido pela Amazon Web Services (AWS) que permite o treinamento e a avaliação de modelos de condução autônoma em ambientes de simulação 3D e, opcionalmente, em veículos físicos.

A motivação central deste trabalho surge da necessidade de compreender o funcionamento interno desse ambiente de treinamento — em especial seus algoritmos de aprendizado por reforço — e explorar alternativas que permitam replicar o processo de treinamento fora da nuvem da AWS. Embora o serviço original da AWS ofereça uma infraestrutura robusta e uma interface amigável, seu uso contínuo pode gerar custos significativos e limitar a flexibilidade de experimentação e personalização do ambiente.

O problema que se propõe investigar consiste, portanto, em avaliar o comportamento e o desempenho de algoritmos de aprendizado por reforço — especificamente *Soft Actor-Critic* (SAC) e *Proximal Policy Optimization* (PPO) — aplicados ao treinamento de veículos autônomos em ambiente de simulação local, utilizando a infraestrutura disponibilizada pela comunidade *DeepRacer-for-Cloud* (DRfC) (COMMUNITY, 2024). Essa abordagem permite reproduzir, com menor custo, a lógica do treinamento oferecido pela AWS, possibilitando o controle direto sobre parâmetros e recursos computacionais.

A relevância deste estudo reside na possibilidade de compreender, em detalhes, o comportamento de algoritmos de aprendizado por reforço em cenários de navegação autônoma e na demonstração de que é viável realizar treinamentos complexos sem depender integralmente de soluções proprietárias ou de alto custo. Além disso, a análise comparativa entre os algoritmos SAC e PPO, bem como a influência de diferentes configurações de pista e parâmetros de treinamento, contribui para o entendimento de como ajustes finos impactam o desempenho final dos modelos.

Em termos de ambiente computacional, o trabalho foi desenvolvido sobre uma infraestrutura local composta por contêineres Docker configurados via o repositório *DeepRacer-for-Cloud*, que recria a arquitetura da AWS DeepRacer.

O sistema foi executado em plataforma PC com sistema operacional Ubuntu, utilizando bibliotecas Python voltadas ao aprendizado por reforço, como *TensorFlow* e *OpenAI Gym*. O ambiente de simulação 3D e os logs de treinamento foram gerenciados em ambiente Linux, permitindo o acompanhamento detalhado do processo de aprendizado e a análise quantitativa dos resultados.

Por fim, quanto à adequação como Projeto Final, o trabalho integra conceitos de diversas disciplinas do curso de Engenharia da Computação, incluindo aprendizado de máquina, inteligência artificial, sistemas distribuídos, virtualização e programação em Python. A execução prática do projeto envolveu o uso de ferramentas de desenvolvimento modernas, análise experimental, interpretação de métricas de desempenho e compreensão dos princípios fundamentais do aprendizado por reforço, evidenciando a aplicação integrada dos conhecimentos adquiridos ao longo da formação.

2
Situação Atual

O desenvolvimento de agentes autônomos capazes de tomar decisões em tempo real é um dos principais desafios do campo da Inteligência Artificial aplicada à robótica e à condução autônoma. No contexto da simulação virtual, diversas plataformas têm sido utilizadas para o treinamento e a avaliação desses agentes, como o *CARLA*, o *AirSim* e o *AWS DeepRacer*. Cada uma oferece diferentes graus de fidelidade física, capacidade de personalização e integração com algoritmos de aprendizado por reforço profundo (*Deep Reinforcement Learning — DRL*) (DOSOVITSKIY et al., 2017; SHAH et al., 2018; SERVICES, 2019).

Entre essas alternativas, a escolha pela plataforma *AWS DeepRacer* mostra-se estratégica: ela fornece um ambiente acessível, com infraestrutura em nuvem dedicada ao treinamento de modelos de DRL, e integração direta entre o simulador virtual e o veículo físico. Além disso, sua ampla adoção pela comunidade, tanto acadêmica quanto corporativa, favorece a reprodutibilidade dos experimentos e a comparação entre resultados.

Nos últimos anos, surgiram também iniciativas da comunidade que visam reproduzir localmente o ambiente da AWS, como o projeto *DeepRacer-for-Cloud* (DRfC). Essa solução possibilita a execução dos mesmos algoritmos e funções de recompensa utilizados na plataforma oficial, porém em ambiente controlado e de custo reduzido, o que amplia o potencial de pesquisa e experimentação.

O presente trabalho insere-se nesse contexto, concentrando-se na análise comparativa de dois algoritmos amplamente utilizados em tarefas de controle contínuo: *Proximal Policy Optimization* (PPO) e *Soft Actor-Critic* (SAC). Ambos são algoritmos baseados em gradiente de política e representam abordagens modernas e consolidadas no campo do aprendizado por reforço profundo.

2.1
Trabalhos Relacionados

O *AWS DeepRacer* foi introduzido pela Amazon em 2018 com o objetivo de democratizar o acesso a técnicas de aprendizado por reforço, permitindo que desenvolvedores treinem e implantem políticas de condução autônoma em um veículo físico em escala reduzida. Desde então, diversos estudos exploraram seu potencial, tanto como ferramenta educacional quanto como ambiente de pesquisa.

Em (NGUYEN; SILVA; PATEL, 2021), por exemplo, o DeepRacer é utilizado em cursos introdutórios de aprendizado por reforço, destacando-se pela facilidade de visualização do processo de aprendizado e pela curva de entrada reduzida. Já trabalhos como (KUMAR; LEE; ALMEIDA, 2022) analisam o impacto de hiperparâmetros e funções de recompensa no desempenho dos agentes, indicando que pequenas modificações podem alterar significativamente o comportamento do veículo.

Outros estudos buscaram estender as capacidades do ambiente original. Em (COMMUNITY, 2024), é apresentado o uso do repositório *DeepRacer-for-Cloud*, que permite a execução local do simulador e a modificação de componentes internos, como o mecanismo de física e o sistema de coleta de métricas. Essa abordagem reduz os custos de treinamento e facilita a experimentação de novos algoritmos e configurações.

Comparações entre algoritmos de aprendizado por reforço também são recorrentes na literatura. Em (LI; TORRES; CHEN, 2023), por exemplo, os autores comparam PPO e SAC em tarefas de controle contínuo, como robôs articulados e locomoção bípede, destacando que o SAC tende a apresentar maior estabilidade e eficiência amostral, enquanto o PPO oferece maior previsibilidade e facilidade de ajuste. Estudos similares, como (MARTINS; PARK; OLIVEIRA, 2023), reforçam a importância da escolha do algoritmo em função do tipo de ambiente e da função de recompensa.

Apesar desses avanços, poucos trabalhos se dedicam a investigar o desempenho de diferentes algoritmos de aprendizado por reforço dentro do ecossistema do DeepRacer, especialmente considerando a execução em ambiente local. Assim, este trabalho contribui ao realizar uma análise controlada do treinamento de modelos PPO e SAC no *DeepRacer-for-Cloud*, explorando o impacto de variações de pista, parâmetros de treinamento e funções de recompensa, sem depender da infraestrutura paga da AWS.

2.2 Fundamentos do Aprendizado por Reforço

O aprendizado por reforço (*Reinforcement Learning – RL*) trata problemas de tomada de decisão sequencial, nos quais um agente aprende a interagir com um ambiente ao longo do tempo por meio de tentativas e erros. A cada passo de tempo, o agente observa o estado do ambiente, seleciona uma ação e recebe uma recompensa numérica que indica a qualidade dessa ação.

O objetivo do agente é aprender uma política que maximize o retorno acumulado esperado ao longo do tempo. Esse tipo de problema é formalmente modelado por meio dos chamados Processos de Decisão de Markov (SUTTON;

BARTO, 2018).

2.2.1 Processos de Decisão de Markov

Um *Processo de Decisão de Markov* (*Markov Decision Process – MDP*) é definido pela quíntupla:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{2-1}$$

onde:

- $\mathcal{S}$ é o conjunto de estados possíveis do ambiente;
- $\mathcal{A}$ é o conjunto de ações disponíveis ao agente;
- $P(s'|s, a)$ representa a dinâmica do ambiente, isto é, a probabilidade de transição para o estado s' após executar a ação a no estado s;
- $R(s, a)$ é a função de recompensa associada à ação a no estado s;
- $\gamma \in [0, 1]$ é o fator de desconto, que controla a importância de recompensas futuras.

A propriedade de Markov assume que o próximo estado depende apenas do estado atual e da ação executada, e não do histórico completo de interações. Essa suposição é amplamente adotada em tarefas de controle e robótica, incluindo condução autônoma em ambientes simulados.

2.2.2 Política e Retorno

A política do agente é representada por $\pi(a|s)$, que define a probabilidade de selecionar a ação a quando o agente se encontra no estado s. O desempenho de uma política é avaliado por meio do retorno acumulado:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \tag{2-2}$$

onde r_t representa a recompensa recebida no instante t. O objetivo do aprendizado por reforço é encontrar uma política π^* que maximize o valor esperado desse retorno.

2.2.3 Funções de Valor: $V(s)$ e $Q(s,a)$

As funções de valor quantificam a qualidade de estados e ações sob uma política π, sendo fundamentais para a maioria dos algoritmos de aprendizado por reforço.

A função de valor de estado $V^{\pi}(s)$ representa o retorno esperado ao iniciar no estado s e seguir a política π a partir desse ponto:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]. \tag{2-3}$$

Já a função de valor de ação $Q^{\pi}(s,a)$ indica o retorno esperado ao executar uma ação a no estado s e, em seguida, seguir a política π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right]. \tag{2-4}$$

Intuitivamente, $V(s)$ indica o quão vantajoso é estar em um estado, enquanto $Q(s,a)$ avalia a qualidade de uma ação específica nesse estado. Ambas as funções estão relacionadas por:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^{\pi}(s,a)\right]. \tag{2-5}$$

2.3 Algoritmos Considerados

O aprendizado por reforço (*Reinforcement Learning – RL*) baseia-se na interação contínua entre agente e ambiente. A cada ação, o agente recebe recompensas ou punições que orientam a adaptação de sua política, cujo objetivo é maximizar o retorno acumulado ao longo do tempo. A combinação de RL com redes neurais profundas originou o *Deep Reinforcement Learning* (Deep RL), capaz de lidar com entradas de alta dimensionalidade, como imagens e dados contínuos. No contexto da AWS DeepRacer, dois algoritmos são explorados: *Proximal Policy Optimization* (PPO) e *Soft Actor-Critic* (SAC).

O PPO é um método *on-policy*, que aprende diretamente dos dados gerados pela política corrente. Sua principal vantagem está na estabilidade de treinamento, ainda que com maior demanda de dados. Já o SAC é um algoritmo *off-policy*, que reaproveita dados coletados por políticas anteriores, sendo mais eficiente em termos de amostragem, embora potencialmente menos estável. A forma como tratam a entropia também difere: o PPO a utiliza como regularizador, enquanto o SAC a maximiza explicitamente em sua função objetivo, incentivando maior exploração. Outra distinção relevante é que o SAC opera apenas em espaços de ação contínuos, favorecendo maior precisão nos

movimentos, enquanto o PPO pode ser configurado em espaços discretos ou contínuos. Estudos recentes, como (PETRYSHYN et al., 2024), reforçam a pertinência da escolha desses algoritmos ao mostrarem seus desempenhos em diferentes cenários de condução autônoma, incluindo o uso de diversos sensores e funções de recompensa.

2.4 Métodos de Gradiente de Política

Tradicionalmente, os algoritmos de aprendizado por reforço buscavam estimar funções de valor, como $Q(s, a)$ ou $V(s)$, a partir das quais uma política ótima poderia ser derivada. No entanto, esse enfoque frequentemente leva a políticas determinísticas e dificulta a aplicação de técnicas baseadas em gradiente quando a política é parametrizada por funções complexas (como redes neurais).

Os métodos de *policy gradient*, introduzidos formalmente por Sutton et al. (2000) (SUTTON et al., 2000), propõem uma abordagem diferente: em vez de aprender diretamente a função de valor, o objetivo é parametrizar a política $\pi_\theta(a|s)$ por meio de parâmetros θ e ajustá-los de forma a maximizar a recompensa acumulada esperada.

A função objetivo pode ser escrita como:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\Big[R\Big], \tag{2-6}$$

onde R representa o retorno total obtido pela política π_θ. O *Policy Gradient Theorem* demonstra que o gradiente dessa função objetivo pode ser expresso como:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s)\, Q^{\pi_\theta}(s, a)\Big]. \tag{2-7}$$

Essa equação tem uma interpretação intuitiva:

- O termo $\nabla_\theta \log \pi_\theta(a|s)$ mede como a probabilidade de uma ação a em um estado s muda em relação aos parâmetros θ;
- O termo $Q^{\pi_\theta}(s, a)$ avalia a qualidade da ação escolhida naquele estado, isto é, o retorno esperado a partir dela.

Assim, quando uma ação leva a bons resultados (alto Q), a probabilidade de escolhê-la aumenta. Se, por outro lado, o resultado for ruim (baixo Q), a probabilidade diminui. Esse mecanismo ajusta gradualmente os parâmetros da política em direção a melhores decisões.

Entretanto, embora elegantes do ponto de vista teórico, os métodos de gradiente de política apresentam desafios práticos, em especial a alta variância

de suas estimativas. É nesse contexto que surge a família de algoritmos conhecidos como Actor-Critic, que combina as vantagens das abordagens baseadas em política e em valor (KONDA; TSITSIKLIS, 2000).

2.4.1 Métodos Actor-Critic

Um desafio prático dos métodos de gradiente de política puros é a alta variância nas estimativas de gradiente. Isso dificulta a convergência e torna o aprendizado ineficiente em ambientes complexos.

Uma solução proposta é a introdução da arquitetura *Actor-Critic*, que combina elementos baseados em política e em valor (KONDA; TSITSIKLIS, 2000).

- O Ator (Actor) é responsável por parametrizar e atualizar a política $\pi_\theta(a|s)$. Ele escolhe as ações a serem executadas no ambiente, ajustando os parâmetros θ de forma a aumentar a probabilidade de ações que levam a maiores recompensas.
- O Crítico (Critic) é responsável por estimar a função de valor $V^\pi(s)$ ou a função de vantagem $A^\pi(s,a)$. Seu papel é fornecer uma avaliação de quão boa foi a ação escolhida pelo ator, em comparação com a média esperada de recompensas no estado atual.

Matematicamente, o gradiente de política com baseline pode ser reescrito como:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s)\,(Q^\pi(s,a) - V^\pi(s))\Big]. \tag{2-8}$$

O termo $Q^\pi(s,a) - V^\pi(s)$ corresponde à vantagem $A^\pi(s,a)$, isto é, o quanto uma ação foi melhor (ou pior) que a média esperada naquele estado:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{2-9}$$

Essa decomposição reduz a variância do gradiente sem introduzir viés, tornando o aprendizado mais estável.

2.4.1.0 Interação entre Ator e Crítico

A dinâmica entre ator e crítico pode ser resumida da seguinte forma:

1. O ator seleciona uma ação a em um estado s de acordo com $\pi_\theta(a|s)$.
2. O ambiente retorna a recompensa r e o próximo estado s'.

3. O crítico avalia essa transição, atualizando sua estimativa de $V(s)$ para se aproximar do retorno esperado.

4. O ator ajusta seus parâmetros θ usando o sinal de vantagem $A(s, a)$, fornecido pelo crítico, reforçando ou desincentivando a ação tomada.

Assim, o ator aprende *como agir*, enquanto o crítico aprende *como avaliar*. O crítico funciona como um guia para o ator, reduzindo a variância e acelerando a convergência (KONDA; TSITSIKLIS, 2000).

2.4.1.0
Uso de Redes Neurais

No contexto de aprendizado por reforço profundo, tanto o ator quanto o crítico são implementados por redes neurais:

– A rede do ator recebe como entrada o estado s e retorna uma distribuição de probabilidade sobre as ações $\pi_\theta(a|s)$ (no caso discreto) ou os parâmetros de uma distribuição contínua de ações (no caso contínuo).
– A rede do crítico recebe o estado s como entrada e retorna uma estimativa escalar $V(s)$, representando o valor esperado daquele estado sob a política atual.

Essa divisão de responsabilidades permite que cada rede se especialize: o ator em explorar e melhorar políticas, e o crítico em fornecer feedback confiável sobre a qualidade das decisões tomadas.

Apesar dessa melhora em relação ao REINFORCE puro, ainda há limitações quanto à estabilidade e eficiência das atualizações de política. Para lidar com esses desafios, surgiram algoritmos mais avançados, entre os quais se destaca o Proximal Policy Optimization (PPO) (SCHULMAN et al., 2017), considerado atualmente um dos métodos mais eficazes e amplamente utilizados em aprendizado por reforço profundo.

2.5
PPO

O *Proximal Policy Optimization (PPO)* é uma família de algoritmos de gradiente de política para aprendizado por reforço profundo, projetada para ser ao mesmo tempo eficiente, estável e relativamente simples de implementar (SCHULMAN et al., 2017).

O PPO segue a abordagem típica dos métodos de *policy gradient*, alternando entre duas etapas fundamentais:

1. Coleta de experiências por meio da interação do agente com o ambiente;
2. Otimização da função objetivo através de algoritmos de ascensão de gradiente estocástico (OPENAI, 2018).

