aws

Developer Guide

AWS DeepRacer

AWS DeepRacer: Developer Guide

Table of Contents

What is AWS DeepRacer?

AWS DeepRacer is a fully autonomous 1/18th scale race car driven by reinforcement learning. It consists of the following components:

- AWS DeepRacer console: An AWS Machine Learning service for training and evaluating reinforcement learning models in a three-dimensional simulated autonomous-driving environment.
- AWS DeepRacer vehicle: A 1/18th scale RC car capable of running inference on a trained AWS DeepRacer model for autonomous driving.
- AWS DeepRacer League: The world's first global, autonomous racing league. Race for prizes, glory, and an opportunity to advance to the World Championship Cup. For more information, see the terms and conditions.

Topics

The AWS DeepRacer console

The AWS DeepRacer console is a graphical user interface for interacting with the AWS DeepRacer service. You can use the console to train a reinforcement learning model and to evaluate the model performance in the AWS DeepRacer simulator. In the console, you can also download a trained model for deployment to your AWS DeepRacer vehicle for autonomous driving in a physical environment.

In summary, the AWS DeepRacer console supports the following features:

- Create a training job to train a reinforcement learning model with a specified reward function, optimization algorithm, environment, and hyperparameters.
- Choose a simulated track to train and evaluate a model by using SageMaker AI.

- Clone a trained model to improve training by tuning hyperparameters to optimize your model's performance.
- Download a trained model for deployment to your AWS DeepRacer vehicle so it can drive in a physical environment.
- Submit your model to a virtual race and have its performance ranked against other models in a virtual leaderboard.

When you use the AWS DeepRacer service console you are charged based on your usage to train or evaluate and store models.

To get you started, AWS DeepRacer provides a Free Tier to first time AWS DeepRacer users. This is enough time to train and tune your first model and enter the AWS DeepRacer League. There is no cost for submitting a model to take part in any AWS DeepRacer League virtual event.

For details about pricing see the AWS DeepRacer service detail page.

The AWS DeepRacer vehicle

The AWS DeepRacer vehicle is a Wi-Fi enabled, physical vehicle that can drive itself on a physical track by using a reinforcement learning model.

- You can manually control the vehicle or deploy a model for the vehicle to drive autonomously.
- The autonomous mode runs inference on the vehicle's compute module. Inference uses images that are captured from the camera that is mounted on the front.
- A Wi-Fi connection allows the vehicle to download software. The connection also allows the user to access the device console to operate the vehicle by using a computer or mobile device.

The AWS DeepRacer League

The AWS DeepRacer League is an important component of AWS DeepRacer. The AWS DeepRacer League is intended to foster community and competition.

With the AWS DeepRacer League, you can compare your ML skills with other AWS DeepRacer developers in a physical or virtual racing event. Not only do you have the opportunity to earn prizes and achievements, you also have a way to measure your reinforcement learning models.

You can compete with other participants, learn from each other, and inspire each other. If you win achievements for your performance in the AWS DeepRacer League, you can share them with your community on social media. For more information, see the terms and conditions.

Join a race or learn how to train a model in the League.

Use AWS DeepRacer to explore reinforcement learning

Reinforcement learning, especially deep reinforcement learning, has proven effective in solving a wide array of autonomous decision-making problems. It has applications in financial trading, data center cooling, fleet logistics, and autonomous racing, to name a few.

Reinforcement learning has the potential to solve real-world problems. However, it has a steep learning curve because of its extensive technological scope and depth. Real-world experimentation requires that you construct a physical agent, such as an autonomous racing car. It also requires that you secure a physical environment, such as a driving track or public road. The environment can be costly, hazardous, and time-consuming. These requirements go beyond merely understanding reinforcement learning.

To help reduce the learning curve, AWS DeepRacer simplifies the process in three ways:

- Offering step-by-step guidance when training and evaluating reinforcement learning models. The guidance includes pre-defined environments, states, and actions, and customizable reward functions.
- Providing a simulator to emulate interactions between a virtual agent and a virtual environment.
- Using an AWS DeepRacer vehicle as a physical agent. Use the vehicle to evaluate a trained model in a physical environment. This closely resembles a real-world use case.

If you are a seasoned machine learning practitioner, you will find AWS DeepRacer a welcome opportunity to build reinforcement learning models for autonomous racing in both virtual and physical environments. To summarize, use AWS DeepRacer to create reinforcement learning models for autonomous racing with the following steps:

1. Train a custom reinforcement learning model for autonomous racing. Do this by using the AWS DeepRacer console integrated with SageMaker AI.
2. Use the AWS DeepRacer simulator to evaluate a model and test autonomous racing in a virtual environment.

3. Deploy a trained model to AWS DeepRacer model vehicles to test autonomous racing in a physical environment.

AWS DeepRacer concepts and terminology

AWS DeepRacer builds on the following concepts and uses the following terminology.

AWS DeepRacer service

AWS DeepRacer is an AWS Machine Learning service for exploring reinforcement learning that is focused on autonomous racing. The AWS DeepRacer service supports the following features:

1. Train a reinforcement learning model on the cloud.
2. Evaluate a trained model in the AWS DeepRacer console.
3. Submit a trained model to a virtual race and, if qualified, have its performance posted to the event's leaderboard.
4. Clone a trained model to continue training for improved performances.
5. Download the trained model artifacts for uploading to an AWS DeepRacer vehicle.
6. Place the vehicle on a physical track for autonomous driving and evaluate the model for real-world performances.
7. Remove unnecessary charges by deleting models that you don't need.

AWS DeepRacer

"AWS DeepRacer" can refer to three different vehicles:

- **The virtual race car** can take the form of the original AWS DeepRacer device, the Evo device, or various digital rewards that can be earned by participating in AWS DeepRacer League Virtual Circuit races. You can also customize the virtual car by changing its color.
- **The original AWS DeepRacer device** is a physical 1/18th-scale model car. It has a mounted camera and an on-board compute module. The compute module runs inference in order to drive itself along a track. The compute module and the vehicle chassis are powered by dedicated batteries known as the compute battery and the drive battery, respectively.
- **The AWS DeepRacer Evo device** is the original device with an optional sensor kit. The kit includes an additional camera and LIDAR (light detection and ranging), which allow the car to detect objects behind and lateral to itself. The kit also includes a new shell.

Reinforcement learning

Reinforcement learning is a machine learning method that is focused on autonomous decision-making by an agent in order to achieve specified goals through interactions with an environment. In reinforcement learning, learning is achieved through trial and error and training does not require labeled input. Training relies on the *reward hypothesis*, which posits that all goals can be achieved by maximizing a future reward after action sequences. In reinforcement learning, designing the reward function is important. Better-crafted reward functions result in better decisions by the agent.

For autonomous racing, the agent is a vehicle. The environment includes traveling routes and traffic conditions. The goal is for the vehicle to reach its destination quickly without accidents. Rewards are scores used to encourage safe and speedy travel to the destination. The scores penalize dangerous and wasteful driving.

To encourage learning during training, the learning agent must be allowed to sometimes pursue actions that might not result in rewards. This is referred to as the exploration and exploitation trade-off. It helps reduce or remove the likelihood that the agent might be misguided into false destinations.

For a more formal definition, see reinforcement learning on Wikipedia.

Reinforcement learning model

A reinforcement learning model is an environment in which an agent acts that establishes three things: The states that the agent has, the actions that the agent can take, and the rewards that are received by taking action. The strategy with which the agent decides its action is referred to as a *policy*. The policy takes the environment state as input and outputs the action to take. In reinforcement learning, the policy is often represented by a deep neural network. We refer to this as the reinforcement learning model. Each training job generates one model. A model can be generated even if the training job is stopped early. A model is immutable, which means it cannot be modified and overwritten after it's created.

AWS DeepRacer simulator

The AWS DeepRacer simulator is a virtual environment for visualizing training and evaluating AWS DeepRacer models.

AWS DeepRacer vehicle

See AWS DeepRacer.

AWS DeepRacer car

This type of AWS DeepRacer vehicle is a 1/18th-scale model car.

Leaderboard

A *leaderboard* is a ranked list of AWS DeepRacer vehicle performances in an AWS DeepRacer League racing event. The race can be a virtual event, carried out in the simulated environment, or a physical event, carried out in a real-world environment. The performance metric depends on the race type. It can be the fastest lap time, total time, or average lap time submitted by AWS DeepRacer users who have evaluated their trained models on a track identical or similar to the given track of the race.

If a vehicle completes three laps consecutively, then it qualifies to be ranked on a leaderboard. The average lap time for the first three consecutive laps is submitted to the leaderboard.

Machine learning frameworks

Machine learning frameworks are the software libraries used to build machine learning algorithms. Supported frameworks for AWS DeepRacer include Tensorflow.

Policy network

A policy network is a neural network that is trained. The policy network takes video images as input and predicts the next action for the agent. Depending on the algorithm, it may also evaluate the value of current state of the agent.

Optimization algorithm

An optimization algorithm is the algorithm used to train a model. For supervised training, the algorithm is optimized by minimizing a loss function with a particular strategy to update weights. For reinforcement learning, the algorithm is optimized by maximizing the expected future rewards with a particular reward function.

Neural network

A neural network (also known as an *artificial neural network*) is a collection of connected units or nodes that are used to build an information model based on biological systems. Each node is called an *artificial neuron* and mimics a biological neuron in that it receives an input (stimulus), becomes activated if the input signal is strong enough (activation), and produces an output predicated upon the input and activation. It's widely used in machine learning because an artificial neural network can serve as a general-purpose approximation to any function. Teaching machines to learn becomes finding the optimal function approximation for the given input and output. In deep reinforcement learning, the neural network represents the policy

and is often referred to as the policy network. Training the policy network amounts to iterating through steps that involve generating experiences based on the current policy, followed by optimizing the policy network with the newly generated experiences. The process continues until certain performance metrics meet required criteria.

Hyperparameters

Hyperparameters are algorithm-dependent variables that control the performance of neural network training. An example hyperparameter is the learning rate that controls how many new experiences are counted in learning at each step. A larger learning rate results in a faster training but may make the trained model lower quality. Hyperparameters are empirical and require systematic tuning for each training.

AWS DeepRacer track

A track is a path or course on which an AWS DeepRacer vehicle drives. The track can exist in either a simulated environment or a real-world, physical environment. You use a simulated environment for training an AWS DeepRacer model on a virtual track. The AWS DeepRacer console makes virtual tracks available. You use a real-world environment for running an AWS DeepRacer vehicle on a physical track. The AWS DeepRacer League provides physical tracks for event participants to compete. You must create your own physical track if you want to run your AWS DeepRacer vehicle in any other situation. To learn more about how to build your own track, see Build Your Physical Track.

Reward function

A reward function is an algorithm within a learning model that tells the agent whether the action performed resulted in:

- A good outcome that should be reinforced.
- A neutral outcome.
- A bad outcome that should be discouraged.

The reward function is a key part of reinforcement learning. It determines the behavior that the agent learns by incentivizing specific actions over others. The user provides the reward function by using Python. This reward function is used by an optimizing algorithm to train the reinforcement learning model.

Experience episode

An experience episode is a period in which the agent collects experiences as training data from the environment by running from a given starting point to completing the track or going off

the track. Different episodes can have different lengths. This is also referred to as an *episode* or *experience-generating episode*.

Experience iteration

Experience iteration (also known as *experience-generating iteration*) is a set of consecutive experiences between each policy iteration that performs updates of the policy network weights. At the end of each experience iteration, the collected episodes are added to an experience replay or buffer. The size can be set in one of the hyperparameters for training. The neural network is updated by using random samples of the experiences.

Policy iteration

Policy iteration (also known as *policy-updating iteration*) is any number of passes through the randomly sampled training data to update the policy neural network weights during gradient ascent. A single pass through the training data to update the weights is also known as an *epoch*.

Training job

A training job is a workload that trains a reinforcement learning model and creates trained model artifacts on which to run inference. Each training job has two sub-processes:

1. Start the agent to follow the current policy. The agent explores the environment in a number of *episodes* and creates training data. This data generation is an iterative process itself.
2. Apply the new training data to compute new policy gradients. Update the network weights and continue training. Repeat Step 1 until a stop condition is met.

Each training job produces a trained model and outputs the model artifacts to a specified data store.

Evaluation job

An evaluation job is a workload that tests the performance of a model. Performance is measured by given metrics after the training job is done. The standard AWS DeepRacer performance metric is the driving time that an agent takes to complete a lap on a track. Another metric is the percentage of the lap completed.

Racing event terminology

AWS DeepRacer racing events use the following concepts and terminology.

League/Competition

In the context of AWS DeepRacer League events, the terms *league* and *competition* relate to the competition structure. AWS sponsors the AWS DeepRacer League, which means we own it, design it, and run it. A competition has a start and end date.

Season

A competition can repeat in subsequent years. We call these different seasons (for example, the 2019 season or 2020 season). Rules can change from season to season, but are typically consistent within a season. Terms and conditions for the AWS DeepRacer League can vary from season to season.

The Virtual Circuit

The Virtual Circuit refers to the races sponsored by AWS happening in the AWS DeepRacer console during the AWS DeepRacer League season.

Event

As defined by the rules, an event is an AWS DeepRacer League occurrence in which you can participate in a race. An event has a start and end date. Virtual Circuit events typically last one month. There can be many events in a season, and some rules—such as how we rank those participating in an event, select who wins, and what happens thereafter—are subject to change.

Race type

All racers can race in time-trial (TT), object-avoidance (OA), or head-to-bot (H2B) races. Each race type will specify the number of laps and how racers are ranked.

National Season Standing

A national season standing refers to a racer's leaderboard ranking among other racers in their country. All racers can compete against other racers in their country in monthly virtual races.

Regional Season Standing

A regional season standing refers to a racer's leaderboard ranking among other racers in their region.

World Championship

The AWS DeepRacer League's Virtual Circuit monthly leaderboard is divided by nation and region. The top racer from each region will have the opportunity to qualify for the World Championships at AWS re:Invent. For more information, see the terms and conditions.

How AWS DeepRacer works

AWS DeepRacer vehicle is a 1/18th scale vehicle that can autonomously drive along a track by itself or race against another vehicle. The vehicle can be equipped with various sensors that include a front-facing camera, stereo cameras, radars or a LiDAR. The sensors collect data about the environment the vehicle operates in. Different sensors provide the view at different scales.

AWS DeepRacer uses reinforcement learning to enable autonomous driving for the AWS DeepRacer vehicle. To achieve this, you train and evaluate a reinforcement learning model in a virtual environment with a simulated track. After the training, you upload the trained model artifacts to your AWS DeepRacer vehicle. You can then set the vehicle for autonomous driving in a physical environment with a real track.

Training a reinforcement learning model can be challenging, especially if you're new to the field. AWS DeepRacer simplifies the process by integrating required components together and providing easy-to-follow wizard-like task templates. However, it's helpful to have a good understanding of the basics of reinforcement learning training implemented in AWS DeepRacer.

Topics

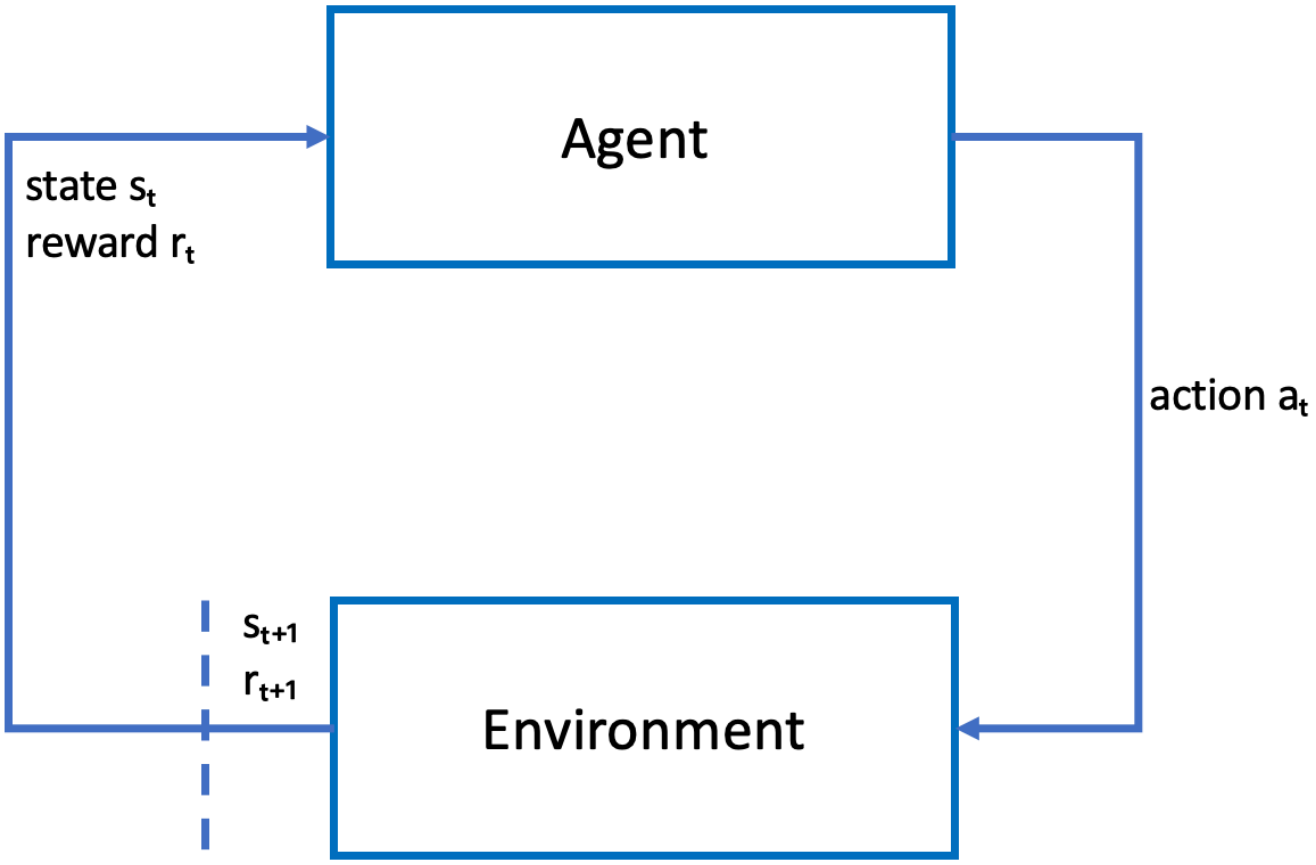
Reinforcement learning in AWS DeepRacer

In reinforcement learning, an *agent*, such as a physical or virtual AWS DeepRacer vehicle, with an objective to achieve an intended goal interacts with an *environment* to maximize the agent's total reward. The agent takes an *action*, guided by a strategy referred to as a *policy*, at a given environment *state* and reaches a new state. There is an immediate *reward* associated with any action. The reward is a measure of the desirability of the action. This immediate reward is considered to be returned by the environment.

The goal of the reinforcement learning in AWS DeepRacer is to learn the optimal policy in a given environment. Learning is an iterative process of trials and errors. The agent takes the random initial

action to arrive at a new state. Then the agent iterates the step from the new state to the next one. Over time, the agent discovers actions that lead to the maximum long-term rewards. The interaction of the agent from an initial state to a terminal state is called an *episode*.

The following sketch illustrates this learning process:



The *agent* embodies a neural network that represents a function to approximate the agent's policy. The image from the vehicle's front camera is the environment *state* and the agent *action* is defined by the agent's speed and steering angles.

The agent receives positive *rewards* if it stays on-track to finish the race and negative rewards for going off-track. An *episode* starts with the agent somewhere on the race track and finishes when the agent either goes off-track or completes a lap.

> **Note**
>
> Strictly speaking, the environment state refers to everything relevant to the problem. For example, the vehicle's position on the track as well as the shape of the track. The

image fed through the camera mounted on the vehicle's front does not capture the entire environment state. For this reason, the environment is deemed partially observed and the input to the agent is referred to as *observation* rather than state. For simplicity, we use *state* and *observation* interchangeably throughout this documentation.

Training the agent in a simulated environment has the following advantages:

- The simulation can estimate how much progress the agent has made and identify when it goes off the track to compute a reward.
- The simulation relieves the trainer from tedious chores to reset the vehicle each time it goes off the track, as is done in a physical environment.
- The simulation can speed up training.
- The simulation provides better controls of the environment conditions, for example selecting different tracks, backgrounds, and vehicle conditions.

The alternative to reinforcement learning is *supervised learning*, also referred to as *imitation learning*. Here a known dataset (of [image, action] tuples) collected from a given environment is used to train the agent. Models that are trained through imitation learning can be applied to autonomous driving. They work well only when the images from the camera look similar to the images in the training dataset. For robust driving, the training dataset must be comprehensive. In contrast, reinforcement learning does not require such extensive labeling efforts and can be trained entirely in simulation. Because reinforcement learning starts with random actions, the agent learns a variety of environment and track conditions. This makes the trained model robust.

AWS DeepRacer action space and reward function

Action space

In reinforcement learning, the set of all valid actions, or choices, available to an agent as it interacts with an environment is called an action space. In the AWS DeepRacer console, you can train agents in either a discrete or continuous action space.

Discrete action space

A discrete action space represents all of an agent's possible actions for each state in a finite set. For AWS DeepRacer, this means that for every incrementally different environmental situation, the agent's neural network selects a speed and direction for the car based on input from its camera(s)

and (optional) LiDAR sensor. The choice is limited to a grouping of predefined steering angle and throttle value combinations.

An AWS DeepRacer car in a discrete action space approaching a turn can choose to accelerate or brake and turn left, right, or go straight. These actions are defined as a combination of steering angle and speed creating a menu of options, 0-9, for the agent. For example, 0 could represent -30 degrees and 0.4 m/s, 1 could represent -30 degrees and 0.8 m/s, 2 could represent -15 degrees and 0.4 m/s, 3 could represent -15 degrees and 0.8 m/s and so on through 9. Negative degrees turn the car right, positive degrees turn the car left and 0 keeps the wheels straight.

The AWS DeepRacer default discrete action space contains the following actions:

AWS DeepRacer default discrete action space

Action number	Steering	Speed
0	-30 degrees	0.4 m/s
1	-30 degrees	0.8 m/s
2	-15 degrees	0.4 m/s
3	-15 degrees	0.8 m/s
4	0 degrees	0.4 m/s
5	0 degrees	0.8 m/s
6	15 degrees	0.4 m/s
7	15 degrees	0.8 m/s
8	30 degrees	0.4 m/s
9	30 degrees	0.8 m/s

Continuous action space

A continuous action space allows the agent to select an action from a range of values for each state. Just as with a discrete action space, this means for every incrementally different

environmental situation, the agent's neural network selects a speed and direction for the car based on input from its camera(s) and (optional) LiDAR sensor. However, in a continuous action space, you can define the range of options the agent picks its action from.

