
Learn To Race - Distributed Reinforcement Learning and Optimization Thrust

Shijie Bian

Mentors: Arav Agarwal Jonathan Francis

Language Technologies Institute

Carnegie Mellon University

Pittsburgh, PA 15213

{sbian,arava,jmf1}@cs.cmu.edu

1

Abstract

2

This report delves into the research thrust of distributed reinforcement learning (RL) and
3 optimization within the Learn to Race (L2R) project. Our focus centers on propelling
4 autonomous racing agents by leveraging distributed training to expedite the learning process.
5 L2R integrates a multimodal environment, utilizing diverse sensors and computer vision
6 encoders to empower autonomous agents in real-time vehicle control and environment
7 perception. A pivotal innovation in our approach is distributed training, accelerating the
8 training of our autonomous system by harnessing computing resources for parallel processing.
9 This not only expedites the learning process but also facilitates the collection of diverse
10 environments, enhancing the robustness of training. We present two primary distributed
11 training paradigms: the *centralized distributed experience collection paradigm*, inspired
12 by IMPALA [1]; and the *decentralized distributed parameter update paradigm*, inspired
13 by the federated learning approach. This work offers a comprehensive exploration of the
14 detailed methodology, experimental results, and performance analysis for both distributed
15 paradigms tested across various RL environments. Our findings mark a significant stride
16 toward developing autonomous systems with heightened efficiency and precision, adept at
17 navigating complex environments. The codebase of our implementation for the distributed
18 RL research thrust is publicly accessible via GitHub¹.

19

1 Introduction

20

The advancement of autonomous vehicles navigating intricate road conditions owes much to the
21 advent of powerful computer vision models which enable efficient and effective pattern recognition
22 and visual understanding. These vehicles leverage an array of sensors to collect real-time data,
23 analyzing and learning from it using machine learning algorithms [2]. Despite these accomplishments,
24 traditional reinforcement learning and large-scale computer vision architectures pose challenges in
25 real-time applications like racing cars. Their reliance on extensive data and extended training and
26 inference times can compromise reliability.

27

In response to these challenges, Learn-to-Race (L2R) [3] emerges as a multimodal environment within
28 the Arrival Autonomous Racing Simulator. Engineered to deliver robust and efficient autonomous
29 racing solutions, L2R empowers agents through sensory inputs, computer vision encoders, and
30 distributed reinforcement learning agents. However, prior approaches and iterations of the project
31 predominantly focused on the sequential training of RL agents, which greatly limited the scalability
32 when multiple GPUs or computing resources were available. In particular, a traditional sequential
33 reinforcement training paradigm that performs agent update, environment simulation, and experience
34 collection and sampling, is limited. We aim to utilize distributed training to support the paralleled
35 experience collection from diverse environment settings, and fully utilized multiple GPUs to perform

¹<https://github.com/BrandonBian/l2r-distributed>

parameter updated, as illustrated in Figure 1. Acknowledging the pivotal role of powerful GPUs in accelerating training processes, we believe the design of distributed systems and training paradigms that harness their capabilities for parallel processing can be particularly useful for the scope of the L2R project.

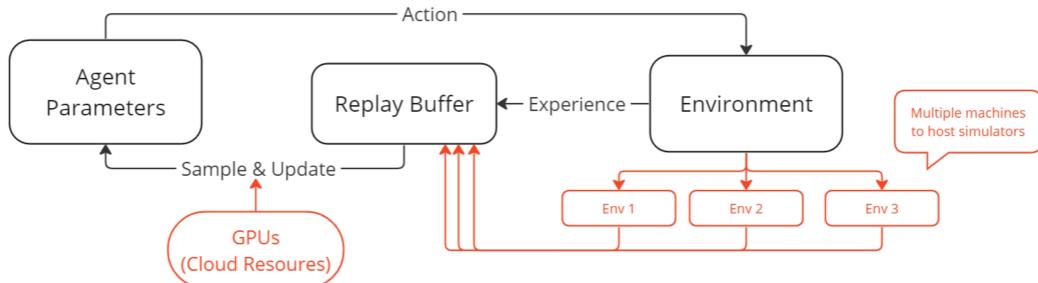


Figure 1: Motivation of using distributed training to support diverse environment simulation and utilize GPU resources

40 2 Hypothesis/Project Goal

- 41 1. **Centralized Learning Bottleneck:** We hypothesize that the observed degradation in the
42 performance of the current distributed learning setup for RL agents from the previous
43 iteration stems from the rapid generation of simulation data by the worker. This rapid influx
44 of data poses a challenge for the centralized learner, turning it into a bottleneck that struggles
45 to effectively handle the data.

46 2. **Distributed Training Scalability:** We hypothesize that distributing training tasks across
47 multiple actors has the potential to alleviate the burden on the learner, consequently reducing
48 training time. This distributed approach aims to enhance the scalability of the RL training
49 process, allowing for the management of larger and more complex environments. Additionally,
50 the utilization of distributed training is expected to tap into the additional computational
51 resources available in a cluster or cloud computing environment, resulting in accelerated
52 training times and improved performance compared to a non-distributed architecture.

53 3. **Importance Sampling and Replay Ratio Tuning:** We hypothesize that implementing
54 importance sampling and replay ratio tuning methods will enable the identification of
55 useful feedback while discarding irrelevant information. This targeted approach aims
56 to enhance the quality of the experience data collected in a distributed manner, thereby
57 contributing to improved training outcomes.increasing the interpretability and effectiveness
58 of the distributed training process.

59 4. **Modularization of Paradigms and Environments:** We hypothesize that the modularization
60 of paradigms and environments within a unified framework is anticipated to provide several
61 benefits. It offers flexibility for experimenting with different combinations of paradigms
62 and RL environments. This modularity also facilitates efficient debugging, allowing the
63 testing of various paradigm and environment combinations. Moreover, the modular approach
64 benefits future iterations of development by enabling the seamless integration and removal
65 of different environments and paradigms for iterative testing and improvement.

66 3 Relationship to Prior Work

- ⁶⁷ The standard IMPALA [1] architecture has been utilized in the previous work, where one centralized
⁶⁸ learner updates the RL agent’s parameters, and multiple workers generate reward and replay buffers
⁶⁹ from performing simulations. More specifically, the centralized learner broadcasts its parameters to

70 the workers so that the workers can periodically synchronize and reproduce the learner's agent state;
 71 the workers perform simulation on their local copy of environment, and send the collected experience
 72 back to the learner for centralized agent update, as illustrated in Figure 2. However, previous
 73 iterations of distributed RL faced a significant issue where the reward did not converge to a maximum
 74 effectively, even with an increasing number of workers. This degradation of performance is a primary
 75 problem as the distributed methodology should provide faster training and convergence speed due to
 76 more resources running in parallel and should at least perform similarly to the sequential version.
 77 In the current iteration of the project, we aim to address this issue by profiling the performance and
 78 workload distribution among the workers and learner, as well as the update speed, and hypothesize
 79 that the degradation of performance may be due to the overhead on the learner when the worker
 80 generates buffers too fast, hyperparameter tuning, and network latency. To resolve this, we will
 81 focus on two aspects: (1) moving the training to the workers so that the learner does not become a
 82 bottleneck and (2) implementing importance sampling with (1) to filter only the most useful replay
 83 buffers, reduce contention and confusion, and mitigate the issues that arise with an excessive number
 84 of buffers generated by the workers.