O PPO foi inspirado diretamente no *Trust Region Policy Optimization (TRPO)*, herdando muitas de suas vantagens, como maior estabilidade no treinamento e políticas mais consistentes, mas com uma formulação muito mais simples e prática. Enquanto o TRPO impõe restrições complexas de segunda ordem para limitar o tamanho da atualização da política (o que dificulta a implementação e exige alto custo computacional), o PPO introduz um mecanismo mais acessível: uma penalização ou *clipping* na função objetivo, que garante que a nova política não se afaste demais da política anterior (SCHULMAN et al., 2017).

Esse aspecto é um dos diferenciais centrais do PPO em relação a outros algoritmos de gradiente de política, pois mantém as atualizações "proximais" (daí o nome), evitando que mudanças bruscas na política provoquem colapsos de desempenho. Assim, o algoritmo consegue melhorar a eficiência da exploração e, ao mesmo tempo, preservar a estabilidade durante o treinamento (OPENAI, 2018).

Nos experimentos originais apresentados no artigo de Schulman et al. (2017), o PPO foi testado em diferentes benchmarks, incluindo tarefas de locomoção robótica em simulação e jogos clássicos da plataforma Atari. Nessas avaliações, o algoritmo demonstrou desempenho superior ou equivalente a outros métodos da mesma família, como *REINFORCE*, *Actor-Critic* tradicional, *A2C/A3C* e até o próprio TRPO, consolidando-se como um dos algoritmos mais utilizados em aplicações práticas de aprendizado por reforço profundo (SCHULMAN et al., 2017).

A função objetivo principal proposta pelo PPO é a seguinte:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\Big(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_t\Big)\Big], \tag{2-10}$$

onde ϵ é um hiperparâmetro (tipicamente $\epsilon = 0.2$).

A motivação para esta função é a seguinte: - O primeiro termo dentro do operador min corresponde a L^{CPI}. - O segundo termo, $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$, modifica a função objetivo substituta (*surrogate objective*) por meio de um *clipping* da razão de probabilidades, o que elimina o incentivo de mover r_t para fora do intervalo $[1-\epsilon, 1+\epsilon]$. - Ao tomar o mínimo entre o objetivo original e o objetivo com *clipping*, a função final representa um *lower bound* (um limite inferior, ou seja, uma estimativa pessimista) da função não-*clipped*.

Esse esquema garante que as mudanças na razão de probabilidades são ignoradas apenas quando elas melhorariam artificialmente a função objetivo; em contrapartida, quando a mudança poderia piorar o desempenho, a penalização é aplicada.

Vale observar que $L^{CLIP}(\theta) = L^{CPI}(\theta)$ na primeira ordem em torno de θ_{old} (isto é, quando $r_t(\theta) \approx 1$). Entretanto, à medida que θ se afasta de θ_{old}, as duas funções passam a divergir.

2.6 Soft Actor-Critic (SAC)

O algoritmo *Soft Actor-Critic* (SAC)(HAARNOJA et al., 2018) é um método de aprendizado por reforço profundo do tipo *off-policy actor-critic*, formulado no contexto do aprendizado por reforço com entropia máxima. Diferentemente dos algoritmos tradicionais, que buscam apenas maximizar a recompensa esperada, o SAC introduz explicitamente um termo de entropia na função objetivo. Com isso, o agente não só procura realizar a tarefa com sucesso, mas também mantém suas ações suficientemente estocásticas, o que promove exploração. Essa abordagem tende a produzir agentes mais estáveis e com desempenho consistente, mesmo sob diferentes inicializações aleatórias (*random seeds*).

Um dos principais problemas que o SAC procura mitigar é a baixa eficiência amostral dos algoritmos *on-policy*. Nesse caso, cada atualização de gradiente exige novas amostras, tornando o processo de treinamento caro e, em tarefas complexas, inviável. Por ser *off-policy*, o SAC reutiliza amostras passadas por meio de um *replay buffer*, o que melhora substancialmente a eficiência amostral. No entanto, combinar aprendizado fora da política com aproximações de funções não lineares de alta dimensão — como redes neurais profundas — introduz desafios de estabilidade e convergência. O SAC endereça esse problema utilizando técnicas de regularização baseadas em entropia, além de mecanismos que ajustam automaticamente o parâmetro de temperatura, equilibrando a busca por novas estratégias (exploração) e o aproveitamento das estratégias já conhecidas com bom desempenho (exploitation).

2.6.1 Função Objetivo

A formulação matemática do SAC é inspirada no *framework* de entropia máxima, em que a política ótima deve maximizar não apenas o retorno esperado, mas também a entropia da distribuição de ações. A função objetivo é definida como:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho^\pi} \left[r(s_t, a_t) + \alpha\, \mathcal{H}(\pi(\cdot|s_t)) \right], \tag{2-11}$$

onde s_t representa o estado no tempo t, a_t a ação escolhida, $r(s_t, a_t)$ a recompensa recebida, e ρ^π a distribuição induzida pela política π sobre as trajetórias. O termo $\mathcal{H}(\pi(\cdot|s_t))$ corresponde à entropia da política no estado s_t, e o parâmetro α controla a importância relativa da entropia em relação à recompensa.

O papel de α, chamado *temperatura*, é central para o algoritmo: valores maiores de α incentivam políticas mais estocásticas, favorecendo exploração, enquanto valores menores aproximam o objetivo do SAC do aprendizado por reforço convencional. No limite $\alpha \to 0$, a função objetivo se reduz ao caso tradicional de maximização do retorno esperado.

2.6.2 Arquitetura do Algoritmo

O SAC utiliza a estrutura *actor-critic*, composta por uma política estocástica (*actor*) e funções de valor de ação (*critics*). Os principais componentes são:

- Política estocástica (π_θ): a política é parametrizada por uma distribuição Gaussiana, cuja média e variância são produzidas por uma rede neural. A ação a_t é amostrada de

  $$a_t \sim \pi_\theta(\cdot|s_t), \tag{2-12}$$

  permitindo que o agente mantenha estocasticidade e otimize o termo de entropia da função objetivo.
- Duas redes críticas (Q_{ϕ_1} e Q_{ϕ_2}): semelhantes ao *Twin Delayed DDPG* (TD3), o SAC utiliza duas aproximações da função-valor de ação para reduzir o viés de superestimação comum em algoritmos baseados em Q-learning profundo. A função de perda para cada crítico é definida como:

  $$J_Q(\phi_i) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \Big(Q_{\phi_i}(s_t, a_t) - \hat{Q}(s_t, a_t) \Big)^2 \right], \tag{2-13}$$

  onde $\mathcal{D}$ é o *replay buffer* e $\hat{Q}$ é a estimativa da função de valor alvo.
- Rede de valor de estado (opcional): em versões iniciais do SAC, também era empregada uma rede de valor $V(s)$, mas em versões mais recentes essa rede foi eliminada, simplificando o algoritmo sem perda de desempenho.

2.6.3 Ajuste Automático da Temperatura

O parâmetro α pode ser tratado como um hiperparâmetro fixo, mas uma das contribuições do SAC é o ajuste automático da temperatura. O objetivo é garantir que a entropia média da política permaneça próxima a um valor alvo $\mathcal{H}_{\text{target}}$. A função de perda associada ao ajuste de α é:

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_\theta} \left[-\alpha \, \log \pi_\theta(a_t|s_t) - \alpha \, \mathcal{H}_{\text{target}} \right]. \tag{2-14}$$

Esse mecanismo permite que o algoritmo se adapte dinamicamente, equilibrando exploração e *exploitation* de forma mais robusta em diferentes tarefas, reduzindo a necessidade de ajuste manual de hiperparâmetros.

3
Objetivos do trabalho

3.1
Objetivo Geral

O objetivo geral deste trabalho é compreender e comparar o desempenho de diferentes configurações de modelos de aprendizado por reforço profundo aplicados ao contexto do *AWS DeepRacer*. Busca-se analisar como variações em aspectos como o algoritmo de treinamento — em especial o *Proximal Policy Optimization* (PPO) e o *Soft Actor-Critic* (SAC) —, a função de recompensa, o espaço de ação e o ambiente de simulação influenciam o processo de aprendizado e a capacidade de generalização dos agentes. Além disso, o trabalho tem como propósito explorar alternativas de execução e treinamento que reduzam os custos associados ao uso da plataforma da AWS, por meio da utilização do *DeepRacer for Cloud* (DRfC), ferramenta que permite a realização de treinamentos de forma local, sem necessidade de infraestrutura em nuvem.

3.2
Objetivos Específicos

De forma mais detalhada, este trabalho tem como objetivo explicar o funcionamento dos algoritmos PPO e SAC, destacando suas diferenças conceituais e operacionais, bem como suas implicações no desempenho de agentes de aprendizado por reforço profundo. Também busca apresentar o funcionamento do *AWS DeepRacer*, incluindo sua arquitetura, os custos envolvidos nos treinamentos e as principais possibilidades de configuração do modelo, como a escolha da função de recompensa, o ajuste de hiperparâmetros e a seleção da pista de simulação. Em complemento, é abordado o *DeepRacer for Cloud* (DRfC), demonstrando como ele reproduz localmente o ambiente de treinamento da AWS, além de detalhar o processo de instalação, configuração e execução de experimentos em ambiente local.

Por meio de experimentos realizados em diferentes contextos, o trabalho explora variações de fatores de treinamento, como algoritmo, função de recompensa, tipo de pista, espaço de ação e alternância de direção, de modo a observar como cada combinação impacta métricas de desempenho como o tempo de conclusão, o percentual médio de completude e a média de recompensa. A partir desses resultados, busca-se comparar o comportamento dos algoritmos

PPO e SAC, identificando vantagens, limitações e padrões de aprendizado sob diferentes condições de treino.

Por fim, pretende-se discutir os resultados obtidos sob a perspectiva da eficiência de aprendizado, do custo computacional e do tempo de treinamento, destacando as vantagens e limitações do uso do DRfC em relação ao ambiente da AWS. Dessa forma, este trabalho contribui para a compreensão do processo de treinamento de agentes autônomos em ambientes de simulação, oferecendo uma análise integrada que combina aspectos teóricos, práticos e de infraestrutura computacional.

4
Ambiente da AWS DeepRacer

4.1
Visão Geral da Plataforma

O AWS DeepRacer é uma plataforma desenvolvida para permitir que desenvolvedores de diferentes níveis de habilidade tenham uma experiência prática com *machine learning*, especificamente com aprendizado por reforço. A plataforma combina um simulador de corridas 3D baseado na nuvem com um carro autônomo em escala 1:18, possibilitando o treinamento e a avaliação de modelos de controle autônomo em um ambiente seguro e controlado (SERVICES, 2025a).

Através do AWS DeepRacer, os usuários podem aprender conceitos fundamentais e técnicas de *machine learning*, incluindo definição de funções de recompensa, ajuste de hiperparâmetros e estratégias de exploração de políticas de controle. A plataforma permite que os modelos treinados no simulador sejam transferidos para o veículo físico, proporcionando uma experiência completa de desenvolvimento, teste e validação de carros autônomos em escala reduzida. Além disso, todos os recursos necessários para a construção e operação do carro — como sensores e demais equipamentos — estão centralizados, possibilitando que os usuários configurem o kit de forma integrada e façam a transição do ambiente virtual para o físico de maneira mais eficiente.

O AWS DeepRacer Console oferece uma interface gráfica completa para interação com os serviços da plataforma. Por meio do console, os usuários podem criar tarefas de treinamento definindo funções de recompensa, algoritmos de otimização, ambiente de simulação e hiperparâmetros, selecionar pistas simuladas para avaliação utilizando a infraestrutura de IA do Amazon SageMaker, clonar modelos treinados para ajustes finos visando otimizar o desempenho e, finalmente, baixar os modelos para implantação em veículos físicos. Dessa forma, o console centraliza o controle do processo de treinamento e avaliação, tornando a experimentação mais prática e acessível (SERVICES, 2025d).

Com essa integração entre simulador, console e veículo físico, o AWS DeepRacer fornece um ambiente completo para experimentação, análise de desempenho e aplicação de conceitos de aprendizado por reforço, permitindo que desenvolvedores adquiram experiência prática de forma estruturada e segura.

4.2 Ambiente de Simulação e Infraestrutura

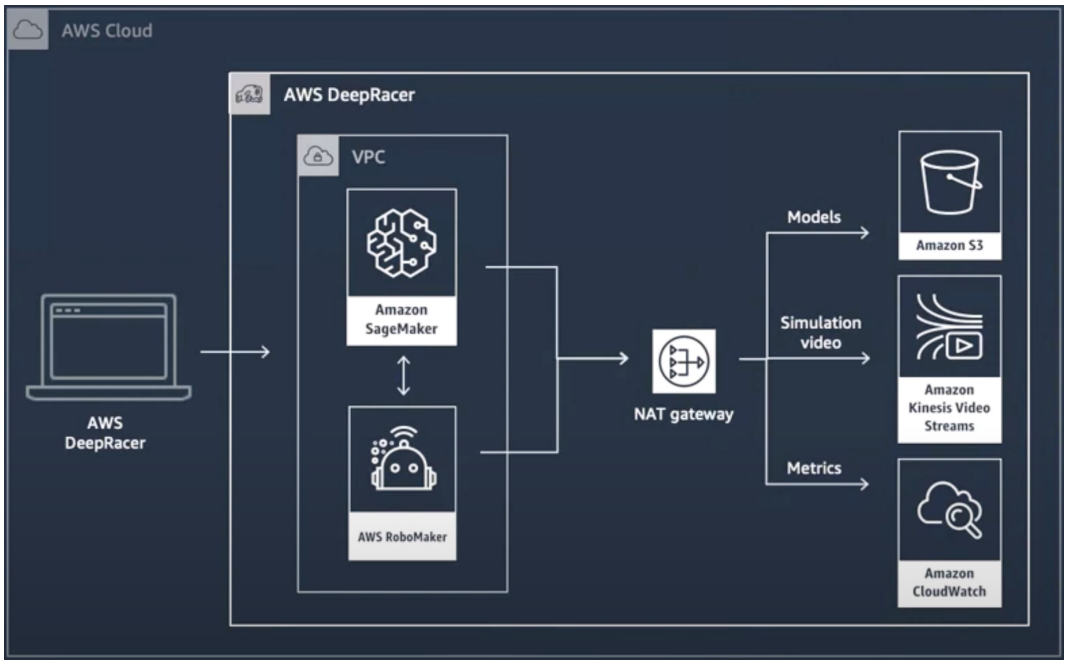

Figura 4.1: Arquitetura do simulador AWS DeepRacer (SERVICES, 2020)

O *AWS DeepRacer* pode ser compreendido como um serviço que integra e coordena diversos componentes da infraestrutura da Amazon Web Services. Em sua arquitetura interna, o *Amazon SageMaker* é responsável pelo treinamento dos modelos de *Reinforcement Learning* (RL), enquanto o *AWS RoboMaker* provê o ambiente de simulação necessário para a geração de experiências de interação do agente. Os artefatos resultantes do treinamento, como modelos e métricas, são armazenados no *Amazon S3*, e os registros de execução (*logs*) são coletados e gerenciados pelo *Amazon CloudWatch*. Adicionalmente, o *Amazon Kinesis Video Streams* é utilizado para a transmissão do vídeo da simulação exibido no console do DeepRacer (SERVICES, 2020).

Quando um treinamento é iniciado, o serviço *AWS DeepRacer* instancia um contêiner do *SageMaker* e outro do *RoboMaker*. Esses contêineres são então interconectados e configurados com os parâmetros necessários para iniciar o processo de aprendizado. Durante a simulação, o *RoboMaker* gera as experiências de aprendizado — compostas por tuplas de estado, ação, novo estado e recompensa — que, após determinado volume de dados coletados, são enviadas ao *SageMaker* para atualização do modelo. O modelo atualizado é posteriormente reenviado ao *RoboMaker*, dando continuidade ao ciclo iterativo de treinamento.

As saídas do processo, incluindo os modelos treinados, registros de vídeo e métricas de desempenho, são armazenadas e disponibilizadas por meio dos

serviços da AWS associados à conta do usuário, garantindo a integração completa entre as etapas de simulação, treinamento e monitoramento. Essa arquitetura pode ser visualizada na Figura 4.1.

4.3 Configuração dos Modelos

O processo de criação de um modelo no AWS DeepRacer inicia-se na página principal do serviço, onde o usuário pode selecionar *Create model* ou *Get started with reinforcement learning*. Nessa etapa, define-se o nome do modelo a ser treinado e, em seguida, escolhe-se a pista de simulação, incluindo o sentido de percurso (horário ou anti-horário). Posteriormente, é selecionado o tipo de corrida, podendo ser *Time trial*, em que o objetivo é completar a pista no menor tempo possível; *Object avoidance*, cujo foco está em detectar e desviar de obstáculos fixos; ou ainda *Head-to-bot*, modalidade em que o modelo deve competir com agentes autônomos (*bots*) enquanto evita colisões e saídas de pista. Na sequência, define-se o algoritmo de aprendizado por reforço a ser utilizado — as opções disponíveis no console são *Soft Actor-Critic* (SAC) e *Proximal Policy Optimization* (PPO). O primeiro opera em um espaço de ações contínuo, enquanto o segundo permite tanto espaços discretos quanto contínuos. Após a escolha do algoritmo, procede-se à configuração dos hiperparâmetros de treinamento, que variam conforme o método selecionado. Em seguida, especifica-se o espaço de ações, determinando o conjunto de comandos possíveis que o agente pode executar durante o aprendizado. Por fim, é implementada a função de recompensa, responsável por orientar o comportamento do modelo ao atribuir valores de incentivo a cada ação executada no ambiente de simulação (SERVICES, 2024).