In this example, the AWS DeepRacer car in a continuous action space approaching a turn can choose a speed from 0.75 m/s to 4 m/s and turn left, right, or go straight by choosing a steering angle from -20 to 20 degrees.

Discrete vs. continuous

The benefit of using a continuous action space is that you can write reward functions that train models to incentivize speed/steering actions at specific points on a track that optimize performance. Picking from a range of actions also creates the potential for smooth changes in speed and steering values that, in a well trained model, may produce better results in real-life conditions.

In the discrete action space setting, limiting an agent's choices to a finite number of predefined actions puts the onus on you to understand the impact of these actions and define them based on the environment (track, racing format) and your reward functions. However, in a continuous action space setting, the agent learns to pick the optimal speed and steering values from the min/max bounds you provide through training.

Though providing a range of values for the model to pick from seems to be the better option, the agent has to train longer to learn to choose the optimal actions. Success is also dependent upon the reward function definition.

Reward function

As the agent explores the environment, the agent learns a value function. The value function helps your agent judge how good an action taken is, after observing the environment. The value function uses the reward function that you write in the AWS DeepRacer console to score the action. For example, in the follow the center line sample reward function in the AWS DeepRacer console, a good action would keep the agent near the center of the track and be scored higher than a bad action, which would move the agent away from the center of the track.

Over time, the value function helps the agent learn policies that increase the total reward. The optimal, or best policy, would balance the amount of time the agent spends exploring the environment with the amount of time it spends exploiting, or making the best use of, what the policy has learned through experience.

In the follow the center line AWS DeepRacer sample reward function example, the agent first takes random actions to explore the environment, which means it doesn't do a very good job of staying in the center of the track. Over time, the agent begins to learn which actions keep it near the center line, but if it does this by continuing to take random actions, it will take a long time to learn to stay near the center of the track for the entire lap. So, as the policy begins to learn good actions, the agent begins to use those actions instead of taking random actions. However, if it always uses or exploits the good actions, the agent won't make any new discoveries, because it's no longer exploring the environment. This trade-off is often referred to as the exploration vs exploitation problem in RL.

Experiment with the default action spaces and sample reward functions. Once you've explored them all, put your knowledge to use by designing your own custom action spaces and custom reward functions.

AWS DeepRacer training algorithms

Proximal Policy Optimization (PPO) versus Soft Actor Critic (SAC)

The algorithms SAC and PPO both learn a policy and value function at the same time, but their strategies vary in three notable ways:

PPO	SAC
Works in both discrete and continuous action spaces	Works in a continuous action space
On-policy	Off-policy
Uses entropy regularization	Adds entropy to the maximization objective

Stable vs. data hungry

The information learned by the PPO and SAC algorithms' policies while exploring an environment is utilized differently. PPO uses on-policy learning which means that it learns its value function from observations made by the current policy exploring the environment. SAC uses off-policy learning which means that it can use observations made by previous policies' exploration of the environment. The trade-off between off-policy and on-policy learning is often stability vs. data

efficiency. On-policy algorithms tend to be more stable but data hungry, whereas off-policy algorithms tend to be the opposite.

Exploration vs. exploitation

Exploration vs. exploitation is a key challenge in RL. An algorithm should exploit known information from previous experiences to achieve higher cumulative rewards, but it also needs to explore to gain new experiences that can be used in finding the optimum policy in the future. As a policy is trained over multiple iterations and learns more about an environment, it becomes more certain about choosing an action for a given observation. However, if the policy doesn't explore enough, it will likely stick to information already learned even if it's not at an optimum. The PPO algorithm encourages exploration by using entropy regularization, which prevents agents from converging to local optima. The SAC algorithm strikes an exceptional balance between exploration and exploitation by adding entropy to its maximization objective.

Entropy

In this context, "entropy" is a measure of the uncertainty in the policy, so it can be interpreted as a measure of how confident a policy is at choosing an action for a given state. A policy with low entropy is very confident at choosing an action, whereas a policy with high entropy is unsure of which action to choose.

The SAC algorithm's entropy maximization strategy has similar advantages to the PPO algorithm's use of entropy as a regularizer. Like PPO, it encourages wider exploration and avoids convergence to a bad local optimum by incentivizing the agent to choose an action with higher entropy. Unlike entropy regulation, entropy maximization has a unique advantage. It tends to give up on policies that choose unpromising behavior, which is another reason that the SAC algorithm tends to be more data efficient than PPO.

Tune the amount of entropy in SAC by using the SAC alpha hyperparameter. The maximum SAC alpha entropy value (1.0) favors exploration. The minimum value (0.0) recovers the standard RL objective and neutralizes the entropy bonus that incentivizes exploration. A good SAC alpha value to begin experimenting with is 0.5. Tune accordingly as you iterate on your models.

Try both PPO and SAC algorithms, experiment with their hyperparameters, and explore with them in different action spaces.

AWS DeepRacer solution workflow

Training an AWS DeepRacer model involves the following general tasks:

1. The AWS DeepRacer service initializes the simulation with a virtual track, an agent representing the vehicle, and the background. The agent embodies a policy neural network that can be tuned with hyper-parameters as defined in the PPO algorithm.
2. The agent acts (as specified with a steering angle and a speed) based on a given state (represented by an image from the front camera).
3. The simulated environment updates the agent's position based on the agent action and returns a reward and an updated camera image. The experiences collected in the form of state, action, reward, and new state are used to update the neural network periodically. The updated network models are used to create more experiences.
4. You can monitor the training in progress along the simulated track with a first-person view as seen by the agent. You can display metrics such as rewards per episode, the loss function value, the entropy of the policy. CPU or memory utilization can also be displayed as training progresses. In addition, detailed logs are recorded for analysis and debugging.
5. The AWS DeepRacer service periodically saves the neural network model to persistent storage.
6. The training stops based on a time limit.
7. You can evaluate the trained model in a simulator. To do this, submit the trained model for time trials for a selected number runs on the selected track.

After the model is successfully trained and evaluated, it can be uploaded to a physical agent (an AWS DeepRacer vehicle). The process involves the following steps:

1. Download the trained model from its persistent storage (an Amazon S3 bucket).
2. Use the vehicle's device control console to upload the trained model to the device. Use the console to calibrate the vehicle for mapping the simulated action space to the physical action space. You can also use the console to check the throttling parity, view the front camera feed, load a model into the inference engine, and watch the vehicle driving on a real track.

   The vehicle's device control console is a web server hosted on the vehicle's compute module. The console is accessible from the vehicle IP address with a connected Wi-Fi network and a web browser on a computer or a mobile device.
3. Experiment with the vehicle driving under different lighting, battery levels, and surface textures and colors.

   The device's performance in a physical environment may not match the performance in a simulated environment due to model limitations or insufficient training. The phenomenon

is referred to as the *sim2real* performance gap. To reduce the gap, see the section called "Simulated-to-real performance gaps".

Simulated-to-real performance gaps

Because the simulation cannot capture all aspects of the real world accurately, the models trained in simulation may not work well in the real world. Such discrepancies are often referred to as simulated-to-real (*sim2real*) performance gaps.

Efforts have been made in AWS DeepRacer to minimize the *sim2real* performance gap. For example, the simulated agent is programmed to take about 10 actions per second. This matches the frequency the AWS DeepRacer device runs inference with, about 10 inferences per second. As another example, at the start of each episode in training, the agent's position is randomized. This maximizes the likelihood that the agent learns all parts of the track evenly.

To help reduce *real2sim* performance gaps, make sure to use the same or similar color, shape and dimensions for both the simulated and real tracks. To reduce visual distractions, use barricades around the real track. Also, carefully calibrate the ranges of the device's speed and steering angles so that the action space used in training matches the real world. Evaluating model performance in a different simulation track than the one used in training can show the extent of the *real2real* performance gap.

For more information about how to reduce the *sim2real* gap when training an AWS DeepRacer model, see the section called "Optimize training for real environments".

Get started with AWS DeepRacer

To get started with AWS DeepRacer, let's first walk through the steps to use the AWS DeepRacer console to configure an agent with appropriate sensors for your autonomous driving requirements, to train a reinforcement learning model for the agent with the specified sensors, and to evaluate the trained model to determine the quality of the model. Once you've trained your model, you can iterate on it and submit it to a race.

Topics

Train your first AWS DeepRacer model

This walkthrough demonstrates how to train your first model using the AWS DeepRacer console.

Train a reinforcement learning model using the AWS DeepRacer console

Learn where to find the **Create model** button in the AWS DeepRacer console to start your model training journey.

To train a reinforcement learning model

1. If this is your first time using AWS DeepRacer, choose **Create model** from the service landing page or select **Get started** under the **Reinforcement learning** heading on the main navigation pane.
2. On the **Get started with reinforcement learning** page, under **Step 2: Create a model**, choose **Create model**.

   Alternatively, choose **Your models** under the **Reinforcement learning** heading from the main navigation pane. On the **Your models** page, choose **Create model**.

Specify the model name and environment

Name your model and learn how to pick the simulation track that's right for you.

To specify the model name and environment

1. On the **Create model** page, under **Training details**, enter a name for your model.
2. Optionally, add a training job description.
3. To learn more about adding optional tags, see *Tagging*.
4. Under **Environment simulation**, choose a track to serve as a training environment for your AWS DeepRacer agent. Under **Track direction**, choose **Clockwise** or **Counterclockwise**. Then, choose **Next**.

   For your first run, choose a track with a simple shape and smooth turns. In later iterations, you can choose more complex tracks to progressively improve your models. To train a model for a particular racing event, choose the track most similar to the event track.
5. Choose **Next** at the bottom of the page.

Choose a race type and training algorithm

The AWS DeepRacer console has three race types and two training algorithms from which to choose. Learn which are appropriate for your skill level and training goals.

To choose a race type and training algorithm

1. On the **Create model** page, under **Race type**, select **Time trial**, **Object avoidance**, or **Head-to-bot**.

   For your first run, we recommend choosing **Time trial**. For guidance on optimizing your agent's sensor configuration for this race type, see the section called “Tailor training for time trials”.
2. Optionally, on later runs, choose **Object avoidance** to go around stationary obstacles placed at fixed or random locations along the chosen track. For more information, see the section called “Tailor training for object avoidance races”.

   a. Choose **Fixed location** to generate boxes in fixed, user designated locations across the two lanes of the track or select **Random location** to generate objects that are randomly distributed across the two lanes at the beginning of each episode of your training simulation.

   b. Next, choose a value for the **Number of objects on a track**.

   c. If you chose **Fixed location** you can adjust each object's placement on the track. For **Lane placement**, choose between the inside lane and the outside lane. By default, objects

are evenly distributed across the track. To change how far between the start and the finish line an object is, enter a percentage of that distance between seven and 90 on the **Location (%) between start and finish** field.

3. Optionally, for more ambitious runs, choose **Head-to-bot racing** to race against up to four bot vehicles moving at a constant speed. To learn more, see the section called “Tailor training for head-to-bot races”.

   a. Under **Choose the number of bot vehicles**, select with how many bot vehicles you want your agent to train.

   b. Next, choose the speed in millimeters per second at which you want the bot vehicles to travel around the track.

   c. Optionally, check the **Enable lane changes** box to give the bot vehicles the ability to randomly change lanes every 1-5 seconds.

4. Under **Training algorithm and hyperparameters**, choose the **Soft Actor Critic (SAC)** or **Proximal Policy Optimization (PPO)** algorithm. In the AWS DeepRacer console, SAC models must be trained in continuous action spaces. PPO models can be trained in either continuous or discrete action spaces.

5. Under **Training algorithm and hyperparameters**, use the default hyperparameter values as-is.

   Later on, to improve training performance, expand **Hyperparameters** and modify the default hyperparameter values as follows:

   a. For **Gradient descent batch size**, choose available options.

   b. For **Number of epochs**, set a valid value.

   c. For **Learning rate**, set a valid value.

   d. For **SAC alpha value** (SAC algorithm only), set a valid value.

   e. For **Entropy**, set a valid value.

   f. For **Discount factor**, set a valid value.

   g. For **Loss type**, choose available options.

   h. For **Number of experience episodes between each policy-updating iteration**, set a valid value.

   For more information about hyperparameters, see Systematically tune hyperparameters.

6. Choose **Next**.

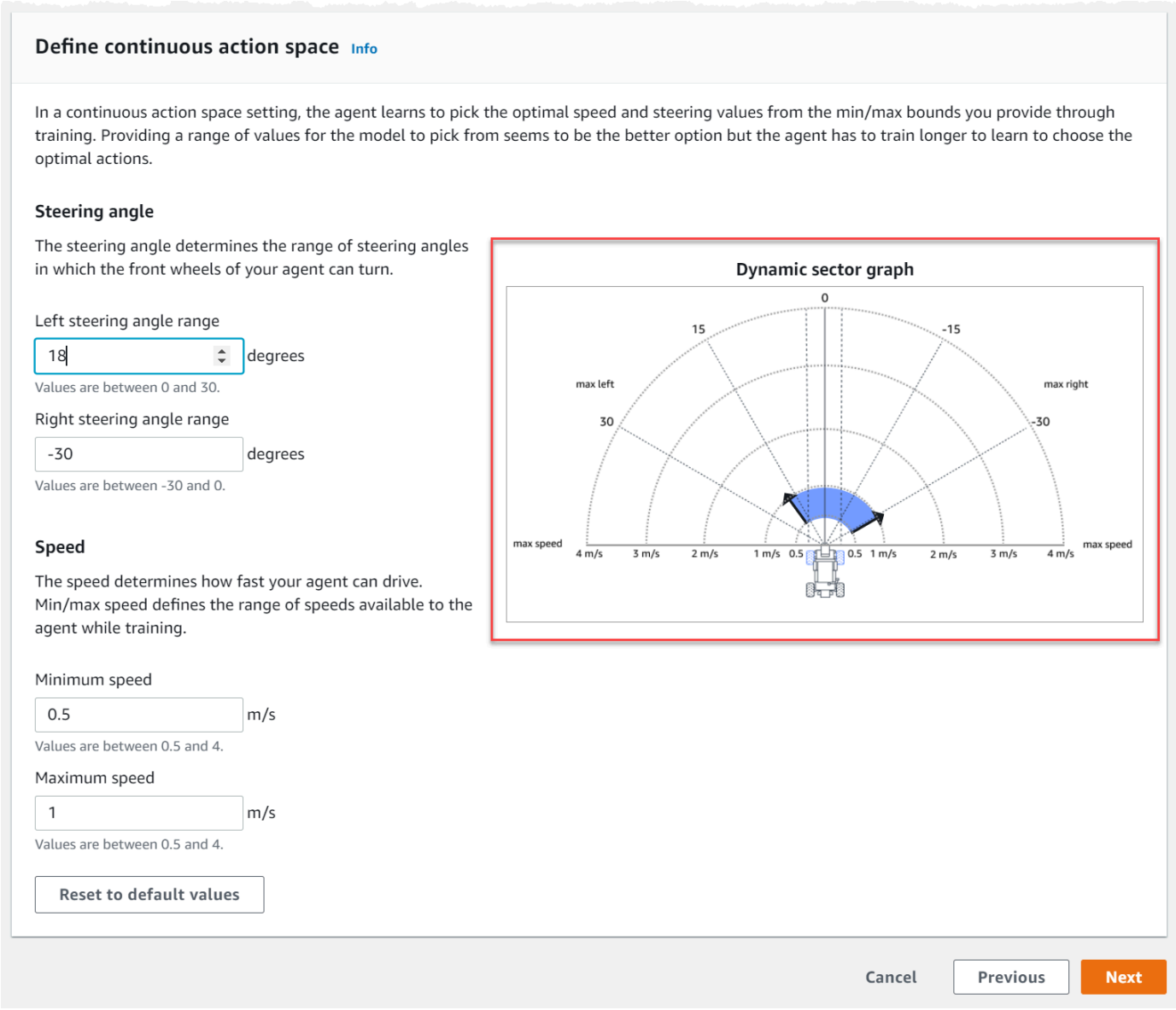
Define action space

On the **Define action space** page, if you've chosen to train with the Soft Actor Critic (SAC) algorithm, your default action space is the continuous action space. If you've chosen to train with the Proximal Policy Optimization (PPO) algorithm, choose between **Continuous action space** and **Discrete action space**. To learn more about how each action space and algorithm shapes the agent's training experience, see the section called “Action space and reward function”.

To define continuous action space (SAC or PPO algorithms)

1. Under **Define continuous action space**, choose the degrees of your **Left steering angle range** and **Right steering angle range**.

   Try entering different degrees for each steering angle range and watch the visualization of your range change to represent your choices on the **Dynamic sector graph**.



2. Under **Speed**, enter a minimum and maximum speed for your agent in millimeters per second.

   Notice how your changes are reflected on the **Dynamic sector graph**.

3. Optionally, choose **Reset to default values** to clear unwanted values. We encourage trying out different values on the graph to experiment and learn.

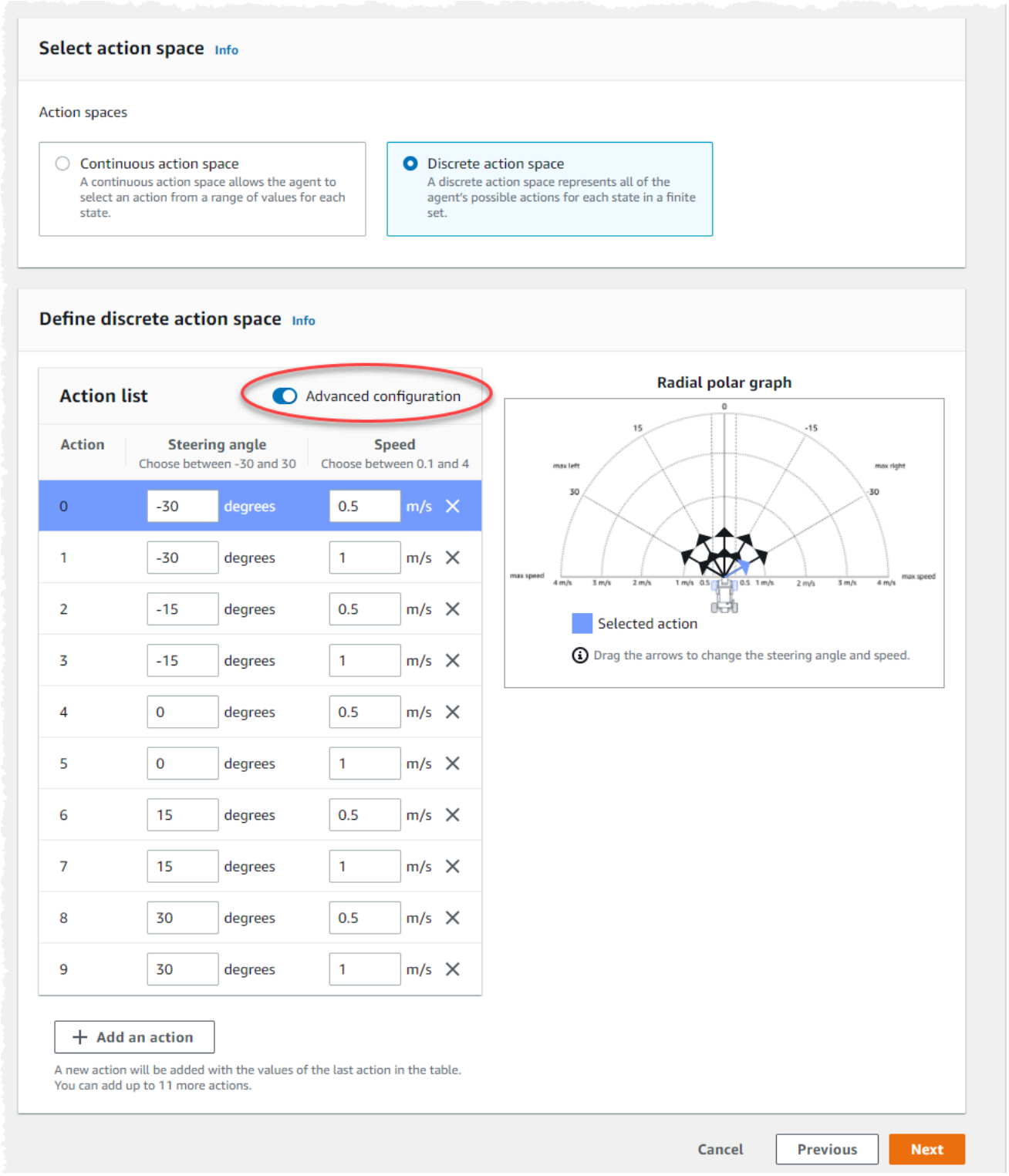
4. Choose **Next**.

