FEDERAL UNIVERSITY OF CEARÁ

QUIXADA CAMPUS

UNDERGRADUATE COURSE IN SOFTWARE ENGINEERING

JOÃO VICTOR CARVALHO DE OLIVEIRA

# STREAMLINING AWS DEEPRACER MODEL TRAINING: A PYTHON LIBRARY FOR JUPYTER NOTEBOOK INTEGRATION WITH DEEPRACER FOR CLOUD

QUIXADÁ

2025

JOÃO VICTOR CARVALHO DE OLIVEIRA

# STREAMLINING AWS DEEPRACER MODEL TRAINING: A PYTHON LIBRARY FOR JUPYTER NOTEBOOK INTEGRATION WITH DEEPRACER FOR CLOUD

Undergraduate Thesis submitted to the Software Engineering Course of the Quixada Campus of the Federal University of Ceará, as a requirement for obtaining the Bachelor Degree in Software Engineering.

Advisor: Prof. Dr. João Marcelo Uchôa de Alencar

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

O47s Oliveira, João Victor Carvalho de.
Streamlining AWS DeepRacer Model Training : A Python Library for Jupyter Notebook Integration with DeepRacer For Cloud / João Victor Carvalho de Oliveira. – 2025.
68 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2025.
Orientação: Prof. Dr. João Marcelo Uchôa de Alencar.

1. Machine Learning. 2. DeepRacer. 3. Amazon Web Services. 4. Shell Scripts. 5. Python. I. Título.
CDD 005.1

---

JOÃO VICTOR CARVALHO DE OLIVEIRA

# STREAMLINING AWS DEEPRACER MODEL TRAINING: A PYTHON LIBRARY FOR JUPYTER NOTEBOOK INTEGRATION WITH DEEPRACER FOR CLOUD

Undergraduate Thesis submitted to the Software Engineering Course of the Quixada Campus of the Federal University of Ceará, as a requirement for obtaining the Bachelor Degree in Software Engineering.

Approved on: 25th July 2025

To God, for granting me the wisdom, strength, and perseverance necessary to overcome each of the challenges of this journey. I acknowledge that the completion of this work is, above all, a reflection of His grace and permission.

This divine care was made manifest through the unconditional love of my family, my foundation and greatest encouragement. To my parents, for investing in my dreams and sustaining me with their constant prayers. To my sister, for the support and companionship that made the path lighter. This work is the fruit of the love, faith, and effort of us all.

# ACKNOWLEDGEMENTS

First, I express my deepest gratitude to God for His infinite grace and providence. The completion of this work is a testament to the strength I found in faith to overcome every obstacle. To Him, I dedicate my gratitude for guiding my steps, transforming effort into achievement.

To my family, my foundation, and safe harbor, I express all my love and appreciation. To my parents, for their unconditional support, sustaining love, and the prayers that always surrounded me. This achievement is the fruit of the trust and investment they placed in me. To my dear sister, I am grateful for being a constant source of inspiration and a benchmark. Walking this academic path with you made everything lighter and more meaningful.

Professor Regis Pires, my sincere thanks Dr. for introducing me to the fascinating field of Artificial Intelligence. His classes and guidance were fundamental in sparking my interest and solidifying my passion for the area, especially through the opportunities in undergraduate research and at the Insight Data Science Laboratory.

Professor Dr. João Marcelo, I am grateful for the trust and for the invitation to represent the Federal University of Ceará in the first AWS DeepRacer competition for students in Brazil and also be the adivisor of this work. The honor of participating and emerging as the champion of this event was a turning point in my academic and professional journey.

This work would not have been the same without the inspiration of the AWS community, especially Lenilson Vilas Boas and RubeM Saldanha. Your encouragement was crucial for the conception of this research and opened the doors to the global community, allowing invaluable interaction with the creators of the solution I studied.

To all those who were, directly or indirectly, a part of this story, my eternal thanks.

## ABSTRACT

The open-source DeepRacer for Cloud (DRfC) framework democratizes access to Reinforcement Learning (RL) experimentation, but its command-line interface imposes a high usability barrier, especially for new users. This thesis aimed to design, implement, and empirically validate a high-level abstraction layer for DRfC to simplify its workflow and improve the user experience. The solution, a Python library named `drfc_manager`, was developed to orchestrate DRfC operations within an interactive Jupyter Notebook environment. Its effectiveness was evaluated through a comparative usability study with 21 participants, who performed a set of standardized tasks using both the proposed library and the traditional script-based method. The results demonstrate that the `drfc_manager` library significantly reduces task completion time; for instance, 'Training Initiation' was completed in under 2 minutes by 85.7% of library users, in contrast to 90.5% of baseline users requiring 5 minutes or more. The solution was perceived as vastly easier to use, culminating in an overall user preference of 95.2% for the library. This work contributes an empirically validated, publicly available open-source tool that reduces the complexity of DRfC, making RL experimentation more accessible and efficient. The findings reinforce the importance of user-centric interfaces for the adoption of complex software platforms.

**Keywords:** machine learning; DeepRacer; Amazon Web Services; shell scripts; Python.

# RESUMO

A ferramenta de código aberto DeepRacer for Cloud (DRfC) democratiza o acesso à experimentação com Aprendizagem por Reforço (Reinforcement Learning - RL), mas sua interface baseada em linha de comando impõe uma alta barreira de usabilidade, especialmente para novos usuários. Este trabalho teve como objetivo projetar, implementar e validar empiricamente uma camada de abstração de alto nível para o DRfC, visando simplificar seu fluxo de trabalho e melhorar a experiência do usuário. A solução, uma biblioteca Python chamada `drfc_manager`, foi desenvolvida para orquestrar as operações do DRfC dentro de um ambiente interativo Jupyter Notebook. A sua eficácia foi avaliada através de um estudo de usabilidade comparativo com 21 participantes, que realizaram um conjunto de tarefas padronizadas utilizando tanto a biblioteca proposta quanto o método tradicional baseado em scripts. Os resultados demonstram que a biblioteca `drfc_manager` reduz significativamente o tempo de conclusão das tarefas; por exemplo, a etapa de 'Iniciação do Treinamento' foi concluída em menos de 2 minutos por 85,7% dos usuários da biblioteca, em contraste com 90,5% dos usuários da linha de base que necessitaram de 5 minutos ou mais. A solução foi percebida como massivamente mais fácil de usar, culminando em uma preferência geral de 95,2% dos participantes pela biblioteca. Este trabalho contribui com uma ferramenta de código aberto, validada empiricamente e publicamente disponível, que reduz a complexidade do DRfC, tornando a experimentação com RL mais acessível e eficiente. Os resultados reforçam a importância de interfaces centradas no usuário para a adoção de plataformas de software complexas.

**Palavras-chave:** aprendizado de máquina; DeepRacer; Amazon Web Services; shell scripts; Python.

## LIST OF ABBREVIATIONS AND ACRONYMS

| | |
|---|---|
| *AWS* | *Amazon Web Services* |
| *DL* | *Deep Learning* |
| *DRL* | *Deep Reinforcement Learning* |
| *DRfC* | *DeepRacer for Cloud* |
| *DR* | *AWS DeepRacer* |
| *LLMS* | *Large Language Models* |
| *ML* | *Machine Learning* |
| *NB* | *Jupyter Notebook* |
| *PPO* | *Proximal Policy Optimization* |
| *RL* | *Reinforcement Learning* |
| *SAC* | *Soft Actor-Critic* |
| *SGM* | *AWS SageMaker* |

# LIST OF SYMBOLS

| | |
|---|---|
| $s$ | State of the agent |
| $a$ | Action performed by the agent |
| $t$ | Time step |
| $S_t$ | State of the agent based on $t$ |
| $A_t$ | Action of the agent based on $t$ |
| $R_t$ | Reward at $t$ based on $A_{t-1}$ and $S_{t-1}$ |
| $\pi$ | Policy |

CONTENTS

# 1 INTRODUCTION

The proliferation of *Deep Learning* (*DL*) has revolutionized numerous fields, powering advances from computer vision to natural language processing. The success of large-scale models, such as the transformer architectures foundational to modern *Large Language Models* (*LLMS*) (Vaswani *et al.*, 2017), has demonstrated the profound capabilities of deep neural networks. However, mastering the practical application of *DL* concepts, particularly in specialized areas like *Reinforcement Learning* (*RL*), presents a significant pedagogical challenge. Bridging the gap between theoretical knowledge and hands-on implementation is crucial for training the next generation of engineers and researchers.

Educational platforms are essential for translating DL theory into practice. One prominent platform is DeepRacer (Amazon Web Services, 2024), an interactive service designed to teach *RL* through a competitive, hands-on approach. DeepRacer employs Challenge-Based Learning Gallagher and Savage (2023), allowing users with basic Python skills to engage with complex AI concepts by training a model for an autonomous race car in a simulated environment.

While effective, the reliance of DeepRacer on the proprietary *Amazon Web Services* (*AWS*) cloud console introduces barriers related to cost and customizability. To address these limitations, the open-source *DeepRacer for Cloud* (*DRfC*) project was developed (AWS Open Source Community, 2024). *DRfC* provides a local, containerized environment that replicates the core functionalities of the *AWS* service, allowing users to train and evaluate models on personal hardware. By utilizing the same underlying simulation and training containers, *DRfC* ensures that models developed locally are functionally equivalent to their cloud-trained counterparts, thus democratizing access to *RL* experimentation.

Despite its success, first-hand observations from the AWS DeepRacer study group at the Federal University of Ceará, corroborated by interactions with the global user community, revealed that the usability of *DRfC* remains a significant hurdle. Its current command-line interface places a heavy burden on the user, requiring manual configuration and lacking integrated tools for monitoring and analysis. This steepens the learning curve and increases the likelihood of user error, particularly in critical tasks such as experiment tracking, hyperparameter tuning, and results visualization. These deficiencies create a fragmented workflow that detracts from the core learning objectives.

To remedy these issues, this thesis proposes the integration of *DRfC* with *Jupyter Notebook* (*NB*). As an interactive computing environment, *NB* is a *de facto* standard for data science and machine learning workflows. Its cell-based execution, support for inline visualizations, and narrative capabilities make it an ideal platform for streamlining the *RL* development process. By creating an abstraction layer over *DRfC*, this work aims to provide a cohesive, programmatic interface within a notebook environment, thus simplifying training, evaluation, and analysis.

## 1.1 Objectives

This section outlines the goals of this work.

### *1.1.1 General*

The primary objective of this work is to design, implement, and evaluate an abstraction layer for the *DRfC* framework to enable its seamless integration with Jupyter Notebooks, thereby improving the user experience and workflow for *RL* experimentation.

### *1.1.2 Specific*

- Propose and implement an integration of *DRfC* with Jupyter Notebooks, allowing users to manage the entire RL workflow within a single environment.
- Develop a Python-based Application Programming Interface (API) that serves as the abstraction layer, providing programmatic access to *DRfC*'s core functionalities.
- Implement features for the real-time tracking and visualization of the training.
- Enhance the system's robustness and usability by incorporating descriptive error reporting and type safety into the API design.

## 1.2 Thesis Structure

This thesis is organized into six chapters. Chapter 2 provides the Theoretical Foundation, covering the core concepts of Reinforcement Learning, the Jupyter Notebook environment, and the AWS DeepRacer architecture. Chapter 3 reviews Related Work, analyzing existing solutions like DRfC and the AWS L400 Workshop to identify the research gap this thesis addresses. Chapter 4 details the Methodology , describing the design and implementation of the `drfc_manager` library and the protocol for the comparative usability study. Chapter 5 presents the Results and Analysis, detailing the quantitative and qualitative findings from the user study. Finally, Chapter 6 offers Conclusions and Future Work, summarizing the contributions, discussing limitations, and outlining a user-driven roadmap for future development. The appendices provide supplementary materials, including a user guide and the consolidated questionnaire data.

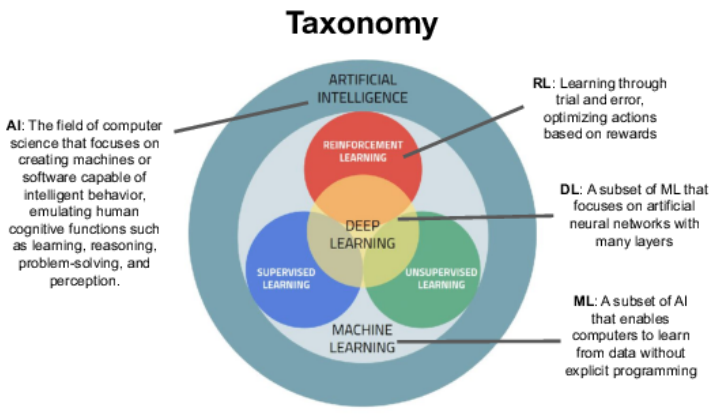
# 2 THEORETICAL FOUNDATION

In this chapter, we will explore the foundational concepts relevant to *AWS DeepRacer* (*DR*), establishing the basis for developing and validating our ideas and conducting experiments. We will start by providing technical definitions of key *RL* terms, then examine their relationship to *DR* and *NB*. Finally, we will explain the architecture of *DR*.

## 2.1 Reinforcement Learning

Reinforcement Learning *RL* is a paradigm of machine learning concerned with how an intelligent **agent** ought to take **actions** in an **environment** to maximize a cumulative reward signal. As illustrated in Figure 1, RL is a distinct subfield of machine learning that often intersects with and leverages techniques from other domains, most notably using deep neural networks from *DL* as powerful function approximators by Li (2018).

Figure 1 – Venn diagram illustrating the relationships between AI subfields.


Taxonomy
ARTIFICIAL INTELLIGENCE
AI: The field of computer science that focuses on creating machines or software capable of intelligent behavior, emulating human cognitive functions such as learning, reasoning, problem-solving, and perception.
RL: Learning through trial and error, optimizing actions based on rewards
REINFORCEMENT LEARNING
DEEP LEARNING
DL: A subset of ML that focuses on artificial neural networks with many layers
SUPERVISED LEARNING
UNSUPERVISED LEARNING
ML: A subset of AI that enables computers to learn from data without explicit programming
MACHINE LEARNING

Source: AI4EIC Working Group (2023)

The fundamental learning mechanism in *RL* is direct interaction. The agent observes the current **state** of the environment, selects an action based on its **policy**, and in return, receives a numerical **reward** and transitions to a new state. This trial-and-error process, known as the agent-environment interaction loop, allows the agent to learn a strategy, or policy, that yields the most reward over time by Sutton and Barto (1998). This approach is particularly well-suited for problems involving sequential decision-making under uncertainty, such as game playing, robotics, and autonomous vehicle control, where the state-action space is vast and optimal behavior must be discovered through exploration.

### *2.1.1 Definition*

Formally, an *RL* problem can be modeled as a **Markov Decision Process (MDP)** by Garcia and Rachelson (2013), which is defined by a tuple $(S, A, P, R, \gamma)$. We will define the core components based on the framework provided by Sutton and Barto (1998) and Li (2018):

- **Agent:** The learner and decision-maker that interacts with the environment.
- **Environment:** The external world with which the agent interacts, comprising everything outside the agent.
- **State** $(S_t)$**:** A complete description of the state of the environment at a discrete time step $t$. A state is said to possess the **Markov Property** if it contains all relevant information from the history of interaction to determine future transitions.
- **Action** $(A_t)$**:** One of the choices available to the agent. The set of all available actions in a state $S_t$ is denoted by $A(S_t)$.
- **Policy** $(\pi)$**:** The agent's strategy, which maps states to actions. A policy can be deterministic, $\pi(s) = a$, or stochastic, $\pi(a|s) = P[A_t = a|S_t = s]$, providing a probability for each action in a given state.
- **Reward** $(R_{t+1})$**:** A scalar feedback signal received by the agent after transitioning from state $S_t$ to $S_{t+1}$ as a result of action $A_t$. The agent's objective is to maximize the total cumulative reward.

### *2.1.2 The Agent-Environment Interaction Loop*

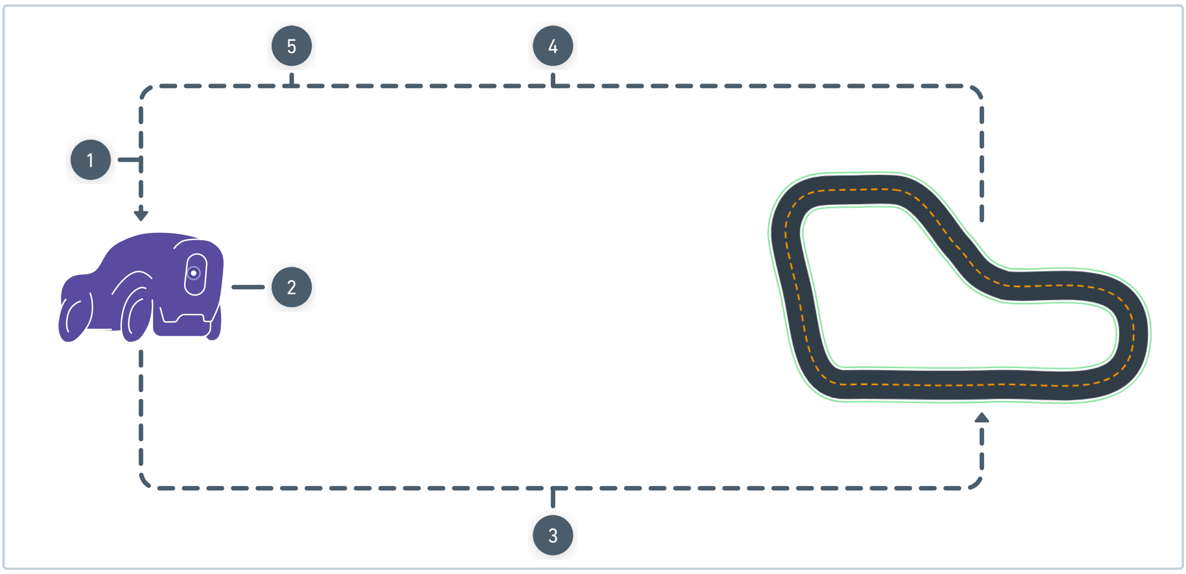
The learning process in RL is characterized by a continuous, iterative feedback loop between the agent and its environment. This interaction, depicted in Figure 2, forms the basis for how an agent learns from experience. According to the foundational framework established by Sutton and Barto (2018), this loop can be broken down into the following sequence of events at each time step $t$:

1. **Observe State:** The agent observes the environment's current state, $S_t$.
2. **Select Action:** Based on state $S_t$, the agent selects an action $A_t$ according to its current policy, $\pi(A_t|S_t)$.
3. **Interact and Receive Feedback:** The agent executes action $A_t$. In response, the environment transitions to a new state, $S_{t+1}$, and provides a scalar reward, $R_{t+1}$, as feedback for the preceding action.

4. **Learn from Experience:** The agent uses the resulting experience tuple—$(S_t, A_t, R_{t+1}, S_{t+1})$—to update its internal policy $\pi$.
5. **Iterate:** The process repeats from the new state $S_{t+1}$, forming a trajectory of states, actions, and rewards from which the agent continuously learns.

Through this iterative cycle, the agent's policy gradually improves by managing the trade-off between exploring new actions and exploiting known good ones to maximize its long-term reward.

Figure 2 – The RL agent-environment interaction loop applied to DeepRacer. The car (agent) takes actions (steering, throttle) in the racetrack (environment) and receives new states (sensor data) and rewards to update its policy.



Source: The author.

This abstract loop maps directly to the training process of the *DR* vehicle, as detailed by the official AWS documentation (Amazon Web Services, 2025a):

1. **Initial State:** The car begins a training episode and receives its initial state $S_0$ from its sensors.
2. **Action Selection:** Using its neural network policy, the car selects an action $A_0$ based on the input state $S_0$.
3. **Environment Interaction:** The car executes action $A_0$. The simulation engine updates the car's position, resulting in a new state $S_1$ and a reward $R_1$.
4. **Policy Update:** The car's learning algorithm, such as *Proximal Policy Optimization* (*PPO*) or *Soft Actor-Critic* (*SAC*), uses the collected experience tuple $(S_0, A_0, R_1, S_1)$ to update

the weights of its policy network.

### *2.1.3 Policy Learning Strategies*

The policy, $\pi$, is the agent's strategy for mapping states to actions. A central challenge in finding an optimal policy, $\pi^*$, is managing the **exploration-exploitation trade-off** (Sutton; Barto, 2018). **Exploration** is the process of trying new, uncertain actions to discover more information about the environment. **Exploitation** involves choosing actions that are already known to yield high rewards. An agent must explore to avoid converging on a suboptimal strategy, but it must exploit its knowledge to maximize performance. The distinction between on-policy and off-policy algorithms represents different philosophies for managing this fundamental trade-off.

#### *2.1.3.1 On-Policy vs. Off-Policy Learning*

The distinction between on-policy and off-policy algorithms lies in the source of the data used for policy updates.

**On-policy** algorithms learn from experience generated by the *current* policy being executed. The agent follows policy $\pi$, collects experience, updates $\pi$ to $\pi'$, and then discards the old data. This creates a direct feedback loop where the policy is improved based on its own performance. While this can lead to stable and straightforward convergence, it is often **sample inefficient**, as each piece of experience is used for only one update (Schulman *et al.*, 2017).

**Off-policy** algorithms, in contrast, learn from experience generated by *any* policy. The foundational off-policy algorithm, **Q-Learning** by Watkins (1989), learns an action-value function (the Q-function) that estimates the long-term reward of taking an action from a given state. Because it directly learns the optimal Q-function, it can do so using data collected from any behavior policy. This established the principle of learning from a **replay buffer**, a large history of past experiences. Modern off-policy methods build on this by sampling from the buffer to update the current "target policy," making them significantly more **sample efficient** (Haarnoja *et al.*, 2018).