4.3.1 Escolha da pista

A plataforma AWS DeepRacer disponibiliza diversos modelos de pistas (*track templates*) que podem ser utilizados durante o processo de treinamento dos agentes de aprendizado por reforço. Essas pistas são classificadas em três níveis de complexidade — básico, intermediário e avançado — conforme o grau de dificuldade da pista, a presença de curvas fechadas e a extensão total do percurso. Exemplos representativos de cada categoria são ilustrados nas Figuras 4.2, 4.3 e 4.4.

A seleção adequada da pista é um fator determinante para o sucesso do treinamento, uma vez que diferentes configurações podem influenciar diretamente o tempo de convergência do modelo e o tipo de comportamento

aprendido pelo agente. Pistas mais simples favorecem o aprendizado inicial e a estabilização de políticas básicas de navegação, enquanto pistas intermediárias e avançadas são mais indicadas para refinar estratégias de controle, curvas em alta velocidade e detecção de limites de pista. Assim, a escolha deve ser orientada pelos objetivos específicos de cada experimento e pelo estágio de maturidade do modelo em treinamento.

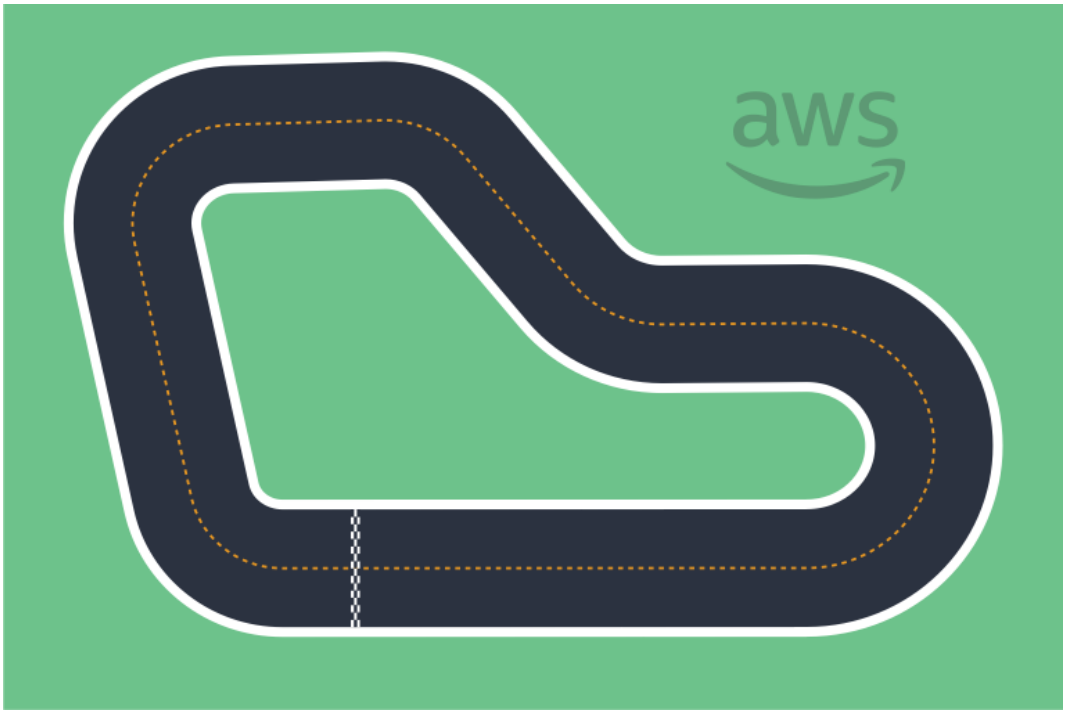

Figura 4.2: AWS DeepRacer A to Z Speedway (Basic) track template(SERVICES, 2023c)

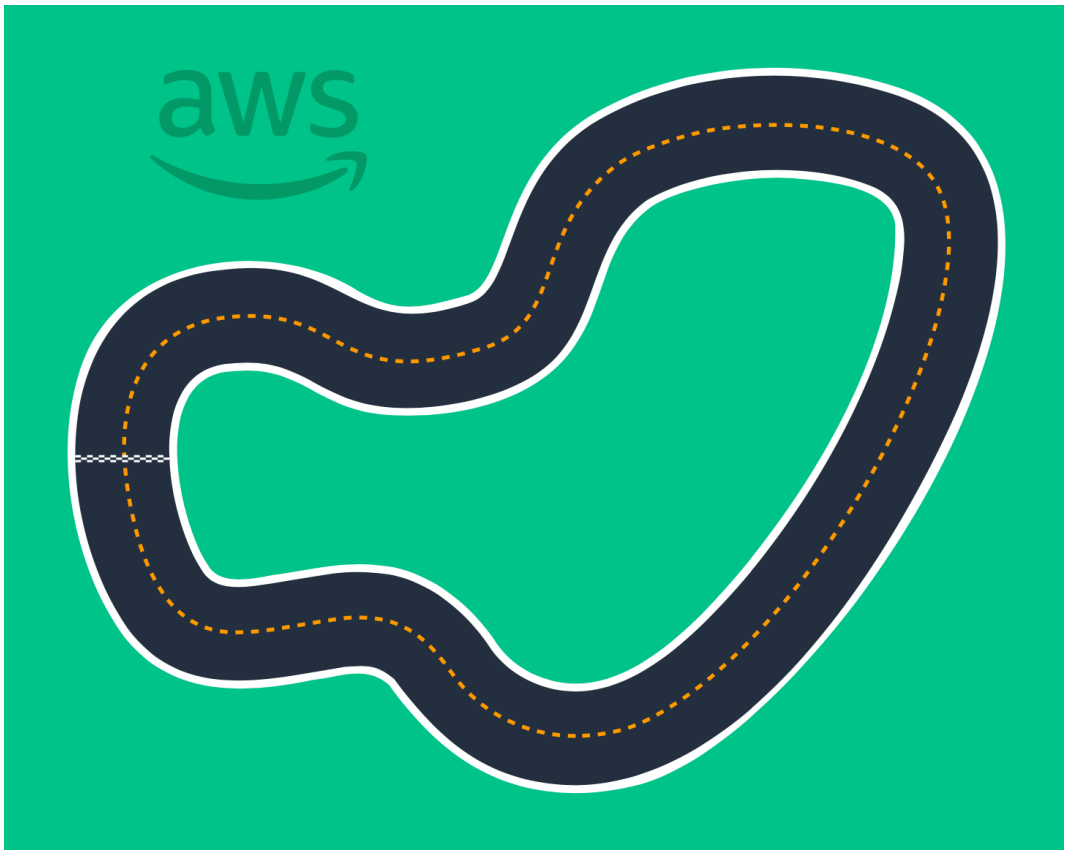

Figura 4.3: AWS DeepRacer Smile Speedway (Intermediate) track template (SERVICES, 2023c)

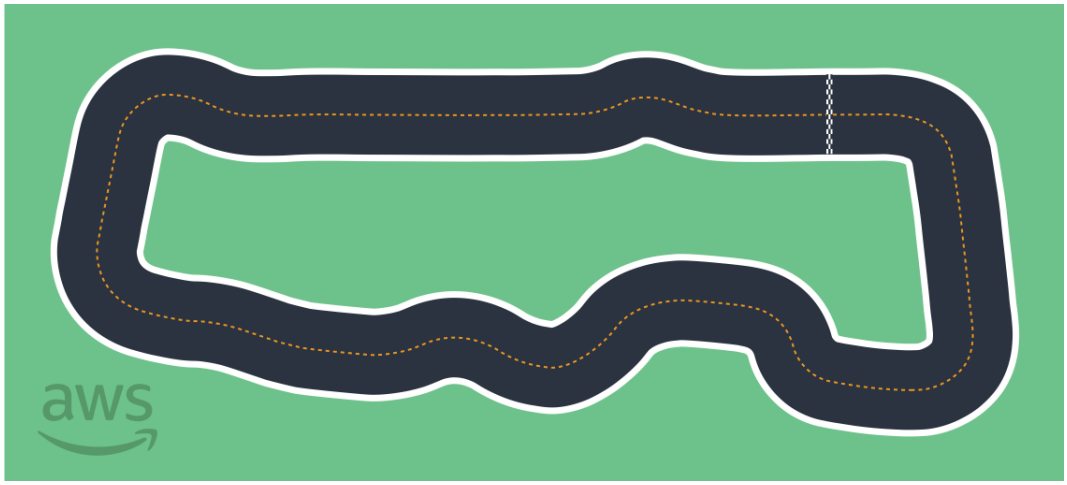

Figura 4.4: AWS DeepRacer RL Speedway (Advanced) track template (SERVICES, 2023c)

4.3.2 Tipo de corrida e sensores

O conteúdo a seguir é praticamente uma transcrição da documentação oficial do AWS DeepRacer (SERVICES, 2023g), apresentando os sensores

disponíveis e suas respectivas aplicações. Na AWS DeepRacer, diversos sensores podem ser utilizados durante o treinamento dos modelos de aprendizado por reforço. É importante destacar que a escolha do sensor deve estar alinhada com o tipo de corrida a ser realizada, uma vez que diferentes tarefas podem se beneficiar de maneira distinta das informações fornecidas por cada sensor. O sensor padrão do veículo é a *Câmera monocular frontal*, mas também é possível utilizar dois sensores para configuração de *Câmera estéreo frontal* ou optar pelo sensor LiDAR.

A Câmera monocular frontal é uma câmera frontal que captura imagens do ambiente à frente do veículo, incluindo a borda e o formato da pista. Este sensor apresenta baixo custo e é adequado para tarefas básicas de direção autônoma, como percursos simples e bem demarcados. Quando bem treinado, o modelo pode até evitar obstáculos, mas há maior propensão ao *overfitting*, e o desempenho pode ser comprometido caso obstáculos estáticos ou veículos em movimento sejam posicionados aleatoriamente.

A Câmera estéreo frontal possui duas lentes que capturam imagens simultaneamente na mesma resolução e frequência, permitindo estimar a profundidade dos objetos observados. Esta informação adicional é valiosa para evitar colisões com obstáculos ou outros veículos, embora a maior complexidade do sensor aumente o tempo necessário para que os modelos atinjam a convergência.

O LiDAR sensor emite pulsos de laser fora do espectro visível de luz e calcula o tempo de retorno de cada pulso, gerando um mapa tridimensional detalhado do ambiente em torno do sensor. Essa tecnologia auxilia na detecção precisa de objetos, incluindo veículos próximos, e pode ser combinada com câmeras estéreo para fornecer informações suficientes para decisões complexas, como mudança de pista e prevenção de colisões.

De maneira resumida, as combinações de sensores e suas aplicações mais indicadas são:

- Câmera monocular frontal: adequada para corridas de tempo (*time trials*) e para obstáculos fixos.
- Câmera estéreo frontal: indicada para desvio de obstáculos em posições fixas ou aleatórias.
- Câmera monocular frontal com LiDAR: recomendada para desvio de obstáculos ou corridas contra bots.
- Câmera estéreo frontal com LiDAR: adequada para desvio de obstáculos ou corridas contra bots, embora não seja a configuração mais econômica para corridas de tempo.

4.3.3 Seleção do Algoritmo e Definição dos Hiperparâmetros

Tabela 4.1: Proximal Policy Optimization (PPO) versus Soft Actor-Critic (SAC) (SERVICES, 2023d)

PPO	**SAC**
Funciona em espaços de ação discretos e contínuos	Funciona em espaço de ação contínuo
On-policy	Off-policy
Utiliza regularização por entropia	Adiciona entropia ao objetivo de maximização

A escolha do algoritmo de aprendizado por reforço é um passo fundamental no treinamento de modelos para o AWS DeepRacer, pois influencia diretamente a dinâmica de aprendizado e os hiperparâmetros que podem ser ajustados. No console da plataforma, os usuários podem optar por Proximal Policy Optimization (PPO) ou Soft Actor-Critic (SAC), cada um apresentando características distintas conforme resumido na Tabela 4.1. O PPO pode ser treinado em espaços de ação discretos ou contínuos, opera sob a política atual (*on-policy*) e utiliza regularização por entropia, enquanto o SAC funciona apenas em espaços de ação contínuos, é fora da política (*off-policy*) e adiciona entropia ao objetivo de maximização.

Cada algoritmo possui conjuntos específicos de hiperparâmetros que impactam diretamente o desempenho e a convergência do modelo. No caso do PPO, parâmetros como taxa de aprendizado (*learning rate*), tamanho do lote de gradiente (*gradient descent batch size*), número de épocas (*number of epochs*), entropia (*entropy*) e tipo de perda (*loss type*) são críticos para garantir estabilidade e eficiência no treinamento. Já o SAC requer atenção especial à taxa de aprendizado do ator e do crítico, número de episódios de experiência entre cada atualização da política, fator de desconto (γ) e coeficiente de entropia, refletindo sua natureza *off-policy* e contínua (SERVICES, 2023f).

Além desses parâmetros específicos, existem hiperparâmetros comuns a ambos os algoritmos, como tamanho do lote, taxa de aprendizado, entropia e fator de desconto, que devem ser ajustados de acordo com a complexidade da pista, a estratégia de exploração desejada e a velocidade de convergência esperada. A escolha adequada desses parâmetros é essencial para que o modelo aprenda de forma eficiente, alcance o desempenho esperado e minimize riscos de instabilidade ou *overfitting* durante o treinamento.

4.3.4 Espaço de ações

No contexto do aprendizado por reforço, o *espaço de ação* representa o conjunto de todas as ações possíveis que um agente pode executar enquanto interage com o ambiente. No caso do AWS DeepRacer, esse espaço define as combinações de velocidade e ângulo de direção que o veículo autônomo pode adotar durante o treinamento e a simulação. A forma como o espaço de ação é configurado influencia diretamente a capacidade do modelo de aprender políticas eficazes e de generalizar para diferentes tipos de pistas e situações.

A plataforma AWS DeepRacer permite configurar o treinamento dos agentes em dois tipos de espaços de ação: discreto e contínuo, cada um com suas vantagens, limitações e aplicações específicas.

No espaço de ação discreto, o conjunto de ações possíveis é finito e pré-definido. Isso significa que, para cada situação observada pelo agente, a rede neural do modelo seleciona uma ação a partir de uma lista limitada de combinações de ângulo de direção e velocidade. Por exemplo, ao se aproximar de uma curva, o agente pode escolher entre ações como acelerar, frear, virar à esquerda, virar à direita ou seguir em frente. Cada ação é representada por uma combinação específica de ângulo e velocidade, conforme mostrado na Tabela 4.2, baseada na configuração padrão do AWS DeepRacer.

Tabela 4.2: Espaço de ação discreto padrão do AWS DeepRacer (SERVICES, 2023a)

Número da ação	**Ângulo de direção (°)**	**Velocidade (m/s)**
0	-30	0.4
1	-30	0.8
2	-15	0.4
3	-15	0.8
4	0	0.4
5	0	0.8
6	15	0.4
7	15	0.8
8	30	0.4
9	30	0.8

Nesse modo, o número total de ações possíveis é limitado, e cada uma corresponde a um par fixo de ângulo e velocidade. Essa abordagem simplifica o processo de aprendizado, pois reduz o número de decisões que o agente precisa explorar. No entanto, ela também restringe a suavidade dos movimentos e pode dificultar a obtenção de trajetórias otimizadas, especialmente em curvas mais complexas.

Por outro lado, no espaço de ação contínuo, o agente não escolhe entre ações discretas, mas sim valores contínuos dentro de um intervalo definido para cada parâmetro de ação. Assim, tanto o ângulo de direção quanto a velocidade podem variar continuamente dentro de faixas numéricas configuradas pelo usuário. Por exemplo, é possível permitir que o veículo selecione velocidades entre 0,75 m/s e 4,0 m/s e ângulos de direção entre -20° e 20°. Essa configuração proporciona maior flexibilidade e permite gerar comportamentos mais suaves e realistas, especialmente em situações que exigem ajustes finos na direção e na aceleração.

A principal diferença entre os dois tipos de configuração está no equilíbrio entre simplicidade e precisão. O espaço contínuo oferece potencial para movimentos mais fluidos e trajetórias otimizadas, mas demanda maior tempo de treinamento e estabilidade do algoritmo, já que o agente precisa aprender a escolher valores ideais dentro de um intervalo infinito de possibilidades. Já o espaço discreto acelera o aprendizado inicial e reduz a complexidade computacional, mas limita o desempenho máximo do modelo em ambientes dinâmicos e pistas complexas.

A configuração do *action space* é realizada durante a criação do modelo no console do AWS DeepRacer. O usuário pode escolher entre o modo discreto ou contínuo e, em seguida, definir os parâmetros correspondentes. No modo discreto, são especificados o número de ações, os ângulos de direção e as velocidades associadas a cada uma delas. Já no modo contínuo, devem ser definidos os valores mínimos e máximos de velocidade e ângulo de direção, delimitando a faixa de opções a partir da qual o agente selecionará suas ações.

Alterações no espaço de ação podem ser realizadas antes do início do treinamento ou ao clonar um modelo existente, permitindo ajustar a granularidade das ações de acordo com a complexidade da pista, o tipo de corrida e o comportamento desejado do agente.

4.3.5 Função de Recompensa

A definição da função de recompensa constitui uma etapa crucial no treinamento de modelos de aprendizado por reforço, uma vez que ela determina se as ações executadas pelo agente são desejáveis em relação aos objetivos da tarefa. A função de recompensa fornece feedback contínuo ao agente, orientando o aprendizado e influenciando diretamente a convergência e o desempenho do modelo.