To define discrete action space (PPO algorithm only)

1. Choose a value for **Steering angle granularity** from the dropdown list.

2. Choose a value in degrees between 1-30 for your agent's **Maximum steering angle**.

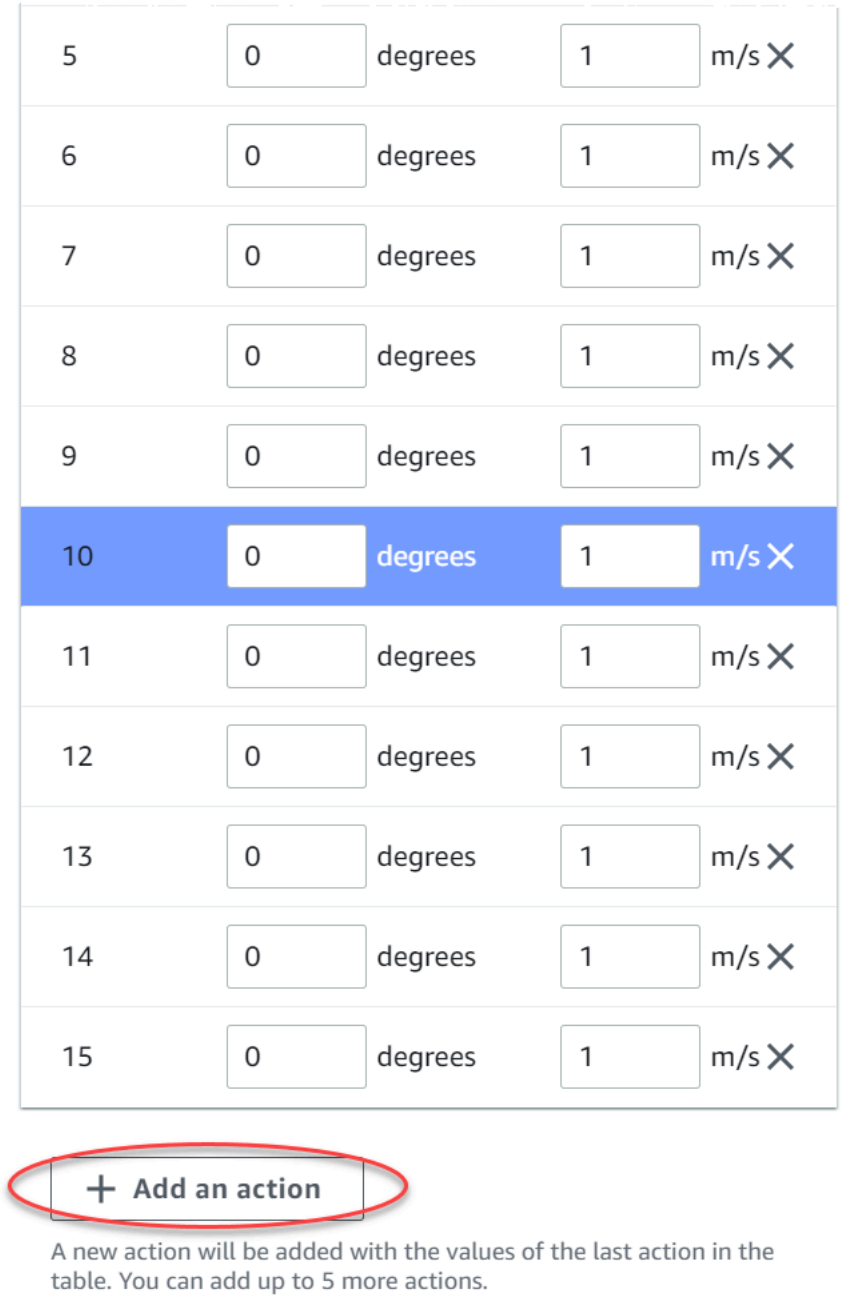
3. Choose a value for **Speed granularity** from the dropdown list.
4. Choose a value in millimeters per second between 0.1-4 for your agent's **Maximum speed**.
5. Use the default action settings on the **Action list** or, optionally, toggle on **Advanced configuration** to fine tune your settings. If you choose **Previous** or toggle off **Advanced configuration** after adjusting values, you lose your changes.



a. Enter a value in degrees between -30 and 30 in the **Steering angle** column.

b. Enter a value between 0.1 and 4 in millimeters per second for up to nine actions in the **Speed** column.

c. Optionally, select **Add an action** to increase the number of rows in the action list.



d. Optionally, select **X** on a row to remove it.

6. Choose **Next**.

Choose a virtual car

Learn how to get started with virtual cars. Earn new custom cars, paint jobs, and modifications by competing in the Open Division each month.

To choose a virtual car

1. On the **Choose vehicle shell and sensor configuration** page, choose a shell that is compatible with your race type and action space. If you don't have a car in your garage that matches, go to **Your garage** under the **Reinforcement learning** heading on the main navigation pane to create one.

   For **Time trial** training, the default sensor configuration and single-lens camera of **The Original DeepRacer** is all you need, but all other shells and sensor configurations work as long as the action space matches. For more information, see the section called “Tailor training for time trials”.

   For **Object avoidance** training, stereo cameras are helpful, but a single camera can also be used for avoiding stationary obstacles in fixed locations. A LiDAR sensor is optional. See the section called “Action space and reward function”.

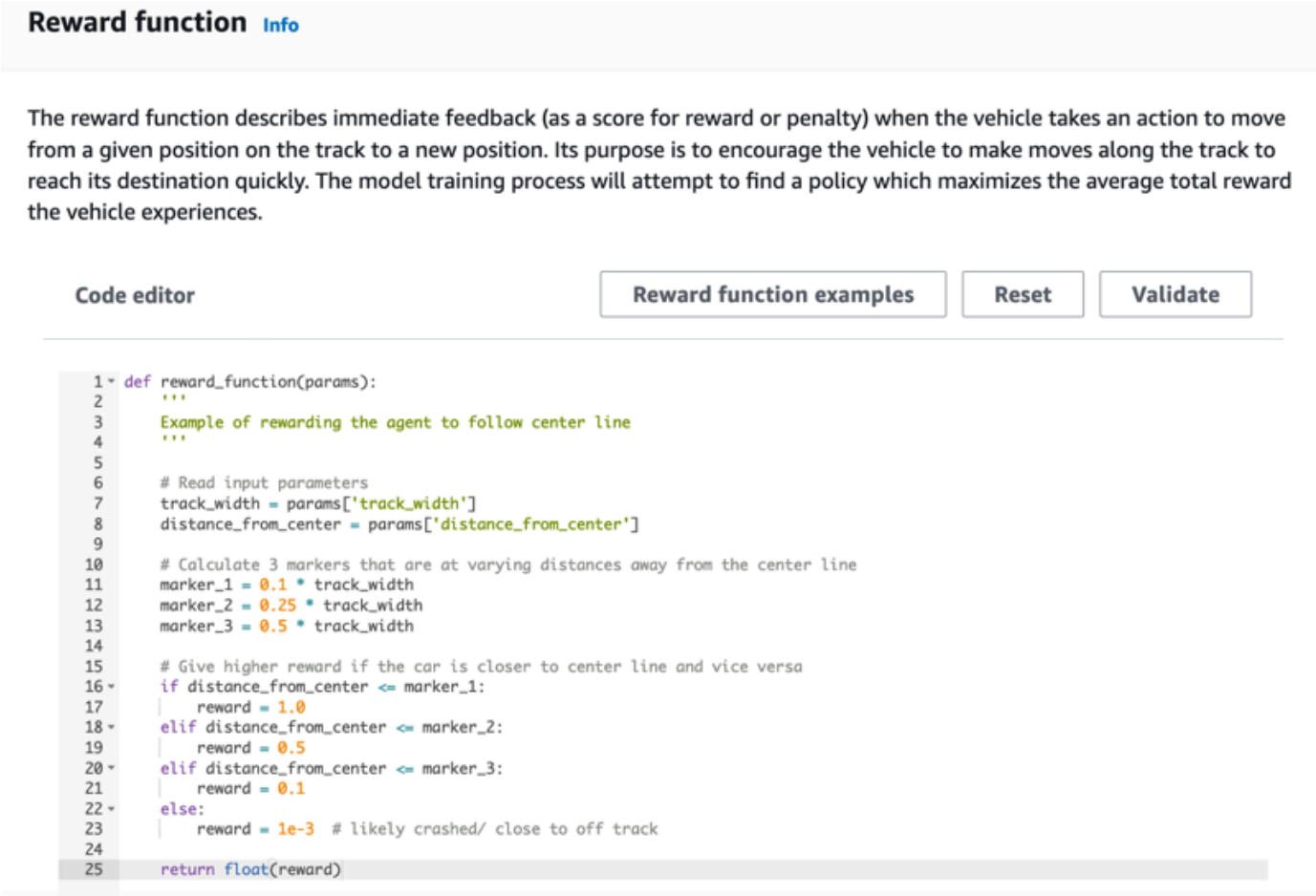
   For **Head-to-bot** training, in addition to either a single camera or a stereo camera, a LiDAR unit is optimal for detecting and avoiding blind spots while passing other moving vehicles. To learn more, see the section called “Tailor training for head-to-bot races”.
2. Choose **Next**.

Customize your reward function

The reward function is at the core of reinforcement learning. Learn to use it to incentivize your car (agent) to take specific actions as it explores the track (environment). Like encouraging and discouraging certain behaviors in a pet, you can use this tool to encourage your car to finish a lap as fast as possible and discourage it from driving off of the track or colliding with objects.

To customize your reward function

1. On the **Create model** page, under **Reward function**, use the default reward function example as-is for your first model.



Later on, you can choose **Reward function examples** to select another example function and then choose **Use code** to accept the selected reward function.

There are four example functions with which you can start. They illustrate how to follow the track center (default), how to keep the agent inside the track borders, how to prevent zig-zag driving, and how to avoid crashing into stationary obstacles or other moving vehicles.

To learn more about the reward function, see the section called “Reward function reference”.

2. Under **Stop conditions**, leave the default **Maximum time** value as-is, or set a new value to terminate the training job, to help prevent long-running (and possible run-away) training jobs.

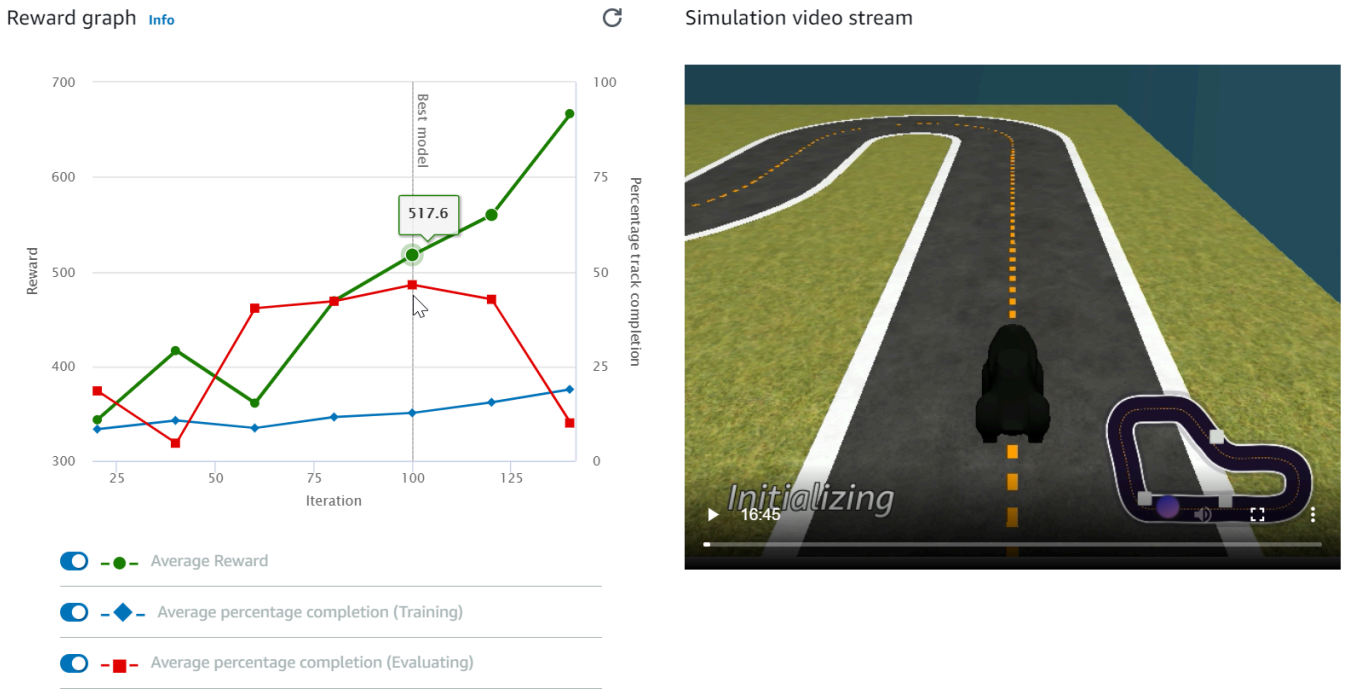
   When experimenting in the early phase of training, you should start with a small value for this parameter and then progressively train for longer amounts of time.

3. Under **Automatically submit to the AWS DeepRacer**, **Submit this model to the AWS DeepRacer automatically after training completion and get a chance to win prizes** is

checked by default. Optionally, you may opt out of entering your model by selecting the checkmark.

4. Under **League requirements**, select your **Country of residence** and accept the the terms and conditions by checking the box.
5. Choose **Create model** to start creating the model and provisioning the training job instance.
6. After the submission, watch your training job being initialized and then run.

   The initialization process takes a few minutes to change from **Initializing** to **In progress**.
7. Watch the **Reward graph** and **Simulation video stream** to observe the progress of your training job. You can choose the refresh button next to **Reward graph** periodically to refresh the **Reward graph** until the training job is complete.



The training job runs on the AWS Cloud, so you don't need to keep the AWS DeepRacer console open. You can always come back to the console to check on your model at any point while the job is in progress.

If the **Simulation video stream** window or the **Reward graph** display become unresponsive, refresh the browser page to get the training progress updated.

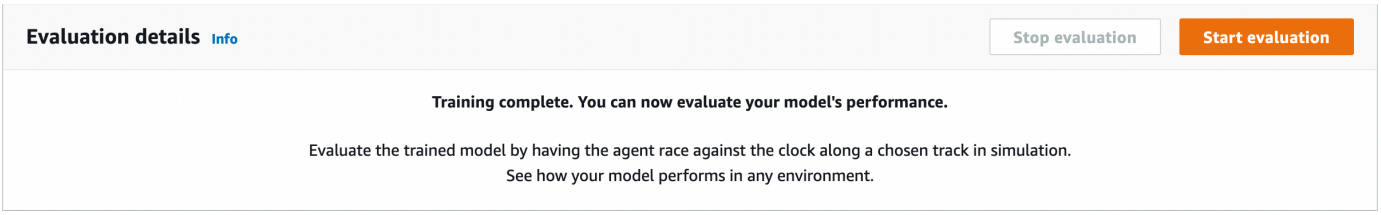
Evaluate your AWS DeepRacer models in simulation

After your training job is complete, you should evaluate the trained model to assess its convergency behavior. The evaluation proceeds by completing a number of trials on a chosen track and having the agent move on the track according to likely actions inferred by the trained model. The performance metrics include a percentage of track completion and the time running on each track from start to finish or going off-track.

To evaluate your trained model, you can use the AWS DeepRacer console. To do so, follow the steps in this topic.

To evaluate a trained model in the AWS DeepRacer console

1. Open the AWS DeepRacer console at https://console.aws.amazon.com/deepracer.
2. From the main navigation pane, choose **Models** and then choose the model you just trained from the **Models** list to open the model details page.
3. Select the **Evaluation** tab.
4. In **Evaluation details**, choose **Start evaluation**.

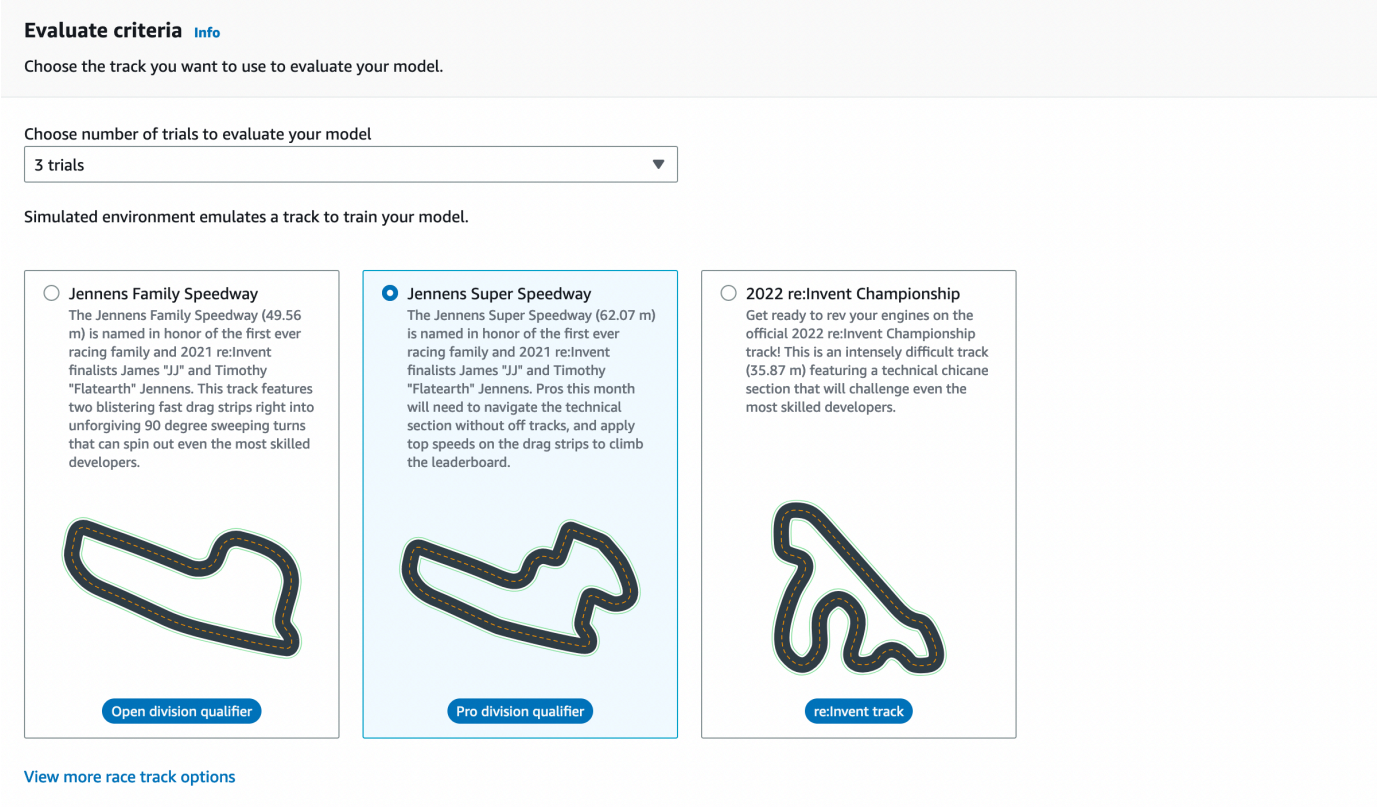


   You can start an evaluation after your training job status changes to **Completed** or the model's status changes to **Ready** if the training job wasn't completed.

   A model is ready when the training job is complete. If the training wasn't completed, the model can also be in a **Ready** state if it's trained up to the failing point.
5. On the **Evaluate model** page, under **Race type**, enter a name for your evaluation, then choose the racing type that you chose to train the model.

   For evaluation you can choose a race type different from the race type used in training. For example, you can train a model for head-to-bot races and then evaluate it for time trials. In general, the model must generalize well if the training race type differs from the evaluation race type. For your first run, you should use the same race type for both evaluation and training.

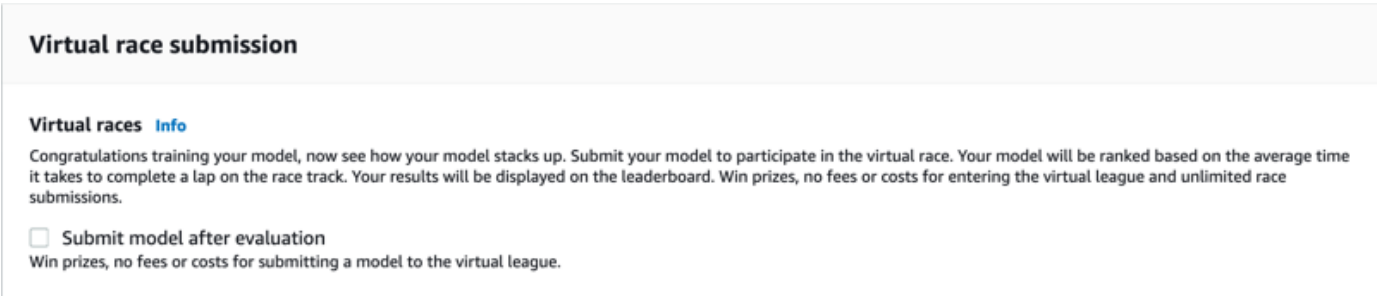
6. On the **Evaluate model** page, under **Evaluate criteria**, choose the number of trials you want to run, then choose a track to evaluate the model on.



Typically, you want to choose a track that is the same as or similar to the one you used in training the model. You can choose any track for evaluating your model, however, you can expect the best performance on a track most similar to the one used in training.

To see if your model generalizes well, choose an evaluation track different from the one used in training.

7. On the **Evaluate model** page, under **Virtual Race Submission**, for your first model, turn off the **Submit model after evaluation** option. Later, if you want to participate in a racing event, leave this option turned on.



8. On the **Evaluate model** page, choose **Start evaluation** to start creating and initializing the evaluation job.

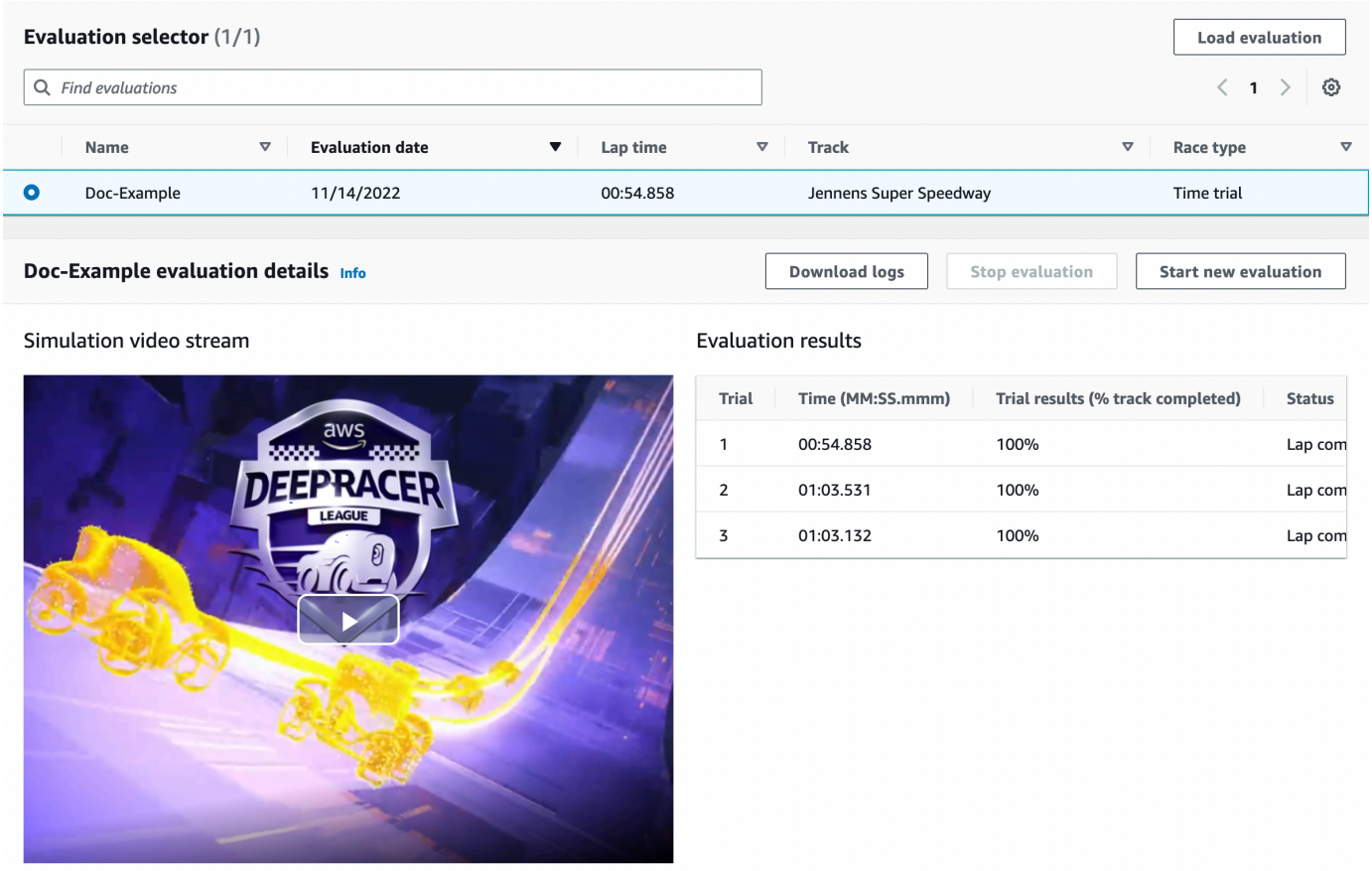
   This initialization process takes about 3 minutes to complete.