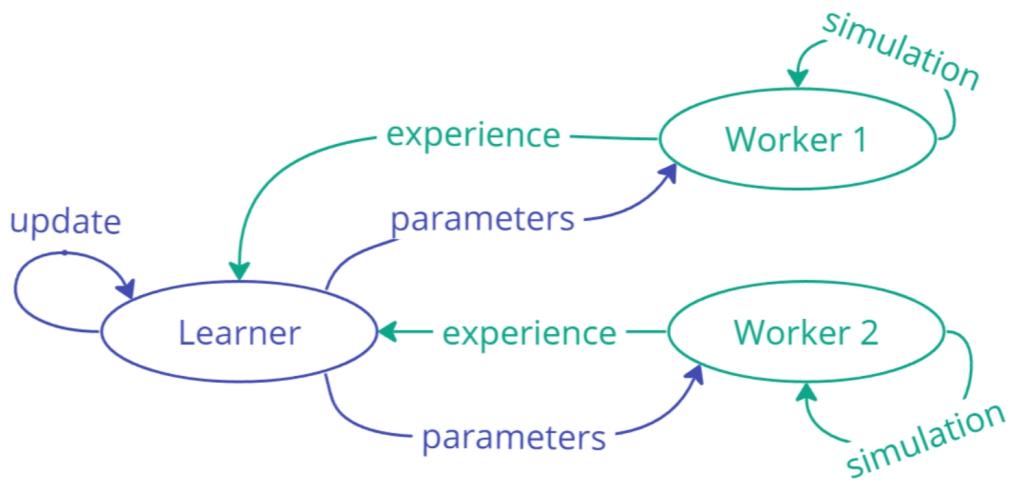


Figure 2: The distributed architecture from prior iteration, where the learner is responsible for agent update, and the workers are responsible for environment simulation and experience collection

85 In addressing the first aspect, we aim to delve into a distributed training paradigm similar to the
 86 federated learning process. A recent research paper on distributed RL paradigms [4] provided many
 87 useful insights. Specifically, prior works derived from the IMPALA architecture, such as GORILA
 88 [5], have sought to overcome challenges similar to those outlined earlier. GORILA achieves this
 89 by decoupling parameter synchronization and experience collection into two distinct functional
 90 modules that feature a parameter server and a dedicated learner. Similarly, the Asynchronous
 91 Advantage Actor-Critic (A3C) framework [6] introduces parameter sharing between workers and a
 92 global network, facilitating decentralized gradient update tasks. Building upon insights from these
 93 established methodologies, we propose a novel training paradigm. In our design, the learner serves a
 94 dual role as a server for worker task allocation and control while functioning as a learner responsible
 95 for parameter update and synchronization, thus referred to as the learner server. Distinctly, workers
 96 adopt versatile roles as experience collectors, parameter update trainers, or evaluators based on tasks
 97 assigned by the learner server. This approach effectively redistributes the training burden from the
 98 learner to the workers, eliminating the need for an additional stand-alone server node solely for
 99 parameter-sharing control. Furthermore, this design empowers workers to multi-function based on
 100 assigned tasks, thereby maximizing the utilization of decentralized computing resources distributed
 101 across the worker nodes.

102 **4 Fall Semester Development Goals**

103 The objectives for the fall semester encompass four primary facets, as illustrated in Figure 3. Firstly,
104 we leverage the compatibility of our system with OpenAI's Gymnasium environments to incorporate
105 a medium-difficulty typical 2D environment, named Lunar Lander for additional testing of the
106 distributed paradigm. Secondly, we reproduce the SAC actor-critic algorithm using OpenAI Spinning-
107 up vanilla implementation along with hyperparameter configurations shared by HuggingFace to
108 fine-tune RL agents. After the replication process, a thorough analysis of the obtained results will
109 be undertaken to discern insights and trends that can inform future developments and optimizations.
110 Thirdly, we design and develop a new distributed training paradigm, where the training tasks are
111 allocated to workers to alleviate the workload on the learner. Finally, we aim to systematically
112 restructure the existing codebase, implement automation in resource management processes, and
113 consolidate branches and stand-alone module implementations. This holistic reorganization is poised
114 to untangle the complex interplay between deep reinforcement learning and distributed system design,
115 providing a streamlined and more accessible framework for researchers. The intention is to simplify
116 the replication of baselines and the execution of experiments.

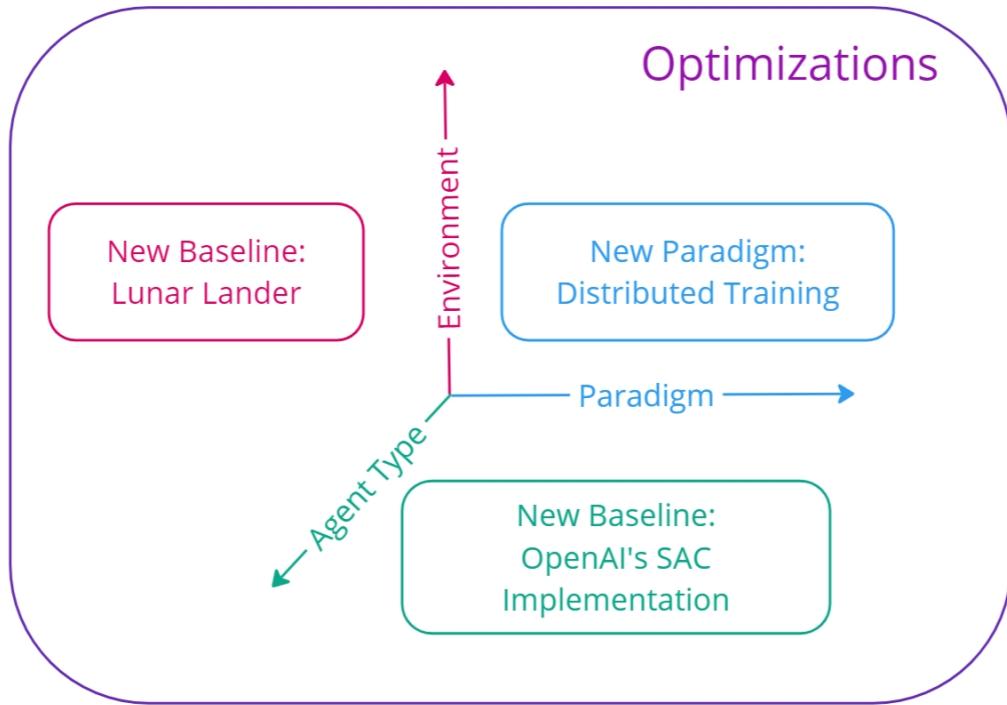


Figure 3: The distributed reinforcement learning and optimization thrust has 4 main dimensions: (1) compatibility with diverse RL environments, (2) implementation of different agent algorithms, (3) new distributed training system design paradigm, (4) optimization of codebase and resource control

117 **5 Project Requirements**

118 **5.1 Intended Users**

- 119 1. **Researchers in academia:** Our system serves as a robust platform for researchers and
120 academics in the autonomous vehicle domain. It offers an environment for the evaluation
121 and refinement of algorithms and sensor systems, thereby enriching the understanding of
122 autonomous vehicle technology. Furthermore, with our modularized code base that allows
123 for the easy plug-in testing of diverse RL environments and training paradigms that are fully

124 compatible with the L2R and OpenAI Gymnasium environments, researchers can easily
 125 explore innovative approaches to enhance the performance of their autonomous driving
 126 models.

127 **2. Businesses:** For businesses focused on self-driving car development, our system provides
 128 a valuable resource for optimizing vehicle performance. It offers insights derived from
 129 extensive experiments and ablations, eliminating the need for costly real-world testing.
 130 Through thorough analysis, businesses can pinpoint areas for improvement and fine-tune
 131 their systems, resulting in more efficient and reliable transportation solutions.

132 **5.2 Functional Requirements**

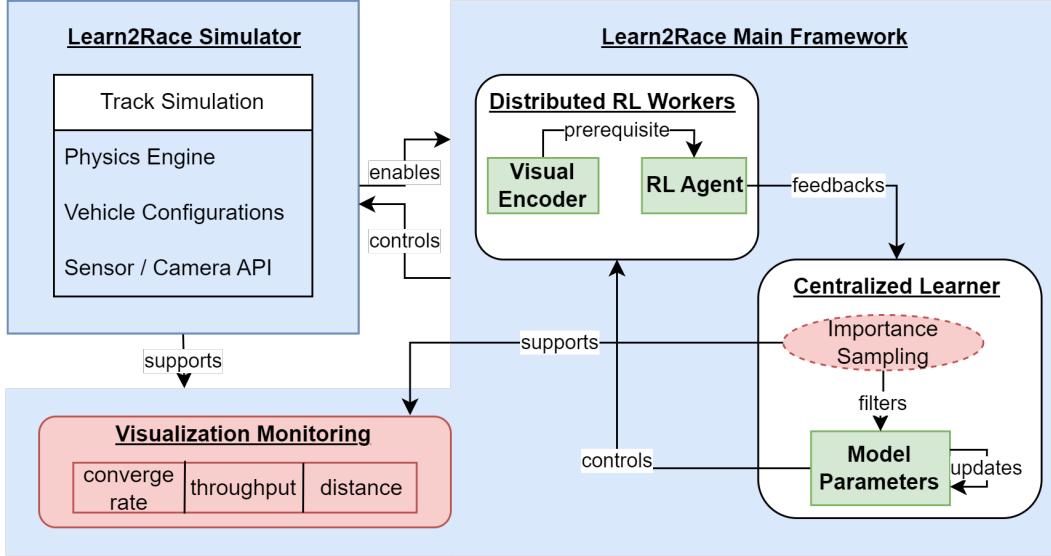


Figure 4: Context diagram for the distributed RL research thrust of Learn-to-Race

133 Figure 4 offers a comprehensive overview of the context diagram central to the proposed distributed
 134 RL and optimization research thrust. Specifically tailored for the Learn-to-Race project, this thrust
 135 plays a foundational role, serving as the backbone of the entire system architecture. The simulator
 136 captures visual inputs from the simulated environment. These inputs undergo encoding through
 137 visual encoders and are subsequently relayed to the RL agent. The RL agent, in turn, is responsible
 138 for selecting the most suitable action based on the interpreted status of the current environment.
 139 The distributed paradigm takes charge of automating and orchestrating communication between
 140 the learner server and the workers. Within decentralized workers, all crucial steps—simulation,
 141 experience collection and evaluation, as well as RL agent steps and updates—are carried out. The
 142 learner, on the other hand, handles the synchronization and updating of parameters collected from
 143 all distributed workers. The resulting action learned by the RL agent is then transmitted back to
 144 the environments within the workers. This action triggers the subsequent iteration of the simulator,
 145 generating visual input for the next timestamp.