A seguir, apresenta-se um exemplo de função de recompensa cujo objetivo é manter o veículo próximo ao centro da linha de referência no meio da pista,

incentivando uma condução estável e precisa:

Código 1: Função de Recompensa: Manter-se no Centro da Pista (SERVICES, 2023b)

```
def reward_function(params):
    '''
    Example of rewarding the agent to follow center line
    '''

    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Calculate 3 markers that are increasingly further away from
        the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and
        vice versa
    if distance_from_center <= marker_1:
        reward = 1
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    return reward
```

Para construir uma função de recompensa eficaz, é fundamental compreender os parâmetros de entrada disponíveis, conforme documentado pela AWS (SERVICES, 2023e). Estes parâmetros fornecem informações detalhadas sobre o estado do veículo, a pista e o ambiente de simulação, permitindo que a função de recompensa seja formulada de maneira a orientar o comportamento do agente de forma eficiente. Um exemplo de conjunto de parâmetros de entrada é apresentado a seguir:

Código 2: Parâmetros de Entrada da Função de Recompensa

```
{
    "all_wheels_on_track": Boolean,        # flag to indicate if the
        agent is on the track
    "x": float,                            # agent's x-coordinate in
        meters
    "y": float,                            # agent's y-coordinate in
        meters
```

```
    "closest_objects": [int, int],          # zero-based indices of
        the two closest objects to the agent's current position of (
        x, y).
    "closest_waypoints": [int, int],        # indices of the two
        nearest waypoints.
    "distance_from_center": float,          # distance in meters from
         the track center
    "is_crashed": Boolean,                  # Boolean flag to
        indicate whether the agent has crashed.
    "is_left_of_center": Boolean,           # Flag to indicate if the
         agent is on the left side to the track center or not.
    "is_offtrack": Boolean,                 # Boolean flag to
        indicate whether the agent has gone off track.
    "is_reversed": Boolean,                 # flag to indicate if the
         agent is driving clockwise (True) or counter clockwise (
        False).
    "heading": float,                       # agent's yaw in degrees
    "objects_distance": [float, ],          # list of the objects'
        distances in meters between 0 and track_length in relation
        to the starting line.
    "objects_heading": [float, ],           # list of the objects'
        headings in degrees between -180 and 180.
    "objects_left_of_center": [Boolean, ], # list of Boolean flags
        indicating whether elements' objects are left of the center
        (True) or not (False).
    "objects_location": [(float, float),], # list of object
        locations [(x,y), ...].
    "objects_speed": [float, ],             # list of the objects'
        speeds in meters per second.
    "progress": float,                      # percentage of track
        completed
    "speed": float,                         # agent's speed in meters
         per second (m/s)
    "steering_angle": float,                # agent's steering angle
        in degrees
    "steps": int,                           # number steps completed
    "track_length": float,                  # track length in meters.
    "track_width": float,                   # width of the track
    "waypoints": [(float, float), ]         # list of (x,y) as
        milestones along the track center

}
```

O entendimento adequado desses parâmetros é essencial para a criação de funções de recompensa robustas e adaptadas a diferentes pistas e cenários, permitindo que o modelo aprenda políticas de navegação mais eficazes e confiáveis.

4.4 Custos de Treinamento e Alternativas

Após compreender o processo de configuração e treinamento dos modelos no ambiente da AWS DeepRacer, é fundamental analisar os custos associados ao uso dessa plataforma, bem como os planos disponíveis e eventuais alternativas para execução dos treinamentos.

Tabela 4.3: Definição de preço do serviço

Serviço AWS DeepRacer	**Preço por unidade**
Treinamento	3,50 USD por hora
Avaliação	3,50 USD por hora
Armazenamento do modelo	0,023 USD por GB por mês

De acordo com a página oficial da AWS DeepRacer (SERVICES, 2025c), o custo por hora de treinamento ou avaliação de modelos é de 3,50 USD, enquanto o armazenamento dos modelos possui um custo adicional de 0,023 USD por gigabyte ao mês, conforme apresentado na Tabela 4.3. Esses valores refletem o custo padrão do serviço sob demanda, sem considerar descontos ou créditos promocionais.

A AWS oferece um plano gratuito inicial, que concede ao usuário 10 horas de treinamento e até 5 GB de armazenamento durante o primeiro mês de uso. Contudo, no modo multiusuário, esse benefício é disponibilizado apenas uma única vez por conta (SERVICES, 2025c). Além disso, novos usuários que criam uma conta na plataforma AWS recebem um crédito promocional de 100 USD, que pode ser ampliado para até 200 USD mediante a realização de determinadas atividades ou treinamentos oferecidos pela própria AWS (SERVICES, 2025b).

Considerando esses benefícios, o cenário mais favorável para o desenvolvimento de experimentos seria composto por 10 horas gratuitas de treinamento, complementadas pelos créditos de 200 USD. Esse montante permitiria aproximadamente 65 horas de execução (entre treinamento e avaliação), além dos custos de armazenamento correspondentes. Apesar disso, para pesquisas mais extensas, especialmente aquelas que envolvem múltiplos experimentos e ajustes de hiperparâmetros, o tempo gratuito se torna rapidamente insuficiente, resultando em custos adicionais significativos.

Diante dessa limitação, buscou-se uma alternativa viável que mantivesse a fidelidade ao ambiente do AWS DeepRacer, mas com maior flexibilidade e menor custo. A opção identificada foi o projeto **DeepRacer-for-Cloud (DRfC)** (COMMUNITY, 2025c), desenvolvido e mantido pela AWS DeepRacer Com-

munity (COMMUNITY, 2025a). Esse repositório de código aberto oferece uma maneira prática e acessível de executar um ambiente de treinamento compatível com o AWS DeepRacer em máquinas virtuais na nuvem ou em computadores locais, proporcionando assim maior autonomia e controle sobre o processo de aprendizado por reforço, sem a necessidade de arcar com os custos diretos do serviço da AWS.

5
DeepRacer for Cloud

5.1
Simulação do Ambiente da AWS DeepRacer

O DeepRacer-for-Cloud (DRfC) foi desenvolvido com o objetivo de reproduzir localmente o ambiente de treinamento da AWS DeepRacer — ilustrado na Figura 4.1 — utilizando contêineres Docker, serviços compatíveis e componentes que simulam a infraestrutura da AWS. Dessa forma, é possível executar o pipeline completo de simulação e aprendizado por reforço sem depender da nuvem. O DRfC busca replicar a arquitetura apresentada na Seção 4.2, por meio de componentes equivalentes executados em contêineres independentes.

O contêiner Robomaker é responsável pela simulação do ambiente virtual, incluindo a pista, sensores e ações do carro autônomo. Um ou mais desses contêineres podem ser iniciados em paralelo para gerar experiências de treinamento (estado $\rightarrow$ ação $\rightarrow$ recompensa). O contêiner Sagemaker executa o treinamento do agente, gerenciando episódios, checkpoints e a atualização dos parâmetros da rede neural, desempenhando o mesmo papel do serviço SageMaker. Já o contêiner RL Coach atua como orquestrador do processo, utilizando a SDK do SageMaker no modo local (*SageMaker Local*) para coordenar a comunicação entre simulação e treinamento (COMMUNITY, 2025b).

Para substituir o serviço de armazenamento *Amazon S3*, o DRfC utiliza o MinIO, um sistema compatível com a API S3 que pode ser executado localmente. Ele é responsável por armazenar logs, checkpoints e modelos treinados, garantindo a persistência dos dados de maneira equivalente ao ambiente AWS. A comunicação entre os contêineres é realizada por meio de uma rede Docker interna (COMMUNITY, 2025c).

Essa abordagem permite reproduzir localmente quase todo o pipeline de treinamento da AWS DeepRacer, reduzindo custos e oferecendo maior controle sobre os hiperparâmetros, o número de simulações paralelas e a análise de logs. Apesar disso, podem ocorrer pequenas diferenças de desempenho e comportamento em relação ao ambiente original, devido a variações de hardware, latência e limitações da simulação local.

5.2 Processo de Instalação

O *DeepRacer-for-Cloud* (DRfC) pode ser executado em distribuições *Ubuntu* (versões 20.04, 22.04 e 24.04) ou em ambientes *Windows* utilizando o *Windows Subsystem for Linux 2* (WSL2). Antes de realizar o download do repositório, é necessário garantir que algumas dependências estejam devidamente instaladas, conforme descrito em (COMMUNITY, 2025d).

Caso a configuração seja feita via WSL2, é importante que o *Docker Desktop* esteja instalado no sistema Windows e que a integração com o WSL2 esteja habilitada. Para isso, abra o Docker Desktop e vá em `Settings → Resources → WSL Integration`, marcando a opção para permitir a integração com a distribuição Linux desejada.

Em seguida, execute os seguintes comandos para instalar os pacotes e dependências necessárias no Ubuntu:

```
sudo apt-get install jq awscli python3-boto3 docker-compose
curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
    sudo apt-key add -
sudo add-apt-repository    "deb [arch=amd64] https://download
    .docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update && sudo apt-get install -y --no-install-
    recommends docker-ce docker-ce-cli containerd.io
```

Se o computador possuir uma GPU da *NVIDIA* com pelo menos 8 GB de memória VRAM e o usuário desejar utilizar aceleração via GPU durante o treinamento, é necessário configurar o *NVIDIA Docker*. Para isso, execute os seguintes comandos:

```
curl https://get.docker.com | sh
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey |
    sudo apt-key add -

curl -s -L https://nvidia.github.io/nvidia-docker/
    $distribution/nvidia-docker.list | sudo tee /etc/apt/
    sources.list.d/nvidia-docker.list
sudo apt-get update

sudo apt-get install -y nvidia-docker2
sudo service docker stop

sudo service docker start
```

```
cat /etc/docker/daemon.json | jq 'del(."default-runtime") +
    {"default-runtime": "nvidia"}' | sudo tee /etc/docker/
    daemon.json
sudo usermod -a -G docker $(id -un)
```

Após adicionar o usuário ao grupo `docker`, é necessário reiniciar a sessão ou o sistema operacional para que as permissões sejam aplicadas corretamente.

Com as dependências instaladas, inicie o serviço do Docker e clone o repositório do DRfC:

```
sudo service docker start
git clone https://github.com/aws-deepracer-community/
    deepracer-for-cloud
cd deepracer-for-cloud
```

Em seguida, execute o script de inicialização para configurar o ambiente local. Caso deseje utilizar GPU, use o argumento `gpu`; para CPU, utilize `cpu`:

```
bin/init.sh -a gpu -c local    # Para uso com GPU
# ou
bin/init.sh -a cpu -c local    # Para uso com CPU
```

Após a execução, recomenda-se reiniciar o ambiente (reiniciar o WSL2 ou o sistema operacional Linux). Em seguida, reinicie o Docker manualmente, se necessário:

```
sudo service docker start
```

Se ocorrerem erros de permissão durante a execução do Docker, o problema pode ser corrigido com o comando:

```
sudo chmod 666 /var/run/docker.sock
```

Antes de iniciar o treinamento, é necessário configurar o serviço de armazenamento *MinIO*, utilizado pelo DRfC como alternativa local ao *Amazon S3*. Para isso, acesse sua conta AWS, gere suas credenciais (`Access Key ID` e `Secret Access Key`) através do console do IAM (*Identity and Access Management*) e execute o comando abaixo:

```
aws configure --profile minio
```

Preencha apenas os campos `AWS Access Key ID` e `AWS Secret Access Key`, deixando os demais vazios.

Por fim, ative o ambiente do DRfC com o comando:

```
source bin/activate.sh
```

Se todas as etapas anteriores tiverem sido concluídas com sucesso, os comandos `dr-*` estarão disponíveis no terminal. Esses comandos permitem gerenciar o processo de treinamento, avaliação de modelos, configuração de buckets e demais operações relacionadas ao DRfC.

5.3 Estrutura de Arquivos do DRfC

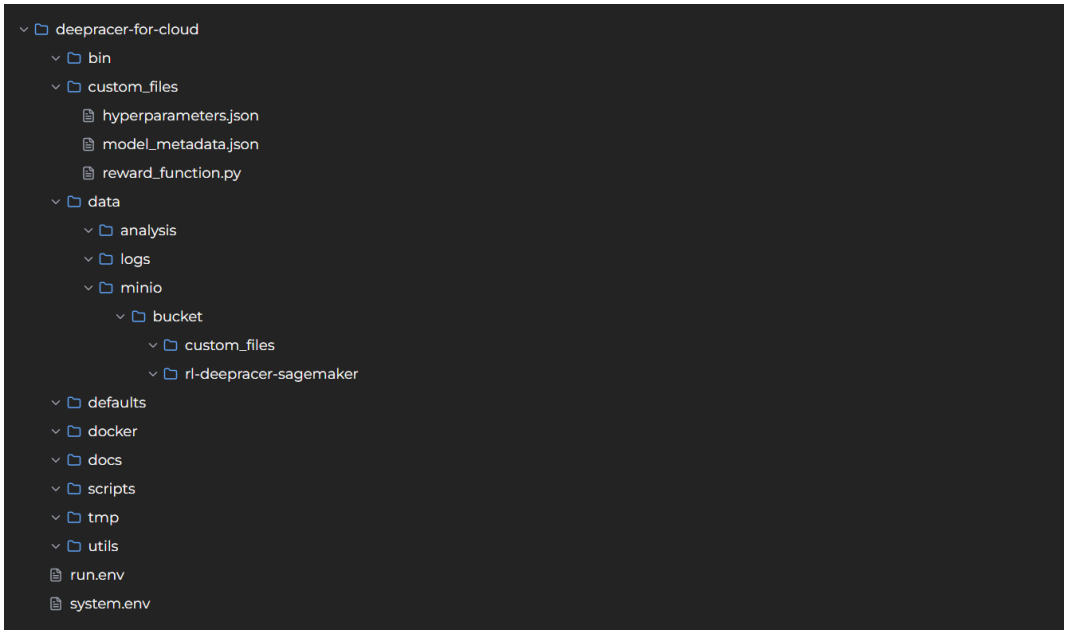

Figura 5.1: Estrutura de Arquivos do DRfC

A Figura 5.1 apresenta a estrutura básica do repositório do *DeepRacer-for-Cloud* (DRfC). A pasta `bin/` contém scripts de inicialização e comandos utilitários para configurar e ativar o ambiente local. O diretório `custom_files/` é destinado à edição e criação dos arquivos de configuração de treinamento dos modelos, permitindo especificar hiperparâmetros, funções de recompensa, sensores e demais definições personalizadas. A pasta `data/minio/bucket/` é utilizada pelos contêineres de treinamento para armazenar os dados dos modelos treinados, logs e checkpoints, funcionando como alternativa local ao serviço *Amazon S3*. O diretório `defaults/` contém exemplos de arquivos de configuração que podem ser utilizados como referência para criar ou ajustar os arquivos de treinamento no diretório `custom_files/`. Por fim, os arquivos `run.env` e `system.env` armazenam parâmetros gerais do treinamento, como seleção da pista, nome do modelo e outras definições de ambiente, que serão detalhadas na seção de Comandos. Essa organização modular facilita

a gestão dos arquivos de configuração, a execução de múltiplos treinamentos simultâneos e a manutenção dos modelos treinados, garantindo maior controle sobre o processo de aprendizado por reforço.

5.4 Comandos

Para iniciar o servidor do *MinIO* e executar os comandos do DRfC no diretório do projeto, utilize o comando:

```
source bin/activate.sh
```

Caso tenha dúvida se o servidor do *MinIO* já foi iniciado, basta utilizar o comando:

```
docker ps
```

Esse comando lista todos os contêineres ativos do Docker. Se o servidor do *MinIO* estiver em execução, ele aparecerá na lista.

Os comandos `dr-*` disponíveis são apresentados na Tabela 5.1:

Tabela 5.1: Comandos do *DeepRacer-for-Cloud* (DRfC)(COMMUNITY, 2025e)

Comando	Descrição
`dr-update`	Recarrega todos os scripts e variáveis de ambiente.
`dr-update-env`	Recarrega todas as variáveis de ambiente dos arquivos `system.env` e `run.env`.
`dr-upload-custom-files`	Atualiza as alterações de configuração do diretório `custom_files` para o bucket do *MinIO*.
`dr-download-custom-files`	Faz o download das alterações de configuração do bucket do *MinIO* para o diretório `custom_files`.
`dr-start-training`	Inicia uma sessão de treinamento local baseada nas últimas configurações submetidas.

Comando	**Descrição**
`dr-increment-training`	Atualiza a configuração do modelo atual para o prefixo pré-treinado, incrementando um número serial.
`dr-stop-training`	Interrompe a sessão de treinamento atual e atualiza os arquivos de log.
`dr-start-evaluation`	Inicia a avaliação de uma sessão de treinamento local baseada na última configuração submetida.
`dr-stop-evaluation`	Interrompe a sessão de avaliação atual e atualiza os arquivos de log.
`dr-start-loganalysis`	Inicia um contêiner Jupyter que realiza análise de logs, disponível na porta 8888.
`dr-stop-loganalysis`	Interrompe o contêiner Jupyter responsável pela análise dos logs.
`dr-start-viewer`	Inicia um proxy NGINX que transmite todos os processos do Robomaker.
`dr-stop-viewer`	Interrompe o proxy NGINX.
`dr-logs-sagemaker`	Exibe os logs do contêiner do SageMaker.
`dr-logs-robomaker`	Exibe os logs do contêiner do RoboMaker.
`dr-list-aws-models`	Lista os modelos armazenados no bucket da AWS DeepRacer S3.
`dr-set-upload-model`	Atualiza o arquivo `run.env` com o prefixo e nome do modelo selecionado.
`dr-upload-model`	Envia o modelo definido em `DR_LOCAL_S3_MODEL_PREFIX` para o prefixo do AWS DeepRacer S3 (`DR_UPLOAD_S3_PREFIX`).
`dr-download-model`	Faz o download de um arquivo de um bucket S3 real para um prefixo local de sua escolha.