9. As the evaluation progresses, the evaluation results, including the trial time and track completion rate, are displayed under **evaluation details** after each trial. In the **Simulation video stream** window, you can watch how the agent performs on the chosen track.

   You can stop an evaluation job before it completes. To stop the evaluation job, choose **Stop evaluation** on the upper-right corner of the **Evaluation** card and then confirm to stop the evaluation.

10. After the evaluation job is complete, examine the performance metrics of all the trials under **Evaluation results**. The accompanying simulation video stream is no longer available.

    A history of your model's evaluations is available in the **Evaluation selector**. To view the details of a specific evaluation, select the evaluation from the **Evaluation selector** list, then choose **Load evaluation** from the top-right corner of the **Evaluation selector** card.



For this particular evaluation job, the trained model completes the trials with a significant off-track time penalty. As a first run, this is not unusual. Possible reasons include that the training didn't converge and the training needs more time, the action space needs to be enlarged to give the agent more room to react, or the reward function needs to be updated to handle varying environments.

You can continue to improve the model by cloning a previously trained one, modifying the reward function, tuning hyperparameters, and then iterating the process until the total reward converges and the performance metrics improve. For more information on how to improve the training, see *Train and evaluate models*.

To transfer your completely trained model to your AWS DeepRacer device for driving in a physical environment, you need to download the model artifacts. To do so, choose **Download model** on the model's details page. If your AWS DeepRacer physical device doesn't support new sensors and your model has been trained with the new sensor types, you'll get an error message when you use the model on your AWS DeepRacer device in a real-world environment. For more information about testing an AWS DeepRacer model with a physical device, see *Operate your vehicle*.

Once you've trained your model on a track identical or similar to the one specified in an AWS DeepRacer League racing event or an AWS DeepRacer community race, you can submit the model to the virtual races in the AWS DeepRacer console. To do this, follow **AWS virtual circuit** or **Community races** on the main navigation pane. For more information, see *Join a race*.

To train a model for obstacle avoidance or head-to-bot racing, you may need to add new sensors to the model and the physical device. For more information, see the section called “Understanding racing types and enabling sensors”.

Train and evaluate AWS DeepRacer models

When your AWS DeepRacer vehicle drives itself along a track, it captures environmental states with the camera mounted on the front and takes actions in response to the observations. Your AWS DeepRacer model is a function that maps the observations and actions to the expected reward. To train your model is to find or learn the function that maximize the expected reward so that the optimized model prescribes what actions (speed and steering angle pairs) your vehicle can take to move itself along the track from start to finish.

In practice, the function is represented by a neural network and training the network involves finding the optimal network weights given sequences of observed environmental states and the responding vehicle's actions. The underlying criteria of optimality are described by the model's reward function that encourages the vehicle to make legal and productive moves without causing traffic accidents or infractions. A simple reward function could return a reward of 0 if the vehicle is on the track, -1 if it's off the track, and +1 if it reaches the finish line. With this reward function, the vehicle gets penalized for going off the track and rewarded for reaching the destination. This can be a good reward function if time or speed is not an issue.

Suppose that you're interested in having the vehicle drive as fast as it can without getting off a straight track. As the vehicle speeds up and down, the vehicle may steer left or right to avoid obstacles or to remain inside. Making too big a turn at a high speed could easily lead the vehicle off the track. Making too small a turn may not help avoid colliding with an obstacle or another vehicle. Generally speaking, optimal actions would be to make a bigger turn at a lower speed or to steer less along a sharper curve. To encourage this behavior, your reward function must assign a positive score to reward smaller turns at a higher speed and/or a negative score to punish bigger turns at a higher speed. Similarly, the reward function can return a positive reward for speeding up along a straighter course or speeding down when it's near an obstacle.

The reward function is an important part of your AWS DeepRacer model. You must provide it when training your AWS DeepRacer model. The training involves repeated episodes along the track from start to end. In an episode the agent interacts with the track to learn the optimal course of actions by maximizing the expected cumulative reward. At the end, the training produces a reinforcement learning model. After the training, the agent executes autonomous driving by running inference on the model to take an optimal action in any given state. This can be done in either the simulated environment with a virtual agent or a real-world environment with a physical agent, such as an AWS DeepRacer scale vehicle.

To train a reinforcement learning model in practice, you must choose a learning algorithm. Currently, the AWS DeepRacer console supports only the proximal policy optimization (PPO) and soft actor critic (SAC) algorithms. You can then choose a deep-learning framework supporting the chosen algorithm, unless you want to write one from scratch. AWS DeepRacer integrates with SageMaker AI to make some popular deep-learning frameworks, such as TensorFlow, readily available in the AWS DeepRacer console. Using a framework simplifies configuring and executing training jobs and lets you focus on creating and enhancing reward functions specific to your problems.

Training a reinforcement learning model is an iterative process. First, it's challenging to define a reward function to cover all important behaviors of an agent in an environment at once. Second, hyperparameters are often tuned to ensure satisfactory training performance. Both require experimentation. A prudent approach is to start with a simple reward function and then progressively enhance it. AWS DeepRacer facilitates this iterative process by enabling you to clone a trained model and then use it to jump-start the next round of training. At each iteration you can introduce one or a few more sophisticated treatments to the reward function to handle previously ignored variables or you can systematically adjust hyperparameters until the result converges.

As with general practice in machine learning, you must evaluate a trained reinforcement learning model to ascertain its efficacy before deploying it to a physical agent for running inference in a real-world situation. For autonomous driving, the evaluation can be based on how often a vehicle stays on a given track from start to finish or how fast it can finish the course without getting off the track. The AWS DeepRacer simulation lets you run the evaluation and post the performance metrics for comparison with models trained by other AWS DeepRacer users on a leaderboard.

Topics

Understanding racing types and enabling sensors supported by AWS DeepRacer

In AWS DeepRacer League, you can participate in the following types of racing events:

- **Time trial**: race against the clock on an unobstructed track and aim to get the fastest lap time possible.

- **Object avoidance**: race against the clock on a track with stationary obstacles and aim to get the fastest lap time possible.
- **Head-to-bot racing**: race against one or more other vehicles on the same track and aim to cross the finish line before other vehicles.

AWS DeepRacer community races currently supports time trials only.

You should experiment with different sensors on your AWS DeepRacer vehicle to provide it with sufficient capabilities to observe its surroundings for a given race type. The next section describes the AWS DeepRacer-supported sensors that can enable the supported types of autonomous racing events.

Topics

Choose sensors for AWS DeepRacer racing types

Your AWS DeepRacer vehicle comes with a front-facing monocular camera as the default sensor. You can add another front-facing monocular camera to make front-facing stereo cameras or to supplement either the monocular camera or stereo cameras with a LiDAR unit.

The following list summarizes the functional capabilities of AWS DeepRacer-supported sensors, together with brief cost-and-benefit analyses:

Front-facing camera

A single-lens front-facing camera can capture images of the environment in front of the host vehicle, including track borders and shapes. It's the least expensive sensor and is suitable to handle simpler autonomous driving tasks, such as obstacle-free time trials on well-marked tracks. With proper training, it can avoid stationary obstacles on fixed locations on the track. However, the obstacle location information is built into the trained model and, as the result, the model is likely to be overfitted and may not generalize to other obstacle placements. With

stationary objects placed at random locations or other moving vehicles on the track, the model is unlikely to converge.

In the real world, the AWS DeepRacer vehicle comes with a single-lens front-facing camera as the default sensor. The camera has 120-degree wide angle lens and captures RGB images that are then converted to grey-scale images of 160 x 120 pixels at 15 frames per second (fps). These sensor properties are preserved in the simulator to maximize the chance that the trained model transfers well from simulation to the real world.

Front-facing stereo camera

A stereo camera has two or more lenses that capture images with the same resolution and frequency. Images from the both lens are used to determine the depth of observed objects. The depth information from a stereo camera is valuable for the host vehicle to avoid crashing into the obstacles or other vehicles in the front, especially under more dynamic environment. However, added depth information makes trainings to converge more slowly.

On the AWS DeepRacer physical vehicle, the double-lens stereo camera is constructed by adding another single-lens camera and mounting each camera on the left and right sides of the vehicle. The AWS DeepRacer software synchronizes image captures from both cameras. The captured images are converted into greyscale, stacked, and fed into the neural network for inferencing. The same mechanism is duplicated in the simulator in order to train the model to generalize well to a real-world environment.

LiDAR sensor

A LiDAR sensor uses rotating lasers to send out pulses of light outside the visible spectrum and time how long it takes each pulse to return. The direction of and distance to the objects that a specific pulse hits are recorded as a point in a large 3D map centered around the LiDAR unit.

For example, LiDAR helps detect blind spots of the host vehicle to avoid collisions while the vehicle changes lanes. By combining LiDAR with mono or stereo cameras, you enable the host vehicle to capture sufficient information to take appropriate actions. However, a LiDAR sensor costs more compared to cameras. The neural network must learn how to interpret the LiDAR data. Thus, trainings will take longer to converge.

On the AWS DeepRacer physical vehicle a LiDAR sensor is mounted on the rear and tilted down by 6 degrees. It rotates at the angular velocity of 10 rotations per second and has a range of 15cm to 2m. It can detect objects behind and beside the host vehicle as well as tall objects unobstructed by the vehicle parts in the front. The angle and range are chosen to make the LiDAR unit less susceptible to environmental noise.

You can configure your AWS DeepRacer vehicle with the following combination of the supported sensors:

- Front-facing single-lens camera only.

  This configuration is good for time trials, as well as obstacle avoidance with objects at fixed locations.
- Front-facing stereo camera only.

  This configuration is good for obstacle avoidance with objects at fixed or random locations.
- Front-facing single-lens camera with LiDAR.

  This configuration is good for obstacle avoidance or head-to-bot racing.
- Front-facing stereo camera with LiDAR.

  This configuration is good for obstacle avoidance or head-to-bot racing, but probably not most economical for time trials.

As you add more sensors to make your AWS DeepRacer vehicle to go from time trials to object avoidance to head-to-bot racing, the vehicle collects more data about the environment to feed into the underlying neural network in training. This makes training more challenging because the model is required to handle increased complexities. In the end, your tasks of learning to train models become more demanding.

To learn progressively, you should start training for time trials first before moving on to object avoidance and then to head-to-bot racing. You'll find more detailed recommendations in the next section.

Configure agent for training AWS DeepRacer models

To train a reinforcement learning model for the AWS DeepRacer vehicle to race in obstacle avoidance or head-to-bot racing, you need to configure the agent with appropriate sensors. For simple time trials, you could use the default agent configured with a single-lens camera. In configuring the agent you can customize the action space and choose a neural network topology so that they work better with the selected sensors to meet the intended driving requirements. In addition, you can change the agent's appearance for visual identification during training.

After you configure it, the agent configuration is recorded as part of the model's metadata for training and evaluation. For evaluation, the agent automatically retrieves the recorded configuration to use the specified sensors, action space, and neural network technology.

This section walks you through the steps to configure an agent in the AWS DeepRacer console.

To configure an AWS DeepRacer agent in the AWS DeepRacer console

1. Sign in to the AWS DeepRacer console.
2. On the primary navigation pane, choose **Garage**.
3. For the first time you use **Garage**, you're presented with the **WELCOME TO THE GARAGE** dialog box. Choose **>** or **<** browse through the introduction to various sensors supported for the AWS DeepRacer vehicle or choose **X** to close the dialog box. You can find this introductory information on the help panel in **Garage**.
4. On the **Garage** page, choose **Build new vehicle**.
5. On the **Mod your own vehicle** page, under **Mod specifications**, choose one or more sensors to try and learn the best combination that can meet your intended racing types.

   To train for your AWS DeepRacer vehicle time trials, choose **Camera**. For obstacle avoidance or head-to-bot racing, you want to use other sensor types. To choose **Stereo camera**, make sure you have acquired an additional single-lens camera. AWS DeepRacer makes the stereo camera out two single-lens cameras. You can have either a single-lens camera or a double-lens stereo cameras on one vehicle. In either case, you can add a LiDAR sensor to the agent if you just want the trained model to be able to detect and avoid blind spots in obstacle avoidance or head-to-bot racing.
6. On the **Garage** page and under **Neural network topologies**, choose a supported network topology.

   In general, a deeper neural network (with more layers) is more suitable for driving on more complicated tracks with sharp curves and numerous turns, for racing to avoid stationary obstacles, or for competing against other moving vehicles. But a deeper neural network is more costly to train and the model takes longer to converge. On the other hand, a shallower network (with fewer layers) costs less and takes a shorter time to train. The trained model is capable of handling simpler track conditions or driving requirements, such as time trials on a obstacle-free track without competitors.

   Specifically, AWS DeepRacer supports **3-layer CNN** or **5-layer CNN**.

7. On the **Garage** page, choose **Next** to proceed to setting up the agent's action space.
8. On the **Action space** page, leave the default settings for your first training. For subsequent trainings, experiment with different settings for the steering angle, top speed, and their granularities. Then, choose **Next**.
9. On the **Color your vehicle to stand out in the crowd** page, enter a name in **Name your DeepRacer** and then choose a color for the agent from the **Vehicle color** list. Then, choose **Submit**.
10. On the **Garage** page, examine the settings of the new agent. To make further modifications, choose **Mod vehicle** and repeat the previous steps starting at **Step 4**.

Now, your agent is ready for training.

Tailor AWS DeepRacer training for time trials

If this is your first time to use AWS DeepRacer, you should start with a simple time trial to become familiar with how to train AWS DeepRacer models to drive your vehicle. This way, you get a gentler introduction to basic concepts of reward function, agent, environment, etc. Your goal is to train a model to make the vehicle stay on the track and finish a lap as fast as possible. You can then deploy the trained model to your AWS DeepRacer vehicle to test driving on a physical track without any additional sensors.

To train a model for this scenario, you can choose the default agent from **Garage** on the AWS DeepRacer console. The default agent has been configured with a single front-facing camera, a default action space and a default neural network topology. It is helpful to start training an AWS DeepRacer model with the default agent before moving on to more sophisticated ones.

To train your model with the default agent, follow the recommendations below.

1. Start training your model with a simple track of more regular shapes and of less sharp turns. Use the default reward function. And train the model for 30 minutes. After the training job is completed, evaluate your model on the same track to watch if the agent can finish a lap.
2. Read about the reward function parameters. Continue the training with different incentives to reward the agent to go faster. Lengthen the training time for the next model to 1 - 2 hours. Compare the reward graph between the first training and this second one. Keep experimenting until the reward graph stops improving.
3. Read more about action space. Train the model the 3rd time by increasing the top speed (for example 1 m/s). To modify the action space, you must build in **Garage** a new agent, when

you get the chance to make the modification. When updating the top speed of your agent, be aware of that the higher the top speed, the faster the agent can complete the track in evaluation and the faster your AWS DeepRacer vehicle can finish a lap on a physical track. However, a higher top speed often means a longer time for the training to converge because the agent is more likely to overshoot on a curve and thus get off track. You may want to decrease granularities to give the agent more rooms to accelerate or decelerate and further tweak the reward function in other ways to help training converge faster. After the training converges, evaluate the 3rd model to see if the lap time improves. Keep exploring until there is no more improvement.

4. Choose a more complicated track and repeat **Step 1** to **Step 3**. Evaluate your model on a track that is different from the one you used to train on to see how the model can generalize to different virtual tracks generalize to real-world environments.
5. (Optional) Experiment with different values of the hyperparameters to improve the training process and repeat **Step 1** to **Step 3**.
6. (Optional) Examine and analyze the AWS DeepRacer logs. For sample code that you can use to analyze the logs, see https://github.com/aws-samples/aws-deepracer-workshops/tree/master/log-analysis.

Tailor AWS DeepRacer training for object avoidance races

After you become familiar with time trials and have trained a few converged models, move on to the next more demanding challenge—obstacle avoidance. Here, your goal is to train a model that can complete a lap as fast as possible without going off track, while avoiding crashing into the objects placed on the track. This is obviously a harder problem for the agent to learn, and training takes longer to converge.

The AWS DeepRacer console supports two types of obstacle avoidance training: obstacles can be placed at fixed or random locations along the track. With fixed locations, the obstacles remain fixed to the same place throughout the training job. With random locations, the obstacles change their respective places at random from episode to episode.

It is easier for trainings to converge for location-fixed obstacle avoidance because the system has less degrees of freedom. However, models can overfit when the location information is built in to the trained models. As a result, the models may be overfitted and may not generalize well. For randomly positioned obstacle avoidance, it's harder for trainings to converge because the agent must keep learning to avoid crashing into obstacles at locations it hasn't seen before. However,

models trained with this option tend to generalize better and transfer well to the real-world races. To begin, have obstacles placed at fixed locations, get familiar with the behaviors, and then tackle the random locations.

In the AWS DeepRacer simulator, the obstacles are cuboid boxes with the same dimensions (9.5" (L) x 15.25" (W) x 10/5" (H)) as the AWS DeepRacer vehicle's package box. This makes it simpler to transfer the trained model from the simulator to the real world if you place the packaging box as an obstacle on the physical track.

To experiment with obstacle avoidance, follow the recommended practice outlined in the steps below:

1. Use the default agent or experiment with new sensors and action spaces by customizing an existing agent or building a new one. You should limit the top speed to below 0.8 m/s and the speed granularity to 1 or 2 levels.

   Start training a model for around 3 hours with 2 objects at fixed locations. Use the example reward function and train the model on the track that you will be racing on, or a track that closely resembles that track. The **AWS DeepRacer Smile Speedway (Intermediate)** track is a simple track, which makes it a good choice for summit race preparation. Evaluate the model on the same track with the same number of obstacles. Watch how the total expected reward converges, if at all.
2. Read about the reward function parameters. Experiment with variations of your reward function. Increase the obstacle number to 4. Train the agent to see if the training converges in the same amount of training time. If it doesn't, tweak your reward function again, lower the top speed or reduce the number of obstacles, and the train the agent again. Repeat experimenting until there is no more significant improvement.
3. Now, move on to training avoiding obstacles at random locations. You'll need to configure the agent with additional sensors, which are available from **Garage** in the AWS DeepRacer console. You can use a stereo camera. Or you can combine a LiDAR unit with either a single-lens camera or a stereo camera, but should expect a longer training time. Set the action space with a relatively low top speed (for example 2 m/s) for the training to converge quicker. For the network architecture, use a shallow neural network, which has been found sufficient for obstacle avoidance.
4. Start training for 4 hours the new agent for obstacle avoidance with 4 randomly placed objects on an simple track. Then evaluate your model on the same track to see if it can finish laps with randomly positioned obstacles. If not, you may want to tweak your reward function, try

different sensors and have longer training time. As another tip, you can try cloning an existing model to continue training to leverage previously learned experience.

5. (Optional) Choose a higher top speed for the action space or have more obstacles randomly placed along the track. Experiment with different combination of sensors and tweak the reward functions and hyperparameter values. Experiment with the **5-layer CNN** network topology. Then, retrain the model to determine how they affect convergence of the training.

Tailor AWS DeepRacer training for head-to-bot races

Having gone through training obstacle avoidance, you're now ready to tackle the next level of challenge: training models for head-to-bot races. Unlike the obstacle avoidance events, head-to-bot racing has a dynamic environment with moving vehicles. Your goal is to train models for your AWS DeepRacer vehicle to compete against other moving vehicles in order to reach the finish line first without going off track or crashing to any of other vehicles. In the AWS DeepRacer console you can train a head-to-bot racing model by having your agent to compete against 1-4 bot vehicles. Generally speaking, you should have more obstacles placed on a longer track.

Each bot vehicle follows a predefined path at constant speed. You can enable it to change lanes or to remains on its starting lane. Similar to training for obstacle avoidance, you can have the bot vehicles evenly distributed across the track on both lanes. The console limits you to have up to 4 bot vehicles on the track. Having more competing vehicles on the track provides the learning agent with more opportunities to encounter more varied situations with the other vehicles. This way, it learns more in one training job and the agent gets trained faster. However, each training is likely to take longer to converge.

To train an agent with bot vehicles, you should set the top speed of the agent's action space higher than the (constant) speed of the bot vehicles so that the agent has more passing opportunities during training. As a good starting point, you should set the agent's top speed at 0.8 m/s and the bot vehicle's moving speed at 0.4 m/s. If you enable the bots to change lanes, the training becomes more challenging because the agent must learn not only how to avoid crashing into a moving vehicle in the front on the same lane but also how to avoid crashing into another moving vehicle in the front on the other lane. You can set the bots to change lanes at random intervals. The length of an interval is randomly selected from a range of time (for example 1s to 5s) that you specify before starting the training job. This lane-changing behavior is more similar to the real-world head-to-bot racing behaviors and the trained agent should generate better. However, it takes longer to train the model to converge.

Follow these suggested steps to iterate your training for head-to-bot racing:

1. In **Garage** of the AWS DeepRacer console, build a new training agent configured with both stereo cameras and a LiDAR unit. It is possible to train a relatively good model using only stereo camera against bot vehicles. LiDAR helps reduce blind spots when the agent changes lanes. Do not set the top speed too high. A good starting point is 1 m/s.
2. To train for head-to-bot racing, start with two bot vehicles. Set the bot's moving speed lower than your agent's top speed (for example 0.5 m/s if the agent's top speed is 1 m/s). Disable the lane-changing option, and then choose the training agent you just created. Use one of the reward function examples or make minimally necessary modifications, and then train for 3 hours. Use the track that you will be racing on, or a track that closely resembles that track. The **AWS DeepRacer Smile Speedway (Intermediate)** track is a simple track, which makes it a good choice for summit race preparation. After the training is complete, evaluate the trained model on the same track.
3. For more challenging tasks, clone your trained model for a second head-to-bot racing model. Proceed to either experiment with more bot vehicles or enable lane-changing options. Start with slow lane-changing operations at random intervals longer than 2 seconds. You may also want to experiment with custom reward functions. In general, your custom reward function logic can be similar to those for obstacle avoidance, if you don't take into consideration a balance between surpassing other vehicles and staying on track. Depends on how good your previous model is, you may need to train another 3 to 6 hours. Evaluate your models and see how the model performs.