- 146 1. The users running the distributed RL training architecture should be able to observe sub-
 147 stantial performance improvement with the addition of distributed workers and computing
 148 resources. The current distributed framework observed a degradation of performance in
 149 terms of learning convergence. This is why we should distribute the training workload to
 150 workers to improve convergence speed.
- 151 2. We aim to provide users with a modularized code space for plug-in testing of diverse
 152 training paradigms and RL environments. Furthermore, the workspace should be com-
 153 patible with OpenAI's Gymnasium library of common RL environment collections. The

154 current distributed framework lacks modularization and unification of code, where different
155 combinations of training paradigms and RL environments are segregated into distinct
156 branches, controlled by standalone Kubernetes configuration files. This fragmented structure
157 poses challenges, particularly in implementing systematic changes uniformly across all RL
158 environment and training paradigm designs.

159 **5.3 Non-Functional Requirements**

160 One of the primary nonfunctional performance goals is to mitigate any hindrance in terms of both
161 speed and convergence during the training of autonomous racing agents. Users should seamlessly
162 transition to the distributed framework without significant differences or latency issues. In terms of
163 usability, users should be able to effortlessly set up the system and run existing implementations,
164 reproducing baselines within a short timeframe. The unified implementation across paradigms and
165 environments, coupled with a modularized coding style, ensures compatibility with the software and
166 hardware requirements of the existing framework. Our goal is to guarantee a smooth experience for
167 intended users, particularly under the distributed RL thrust, where the system intertwines with deep
168 reinforcement learning algorithms.

169 **5.4 Resource Requirements**

170 To conduct our experiments and train models, we leverage the data generated by the Arrival Simulator,
171 which is accessible on the Learn-to-Race challenge website[3]. This simulator produces datasets
172 from various tracks featuring diverse backgrounds and environments, including *Track01: Thruxton*,
173 *Track02: Anglesey*, and *Track03: Vegas*. Given the implementation of distributed training paradigms,
174 ample GPU resources are essential. Our primary resource for this project is the Carnegie Mellon
175 University (CMU) Phoebe cluster, equipped with batches of GPUs such as the 2080 Ti, enabling
176 parallel testing of diverse training paradigms. Additionally, tools like Weights-and-biases, Docker,
177 and Kubernetes are utilized for experiment logging, environment encapsulation, and resource control,
178 respectively. Emphasizing automation of configurations and resource management steps, we aim
179 to minimize hardcoding, ensuring scalability and ease of use, as outlined in the non-functional
180 requirements section.

181 **6 Experiment/System Design Overview**

182 Illustrated in Figure 5 is a high-level overview of the system architecture for the distributed RL
183 thrust (more detailed implementation configurations and diagrams are included in the following
184 sections). The following three components (corresponding to the three blue blocks in the figure)
185 are the main building blocks for the distributed architecture, and all training paradigms within the
186 following methodology sections are built as the variations of these modules:

187 **6.1 Learn-to-race Simulator**

188 As highlighted earlier, our experimentation relies on the Arrival autonomous simulator, equipped
189 with sensors like cameras and auxiliary devices such as an inertial measurement unit. This simulator
190 creates a controlled racing environment, configured with a variety of maps and tracks, ensuring a
191 wealth of training samples for the RL agent. The simulator captures raw visual data representing the
192 current state of the racing agent through interfaces like cameras, sensors, and actions. These data are
193 then transmitted to the distributed RL workers, where they are used to generate the representation
194 (embedding) of the current environmental state.

195 **6.2 Distributed RL Workers**

196 Upon receiving the raw simulator output, the RL workers employ visual encoders to produce
197 embeddings, serving as latent representations of the current environment state. These representations
198 are then forwarded to the RL agent (in our baseline scenario, the SAC agent), which selects the

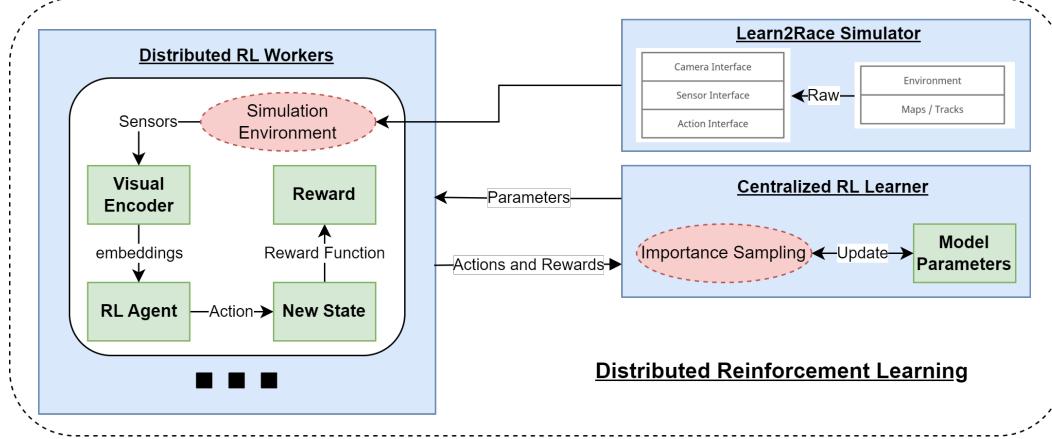


Figure 5: High-level Overview of System Architecture for the Distributed RL Thrust

199 most optimal action based on its interpretation of the current state representation. Following the
 200 execution of an action, such as adjusting the current direction or angle, the new state is leveraged to
 201 compute a reward, providing an assessment of the racing car's performance. Subsequently, the action
 202 taken and the corresponding reward are encapsulated as an experience instance and transmitted to the
 203 centralized RL learner module.

204 6.3 Centralized RL Learner

205 Ultimately, the centralized RL learner initiates parameter updates after collecting actions and rewards
 206 from the distributed RL workers. It disseminates the latest policies to these workers, encompassing
 207 the updated agent parameters and the pertinent training or inference settings. It's essential to highlight
 208 that in the latter version of the training paradigm, the centralized RL learner assumes the role of task
 209 allocation and assignment to the workers, while the workers shoulder the added responsibility of
 210 implementing model parameter updates.

211 7 Experimental Design

212 7.1 Dataset

213 We will be using the Arrival Autonomous Racing Simulator to generate the simulation environment
 214 for our reinforcement learning agent. The simulator provides a realistic racing environment that
 215 simulates the physics and mechanics of a real-world car race. In our distributed learning scenario,
 216 we will perform a simulation in the Arrival Autonomous Racing Simulator's environment by letting
 217 the car take an action, then run all the way until it cannot progress any further, and then collect the
 218 feedback as the replay buffer for further update of the agent. This process will be performed by each
 219 of the workers in parallel, and the collected replay buffers will be sent to the learner for parameter
 220 updates.

221 7.2 Machine Learning Models/Algorithms/Pipeline

222 7.2.1 Agents and Environments

223 We chose to use the Soft-Actor-Critic (SAC) [7] algorithm as our base reinforcement learning model
 224 due to its effectiveness in continuous control tasks and its ability to handle stochastic policies. SAC
 225 is a model-free, off-policy algorithm that optimizes a stochastic policy with entropy regularization.
 226 The variational autoencoder (VAE) [8] is used as a preprocessor to learn a compact representation of
 227 the high-dimensional image inputs from the racing simulator. This helps to reduce the dimensionality

228 of the input data, making the learning process more efficient. Furthermore, we intend to replace
229 the existing VAE with the encoders from the visual representation learning thrust to combine the
230 development of both thrusts in one.