Em especial, o comando `dr-start-training` aceita diferentes tipos de

argumentos. O argumento `-v` pode ser utilizado ao iniciar o treinamento para que o proxy NGINX seja criado automaticamente. O argumento `-w` é obrigatório para garantir que a exclusão dos dados de treinamentos anteriores seja intencional; ele deve ser utilizado sempre que o modelo já possuir dados existentes ou quando for necessário reiniciar o treinamento do zero devido a algum problema ocorrido durante execuções anteriores.

5.5 Configurações dos Modelos

O DRfC oferece uma configuracao bem similar ao que eh oferecido pela AWS DeepRacer. O contexto completo da configuracao de um modelo passa por cinco arquivos principais: run.env, system.env, hyperparameters.json, `model_metadata.json` e `reward_function.py`.

Os primeiros dois arquivos run.env e system.env sao arquivos que possuem uma serie de constantes de configuracao mostradas na Tabela 5.2.

Tabela 5.2: Constantes de configuração do `run.env` e `system.env` (COMMUNITY, 2025e)

Variáveis	**Descrição**
`DR_RUN_ID`	Utilizada quando há múltiplos treinamentos independentes em uma única instância do DRfC. Essa é uma configuração avançada e, normalmente, deve permanecer com o valor padrão 0.
`DR_WORLD_NAME`	Define a pista que será utilizada.
`DR_RACE_TYPE`	Opções válidas: `TIME_TRIAL`, `OBJECT_AVOIDANCE` e `HEAD_TO_BOT`.
`DR_CAR_COLOR`	Cor do carro. Opções válidas: Black, Grey, Blue, Red, Orange, White e Purple.
`DR_CAR_NAME`	Nome do carro exibido ao fazer o upload no console do DeepRacer.

Variáveis	**Descrição**
`DR_ENABLE_DOMAIN_RANDOMIZATION`	Se `True`, alterna entre diferentes cores e condições de iluminação a cada episódio. Isso é utilizado para tornar o modelo mais robusto e generalizado, em vez de excessivamente ajustado ao simulador.
`DR_UPLOAD_S3_PREFIX`	Prefixo do destino de upload (geralmente inicia com `DeepRacer-SageMaker-RoboMaker-comm-`).
`DR_EVAL_NUMBER_OF_TRIALS`	Número de voltas a serem completadas durante a avaliação.
`DR_EVAL_IS_CONTINUOUS`	Se `False`, o teste de avaliação será encerrado caso o carro saia da pista ou colida. Se `True`, o carro recebe penalidades de tempo conforme configurado, mas continua a avaliação.
`DR_EVAL_OFF_TRACK_PENALTY`	Penalidade (em segundos) aplicada caso o carro saia da pista durante a avaliação. Só tem efeito se `DR_EVAL_IS_CONTINUOUS = True`.
`DR_EVAL_COLLISION_PENALTY`	Penalidade (em segundos) aplicada em caso de colisão durante a avaliação. Só tem efeito se `DR_EVAL_IS_CONTINUOUS = True`.
`DR_EVAL_SAVE_MP4`	Define se o vídeo da avaliação será salvo (`True`).
`DR_EVAL_REVERSE_DIRECTION`	Inverte o sentido em que o carro percorre a pista quando definido como `True`.
`DR_TRAIN_CHANGE_START_POSITION`	Define se o carro deve alterar a posição inicial a cada episódio, de forma sequencial (round-robin).
`DR_TRAIN_ALTERNATE_DRIVING_DIRECTION`	Se `True`, alterna a direção de treinamento (horário/anti-horário) a cada episódio.

Variáveis	**Descrição**
`DR_TRAIN_START_POSITION_OFFSET`	Define o deslocamento da posição inicial no primeiro episódio.
`DR_TRAIN_ROUND_ROBIN_ADVANCE_DISTANCE`	Define quanto o round-robin deve avançar a cada episódio. É recomendável escolher valores proporcionais ao número de episódios por iteração (por exemplo, 0.05, 0.10 ou 0.20 para 20 episódios).
`DR_TRAIN_MULTI_CONFIG`	Se `True`, permite o uso de diferentes arquivos `run.env` em cada worker em treinamentos distribuídos.
`DR_TRAIN_MIN_EVAL_TRIALS`	Número mínimo de avaliações entre iterações de treinamento. A avaliação pode durar mais que o mínimo, dependendo do tempo de treinamento da política. Útil para acelerar o processo quando há uso de GPU.
`DR_TRAIN_REVERSE_DIRECTION`	Se `True`, inverte o sentido em que o carro percorre a pista durante o treinamento.
`DR_TRAIN_BEST_MODEL_METRIC`	Define o critério para escolher o "melhor" modelo. Opções: `progress` (maior porcentagem de completude da pista) ou `reward` (maior recompensa obtida).
`DR_TRAIN_MAX_STEPS_PER_ITERATION`	Limita o número máximo de passos por iteração. Passos excedentes são descartados para evitar vazamento de memória. O padrão é 10.000.
`DR_LOCAL_S3_PRETRAINED`	Define se o treinamento/avaliação deve utilizar um modelo pré-treinado armazenado em `s3://{DR_LOCAL_S3_BUCKET}/{LOCAL_S3_PRETRAINED_PREFIX}`.
`DR_LOCAL_S3_PRETRAINED_PREFIX`	Prefixo do modelo pré-treinado no bucket S3.
`DR_LOCAL_S3_MODEL_PREFIX`	Prefixo do modelo atual no bucket S3.
`DR_LOCAL_S3_BUCKET`	Nome do bucket S3 utilizado na sessão.

Variáveis	**Descrição**
`DR_LOCAL_S3_CUSTOM_FILES_PREFIX`	Prefixo dos arquivos de configuração no bucket S3.
`DR_LOCAL_S3_TRAINING_PARAMS_FILE`	Nome do arquivo YAML que contém os parâmetros enviados ao contêiner do RoboMaker durante o treinamento (relativo a `s3://{DR_LOCAL_S3_BUCKET}/{LOCAL_S3_PRETRAINED_PREFIX}`).
`DR_LOCAL_S3_EVAL_PARAMS_FILE`	Nome do arquivo YAML que contém os parâmetros enviados ao contêiner do RoboMaker durante a avaliação.
`DR_LOCAL_S3_MODEL_METADATA_KEY`	Caminho do arquivo `model_metadata.json`.
`DR_LOCAL_S3_HYPERPARAMETERS_KEY`	Caminho do arquivo `hyperparameters.json`.
`DR_LOCAL_S3_REWARD_KEY`	Caminho do arquivo `reward_function.py`.
`DR_LOCAL_S3_METRICS_PREFIX`	Caminho onde as métricas serão armazenadas.
`DR_OA_NUMBER_OF_OBSTACLES`	Número de obstáculos na pista (modo *Object Avoidance*).
`DR_OA_MIN_DISTANCE_BETWEEN_OBSTACLES`	Distância mínima (em metros) entre obstáculos.
`DR_OA_RANDOMIZE_OBSTACLE_LOCATIONS`	Se `True`, randomiza a posição dos obstáculos a cada episódio.
`DR_OA_IS_OBSTACLES_BOT_CAR`	Se `True`, os obstáculos são representados como carros parados em vez de caixas.
`DR_OA_OBJECT_POSITIONS`	Define as posições dos obstáculos na pista. Cada tupla representa o progresso (fração [0..1]) e o lado da pista (-1 ou 1). Exemplo: "0.23,-1;0.46,1".
`DR_H2B_IS_LANE_CHANGE`	Se `True`, os carros bots mudam de faixa de acordo com a configuração.

Variáveis	**Descrição**
`DR_H2B_LOWER_LANE_CHANGE_TIME`	Tempo mínimo (em segundos) antes de mudar de faixa.
`DR_H2B_UPPER_LANE_CHANGE_TIME`	Tempo máximo (em segundos) antes de mudar de faixa.
`DR_H2B_LANE_CHANGE_DISTANCE`	Distância (em metros) até a troca de faixa.
`DR_H2B_NUMBER_OF_BOT_CARS`	Número de carros bots na pista.
`DR_H2B_MIN_DISTANCE_BETWEEN_BOT_CARS`	Distância mínima entre carros bots.
`DR_H2B_RANDOMIZE_BOT_CARS_LOCATIONS`	Se `True`, randomiza as posições dos carros bots após cada episódio.
`DR_H2B_BOT_CAR_SPEED`	Velocidade máxima dos carros bots (em m/s).
`DR_CLOUD`	Define o tipo de ambiente de nuvem: `azure`, `aws`, `local` ou `remote`.
`DR_AWS_APP_REGION`	(Apenas na AWS) Região utilizada por outros recursos AWS (por exemplo, Kinesis).
`DR_UPLOAD_S3_PROFILE`	Perfil AWS CLI utilizado para upload dos modelos no DeepRacer (contendo as credenciais do S3).
`DR_UPLOAD_S3_BUCKET`	Nome do bucket da AWS DeepRacer onde os modelos são carregados (geralmente inicia com `aws-deepracer-`).
`DR_LOCAL_S3_PROFILE`	Nome do perfil AWS utilizado localmente. As credenciais são armazenadas em `~/.aws/credentials`, a menos que sejam usadas funções IAM.
`DR_GUI_ENABLE`	Habilita ou desabilita a interface gráfica do Gazebo no RoboMaker.
`DR_KINESIS_STREAM_NAME`	Nome do stream no Kinesis. Deixe vazio se não desejar publicar no AWS KVS.

Variáveis	**Descrição**
`DR_KINESIS_STREAM_ENABLE`	Se `True`, habilita o Kinesis Stream (AWS KVS) e o tópico `/racecar/deepracer/kvs_stream`.
`DR_SAGEMAKER_IMAGE`	Define a imagem do SageMaker a ser utilizada no treinamento.
`DR_ROBOMAKER_IMAGE`	Define a imagem do RoboMaker utilizada no treinamento ou avaliação.
`DR_MINIO_IMAGE`	Define a imagem do MinIO utilizada.
`DR_COACH_IMAGE`	Define a imagem do Coach utilizada no treinamento.
`DR_WORKERS`	Número de workers RoboMaker utilizados no treinamento.
`DR_ROBOMAKER_MOUNT_LOGS`	Se `True`, monta os logs em `$DR_DIR/data/logs/robomaker/ $DR_LOCAL_S3_MODEL_PREFIX`.
`DR_ROBOMAKER_MOUNT_SIMAPP_DIR`	Caminho para o pacote modificado do RoboMaker (exemplo: `/home/ubuntu/deepracer-simapp/bundle`).
`DR_CLOUD_WATCH_ENABLE`	Envia arquivos de log para o AWS CloudWatch.
`DR_CLOUD_WATCH_LOG_STREAM_PREFIX`	Prefixo adicionado ao nome do fluxo de logs do CloudWatch.
`DR_DOCKER_STYLE`	Opções válidas: `Swarm` e `Compose`. Recomenda-se `Compose` para otimização de contêineres OpenGL.
`DR_HOST_X`	Utiliza o servidor X-Windows do host em vez de iniciar um dentro do RoboMaker (necessário para imagens OpenGL).
`DR_WEBVIEWER_PORT`	Porta do proxy do visualizador web, que permite a transmissão simultânea de todos os processos do RoboMaker.

Variáveis	**Descrição**
`CUDA_VISIBLE_DEVICES`	Define quais GPUs serão utilizadas. Consulte a documentação adicional para mais detalhes.
`DR_TELEGRAF_HOST`	Nome do host para onde as métricas em tempo real serão enviadas. Descomentar esta linha habilita a coleta via Telegraf. É necessário que o stack `telegraf/influxdb/grafana` esteja ativo (via `dr-start-metrics`).
`DR_TELEGRAF_PORT`	Porta UDP para envio de métricas em tempo real (padrão: 8092).

O arquivo `hyperparameters.json` contém os parâmetros de configuração utilizados pelos algoritmos de aprendizado por reforço do *DeepRacer-for-Cloud* (DRfC), incluindo o *Proximal Policy Optimization* (PPO) e o *Soft Actor-Critic* (SAC). Entre os principais parâmetros definidos nesse arquivo estão:

O parâmetro `batch_size` define o tamanho do lote de amostras utilizado em cada etapa de atualização da rede neural; `beta_entropy` corresponde ao coeficiente de entropia usado para incentivar a exploração nas políticas estocásticas (no caso do PPO); `discount_factor` é o fator de desconto γ, que determina o quanto o agente valoriza recompensas futuras em relação às imediatas; `e_greedy_value` representa o valor máximo de ϵ utilizado na estratégia de exploração ϵ-*greedy*, aplicável a determinados modos de exploração; `epsilon_steps` define o número de passos necessários para reduzir ϵ do valor inicial até o mínimo configurado, controlando a taxa de redução da exploração; `exploration_type` indica o tipo de estratégia de exploração adotada (por exemplo, *additive_noise*); `loss_type` especifica o tipo de função de perda empregada durante o treinamento do agente, podendo variar conforme o algoritmo (por exemplo, *huber* ou *mean squared error*); `lr` define a taxa de aprendizado (*learning rate*) utilizada pelo otimizador para ajustar os pesos da rede; `num_episodes_between_training` determina o número de episódios de simulação executados antes de iniciar uma nova etapa de atualização dos pesos da política; `num_epochs` representa o número de épocas (iterações completas sobre o conjunto de dados) durante o treinamento em lote do PPO; `stack_size` define quantos quadros consecutivos (frames) são empilhados para formar o estado de entrada da rede neural, permitindo que o agente

perceba a dinâmica temporal do ambiente; `term_cond_avg_score` representa uma condição de parada baseada na média de pontuação — o treinamento é interrompido quando o agente atinge um desempenho médio considerado satisfatório; `term_cond_max_episodes` define o número máximo de episódios de treinamento antes da finalização forçada; e `sac_alpha` corresponde ao coeficiente α do SAC, que controla o equilíbrio entre exploração (entropia) e maximização da recompensa esperada.

No Código 3 é apresentado um exemplo de arquivo `hyperparameters.json` utilizado pelo DRfC.

Código 3: Arquivo `hyperparameters.json` do DRfC

```
{
    "batch_size": 64,
    "beta_entropy": 0.01,
    "discount_factor": 0.99,
    "e_greedy_value": 0.05,
    "epsilon_steps": 10000,
    "exploration_type": "additive_noise",
    "loss_type": "mean squared error",
    "lr": 0.0003,
    "num_episodes_between_training": 1,
    "num_epochs": 4,
    "stack_size": 1,
    "term_cond_avg_score": 900.0,
    "term_cond_max_episodes": 300,
    "sac_alpha": 0.2
}
```

O arquivo `model_metadata.json` armazena os principais parâmetros de configuração do modelo utilizados durante o treinamento.

O parâmetro `action_space` define o espaço de ações disponível para o agente, isto é, o conjunto de movimentos possíveis que o modelo pode executar. A forma de configurá-lo depende do valor de `action_space_type`, que pode ser `continuous` (ações contínuas) ou `discrete` (ações discretas).

O parâmetro `sensor` especifica os tipos de sensores utilizados no processo de treinamento, como, por exemplo, `["FRONT_FACING_CAMERA"]`.

Já o parâmetro `neural_network` indica o tipo de arquitetura de rede neural empregada pelo agente, enquanto `version` representa a versão do modelo. Por fim, o parâmetro `training_algorithm` define o algoritmo de aprendizado adotado, que pode ser `sac` (Soft Actor-Critic) ou `clipped_ppo` (Clipped Proximal Policy Optimization).

O Código 4 apresenta um exemplo de arquivo `model_metadata.json`.

Código 4: Exemplo de arquivo `model_metadata.json` do DRfC

```
{
    "action_space": {
        "steering_angle": {
            "high": 30.0,
            "low": -30.0
        },
        "speed": {
            "high": 2.0,
            "low": 1.0
        }
    },
    "sensor": [
    "FRONT_FACING_CAMERA"
    ],
    "neural_network": "DEEP_CONVOLUTIONAL_NETWORK_SHALLOW",
    "version": "5",
    "training_algorithm": "sac",
    "action_space_type": "continuous"
}
```

O arquivo `reward_function.py` é idêntico ao utilizado no AWS DeepRacer, possuindo os mesmos tipos de entrada (*inputs*). Alguns detalhes adicionais na configuração dos modelos devem ser observados: para o treinamento com o `clipped_ppo`, é possível utilizar tanto o espaço de ações contínuo quanto o discreto; já para o `sac`, apenas o espaço contínuo é suportado. Além disso, devido às características do algoritmo SAC, o parâmetro `num_episodes_between_training` deve obrigatoriamente ser configurado com o valor `1`. Caso contrário, após o primeiro episódio o modelo entra em modo de avaliação, finaliza essa etapa e, ao iniciar o segundo episódio, entra em estado de *deadlock*, impossibilitando a continuidade do treinamento.