#### *2.1.3.2 DeepRacer Policy Algorithms*

The algorithms used by *DR*, *PPO*, and *SAC*, are state-of-the-art *Deep Reinforcement Learning* (*DRL*) methods. They employ deep neural networks to approximate the policy and/or value functions, enabling them to handle the complex, high-dimensional state spaces derived from the vehicle's sensors. Table 1 provides a brief classification of these and other common *DRL* algorithms.

Table 1 – Classification of common Reinforcement Learning algorithms.

| **Algorithm** | **Learning Strategy** | **Policy Type** |
|---|---|---|
| PPO by Schulman *et al.* (2017) | On-Policy | Stochastic |
| REINFORCE by Williams (1992) | On-Policy | Stochastic |
| A3C by Mnih *et al.* (2016) | On-Policy | Stochastic |
| SAC by Haarnoja *et al.* (2018) | Off-Policy | Stochastic |
| DDPG by Lillicrap *et al.* (2015) | Off-Policy | Deterministic |

Source: Author

- **PPO (Proximal Policy Optimization):**
  PPO is a state-of-the-art *on-policy* algorithm renowned for its stability and reliability. It uses a neural network to represent the policy directly. Its key innovation, as detailed by Schulman *et al.* (2017), is the **clipped surrogate objective function**. This mechanism prevents the policy from changing too drastically in a single update by constraining the size of the policy ratio between iterations. By ensuring smoother updates, *PPO* mitigates the instability of older on-policy methods.
- **SAC (Soft Actor-Critic):**
  SAC is a modern *off-policy* algorithm that uses multiple neural networks: an **actor** network to represent the policy and **critic** networks to estimate value functions. Its core principle is **maximum entropy reinforcement learning** (Haarnoja *et al.*, 2018). The algorithm modifies the standard *RL* objective to maximize not only the expected cumulative reward but also the policy's entropy. This entropy bonus explicitly encourages exploration, making the algorithm highly sample-efficient and adept at solving complex control tasks.

The table 2 provides a summary comparison of these two algorithms.

Table 2 – Comparison between *PPO* and *SAC*

| **Attribute** | ***PPO*** | ***SAC*** |
|---|---|---|
| **Sample Efficiency** | Less efficient. Learns from data generated by the current policy only. | More efficient. Reuses past experiences from a replay buffer. |
| **Stability & Tuning** | Generally more stable and less sensitive to hyperparameters. Simpler to implement. | Can be more sensitive to hyperparameters but may achieve higher asymptotic performance. |
| **Exploration** | Exploration is driven by the stochasticity of the policy itself. | Exploration is explicitly encouraged by maximizing the policy's entropy. |
| **Complexity** | Conceptually simpler, built on a clipped surrogate objective function. | More complex, involving actor, critic, and value networks, plus an entropy term. |

Source: The author, based on Schulman *et al.* (2017) and Haarnoja *et al.* (2018).

### *2.1.4 The Reward Function and Reward Shaping*

In reinforcement learning, the goal of the agent is communicated through a single scalar signal: the **reward**. The reward function defines the immediate desirability of being in a particular state or taking a specific action. As stated by Sutton and Barto (2018), the reward signal is the agent's sole objective; its goal is to maximize the cumulative reward it receives in the long run.

In many complex problems, providing a reward only upon task completion (e.g., only when a lap is finished) results in a **sparse reward** problem, making learning extremely slow or intractable. To address this, designers often engineer the reward function to provide more frequent, intermediate feedback that guides the agent toward the desired behavior. This practice is formally known as **reward shaping** (Ng *et al.*, 1999).

The core idea of reward shaping is to augment the environment's intrinsic reward with an additional, artificial reward potential. This shaped reward, $R'$, is given by:

$$R'(s,a,s') = R(s,a,s') + F(s') - F(s)$$

where $R$ is the original reward and $F$ is a potential function defined over states. The foundational work by Ng *et al.* (1999) proved that this form of potential-based reward shaping is guaranteed to preserve the optimal policy of the original problem, thereby guiding the agent without inadvertently changing the task's fundamental goal. Designing an effective reward function is thus a practical exercise in reward shaping.

In the context of *DR*, the developer acts as the designer, creating a reward function using up to 23 input parameters provided by the simulation environment[1]. These parameters—such as distance_from_center, speed, steering_angle, and is_offtrack—serve as the basis for building a potential function that guides the car.

The following Python code, adapted from the *DR* documentation, provides a simple but effective example of potential-based reward shaping.

```python
def reward_function(params):
    # Read input parameters
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']

    # Define a potential based on distance from the center
    # Closer to the center has higher potential (higher reward)
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Assign reward based on tiered proximity to the center line
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # Likely off-track

    return float(reward)
```

Algorithm 1 – Example of a reward function for center-line following.

This function directly shapes the agent's behavior by providing immediate, positive feedback for staying near the track's center, a heuristic that is highly correlated with completing a lap successfully. The careful crafting of this function is one of the most critical aspects of achieving high performance in *DR*.

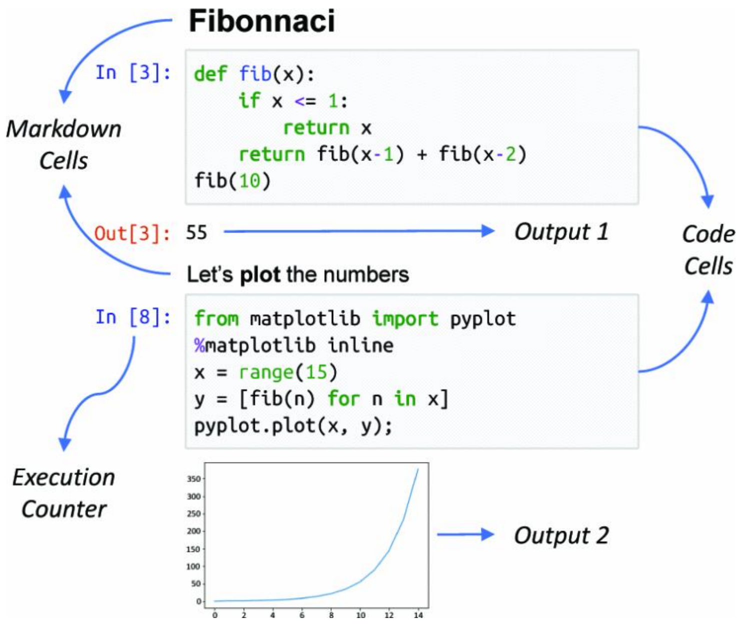
[1] <https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-reward-function-input.html>

## 2.2 Jupyter Notebook

The Jupyter Notebook (*NB*) is an open-source, interactive web application that allows users to create and share documents containing live code, equations, visualizations, and narrative text by Perkel (2018). Its design is a modern application of Donald Knuth's literate programming paradigm, which advocates for weaving documentation and source code together to create a more intelligible and human-centered representation of a program. This approach enhances clarity and reproducibility by intertwining executable code with the narrative and reasoning behind it.

A *NB* is structured into individual "cells" which can contain either executable code or formatted text using Markdown. This cell-based structure is ideal for iterative development and data analysis, as it allows for rapid prototyping and immediate visualization of results. While over 40 programming languages are supported, Python has emerged as the *de facto* standard for data science and machine learning workflows within this environment (Pérez *et al.*, 2014). Figure 3 shows a typical notebook interface.

Figure 3 – An example of a Jupyter Notebook, demonstrating the integration of a Markdown cell for text, a code cell for Python, and its corresponding output.



```
Fibonnaci

In [3]: def fib(x):
            if x <= 1:
                return x
            return fib(x-1) + fib(x-2)
        fib(10)

Out[3]: 55

Let's plot the numbers

In [8]: from matplotlib import pyplot
        %matplotlib inline
        x = range(15)
        y = [fib(n) for n in x]
        pyplot.plot(x, y);
```

Source: Pimentel *et al.* (2019)

The motivation for integrating *NB* in this work stems from the significant operational complexity of the standard *DRfC* workflow. The existing process requires users to manage numerous command-line flags, manually edit configuration files, and execute separate scripts for training, evaluation,and analysis. This fragmented and error-prone process presents a steep learning curve and detracts from the primary goal of learning reinforcement learning concepts.

The utilization of *NB* in this work aims to enhance the transparency of the model training process and associated tasks. By integrating *NB*, the focus is on reducing the complexity of configuration required for training and increasing the explainability of the tasks performed. While the primary focus of this work centers on simplifying the training process by minimizing configuration, relying solely on a Python script may not fully exploit the explanatory benefits provided by notebooks. Furthermore, the incorporation of visualization tools, which might prove challenging to manage exclusively within a Python script, becomes more manageable within the notebook environment.

## 2.3 AWS DeepRacer

The architecture of AWS DeepRacer comprises five key services, as illustrated in Figure 4: Amazon SageMaker, S3, RoboMaker, CloudWatch, and Kinesis Video Streams. Each of these components plays a vital role in the service workflow. In the following sections, we will delve into the communication process and discuss the significance of each component.

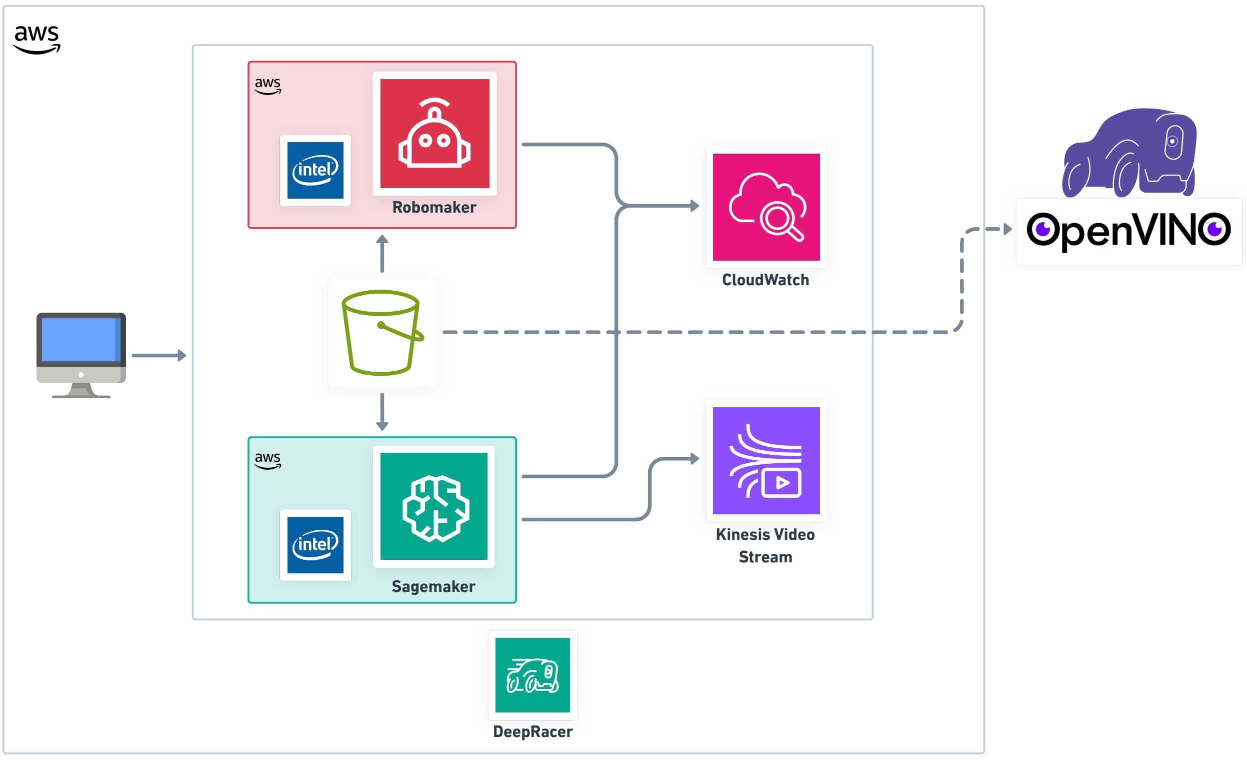
### *2.3.1 Components*

The section overviews the components used in the AWS DeepRacer ecosystem.

#### *2.3.1.1 Amazon SageMaker*

Amazon SageMaker is a fully managed service that accelerates the ability of developers and data scientists to build, train, and deploy machine learning models. In the AWS DeepRacer ecosystem, Amazon SageMaker is the tool of choice for creating and training the machine learning models that drive the DeepRacer cars autonomously. It simplifies the machine-learning workflow by providing built-in algorithms and managing infrastructure, which allows developers to focus on the model's logic. Once the models are trained, they can be deployed directly from Amazon SageMaker to the DeepRacer console for testing in simulations or on the physical car.

Figure 4 – DeepRacer Architecture



Source: Adapted from Services (2019)

### *2.3.1.2 AWS RoboMaker*

AWS RoboMaker is a service that aids developers in building, testing, and deploying intelligent robotics applications at scale. It enhances the Robot Operating System (ROS) framework with AWS services connectivity. For AWS DeepRacer, AWS RoboMaker offers a simulation environment replicating real-world racing tracks, enabling developers to test their reinforcement learning models in a virtual setting before deploying them to the physical car, thus ensuring optimized performance under various conditions.

### *2.3.1.3 Amazon Simple Storage Service (S3)*

Amazon Simple Storage Service (S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. It is the data repository for AWS DeepRacer, storing all the necessary training data, model artifacts, and log files. This provides a scalable and secure storage solution for the large amounts of data involved in training and evaluating reinforcement learning models.

#### *2.3.1.4 Amazon Kinesis Video Streams*

Amazon Kinesis Video Streams facilitates the secure streaming of video from connected devices to AWS for analytics, machine learning, and other processing. Within the AWS DeepRacer framework, Amazon Kinesis Video Streams is used to stream live video feeds from DeepRacer cars during races or training sessions, enabling real-time monitoring and analysis of the car's performance.

#### *2.3.1.5 Amazon CloudWatch*

Amazon CloudWatch is a monitoring and observability service that provides data and actionable insights to monitor applications, respond to system-wide performance changes, and get a unified view of operational health. In the AWS DeepRacer context, Amazon CloudWatch is instrumental in monitoring and logging various metrics and events, helping developers track the performance of the DeepRacer models and providing insights into the training and racing processes.

### ***2.3.2 Communication***

The components outlined previously are integrated to constitute the AWS DeepRacer system, as depicted in Figure 4. Communication between these components is essential for the DeepRacer's operation.

1. **Data Storage (Step 1):** The foundational training data, encompassing the simulation environment (track), reward function, model metadata (such as action space), and hyperparameters, are securely housed in Amazon Simple Storage Service (S3).
2. **Initiation of Training (Step 2):** Amazon SageMaker commences the training job by employing the selected reinforcement learning algorithm. It retrieves the requisite data from Amazon Simple Storage Service (S3) to initiate the training process and allocates the necessary computational resources.
3. **Simulation Execution (Step 3):** AWS RoboMaker initiates a simulation job, sourcing track configurations and agent specifications from Amazon Simple Storage Service (S3). It then establishes the simulation environment, allowing the agent to commence interaction.

4. **Agent Interaction (Step 4):** As the simulation unfolds, the agent executes actions, prompting updates within the environment. The accumulated experiences, comprising state, action, reward, and subsequent state, are intermittently transmitted back to Amazon SageMaker for model refinement.
5. **Model Enhancement (Step 5):** Amazon SageMaker periodically archives the enhanced neural network models into Amazon Simple Storage Service (S3), ensuring that the most recent iterations are preserved and retrievable.
6. **Performance Monitoring (Step 6):** AWS RoboMaker dispatches performance metrics, logs, and ancillary data to Amazon CloudWatch. This enables users to oversee the training's progression and the utilization of resources via the AWS console.
7. **Live Streaming (Step 7):** Real-time video feeds from the simulation are relayed to Amazon Kinesis Video Streams, facilitating immediate monitoring and analytical review of the simulation.
8. **Model Preservation (Step 8):** The AWS DeepRacer service intermittently commits the trained model to Amazon Simple Storage Service (S3), ensuring its enduring preservation.
9. **Concluding Training (Step 9):** The training endeavor ceases upon reaching predetermined endpoints, such as a time threshold. The ultimate model rendition is then stored within Amazon Simple Storage Service (S3).
10. **Model Assessment (Step 10):** Post-training, the cultivated model is extracted from Amazon Simple Storage Service (S3) and subjected to evaluation within a simulated setting by AWS RoboMaker. The performance metrics are scrutinized to gauge the model's efficacy.

The seamless communication between these components ensures the efficient operation of the AWS DeepRacer system. Each step, from data storage to model assessment, relies on precise and timely data exchange. This orchestration not only facilitates the training and refinement of the model but also enables real-time monitoring and evaluation, ensuring that the AWS DeepRacer system operates at its optimal potential. The integration of these services underscores the robustness and scalability of the AWS infrastructure, making it a powerful platform for developing and deploying advanced machine-learning models.

# 3 RELATED WORK

This chapter analyzes existing solutions and practices related to the AWS DeepRacer ecosystem. We examine three key areas: the community-driven open-source alternative for local training (*DRfC*), an official AWS workshop that uses notebooks for low-level configuration, and industry best practices for user-friendly machine learning workflows in Jupyter Notebooks. This analysis will establish the context and identify the specific gap in functionality and usability that the present thesis aims to fill.

## 3.1 DeepRacer for Cloud (DRfC)

Developed by AWS Open Source Community (2024), *DRfC* is an open-source framework that addresses two primary limitations of the official AWS DeepRacer service: high operational costs and limited control over the training environment. The official service incurs costs for both training and evaluation, which can be substantial. For instance, pricing can be upwards of **$3.50 per hour for training and $8.00 per hour for evaluation** from Amazon Web Services (2025b), making extensive experimentation prohibitively expensive. *DRfC* mitigates this by enabling users to run all associated services on local hardware or any cloud provider (IaaS), shifting the cost model to standard compute resources.

*DRfC* provides a powerful command-line interface for managing the entire training lifecycle. Users have fine-grained control over hyperparameters, the reward function, and even the neural network architecture by choosing between configurations like Deep Convolutional Network (DCN), DCN-Shallow, or DCN-Deep. It also supports advanced features such as distributed rollouts using a swarm mode, which significantly accelerates training by parallelizing the simulation across multiple workers.

However, the primary user interface for *DRfC* is the command line and manual editing of configuration files (e.g., YAML). While this offers maximum flexibility, it also presents a steep learning curve and introduces numerous potential failure points from misconfiguration. The user is responsible for managing a complex set of scripts and parameters, which can be cumbersome and counter-intuitive for those whose main goal is to learn and apply reinforcement learning concepts rather than manage infrastructure.

## 3.2 AWS DeepRacer Workshop L400

The *AWS* DeepRacer Workshop L400 on Services (2024) is an advanced, official guide from AWS that demonstrates how to orchestrate the DeepRacer training process using core AWS services like *AWS SageMaker* (*SGM*). A key feature of this workshop is its use of a Jupyter Notebook as the primary interface for managing the workflow.

While it successfully integrates notebooks, the approach taken is that of a low-level infrastructure script. The notebook is filled with boilerplate code for managing *AWS* resources, such as building and pushing Docker images, setting environment variables, configuring VPC network settings, and defining *SGM* jobs via raw Python SDK calls and YAML adjustments. This results in a user experience that, while centralized in a notebook, remains highly complex and exposes implementation details that are irrelevant to the learning task. It solves the problem of a fragmented workflow but fails to provide a high-level, user-friendly abstraction. It also operates entirely within the AWS ecosystem, inheriting the same high-cost model as the standard service.

## 3.3 Jupyter Notebook Best Practices

The use of Jupyter Notebooks as a high-level interface for complex machine learning tasks is a well-established best practice in the industry. This is demonstrated by several prominent examples:

- **Google Vertex AI**: Google's flagship *Machine Learning* (*ML*) platform provides a repository of sample notebooks by googlevertexai that abstract away underlying infrastructure. Users interact with a high-level Python API to define, train, and deploy models, rather than managing containers or virtual machines directly.
- **Facebook's Segment Anything Model (SAM)**: The official repository for SAM by Kirillov *et al.* (2023) uses notebooks to provide a clean, interactive way to load the pre-trained model and experiment with its capabilities. The focus is on the model's application, not its operational deployment.
- **Fastai Fastbook**: As a leading educational resource, the Fastai book by Howard and Gugger (2020) is entirely built on Jupyter Notebooks. It provides a high-level API that allows learners to build and train state-of-the-art deep learning models in just a few lines of code, hiding the complex PyTorch implementation details.

These examples establish a clear precedent: the most effective and user-friendly ML tools provide high-level, abstracted interfaces within a notebook environment, allowing users to focus on the "what" (the model and logic) rather than the "how" (the infrastructure and configuration).

## 3.4 Comparative Analysis

The related works reviewed highlight a significant gap: there is no solution in the DeepRacer ecosystem that combines the low-cost, open-source flexibility of *DRfC* with the user-friendly, high-level, interactive workflow established as a best practice by tools like Vertex AI and Fastai. The AWS L400 workshop attempts to use notebooks but does so in a low-level manner that does not sufficiently reduce complexity for the end-user.

This thesis directly addresses this gap. Table 3 presents a comparative analysis of the existing solutions against the proposed work, using a set of specific technical attributes that underscore the novelty of our contribution.