231 In addition to the SAC algorithm, we also used the Bipedal Walker and Mountain Car agents from
232 OpenAI Gym [9] as the baseline for distributed algorithm development. These agents are simple and
233 well-established models that are commonly used in reinforcement learning research. By using these
234 agents as baselines, we can more easily compare the performance of our distributed architecture and
235 analyze any potential issues.

236 **7.2.2 Centralized Distributed Experience Collection Paradigm**

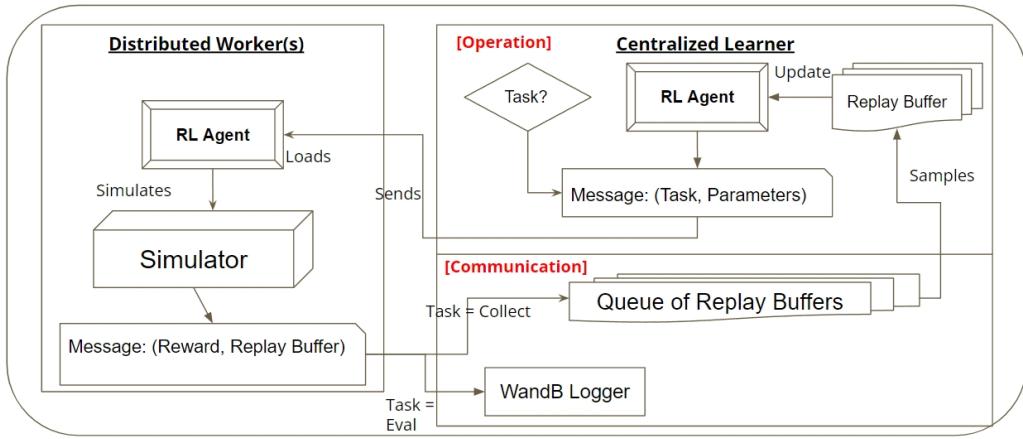


Figure 6: System pipeline for Centralized Distributed Experience Collection Paradigm

237 The first training paradigm, referred to as the Centralized Distributed Experience Collection Paradigm
238 and depicted in Figure 6, centers on the role of the centralized learner in updating the RL agent.
239 Periodically, the learner samples a batch of experiences from the replay buffer, which originates from
240 distributed workers. The RL agent outputs its current parameters and a binary task, indicating whether
241 the worker should engage in experience collection via simulated steps or perform evaluation greedily.
242 These details are encapsulated in a message and broadcasted to the distributed workers. It's crucial to
243 note that, given the presence of multiple workers, the learner can allocate diverse tasks to different
244 workers, such as some workers focusing on evaluation while others engage in experience collection.
245 However, the shared parameters must remain consistent across all workers to ensure synchronization.

246 Upon receiving parameters from the centralized learner, the distributed workers load them into their
247 individual RL agent copies, overwriting any prior parameters to replicate an identical RL agent
248 state. Depending on the allocated task, a worker will either collect experience by probabilistically
249 exploring the action space or conduct evaluation by taking the optimal greedy action and recording
250 the corresponding reward. Finally, the worker encapsulates the reward and collected experience
251 within a replay buffer, sending this information back to the centralized learner.

252 It's worth noting that the centralized learner concurrently runs two sub-processes: the *operation*
253 process, responsible for updating the RL agent and broadcasting messages, and the *communication*
254 process, continuously listening to feedback messages from the workers. In this paradigm, the
255 communication process aggregates all messages from all workers, placing them into a large queue of
256 replay buffers, which the operation process periodically fetches and samples.

257 In summary, for this distributed training paradigm, the learner's responsibility is three-fold: (1)
258 updating the RL agent by continually sampling batches of experience from the replay buffer; (2)

259 allocating tasks and broadcasting the tasks along with its latest parameters to the distributed workers;
 260 (3) collecting and storing the responses from the workers. In this case, the bottleneck exists for (1),
 261 where the replay buffer can be overflowed due to an abundance of workers working collaboratively to
 262 generate experience, whereas the learner is overwhelmed due to the slower process of updating the
 263 parameters.

264 **7.2.3 Decentralized Distributed Parameter Update Paradigm**

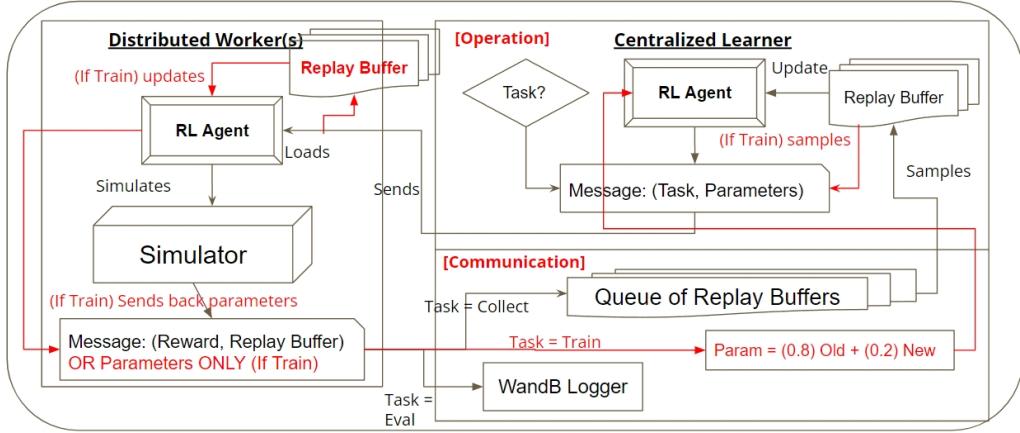


Figure 7: System pipeline for Decentralized Distributed Parameter Update Paradigm

265 For the second training paradigm, we introduce modifications to the first paradigm aimed at alleviating
 266 the parameter update workload from the centralized server. Two significant changes are implemented:
 267 first, the introduction of a new task type called "training," assigning the responsibility of parameter
 268 updates to distributed workers; second, the addition of replay buffers to these workers, allowing
 269 them to locally accumulate collected experiences for subsequent sampling and updating. As depicted
 270 in Figure 7, the highlighted modules represent the newly integrated components. When the task
 271 assigned is training, a distributed worker updates its local RL agent by sampling a batch of experience
 272 from its local replay buffer. In this process, it skips the simulation step since it doesn't interact with
 273 the environment during parameter updates. The worker then encapsulates the updated parameters into
 274 a message, sending it to the learner. The learner, in turn, performs a weighted update of its own RL
 275 agent. Notably, this update is less intricate than a standard update involving gradient calculations—it
 276 simply entails overwriting parameters. In adopting this approach, we effectively distribute the
 277 sampling and updating procedure from the learner to the workers, allowing the workers to assume
 278 this responsibility and mitigating the bottleneck on the learner. In this paradigm, the learner's role
 279 is akin to that of a server, primarily tasked with assigning and coordinating responsibilities among
 280 workers and serving as a container for the most up-to-date RL parameters. These parameters are
 281 periodically broadcasted to the workers for RL agent synchronization through the evaluation task.

282 **7.3 Evaluation Metrics**

283 Since the distributed RL thrust focuses most on the effectiveness of the system design level of the
 284 project, the performance of the RL agent both in terms of convergence speed and convergence quality,
 285 is considered as the primary evaluation metric. The performance of the VAE is not considered a
 286 metric within our scope. The relevant metrics include:

- 287 1. Total Distance: The total distance that the car has traveled during an episode.
 288 2. Total Time: The total time taken to complete the race by the car during an episode.

- 289 3. Number of Infractions: The number of infractions committed by the car during an episode,
 290 such as collisions with track barriers or other cars.
 291 4. Average Speed: The average speed of the car during an episode, measured in kilometers per
 292 hour (kph).
 293 5. Average Displacement Error: The average distance between the car’s position and the ideal
 294 racing line during an episode.
 295 6. Trajectory Efficiency: A measure of how well the car is able to follow the racing line during
 296 an episode, expressed as a percentage.
 297 7. Trajectory Admissibility: A measure of how closely the car is able to follow the racing line
 298 during an episode, expressed as a percentage.
 299 8. Movement Smoothness: A measure of how smoothly the car moves during an episode.
 300 9. Timestep/sec: The number of timesteps per second during an episode.
 301 10. Laps Completed: The number of laps completed by the car during an episode.