5.6 Análise dos Modelos

O *DeepRacer-for-Cloud* (DRfC), ao executar o comando `dr-start-loganalysis`, cria um contêiner Jupyter que pode ser acessado por uma porta específica, conforme descrito na Seção 5.4. Esse contêiner disponibiliza seis notebooks principais: `Visual_Analysis.ipynb`, `Training_analysis.ipynb`, `Metrics.ipynb`, `Evaluation_analysis.ipynb`, `Console.ipynb` e `ActionSpace_analysis.ipynb`.

O notebook `Visual_Analysis.ipynb` tem como objetivo exibir o que o modelo "enxerga" ao tomar suas decisões de ação, permitindo visualizar as imagens de entrada e o comportamento da rede neural.

O `Training_analysis.ipynb` gera diversos gráficos e estatísticas que

permitem analisar o comportamento, evolução e progressão do modelo ao longo dos episódios e iterações. São avaliados parâmetros como recompensa, tempo de execução, percentual de completude da pista e métricas de desempenho. Esse notebook também realiza uma análise por quintiles, separando os episódios em cinco grupos e comparando sua evolução em termos de progresso, tempo e recompensa média. Além disso, é possível visualizar tabelas detalhadas com os dados de cada episódio (posição inicial, recompensa total, progresso percentual etc.) e trajetórias percorridas pelo agente em diferentes iterações.

O notebook `Metrics.ipynb` possui aparência e estrutura semelhantes ao `Training_analysis.ipynb`, mas se diferencia por incluir também a análise das avaliações (evaluations) realizadas entre as iterações. Essa análise é fundamental, pois um modelo pode não completar a pista durante o treinamento, mas fazê-lo nas etapas de avaliação — o que reflete o comportamento típico do aprendizado por reforço, em que o agente equilibra exploração e refinamento da política durante o treinamento, enquanto nas avaliações tende a adotar as melhores ações já aprendidas.

O notebook `Evaluation_analysis.ipynb` apresenta uma análise detalhada dos trajetos realizados durante o processo de avaliação, exibindo inclusive a velocidade do agente em diferentes trechos da pista. Os dados utilizados são provenientes especificamente das execuções iniciadas pelo comando `dr-start-evaluation`.

O notebook `Console.ipynb` tem como finalidade oferecer uma visualização simplificada e de fácil acesso do progresso do treinamento, atuando como uma alternativa leve aos notebooks `Training_analysis.ipynb` e `Metrics.ipynb`. Ele se conecta diretamente aos dados do console do DRfC (ou do AWS DeepRacer), permitindo carregar rapidamente métricas e gráficos atualizados sobre o desempenho do modelo, sem necessidade de realizar análises detalhadas ou processamento adicional de dados.

Esse notebook é especialmente útil para acompanhamento em tempo real e para usuários que desejam uma visão geral do treinamento, com foco em métricas resumidas de desempenho e convergência. Sua principal diferença em relação aos outros notebooks é a simplicidade: ele prioriza a atualização rápida das informações e a integração direta com a interface de monitoramento, em vez de análises exploratórias mais profundas.

Antes de utilizá-lo, é necessário garantir que as dependências estejam instaladas, incluindo a biblioteca `deepracer-utils (>=0.23)`. O acesso programático à AWS também deve estar configurado por meio do `AWS CLI` e do `Boto3`.

O notebook `ActionSpace_analysis.ipynb` é utilizado para analisar o

espaço de ações definido para o agente — seja ele contínuo ou discreto — no contexto do treinamento do AWS DeepRacer e do DRfC. Ele faz parte do conjunto de ferramentas da comunidade *AWS DeepRacer-Analysis* e permite avaliar como diferentes configurações do espaço de ações (como combinações de velocidade e ângulo de direção) impactam o desempenho e a eficiência do aprendizado.

Esse notebook é especialmente útil para comparar políticas ou modelos treinados com diferentes configurações de `action_space`, ajudando o usuário a compreender melhor o equilíbrio entre precisão de controle e complexidade computacional.

Assim como os demais notebooks, ele requer que o ambiente do *AWS DeepRacer-Analysis* esteja devidamente instalado e configurado.

5.7 Vantagens e Limitações

O *DeepRacer-for-Cloud* (DRfC) foi desenvolvido pela comunidade do AWS DeepRacer com o objetivo de oferecer uma alternativa local e de código aberto ao serviço proprietário da AWS. Embora ambos compartilhem a mesma base conceitual e utilizem algoritmos idênticos de aprendizado por reforço, como o *Proximal Policy Optimization* (PPO) e o *Soft Actor-Critic* (SAC), há diferenças significativas em termos de custo, acessibilidade, flexibilidade e experiência do usuário.

A principal vantagem do DRfC está na redução de custos. Enquanto o serviço AWS DeepRacer depende de recursos de computação em nuvem, com cobrança proporcional ao tempo de uso das instâncias, o DRfC permite que todo o processo de treinamento seja executado localmente, utilizando a infraestrutura do próprio usuário. Dessa forma, elimina-se o custo contínuo associado à execução de múltiplas simulações e treinamentos em nuvem, tornando o processo mais acessível para pesquisa, testes e desenvolvimento de modelos.

Outra vantagem importante é a flexibilidade do ambiente. O DRfC é totalmente configurável e pode ser ajustado de acordo com as necessidades do usuário, permitindo, por exemplo, alterar diretamente os parâmetros dos contêineres, modificar funções internas de treinamento e personalizar a forma de coleta e análise de métricas. Essa característica torna o DRfC uma ferramenta especialmente útil em contextos acadêmicos e experimentais, em que é desejável compreender e manipular os detalhes internos do processo de aprendizado por reforço.

Além disso, o DRfC disponibiliza uma série de notebooks de análise

(`Training_analysis.ipynb`, `Metrics.ipynb`, `Evaluation_analysis.ipynb`, `Console.ipynb`, entre outros) que permitem investigar com profundidade o comportamento do agente durante o treinamento e a avaliação. Esses notebooks facilitam a identificação de gargalos, a visualização do progresso do modelo e a compreensão dos efeitos de diferentes configurações de hiperparâmetros sobre o desempenho final — algo que é limitado na interface web do AWS DeepRacer.

Por outro lado, o serviço AWS DeepRacer apresenta uma interface significativamente mais amigável e intuitiva. Sua plataforma web oferece recursos prontos para uso, como acompanhamento gráfico do progresso, comparação automática entre modelos, gerenciamento simplificado de pistas e integração direta com competições oficiais da AWS. Isso torna o serviço ideal para iniciantes ou usuários que desejam focar no comportamento do agente, sem a necessidade de lidar com detalhes técnicos da infraestrutura.

Entre as limitações do DRfC, destacam-se a necessidade de maior conhecimento técnico para instalação e configuração do ambiente, a dependência de recursos computacionais locais (CPU e GPU) e a ausência de integração direta com o ecossistema de competições e rankings da AWS. Além disso, a interface de uso, baseada em terminal e notebooks Jupyter, não oferece a mesma fluidez e simplicidade visual da plataforma original.

Em síntese, o DRfC apresenta-se como uma alternativa poderosa e de baixo custo ao serviço AWS DeepRacer, especialmente voltada a usuários avançados, pesquisadores e desenvolvedores que buscam controle total sobre o processo de aprendizado. Já o serviço oficial da AWS mantém sua vantagem em termos de praticidade, escalabilidade e experiência de uso, sendo mais adequado para fins educacionais, competitivos e de demonstração.

6
Experimentos e Resultados

Os experimentos realizados neste trabalho tiveram como objetivo avaliar o impacto de diferentes configurações de treinamento no desempenho dos agentes de *reinforcement learning* (RL) no ambiente do *DeepRacer-for-Cloud* (DRfC). Cada variação experimental foi planejada para isolar e compreender a influência de aspectos específicos da configuração do ambiente, da política de controle e do processo de coleta de dados.

Todos os modelos seguiram uma configuração-base comum, definida para garantir comparabilidade entre os resultados. Entre essas configurações, destacam-se o número máximo de episódios de treinamento (`term_cond_max_episodes`) igual a 300 e o parâmetro `epsilon_steps` definido como 10.000. Assim, a maioria dos modelos atingiu 300 episódios de treinamento, e os poucos que não chegaram a esse valor foram interrompidos devido à condição de número máximo de passos por iteração, ainda assim apresentando número de episódios muito próximo do limite estabelecido.

6.1
Pistas Utilizadas no Treinamento

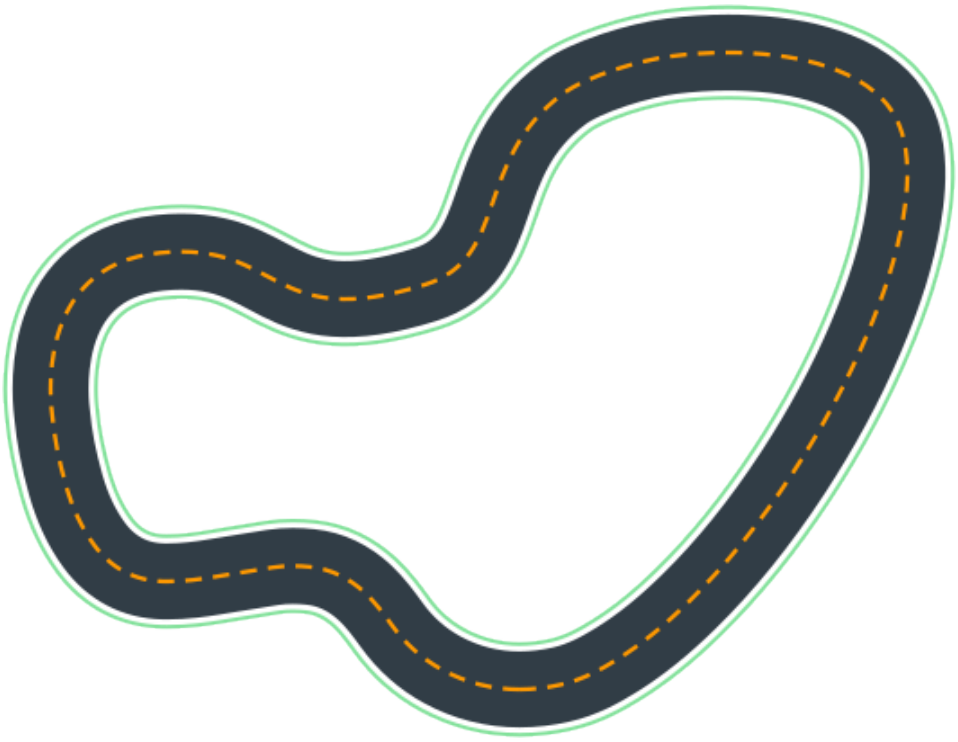

Figura 6.1: Pista *Smile Speedway* (`reInvent2019_track`)

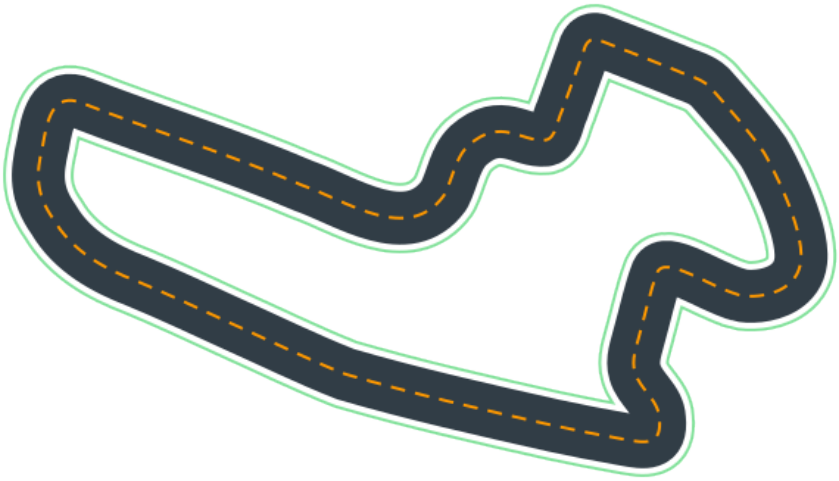

Figura 6.2: Pista *Jennens Super Speedway* (`2022_october_pro`)

Os modelos foram treinados em duas pistas diferentes. A primeira, denominada *Smile Speedway*, apresenta curvas suaves e um traçado mais simples. Para utilizá-la no DRfC, a variável `DR_WORLD_NAME` foi configurada com o valor *reInvent2019_track*; nas tabelas de síntese, essa pista é identificada como **Básica**.

A segunda pista, chamada *Jennens Super Speedway*, possui um traçado mais extenso e desafiador, com trechos longos e curvas acentuadas. Nesse caso, a variável `DR_WORLD_NAME` foi definida como *2022_october_pro*; nas tabelas, ela é referenciada como **Avançada**. A correspondência entre o nome da pista e o valor da variável foi verificada no repositório de dados da AWS DeepRacer Community (COMMUNITY, 2025f).

O objetivo dessa variação foi analisar como o nível de dificuldade e a complexidade geométrica da pista afetam a capacidade do agente de aprender trajetórias estáveis e evitar comportamentos erráticos. Esperava-se que a pista mais simples favorecesse uma convergência mais rápida, enquanto a pista mais complexa exigisse políticas mais generalistas e robustas.

6.2 Alternância de Direção Durante o Treinamento

Outro conjunto de experimentos avaliou o impacto da alternância de direção — sentido horário e anti-horário — entre os episódios de treinamento.

Essa configuração foi implementada de modo que, a cada novo episódio, o carro trocasse o sentido de deslocamento na pista.

O objetivo dessa variação foi promover uma melhor generalização do modelo, evitando o sobreajuste a um único padrão de curva. Por exemplo, uma pista que apresenta predominantemente curvas à direita no sentido horário passa a oferecer curvas à esquerda quando percorrida no sentido inverso, permitindo que o agente tenha uma experiência mais ampla de condução e desenvolva políticas simétricas. Esperava-se, portanto, que modelos treinados com alternância de direção apresentassem maior capacidade de adaptação a pistas inéditas.

6.3 Tipo de Espaço de Ações

Também foram realizados experimentos comparando espaços de ações discretos e contínuos. Essa escolha influencia diretamente o comportamento do agente durante o treinamento e a capacidade do modelo de lidar com situações mais complexas.

O espaço de ações discreto foi testado em duas configurações distintas. A primeira apresenta cinco combinações possíveis de ângulo de direção e uma velocidade constante baixa de 0.6 m/s. Essa abordagem simplifica o processo de decisão e favorece a estabilidade inicial do treinamento, pois o agente possui um conjunto pequeno e fixo de ações disponíveis; nas tabelas de síntese, essa configuração é identificada como **Discreto básico**.

Código 5: Espaço de ações discreto com velocidade constante baixa

```
{
    "action_space": [
        {
            "steering_angle": -30,
            "speed": 0.6
        },
        {
            "steering_angle": -15,
            "speed": 0.6
        },
        {
            "steering_angle": 0,
            "speed": 0.6
        },
        {
            "steering_angle": 15,
            "speed": 0.6
        },
        {
```

```
            "steering_angle": 30,
            "speed": 0.6
        }
    ]
}
```

A segunda configuração discreta amplia o número de combinações possíveis, permitindo que o modelo escolha diferentes velocidades conforme a necessidade de fazer curvas ou seguir em linha reta; nas tabelas, essa configuração é denominada **Discreto complexo**.

Código 6: Espaço de ações discreto com múltiplas velocidades

```
{
    "action_space": [
        {
            "steering_angle": -30,
            "speed": 1.0
        },
        {
            "steering_angle": -15,
            "speed": 1.0
        },
        {
            "steering_angle": 0,
            "speed": 1.0
        },
        {
            "steering_angle": 15,
            "speed": 1.0
        },
        {
            "steering_angle": 30,
            "speed": 1.0
        },
        {
            "steering_angle": -30,
            "speed": 1.5
        },
        {
            "steering_angle": -15,
            "speed": 1.5
        },
        {
            "steering_angle": 0,
            "speed": 1.5
        },
        {
            "steering_angle": 15,
            "speed": 1.5
        },
```

```
39          {
40              "steering_angle": 30,
41              "speed": 1.5
42          },
43          {
44              "steering_angle": -30,
45              "speed": 2.0
46          },
47          {
48              "steering_angle": -15,
49              "speed": 2.0
50          },
51          {
52              "steering_angle": 0,
53              "speed": 2.0
54          },
55          {
56              "steering_angle": 15,
57              "speed": 2.0
58          },
59          {
60              "steering_angle": 30,
61              "speed": 2.0
62          }
63      ]
64 }
```

Em geral, o uso do espaço discreto facilita a exploração inicial e tende a acelerar a convergência do modelo, já que reduz a dimensionalidade do problema de decisão.

O espaço contínuo, por sua vez, permite que o agente escolha qualquer valor dentro de um intervalo definido para cada variável de ação; nas tabelas, essa configuração é identificada como **Contínuo**. No experimento, o intervalo adotado foi de -30 a 30 para o ângulo de direção e de 1 m/s a 2 m/s para a velocidade. Esse tipo de controle proporciona movimentos mais suaves e realistas, embora torne o processo de otimização mais complexo e demorado.

Código 7: Espaço de ações contínuo

```
1 {
2     "action_space": {
3         "speed": {
4             "high": 2,
5             "low": 1
6         },
7         "steering_angle": {
8             "high": 30,
9             "low": -30
10        }
```

```
11      }
12 }
```

Apesar de exigir mais tempo para convergir, o espaço contínuo oferece maior precisão no controle do veículo e tende a resultar em trajetórias mais estáveis e naturais.