Table 3 – Comparative analysis of related works based on technical attributes.

| **Attribute** | **DRfC** | **AWS Workshop L400** | **Industry Notebooks** | **This work** |
|---|---|---|---|---|
| **User Interface** | Command Line | Notebook (Low-Level) | Notebook (High-Level) | **Notebook (High-Level)** |
| **Config. Method** | JSON Files | Python SDK/ YAML | High-Level API | **High-Level API** |
| **Cost Model** | Local | Pay-per-use Service | N/A | **Local** |
| **Cloud-Agnostic** | ✓ | | ✓ | ✓ |
| **Abstraction Level** | Low | Low | High | **High** |
| **Requires Docker Expertise** | ✓ | ✓ | | |

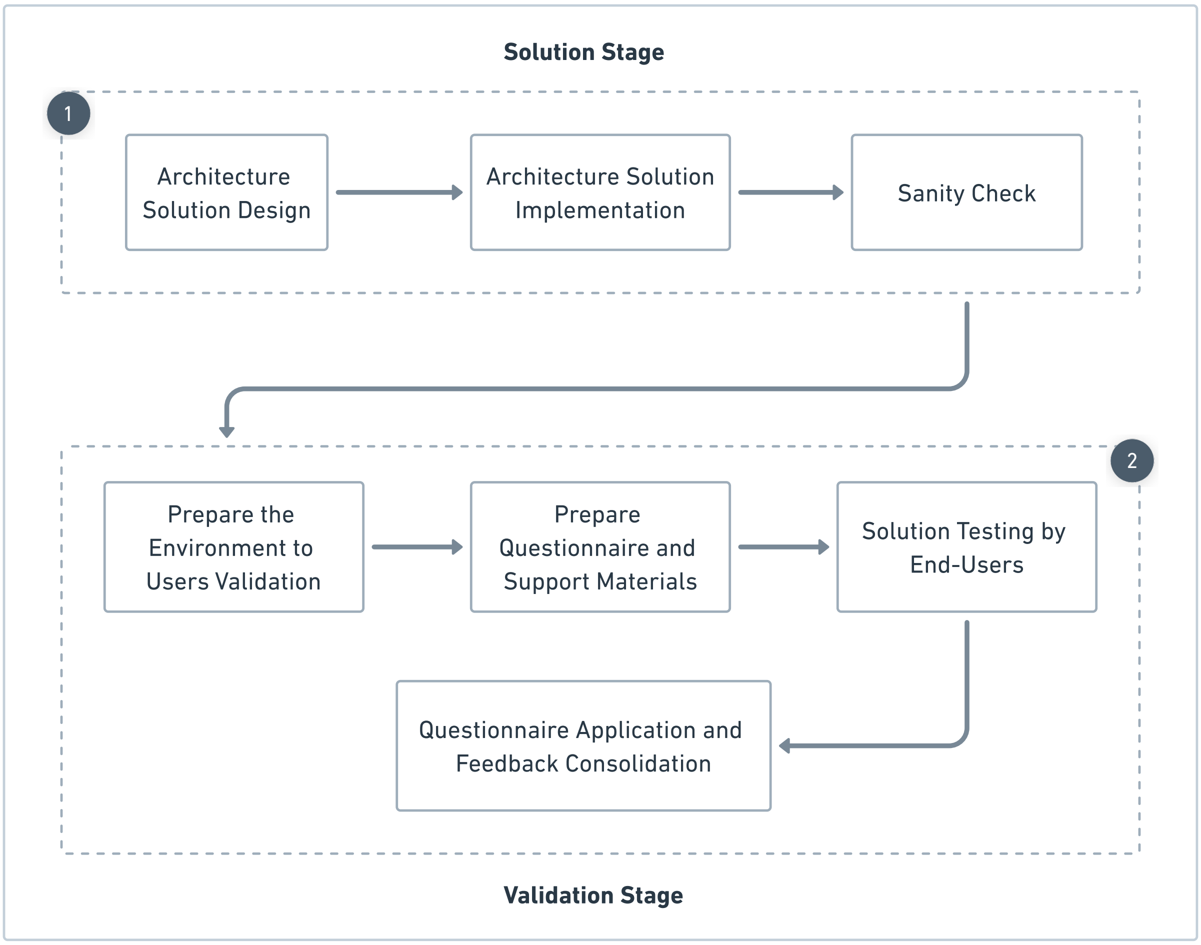
Source: Author

As the analysis shows, the solution developed in this thesis is the first to provide a high-level, abstracted, and cloud-agnostic interface for the DeepRacer ecosystem, combining the benefits of all reviewed approaches while mitigating their primary drawbacks.

# 4 METHODOLOGY

This chapter outlines the methodological procedures employed to develop and validate the proposed solution. The process is divided into two primary stages: the **Solution Stage**, where the drfc_manager Python library was designed and implemented, and the **Validation Stage**, where a formal user study was conducted to compare the usability of the proposed solution against the baseline *DRfC* workflow. Figure 5 illustrates these overarching stages.

Figure 5 – The two primary stages of the methodology: Solution and Validation.


Solution Stage
1
Architecture Solution Design
Architecture Solution Implementation
Sanity Check
2
Prepare the Environment to Users Validation
Prepare Questionnaire and Support Materials
Solution Testing by End-Users
Questionnaire Application and Feedback Consolidation
Validation Stage

Source: The author.

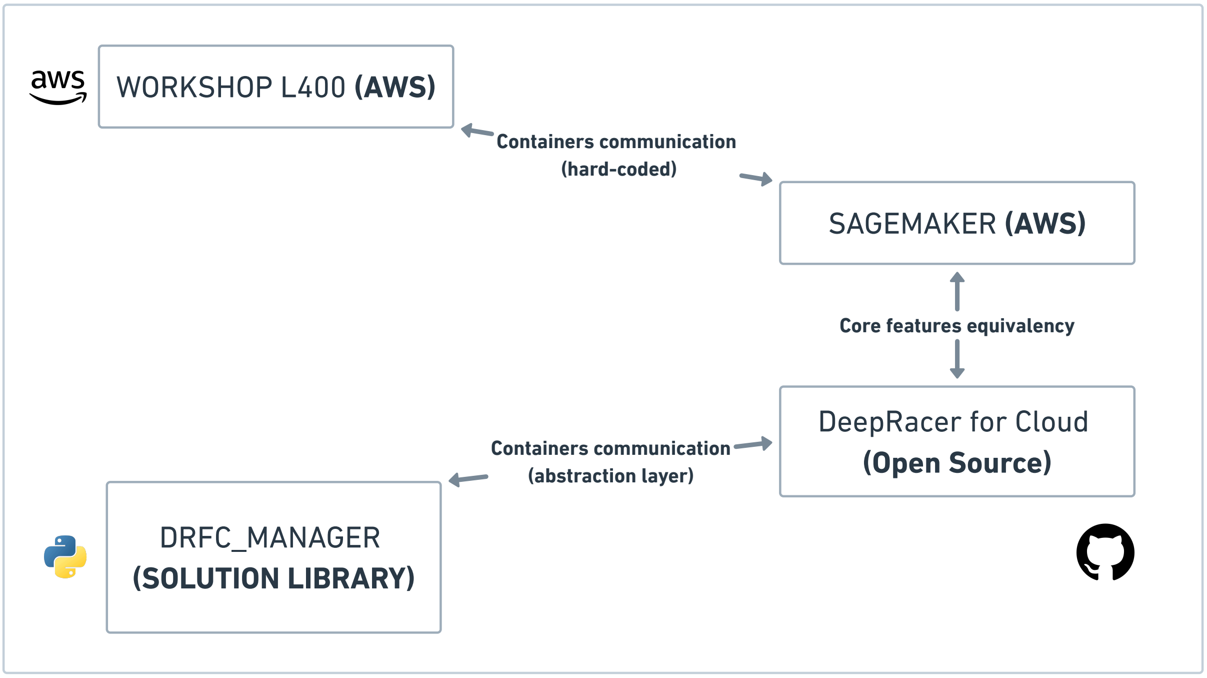
## 4.1 Solution Stage: The `drfc_manager` Library

The core technical contribution of this undergraduate thesis is a Python library named `drfc_manager`. The following sections detail the architectural design and implementation of this library.

### *4.1.1 Architecture Solution Design*

The fundamental purpose of `drfc_manager` is to serve as a high-level abstraction layer over the low-level functionalities of *DRfC*. As shown in Figure 6, this approach is designed to simplify the user workflow significantly. While the AWS Workshop L400 also uses a notebook, it does so by exposing low-level service configurations, whereas our solution provides a clean, high-level API.

Figure 6 – Comparison between the low-level notebook approach (Workshop L400) and the proposed high-level library abstraction.


WORKSHOP L400 (AWS)
Containers communication (hard-coded)
SAGEMAKER (AWS)
Core features equivalency
DeepRacer for Cloud (Open Source)
Containers communication (abstraction layer)
DRFC_MANAGER (SOLUTION LIBRARY)

Source: *

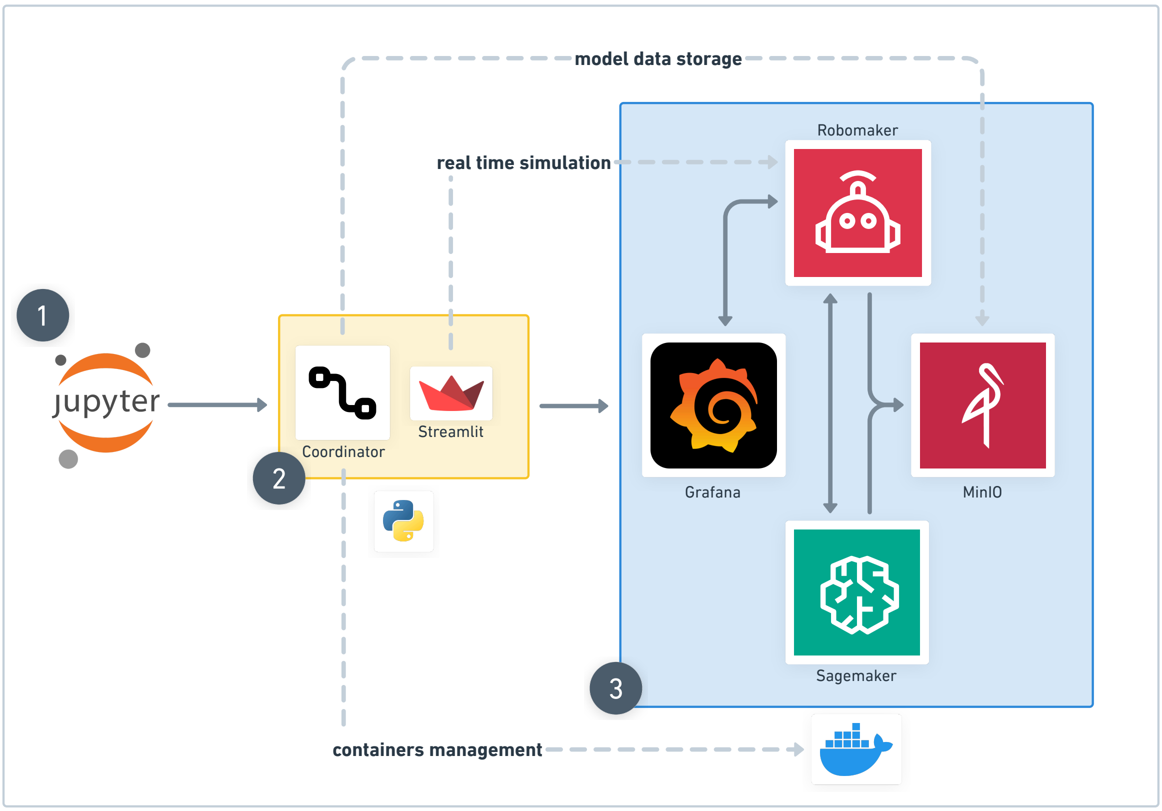
The author

The library's architecture (Figure 7) was designed to integrate cleanly with the existing *DRfC* containers (RoboMaker, SageMaker, Minio, etc.) without altering their validated, community-vetted behavior. The library's role is not to change what *DRfC* does, but to radically simplify *how* a user interacts with it.

This solution was engineered with key software engineering principles in mind to address the observed shortcomings of the standard DRfC script-based workflow:

- **User-Friendly Interface:** Provide an intuitive Python API to replace the complex and error-prone reliance on shell scripts and manual file editing.
- **Security:** Encapsulate interactions with the host system (e.g., Docker daemon) to prevent

Figure 7 – The proposed three-module architecture for the `drfc_manager` solution.



Source: The author

user error from impacting system stability.

- **Type Safety:** Enforce strict data types for all inputs to prevent configuration errors at the source.
- **Observability & Error Reporting:** Provide clear, descriptive error messages and suggestions instead of cryptic script failures.
- **Progress Tracking:** Offer a structured way to monitor the status of long-running processes such as model training.

### *4.1.2 Architecture Solution Implementation*

This section explains the proposed architecture, its components, their integration, and the technologies used for implementation, as shown in Figure 7. The implementation follows the SOLID principles by Martin (2003) and the Test-Driven Development (TDD) methodology by Martin (2008) to ensure a robust and maintainable system.

#### *4.1.2.1 Module 1 - Jupyter Notebook*

The Jupyter Notebook module leverages the Python library to organize all available workflows. This includes defining model hyperparameters, metadata, and the reward function used during model training. In addition, it manages the execution of various pipelines such as `start/stop training`, `start/stop evaluation`, `start/stop metrics`, and `start/stop simulation viewer`.

The use of Jupyter Notebooks is justified by the numerous benefits they offer, as mentioned in section 2.2, including support for future enhancements. For example, they facilitate the integration of *ML* algorithms to enhance the training of *RL* models.

#### *4.1.2.2 Module 2 - Python Library*

We propose a Python library called `drfc_manager` library that will consist of two sub-modules: the Software Development Kits (SDKs) and Streamlit[1], a Python visualization library. This proposed library aims to encapsulate all existing functionalities from *DRfC* and provide an easy-to-use, secure, type-safe, and error-resilient interface for optimizing model training. As discussed in the previous section, the library conceptualizes *DRfC* workflows as pipelines, where actions follow sequential logical steps.

Streamlit plays a crucial role in presenting real-time simulations generated by Robomaker. It provides an interactive and dynamic interface for visualizing data, model outputs, and other relevant information during training.

Although the specific SDKs used are not depicted in Figure 7, it's essential to understand that they facilitate communication with the Docker daemon and the Minio container. The latter is crucial because any files users send via `drfc_manager` pipelines (from Jupyter Notebook) must be stored for subsequent model usage. Therefore, the `drfc_manager` module handles file storage and provides relevant data (such as log data) after model training for further analysis.

Additionally, communication with the Docker SDK is vital. The tasks optimized by the library rely on container communication. This includes starting or stopping containers, accessing their logs, opening multiple instances, and supporting multiple hosts to maximize training and balance resource loads. While some of these tasks can be accomplished through the

[1] <https://github.com/streamlit/streamlit>

Docker CLI, the `drfc_manager` library offers additional guarantees and ease of use.

#### *4.1.2.3 Module 3 - Docker Containers*

The AWS Open Source Community[2] built the Docker containers based on the official containers from AWS services. Their behavior was designed to be safe and has been validated by many users and AWS employees, which also aids in their maintenance. The code within the container's image is available and can be customized by anyone.

The solution doesn't modify these containers internally; instead, it utilizes them to ensure that the role of the `drfc_manager` is not to alter the existing behavior (thus avoiding the introduction of vulnerabilities) but to streamline the workflow.

### ***4.1.3 Sanity Check***

The sanity check process ensures that the proposed architecture is correctly set up and operational before moving forward. This involves the following steps:

#### *4.1.3.1 Environment Setup*

Start by creating a virtual machine with the necessary specifications. Install the Docker environment and set up the *DRfC* framework within this virtual environment to ensure that all tools and dependencies are correctly configured.

#### *4.1.3.2 Service Initialization*

Load all the services defined in the architecture. This includes:

- Launching the Docker containers and verifying their functionality.
- Ensuring that components such as the `drfc_manager` library and Jupyter Notebook module are properly integrated.
- Establishing communication pathways between components.

#### *4.1.3.3 System Validation*

Conduct a series of preliminary checks to confirm that the overall system is stable and functioning as intended. This step involves:

[2] https://github.com/aws-deepracer-community

- Validating that workflows, such as starting and stopping training pipelines, work seamlessly within the architecture.

#### *4.1.3.4 Final Verification*

After the initial validation, monitor the system for any anomalies or unexpected behavior. Ensure that all components, including external integrations like Streamlit for data visualization, are fully operational.

## 4.2 Validation Methodology: A Comparative Usability Study

To empirically validate the hypothesis that the `drfc_manager` library provides a more usable and efficient interface than the standard DRfC workflow, a **comparative usability study** was designed and conducted.

This study directly compares two user experiences for performing the same set of DeepRacer tasks:

1. **The Baseline Condition:** Users interact with the standard *DRfC* framework using its native command-line shell scripts and by manually editing configuration files.
2. **The Proposed Solution Condition:** Users interact with the `drfc_manager` Python library within a Jupyter Notebook environment to accomplish the exact same tasks.

The following sections detail the methodology used for this comparative study.

### *4.2.1 Participant Recruitment*

Participants for the study were recruited from two primary pools: experienced users from the national (Brazilian Community) AWS DeepRacer community Discord server and students from the Federal University of Ceará (UFC) at Brazil. This provided a mix of expert users, who are familiar with the baseline workflow's pain points, and novice users, who could provide insight into the initial learning curve of both systems.

### *4.2.2 Task Design*

To ensure a direct and fair comparison, a standardized workflow composed of 10 distinct tasks was defined. These tasks represent the end-to-end process of configuring and running a DeepRacer experiment. The questionnaire was structured around these 10 tasks, prompting users to evaluate each one individually. The defined tasks were:

1. Environment Setup
2. Metadata Configuration
3. Hyperparameter Definition
4. Training Initiation
5. Status Monitoring
6. Live Video Visualization
7. Training Finalization
8. Model Cloning
9. Model Evaluation
10. Error Handling

### *4.2.3 Data Collection and Metrics*

To quantify the difference in user experience, a mixed-methods approach was used, gathering data directly through the structured online questionnaire. The metrics were designed to capture both performance and perception for each of the 10 defined tasks.

#### *4.2.3.1 Quantitative Metrics*

- **Self-Reported Task Completion Time:** For each task, participants were asked to report the approximate time (in minutes) it took to complete the task using both the baseline *DRfC* scripts and the proposed drfc_manager library.

#### *4.2.3.2 Qualitative and Ordinal Metrics*

- **Comparative Ease-of-Use Rating:** The core metric for perceived usability. For each task, participants rated the comparative difficulty on a 5-point Likert scale, where the endpoints were defined as:

  **1** = Much easier in *DRfC* vs **5** = Much easier in `drfc_manager`

  A score of 3 indicated neutral or comparable difficulty.
- **Overall Preference:** A final question asked participants to indicate their overall preferred tool after completing the entire workflow.

- **Open-Ended Feedback:** Participants were given the opportunity to provide unstructured, free-form comments about their experience, frustrations, and suggestions for improvement.

### *4.2.4 Procedure*

Participants were provided with the testing environments and a link to the online questionnaire hosted on Google Forms. The questionnaire itself structured the experimental procedure. It guided each participant through the 10-task workflow sequentially.

For each task, participants were instructed to first attempt it using the baseline *DRfC* command-line method and then perform the identical task using the proposed drfc_manager library within a Jupyter Notebook. Immediately after using both methods for a given task, they recorded the comparative ease-of-use rating and the self-reported completion times in the corresponding section of the form before proceeding to the next task.

Upon completing all tasks, the final section of the questionnaire collected their overall preference and open-ended feedback. The anonymous responses were then consolidated for analysis.

# 5 RESULTS AND ANALYSIS

This chapter presents the empirical findings from the comparative usability study outlined in the methodology. The study was designed to evaluate the effectiveness of the proposed `drfc_manager` library against the baseline command-line workflow of the *DRfC* framework. The results are organized according to the quantitative and qualitative metrics collected from the 21 study participants, providing a comprehensive analysis of user performance and perception.

## 5.1 Participant Demographics

A total of 21 participants completed the usability study. The cohort was composed of two main groups: 18 participants (85.7%) identified as experienced users from the AWS DeepRacer community, while 3 participants (14.3%) were students from the Federal University of Ceará, representing novice users. Within the novice group, 6 participants were completely new to the entire DeepRacer ecosystem, making their feedback particularly valuable for evaluating the initial learning curve.

## 5.2 Qualitative and Ordinal Analysis

The qualitative and ordinal data reveals a decisive user preference for the `drfc_manager` library. This was measured through a 5-point comparative ease-of-use rating for each task and a final overall preference question.

### *5.2.1 Comparative Ease-of-Use*

Participants were asked to rate the comparative ease of completing each task on a 5-point scale. The full distribution of these ratings is presented in Table 4. The data shows a strong skew in favor of the `drfc_manager` library, particularly for the most complex tasks.

For the three **Configuration Tasks**, a combined average of 67.4% of users rated the library as "Easier" (4) or "Much Easier" (5). The preference was most pronounced for **"Training Initiation,"** which received the highest possible score of "5" from a remarkable **81%** of participants.

Interestingly, the task of **"Error Handling"** was a significant outlier. A plurality of users (52.6%) found the baseline DRfC scripts easier (scores 1 and 2). This counterintuitive result is directly explained by the thematic analysis in Section **??**, where users linked the library's lack of status feedback to a confusing error-handling experience.