Train and evaluate AWS DeepRacer models using the AWS DeepRacer console

To train a reinforcement learning model, you can use the AWS DeepRacer console. In the console, create a training job, choose a supported framework and an available algorithm, add a reward function, and configure training settings. You can also watch training proceed in a simulator. You can find the step-by-step instructions in the section called “Train your first model”.

This section explains how to train and evaluate an AWS DeepRacer model. It also shows how to create and improve a reward function, how an action space affects model performance, and how hyperparameters affect training performance. You can also learn how to clone a training model to extend a training session, how to use the simulator to evaluate training performance, and how to address some of the simulation to real-world challenges.

Topics

Create your reward function

A reward function describes immediate feedback (as a reward or penalty score) when your AWS DeepRacer vehicle moves from one position on the track to a new position. The function's purpose is to encourage the vehicle to make moves along the track to reach a destination quickly without accident or infraction. A desirable move earns a higher score for the action or its target state. An illegal or wasteful move earns a lower score. When training an AWS DeepRacer model, the reward function is the only application-specific part.

In general, you design your reward function to act like an incentive plan. Different incentive strategies could result in different vehicle behaviors. To make the vehicle drive faster, the function should give rewards for the vehicle to follow the track. The function should dispense penalties when the vehicle takes too long to finish a lap or goes off the track. To avoid zig-zag driving patterns, it could reward the vehicle to steer less on straighter portions of the track. The reward function might give positive scores when the vehicle passes certain milestones, as measured by `waypoints`. This could alleviate waiting or driving in the wrong direction. It is also likely that you would change the reward function to account for the track conditions. However, the more your reward function takes into account environment-specific information, the more likely your trained model is over-fitted and less general. To make your model more generally applicable, you can explore action space.

If an incentive plan is not carefully considered, it can lead to unintended consequences of opposite effect. This is possible because the immediate feedback is a necessary but not sufficient condition for reinforcement learning. An individual immediate reward by itself also can't determine if the move is desirable. At a given position, a move can earn a high reward. A subsequent move could go off the track and earn a low score. In such case, the vehicle should avoid the move of the high score at that position. Only when all future moves from a given position yield a high score on average

should the move to the next position be deemed desirable. Future feedback is discounted at a rate that allows for only a small number of future moves or positions to be included in the average reward calculation.

A good practice to create a reward function is to start with a simple one that covers basic scenarios. You can enhance the function to handle more actions. Let's now look at some simple reward functions.

Topics

Simple reward function examples

We can start building the reward function by first considering the most basic situation. The situation is driving on a straight track from start to finish without going off the track. In this scenario, the reward function logic depends only on `all_wheels_on_track` and `progress`. As a trial, you could start with the following logic:

```
def reward_function(params):
    if not params["all_wheels_on_track"]:
        reward = -1
    else if params["progress"] == 1 :
        reward = 10
    return reward
```

This logic penalizes the agent when it drives itself off the track. It rewards the agent when it drives to the finishing line. It's reasonable for achieving the stated goal. However, the agent roams freely between the starting point and the finishing line, including driving backwards on the track. Not only could the training take a long time to complete, but also the trained model would lead to less efficient driving when deployed to a real-world vehicle.

In practice, an agent learns more effectively if it can do so bit-by-bit throughout the course of training. This implies that a reward function should give out smaller rewards step by step along the track. For the agent to drive on the straight track, we can improve the reward function as follows:

```
def reward_function(params):
    if not params["all_wheels_on_track"]:
        reward = -1
```

```
    else:
        reward = params["progress"]
    return reward
```

With this function, the agent gets more reward the closer it reaches the finishing line. This should reduce or eliminate unproductive trials of driving backwards. In general, we want the reward function to distribute the reward more evenly over the action space. Creating an effective reward function can be a challenging undertaking. You should start with a simple one and progressively enhance or improve the function. With systematic experimentation, the function can become more robust and efficient.

Enhance your reward function

After you have successfully trained your AWS DeepRacer model for the simple straight track, the AWS DeepRacer vehicle (virtual or physical) can drive itself without going off the track. If you let the vehicle run on a looped track, it won't stay on the track. The reward function has ignored the actions to make turns to follow the track.

To make your vehicle handle those actions, you must enhance the reward function. The function must give a reward when the agent makes a permissible turn and produce a penalty if the agent makes an illegal turn. Then, you're ready to start another round of training. To take advantage of the prior training, you can start the new training by cloning the previously trained model, passing along the previously learned knowledge. You can follow this pattern to gradually add more features to the reward function to train your AWS DeepRacer vehicle to drive in increasingly more complex environments.

For more advanced reward functions, see the following examples:

- the section called “Example 1: Follow the center line in time trials”
- the section called “Example 2: Stay inside the two borders in time trials”
- the section called “Example 3: Prevent zig-zag in time trials”
- the section called “Example 4: Stay in one lane without crashing into stationary obstacles or moving vehicles”

Explore action space to train a robust model

As a general rule, train your model to be as robust as possible so that you can apply it to as many environments as possible. A robust model is one that can be applied to a wide range of track

shapes and conditions. Generally speaking, a robust model is not "smart" because its reward function does not have the ability to contain explicit environment-specific knowledge. Otherwise, your model is likely to be applicable only to an environment similar to the trained one.

Explicitly incorporating environment-specific information into the reward function amounts to feature engineering. Feature engineering helps reduce training time and can be useful in solutions tailor made to a particular environment. To train a model of the general applicability though, you should refrain from attempting a lot of feature engineering.

For example, when training a model on a circular track, you can't expect to obtain a trained model applicable to any non-circular track if you have such geometric properties explicitly incorporated into the reward function.

How would you go about training a model as robust as possible while keeping the reward function as simple as possible? One way is to explore the action space spanning the actions your agent can take. Another is to experiment with hyperparameters of underlying training algorithm. Often times, you do both. Here, we focus on how to explore the action space to train a robust model for your AWS DeepRacer vehicle.

In training an AWS DeepRacer model, an action (a) is a combination of speed (t meters per second) and steering angle (s in degrees). The action space of the agent defines the ranges of speed and steering angle the agent can take. For a discrete action space of m number of speeds, $(v_1, .., v_n)$ and n number of steering angles, $(s_1, .., s_m)$, there are m*n possible actions in the action space:

$$
\begin{aligned}
&a_1: && (v_1, s_1) \\
&... \\
&a_n: && (v_1, s_n) \\
& \\
&... \\
&a_{(i-1)*n+j}: && (v_i, s_j) \\
&... \\
& \\
&a_{(m-1)*n+1}: && (v_m, s_1) \\
&... \\
&a_{m*n}: && (v_m, s_n)
\end{aligned}
$$

The actual values of (v_i, s_j) depend on the ranges of v_{max} and $|s_{max}|$ and are not uniformly distributed.

Each time you begin training or iterating your AWS DeepRacer model, you must first specify the n, m, v_{max} and $|s_{max}|$ or agree to using their default values. Based on your choice, the AWS DeepRacer service generates the available actions your agent can choose in training. The generated actions are not uniformly distributed over the action space.

In general, a larger number of actions and larger action ranges give your agent more room or options to react to more varied track conditions, such as a curved track with irregular turning angles or directions. The more options available to the agent, the more readily it can handle track variations. As a result, you can expect that the trained model to be more widely applicable, even when using a simple reward function .

For example, your agent can learn quickly to handle straight-line track using a coarse-grained action space with small number of speeds and steering angles. On a curved track, this coarse-grained action space is likely to cause the agent to overshoot and go off the track while it turns. This is because there are not enough options at its disposal in order to adjust its speed or steering. Increase the number of speeds or the number of steering angles or both, the agent should become more capable of maneuvering the curves while keeping on the track. Similarly, if your agent moves in a zig-zag fashion, you can try to increase the number of steering ranges to reduce drastic turns at any given step.

When the action space is too large, training performance may suffer, because it takes longer to explore the action space. Be sure to balance the benefits of a model's general applicability against its training performance requirements. This optimization involves systematic experimentation.

Systematically tune hyperparameters

One way to improve your model's performance is to enact a better or more effective training process. For example, to obtain a robust model, training must provide your agent more or less evenly distributed sampling over the agent's action space. This requires a sufficient mix of exploration and exploitation. Variables affecting this include the amount of training data used (`number of episodes between each training` and `batch size`), how fast the agent can learn (`learning rate`), the portion of exploration (`entropy`). To make training practical, you may want to speed the learning process. Variables affecting this include `learning rate`, `batch size`, `number of epochs` and `discount factor`.

The variables affecting the training process are known as hyperparameters of the training. These algorithm attributes are not properties of the underlying model. Unfortunately, hyperparameters are empirical in nature. Their optimal values are not known for all practical purposes and require systematic experimentation to derive.

Before discussing the hyperparameters that can be adjusted to tune the performance of training your AWS DeepRacer model, let's define the following terminology.

Data point

A data point, also known as an *experience*, it is a tuple of (*s,a,r,s'*), where *s* stands for an observation (or state) captured by the camera, *a* for an action taken by the vehicle, *r* for the expected reward incurred by the said action, and *s'* for the new observation after the action is taken.

Episode

An episode is a period in which the vehicle starts from a given starting point and ends up completing the track or going off the track. It embodies a sequence of experiences. Different episodes can have different lengths.

Experience buffer

An experience buffer consists of a number of ordered data points collected over fixed number of episodes of varying lengths during training. For AWS DeepRacer, it corresponds to images captured by the camera mounted on your AWS DeepRacer vehicle and actions taken by the vehicle and serves as the source from which input is drawn for updating the underlying (policy and value) neural networks.

Batch

A batch is an ordered list of experiences, representing a portion of simulation over a period of time, used to update the policy network weights. It is a subset of the experience buffer.

Training data

A training data is a set of batches sampled at random from an experience buffer and used for training the policy network weights.

Algorithmic hyperparameters and their effects

Hyperparameters	Description
Gradient descent batch size	The number recent vehicle experiences sampled at random from an experience buffer and used for updating the underlying deep-learning neural network weights. Random sampling helps reduce correlations inherent in the input data.

Hyperparameters	Description
	Use a larger batch size to promote more stable and smooth updates to the neural network weights, but be aware of the possibility that the training may be longer or slower. Required Yes Valid values Positive integer of `(32, 64, 128, 256, 512)` Default value `64`
Number of epochs	The number of passes through the training data to update the neural network weights during gradient descent. The training data corresponds to random samples from the experience buffer. Use a larger number of epochs to promote more stable updates, but expect a slower training. When the batch size is small, you can use a smaller number of epochs Required No Valid values Positive integer between `[3 - 10]` Default value 3

Hyperparameters	Description
Learning rate	During each update, a portion of the new weight can be from the gradient-descent (or ascent) contribution and the rest from the existing weight value. The learning rate controls how much a gradient-descent (or ascent) update contributes to the network weights. Use a higher learning rate to include more gradient-descent contributions for faster training, but be aware of the possibility that the expected reward may not converge if the learning rate is too large. Required No Valid values Real number between `0.00000001` (or 10^{-8}) and `0.001` (or 10^{-3}) Default value `0.0003`
Entropy	A degree of uncertainty used to determine when to add randomness to the policy distribution. The added uncertainty helps the AWS DeepRacer vehicle explore the action space more broadly. A larger entropy value encourages the vehicle to explore the action space more thoroughly. Required No Valid values Real number between `0` and `1`. Default value `0.01`

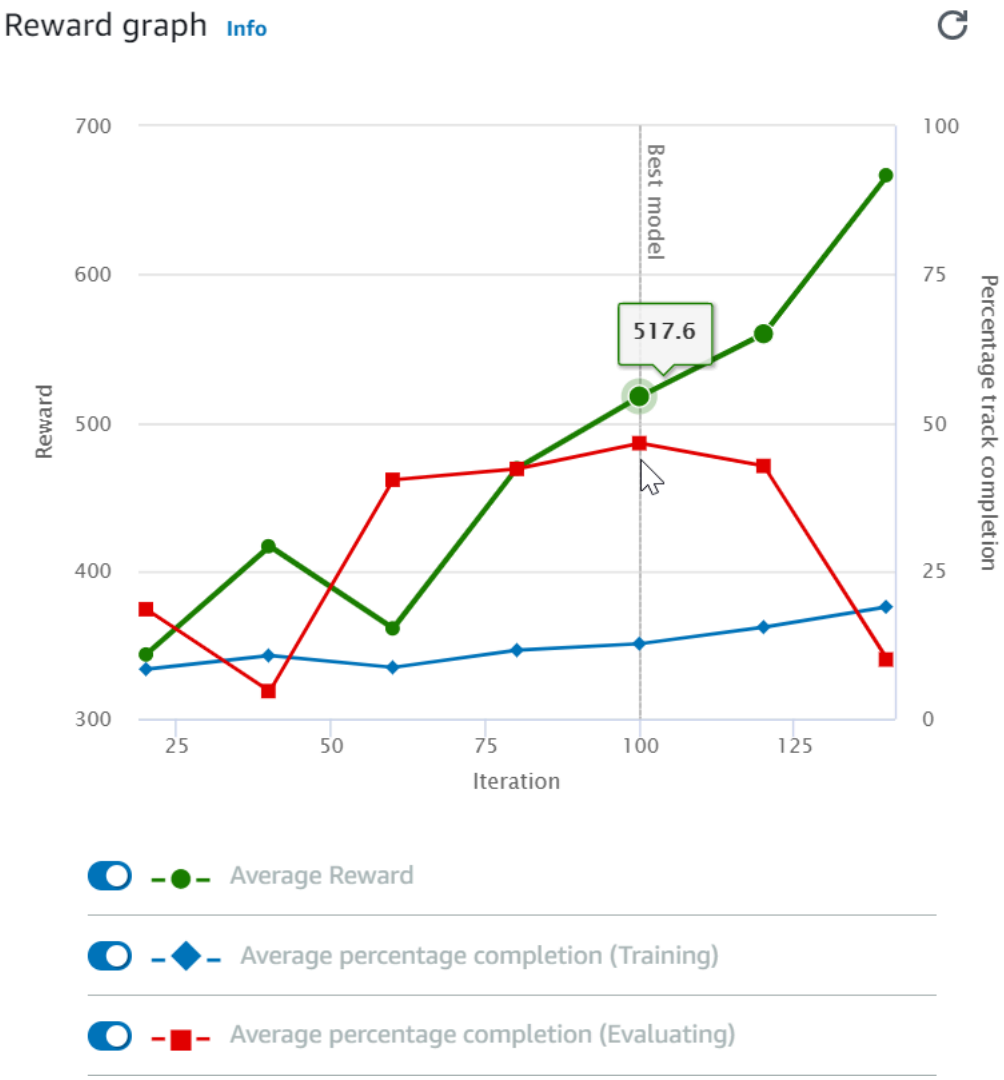
Hyperparameters	Description
Discount factor	A factor specifies how much of the future rewards contribute to the expected reward. The larger the **Discount factor** value is, the farther out contributions the vehicle considers to make a move and the slower the training. With the discount factor of 0.9, the vehicle includes rewards from an order of 10 future steps to make a move. With the discount factor of 0.999, the vehicle considers rewards from an order of 1000 future steps to make a move. The recommended discount factor values are 0.99, 0.999 and 0.9999. Required No Valid values Real number between `0` and 1. Default value `0.999`

Hyperparameters	Description
Loss type	Type of the objective function used to update the network weights. A good training algorithm should make incremental changes to the agent's strategy so that it gradually transitions from taking random actions to taking strategic actions to increase reward. But if it makes too big a change then the training becomes unstable and the agent ends up not learning. The **Huber loss** and **Mean squared error loss** types behave similarly for small updates. But as the updates become larger, **Huber loss** takes smaller increments compared to **Mean squared error loss**. When you have convergence problems, use the **Huber loss** type. When convergence is good and you want to train faster, use the **Mean squared error loss** type. Required No Valid values (**Huber loss**, **Mean squared error loss**) Default value **Huber loss**

Hyperparameters	Description
Number of experience episodes between each policy-updating iteration	The size of the experience buffer used to draw training data from for learning policy network weights. An experience episode is a period in which the agent starts from a given starting point and ends up completing the track or going off the track. It consists of a sequence of experiences. Different episodes can have different lengths. For simple reinforcement-learning problems, a small experience buffer may be sufficient and learning is fast. For more complex problems that have more local maxima, a larger experience buffer is necessary to provide more uncorrelated data points. In this case, training is slower but more stable. The recommended values are 10, 20 and 40. Required No Valid values Integer between 5 and 100 Default value 20

Examine AWS DeepRacer training job progress

After starting your training job, you can examine the training metrics of rewards and track completion per episode to ascertain the training job's performance of your model. On the AWS DeepRacer console, the metrics are displayed in the **Reward graph**, as shown in the following illustration.



You can choose to view the reward gained per episode, the averaged reward per iteration, the progress per episode, the averaged progress per iteration or any combination of them. To do so, toggle the **Reward (Episode, Average)** or **Progress (Episode, Average)** switches at the bottom of **Reward graph**. The reward and progress per episode are displayed as scattered plots in different colors. The averaged reward and track completion are displayed by line plots and start after the first iteration.

The range of rewards is shown on the left side of the graph and the range of progress (0-100) is on the right side. To read the exact value of of a training metric, move the mouse near to the data point on the graph.

The graphs are automatically updated every 10 seconds while training is under way. You can choose the refresh button to manually update the metric display.

A training job is good if the averaged reward and track completion show trends to converge. In particular, the model has likely converged if the progress per episode continuously reach 100% and the reward levels out. If not, clone the model and retrain it.

Clone a trained model to start a new training pass

If you clone a previously trained model as the starting point of a new round of training, you could improve training efficiency. To do this, modify the hyperparameters to make use of already learned knowledge.

In this section, you learn how to clone a trained model using the AWS DeepRacer console.

To iterate training the reinforcement learning model using the AWS DeepRacer console

1. Sign in to the AWS DeepRacer console, if you're not already signed in.
2. On the **Models** page, choose a trained model and then choose **Clone** from the **Action** drop-down menu list.
3. For **Model details**, do the following:

    a. Type `RL_model_1` in **Model name**, if you don't want a name to be generated for the cloned model.

    b. Optionally, give a description for the to-be-cloned model in **Model description - optional**.
4. For **Environment simulation**, choose another track option.
5. For **Reward function**, choose one of the available reward function examples. Modify the reward function. For example, consider steering.
6. Expand **Algorithm settings** and try different options. For example, change the **Gradient descent batch size** value from 32 to 64 or increase the **Learning rate** to speed up the training.
7. Experiment with difference choices of the **Stop conditions**.
8. Choose **Start training** to begin new round of training.

As with training a robust machine learning model in general, it is important that you conduct systematic experimentation to come up with the best solution.

Evaluate AWS DeepRacer models in simulations

To evaluate a model is to test the performance of a trained model. In AWS DeepRacer, the standard performance metric is the average time of finishing three consecutive laps. Using this metric, for

any two models, one model is better than the other if it can make the agent go faster on the same track.

In general, evaluating a model involves the following tasks:

1. Configure and start an evaluation job.
2. Observe the evaluation in progress while the job is running. This can be done in the AWS DeepRacer simulator.
3. Inspect the evaluation summary after the evaluation job is done. You can terminate an evaluation job in progress at any time.

   > **Note**
   >
   > The evaluation time depends on the criteria you select. If your model doesn't meet the evaluation criteria, the evaluation will keep running until it reaches the 20 minute cap.

4. Optionally, submit the evaluation result to an eligible AWS DeepRacer leaderboard. The ranking on the leaderboard lets you know how well your model performs against other participants.

Test an AWS DeepRacer model with an AWS DeepRacer vehicle driving on a physical track, see *Operate your vehicle*.

Optimize training AWS DeepRacer models for real environments

Many factors affect the real-world performance of a trained model, including the choice of the action space, reward function, hyperparameters used in the training, and vehicle calibration as well as real-world track conditions. In addition, the simulation is only an (often crude) approximation of the real world. They make it a challenge to train a model in simulation, to apply it to the real world, and to achieve a satisfactory performance.

Training a model to give a solid real-world performance often requires numerous iterations of exploring the reward function, action spaces, hyperparameters, and evaluation in simulation and testing in a real environment. The last step involves the so-called *simulation-to-real world* (*sim2real*) transfer and can feel unwieldy.

To help tackle the *sim2real* challenges, heed the following considerations:

- Make sure that your vehicle is well calibrated.

This is important because the simulated environment is most likely a partial representation of the real environment. Besides, the agent takes an action based on the current track condition, as captured by an image from the camera, at each step. It can't see far enough to plan its route at a fast speed. To accommodate this, the simulation imposes limits on the speed and steering. To ensure the trained model works in the real world, the vehicle must be properly calibrated to match this and other simulation settings. For more information for calibrating your vehicle, see the section called “Calibrate your vehicle”.

- Test your vehicle with the default model first.

  Your AWS DeepRacer vehicle comes with a pre-trained model loaded into its inference engine. Before testing your own model in the real world, verify that the vehicle performs reasonably well with the default model. If not, check the physical track setup. Testing a model in an incorrectly built physical track is likely to lead to a poor performance. In such cases, reconfigure or repair your track before starting or resuming testing.

  > **Note**
  >
  > When running your AWS DeepRacer vehicle, actions are inferred according to the trained policy network without invoking the reward function.

- Make sure the model works in simulation.

  If your model doesn't work well in the real world, it's possible that either the model or track is defective. To sort out the root causes, you should first evaluate the model in simulations to check if the simulated agent can finish at least one loop without getting off the track. You can do so by inspecting the convergence of the rewards while observing the agent's trajectory in the simulator. If the reward reaches the maximum when the simulated agents completes a loop without faltering, the model is likely to be a good one.

- Do not over train the model.

  Continuing training after the model has consistently completed the track in simulation will cause overfitting in the model. An over-trained model won't perform well in the real world because it can't handle even minor variations between the simulated track and the real environment.

- Use multiple models from different iterations.

  A typical training session produces a range of models that fall between being underfitted and being overfitted. Because there are no a priori criteria to determine a model that is just right, you

should pick a few model candidates from the time when the agent completes a single loop in the simulator to the point where it performs loops consistently.

- Start slow and increase the driving speed gradually in testing.

  When testing the model deployed to your vehicle, start with a small maximum speed value. For example, you can set the testing speed limit to be <10% of the training speed limit. Then gradually increase the testing speed limit until the vehicle starts moving. You set the testing speed limit when calibrating the vehicle using the device control console. If the vehicle goes too fast, for example if the speed exceeds those seen during training in simulator, the model is not likely to perform well on the real track.

- Test a model with your vehicle in different starting positions.