6.4 Tipo de Funções de Recompensa

A função de recompensa é um dos elementos mais importantes em algoritmos de aprendizado por reforço, pois define diretamente o comportamento que o agente deve aprender a otimizar. Pequenas mudanças na forma de calcular a recompensa podem alterar significativamente a trajetória de aprendizado e o desempenho final do modelo.

Nos experimentos realizados, foram utilizados dois tipos principais de funções de recompensa: uma discreta e uma linear. Ambas têm como objetivo central incentivar o veículo a permanecer próximo ao centro da pista, porém diferem na maneira como representam e distribuem a recompensa ao longo da largura da pista.

6.4.1 Função de Recompensa Discreta

A primeira função de recompensa corresponde à configuração padrão do repositório *DeepRacer-for-Cloud*. Ela avalia a posição do carrinho em relação ao centro da pista e atribui recompensas fixas de acordo com faixas discretas de distância. Quanto mais próximo do centro, maior a recompensa recebida, conforme ilustrado na Figura 6.3.

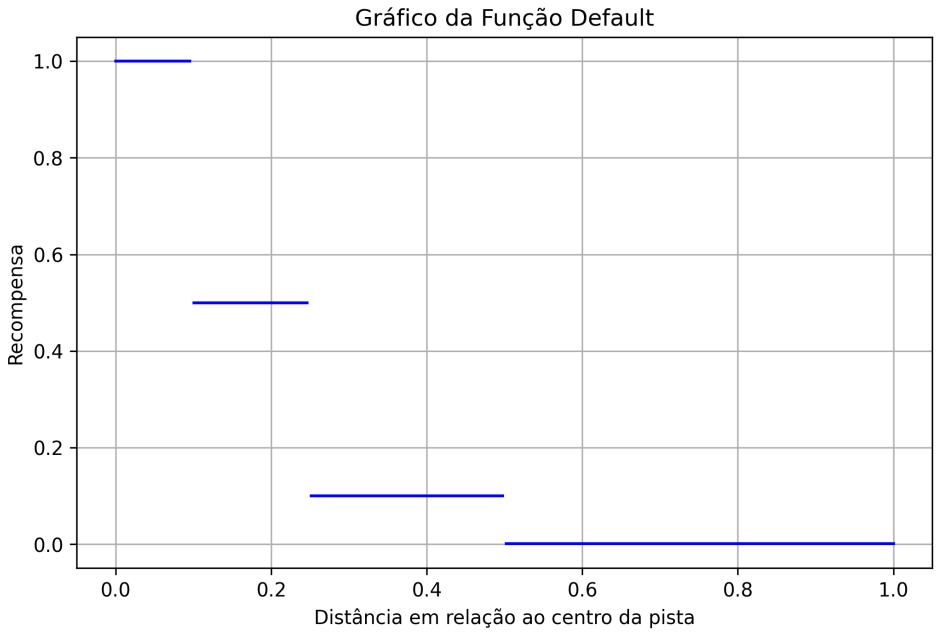

Figura 6.3: Distribuição de recompensa da função discreta.

Além disso, essa função aplica uma penalização a mudanças bruscas de direção, multiplicando a recompensa por 0,8 quando há uma variação acentuada no ângulo de direção. Esse mecanismo reduz o comportamento de *zig-zag* e favorece uma condução mais suave; nas tabelas de síntese, essa função é identificada como **Default**. O código da função de recompensa discreta é apresentado no Código 8.

Código 8: Função de recompensa padrão do DRfC.

```
def reward_function(params):

    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    steering = abs(params['steering_angle'])

    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3

    ABS_STEERING_THRESHOLD = 15
```

```
21
22      if steering > ABS_STEERING_THRESHOLD:
23          reward *= 0.8
24
25      return float(reward)
```

6.4.2 Função de Recompensa Linear

A segunda função de recompensa mantém o mesmo princípio — incentivar o veículo a se manter no centro da pista —, mas substitui a discretização por uma variação contínua dos valores de recompensa. Nessa abordagem, a recompensa diminui linearmente à medida que o carrinho se afasta do centro, como mostrado na Figura 6.4.

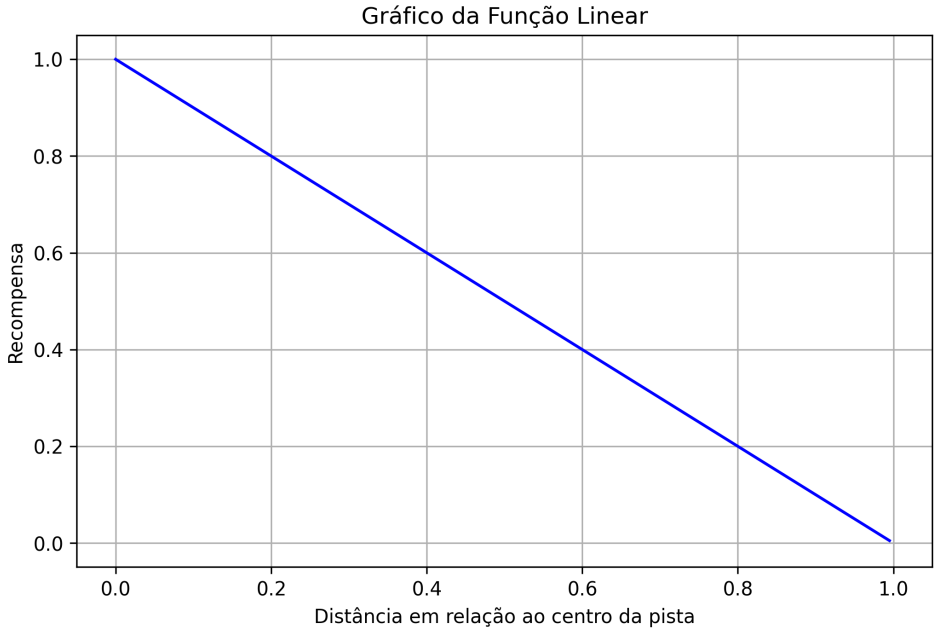

Figura 6.4: Distribuição de recompensa da função linear.

Essa mudança torna o sinal de aprendizado mais granular, fornecendo ao agente informações mais detalhadas sobre a qualidade de suas ações. A expectativa era de que esse formato permitisse ao modelo identificar mais rapidamente os comportamentos que resultam em maiores recompensas. Assim como na versão discreta, essa função mantém a penalização para mudanças bruscas de direção; nas tabelas, ela é referenciada como **Linear**. O código correspondente é apresentado no Código 9.

Código 9: Função de recompensa com distribuição linear.

```
1 def reward_function(params):
```

```
2
3     distance_from_center = params['distance_from_center']
4     track_width = params['track_width']
5     steering = abs(params['steering_angle'])
6
7     max_distance = track_width/2.0
8
9     reward = (max_distance - distance_from_center)/max_distance
10    if reward < 1e-3:
11        reward = 1e-3
12
13    ABS_STEERING_THRESHOLD = 15
14
15    if steering > ABS_STEERING_THRESHOLD:
16        reward *= 0.8
17
18    return float(reward)
```

6.5 Síntese dos Experimentos

Tabela 6.1: Modelos treinados no *DeepRacer-for-Cloud* (DRfC)

Modelo	**Algoritmo**	**Função de recompensa**	**Pista**	**Espaço de ações**	**Direção alternada**
1	SAC	Default	Básica	Contínuo	Não
2	SAC	Default	Básica	Contínuo	Sim
3	SAC	Default	Avançada	Contínuo	Não
4	SAC	Default	Avançada	Contínuo	Sim
5	PPO	Default	Básica	Discreto básico	Não
6	PPO	Default	Básica	Discreto básico	Sim
7	PPO	Default	Avançada	Discreto básico	Não
8	PPO	Default	Básica	Discreto complexo	Não
9	PPO	Default	Básica	Discreto complexo	Sim
10	PPO	Default	Avançada	Discreto complexo	Não

Modelo	**Algoritmo**	**Função de recompensa**	**Pista**	**Espaço de ações**	**Direção alternada**
11	PPO	Default	Avançada	Discreto complexo	Sim
12	PPO	Linear	Básica	Discreto complexo	Não
13	PPO	Linear	Básica	Discreto complexo	Sim
14	PPO	Linear	Avançada	Discreto complexo	Não
15	PPO	Linear	Avançada	Discreto complexo	Sim
16	PPO	Default	Básica	Contínuo	Não
17	PPO	Default	Básica	Contínuo	Sim
18	PPO	Default	Avançada	Contínuo	Não
19	PPO	Default	Avançada	Contínuo	Sim
20	PPO	Linear	Básica	Contínuo	Não
21	PPO	Linear	Básica	Contínuo	Sim

Tabela 6.2: Desempenho no treinamento *DeepRacer-for-Cloud* (DRfC)

Modelo	**Pista**	**Primeiro episódio completo**	**Episódios completos**	**Percentual médio de completude**	**Média de recompensa**	**Menor tempo**
1	Básica	105	131	62,97	75,24	13,804
2	Básica	256	6	19,90	25,89	14,004
3	Avançada	164	79	43,13	158,60	37,135
4	Avançada	199	40	31,43	111,68	36,946
5	Básica	137	93	48,60	181,30	33,213
6	Básica	150	66	43,26	171,78	33,411
7	Avançada	269	5	15,98	167,08	90,912
8	Básica	222	5	22,34	35,03	15,551
9	Básica	221	10	22,09	39,65	15,461
10	Avançada	-	0	8,13	34,80	-
11	Avançada	-	0	9,24	41,12	-
12	Básica	225	8	21,89	38,76	14,686
13	Básica	229	3	18,38	33,00	15,926
14	Avançada	-	0	9,39	44,09	-

Modelo	**Pista**	**Primeiro episódio completo**	**Episódios completos**	**Percentual médio de completude**	**Média de recompensa**	**Menor tempo (s)**
15	Avançada	-	0	5,66	26,42	-
16	Básica	228	13	25,48	41,18	16,593
17	Básica	296	2	16,79	24,67	14,802
18	Avançada	-	0	8,62	35,11	-
19	Avançada	-	0	7,14	28,84	-
20	Básica	214	18	26,58	40,51	14,837
21	Básica	200	2	14,74	22,91	17,984

Antes de iniciar a análise dos resultados, é importante compreender como interpretar as informações apresentadas nas tabelas e em quais contextos as comparações entre modelos fazem sentido. As colunas Primeiro episódio completo, Episódios completos, Percentual médio de completude e Menor tempo devem ser comparadas apenas entre modelos que foram treinados na mesma pista, pois refletem diretamente as características do ambiente de simulação. Já a coluna Média de recompensa deve ser comparada entre modelos treinados na mesma pista e utilizando a mesma função de recompensa, uma vez que essa métrica é sensível às diferenças de formulação da função de avaliação do comportamento do agente.

Ao analisar os resultados apresentados na Tabela 6.2, observa-se que os modelos treinados com o algoritmo *Soft Actor-Critic* (SAC) apresentaram, em média, um desempenho superior aos modelos treinados com o algoritmo *Proximal Policy Optimization* (PPO). Os agentes baseados em SAC conseguiram completar as pistas em menos episódios e atingiram percentuais médios de completude mais elevados, o que indica uma convergência mais rápida e estável durante o treinamento.

Entretanto, é importante ressaltar que, pelas suas características — tanto do SAC, cuja proposta é aprender mais com menos experiência por meio do uso do *replay buffer* para otimizar o aprendizado, quanto pela configuração de hiperparâmetros —, após cada episódio o modelo passava por uma fase de avaliação. Esse conjunto de fatores fez com que, embora o modelo tenha apresentado melhor desempenho em termos de número de episódios, o tempo total de treinamento tenha sido significativamente maior. Alguns modelos levaram mais de 30 horas, enquanto o modelo PPO mais demorado precisou de apenas 8 horas para concluir os 300 episódios.

Como os modelos SAC necessitaram de menos episódios para completar a pista pela primeira vez, é possível afirmar que o SAC se mostrou mais eficiente

em termos de aproveitamento da experiência com o ambiente, ainda que com maior custo de tempo de treinamento.

Entre os modelos SAC, o modelo 2 apresentou desempenho inferior em relação aos demais, registrando baixo número de episódios completos e menor percentual médio de completude. Esse comportamento pode estar relacionado tanto à característica de máxima entropia do algoritmo — que tende a aumentar a exploração em detrimento da estabilidade inicial — quanto ao fato de a alternância de direção estar ativada, o que possivelmente dificultou o processo de convergência do modelo.

Outra característica perceptível é que os modelos treinados com PPO tiveram muita dificuldade para completar uma volta durante a fase de treinamento. Em relação à função de recompensa, a função *Linear* no espaço contínuo de ação parece ter ajudado os modelos 20 e 21 a convergirem mais rapidamente quando comparados aos modelos 16 e 17. Ao observar os modelos PPO treinados com espaço de ação discreto, nota-se que a convergência foi um pouco mais lenta (modelos 12–15 com função *Linear* e 8–11 com a função *Default*), mas a diferença foi pequena. De modo geral, a alternância de direção não parece ter impactado significativamente a convergência dos modelos — em alguns casos acelerou o aprendizado, enquanto em outros o retardou.

Tabela 6.3: Desempenho nas avaliacoes *DeepRacer-for-Cloud* (DRfC)

Modelo	**Pista**	**Primeiro episódio completo**	**Episódios completos**	**Percentual médio de completude**	**Média de recompensa**	**Menor tempo**
1	Básica	26	1433	56,09	60,17	12,435
2	Básica	212	45	8,92	15,54	12,747
3	Avançada	181	593	32,36	113,83	34,132
4	Avançada	218	48	9,89	26,87	34,483
5	Básica	140	75	61,75	240,76	34,067
6	Básica	80	96	75,25	315,46	33,203
7	Avançada	40	32	57,21	618,81	88,620
8	Básica	140	16	51,72	70,34	13,817
9	Básica	100	59	63,40	102,68	14,384
10	Avançada	-	0	23,61	107,52	-
11	Avançada	140	3	29,19	159,24	44,115
12	Básica	180	27	44,63	73,04	14,046
13	Básica	100	47	56,12	90,82	14,095
14	Avançada	-	0	24,88	160,22	-
15	Avançada	-	0	17,56	93,78	-

Modelo	**Pista**	**Primeiro episódio completo**	**Episódios completos**	**Percentual médio de completude**	**Média de recompensa**	**Menor tempo (s)**
16	Básica	160	50	56,35	86,68	14,420
17	Básica	180	37	38,88	54,67	13,275
18	Avançada	-	0	25,53	128,03	-
19	Avançada	300	1	23,55	129,42	41,352
20	Básica	140	58	66,20	98,92	13,568
21	Básica	180	33	37,73	63,31	16,820

Antes de começar a análise dos resultados da Tabela 6.3, é importante ressaltar que os modelos SAC realizavam diversas avaliações a cada episódio, enquanto os modelos treinados com PPO faziam apenas quatro avaliações a cada 20 episódios.

Um dos pontos que se destacam nesta análise, ao comparar com a Tabela 6.2, é que, no caso do SAC, observando a coluna **Percentual médio de completude**, é possível perceber que a alternância de direção impactou significativamente na convergência do modelo — um detalhe que não era evidente na tabela anterior.

Analisando agora os modelos treinados com PPO durante o período de avaliação, nota-se que alguns conseguiram completar a pista avançada em determinados momentos. Destaca-se o modelo 7, que, na bateria de avaliações após o episódio 40, conseguiu completar a pista avançada, embora durante o treinamento tenha completado a pista pela primeira vez apenas no episódio 269.

Outro aspecto relevante é que, de modo geral, a primeira vez que os modelos PPO conseguem completar a pista ocorre durante as avaliações, enquanto nos modelos SAC isso varia mais. Essa diferença se deve à frequência com que cada modelo é reavaliado: os modelos PPO passam por avaliações a cada 20 episódios, enquanto os modelos SAC são avaliados a cada episódio.

O último aspecto analisado nos modelos treinados foi a capacidade de generalização em diferentes pistas de corrida. Para isso, cada modelo foi submetido a uma única tentativa em cinco pistas distintas. Duas delas — *Smile Speedway* e *Jennens Super Speedway* — foram utilizadas durante o treinamento, enquanto as demais — *Roger Ring*, *Jochem Turnpike* e *Expedition Loop* — serviram para avaliar o desempenho fora do ambiente aprendido.

Em cada avaliação, o objetivo do agente era completar o percurso. Caso o veículo saísse da rota em qualquer ponto, o teste era imediatamente encerrado. Para cada pista, são registradas duas métricas: (i) o progresso alcançado,

expresso em porcentagem do trajeto total, e (ii) o tempo de execução, em segundos, correspondente ao período durante o qual o modelo permaneceu ativo no percurso.

Na Tabela 6.4, as pistas são identificadas de forma sequencial para simplificar a apresentação dos resultados:

- **Pista 1**: Smile Speedway
- **Pista 2**: Jennens Super Speedway
- **Pista 3**: Roger Ring
- **Pista 4**: Jochem Turnpike
- **Pista 5**: Expedition Loop

A tabela apresenta, para cada modelo, o percentual completado e o tempo obtido em cada uma das cinco pistas, permitindo comparar sua robustez e capacidade de adaptação a cenários diferentes dos utilizados durante o treinamento.