Table 4 – Distribution of Comparative Ease-of-Use Ratings per Task

| | **Rating (% of Users)** | | | | |
|---|---|---|---|---|---|
| **Task** | **1** | **2** | **3** | **4** | **5** |
| *Configuration Tasks* | | | | | |
| Env. Setup | 9.5 | 9.5 | 14.3 | 23.8 | 42.9 |
| Metadata Config. | 0.0 | 14.3 | 19.0 | 38.1 | 28.6 |
| Hyperparameter Def. | 0.0 | 9.5 | 19.0 | 38.1 | 28.6 |
| *Core Workflow Tasks* | | | | | |
| Training Initiation | 0.0 | 0.0 | 9.5 | 9.5 | 81.0 |
| Status Monitoring | 0.0 | 9.5 | 23.8 | 38.1 | 28.6 |
| Training Finalization | 9.5 | 9.5 | 14.3 | 14.3 | 52.4 |
| *Advanced and Ancillary Tasks* | | | | | |
| Video Visualization | 5.0 | 10.0 | 15.0 | 40.0 | 30.0 |
| Model Cloning | 0.0 | 10.0 | 15.0 | 25.0 | 50.0 |
| Model Evaluation | 9.5 | 9.5 | 19.0 | 23.8 | 38.1 |
| Error Handling | 36.8 | 15.8 | 0.0 | 31.6 | 15.8 |

Note: Ratings on a 5-point scale where 1 = Much easier with DRFC and 5 = Much easier with `drfc_manager`.
Source: Author

### 5.2.2 *Overall User Preference*

This strong task-level preference culminated in the final overall assessment. When asked which tool they would prefer to continue working with, an overwhelming **95.2%** of participants chose `drfc_manager` over the baseline DRfC scripts.

## 5.3 Quantitative Analysis: Task Completion Time

The self-reported time data reveals a stark and consistent reduction in task completion time with the `drfc_manager` library. Table 5 details the full distribution of completion times for both solutions across all tasks.

The library's efficiency gains were most pronounced in the critical setup and execution phases. For "**Metadata Configuration**," 95.2% of users took 5 minutes or more with the baseline scripts; with the library, 66.7% of users finished in under 2 minutes. For "**Training Initiation**," the library enabled 85.7% of users to start in under 2 minutes, while 90.5% of baseline users required 5 minutes or more.

Table 5 – Distribution of Self-Reported Task Completion Times

| | **drfc_manager (% of Users)** | | | | **DRFC (% of Users)** | | | |
|---|---|---|---|---|---|---|---|---|
| **Task** | < 2 min | 5-10 | 10-20 | > 20 | < 2 min | 5-10 | 10-20 | > 20 |
| *Configuration Tasks* | | | | | | | | |
| Env. Setup | 47.6% | 33.3% | 14.3% | 4.8% | 28.6% | 23.8% | 28.6% | 19.0% |
| Metadata | 66.7% | 23.8% | 9.5% | 0.0% | 4.8% | 71.4% | 23.8% | 0.0% |
| Hyperparameter | 76.2% | 19.0% | 4.8% | 0.0% | 47.6% | 52.4% | 0.0% | 0.0% |
| *Core Workflow Tasks* | | | | | | | | |
| Training Init. | 85.7% | 9.5% | 4.8% | 0.0% | 4.8% | 61.9% | 28.6% | 4.8% |
| Status Monitor | 60.0% | 40.0% | 0.0% | 0.0% | 33.3% | 57.1% | 9.5% | 0.0% |
| Training Final. | 81.0% | 19.0% | 0.0% | 0.0% | 57.1% | 38.1% | 4.8% | 0.0% |
| *Advanced and Ancillary Tasks* | | | | | | | | |
| Video View | 65.0% | 30.0% | 5.0% | 0.0% | 30.0% | 55.0% | 10.0% | 5.0% |
| Model Cloning | 60.0% | 35.0% | 5.0% | 0.0% | 25.0% | 60.0% | 15.0% | 0.0% |
| Model Eval. | 52.4% | 33.3% | 9.5% | 4.8% | 28.6% | 57.1% | 9.5% | 4.8% |
| Error Handling | 30.0% | 40.0% | 30.0% | 0.0% | 30.0% | 40.0% | 25.0% | 5.0% |

Source: Author

### *5.3.1 Thematic Analysis of User Feedback*

The 15 participants who left open-ended comments provided a rich dataset for understanding the "why" behind the numbers. The feedback clustered into three primary themes.

#### *5.3.1.1 Positive Theme: A Radically Simplified and Accelerated Workflow*

Users consistently praised the library for its simplicity and efficiency. Comments described the solution as "blatantly easier to use than DRfC" and "very practical and effective". One novice user highlighted the impact on learning: "I had never used either solution before, but the learning process with drfc_manager proved to be much faster". An expert user noted the reduction in cognitive load, stating the library's "linearity is extremely useful. Just by looking at it, we can already see where we should go".

#### *5.3.1.2 Constructive Theme: Critical Need for Improved Observability*

This was the most prominent area for improvement. Multiple users reported frustration with long-running processes that provided no feedback on their status. One user detailed the experience: "...the model took a while to evaluate and gave no indication that it was still evaluating or that it had finished. This caused confusion...". This was a common sentiment, with another stating the need for an "alert that a process is still ongoing... and being notified when it's complete".

#### *5.3.1.3 Constructive Theme: Desire for More In-line Guidance and Better UX*

Participants also suggested improvements to in-context help. Several requested "more suggestive explanations for the steps" and to have available hyperparameters listed directly, "instead of accessing external documentation". One user also suggested a higher-level GUI that might "simulate the original Deepracer system".

## 5.4 Summary of Findings

The empirical results from the comparative usability study robustly support the initial hypothesis. The quantitative data shows that the `drfc_manager` library makes the DeepRacer workflow significantly faster across all stages. Concurrently, the qualitative data confirms that users perceive the library as far easier and more intuitive to use, culminating in a final overall preference of 95.2% for the proposed solution. The detailed user feedback not only validates the work but also provides a clear roadmap for future improvements, focusing on enhancing status monitoring and in-line documentation.

# 6 CONCLUSIONS AND FUTURE WORK

This thesis addressed a significant usability gap in the AWS DeepRacer for Community (DRFC) framework. The standard workflow, reliant on shell scripts and manual configuration, presents a steep learning curve and is prone to user error. The primary objective was to design, implement, and empirically validate a solution that abstracts this complexity into a user-friendly, high-level interface.

The developed solution, the `drfc_manager` Python library, was engineered to streamline the entire DRFC workflow within a Jupyter Notebook. The subsequent comparative usability study provided robust evidence that this new abstraction layer not only accelerates the experimental process but is also overwhelmingly preferred by users. This final chapter summarizes the key contributions, acknowledges limitations, and outlines a promising, user-driven direction for future development.

## 6.1 Summary of Contributions

- **Development and Distribution of the `drfc_manager` Library:** The core technical contribution is a novel, high-level Python library (Oliveira, 2025) that encapsulates the low-level operations of the DRFC framework. The library was published as an open-source project on GitHub and is installable via Pip, ensuring its accessibility and potential for future community-driven maintenance and enhancement. By applying key software engineering principles, the library successfully transforms a cumbersome command-line process into an intuitive, programmatic workflow. The appendix A presents a guide on how to use the solution.
- **Empirical Validation of Usability and Efficiency:** This work provided rigorous, empirical validation of the solution. The comparative usability study demonstrated, with both quantitative time metrics and qualitative ease-of-use ratings, the library's superiority over the baseline. The findings culminated in a decisive overall user preference, with **90.5%** of participants choosing the `drfc_manager` library.
- **A User-Informed Development Roadmap:** The thematic analysis of user feedback not only validated the library's positive impact but also provided a clear, user-driven roadmap. By identifying specific needs—such as improved observability for long-running tasks and more in-line guidance—this research has laid the groundwork for the next iteration of

development.

## 6.2 Limitations

- **Sample Size:** The usability study was conducted with 21 participants. While this provided statistically significant trends, a larger sample would be necessary to generalize the findings more broadly.
- **Self-Reported Time Metrics:** Task completion times were self-reported and categorical. While sufficient for demonstrating a clear trend, automated logging of precise interaction times would provide more granular efficiency metrics.
- **Fixed Task Order and Learning Effects:** The study was conducted with a fixed task order, where all participants used the baseline DRFC tool before using the `drfc_manager` library. This design does not control for learning effects, as participants may have performed tasks faster with the second tool simply due to familiarity. While the scale of the observed improvements suggests a genuine usability difference, future studies should employ a counterbalanced design (randomizing the order of the tools) to isolate this variable.
- **Scope of Functionality:** The library was intentionally designed to simplify the *existing* DRFC workflow, not to introduce fundamentally new Reinforcement Learning capabilities.

## 6.3 Future Work

The success of the library and the constructive user feedback open several promising avenues for future work, which directly address the themes identified in the results analysis.

### *6.3.1 Addressing User Feedback: Observability and Guidance*

The most immediate next steps involve resolving the key usability issues identified by participants.

- **Improved Observability:** To solve the primary issue raised by users—the lack of status indicators for long-running processes—a key priority is to implement a robust monitoring system. This would involve integrating real-time progress bars and completion alerts directly within the notebook, as requested by multiple users.
- **Enhanced In-Notebook Guidance:** To act on the desire for more contextual help, the library and its documentation will be augmented with more in-line guidance. This includes

adding comments in notebook templates that suggest safe default ranges for hyperparameters, directly addressing the user suggestion to reduce reliance on external documentation.

### *6.3.2 Development of a High-Level Graphical User Interface (GUI)*

While the library empowers users comfortable with Python, some feedback expressed a desire for an even simpler interface. The existing API can serve as the perfect backend for a complete graphical user interface.

- A web-based frontend could be developed, consuming the library's API to allow users to manage experiments through an intuitive point-and-click interface. This would directly address the user suggestion for "an interface almost simulating the original Deepracer system", making the platform accessible to an even broader audience with no programming background.

### *6.3.3 Integration of Automated Machine Learning (AutoML) Techniques*

With the core workflow simplified, the next frontier is to introduce intelligence into the experimentation process itself.

- **Hyperparameter Optimization (HPO):** The current workflow simplifies the *manual* setting of hyperparameters. A future version could automate this process entirely by integrating established HPO libraries (e.g., Optuna, Hyperopt) to automatically search for the optimal set of hyperparameters, leading to better model performance with less manual effort.

## 6.4 Final Conclusion

This thesis successfully demonstrated that abstracting a complex, script-based workflow into a high-level Python library significantly improves usability and efficiency. The empirical evidence, culminating in a 95.2% user preference for the `drfc_manager` solution, validates that a user-centric design is critical for the adoption and effective use of powerful open-source platforms. While acknowledging its limitations, the true value of this work lies in the clear, user-validated roadmap it provides. The feedback has illuminated a path forward—a path that begins with immediate enhancements to observability and guidance, progresses toward a full graphical interface, and ultimately aims to transform the tool from a simple facilitator into an intelligent partner through AutoML. This work contributes a robust and validated solution to the DeepRacer community and opens the door to a more accessible and efficient future for applied Reinforcement Learning experimentation.

## REFERENCES

AI4EIC Working Group. **Taxonomy: A diagrammatic representation of artificial intelligence, machine learning**. 2023. ResearchGate. Available at: https://www.researchgate.net/figure/Taxonomy-A-diagrammatic-representation-of-artificial-intelligence-machine-learning-and_fig1_372415967. Accessed on: 1 July 2025.

Amazon Web Services. **Amazon DeepRacer - Overview**. 2024. Available at: https://aws.amazon.com/deepracer/. Accessed on: 2024.

Amazon Web Services. **AWS DeepRacer Developer Guide**. 2025. Available at: https://docs.aws.amazon.com/deepracer/latest/developerguide/what-is-deepracer.html. Accessed on: 14 July 2025.

Amazon Web Services. **AWS DeepRacer Pricing**. 2025. Available at: https://aws.amazon.com/deepracer/pricing/. Accessed on: 14 July 2025.

AWS Open Source Community. **DeepRacer for Cloud**. 2024. Available at: https://github.com/aws-deepracer-community/deepracer-for-cloud. Accessed on: 2024.

GALLAGHER, S. E.; SAVAGE, T. Challenge-based learning in higher education: an exploratory literature review. **Teaching in Higher Education**, SRHE Website, v. 28, n. 6, p. 1135–1157, 2023. Available at: https://doi.org/10.1080/13562517.2020.1863354. Accessed on: 2024.

GARCIA, F.; RACHELSON, E. **Markov Decision Processes**. John Wiley Sons, Ltd, 2013. 1–38 p. ISBN 9781118557426. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118557426.ch1. Accessed on: 2024.

HAARNOJA, T.; ZHOU, A.; HARTIKAINEN, K.; TUCKER, G.; HA, S.; TAN, J.; KUMAR, V.; ZHU, H.; GUPTA, A.; ABBEEL, P.; LEVINE, S. Soft actor–critic algorithms and applications. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 35., 2018, Stockholm, Sweden. **Proceedings of the 35th International Conference on Machine Learning**. Stockholm: PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1861–1870.

HOWARD, J.; GUGGER, S. **Deep Learning for Coders with Fastai and Pytorch**: Ai applications without a phd. O'Reilly Media, Incorporated, 2020. ISBN 9781492045526. Available at: https://books.google.no/books?id=xd6LxgEACAAJ. Accessed on: 28 May 2024.

KIRILLOV, A.; MINTUN, E.; RAVI, N.; MAO, H.; ROLLAND, C.; GUSTAFSON, L.; XIAO, T.; WHITEHEAD, S.; BERG, A. C.; LO, W.-Y.; DOLLÁR, P.; GIRSHICK, R. Segment anything. **arXiv:2304.02643**, 2023.

LI, Y. **Deep Reinforcement Learning: An Overview**. 2018.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015.

MARTIN, R. C. **Agile Software Development**: Principles, patterns, and practices. USA: Prentice Hall PTR, 2003. ISBN 0135974445.

MARTIN, R. C. **Clean Code**: A handbook of agile software craftsmanship. 1. ed. USA: Prentice Hall PTR, 2008. ISBN 0132350882.

MNIH, V.; BADIA, A. P.; MIRZA, M.; GRAVES, A.; LILLICRAP, T.; HARLEY, T.; SILVER, D.; KAVUKCUOGLU, K. Asynchronous methods for deep reinforcement learning. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 33., 2016, New York, USA. **Proceedings of the 33rd International Conference on Machine Learning**. New York: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1928–1937.

NG, A. Y.; HARADA, D.; RUSSELL, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 16., 1999, Bled, Slovenia. **Proceedings of the Sixteenth International Conference on Machine Learning**. Bled: Morgan Kaufmann, 1999. p. 278–287.

OLIVEIRA, J. V. C. de. **DRFC Manager - Python Library**. 2025. Available at: https://github.com/joaocarvoli/drfc-manager. Accessed on: 15 July 2025.

PERKEL, J. M. Why jupyter is data scientists' computational notebook of choice. **Nature**, v. 563, p. 145+, Nov 2018. ISSN 00280836. 7732. Available at: https://link.gale.com/apps/doc/A573082717/HRCA?u=anon 6bb2d3a1sid=googleScholarxid=225831ee. Accessed on: 2024.

PIMENTEL, J. F.; MURTA, L.; BRAGANHOLO, V.; FREIRE, J. A large-scale study about quality and reproducibility of jupyter notebooks. In: **Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)**. [S.l.: s.n.], 2019. p. 507–517.

PéREZ, F.; GRANGER, B.; JUPYTER, P. **Project Jupyter**. 2014. Version 7.2. Available at: https://jupyter.org. Accessed on: 2024.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017.

SERVICES, A. W. Get hands-on with AWS DeepRacer and compete in the AWS DeepRacer League - Chicago AWS Summit. 2019. Available at: https://bit.ly/get-handson-with-aws-deepracer-and-compete-in-the-aws-deepracer-league. Accessed on: 28 May 2024.

SERVICES, A. W. **Workshop L400 - DeepRacer**. 2024. Available at: https://catalog.us-east-1.prod.workshops.aws/workshops/66473261-de66-42a1-b280-3e0ec87aee26/en-US. Accessed on: 20 May 2024.

SUTTON, R.; BARTO, A. **Reinforcement Learning**: An introduction. MIT Press, 1998. (A Bradford book). ISBN 9780262193986. Available at: https://books.google.com.br/books?id=CAFR6IBF4xYC. Accessed on: 2025.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning**: An introduction. 2nd. ed. [S.l.]: The MIT Press, 2018.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. **CoRR**, abs/1706.03762, 2017. Available at: http://arxiv.org/abs/1706.03762.

WATKINS, C. J. C. H. **Learning from delayed rewards**. Phd Thesis (PhD Thesis) — King's College, Cambridge, 1989.

WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. **Machine learning**, Springer, v. 8, n. 3-4, p. 229–256, 1992.

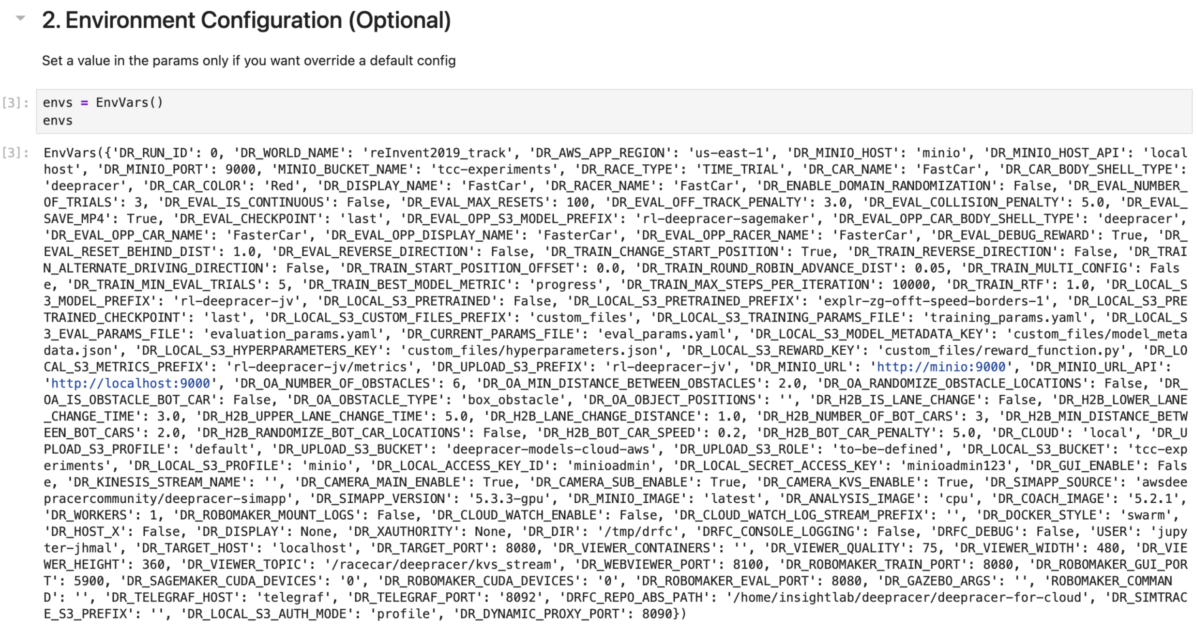
# A USER GUIDE FOR THE `DRFC_MANAGER` LIBRARY

This appendix provides practical examples of how to use the `drfc_manager` library to perform the main tasks of the DeepRacer workflow within a Jupyter Notebook environment. Each section corresponds to a task evaluated in the usability study.

## A.1 Environment Setup

To initialize the environment variables, the user imports the necessary library components and instantiates. It can change whatever he wants. This step is optional because if nothing is passed the default values will be used.

Figure 8 – Example of initializing the environment with the `drfc_manager` library.