  The model learns to take a certain path in simulation and can be sensitive to its position within the track. You should start the vehicle tests with different positions within the track boundaries (from left to center to right) to see if the model performs well from certain positions. Most models tend to make the vehicle stay close to either side of one of the white lines. To help analyze the vehicle's path, plot the vehicle's positions (x, y) step by step from the simulation to identify likely paths to be taken by your vehicle in a real environment.

- Start testing with a straight track.

  A straight track is much easier to navigate compared to a curved track. Starting your test with a straight track is useful to weed out poor models quickly. If a vehicle cannot follow a straight track most of the time, the model will not perform well on curved tracks, either.

- Watch out for the behavior where the vehicle takes only one type of actions,

  When your vehicle can manage to take only one type of actions, for example, to steer the vehicle to the left only, the model is likely over-fitted or under-fitted. With given model parameters, too many iterations in training could make the model over-fitted. Too few iterations could make it under-fitted.

- Watch out for vehicle's ability to correct its path along a track border.

  A good model makes the vehicle to correct itself when nearing the track borders. Most well-trained models have this capability. If the vehicle can correct itself on both the track borders, the model is considered to be more robust and of a higher quality.

- Watch out for inconsistent behaviors exhibited by the vehicle.

A policy model represents a probability distribution for taking an action in a given state. With the trained model loaded to its inference engine, a vehicle will pick the most probable action, one step at time, according to the model's prescription. If the action probabilities are evenly distributed, the vehicle can take any of the actions of the equal or closely similar probabilities. This will lead to an erratic driving behavior. For example, when the vehicle follows a straight path sometimes (for example, half the time) and makes unnecessary turns at other times, the model is either under-fitted or over-fitted.

- Watch out for only one type of turn (left or right) made by the vehicle.

  If the vehicle takes left turns very well but fails to manage steering right, or, similarly, if the vehicle takes only right turns well, but not left steering, you need to carefully calibrate or recalibrate your vehicle's steering. Alternatively, you can try to use a model that is trained with the settings close to the physical settings under testing.
- Watch out for the vehicle making sudden turns and going off-track.

  If the vehicle follows the path correctly most of the way, but suddenly veers off the track, it is likely due to distractions in the environment. Most common distractions include unexpected or unintended light reflections. In such cases, use barriers around the track or other means to reduce glaring lights.

AWS DeepRacer reward function reference

The following is the technical reference of the AWS DeepRacer reward function.

Topics

Input parameters of the AWS DeepRacer reward function

The AWS DeepRacer reward function takes a dictionary object as the input.

```
def reward_function(params) :

    reward = ...
```

```
    return float(reward)
```

The `params` dictionary object contains the following key-value pairs:

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

}
```

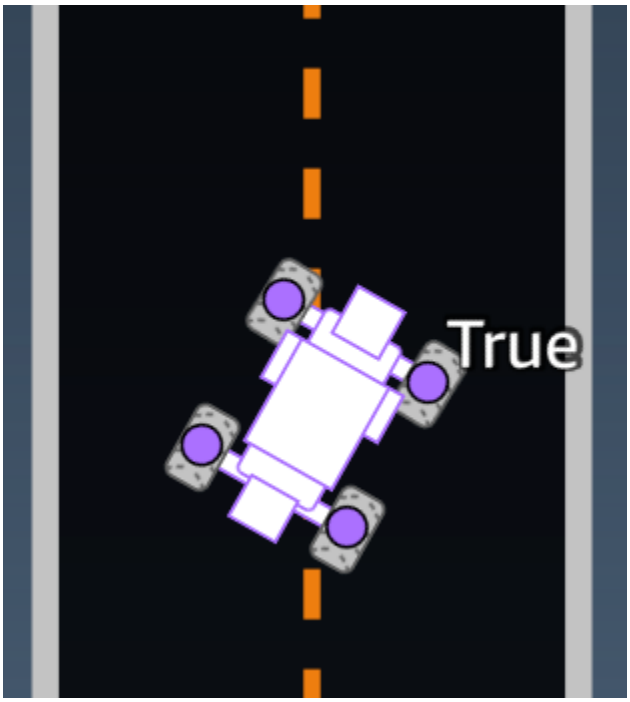
A more detailed technical reference of the input parameters is as follows.

all_wheels_on_track

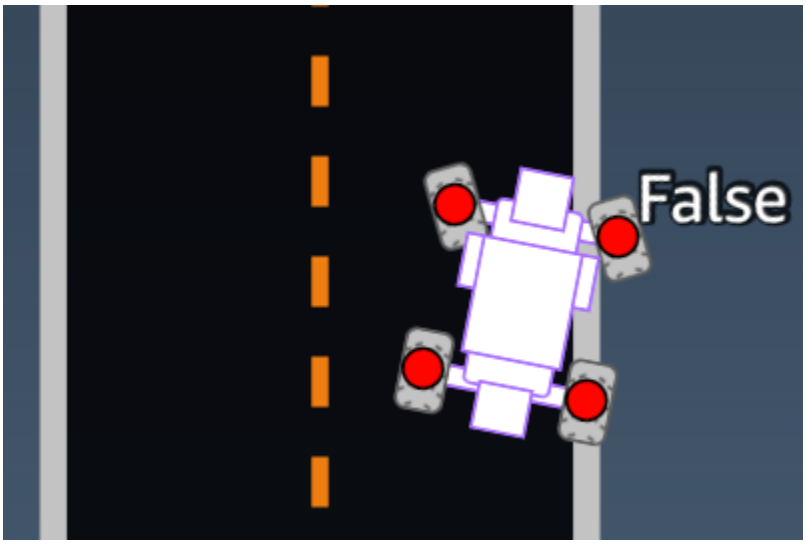
Type: `Boolean`

Range: `(True:False)`

A `Boolean` flag to indicate whether the agent is on-track or off-track. It's off-track (`False`) if any of its wheels are outside of the track borders. It's on-track (`True`) if all of the wheels are inside the two track borders. The following illustration shows that the agent is on-track.



The following illustration shows that the agent is off-track.



Example: *A reward function using the `all_wheels_on_track` parameter*

```
def reward_function(params):
    ################################################################################
    '''
    Example of using all_wheels_on_track and speed
    '''

    # Read input variables
    all_wheels_on_track = params['all_wheels_on_track']
    speed = params['speed']

    # Set the speed threshold based your action space
    SPEED_THRESHOLD = 1.0

    if not all_wheels_on_track:
        # Penalize if the car goes off track
        reward = 1e-3
    elif speed < SPEED_THRESHOLD:
        # Penalize if the car goes too slow
        reward = 0.5
    else:
        # High reward if the car stays on track and goes fast
        reward = 1.0

    return float(reward)
```

closest_waypoints

Type: `[int, int]`

Range: `[(0:Max-1),(1:Max-1)]`

The zero-based indices of the two neighboring `waypoints` closest to the agent's current position of `(x, y)`. The distance is measured by the Euclidean distance from the center of the agent. The first element refers to the closest waypoint behind the agent and the second element refers the closest waypoint in front of the agent. `Max` is the length of the waypoints list. In the illustration shown in waypoints, the `closest_waypoints` would be `[16, 17]`.

Example: A reward function using the `closest_waypoints` parameter.

The following example reward function demonstrates how to use `waypoints` and `closest_waypoints` as well as `heading` to calculate immediate rewards.

AWS DeepRacer supports the following libraries: math, random, NumPy, SciPy, and Shapely. To use one, add an import statement, `import` *`supported library`*, above your function definition, `def function_name(parameters)`.

```
# Place import statement outside of function (supported libraries: math, random, numpy,
 scipy, and shapely)
# Example imports of available libraries
#
# import math
# import random
# import numpy
# import scipy
# import shapely

import math

def reward_function(params):
    ################################################################################
    '''
    Example of using waypoints and heading to make the car point in the right direction
    '''

    # Read input variables
    waypoints = params['waypoints']
    closest_waypoints = params['closest_waypoints']
    heading = params['heading']

    # Initialize the reward with typical value
    reward = 1.0

    # Calculate the direction of the center line based on the closest waypoints
    next_point = waypoints[closest_waypoints[1]]
    prev_point = waypoints[closest_waypoints[0]]

    # Calculate the direction in radius, arctan2(dy, dx), the result is (-pi, pi) in
 radians
    track_direction = math.atan2(next_point[1] - prev_point[1], next_point[0] -
 prev_point[0])
    # Convert to degree
    track_direction = math.degrees(track_direction)
```

```
    # Calculate the difference between the track direction and the heading direction of
the car
    direction_diff = abs(track_direction - heading)
    if direction_diff > 180:
        direction_diff = 360 - direction_diff

    # Penalize the reward if the difference is too large
    DIRECTION_THRESHOLD = 10.0
    if direction_diff > DIRECTION_THRESHOLD:
        reward *= 0.5

    return float(reward)
```

closest_objects

Type: `[int, int]`

Range: `[(0:len(objects_location)-1), (0:len(objects_location)-1)]`

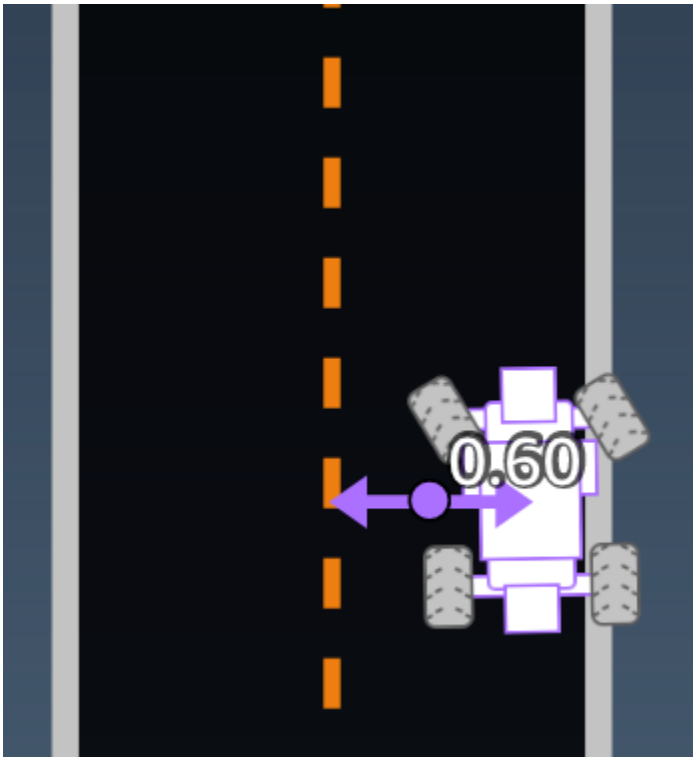
The zero-based indices of the two closest objects to the agent's current position of (x, y). The first index refers to the closest object behind the agent, and the second index refers to the closest object in front of the agent. If there is only one object, both indices are 0.

distance_from_center

Type: `float`

Range: `0:~track_width/2`

Displacement, in meters, between the agent center and the track center. The observable maximum displacement occurs when any of the agent's wheels are outside a track border and, depending on the width of the track border, can be slightly smaller or larger than half the `track_width`.



Example: *A reward function using the `distance_from_center` parameter*

```
def reward_function(params):
    ################################################################################
    '''
    Example of using distance from the center
    '''

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

    # Penalize if the car is too far away from the center
    marker_1 = 0.1 * track_width
    marker_2 = 0.5 * track_width

    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
    else:
        reward = 1e-3  # likely crashed/ close to off track
```

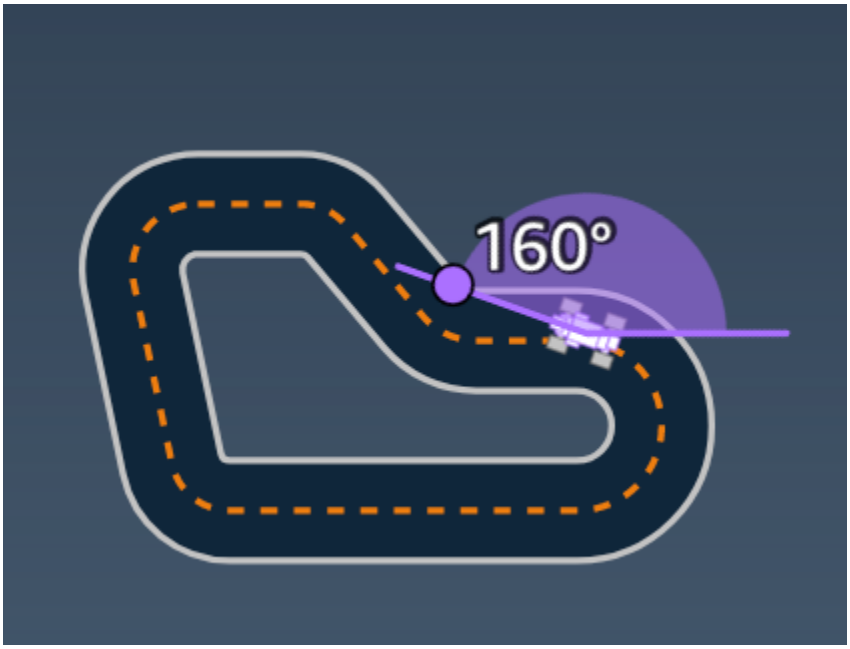
```
    return float(reward)
```

heading

Type: `float`

Range: `-180:+180`

Heading direction, in degrees, of the agent with respect to the x-axis of the coordinate system.



Example: *A reward function using the `heading` parameter*

For more information, see `closest_waypoints`.

is_crashed

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate whether the agent has crashed into another object (`True`) or not (`False`) as a termination status.

is_left_of_center

Type: `Boolean`

Range: `[True : False]`

A `Boolean` flag to indicate if the agent is on the left side to the track center (`True`) or on the right side (`False`).

is_offtrack

Type: `Boolean`

Range: `(True:False)`

A Boolean flag to indicate whether the agent has off track (True) or not (False) as a termination status.

is_reversed

Type: `Boolean`

Range: `[True:False]`

A Boolean flag to indicate if the agent is driving on clock-wise (True) or counter clock-wise (False).

It's used when you enable direction change for each episode.

objects_distance

Type: `[float, … ]`

Range: `[(0:track_length), … ]`

A list of the distances between objects in the environment in relation to the starting line. The i^{th} element measures the distance in meters between the i^{th} object and the starting line along the track center line.

> **Note**
>
> abs | (var1) - (var2)| = how close the car is to an object, WHEN var1 = ["objects_distance"][index] and var2 = params["progress"]*params["track_length"]
> To get an index of the closest object in front of the vehicle and the closest object behind the vehicle, use the "closest_objects" parameter.

objects_heading

Type: `[float, … ]`

Range: `[(-180:180), … ]`

List of the headings of objects in degrees. The i^{th} element measures the heading of the i^{th} object. For stationary objects, their headings are 0. For a bot vehicle, the corresponding element's value is the vehicle's heading angle.

objects_left_of_center

Type: `[Boolean, … ]`

Range: `[True|False, … ]`

List of Boolean flags. The i^{th} element value indicates whether the i^{th} object is to the left (True) or right (False) side of the track center.

objects_location

Type: `[(x,y), … ]`

Range: `[(0:N,0:N), … ]`

List of all object locations, each location is a tuple of (x, y).

The size of the list equals the number of objects on the track. Note the object could be the stationary obstacles, moving bot vehicles.

objects_speed

Type: `[float, … ]`

Range: `[(0:12.0), … ]`

List of speeds (meters per second) for the objects on the track. For stationary objects, their speeds are 0. For a bot vehicle, the value is the speed you set in training.

progress

Type: `float`

Range: `0:100`

Percentage of track completed.

Example: *A reward function using the `progress` parameter*

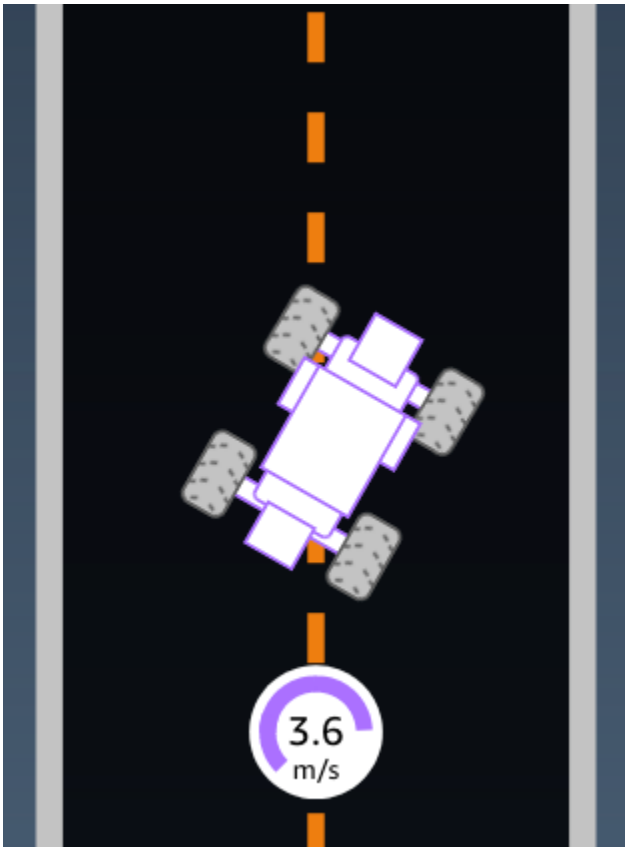
For more information, see steps.

speed

Type: `float`

Range: `0.0:5.0`

The observed speed of the agent, in meters per second (m/s).



Example: *A reward function using the `speed` parameter*

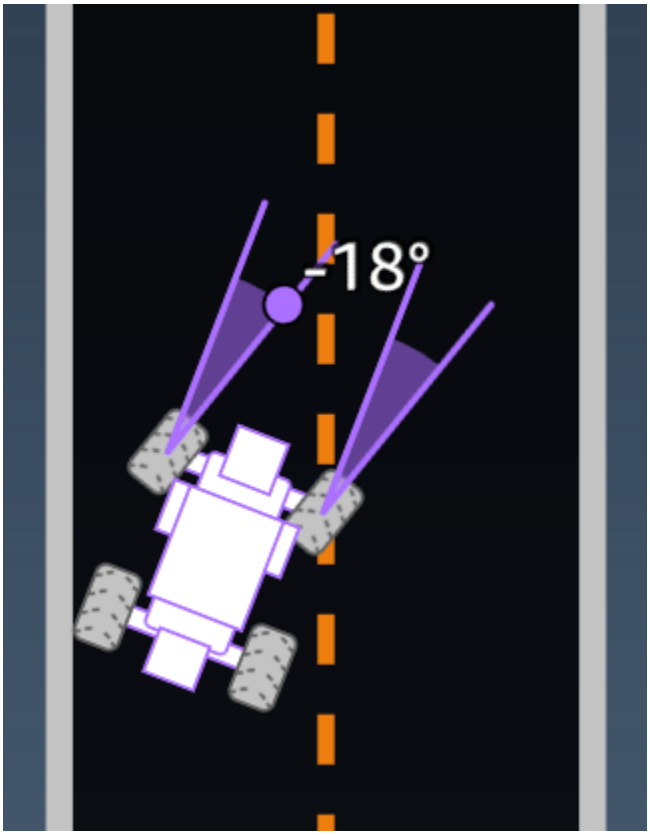
For more information, see all_wheels_on_track.

steering_angle

Type: `float`

Range: `-30:30`

Steering angle, in degrees, of the front wheels from the center line of the agent. The negative sign (-) means steering to the right and the positive (+) sign means steering to the left. The agent center line is not necessarily parallel with the track center line as is shown in the following illustration.



Example: *A reward function using the `steering_angle` parameter*

```
def reward_function(params):
    '''
    Example of using steering angle
    '''

    # Read input variable
    abs_steering = abs(params['steering_angle']) # We don't care whether it is left or
 right steering

    # Initialize the reward with typical value
    reward = 1.0

    # Penalize if car steer too much to prevent zigzag
    ABS_STEERING_THRESHOLD = 20.0
    if abs_steering > ABS_STEERING_THRESHOLD:
        reward *= 0.8
```

```
    return float(reward)
```

steps

Type: `int`

Range: `0:`N_{step}

Number of steps completed. A step corresponds to an action taken by the agent following the current policy.

Example: *A reward function using the `steps` parameter*

```
def reward_function(params):
    #############################################################################
    '''
    Example of using steps and progress
    '''

    # Read input variable
    steps = params['steps']
    progress = params['progress']

    # Total num of steps we want the car to finish the lap, it will vary depends on the
 track length
    TOTAL_NUM_STEPS = 300

    # Initialize the reward with typical value
    reward = 1.0

    # Give additional reward if the car pass every 100 steps faster than expected
    if (steps % 100) == 0 and progress > (steps / TOTAL_NUM_STEPS) * 100 :
        reward += 10.0

    return float(reward)
```

track_length

Type: `float`

Range: `[0:`L_{max}`]`

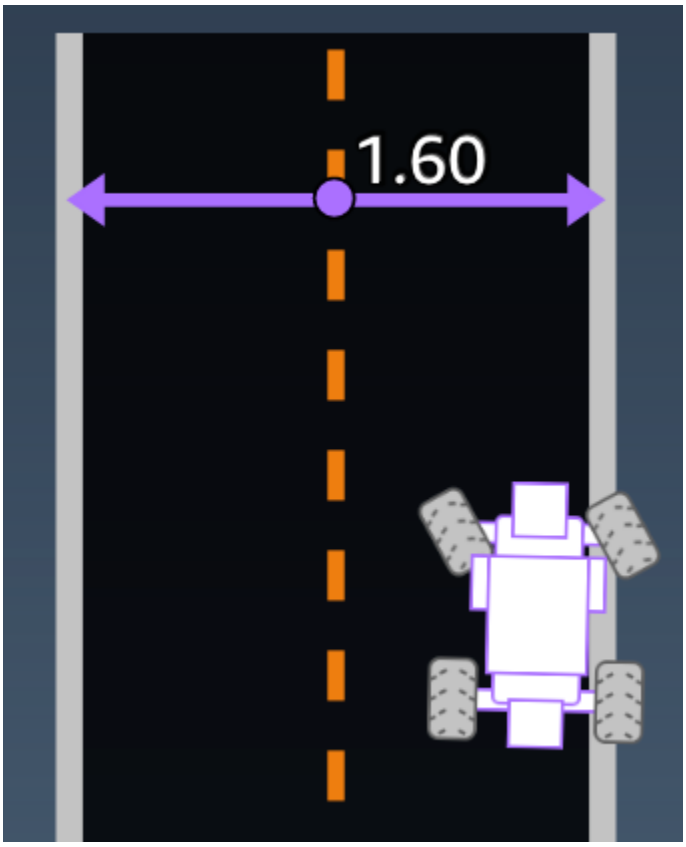
The track length in meters. L_{max} `is track-dependent.`

track_width

Type: `float`

Range: `0:`D_{track}

Track width in meters.



Example: *A reward function using the `track_width` parameter*