302 8 Test Design and Model Deployment

303 8.1 Experiment Launching

304 As emphasized earlier, the automation of resource control is critical for complex distributed training
 305 paradigms, facilitating seamless baseline reproduction without entangling users in intricate Kubernetes
 306 YAML file configurations. As depicted in Figure 8, users can initiate a Python script, input their
 307 RL environment and training paradigm, specify the number of workers for distributed paradigms,
 308 and provide a name for the Weights and Biases experiment. Subsequently, the script dynamically
 309 generates a Kubernetes YAML file based on the user’s input configuration and launches the setup.
 310 Notably, a key optimization is the elimination of static and hard-coded YAML file configurations
 311 from the previous iteration, replaced by the on-the-fly generation of YAML files. This not only
 312 significantly reduces the number of required YAML files but also simplifies the process of propagating
 313 configuration changes.

```
sbian@phoebe-login:~/distributed$ python3 launch.py
Select RL environment (mcar/walker/l2r): mcar
Select training paradigm (sequential/dCollect/dUpdate): dUpdate
Input number of distributed workers: 1
Input WandB experiment name: mcar-dupdate-1worker-test
-----
RL Environment = [mcar] | Training Paradigm = [dUpdate] | Number of Workers = [1] | Experiment Name = [mcar-dupdate-1worker-test] ---
-----
replicaset.apps/mcar-dupdate-workers created
pod/mcar-dupdate-learner created
service/mcar-dupdate-learner created
```

Figure 8: Automated launching process for model deployment and experiment initiation

314 8.2 Experiment Termination

315 Manual termination of distributed resources poses challenges, as users must locate specific pods,
 316 services, and replicsets, and execute Kubernetes commands for termination. A simple typo in these
 317 commands can inadvertently lead to the deletion of an unintended pod, risking the loss of crucial
 318 training progress. In response to this challenge, we’ve developed an automated script, showcased
 319 in Figure 9, which gracefully shuts down and terminates all pertinent resources based on the user’s
 320 specified configurations. This automated approach minimizes the risk of inadvertent errors and
 321 enhances the efficiency of resource shutdown procedures.

322 9 Risks/Challenges

323 The risks and challenges of the proposed thrust for the learn-to-race project is three-fold:

```

sbian@phoebe-login:~/distributed$ python3 shutdown.py
Select RL environment (mcar/walker/l2r): mcar
Select training paradigm (sequential/dCollect/dUpdate): dUpdate
---
--- Shutting down [mcar] with training paradigm [dUpdate] ---
---
replicaset.apps "mcar-dupdate-workers" deleted
service "mcar-dupdate-learner" deleted
pod "mcar-dupdate-learner" deleted

```

Figure 9: Automated termination process for model deployment and experiment initiation

324 9.1 Business Risks

325 During the consideration of product design aspects, a notable business risk has been identified. The
 326 product is targeted for deployment in competitive environments where businesses or individuals
 327 rely heavily on its performance. Any shortcomings in the product's performance, whether in a
 328 simulated or real-world racing scenario, could have adverse effects on the business or user experience.
 329 Potential issues, such as slow responsiveness or the inability to deliver real-time inputs, especially in
 330 challenging weather conditions, may lead to a decrease in the user's fanbase and a loss of interest
 331 from sponsors. This, in turn, could result in negative financial repercussions for the business or
 332 individual involved. Addressing and mitigating these performance-related risks is critical to ensuring
 333 the product's success and maintaining stakeholder satisfaction.

334 9.2 Product Risks

335 We've pinpointed a critical product risk that could lead to failure when the product is deployed in a
 336 real car participating in a competition, whether with or without a human driver. In the event of an
 337 incorrect prediction during a race that leads to a crash, there is a potential for loss of life or damage
 338 to infrastructure, depending on the circumstances. Such an incident would deem the product unsafe
 339 for real-world use and could result in substantial financial losses for the business. Consequently, it is
 340 of utmost importance to proactively prevent such situations before deploying the product in real-life
 341 scenarios, emphasizing thorough testing and validation procedures.

342 9.3 Project Risks

343 The project's scope is inherently limited by constraints such as the available data, short project
 344 duration, and budget considerations. Collecting an exhaustive set of data to enable a machine
 345 learning model to adapt to every conceivable use case is not feasible within these limitations. As
 346 a consequence, there might be instances in both real-world and simulated environments where the
 347 model struggles to comprehend and adapt to unfamiliar conditions. For instance, if the model is
 348 exclusively trained in sunny environments and encounters sudden foggy conditions, its performance
 349 may deviate from intended outcomes, potentially yielding inaccurate or only partially accurate results.
 350 This lack of adaptability introduces a challenge, potentially rendering the product less reliable and
 351 robust, particularly in diverse environmental conditions.

352 10 Tools and Dependencies

353 10.1 Dataset

- 354 1. The Learn-to-Race framework utilizes simulations of three real-world racetracks, adding a
 355 layer of authenticity to the racing experience:

- 356 (a) North Road track at Las Vegas Motor Speedway in the United States.
 357 (b) Thruxton Circuit track located in the United Kingdom.
 358 (c) Anglesey National, modeled after the track at the Anglesey Circuit.
 359 2. Each dataset comprises images captured by the vehicle's camera.
 360 3. For instance, the Thruxton dataset encompasses 10,600 complete transitions, including
 361 sensor data, camera images, and actions executed by the Model Predictive Control (MPC)
 362 agent. This data was gathered over 9 full laps around the Thruxton circuit track.
 363 4. All images are presented in RGB format, with dimensions of 512 pixels in width, 384 pixels
 364 in height, and a field of view spanning 90 degrees.

365 **10.2 Libraries/Services**

- 366 1. Our Learn-To-Race framework will be developed using Python 3.8, adhering to current
 367 standards.
 368 2. The framework will leverage key Python libraries:
 369 (a) PyTorch version 1.7.1 for the creation of our deep learning models.
 370 (b) Torchvision for image transformation in computer vision tasks.
 371 (c) Numpy version greater than 1.19.2 for various computations.
 372 (d) Matplotlib version greater than 3.3.2 for visualizing the racetrack map.
 373 3. The Learn-To-Race framework mandates GPU support to facilitate rapid visual feature
 374 extraction and the training of our safe Reinforcement Learning algorithms.
 375 4. Docker is a prerequisite for the framework, as the simulator typically operates within a
 376 Docker container.
 377 5. Additionally, running the simulator within a Docker container necessitates the use of Nvidia-
 378 container-runtime to access the GPU from the container.

379 **11 Progress/Results**

380 **11.1 Training Paradigm Implementation**

381 As a recap of the past semester, we conducted a comprehensive performance analysis of the original
 382 distributed learning architecture, focusing on the L2R, bipedal walker, and mountain car RL agents.
 383 As outlined in the preceding sections, a significant challenge emerged in the form of deteriorating
 384 training performance, even with an increased number of distributed workers. Notably, there was no
 385 substantial acceleration in convergence, and in some instances, models failed to converge even after
 386 an extended duration, despite scaling up the number of workers engaged in parallel data collection
 387 and model evaluation. The suspected root cause of these issues pointed to the centralized learner
 388 acting as a bottleneck, given that the model parameter updates were taking longer than the evaluation
 389 and collection processes. To address this concern, we introduced the refined *decentralized distributed*
 390 *parameter update paradigm*, a modification of the original system, as illustrated in Figure 7.
 391 Several major refinements have been implemented in our system to enhance its functionality: (1)
 392 Introducing a novel task type called *Train*; when the learner assigns this task to coordinate the
 393 workers, it includes a message containing a list of replay experiences sampled from its own replay
 394 buffer; (2) Upon receiving a *Train* task, a worker loads the parameters, performs parameter update
 395 (training) based on the provided list of experiences, and generates a response message with the trained
 396 parameters; (3) After training, the worker directly sends the response message back to the learner,
 397 bypassing any simulation steps; (4) The learner executes a damped update of its parameters using
 398 the formula $P_{n+1} = 0.8 * P_n + 0.2 * P_{new}$, where n denotes the $n - th$ update iteration, and P_{new}
 399 represents the newly trained parameters obtained from the worker.

400 11.2 Code Base Unification

401 As highlighted earlier, a key aspect of our system is to furnish users with a unified and modularized
 402 codebase. This entails consolidating similar functionalities into shared modules or classes, fostering
 403 a cohesive and adaptable structure. Illustrated in Figure 10, five main steps were undertaken to
 404 achieve this unification. In the initial step, we streamlined the number of branches. In the preceding
 405 iteration, various combinations of RL environments and training paradigms were housed in separate
 406 branches, complicating the propagation of framework changes. By consolidating these branches into
 407 two—dedicated to distributed and sequential paradigms—we merged them seamlessly. This unifica-
 408 tion allows users to access and modify all code within a singular branch. The second step focused on
 409 unifying Kubernetes launching YAML files. Previously, each configuration had a dedicated YAML
 410 file, a complexity mitigated by our automatic resource creation template generation, reducing myriad
 411 combinations to two basic templates. Steps three and four addressed the unification of the worker
 412 runner class. Previously, distinct classes for each training paradigm led to redundancy and intricate
 413 modifications. We amalgamated these classes into one, capable of inferring the training paradigm
 414 based on input, simplifying worker configuration YAML unification. Lastly, we homogenized learner
 415 and worker definitions, eradicating any redundancies.