2. Environment Configuration (Optional)

Set a value in the params only if you want override a default config

```
[3]: envs = EnvVars()
     envs
```

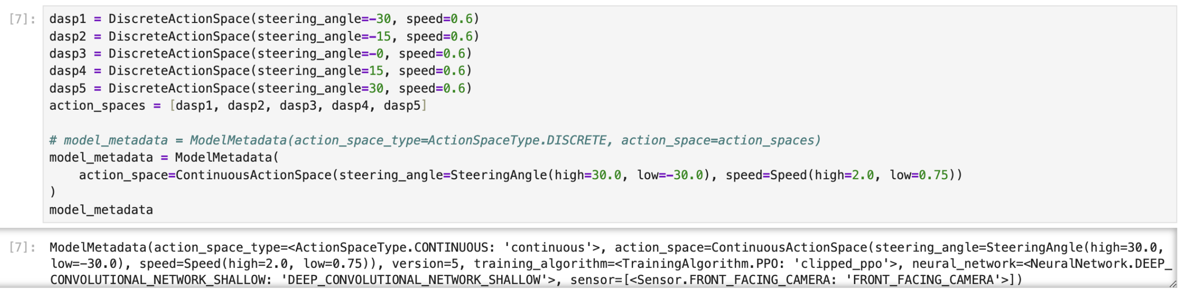
```
[3]: EnvVars({'DR_RUN_ID': 0, 'DR_WORLD_NAME': 'reInvent2019_track', 'DR_AWS_APP_REGION': 'us-east-1', 'DR_MINIO_HOST': 'minio', 'DR_MINIO_HOST_API': 'local
     host', 'DR_MINIO_PORT': 9000, 'MINIO_BUCKET_NAME': 'tcc-experiments', 'DR_RACE_TYPE': 'TIME_TRIAL', 'DR_CAR_NAME': 'FastCar', 'DR_CAR_BODY_SHELL_TYPE':
     'deepracer', 'DR_CAR_COLOR': 'Red', 'DR_DISPLAY_NAME': 'FastCar', 'DR_RACER_NAME': 'FastCar', 'DR_ENABLE_DOMAIN_RANDOMIZATION': False, 'DR_EVAL_NUMBER_
     OF_TRIALS': 3, 'DR_EVAL_IS_CONTINUOUS': False, 'DR_EVAL_MAX_RESETS': 100, 'DR_EVAL_OFF_TRACK_PENALTY': 3.0, 'DR_EVAL_COLLISION_PENALTY': 5.0, 'DR_EVAL_
     SAVE_MP4': True, 'DR_EVAL_CHECKPOINT': 'last', 'DR_EVAL_OPP_S3_MODEL_PREFIX': 'rl-deepracer-sagemaker', 'DR_EVAL_OPP_CAR_BODY_SHELL_TYPE': 'deepracer',
     'DR_EVAL_OPP_CAR_NAME': 'FasterCar', 'DR_EVAL_OPP_DISPLAY_NAME': 'FasterCar', 'DR_EVAL_OPP_RACER_NAME': 'FasterCar', 'DR_EVAL_DEBUG_REWARD': True, 'DR_
     EVAL_RESET_BEHIND_DIST': 1.0, 'DR_EVAL_REVERSE_DIRECTION': False, 'DR_TRAIN_CHANGE_START_POSITION': True, 'DR_TRAIN_REVERSE_DIRECTION': False, 'DR_TRAI
     N_ALTERNATE_DRIVING_DIRECTION': False, 'DR_TRAIN_START_POSITION_OFFSET': 0.0, 'DR_TRAIN_ROUND_ROBIN_ADVANCE_DIST': 0.05, 'DR_TRAIN_MULTI_CONFIG': Fals
     e, 'DR_TRAIN_MIN_EVAL_TRIALS': 5, 'DR_TRAIN_BEST_MODEL_METRIC': 'progress', 'DR_TRAIN_MAX_STEPS_PER_ITERATION': 10000, 'DR_TRAIN_RTF': 1.0, 'DR_LOCAL_S
     3_MODEL_PREFIX': 'rl-deepracer-jv', 'DR_LOCAL_S3_PRETRAINED': False, 'DR_LOCAL_S3_PRETRAINED_PREFIX': 'explr-zg-offt-speed-borders-1', 'DR_LOCAL_S3_PRE
     TRAINED_CHECKPOINT': 'last', 'DR_LOCAL_S3_CUSTOM_FILES_PREFIX': 'custom_files', 'DR_LOCAL_S3_TRAINING_PARAMS_FILE': 'training_params.yaml', 'DR_LOCAL_S
     3_EVAL_PARAMS_FILE': 'evaluation_params.yaml', 'DR_CURRENT_PARAMS_FILE': 'eval_params.yaml', 'DR_LOCAL_S3_MODEL_METADATA_KEY': 'custom_files/model_meta
     data.json', 'DR_LOCAL_S3_HYPERPARAMETERS_KEY': 'custom_files/hyperparameters.json', 'DR_LOCAL_S3_REWARD_KEY': 'custom_files/reward_function.py', 'DR_LO
     CAL_S3_METRICS_PREFIX': 'rl-deepracer-jv/metrics', 'DR_UPLOAD_S3_PREFIX': 'rl-deepracer-jv', 'DR_MINIO_URL': 'http://minio:9000', 'DR_MINIO_URL_API':
     'http://localhost:9000', 'DR_OA_NUMBER_OF_OBSTACLES': 6, 'DR_OA_MIN_DISTANCE_BETWEEN_OBSTACLES': 2.0, 'DR_OA_RANDOMIZE_OBSTACLE_LOCATIONS': False, 'DR_
     OA_IS_OBSTACLE_BOT_CAR': False, 'DR_OA_OBSTACLE_TYPE': 'box_obstacle', 'DR_OA_OBJECT_POSITIONS': '', 'DR_H2B_IS_LANE_CHANGE': False, 'DR_H2B_LOWER_LANE
     _CHANGE_TIME': 3.0, 'DR_H2B_UPPER_LANE_CHANGE_TIME': 5.0, 'DR_H2B_LANE_CHANGE_DISTANCE': 1.0, 'DR_H2B_NUMBER_OF_BOT_CARS': 3, 'DR_H2B_MIN_DISTANCE_BETW
     EEN_BOT_CARS': 2.0, 'DR_H2B_RANDOMIZE_BOT_CAR_LOCATIONS': False, 'DR_H2B_BOT_CAR_SPEED': 0.2, 'DR_H2B_BOT_CAR_PENALTY': 5.0, 'DR_CLOUD': 'local', 'DR_U
     PLOAD_S3_PROFILE': 'default', 'DR_UPLOAD_S3_BUCKET': 'deepracer-models-cloud-aws', 'DR_UPLOAD_S3_ROLE': 'to-be-defined', 'DR_LOCAL_S3_BUCKET': 'tcc-exp
     eriments', 'DR_LOCAL_S3_PROFILE': 'minio', 'DR_LOCAL_ACCESS_KEY_ID': 'minioadmin', 'DR_LOCAL_SECRET_ACCESS_KEY': 'minioadmin123', 'DR_GUI_ENABLE': Fals
     e, 'DR_KINESIS_STREAM_NAME': '', 'DR_CAMERA_MAIN_ENABLE': True, 'DR_CAMERA_SUB_ENABLE': True, 'DR_CAMERA_KVS_ENABLE': True, 'DR_SIMAPP_SOURCE': 'awsdee
     pracercommunity/deepracer-simapp', 'DR_SIMAPP_VERSION': '5.3.3-gpu', 'DR_MINIO_IMAGE': 'latest', 'DR_ANALYSIS_IMAGE': 'cpu', 'DR_COACH_IMAGE': '5.2.1',
     'DR_WORKERS': 1, 'DR_ROBOMAKER_MOUNT_LOGS': False, 'DR_CLOUD_WATCH_ENABLE': False, 'DR_CLOUD_WATCH_LOG_STREAM_PREFIX': '', 'DR_DOCKER_STYLE': 'swarm',
     'DR_HOST_X': False, 'DR_DISPLAY': None, 'DR_XAUTHORITY': None, 'DR_DIR': '/tmp/drfc', 'DRFC_CONSOLE_LOGGING': False, 'DRFC_DEBUG': False, 'USER': 'jupy
     ter-jhmal', 'DR_TARGET_HOST': 'localhost', 'DR_TARGET_PORT': 8080, 'DR_VIEWER_CONTAINERS': '', 'DR_VIEWER_QUALITY': 75, 'DR_VIEWER_WIDTH': 480, 'DR_VIE
     WER_HEIGHT': 360, 'DR_VIEWER_TOPIC': '/racecar/deepracer/kvs_stream', 'DR_WEBVIEWER_PORT': 8100, 'DR_ROBOMAKER_TRAIN_PORT': 8080, 'DR_ROBOMAKER_GUI_POR
     T': 5900, 'DR_SAGEMAKER_CUDA_DEVICES': '0', 'DR_ROBOMAKER_CUDA_DEVICES': '0', 'DR_ROBOMAKER_EVAL_PORT': 8080, 'DR_GAZEBO_ARGS': '', 'ROBOMAKER_COMMAN
     D': '', 'DR_TELEGRAF_HOST': 'telegraf', 'DR_TELEGRAF_PORT': '8092', 'DRFC_REPO_ABS_PATH': '/home/insightlab/deepracer/deepracer-for-cloud', 'DR_SIMTRAC
     E_S3_PREFIX': '', 'DR_LOCAL_S3_AUTH_MODE': 'profile', 'DR_DYNAMIC_PROXY_PORT': 8090})
```

Source: Study questionnaire

## A.2 Metadata Configuration

The model's metadata, such as the action space and sensor type, is configured using specific dataclasses, which ensures data type validation.

Figure 9 – Defining the model’s metadata in a structured way.



```
[7]: dasp1 = DiscreteActionSpace(steering_angle=-30, speed=0.6)
     dasp2 = DiscreteActionSpace(steering_angle=-15, speed=0.6)
     dasp3 = DiscreteActionSpace(steering_angle=-0, speed=0.6)
     dasp4 = DiscreteActionSpace(steering_angle=15, speed=0.6)
     dasp5 = DiscreteActionSpace(steering_angle=30, speed=0.6)
     action_spaces = [dasp1, dasp2, dasp3, dasp4, dasp5]

     # model_metadata = ModelMetadata(action_space_type=ActionSpaceType.DISCRETE, action_space=action_spaces)
     model_metadata = ModelMetadata(
         action_space=ContinuousActionSpace(steering_angle=SteeringAngle(high=30.0, low=-30.0), speed=Speed(high=2.0, low=0.75))
     )
     model_metadata
```

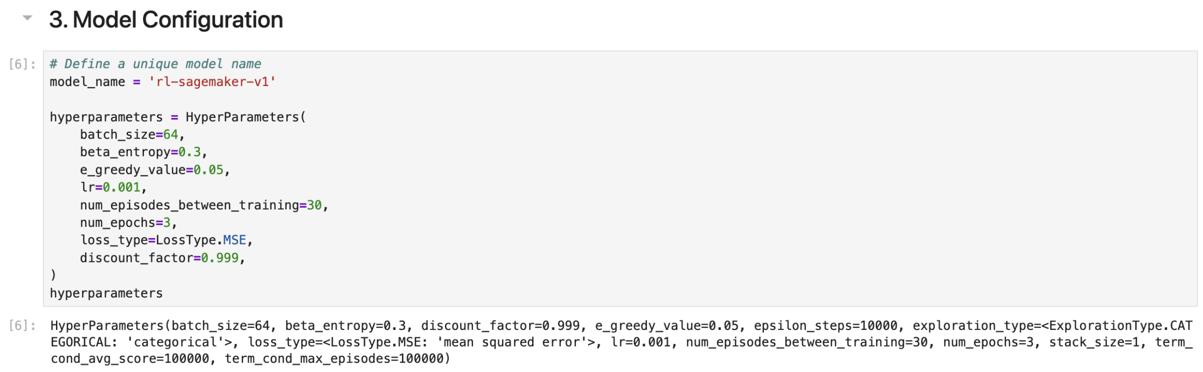
```
[7]: ModelMetadata(action_space_type=<ActionSpaceType.CONTINUOUS: 'continuous'>, action_space=ContinuousActionSpace(steering_angle=SteeringAngle(high=30.0,
     low=-30.0), speed=Speed(high=2.0, low=0.75)), version=5, training_algorithm=<TrainingAlgorithm.PPO: 'clipped_ppo'>, neural_network=<NeuralNetwork.DEEP_
     CONVOLUTIONAL_NETWORK_SHALLOW: 'DEEP_CONVOLUTIONAL_NETWORK_SHALLOW'>, sensor=[<Sensor.FRONT_FACING_CAMERA: 'FRONT_FACING_CAMERA'>])
```

Source: Study questionnaire

## A.3 Hyperparameter Definition

Similarly to metadata, the training hyperparameters are defined in a dedicated dataclass, which simplifies configuration and prevents errors.

Figure 10 – Defining the hyperparameters for model training.



3. Model Configuration

```
[6]: # Define a unique model name
     model_name = 'rl-sagemaker-v1'

     hyperparameters = HyperParameters(
         batch_size=64,
         beta_entropy=0.3,
         e_greedy_value=0.05,
         lr=0.001,
         num_episodes_between_training=30,
         num_epochs=3,
         loss_type=LossType.MSE,
         discount_factor=0.999,
     )
     hyperparameters
```

```
[6]: HyperParameters(batch_size=64, beta_entropy=0.3, discount_factor=0.999, e_greedy_value=0.05, epsilon_steps=10000, exploration_type=<ExplorationType.CAT
     EGORICAL: 'categorical'>, loss_type=<LossType.MSE: 'mean squared error'>, lr=0.001, num_episodes_between_training=30, num_epochs=3, stack_size=1, term_
     cond_avg_score=100000, term_cond_max_episodes=100000)
```

Source: Study questionnaire

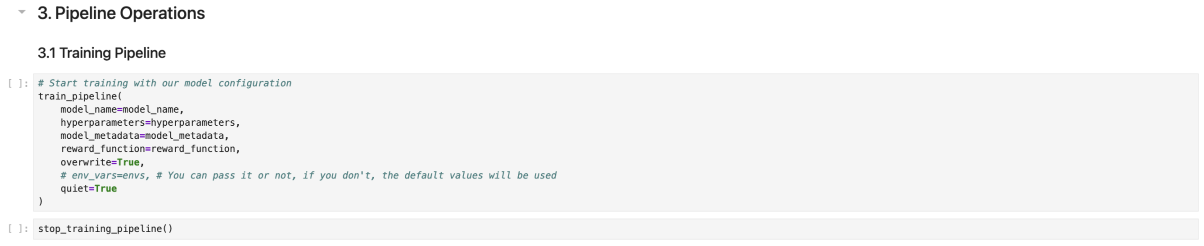
## A.4 Training Initiation and Finalization

The training pipeline is initiated with a single function call, which uses the previously defined configuration objects.

## A.5 Status Monitoring

The library provides functions to programmatically way to start the Grafana visualization.

Figure 11 – Command to start the model training process.



3. Pipeline Operations

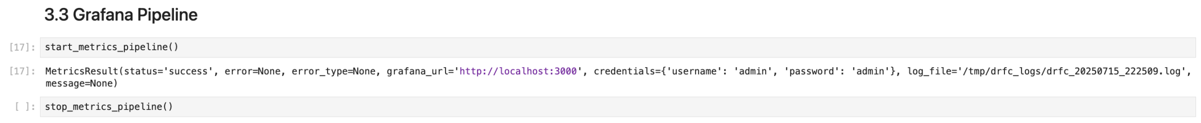
3.1 Training Pipeline

```
[ ]: # Start training with our model configuration
     train_pipeline(
         model_name=model_name,
         hyperparameters=hyperparameters,
         model_metadata=model_metadata,
         reward_function=reward_function,
         overwrite=True,
         # env_vars=envs, # You can pass it or not, if you don't, the default values will be used
         quiet=True
     )
```

```
[ ]: stop_training_pipeline()
```

Source: Study questionnaire

Figure 12 – Start the monitoring.



3.3 Grafana Pipeline

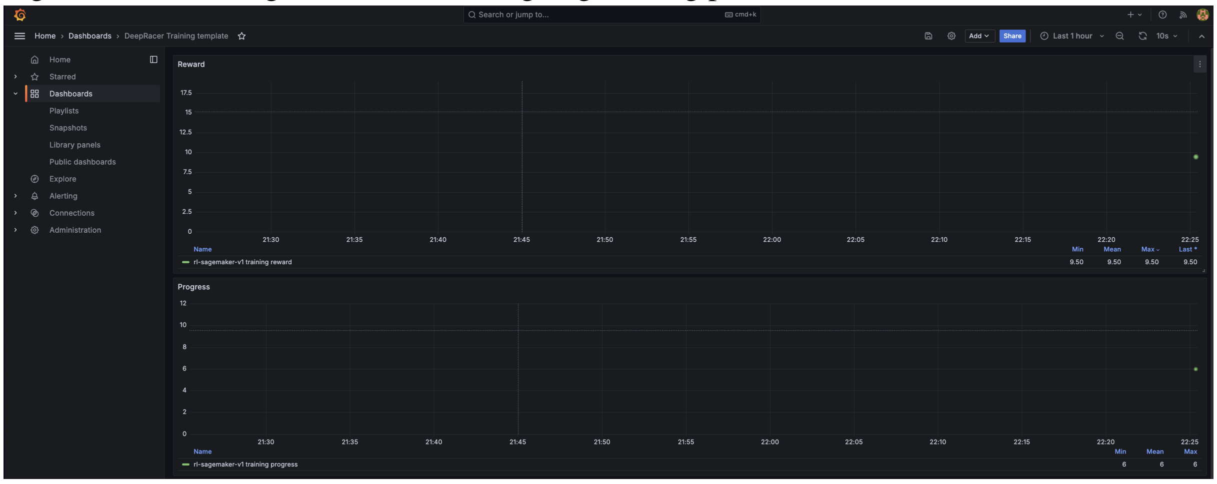
```
[17]: start_metrics_pipeline()
```

```
[17]: MetricsResult(status='success', error=None, error_type=None, grafana_url='http://localhost:3000', credentials={'username': 'admin', 'password': 'admin'}, log_file='/tmp/drfc_logs/drfc_20250715_222509.log',
      message=None)
```

```
[ ]: stop_metrics_pipeline()
```

Source: Study questionnaire

Figure 13 – Checking the status of an ongoing training process.



```
Name                                  Min    Mean   Max    Last *
rl-sagemaker-v1 training reward       9.50   9.50   9.50   9.50
```

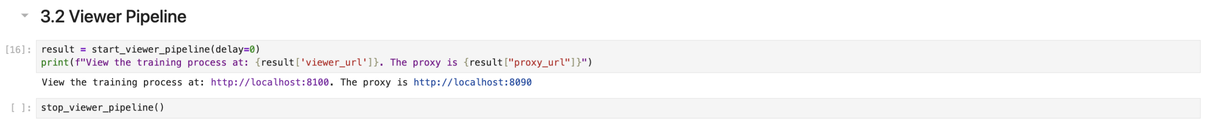
```
Name                                  Min    Mean   Max
rl-sagemaker-v1 training progress     6      6      6
```

Source: Study questionnaire

## A.6 Live Video Visualization

The simulation viewer interface, provided by Streamlit, can be initiated and managed directly from the notebook.

Figure 14 – Starting the simulation viewer to observe the agent's behavior.



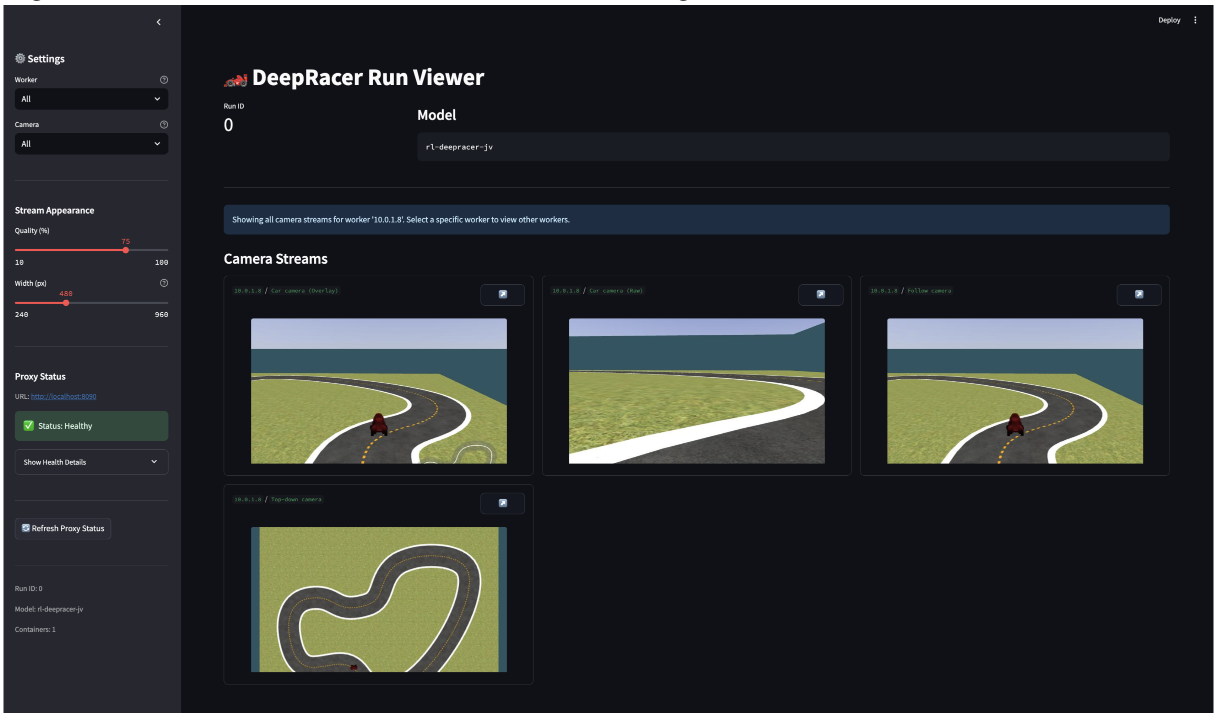
3.2 Viewer Pipeline

```
[16]: result = start_viewer_pipeline(delay=0)
      print(f"View the training process at: {result['viewer_url']}. The proxy is {result["proxy_url"]}")

      View the training process at: http://localhost:8100. The proxy is http://localhost:8090
```

```
[ ]: stop_viewer_pipeline()
```

Source: Study questionnaire

Figure 15 – Track simulation viewer to observe the agent’s behavior.