```
def reward_function(params):
    ################################################################################
    '''
    Example of using track width
    '''

    # Read input variable
    track_width = params['track_width']
    distance_from_center = params['distance_from_center']
```

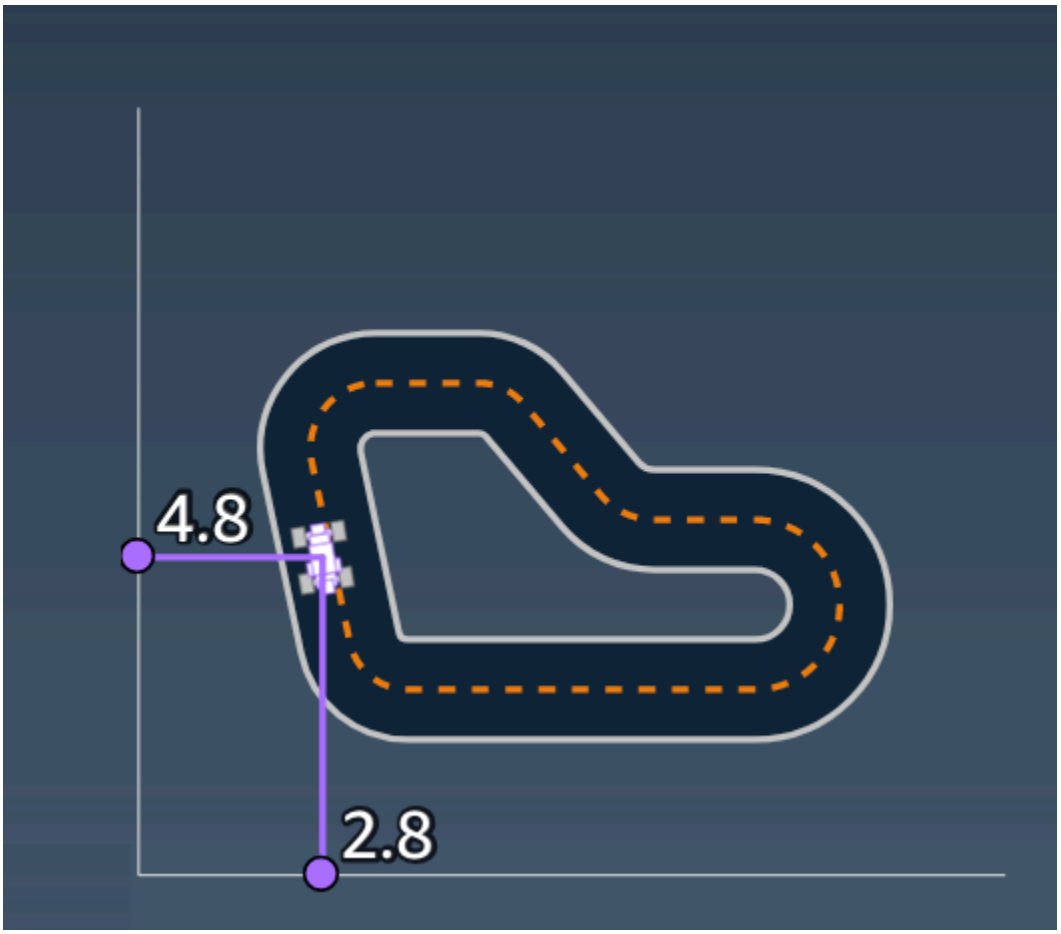
```
    # Calculate the distance from each border
    distance_from_border = 0.5 * track_width - distance_from_center

    # Reward higher if the car stays inside the track borders
    if distance_from_border >= 0.05:
        reward = 1.0
    else:
        reward = 1e-3 # Low reward if too close to the border or goes off the track

    return float(reward)
```

x, y

Type: `float`

Range: `0:N`

Location, in meters, of the agent center along the x and y axes, of the simulated environment containing the track. The origin is at the lower-left corner of the simulated environment.

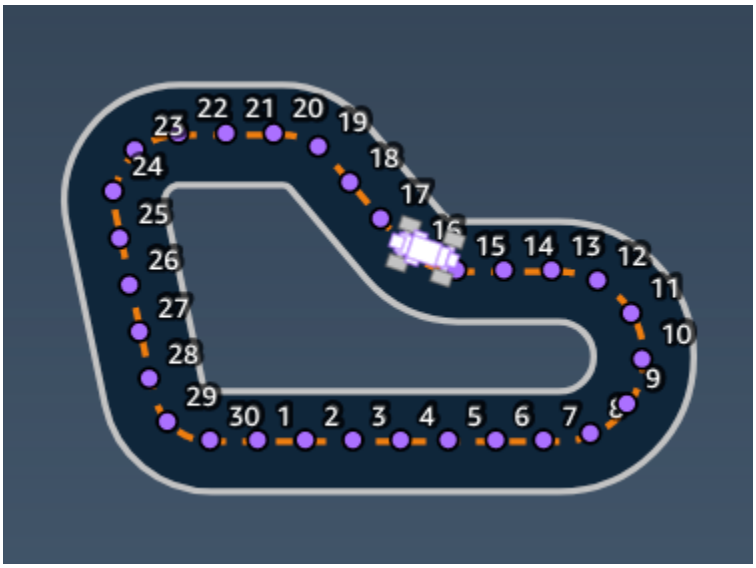


waypoints

Type: `list of [float, float]`

Range: `[[`$x_{w,0}$`,`$y_{w,0}$`] … [`$x_{w,Max-1}$`, `$y_{w,Max-1}$`]]`

An ordered list of track-dependent `Max` milestones along the track center. Each milestone is described by a coordinate of ($x_{w,i}$, $y_{w,i}$). For a looped track, the first and last waypoints are the same. For a straight or other non-looped track, the first and last waypoints are different.



Example *A reward function using the `waypoints` parameter*

For more information, see `closest_waypoints`.

AWS DeepRacer reward function examples

The following lists some examples of the AWS DeepRacer reward function.

Topics

- Example 1: Follow the center line in time trials
- Example 2: Stay inside the two borders in time trials
- Example 3: Prevent zig-zag in time trials
- Example 4: Stay in one lane without crashing into stationary obstacles or moving vehicles

Example 1: Follow the center line in time trials

This example determines how far away the agent is from the center line, and gives higher reward if it is closer to the center of the track, encouraging the agent to closely follow the center line.

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

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

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

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

    return reward
```

Example 2: Stay inside the two borders in time trials

This example simply gives high rewards if the agent stays inside the borders, and lets the agent figure out the best path to finish a lap. It's easy to program and understand, but likely takes longer to converge.

```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside the two borders of the track
    '''
```

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

    # Give a very low reward by default
    reward = 1e-3

    # Give a high reward if no wheels go off the track and
    # the car is somewhere in between the track borders
    if all_wheels_on_track and (0.5*track_width - distance_from_center) >= 0.05:
        reward = 1.0

    # Always return a float value
    return reward
```

Example 3: Prevent zig-zag in time trials

This example incentivizes the agent to follow the center line but penalizes with lower reward if it steers too much, which helps prevent zig-zag behavior. The agent learns to drive smoothly in the simulator and likely keeps the same behavior when deployed to the physical vehicle.

```
def reward_function(params):
    '''
    Example of penalize steering, which helps mitigate zig-zag behaviors
    '''

    # Read input parameters
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    abs_steering = abs(params['steering_angle']) # Only need the absolute steering
 angle

    # Calculate 3 marks that are farther and father away from the center line
    marker_1 = 0.1 * track_width
    marker_2 = 0.25 * track_width
    marker_3 = 0.5 * track_width

    # Give higher reward if the car is closer to center line and vice versa
    if distance_from_center <= marker_1:
        reward = 1.0
    elif distance_from_center <= marker_2:
        reward = 0.5
```

```
    elif distance_from_center <= marker_3:
        reward = 0.1
    else:
        reward = 1e-3  # likely crashed/ close to off track

    # Steering penality threshold, change the number based on your action space setting
    ABS_STEERING_THRESHOLD = 15

    # Penalize reward if the car is steering too much
    if abs_steering > ABS_STEERING_THRESHOLD:
        reward *= 0.8

    return float(reward)
```

Example 4: Stay in one lane without crashing into stationary obstacles or moving vehicles

This reward function rewards the agent for staying inside the track's borders and penalizes the agent for getting too close to objects in front of it. The agent can move from lane to lane to avoid crashes. The total reward is a weighted sum of the reward and penalty. The example gives more weight to the penalty in effort to avoid crashes. Experiment with different averaging weights to train for different behavior outcomes.

```
import math
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside two borders
    and penalizing getting too close to the objects in front
    '''
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    objects_location = params['objects_location']
    agent_x = params['x']
    agent_y = params['y']
    _, next_object_index = params['closest_objects']
    objects_left_of_center = params['objects_left_of_center']
    is_left_of_center = params['is_left_of_center']
    # Initialize reward with a small number but not zero
    # because zero means off-track or crashed
    reward = 1e-3
```

```
    # Reward if the agent stays inside the two borders of the track
    if all_wheels_on_track and (0.5 * track_width - distance_from_center) >= 0.05:
        reward_lane = 1.0
    else:
        reward_lane = 1e-3
    # Penalize if the agent is too close to the next object
    reward_avoid = 1.0
    # Distance to the next object
    next_object_loc = objects_location[next_object_index]
    distance_closest_object = math.sqrt((agent_x - next_object_loc[0])**2 + (agent_y -
next_object_loc[1])**2)
    # Decide if the agent and the next object is on the same lane
    is_same_lane = objects_left_of_center[next_object_index] == is_left_of_center
    if is_same_lane:
        if 0.5 <= distance_closest_object < 0.8:
            reward_avoid *= 0.5
        elif 0.3 <= distance_closest_object < 0.5:
            reward_avoid *= 0.2
        elif distance_closest_object < 0.3:
            reward_avoid = 1e-3  # Likely crashed
    # Calculate reward by putting different weights on
    # the two aspects above
    reward += 1.0 * reward_lane + 4.0 * reward_avoid
    return reward
```

Import and export models in the AWS DeepRacer console

There are scenarios in which you might need to import or export an AWS DeepRacer model. Racers who participated in an employer-sponsored event can export their models to avoid losing access to them, race administrators can provide pre-trained models for attendees to import and use during the event. Use the **Your models** page to import and export AWS DeepRacer models in the console.

Topics

Copy your AWS DeepRacer model to Amazon S3

To copy an AWS DeepRacer model to Amazon S3

1. Log in to the AWS DeepRacer console.
2. In **Reinforcement learning** on the navigation pane, choose **Your models**.
3. Select the model you want to import by selecting the check box next to the model's name. You can only copy one model to Amazon S3 from the console at a time.
4. Choose the **Actions** button dropdown, then choose **Copy to S3**.

   A new **Copy to Amazon S3** page opens.
5. On the Copy to Amazon S3 page, use the Amazon S3 bucket dropdown selector to select an Amazon S3 bucket to export the model to. AWS DeepRacer S3 buckets must include deepacer in the name.

   - If you don't have a valid Amazon S3 bucket, create one by choosing **Create a new bucket**. The dropdown selector will populate a bucket name with the following format `aws-deepracer-assets-XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX`.
6. Add an optional folder prefix to the Amazon S3 object in the S3 object prefix field.
7. After you have set up the S3 bucket, select which assets you want to include. You must select at least one asset type to proceed.

   - **Model:** The model folder contains all required files for a model import.

- **Logs:** Copies the training and evaluation logs for the model. This option includes the `logs/`, `metrics/`, and `sim-trace/` folders.
- **Video:** This option copies the videos folder to your Amazon S3 bucket. The videos folder contains the `evaluation/` and `training/` folders. These folders include videos from the top view, a 45 degree angle view, and a 45 degree angle view with the console overlay showing the location of the car on the track.

8. After you press **Copy**, a pop-up informing you that you are responsible for the Amazon S3 data storage costs appears. If you agree with the terms, press the **Copy** button on the pop-up.
9. Once the copy process starts, you will be taken back to the **Your models** page in the console. A banner on the top of the page will show the current status. When the export process is done the banner will confirm the successful export.

Required files for model import

To upload a model folder for a model trained outside the console, follow the steps on the Uploading objects page of the Amazon S3 documentation. The following table contains a list of the files required for the model import. If any of the required files are missing, the model import will fail.

Models trained in the AWS DeepRacer console have the folder name format `DAY/MONTH/YEAR/TIME GMT`. Our example model was exported on November 30, 2023 and the folder name is `Thu, 30 Nov 2023 19:01:24 GMT`. In this example, we refer to this folder as **root**.

Required files for model imports

File name	Folder path	Description
.coach_checkpoint	root/model/	The coach checkpoint file contains the key for the model checkpoint used in the import.
ckpt files	root/model/	Checkpoint files are snapshots of the model weights taken at different stages during training. They include the `ckpt.inde`

File name	Folder path	Description
		`x ,ckpt.data ,`and `ckpt.meta` files.
model_metadata.json	root/	The model metadata file contains settings that include action space definitions, sensor configuration, and the training algorithm selection.
reward_function.py	root/	A python file that contains the reward function used to train the model.

Metrics files are not necessary for importing your model. If these files are not included, the training metrics and reward graph for the model will not be available on the console.

Optional files for model imports

File name	Folder path	Description
training_params.yaml	root/	The training_params file contains training job data that includes track and vehicle information, racer and model names, and folder paths for training artifacts.
hyperparameters.json	root/ip/	Contains the model's hyperparameter information such as batch size, loss type, learning rate, and number of epochs.
training-*.json	root/metrics/training/	Used to visualize the model's training metrics in the AWS DeepRacer console.

Import your AWS DeepRacer model to the console

This section walks you through the process of importing an AWS DeepRacer model to the console. Before you can import a model, you need to copy the Amazon S3 URL for the model folder.

Copy the AWS DeepRacer Amazon S3 bucket URL

1. Log in to the Amazon S3 console and go to the **Buckets** page.
2. Select the Amazon S3 bucket you created for your AWS DeepRacer model by pressing the link on the name of the bucket. The format of S3 buckets created in the AWS DeepRacer console is `aws-deepracer-assets-XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX`.
3. From the **Objects** tab of the AWS DeepRacer bucket:

   a. Select the model you want to import by pressing the model object's name link in the **Name** field.

   A list of subfolders appears.

   b. Select the root of the model folder by choosing the checkbox next to the folder name. Models trained in the AWS DeepRacer console have the folder name format `DAY/MONTH/YEAR/TIME GMT`.
4. Once you navigate to the root of the model folder, select the **Copy S3 URL** button. The Copy s3 URL, Copy URL, Open, and Delete buttons will be grayed out until you select the checkbox next to the model name.

> **Note**
>
> If you added a prefix during the Copy to S3 process, such as *`my_model/version_2`*, the path of your model folder is `deep_racer_bucket/model_name/`*`my_model/version_2`*`/root/`.

Import your model to the AWS DeepRacer console

1. On the AWS DeepRacer console, go to the **Your models** page.
2. Select the **Import Model** button on the models container.

   The import model page will appear.

3. On the Import section:

   - Enter the Amazon S3 URL for the model folder you want to import. The Amazon S3 URL has the format `s3://deep_racer_bucket/model_name/prefix/root`.

4. On the Details section:

   a. Enter the model name.

   b. Add an optional description for the model.

   c. If you're using an administrator account using multi-user mode, choose the user you're importing the model for from the dropdown selector.

5. Select the **Import** button on the bottom of the screen.

6. Once the import process starts, you will be taken back to the **Your models** page in the console. A banner on the top of the page will show the current status, and the model will appear in your models list with *Importing..* as its status. When the import process is done, the banner will confirm the successful import and the status of your model will change from *Importing..* to *Ready*.

Troubleshooting

Model copy error

We couldn't copy your model despite making several attempts. If the model is still in your S3 bucket, retry the model import by selecting the model from the **Model errors** table, and choosing **Update**, then choosing **Import**. Or, If you have a local copy of the model, you can manually import it by following the steps on the Uploading objects page of the Amazon S3 documentation.

The Amazon S3 bucket doesn't exist

We couldn't copy the model because the S3 bucket where this model was stored has been deleted. If you have a copy of the model, place it in an S3 bucket with `deepracer` in its name, and try to import again following the steps in the Import your AWS DeepRacer model to the console section.

Can't access the Amazon S3 bucket

The permissions for the Amazon S3 bucket where this model is stored have changed, so we couldn't copy the model. This can happen for two reasons, you directly edited the permissions on the AWS DeepRacer S3 or the AWS DeepRacer service role policy. If you directly edited the

permissions on your AWS DeepRacer S3 bucket, restore the bucket permissions by following the steps in the Adding a bucket policy by using the Amazon S3 console page using the following policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1586917903457",
            "Effect": "Allow",
            "Principal": {
                "Service": "deepracer.amazonaws.com"
                },
            "Action": [
                "s3:GetObjectAcl",
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
               "arn:aws:s3:::your-bucket-name",
               "arn:aws:s3:::your-bucket-name/*"
            ]
        }
    ]
}
```

To import the model after restoring the bucket permissions, select the model from the **Model errors** table and choose **Update**. When the **Model import** page appears, choose **Import**.

Model file doesn’t exist

We can't copy the model because it's been deleted from the Amazon S3 bucket. If you still have the file, try to restore it to your AWS DeepRacer bucket, then select the model from the **Model errors** table and choose **Update**. When the **Model import** page appears, choose **Import**. If you have a local copy of the model, you can manually import the files by following the steps on the Uploading objects page of the Amazon S3 documentation.

Coach file doesn’t exist

We can't copy the model because the coach checkpoint metadata has been deleted from the Amazon S3 bucket. If you still have the file, try to restore it to your AWS DeepRacer bucket, then select the model from the **Model errors** table and choose **Update**. When the **Model import** page appears, choose **Import**. If you have a local copy of the model, you can manually import the files by following the steps on the Uploading objects page of the Amazon S3 documentation.

Checkpoint file doesn’t exist

We can't copy the model because the checkpoint files have been deleted from the Amazon S3 bucket. If you still have the files, try to restore them to your AWS DeepRacer bucket, then select the model from the **Model errors** table and choose **Update**. When the **Model import** page appears, choose **Import**. If you have a local copy of the files, you can manually import them by following the steps on the Uploading objects page of the Amazon S3 documentation.

Model file too large

Your model file exceeds the 1 GB file size limit that the service can create, so your file was edited. This model will not be imported. To remove this message, select the model from the **Model errors** table, and choose **Delete**.

Checkpoint file too large

Your checkpoint file exceeds the 1 GB file size limit that the service can create, so your file was edited. This model will not be imported. To remove this message, select the model from the **Model errors** table, and choose **Delete**.

Metadata file too large

Your YAML file exceeds the 10 MB file size limit that the service can create, so your file was edited. This model will not be imported. To remove this message, select the model from the **Model errors** table, and choose **Delete**.

Model not valid

We can't validate your model because it's been edited. If you have a copy of the model, try to replace it in your AWS DeepRacer S3 bucket, then select the model from the **Model errors** table and choose **Update**. When the **Model import** page appears, choose **Import**.

Missing or incorrect permissions

We couldn't copy the model because the permissions that were available with AWS DeepRacer when you trained it have been removed. To authorize AWS DeepRacer to recreate the required permissions, choose the model from the **Model errors** table and then choose **Update**. When the **Model import** page appears, choose **Import**. AWS DeepRacer will recreate the permissions, then copy the model.

Operate your AWS DeepRacer vehicle

After you finish training and evaluating an AWS DeepRacer model in the AWS DeepRacer simulator, you can deploy the model to your AWS DeepRacer vehicle. You can set the vehicle to drive on a track and evaluate the model's performance in a physical environment. This mimics a real-world autonomous race.

Before driving your vehicle for the first time, you must set up the vehicle, install software updates, and calibrate its drive-chain sub-system.

To drive your vehicle on a physical track, you must have a track. For more information, see *Build your physical track*

Topics

Get to know your AWS DeepRacer vehicle

Your AWS DeepRacer vehicle is a machine learning-enabled, battery-powered, and Wi-Fi-connected 1/18th-scale model four-wheel drive car with a front-mounted 4-megapixel camera and an Ubuntu-based compute module.

The vehicle can drive autonomously by running inference that is based on a reinforcement learning model in its compute module. You can also drive the vehicle manually, without deploying any reinforcement learning model. If you have not already obtained an AWS DeepRacer vehicle, you can order one here.

The AWS DeepRacer vehicle is powered by a brushed motor. The driving speed is controlled by a voltage regulator that controls how fast the motor spins. The servomechanism (servo) that operates the steering system is protected by the black cover in the AWS DeepRacer vehicle chassis.

Topics

Inspect your AWS DeepRacer vehicle

When you open your AWS DeepRacer vehicle box, you should find the following components and accessories:

Components	Comments
Vehicle Chassis [1]	Includes a front-mounted camera for capturing vehicle driving experiences and the compute module for autonomous driving. You can view images captured by the camera as a streaming video on the vehicle's device console. The chassis includes a brushed electric motor, an electronic speed controller (ESC), and a servomechanism (servo)
Vehicle body shell [2]	Remove this when setting up the vehicle.
Micro-USB to USB-A cable [3]	Use this to support USB-OTG functionality.
Compute battery [4]	Use this to power the compute module that runs inference on a downloaded AWS DeepRacer reinforcement learning model.
Compute battery connector cable [5]	Use this USB-C to USB-C cable to connect the compute module with the battery. If you have a Dell compute battery, this cable will be longer.
Power cable [6a]	Use this to connect the power adaptor to a power outlet.
Power adapter[6b]	Use this to charge the compute battery and the compute module.
Pins (spare parts) [7]	Use to fasten the compute module to the vehicle chassis. These are extras.
Vehicle battery [8]	A 7.4v LiPo battery pack to power the motor.
Vehicle battery charge adapter [9a]	Use this to charge the vehicle battery that powers the vehicle drive chain.
Vehicle battery charge cable [9b]	Use this to connect the vehicle battery charger to a power outlet.

Components	Comments
Battery unlock cable [10]	Use this if your battery enters lock out state.

To set up your AWS DeepRacer vehicle, you must also have the following items ready:

- A computer with a USB port and access to the internet.
- A Wi-Fi network connected to the internet.
- An AWS account.

Now follow the instructions in the next section to make sure your vehicle battery and the power bank are charged.

Charge and install your AWS DeepRacer batteries

Your AWS DeepRacer vehicle has two power sources: the vehicle battery and the compute module power bank.

The power bank keeps the compute module running. The compute module maintains the Wi-Fi connection, runs inference against a deployed AWS DeepRacer model, and issues a command for the vehicle to take an action.

The vehicle battery powers the motor to move the vehicle. It has two sets of cables. The two-wired set of the red and black cables is used to connect to the vehicle's ESC and the triple-wired blue (or black), white and red cables is to connect to the charger. For driving, only the two-wired cable set should be connected to the vehicle.