Figure 10: Procedure on unifying distributed training paradigms within learn-to-race RL environment space

416 11.3 Automated Resource Control

417 Another significant advancement involves the implementation of automated resource control, as
 418 detailed in Section 8. In the previous iteration, hardcoded pod and service names led to port conflicts
 419 when launching multiple configurations simultaneously, impeding communication between workers
 420 and their respective learners. In our revised iteration, we automated both resource creation and

421 termination processes. Additionally, we dynamically assign port and resource names to services
 422 based on the user’s experiment name input. This not only simplifies resource tracking but also
 423 resolves conflicts, enabling concurrent testing of multiple RL environments and training paradigm
 424 combinations.

425 **11.4 Experimental Results**

426 To assess our two distributed training paradigms, we conducted a total of nine experiments, en-
 427 compassing various combinations of RL environments {learn-to-race, mountain car, bipedal walker,
 428 bipedal walker with OpenAI’s vanilla SAC agent, and lunar lander with OpenAI’s vanilla SAC
 429 agent} and training paradigms {sequential, distributed collection, distributed update}, with the results
 430 summarized in Figure 11. In the subsequent sections, I will present the latest results for each of the
 431 RL environments, along with high-level observations and analysis.

● Legend: ✓ means convergence, ~ means non-convergence

RL Environments	Sequential	Distributed Collection	Distributed Update
Mountain Car	✓ (slow, ~100 h)	✓ (faster with more workers)	✓ (faster with less workers)
Bipedal Walker	~ (reward < 0)	~ (periodical surges)	~ (can cross 0, fluctuations)
Bipedal Walker (OpenAI)	~ (rough converge to reward near 0)	~ (no convergence)	~ (no convergence)
Lunar Lander (OpenAI)	✓ (~10 h, no sign of degradation)	✓ (reach high reward <2h, with <i>slight degradation</i>)	~ (reach high reward <1h, but then degradation)
Learn-to-Race	✓ (speed KPH)	~ (stay at low performance)	~ (periodic improvement)

Figure 11: The combinations of RL environments and training paradigms, along with summarized experimental results

432 **11.4.1 Mountain Car Results**

433 We experimented with the updated architecture on the **Mountain Car** RL agent first, since it is
 434 the most fundamental RL agent that can serve as a sanity check. As recorded in Figure 12, we
 435 observe a significant increase in convergence speed with a scaling of the number of workers for the
 436 Mountain Car RL agent (where the sequential version takes around 100 hours to converge, while
 437 both distributed paradigms achieved convergence within 40 hours), which indicates that the updated
 438 architecture passed the sanity check and can converge with boosting from more paralleled training.
 439 Furthermore, we observed an interesting phenomenon where the distributed update paradigm works
 440 better with less number of workers, while the distributed collection paradigm converges fastest with
 441 more workers. We hypothesis that this is because the mountain car environment is easy to solve, so it
 442 does not benefit much from the distributed parameter update paradigm, which may induce further
 443 network communication overheads. In this case, a distributed experience collection paradigm that
 444 retains centralized parameter update scales the best.

445 **11.4.2 Bipedal Walker Results**

446 We also conducted experiments on the bipedal walker RL environment, which is a much more difficult
 447 environment, and compared the two training paradigms, as depicted in Figure 13. We observe that
 448 there is no significant convergence on a global scale for both paradigms. However, for the centralized

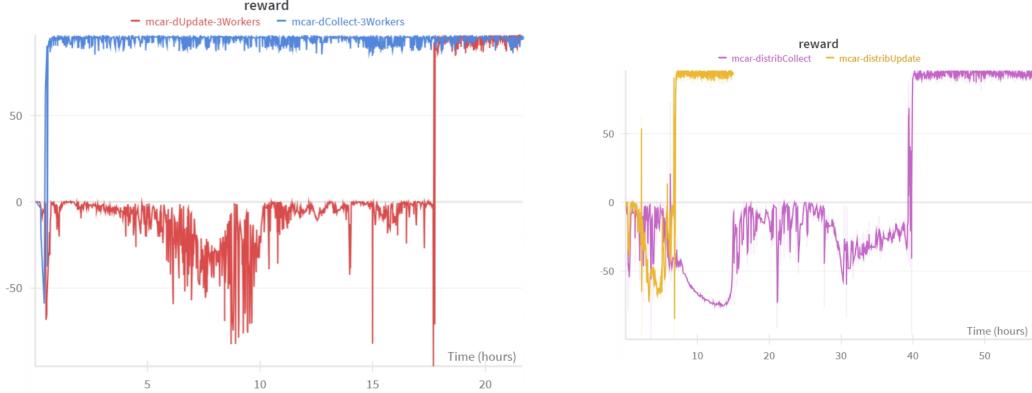


Figure 12: Experimental results for the **Mountain Car** environment, solved by the distributed paradigms with 3 workers (left) and 1 worker (right)

449 distributed collection paradigm, there are periodic surges of rewards, indicating that the agent is
 450 periodically learning to stand up but experiences forgetting of their experience and degradation of
 451 performance. In comparison, for the decentralized update paradigm, there is more fluctuation in the
 452 reward, and we see temporary convergence to a non-negative value before the reward drops once
 453 again, indicating that the agent can learn successfully at one point but suffers from a similar forgetting
 454 as before.

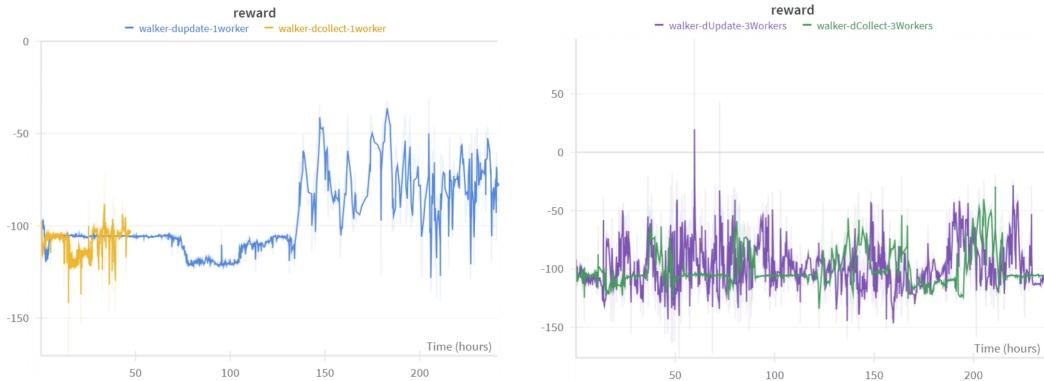


Figure 13: Experimental results for the **Bipedal Walker** environment, solved by the distributed paradigms with 3 workers (right) and 1 worker (left)

455 11.4.3 Lunar Lander Results

456 In our effort to test the distributed paradigms with more comprehensive environments, we conducted
 457 experiments on the Lunar Lander RL environment with OpenAI spinning-up vanilla implementation
 458 of the SAC agents, with the results recorded in Figure 14. We observed that both distributed training
 459 paradigms can achieve high rewards within a very short period, but suffer from degradation forgetting,
 460 something that is also observed in the bipedal walker case. From these results, we hypothesis that
 461 the distributed training paradigms, having the ability to collect diverse experiences from multiple
 462 environments in parallel, has the potential to achieve higher reward in a shorter amount of time than
 463 sequential paradigms, yet suffers from forgetting possibly due to the more frequent reset and update
 464 of the parameters, thus cannot maintain the high rewards parameter states.