Run ID: 0

Model: rl-deepracer-jv

Containers: 1

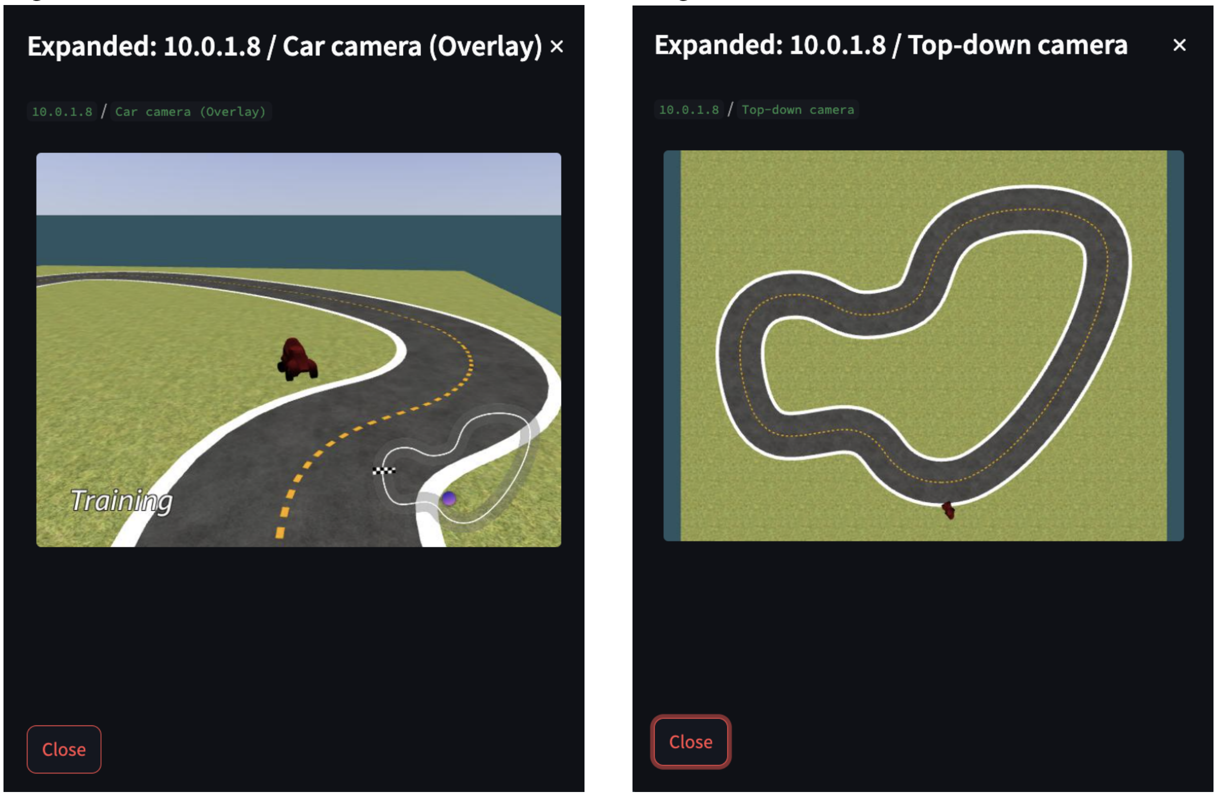
Run ID
0

Model
rl-deepracer-jv

Showing all camera streams for worker '10.0.1.8'. Select a specific worker to view other workers.

Source: Study questionnaire

Figure 16 – Track simulation viewer to observe the agent’s behavior.

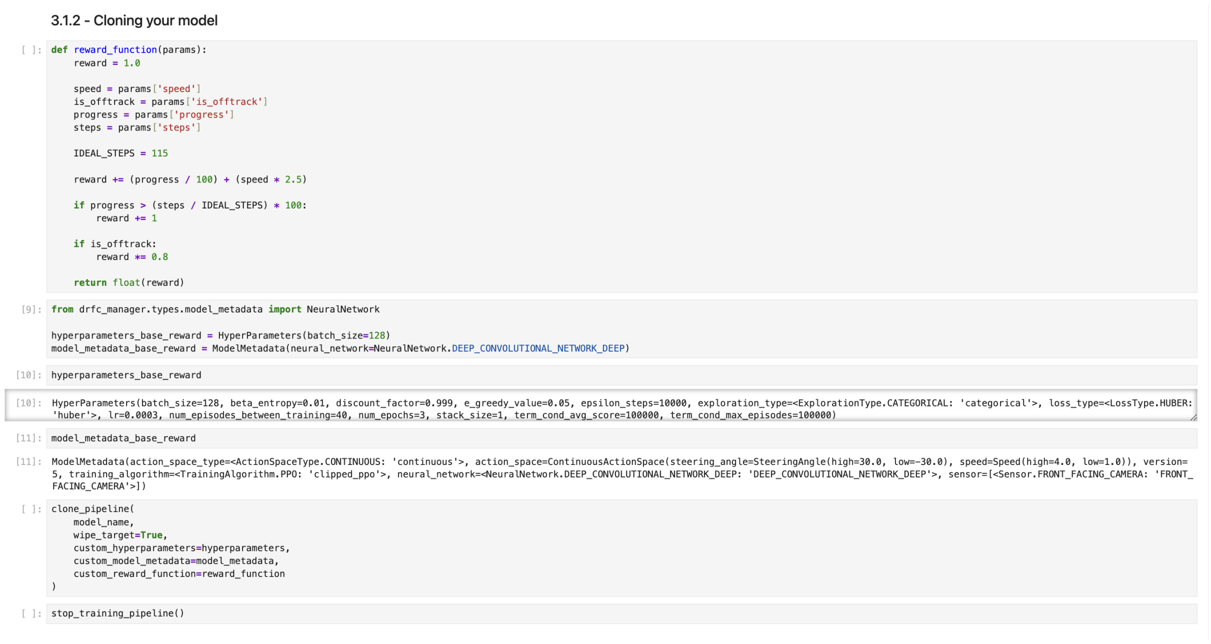

Expanded: 10.0.1.8 / Car camera (Overlay) ×
10.0.1.8 / Car camera (Overlay)
Training
Close
Expanded: 10.0.1.8 / Top-down camera ×
10.0.1.8 / Top-down camera
Close

Source: Study questionnaire

## A.7 Model Cloning

The library simplifies the process of cloning an existing model, allowing for the creation of a new version to start an incremental training session.

Figure 17 – Cloning a model to conduct a new experiment.



3.1.2 - Cloning your model

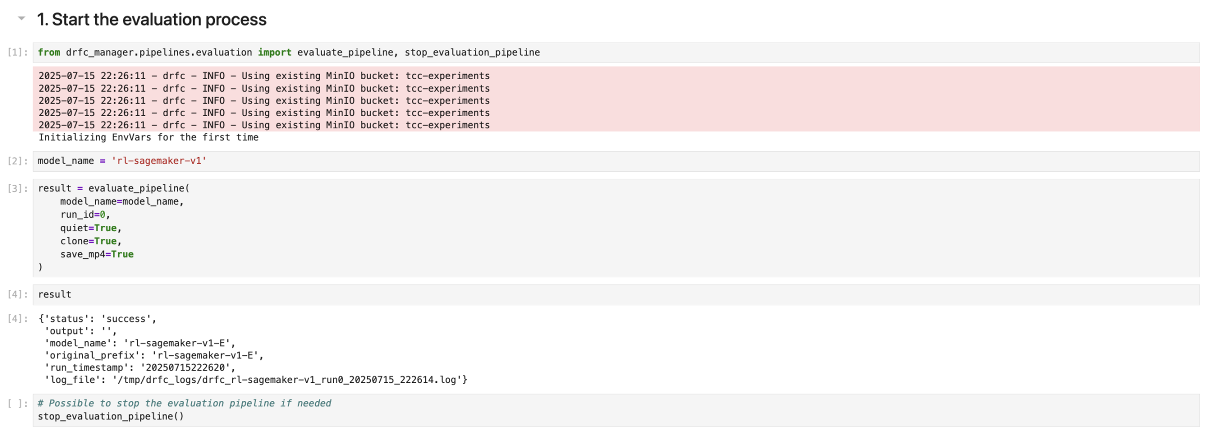
```
[ ]: def reward_function(params):
         reward = 1.0

         speed = params['speed']
         is_offtrack = params['is_offtrack']
         progress = params['progress']
         steps = params['steps']

         IDEAL_STEPS = 115

         reward += (progress / 100) + (speed * 2.5)

         if progress > (steps / IDEAL_STEPS) * 100:
             reward += 1

         if is_offtrack:
             reward *= 0.8

         return float(reward)

[9]: from drfc_manager.types.model_metadata import NeuralNetwork

     hyperparameters_base_reward = HyperParameters(batch_size=128)
     model_metadata_base_reward = ModelMetadata(neural_network=NeuralNetwork.DEEP_CONVOLUTIONAL_NETWORK_DEEP)

[10]: hyperparameters_base_reward

[10]: HyperParameters(batch_size=128, beta_entropy=0.01, discount_factor=0.999, e_greedy_value=0.05, epsilon_steps=10000, exploration_type=<ExplorationType.CATEGORICAL: 'categorical'>, loss_type=<LossType.HUBER:
      'huber'>, lr=0.0003, num_episodes_between_training=40, num_epochs=3, stack_size=1, term_cond_avg_score=100000, term_cond_max_episodes=100000)

[11]: model_metadata_base_reward

[11]: ModelMetadata(action_space_type=<ActionSpaceType.CONTINUOUS: 'continuous'>, action_space=ContinuousActionSpace(steering_angle=SteeringAngle(high=30.0, low=-30.0), speed=Speed(high=4.0, low=1.0)), version=
      5, training_algorithm=<TrainingAlgorithm.PPO: 'clipped_ppo'>, neural_network=<NeuralNetwork.DEEP_CONVOLUTIONAL_NETWORK_DEEP: 'DEEP_CONVOLUTIONAL_NETWORK_DEEP'>, sensor=[<Sensor.FRONT_FACING_CAMERA: 'FRONT_
      FACING_CAMERA'>])

[ ]: clone_pipeline(
         model_name,
         wipe_target=True,
         custom_hyperparameters=hyperparameters,
         custom_model_metadata=model_metadata,
         custom_reward_function=reward_function
     )

[ ]: stop_training_pipeline()
```

Source: Study questionnaire

## A.8 Model Evaluation

The evaluation pipeline is initiated similarly to the training one, selecting a specific model checkpoint to be tested on the track.

Figure 18 – Initiating an evaluation pipeline for a model.

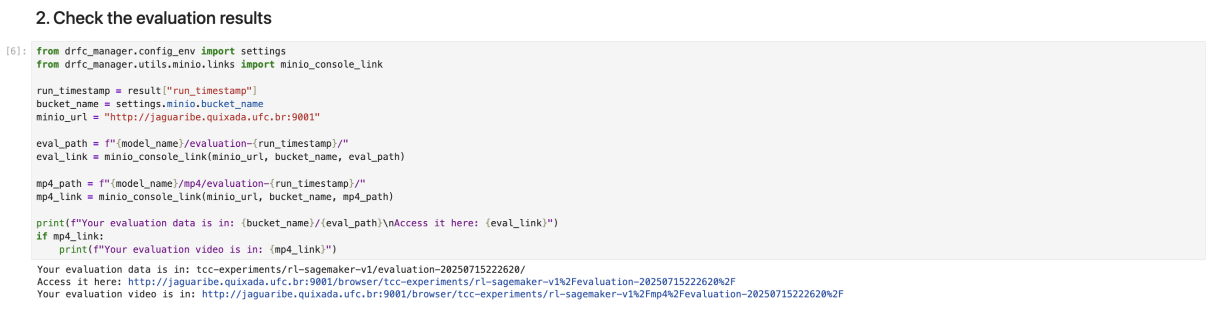


1. Start the evaluation process

```
[1]: from drfc_manager.pipelines.evaluation import evaluate_pipeline, stop_evaluation_pipeline

     2025-07-15 22:26:11 - drfc - INFO - Using existing MinIO bucket: tcc-experiments
     2025-07-15 22:26:11 - drfc - INFO - Using existing MinIO bucket: tcc-experiments
     2025-07-15 22:26:11 - drfc - INFO - Using existing MinIO bucket: tcc-experiments
     2025-07-15 22:26:11 - drfc - INFO - Using existing MinIO bucket: tcc-experiments
     2025-07-15 22:26:11 - drfc - INFO - Using existing MinIO bucket: tcc-experiments
     Initializing EnvVars for the first time

[2]: model_name = 'rl-sagemaker-v1'

[3]: result = evaluate_pipeline(
         model_name=model_name,
         run_id=0,
         quiet=True,
         clone=True,
         save_mp4=True
     )

[4]: result

[4]: {'status': 'success',
      'output': '',
      'model_name': 'rl-sagemaker-v1-E',
      'original_prefix': 'rl-sagemaker-v1-E',
      'run_timestamp': '20250715222620',
      'log_file': '/tmp/drfc_logs/drfc_rl-sagemaker-v1_run0_20250715_222614.log'}

[ ]: # Possible to stop the evaluation pipeline if needed
     stop_evaluation_pipeline()
```

Source: Study questionnaire

Figure 19 – Check the evaluation pipeline result given a model.



2. Check the evaluation results

```
[6]: from drfc_manager.config_env import settings
     from drfc_manager.utils.minio.links import minio_console_link

     run_timestamp = result["run_timestamp"]
     bucket_name = settings.minio.bucket_name
     minio_url = "http://jaguaribe.quixada.ufc.br:9001"

     eval_path = f"{model_name}/evaluation-{run_timestamp}/"
     eval_link = minio_console_link(minio_url, bucket_name, eval_path)

     mp4_path = f"{model_name}/mp4/evaluation-{run_timestamp}/"
     mp4_link = minio_console_link(minio_url, bucket_name, mp4_path)

     print(f"Your evaluation data is in: {bucket_name}/{eval_path}\nAccess it here: {eval_link}")
     if mp4_link:
         print(f"Your evaluation video is in: {mp4_link}")

     Your evaluation data is in: tcc-experiments/rl-sagemaker-v1/evaluation-20250715222620/
     Access it here: http://jaguaribe.quixada.ufc.br:9001/browser/tcc-experiments/rl-sagemaker-v1%2Fevaluation-20250715222620%2F
     Your evaluation video is in: http://jaguaribe.quixada.ufc.br:9001/browser/tcc-experiments/rl-sagemaker-v1%2Fmp4%2Fevaluation-20250715222620%2F
```

Source: Study questionnaire

# B CONSOLIDATED QUESTIONNAIRE FEEDBACK

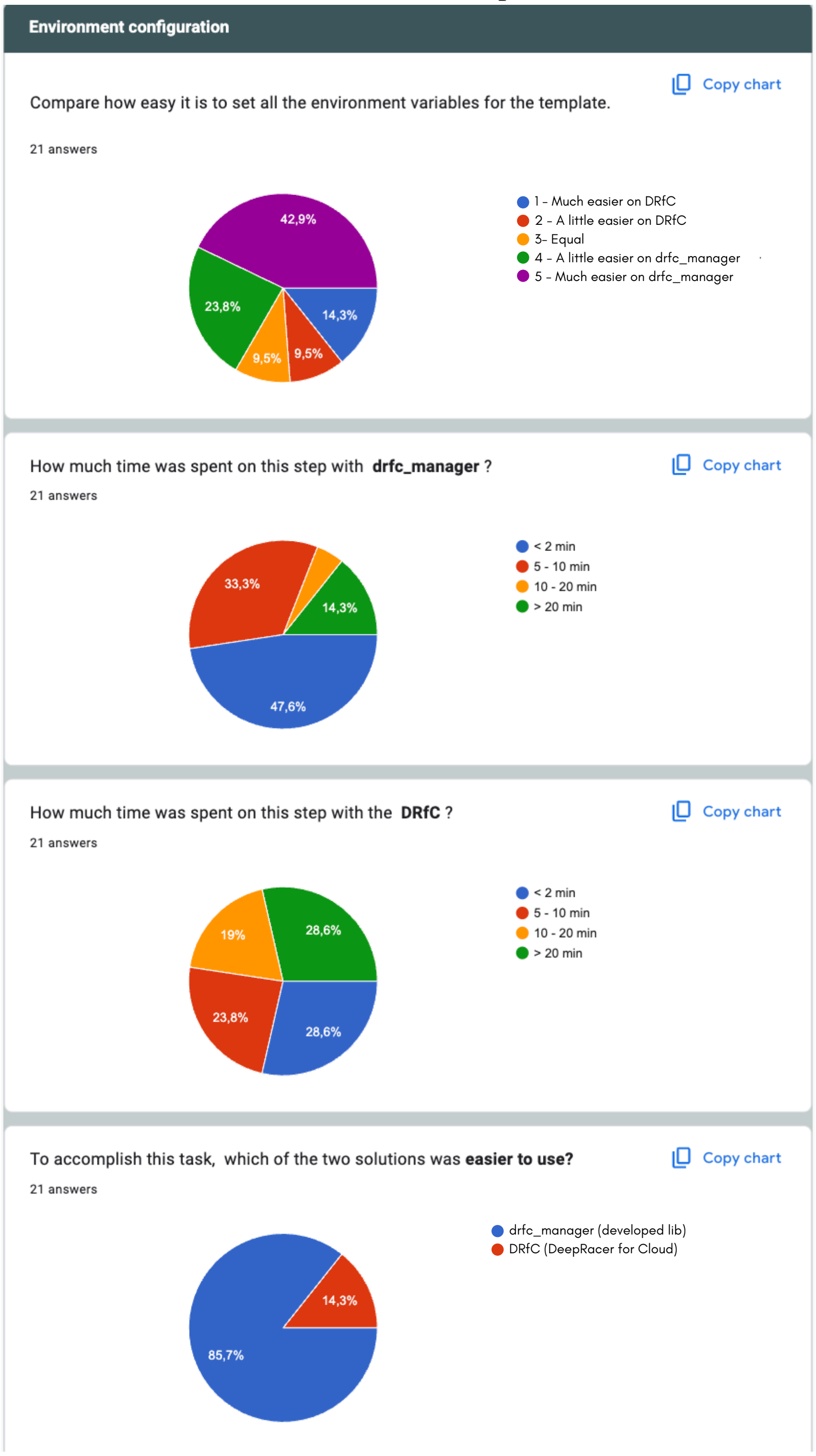
This appendix presents the complete, disaggregated data collected from the comparative usability study questionnaire. Each section corresponds to one of the 10 tasks performed by the participants. For each task, the charts display the user ratings on comparative ease-of-use, the self-reported time spent using each solution (`drfc_manager` vs. baseline DRfC), and the direct preference between the two tools for that specific task.

This raw data forms the basis for the quantitative and qualitative analysis presented in the Results and Analysis chapter.

## B.1 Task 1: Environment Setup

The following charts summarize user feedback regarding the ease and time required to configure the environment variables for the experiment using both solutions.

Figure 20 – Feedback charts for the Environment Setup task.



Environment configuration

Compare how easy it is to set all the environment variables for the template.

21 answers

How much time was spent on this step with **drfc_manager** ?

21 answers

How much time was spent on this step with the **DRfC** ?

21 answers

To accomplish this task, which of the two solutions was **easier to use?**

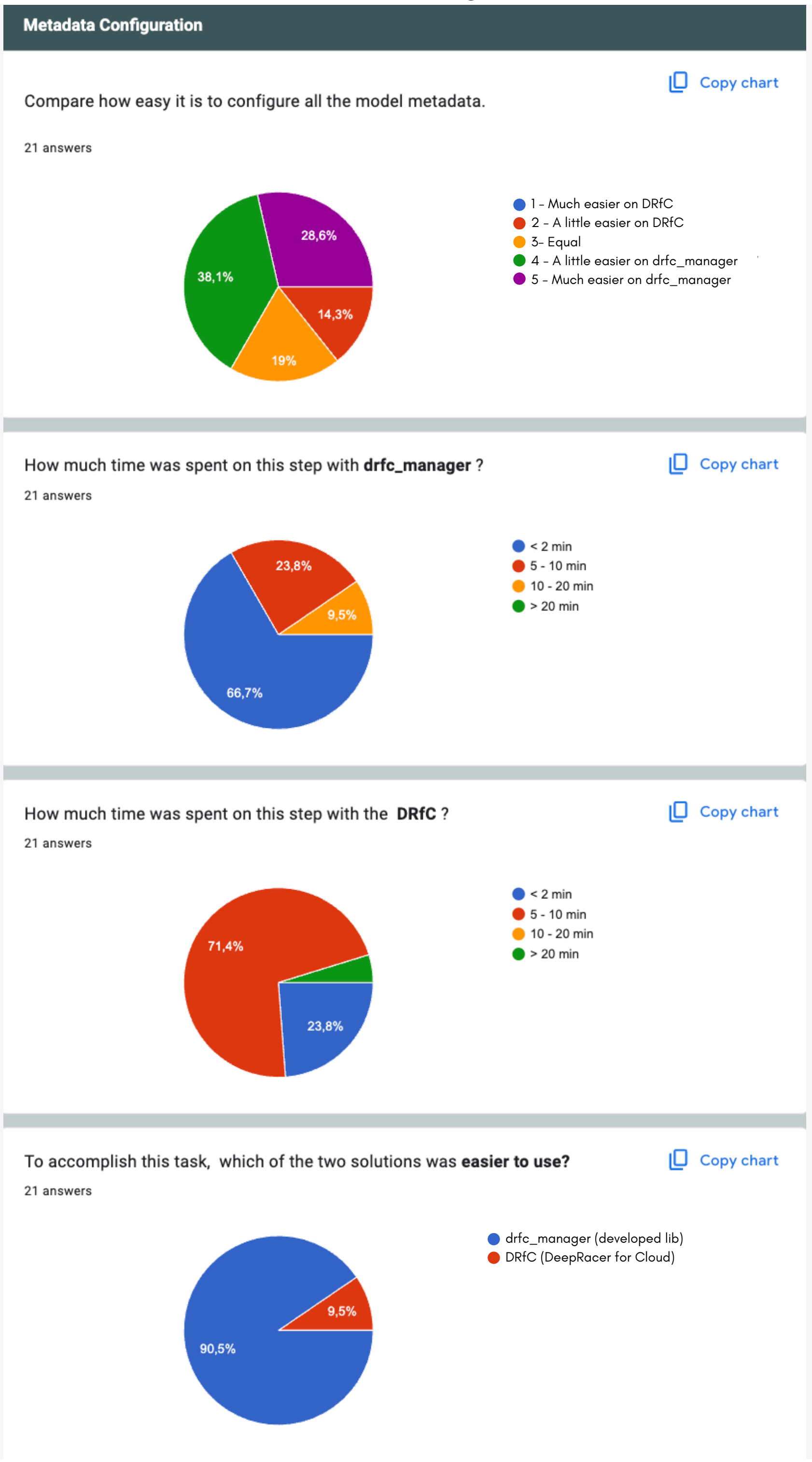
21 answers

Source: Study questionnaire

## B.2 Task 2: Metadata Configuration

The following charts summarize user feedback regarding the ease and time required to configure the model's metadata (e.g., action space, sensors).