Tabela 6.4: Desempenho nas pistas

	Pista 1		**Pista 2**		**Pista 3**		**Pista 4**		**Pista 5**	
Modelo	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**
1	100,00	13,846	6,18	2,617	46,66	13,065	79,88	20,558	41,59	11,103
2	56,20	7,646	27,69	10,51	47,21	13,024	83,36	21,489	12,97	3,364
3	98,69	13,719	100,00	36,723	100,00	26,797	85,68	22,619	2,82	1,132
4	98,10	13,616	26,70	10,258	100,00	25,990	83,27	21,029	42,36	11,178
5	100,00	34,204	100,00	91,758	100,00	68,109	100,00	63,997	60,65	39,816
6	100,00	33,881	100,00	91,352	100,00	67,163	100,00	64,191	72,88	45,718
7	100,00	31,592	100,00	89,231	100,00	65,941	100,00	61,644	100,00	57,943
8	100,00	14,982	34,43	15,179	47,34	15,524	38,11	12,817	11,22	3,830
9	100,00	17,123	90,06	44,763	100,00	35,880	88,89	31,555	100,00	31,212
10	100,00	15,794	34,52	13,555	47,64	14,643	82,71	23,790	43,32	11,118
11	100,00	17,966	26,22	11,718	30,01	9,686	80,35	24,750	5,04	1,707
12	100,00	14,969	34,52	15,307	47,25	15,329	83,36	24,251	60,65	18,083
13	100,00	16,725	34,80	14,607	100,00	30,702	84,04	25,030	14,66	5,258
14	100,00	14,452	33,97	15,580	47,34	16,226	80,51	28,530	14,05	3,579
15	100,00	13,267	33,97	14,513	100,00	29,807	39,72	11,768	2,48	1,107
16	100,00	15,554	34,52	15,481	47,34	14,857	81,37	24,001	60,07	19,050

	Pista 1 (cont.)		**Pista 2 (cont.)**		**Pista 3 (cont.)**		**Pista 4 (cont.)**		**Pista 5 (cont.)**	
Modelo	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**	**Prog. (%)**	**Tempo (s)**
17	100,00	13,482	35,16	13,518	100,00	27,630	89,72	23,309	42,75	11,373
18	100,00	14,936	34,45	14,316	34,05	14,451	100,00	30,168	39,31	11,079
19	18,12	3,297	34,55	15,286	100,00	30,696	100,00	28,027	43,83	12,153
20	100,00	13,708	34,75	14,143	100,00	28,920	89,76	24,175	83,52	22,728
21	100,00	18,821	88,91	44,726	100,00	37,438	88,89	31,029	41,72	16,751

Ao observar a Tabela 6.4, destaca-se que, entre os 21 modelos treinados, apenas um deles — o Modelo 7, treinado com a função de recompensa *default*, espaço de ações discreto com velocidade constante de 0,6 m/s e desenvolvido em uma pista considerada avançada — foi capaz de completar uma volta em todas as pistas avaliadas. Esse resultado inicial sugere que o Modelo 7 apresenta uma capacidade de generalização superior, já que nenhum outro modelo conseguiu concluir todos os percursos.

Entretanto, a simples capacidade de completar todas as pistas não é suficiente para caracterizar o melhor desempenho geral quando se adota uma avaliação sistemática. Por esse motivo, foi definido um critério de ranqueamento que considera, para cada pista, prioritariamente o *progresso* e, em caso de empate, o *tempo* necessário para percorrê-lo. Com base nesse critério, os modelos recebem uma pontuação que varia de 1 (melhor desempenho) a 21 (pior desempenho). Ao final, a soma das pontuações obtidas nas cinco pistas determina o desempenho global de cada modelo.

Tabela 6.5: Ranqueamento dos modelos

Modelo	**Ranque 1**	**Ranque 2**	**Ranque 3**	**Ranque 4**	**Ranque 5**	**Ranque Geral**
1	4	21	19	19	13	76
2	20	18	18	12	17	85
3	18	1	2	10	20	51
4	19	19	1	14	11	64
5	17	4	12	4	6	43
6	16	3	11	5	4	39
7	15	2	10	3	2	32
8	8	15	15	21	18	77
9	12	5	8	9	1	35
10	10	11	13	15	9	58
11	13	20	21	18	19	91

Modelo	**Ranque 1**	**Ranque 2**	**Ranque 3**	**Ranque 4**	**Ranque 5**	**Ranque Geral**
12	7	12	17	13	5	54
13	11	8	7	11	15	52
14	5	17	16	17	16	71
15	1	16	5	20	21	63
16	9	13	14	16	7	59
17	2	7	3	7	10	29
18	6	14	20	2	14	56
19	21	10	6	1	8	46
20	3	9	4	6	3	25
21	14	6	9	8	12	49

A aplicação desse método revela um aspecto importante: embora o Modelo 7 tenha sido o único capaz de completar todas as pistas, ele não apresentou tempos competitivos em algumas delas. Como consequência, sua posição relativa no ranking — determinado não apenas pelo fato de completar a pista, mas também pela eficiência com que o fez — não o coloca como o melhor modelo de acordo com o critério adotado. Assim, o ranking evidencia que completar todas as pistas é um indicativo de robustez, mas não necessariamente de desempenho ideal quando se considera a comparação direta com outros modelos em métricas combinadas de eficácia e eficiência.

Além disso, a análise dos *scores* evidencia um padrão relevante: com exceção dos modelos treinados com SAC, nos conjuntos de comparações entre modelos com ações simétricas (alternando direção) e modelos sem alternância, em seis dos oito pares utilizando PPO, o modelo com alternância apresentou melhor desempenho segundo o critério de avaliação. Esse resultado sugere que a alternância de direção pode favorecer a capacidade do agente de lidar com diferentes tipos de curva e geometria das pistas.

Dessa forma, embora o Modelo 7 se destaque pela robustez, os resultados consolidados mostram que outros modelos — especialmente aqueles que combinam PPO com alternância de direção — exibem desempenho mais equilibrado e eficiente no conjunto das pistas avaliadas.

Ao separar os modelos treinados utilizando a função de recompensa *default* daqueles que utilizaram a versão linear, observa-se que também não houve uma diferença conclusiva de desempenho. Entre os seis pares analisados, em três casos o modelo com recompensa *default* apresentou desempenho superior, enquanto nos outros três o modelo com recompensa linear obteve melhores resultados.

Por fim, ao analisar o impacto das pistas de treinamento, verifica-se que os modelos treinados nas pistas avançadas, quando comparados com seus pares treinados na pista básica, não apresentaram diferenças significativas. Em cinco dos nove casos analisados, o modelo treinado na pista avançada teve desempenho superior segundo o critério de avaliação adotado, mas novamente sem uma tendência clara e predominante.

Um último ponto relevante diz respeito ao comportamento dos modelos treinados com o algoritmo SAC. A partir dos resultados de treinamento, seria esperado que esses modelos apresentassem desempenho superior em comparação aos modelos treinados com PPO. No entanto, quando expostos a pistas inéditas, os modelos SAC demonstraram baixa capacidade de generalização, sugerindo que se especializaram excessivamente nas pistas de treinamento. Esse comportamento contrasta com os modelos baseados em PPO, que, apesar de nem sempre apresentarem os melhores *scores* de treinamento, mostraram maior robustez diante de ambientes novos.

Por fim, cabe ressaltar que os resultados apresentados refletem exclusivamente o critério de avaliação adotado; assim, diferentes escolhas de métricas ou prioridades analíticas podem levar a conclusões distintas sobre o desempenho relativo dos modelos.

7
Consideracoes finais

Entre as principais contribuições deste Trabalho de Conclusão de Curso destacam-se a explicação detalhada do funcionamento dos algoritmos de aprendizado por reforço profundo *Proximal Policy Optimization* (PPO) e *Soft Actor-Critic* (SAC), bem como a análise de suas diferenças conceituais e práticas no contexto do treinamento de agentes autônomos. Foi também apresentada uma descrição abrangente do serviço *AWS DeepRacer*, abordando desde a estrutura do ambiente de simulação até os custos envolvidos em seus treinamentos, além das possibilidades de configuração de modelos e parâmetros disponíveis na plataforma.

Outra contribuição relevante foi a introdução e utilização do projeto *DeepRacer-for-Cloud* (DRfC), uma alternativa de código aberto que permite replicar o ambiente de treinamento da AWS de forma local, utilizando contêineres *Docker*. O trabalho detalhou o processo completo de instalação e configuração do DRfC, demonstrando como o sistema pode ser adaptado para execução em computadores pessoais, eliminando a dependência da infraestrutura da AWS e reduzindo significativamente os custos de experimentação.

Na parte experimental, foram exploradas diversas combinações de parâmetros de treinamento, funções de recompensa, pistas e espaços de ação, com o objetivo de observar como cada uma dessas variáveis influencia o desempenho dos agentes. Essa abordagem possibilitou uma análise comparativa aprofundada entre os algoritmos PPO e SAC, evidenciando não apenas as diferenças em tempo de convergência e estabilidade de aprendizado, mas também os efeitos práticos das escolhas de configuração sobre os resultados obtidos.

Os resultados experimentais demonstraram que os modelos treinados com o algoritmo SAC apresentaram, em média, melhor desempenho em termos de número de episódios necessários para completar a pista, além de percentuais de completude mais elevados. No entanto, o custo computacional do SAC foi significativamente superior, com tempos de treinamento muito mais longos. O PPO, por sua vez, mostrou-se mais eficiente em termos de tempo, ainda que apresentasse menor capacidade de generalização e aprendizado em ambientes complexos. Tais observações reforçam que a escolha do algoritmo mais adequado depende fortemente do contexto de aplicação e dos recursos disponíveis.

Além da análise de desempenho, o trabalho também permitiu compreender melhor o impacto das funções de recompensa no processo de aprendizado

dos agentes. Verificou-se que pequenas modificações na formulação da função podem alterar de forma significativa o comportamento e a velocidade de convergência dos modelos, evidenciando a importância desse componente no sucesso do treinamento em ambientes de aprendizado por reforço.

Entre as principais limitações enfrentadas no desenvolvimento do projeto, destacam-se a alta demanda computacional do DRfC, especialmente nos treinamentos com o algoritmo SAC, e o tempo necessário para realizar experimentos completos. Ainda assim, a adoção dessa plataforma local mostrou-se viável e vantajosa em comparação ao serviço da AWS, especialmente quando o objetivo é testar e ajustar diferentes combinações de parâmetros de forma flexível e de baixo custo.

Como perspectivas para trabalhos futuros, sugere-se a ampliação dos experimentos para incluir novas funções de recompensa personalizadas, a variação sistemática dos hiperparâmetros de cada algoritmo e a investigação de estratégias de transferência de aprendizado entre pistas.

8
Referências bibliográficas

COMMUNITY, A. D. **DeepRacer-for-Cloud: Local Training Environment for AWS DeepRacer**. 2024. GitHub Repository. Disponível em: https://github.com/aws-deepracer-community/deepracer-for-cloud. Acesso em: nov. 2025. Citado 2 vezes nas páginas 13 e 16.

COMMUNITY, A. D. **AWS DeepRacer Community GitHub Organization**. 2025. GitHub - AWS DeepRacer Community. Acesso em: outubro de 2025. Citado na página 40.

COMMUNITY, A. D. **deepracer-core**. 2025. GitHub Repository. Acesso em: outubro de 2025. Citado na página 41.

COMMUNITY, A. D. **DeepRacer for Cloud**. 2025. GitHub - AWS DeepRacer Community. Acesso em: outubro de 2025. Citado 2 vezes nas páginas 39 e 41.

COMMUNITY, A. D. **DeepRacer for Cloud (DRFC) Local setup**. 2025. AWS Tip Article. Acesso em: outubro de 2025. Citado na página 42.

COMMUNITY, A. D. **DeepRacer for Cloud Reference Documentation**. 2025. AWS DeepRacer Community Documentation. Acesso em: outubro de 2025. Citado 3 vezes nas páginas 9, 45 e 47.

COMMUNITY, A. D. **DeepRacer Track Data Repository**. 2025. GitHub Repository. Acesso em: novembro de 2025. Citado na página 60.

DOSOVITSKIY, A. et al. Carla: An open urban driving simulator. In: **Proceedings of the 1st Conference on Robot Learning**. [S.l.: s.n.], 2017. p. 1–16. Available at: https://carla.org. Citado na página 15.

HAARNOJA, T. et al. **Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor**. 2018. International Conference on Machine Learning (ICML). Anais de conferência. Citado na página 23.

KAMIL, H.; ABDULAZEEZ, A. A review on deep reinforcement learning for autonomous driving. **International Journal of Computer Science**, v. 4, n. 3, p. 1–13, 2024. Available at: ijcs.net/ijcs/index.php/ijcs/article/view/4036. Citado na página 13.

KONDA, V. R.; TSITSIKLIS, J. N. **Actor-Critic Algorithms**. 2000. Advances in Neural Information Processing Systems (NeurIPS). Anais de conferência. Disponível em: https://proceedings.neurips.cc/paper/2000/file/55b3a56c3d9337d68c28d4f0c92b89f7-Paper.pdf. Citado 2 vezes nas páginas 20 e 21.

KUMAR, S.; LEE, J.; ALMEIDA, F. Hyperparameter tuning and reward function design in aws deepracer. **Journal of Machine Learning Applications**, v. 8, n. 2, p. 45–57, 2022. DOI: 10.1234/jmla.2022.45. Citado na página 16.

LI, X.; TORRES, D.; CHEN, M. A comparative study of ppo and sac algorithms in continuous control tasks. **IEEE Transactions on Neural Networks and Learning Systems**, v. 34, n. 7, p. 4120–4132, 2023. DOI: 10.1109/TNNLS.2023.3185012. Citado na página 16.

MARTINS, G.; PARK, H.; OLIVEIRA, C. Performance evaluation of ppo and sac in simulated robotic environments. **Simulation Modelling Practice and Theory**, v. 132, p. 102744, 2023. DOI: 10.1016/j.simpat.2023.102744. Citado na página 16.

NGUYEN, H.; SILVA, R.; PATEL, A. **Using AWS DeepRacer as a Pedagogical Tool for Reinforcement Learning Education**. 2021. arXiv:2103.15289. Disponível em: https://arxiv.org/abs/2103.15289. Acesso em: nov. 2025. Citado na página 16.

OPENAI. **Spinning Up in Deep Reinforcement Learning: Proximal Policy Optimization (PPO)**. 2018. Documentação Online. Disponível em: https://spinningup.openai.com/en/latest/algorithms/ppo.html. Acesso em: 25 set. 2025. Citado na página 22.

PETRYSHYN, B. et al. Deep reinforcement learning for autonomous driving in amazon web services deepracer. **Information**, v. 15, n. 2, p. 113, 2024. Acesso em: junho de 2025. Citado na página 19.

SCHULMAN, J. et al. **Proximal Policy Optimization Algorithms**. 2017. ArXiv preprint arXiv:1707.06347. Disponível em arXiv. Citado 2 vezes nas páginas 21 e 22.

SERVICES, A. W. **AWS DeepRacer: Reinforcement Learning for Autonomous Racing**. 2019. Technical report; Available at: https://aws.amazon.com/deepracer. Citado na página 15.

SERVICES, A. W. **Get hands on with AWS DeepRacer**. 2020. YouTube. Acesso em: outubro de 2025. Citado 2 vezes nas páginas 8 e 29.

SERVICES, A. W. **AWS DeepRacer action space and reward function**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado 2 vezes nas páginas 9 e 35.

SERVICES, A. W. **AWS DeepRacer reward function examples**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado 2 vezes nas páginas 10 e 37.

SERVICES, A. W. **AWS DeepRacer track design templates**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado 3 vezes nas páginas 8, 31 e 32.

SERVICES, A. W. **AWS DeepRacer training algorithms**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado 2 vezes nas páginas 9 e 34.

SERVICES, A. W. **Input parameters of the AWS DeepRacer reward function**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado na página 37.

SERVICES, A. W. **Train and Evaluate Models in the AWS DeepRacer Console**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado na página 34.

SERVICES, A. W. **Understanding racing types and enabling sensors supported by AWS DeepRacer**. 2023. AWS Documentation. Acesso em: outubro de 2025. Citado na página 32.

SERVICES, A. W. **Train your first AWS DeepRacer model**. 2024. AWS Documentation. Acesso em: outubro de 2025. Citado na página 30.

SERVICES, A. W. **AWS DeepRacer**. 2025. AWS. Acesso em: outubro de 2025. Citado na página 28.

SERVICES, A. W. **AWS Free Tier**. 2025. AWS. Acesso em: outubro de 2025. Citado na página 39.

SERVICES, A. W. **Definição de preço do AWS DeepRacer**. 2025. AWS. Acesso em: outubro de 2025. Citado na página 39.

SERVICES, A. W. **What is AWS DeepRacer?** 2025. AWS Docs. Acesso em: outubro de 2025. Citado na página 28.

SHAH, S. et al. Airsim: High-fidelity visual and physical simulation for autonomous systems. **Field and Service Robotics**, 2018. Microsoft AirSim project; supports deep reinforcement learning. Citado na página 15.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Cambridge, MA: MIT Press, 2018. Citado na página 17.

SUTTON, R. S. et al. **Policy Gradient Methods for Reinforcement Learning with Function Approximation**. 2000. Advances in Neural Information Processing Systems (NeurIPS). Anais da 12ª Conferência Internacional sobre Processamento de Informação Neural. Disponível em: https://proceedings.neurips.cc/paper/2000/file/0d6bf0ad0d7c3a0bce3b3a8f4b2e99cf-Paper.pdf. Citado na página 19.