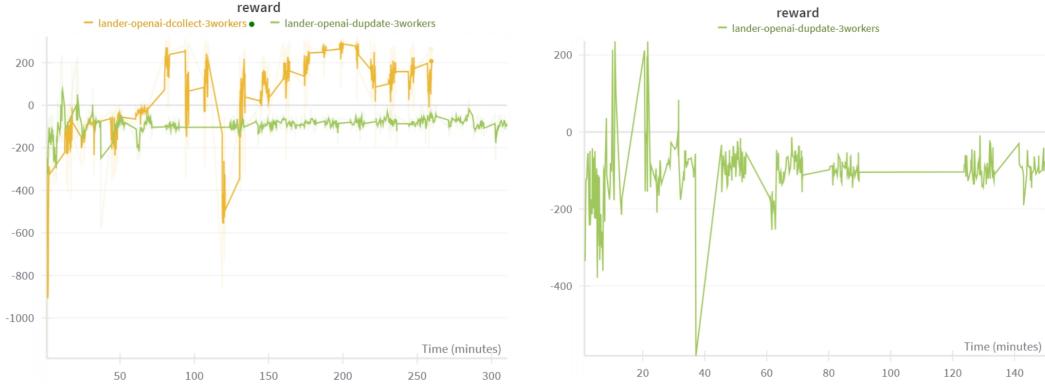


Figure 14: Experimental results for the **Lunar Lander** environment, solved by the distributed paradigms with 3 workers

465 11.4.4 Learn-to-race (L2R) Results

466 Finally, we present a comparison of the learn-to-race environment agent convergence performance
 467 between the two training paradigms in Figure 15. We focus on two main metrics: *Average Speed*
 468 (*KPH*), representing the average speed of the car during a simulation episode, and *Reward*, calculated
 469 as how well the car maintains on the central line of the race lane (and each time it collides with the
 470 edges of the lane, rewards are deducted). The expectation is that both metrics increase as the car/agent
 471 learns to navigate the environment effectively. From the figure, a notable observation emerges. The
 472 decentralized distributed update paradigm exhibits signs of convergence, evident in the increasing
 473 values of both speed and distance. However, this convergence or improvement in metrics happens
 474 periodically and drops to a valley before re-emerging to a high value. These periodic fluctuations
 475 align well with our observations in the prior environments.

476 In contrast, the centralized distributed collection paradigm appears to be stuck at a lower performance
 477 level. We hypothesize that the superior performance of the distributed update paradigm can be
 478 attributed to the inherent difficulty of the learn-to-race environment. In this paradigm, each worker
 479 can independently update and train its local copy of the RL agent before merging parameters at the
 480 learner. This approach facilitates more diverse and parallelized learning, particularly beneficial for a
 481 complex and challenging environment. Further investigation into this observation will be explored in
 482 the subsequent error analysis section.

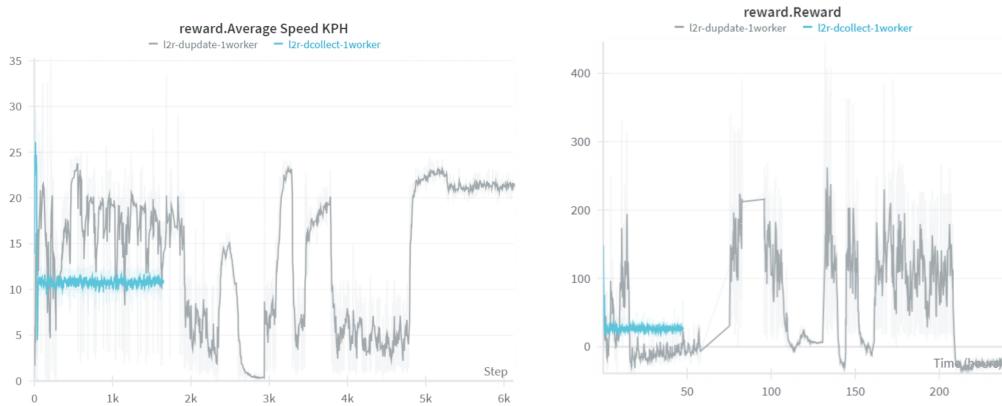


Figure 15: Experimental results for the **Learn-to-Race** environment, solved by the distributed paradigms

483 **12 Error Analysis**

484 From the results discussed in the prior section, we observed that differences across RL environments,
 485 training paradigms, and the number of distributed workers significantly influence the training perfor-
 486 mance. Therefore, we summarized the results in Figure 11. In the mountain car RL environment,
 487 which is considered one of the easiest environments available, our analysis suggests that there is
 488 no fundamental issue with the training paradigm, as indicated by the observed convergence of the
 489 agent within this straightforward environment. Specifically, for the centralized distributed update
 490 paradigm, the addition of more workers seems to lead to a scalable increase in convergence speed
 491 (Figure 12), but only when compared to experiments run on the same training paradigm. The reason
 492 why the decentralized paradigm converged slower than the centralized paradigm when incorporating
 493 multiple workers may be explained by the overhead of message and parameter communication be-
 494 tween the workers and the learners, which becomes redundant and particularly burdensome when the
 495 environment is easily solvable. This leads to the observation and hypothesis that *for easily solvable*
 496 *environments, the centralized distributed paradigm may outperform the decentralized paradigm.*

497 Moving on to analyzing the results of the bipedal walker, as mentioned above in Figure 13, we observe
 498 that the non-convergence issue persists with both paradigms. However, it is worth noting that the
 499 decentralized distributed parameter update paradigm, despite leading to more fluctuations and reward
 500 noises due to the parallel update of parameters, achieved higher convergence than the centralized
 501 distributed experience collection paradigm. We hypothesize that this is due to the alleviation of the
 502 workload on the centralized learner, which is significant when the environment is hard to solve. The
 503 non-convergence issue, however, may be more due to hyperparameter tuning.

504 For the performance analysis of the learn-to-race agent, the non-convergence issue observed in the
 505 distributed collection paradigm may be attributed to two potential factors. First, the communication
 506 overhead could be a contributing factor. Figure 16 demonstrates that when multiple workers are
 507 operational, and the data being transmitted is substantial (as is the case with the complex state
 508 representation in learn-to-race), the time taken for communication becomes a notable bottleneck. This
 509 potentially impedes the timely update of parameters on the learner server. Additionally, the intricate
 510 nature of the learn-to-race environment, coupled with the sensitivity of the SAC RL algorithm to
 511 initializations, requires careful tuning of a multitude of hyperparameters to mitigate the impact of
 512 unfavorable initializations. Further investigation into these aspects is essential to address the observed
 513 non-convergence in the distributed collection paradigm.

Timing	Data preparation time: 2.3935 s
Timing	Data sending time: 3.4823 s
[TRAIN]	Param. Mean = 1.2856837124563754, Param. Std = 21.07892442587763
Timing	Data preparation time: 2.2177 s
Timing	Data sending time: 3.1783 s
[TRAIN]	Param. Mean = 1.2874411861412227, Param. Std = 21.084575068205595
Timing	Data sending time: 161.8266 s
[TRAIN]	Param. Mean = 1.2816175920888782, Param. Std = 21.08507408481273
Timing	Data preparation time: 2.1687 s
[COLLECT]	Buffer Size = 102
Sampling	Epoch = 579 -> Sampled Buffer = 102 from Replay Buffer = 50000, where
Timing	Data sending time: 153.6975 s
[TRAIN]	Param. Mean = 1.2847793616820127, Param. Std = 21.07869869656861
[COLLECT]	Buffer Size = 102

Figure 16: Latency issue due to sending large messages, observed in learn-to-race experiments

514 13 Discussion

515 Recorded in Figure 17 are our observations and analytical conclusions derived from the extensive
 516 combinations of experiments. Firstly, we observed that for the easier RL environments, such as
 517 *mountain car*, the distributed paradigms retain the ability to converge, while successfully accelerating
 518 the training speed. We also observe that the distributed update decentralized paradigm suffers from
 519 communication and worker coordination overhead, which may mean that for easier environments,
 520 falling back to the centralized parameter update is more beneficial due to less randomness and
 521 fluctuation of agent model parameters.