Figure 21 – Feedback charts for the Metadata Configuration task.



**Metadata Configuration**

Compare how easy it is to configure all the model metadata.

21 answers

- ● 1 - Much easier on DRfC
- ● 2 - A little easier on DRfC
- ● 3- Equal
- ● 4 - A little easier on drfc_manager
- ● 5 - Much easier on drfc_manager

How much time was spent on this step with **drfc_manager** ?

21 answers

- ● < 2 min
- ● 5 - 10 min
- ● 10 - 20 min
- ● > 20 min

How much time was spent on this step with the **DRfC** ?

21 answers

- ● < 2 min
- ● 5 - 10 min
- ● 10 - 20 min
- ● > 20 min

To accomplish this task, which of the two solutions was **easier to use?**

21 answers

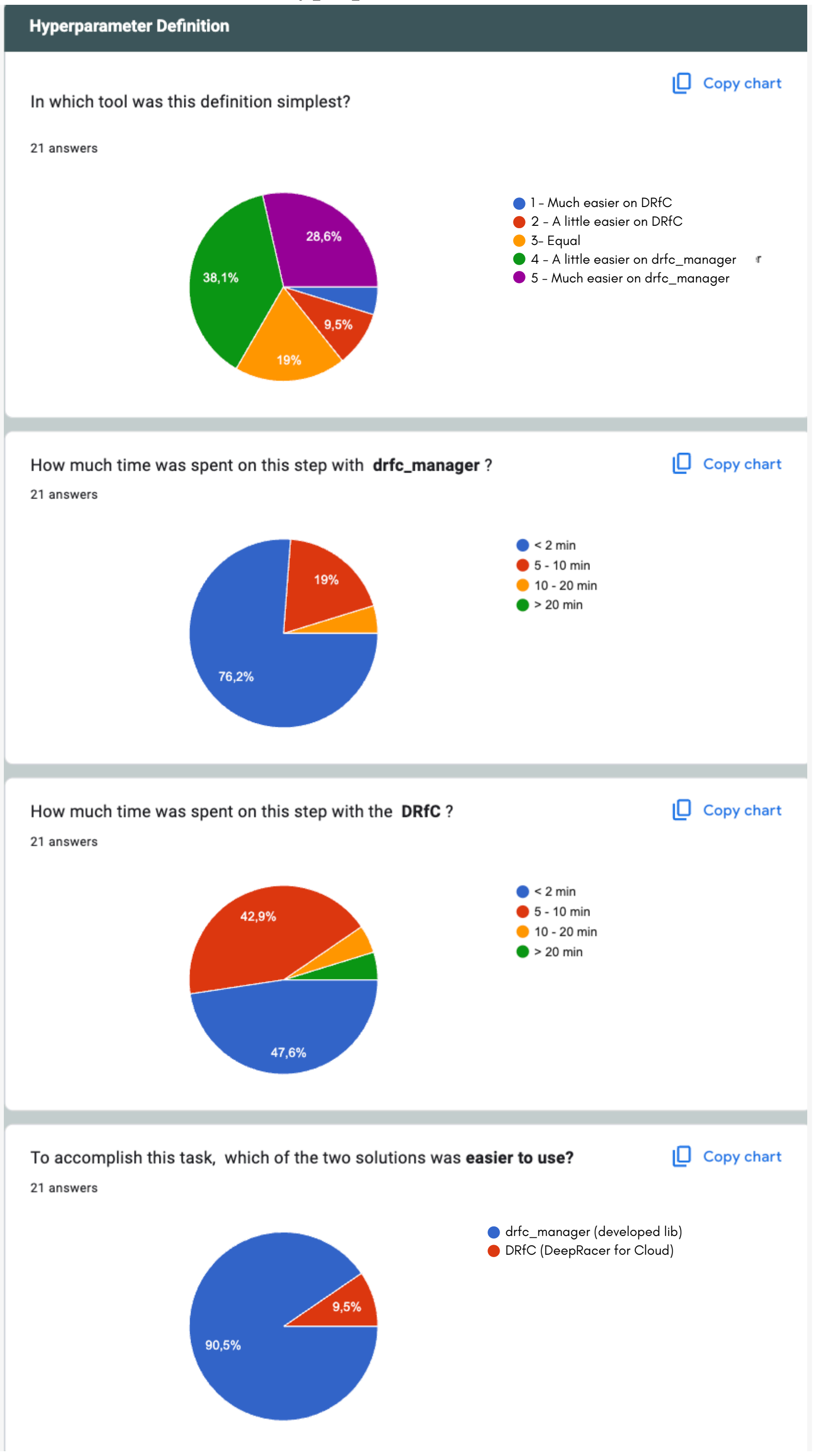
- ● drfc_manager (developed lib)
- ● DRfC (DeepRacer for Cloud)

Source: Study questionnaire

## B.3 Task 3: Hyperparameter Definition

The following charts summarize user feedback regarding the ease and time required to define the training hyperparameters.

Figure 22 – Feedback charts for the Hyperparameter Definition task.



Hyperparameter Definition

In which tool was this definition simplest?

21 answers

How much time was spent on this step with **drfc_manager** ?

21 answers

How much time was spent on this step with the **DRfC** ?

21 answers

To accomplish this task, which of the two solutions was **easier to use?**

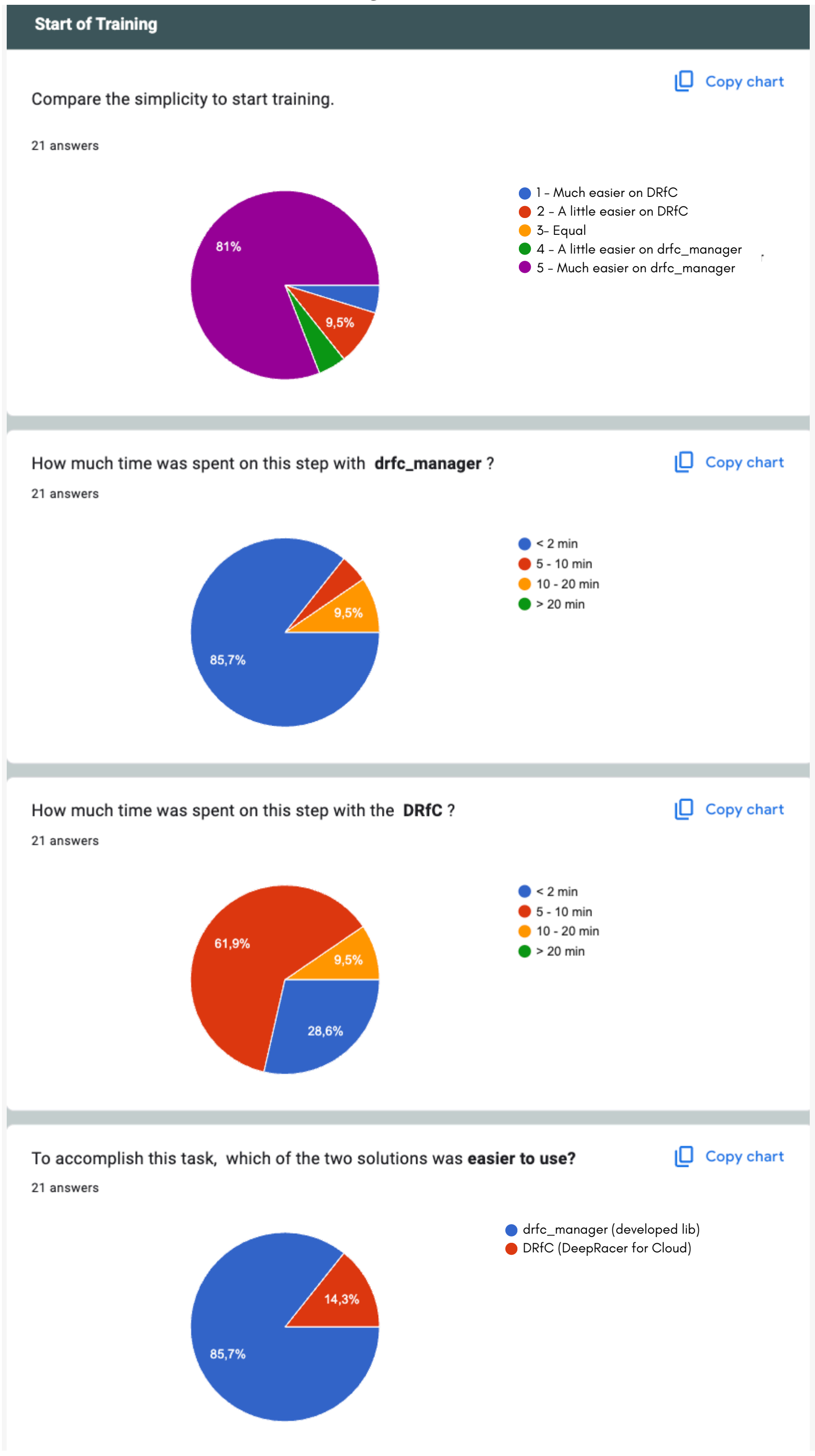
21 answers

Source: Study questionnaire

## B.4 Task 4: Training Initiation

The following charts summarize user feedback regarding the ease and time required to start the training process.

Figure 23 – Feedback charts for the Training Initiation task.

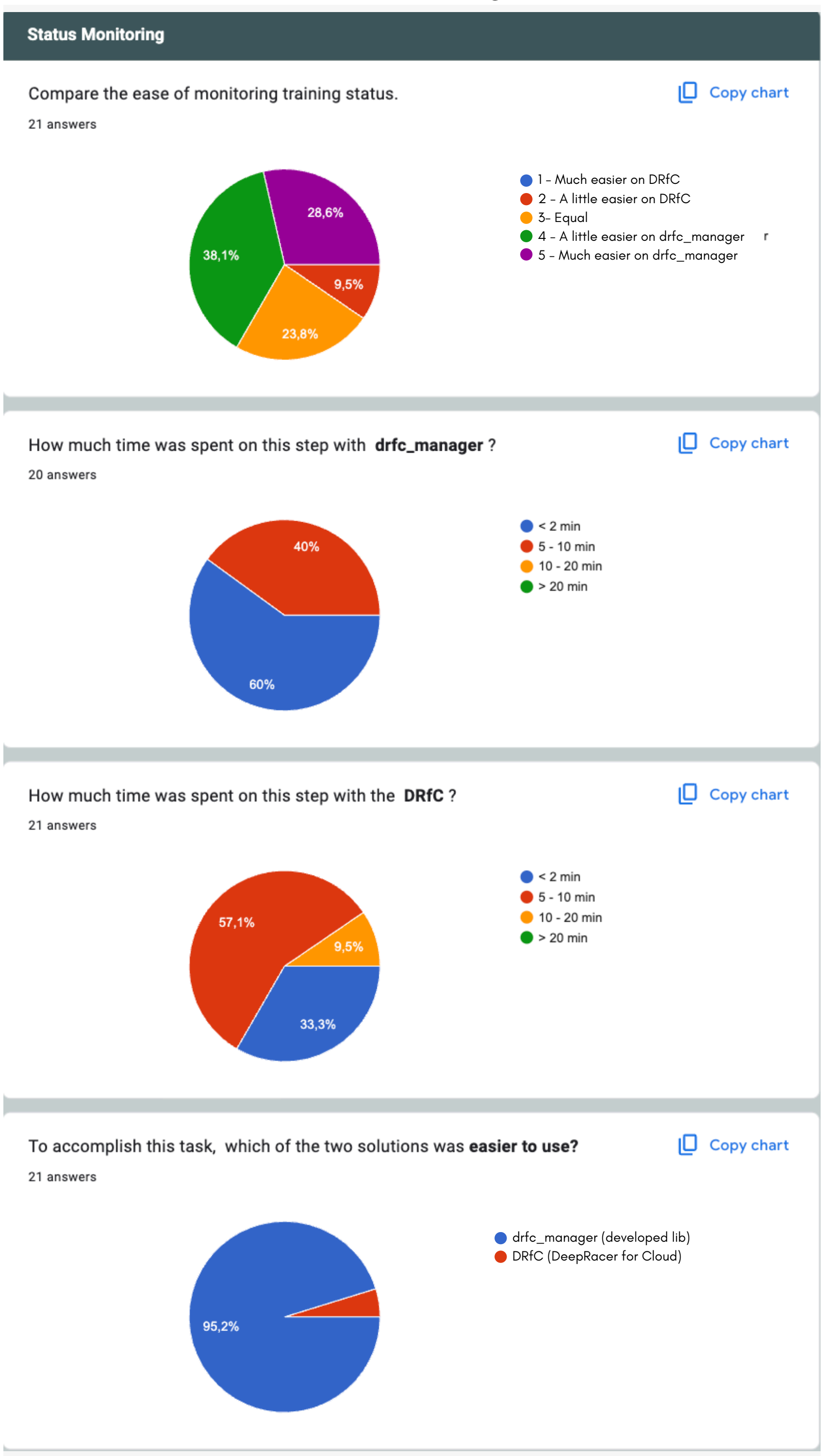


Source: Study questionnaire

## B.5 Task 5: Status Monitoring

The following charts summarize user feedback regarding the ease and time required to monitor the status of an ongoing training process.

Figure 24 – Feedback charts for the Status Monitoring task.



Compare the ease of monitoring training status.
21 answers

- ● 1 - Much easier on DRfC
- ● 2 - A little easier on DRfC
- ● 3- Equal
- ● 4 - A little easier on drfc_manager
- ● 5 - Much easier on drfc_manager

How much time was spent on this step with **drfc_manager** ?
20 answers

- ● < 2 min
- ● 5 - 10 min
- ● 10 - 20 min
- ● > 20 min

How much time was spent on this step with the **DRfC** ?
21 answers

- ● < 2 min
- ● 5 - 10 min
- ● 10 - 20 min
- ● > 20 min

To accomplish this task, which of the two solutions was **easier to use?**
21 answers

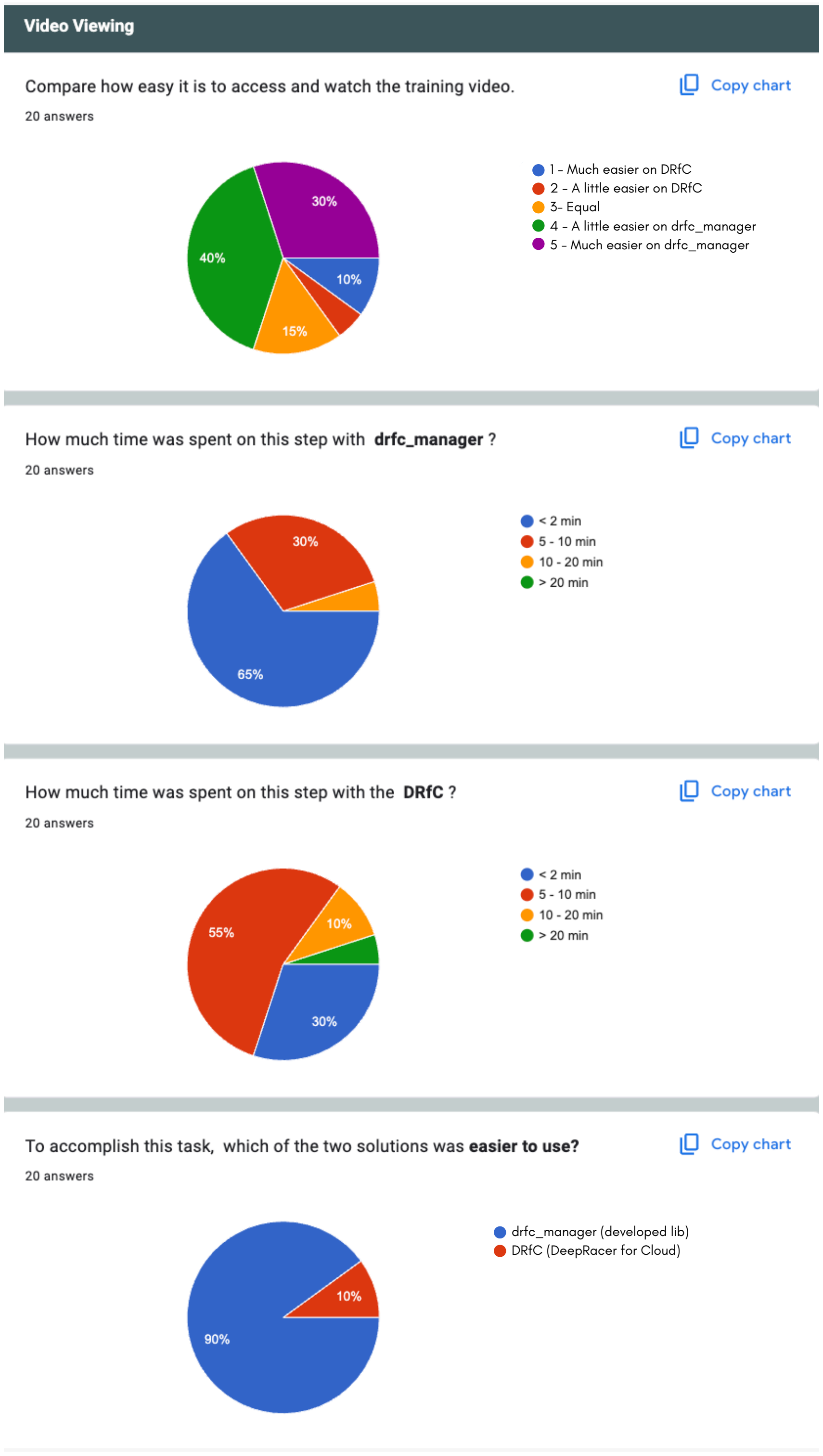
- ● drfc_manager (developed lib)
- ● DRfC (DeepRacer for Cloud)

Source: Study questionnaire

## B.6 Task 6: Live Video Visualization

The following charts summarize user feedback regarding the ease and time required to access and view the live simulation video.

Figure 25 – Feedback charts for the Live Video Visualization task.



Video Viewing

Compare how easy it is to access and watch the training video.

20 answers

How much time was spent on this step with **drfc_manager** ?

20 answers

How much time was spent on this step with the **DRfC** ?

20 answers

To accomplish this task, which of the two solutions was **easier to use?**

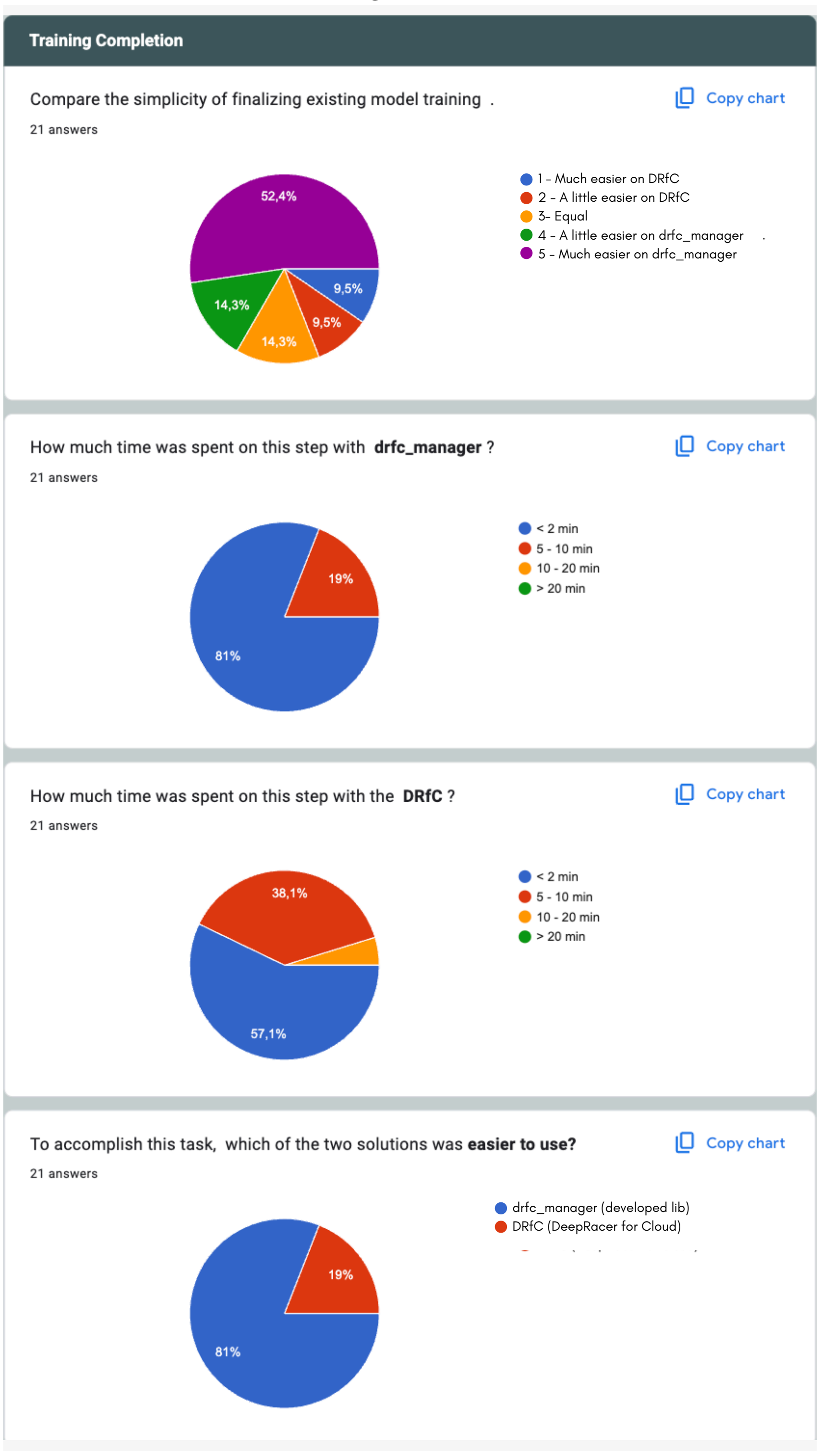
20 answers

Source: Study questionnaire

## B.7 Task 7: Training Finalization

The following charts summarize user feedback regarding the ease and time required to stop the training process correctly.

Figure 26 – Feedback charts for the Training Finalization task.



**Training Completion**

Compare the simplicity of finalizing existing model training .

21 answers

How much time was spent on this step with **drfc_manager** ?

21 answers

How much time was spent on this step with the **DRfC** ?

21 answers

To accomplish this task, which of the two solutions was **easier to use?**

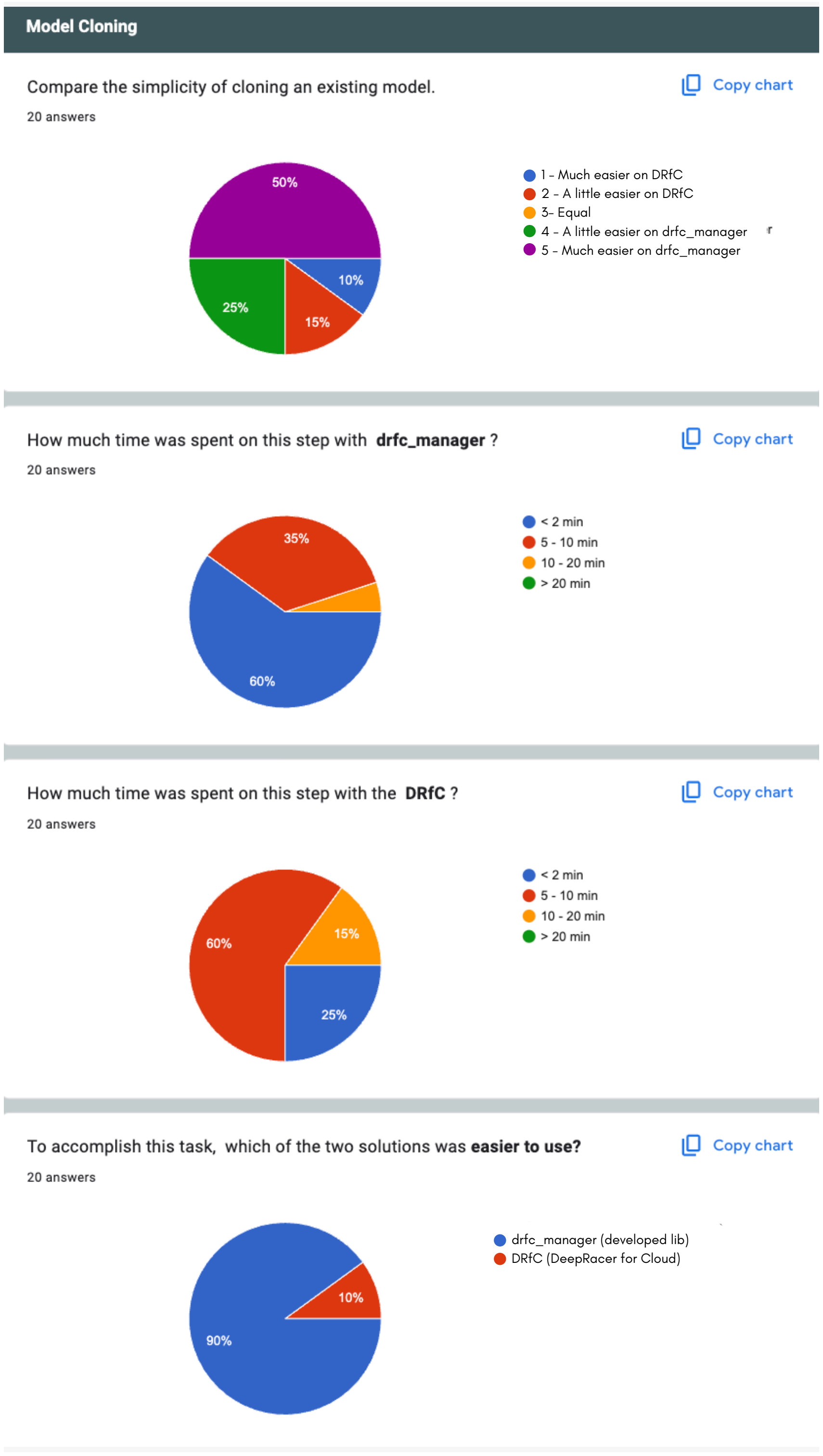
21 answers

Source: Study questionnaire

## B.8 Task 8: Model Cloning

The following charts summarize user feedback regarding the ease and time required to clone an existing model for a new experiment.

Figure 27 – Feedback charts for the Model Cloning task.

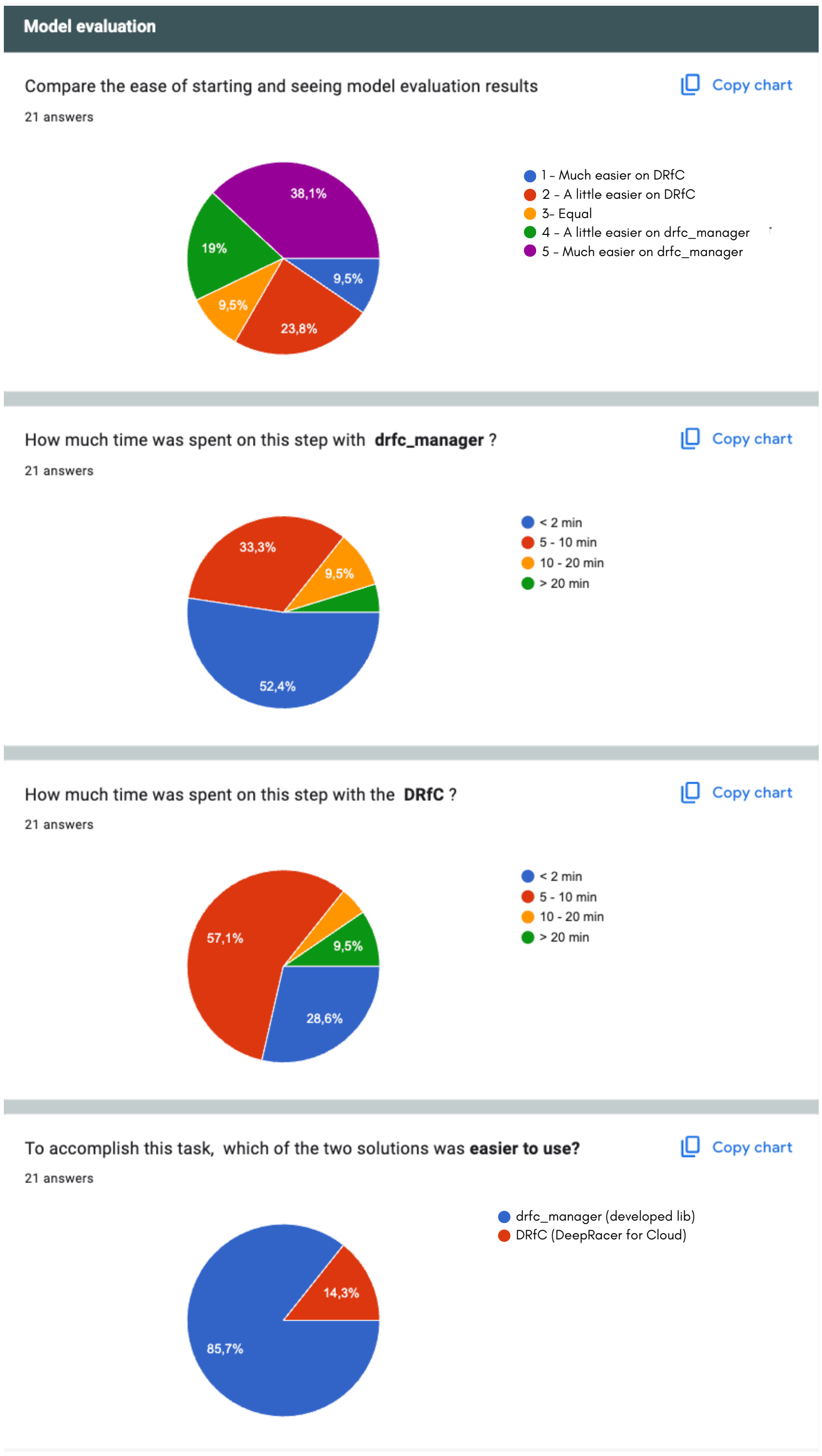


Source: Study questionnaire

## B.9 Task 9: Model Evaluation

The following charts summarize user feedback regarding the ease and time required to start and review the results of a model evaluation run.

Figure 28 – Feedback charts for the Model Evaluation task.

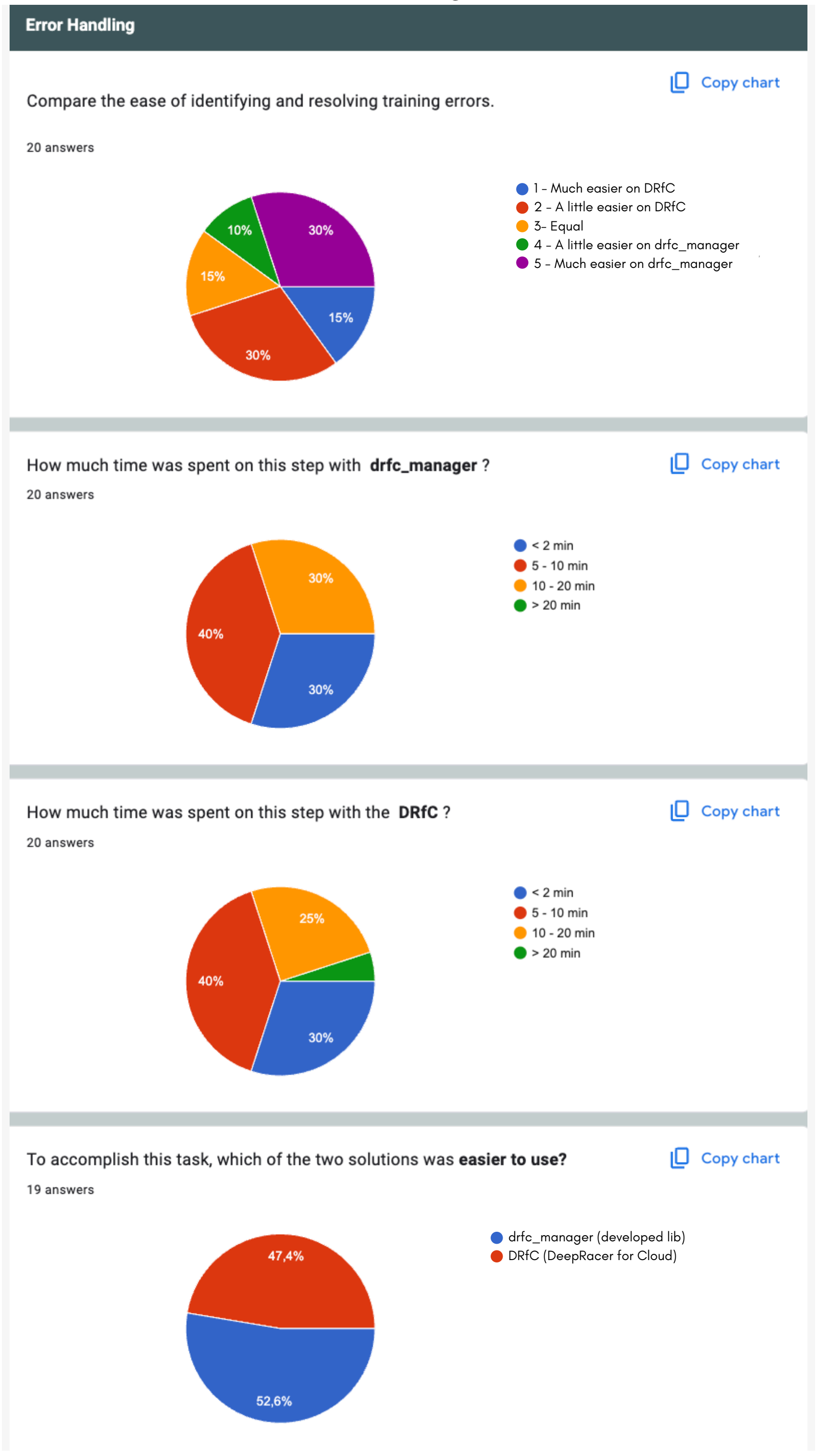


Source: Study questionnaire

## B.10 Task 10: Error Handling

The following charts summarize user feedback regarding the ease and time required to identify and resolve errors during the workflow.

Figure 29 – Feedback charts for the Error Handling task.



Error Handling

Compare the ease of identifying and resolving training errors.

20 answers

How much time was spent on this step with **drfc_manager** ?

20 answers

How much time was spent on this step with the **DRfC** ?

20 answers

To accomplish this task, which of the two solutions was **easier to use?**

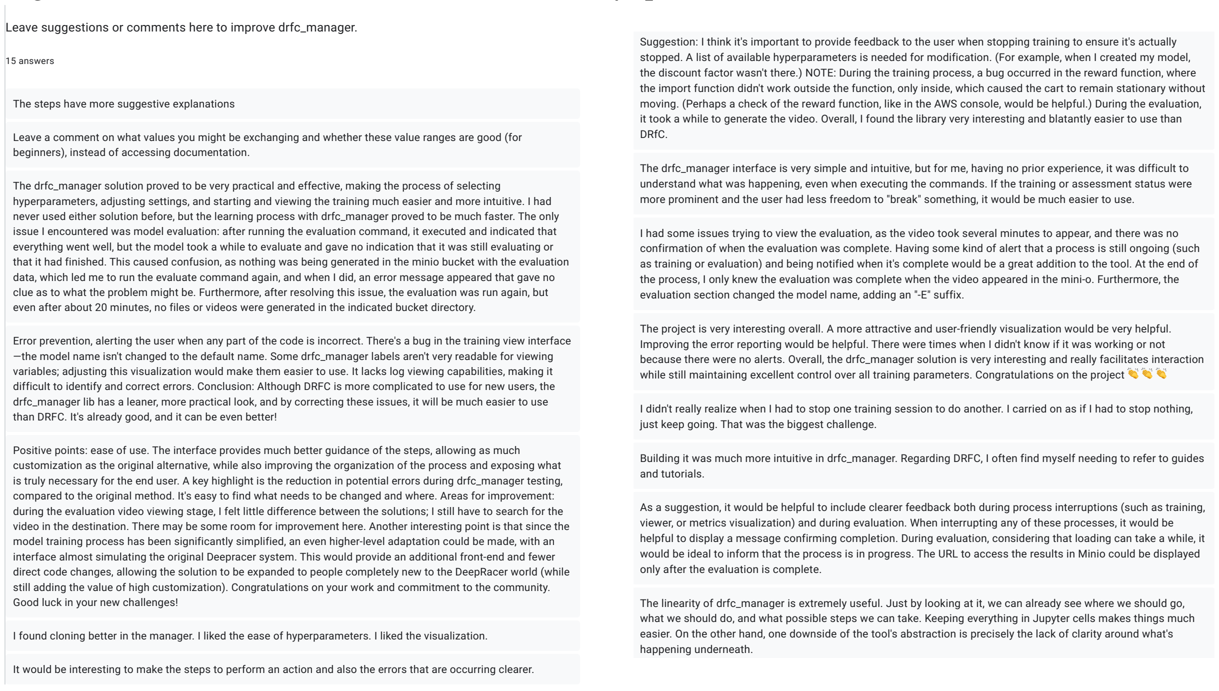
19 answers

Source: Study questionnaire

## B.11 Qualitative Feedback

This section presents a sample of the open-ended qualitative feedback provided by participants, highlighting common themes and suggestions for improvement.

Figure 30 – Qualitative feedbacks from the study questionnaire.



Leave suggestions or comments here to improve drfc_manager.

15 answers

The steps have more suggestive explanations

Leave a comment on what values you might be exchanging and whether these value ranges are good (for beginners), instead of accessing documentation.

The drfc_manager solution proved to be very practical and effective, making the process of selecting hyperparameters, adjusting settings, and starting and viewing the training much easier and more intuitive. I had never used either solution before, but the learning process with drfc_manager proved to be much faster. The only issue I encountered was model evaluation: after running the evaluation command, it executed and indicated that everything went well, but the model took a while to evaluate and gave no indication that it was still evaluating or that it had finished. This caused confusion, as nothing was being generated in the minio bucket with the evaluation data, which led me to run the evaluate command again, and when I did, an error message appeared that gave no clue as to what the problem might be. Furthermore, after resolving this issue, the evaluation was run again, but even after about 20 minutes, no files or videos were generated in the indicated bucket directory.

Error prevention, alerting the user when any part of the code is incorrect. There's a bug in the training view interface—the model name isn't changed to the default name. Some drfc_manager labels aren't very readable for viewing variables; adjusting this visualization would make them easier to use. It lacks log viewing capabilities, making it difficult to identify and correct errors. Conclusion: Although DRFC is more complicated to use for new users, the drfc_manager lib has a leaner, more practical look, and by correcting these issues, it will be much easier to use than DRFC. It's already good, and it can be even better!

Positive points: ease of use. The interface provides much better guidance of the steps, allowing as much customization as the original alternative, while also improving the organization of the process and exposing what is truly necessary for the end user. A key highlight is the reduction in potential errors during drfc_manager testing, compared to the original method. It's easy to find what needs to be changed and where. Areas for improvement: during the evaluation video viewing stage, I felt little difference between the solutions; I still have to search for the video in the destination. There may be some room for improvement here. Another interesting point is that since the model training process has been significantly simplified, an even higher-level adaptation could be made, with an interface almost simulating the original Deepracer system. This would provide an additional front-end and fewer direct code changes, allowing the solution to be expanded to people completely new to the DeepRacer world (while still adding the value of high customization). Congratulations on your work and commitment to the community. Good luck in your new challenges!

I found cloning better in the manager. I liked the ease of hyperparameters. I liked the visualization.

It would be interesting to make the steps to perform an action and also the errors that are occurring clearer.

Suggestion: I think it's important to provide feedback to the user when stopping training to ensure it's actually stopped. A list of available hyperparameters is needed for modification. (For example, when I created my model, the discount factor wasn't there.) NOTE: During the training process, a bug occurred in the reward function, where the import function didn't work outside the function, only inside, which caused the cart to remain stationary without moving. (Perhaps a check of the reward function, like in the AWS console, would be helpful.) During the evaluation, it took a while to generate the video. Overall, I found the library very interesting and blatantly easier to use than DRfC.

The drfc_manager interface is very simple and intuitive, but for me, having no prior experience, it was difficult to understand what was happening, even when executing the commands. If the training or assessment status were more prominent and the user had less freedom to "break" something, it would be much easier to use.

I had some issues trying to view the evaluation, as the video took several minutes to appear, and there was no confirmation of when the evaluation was complete. Having some kind of alert that a process is still ongoing (such as training or evaluation) and being notified when it's complete would be a great addition to the tool. At the end of the process, I only knew the evaluation was complete when the video appeared in the mini-o. Furthermore, the evaluation section changed the model name, adding an "-E" suffix.

The project is very interesting overall. A more attractive and user-friendly visualization would be very helpful. Improving the error reporting would be helpful. There were times when I didn't know if it was working or not because there were no alerts. Overall, the drfc_manager solution is very interesting and really facilitates interaction while still maintaining excellent control over all training parameters. Congratulations on the project 👏👏👏

I didn't really realize when I had to stop one training session to do another. I carried on as if I had to stop nothing, just keep going. That was the biggest challenge.

Building it was much more intuitive in drfc_manager. Regarding DRFC, I often find myself needing to refer to guides and tutorials.

As a suggestion, it would be helpful to include clearer feedback both during process interruptions (such as training, viewer, or metrics visualization) and during evaluation. When interrupting any of these processes, it would be helpful to display a message confirming completion. During evaluation, considering that loading can take a while, it would be ideal to inform that the process is in progress. The URL to access the results in Minio could be displayed only after the evaluation is complete.

The linearity of drfc_manager is extremely useful. Just by looking at it, we can already see where we should go, what we should do, and what possible steps we can take. Keeping everything in Jupyter cells makes things much easier. On the other hand, one downside of the tool's abstraction is precisely the lack of clarity around what's happening underneath.

Source: Study questionnaire