522 Furthermore, for medium and high difficulty RL environments, such as *lunar lander* and *bipedal*
 523 *walker* respectively, we start to observe that the distributed paradigms, despite being able to reach
 524 high rewards, cannot retain the high performance without suffering from degradation. We analyze
 525 and hypothesize that the faster efficiency in achieving high rewards may be due to the distributed
 526 training paradigms' ability to collect experience from diverse environments located within each
 527 worker at the same time. So in comparison with the sequential version where the agent performs
 528 experience sampling by simulating in a fixed environment, the distributed paradigms allow more
 529 opportunity for the agents to explore in different ways, thus giving it a better chance to find an optimal
 530 action within a certain state space. This allows the learner to learn diverse and useful parameters,
 531 inducing fluctuations but also the potential to reach high rewards. However, due to the frequent
 532 and decentralized update of the parameters, it is highly likely that *a newly learned parameter from*
 533 *a certain worker is lost due to being updated by another worker at the same time, causing the*
 534 *performance to degrade as the agent forgets the newly learned knowledge*. To combat this, we believe
 535 that a potential way would be to **allow each worker to retain a local buffer for storing all the**
 536 **"good experiences" with high rewards it explored, instead of sending them to the learner and**
 537 **discarding them. Finally, the learner can aggregate the local buffers across all workers, thus**
 538 **obtaining a small pool of highly refined experience for fine-tuning.**

RL Environments	Sequential Paradigm	Distributed Paradigms
Easy Gym (mountain car)	✓ stable convergence ✗ very slow	✓ stable convergence, much faster ✗ communication overhead
Hard Gym (bipedal walker)	✓ if convergence, should stay stable ✗ often stuck at worse reward	✓ reach high reward fast (learn fast) due to diverse simulations ✗ suffer from degradation (forget fast)
Vanilla SAC	✓ better convergence performance than non-Vanilla	✗ generally poor behavior (need to investigate further)
Learn-to-Race	✓ observed stable improvement ✗ often stuck at worse reward	✓ observed periodical improvement, faster in comparison ✗ fluctuations, unstable

Figure 17: Summary of observations and analysis from experimental evaluation.

539 14 Lessons Learned and Reflections

540 A critical reflection on this research thrust underscores the indispensable role played by our earlier
 541 documents in establishing a robust foundation for the capstone project and shaping this final report.
 542 The *Vision Document* served as a compass, defining the project's scope, goals, and objectives
 543 within the distributed thrust of Learn-to-Race. This enabled me to concentrate on tasks tailored
 544 to my thrust, unaffected by the progress of other thrusts. The *Requirement Document* proved

545 instrumental in identifying and prioritizing project requirements, providing a framework for setting
546 goals and planning milestones. The *Design Document* deepened my understanding of modularized
547 dependencies and illuminated potential risks inherent in my design. Lastly, the *Plan Document*
548 facilitated synchronization across research thrusts.

549 A pivotal lesson learned centers on the paramount importance of communication and timely updates
550 with my supervisor for the success and progress of my research thrust. During a critical phase
551 in development, I encountered challenges in navigating a convoluted and disorganized code base.
552 Seeking assistance from my research assistant mentor proved invaluable, as they aided in reorganizing
553 and modularizing a specific module of my design. This structured approach was then applied to refine
554 all other designs. Going forward, I recognize the necessity of more frequent communication with
555 supervisors and teammates to ensure alignment with requirements, understand progress, and address
556 concerns effectively.

557 15 Future Work

558 Integrating distributed systems design and deep reinforcement learning (RL) within the complex do-
559 main of autonomous racing presents a multifaceted challenge. As observed in the current study, there
560 is no one-size-fits-all distributed paradigm that universally resolves the intricacies of the learn-to-race
561 environment. Future endeavors in this field should entail a thorough exploration of additional tuning
562 parameters and a deeper investigation into the interactions between distributed system components
563 and RL algorithms. The observed non-convergence issues in certain RL environments, such as the
564 learn-to-race scenario, prompt extensive hyperparameter tuning. The sensitivity of RL algorithms,
565 particularly the SAC algorithm, to initialization parameters underscores the importance of meticulous
566 optimization. Future work should delve into systematic hyperparameter tuning and optimization
567 strategies to enhance the robustness and convergence speed of the distributed training paradigms. The
568 current study focused on four diverse RL environments (learn-to-race, mountain car, lunar lander,
569 and bipedal walker). However, expanding the exploration to a broader array of RL environments will
570 provide a more comprehensive understanding of the strengths and weaknesses of different distributed
571 paradigms. Assessing the scalability and adaptability of the proposed paradigms across a spectrum of
572 environments will contribute to their refinement and generalizability.

573 16 Conclusion

574 In conclusion, our efforts to develop robust and efficient solutions for autonomous driving have
575 evolved from a combined perspective of reinforcement learning and system design. The proposed
576 thrust, Distributed Reinforcement Learning with Optimization, aims to enhance the scalability
577 of the reinforcement learning training process by distributing tasks across multiple workers and
578 incorporating real-time visualization for increased transparency. Our innovative approach to training
579 autonomous racing cars using distributed reinforcement learning enables the distribution of workloads
580 and simulations to individual workers, achieving parallel acceleration and fully leveraging scalable
581 computing power.

582 In this work, we introduced two distributed training paradigms designed to scale the training efficiency
583 of agents within the learn-to-race project environment by delegating tasks to workers and coordinating
584 them with a learner. First, we proposed the *centralized distributed experience collection paradigm*,
585 where workers perform experience collection and RL agent evaluation within their local simulation
586 environments. The learner then collects feedback from the workers and performs a complete update
587 of its RL agent with sampled experiences. Second, we introduced the *decentralized distributed*
588 *parameter update paradigm*, where workers take on the additional responsibility of updating the
589 parameters for their local RL agents. The learner performs a weighted overwrite of its RL agent
590 based on parameters collected from the workers, thereby reducing the workload on the learner for
591 full updates through batch sampling.

592 Experimental results on three different RL environments and training paradigms provided valuable
593 insights into the effectiveness of these paradigms and revealed bottlenecks and limitations, prompting
594 considerations for future work in this area. Additionally, the refactoring and cleaning of the code
595 base through unification and modularization enhance the user experience, allowing researchers and
596 users to interact with the system more easily and facilitating the testing of various combinations of
597 RL environments and training paradigms.

598 17 Acknowledgments

599 We would like to sincerely thank our mentors, Jonathan Francis and Arav Agrawal, for their con-
600 tinuous feedback and invaluable guidance, steering us in the right direction throughout the project.
601 Additionally, we appreciate Prof. Eric Nyberg for providing a structured outline of the steps to follow.
602 Special thanks to Neeraj Panse for his exceptional support as our TA, patiently addressing our doubts
603 and queries, and contributing significantly to our learning experience throughout the course.

604 18 Terminology, Definitions, Acronyms, and Abbreviations

- 605 1. Reinforcement learning (RL): a machine learning training procedure in which an agent
606 learns how to take actions within an environment based on reward and penalty feedback.
- 607 2. Agents: an intelligent artificial entity can perceive its environment and respond reasonably.
- 608 3. State: the parameters that define the status of the agent.
- 609 4. Action: the action taken by the agent based on its environment and state to maximize the
610 reward function.
- 611 5. Environment: the environmental feedback that the agent is within.
- 612 6. Reward function: a formulated representation indicating the gain of the agent at a certain
613 state and environment through taking a certain action.

614 References

- 615 [1] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward,
616 Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu.
617 Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.
- 618 [2] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous
619 vehicles: Problems, datasets and state of the art, 2017.
- 620 [3] Kapthpal Sidharth, Agarwal Arav, Qin Yujin, Ganey Tanay, Chian Kevin, Francis Jonathan,
621 Ganju Siddha, Koul Anirudh, Chen Bingqing, Oh Jean, and Nyberg Eric. Learn-to-race: A
622 multimodal control environment for autonomous racing. *CMU MCDS Capstone Project Final
623 Report*, 2022.
- 624 [4] Mohammad Reza Samsami and Hossein Alimadad. Distributed deep reinforcement learning: An
625 overview, 2020.
- 626 [5] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De
627 Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane
628 Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for
629 deep reinforcement learning, 2015.
- 630 [6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap,
631 Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep
632 reinforcement learning, 2016.
- 633 [7] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy
634 maximum entropy deep reinforcement learning with a stochastic actor, 2018.

- 635 [8] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations*
636 *and Trends® in Machine Learning*, 12(4):307–392, 2019.
- 637 [9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang,
638 and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

639 **A Changes To Previous Deliverables**

640 A pivotal adjustment in our methodology involved the adoption of a new Docker image, specifically,
641 *jingyua*, in terms of the previously utilized *l2r2022* image from the prior iteration. This transition was
642 prompted by the apparent performance improvements observed with the *jingyua* image, particularly
643 in the learn-to-race environment. Initial tests revealed that the learn-to-race agent exhibited a higher
644 initial speed (measured in KPH) when using the updated Docker image. The exact factors contributing
645 to this performance disparity are still under investigation, with our current hypothesis leaning toward
646 additional CUDA support libraries in the *jingyua* image or potential improvements in the simulator
647 version.

648 Additionally, substantial modifications were implemented in our GitHub repositories. These changes
649 aimed to streamline and enhance the efficiency of our development process. The consolidation
650 of branches, automation of resource creation and termination files, and the unification of modules
651 independent of training paradigms were key strategies employed to simplify the codebase and foster
652 a more cohesive and manageable development environment. These enhancements ensure that future
653 modifications and additions to the codebase can be executed with greater ease and clarity.