After fully charged, the battery voltage will drop as the batteries discharge. When the voltage drops, the available torque also drops. As a consequence, the same speed setting will result in slower speed on the track. When the battery is fully empty, the vehicle stops moving. For autonomous driving under normal conditions, the battery usually lasts 15-25 minutes. To ensure consistent behavior, it is recommended that you charge the battery after every 15 minutes of use.

To install and charge the vehicle battery and the power bank, follow the steps below.

1. Remove your AWS DeepRacer vehicle shell.
2. Remove the four vehicle chassis pins. Carefully lift vehicle chassis while keeping wires connected.

3. To charge and install the vehicle battery, do the following:

   a. To charge the battery, plug the three-wired cable set from the batter to the charger to connect the battery to the power adapter and then plug the power adapter to a wall outlet or to a USB port if a USB cable is used to charge the battery.

      For a graphical illustration of how to charge the vehicle battery using the enclosed charger, see the section called “How to charge the vehicle's drive module battery”.

   b. After the battery is charged, plug the two-wired cable set of the vehicle battery cable into the black and red cable connector on your vehicle.

   c. To secure the vehicle battery, tie the battery under the vehicle chassis with the attached straps.

      Make sure to keep all the cables inside the vehicle.

   d. To check if the vehicle battery is charged, do the following:

      i. Slide the vehicle power switch to turn on the vehicle.

      ii. Listen for two short beeps.

         If you don't hear the beeps, the vehicle is not charged. Remove the battery from the vehicle and repeat Step 1 above to recharge the battery.

      iii. When not using the vehicle, slide the vehicle power switch back to turn off the vehicle battery.

4. To check the power bank charging level, do the following:

   a. Press the power button on the power bank.

   b. Check the four LED lights next to the power button to determine the charging level.

      If all the four LED lights are lit, the power bank is fully changed. If none of the LED lights are lit, the power bank needs to be charged.

   c. To charge the power bank, insert the USB C plug from the power adapter into the USB C port of the power bank. It takes some time for the power bank to be fully charged. When it is charged, repeat **Step 4** to confirm that the power bank is fully charged.

5. To install the power bank, do the following:

a. Insert the power bank into its holder with the power button and USB C port facing the back of the vehicle.

b. Use the strap to tie the power bank to the vehicle chassis securely.

> **Note**
>
> Do not connect the power bank to the compute module in this step.

Test your AWS DeepRacer compute module

Test the compute module to verify that it can be started successfully. To test the module by using an external power source, follow the steps below:

To test your vehicle's compute module

1. Connect the compute module to a power source. Connect the power cord to the power adapter, plug the power cord to a power outlet, and insert the power adapter's USB C plug into the USB C port on the compute module.
2. Turn on the vehicle's compute module by pressing the power button on the compute module.
3. To verify the compute module's status, check that the LED lights are shown as follows:

   - Solid blue

     The compute module is started, connected to the specified Wi-Fi, and ready to go.

     In this state, you can log in to the compute module after you attach it to a monitor using an HDMI cable, a USB mouse and a USB keyboard. For the first-time login, use `deepracer` for both the **username** and **password**. You will then be asked to reset the password for future logins. For security reasons, choose a strong password phrase for the new password.

   - Blinking red

     The compute module is in setup mode.

   - Solid yellow

     The compute module is initializing.

   - Solid red

The compute module failed to connect to the Wi-Fi network.

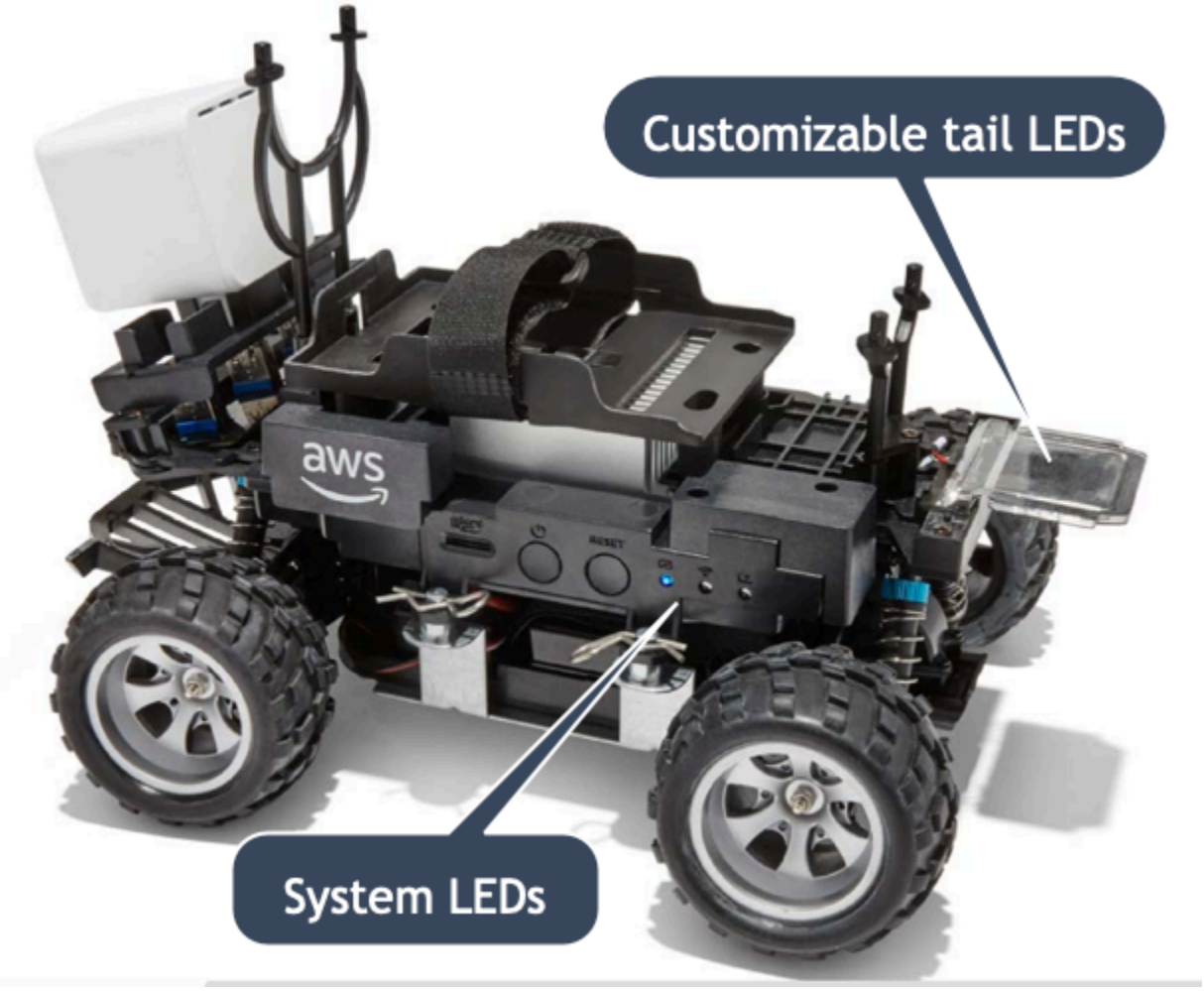
4. When you're done with the test, press the power button on the compute module to turn it off and then unplug it from the external power source.

Turn off your AWS DeepRacer vehicle

To turn off your AWS DeepRacer vehicle, unplug the vehicle from the external power source. You can also press the power button on the device until power indicator is off.

AWS DeepRacer vehicle LED indicators

Your AWS DeepRacer vehicle has two sets of LED indicators for the vehicle status and for customizable visual identification of your vehicle, respectively.



The details are discussed as follows.

Topics

AWS DeepRacer vehicle system LED indicators

The AWS DeepRacer vehicle system LED indicators are located on the left side of the vehicle chassis when the vehicle is in the forward position in front of you.

The three system LEDs are positioned after the **RESET** button. The first LED (on the left side of your field of view) shows the status of the system power. The second (middle) LED is reserved for future use. The last (right) LED shows the status of the Wi-Fi connection.

LED type	Color	Status
Power	Off	There is no power supply.
	Blinking yellow	BIOS and OS are being loaded.
	Steady yellow	OS is loaded.
	Steady blue	An application is running.
	Blinking blue	A software update is in progress.
	Steady red	An error is encountered while the system is being booted or an application is being started.
Wi-Fi	Off	There is no Wi-Fi connection.
	Blinking blue	The vehicle is connecting to the Wi-Fi network.
	Steady red for 2 seconds and then off	The Wi-Fi connection failed.

LED type	Color	Status
	Steady blue	The Wi-Fi connection is established.

AWS DeepRacer vehicle identification LEDs

The AWS DeepRacer vehicle custom LEDs are located at the tail of the vehicle. They're used to help identifying your vehicle in races when multiple vehicles are present. You can use the AWS DeepRacer device console to set them a supported color of your choosing.

AWS DeepRacer device spare parts

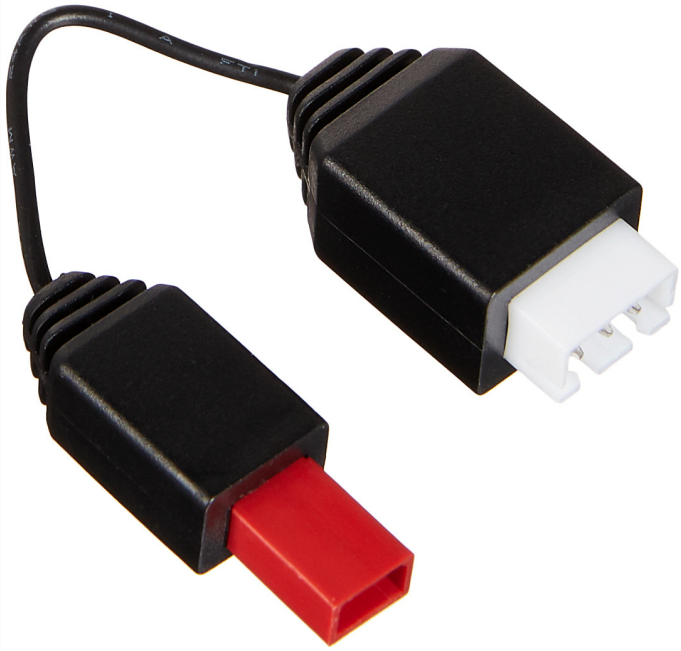
> **Note**
>
> The AWS DeepRacer device uses the WLToys A949 and A979 Remote Control (RC) car chassis. To browse an up to date list of available parts for your AWS DeepRacer device, visit the AWS DeepRacer storefront.

Spare AWS DeepRacer device parts

Part	Name
	Spare compute battery

Part	Name
	Spare compute battery

Part	Name
	Spare compute battery

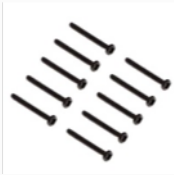
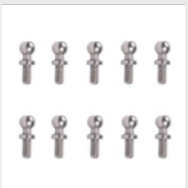
Part	Name
	Lithium battery 7.4V 1100mAh
	AWS DeepRacer Car Battery Unlock Cable
	Tire
	Front bumper

Part	Name
	Suspension arm
	Pull rod
	C style seat
	Transmission shaft
	Round head screw, M2x17.5mm
	Chassis car bottom
	Turning seat

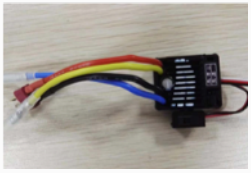
Part	Name
	Rear suspension frame
	Metal hexagonal combiner set
	Gear box shell
	Differential box case
	Differential drive cup
	Front rear shelter
	Servo seat

Part	Name
	Central driving shaft
	Shock frame
	Servo arm
	Differential mechanism
	Reduction gear
	Motor base
	17g steering engine

Part	Name
	Screw gasket of motor, fixed seat
	390 motor
	Hexagon connector 4x8x3mm
	Hexagon connector 8x12x3.5mm
	Ball bearing 7x11x3mm
	Ball bearing 8x12x3.5mm
	Middle axle disc plate

Part	Name
	Screw 2.6x6mm
	Screw 2x7mm
	Screw 2.5x8mm
	Screw 2x16mm
	Screw 2.5x6x6mm
	Screw M3x5mm
	Ball screw 10.8x4mm

Part	Name
	Screw 2x6mm
	Screw 2x9.5mm
	M3 locknut
	Axle hinge pin
	Drive shaft
	Swing arm pin
	Screw 2*29KM

Part	Name
	Hair pin
	Front shock absorber
	Charger
	Metal motor pinion gear
	Back shock absorber
	ESC

Choose a Wi-Fi network for your AWS DeepRacer vehicle

The first time you open your AWS DeepRacer vehicle, you must set it up to connect to a Wi-Fi network. Complete this setup to get the vehicle's software updated and to get the IP address to access the vehicle's device console.

This section walks you through the steps to perform the following tasks:

- Connect your laptop or desktop computer to your vehicle.
- Set up the vehicle's Wi-Fi connection.
- Update the vehicle's software.
- Get the vehicle's IP address.
- Test drive the vehicle.

Use a laptop or desktop computer to perform the setup tasks. We'll refer to this setup computer as your computer, to avoid possible confusion with the vehicle's compute module, which is running the Ubuntu operating system.

After the initial setup of the Wi-Fi connection, you can follow the same instructions to choose a different Wi-Fi network.

> **Note**
>
> AWS DeepRacer does not support Wi-Fi network that requires active captcha verification for use sign-in.

Topics

Get ready to set up Wi-Fi connection for your AWS DeepRacer vehicle

To set up your vehicle's Wi-Fi connection, connect your a laptop or desktop computer to your vehicle's compute module using the included *USB-to-USB C* cable.

To connect your computer to your vehicle's compute module, follow the steps below.

1. Make sure your computer is disconnected from Wi-Fi before connecting your device.
2. Insert the USB end of the *USB-to-USB C* cable into your computer's USB port.
3. Insert the cable's USB C end into your vehicle's USB C port.

You're now ready to proceed to setting up your vehicle's Wi-Fi connection.

Set up Wi-Fi connection and update your AWS DeepRacer vehicle's software

Before you follow the steps here to set up the Wi-Fi connection, be sure you complete the steps in the section called “ Get ready to set up Wi-Fi ”.

1. Look at the bottom of your vehicle and make note of the password printed under **Host name**. You'll need it to log in to the device control console to perform the setup.
2. On your computer, go to `https://deepracer.aws` to launch the device control console of your vehicle.
3. When prompted with a message that the connection is not private or secure, do one of the following.

    a. In Chrome, choose **Avanced** and then choose **Proceed to *<device_console_ip_address>* (unsafe)**.

    b. In Safari, choose **Details**, follow the **visit this website** link, and the choose **Visit Websites**. If prompted for your password to update the certificate trust settings, type the password and then choose **Update settings**.

    c. In Opera, choose **Continue Anyway** when warned of an invalid certificate.

    d. In Edge, choose **Details** and then choose **Go on to the webpage (Note recommended)**.

    e. In Firefox, choose **Advanced**, choose **Add Exception**, and then choose **Confirm Security Exception**.
4. Under **Unlock your AWS DeepRacer vehicle**, enter the password noted in **Step 1** and then choose **Access vehicle**.
5. On the **Connect your vehicle to your Wi-Fi network** pane, choose your Wi-Fi network name from the **Wi-Fi network name (SSID)** drop-down menu, type the password of your Wi-Fi network under **Wi-Fi password**, and choose **Connect**.

6. Wait until the Wi-Fi connection status changes from **Connecting to Wi-Fi network...** to **Connected**. Then, choose **Next**.
7. On the **Software update** pane, if a software update is required, turn on the vehicle's compute module, with the included power cord and power adapter, and then choose **Install software update**.

   Powering the vehicle with an external power source helps avoid interruption of the software update if the compute module's power bank become discharged.
8. Wait until the software update status changes from **Installing software update** to **Software update installed successfully**.
9. Note the IP address shown under **Wi-Fi network details**. You'll need it to open the vehicle's device control console after the initial setup and any subsequent modification of the Wi-Fi network settings.

Launch the AWS DeepRacer vehicle's device console

After you set up the vehicle's Wi-Fi connection and install required software updates, you should open the device console to verify if the vehicle's network connection is working. Subsequently, you can launch the device console to inspect, calibrate and manage the vehicle's other settings. The process involves signing in to your vehicle's device console using the IP address of your vehicle.

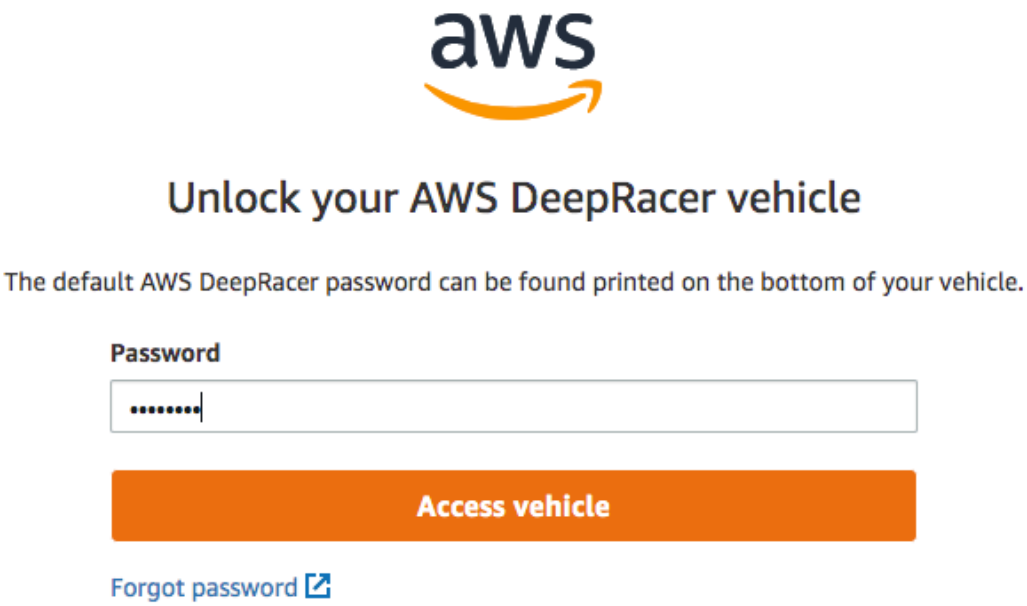
The device control console is hosted on the vehicle and is accessed with the IP address you obtained at the end of the Wi-Fi setup section.

To access the device console of your AWS DeepRacer vehicle through the Wi-Fi connection

1. To access the device console of your vehicle, open a web browser on your computer, tablet or a smart phone and type your vehicle's IP address into the address bar.

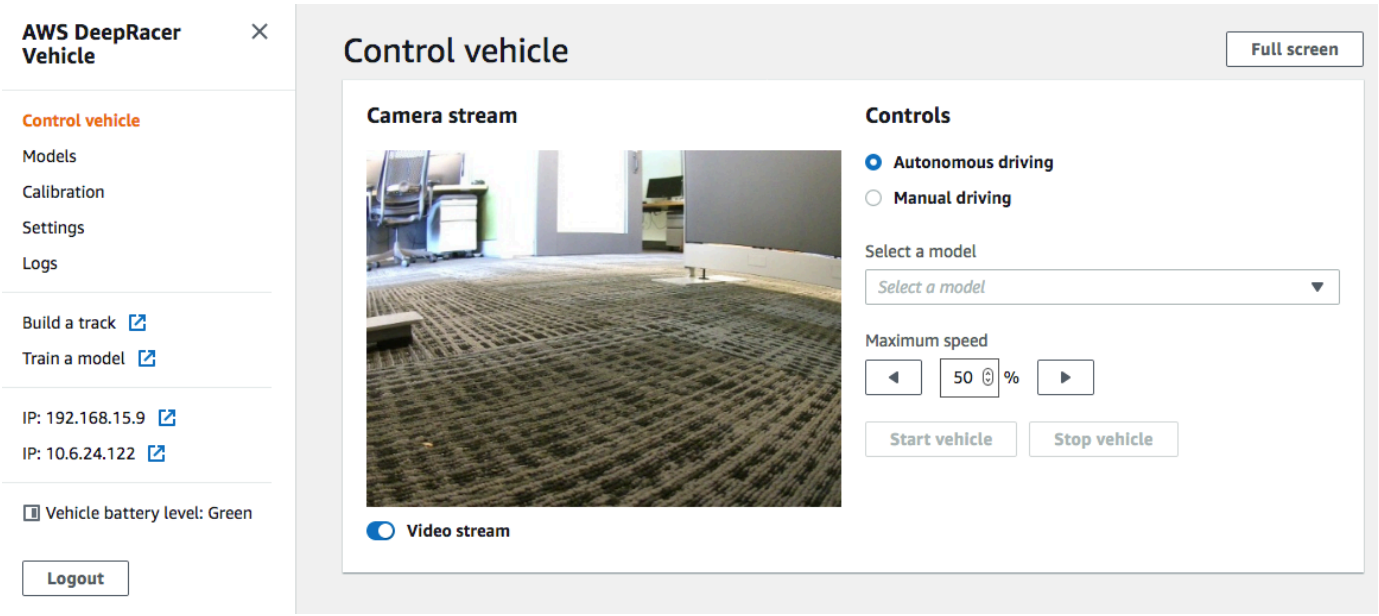
   You can get this IP address when setting up the vehicle's Wi-Fi connection . For illustration, we use `10.92.206.61` as an example.

   If you are prompted with a warning that the connection is not private or secure, ignore the message and continue to connect to the device console.
2. Under **Unlock your AWS DeepRacer vehicle**, type the device console's password in **Password** and then choose **Access vehicle**.



You can find the default password printed on the bottom of your vehicle (under **Host Name**).

3. When you are successfully signed in, you see the device console's home page as follows.



You're now ready to calibrate and operate your vehicle. If this is your first time operating the vehicle, proceed to calibrating the vehicle now.

Calibrate your AWS DeepRacer vehicle

To achieve the best performance, it's essential that you calibrate some physical parts of your AWS DeepRacer vehicle. If you use an uncalibrated vehicle, it can add uncertainty when testing your model. If the vehicle's performance is not optimal, you might be tempted to only adjust the deep learning model code. However, you won't be able to improve the vehicle performance if the root cause is mechanical. Adjust the mechanics by calibration.

To calibrate your AWS DeepRacer vehicle, set the duty cycle range for the vehicle's electronic control system (ECS) and its servomechanism (servo), respectively. Both the servo and ECS accept pulse-width modulation (PWM)) signals as control input from the vehicle's compute module. The compute module adjusts both of the vehicle's speed and steering angle by changing the duty cycles of the PWM signals.

The maximum speed and steering angle defines the span of the action space. You can specify the maximum speed and maximum steering angle during training in simulation. When deploying the trained model to your AWS DeepRacer vehicle for driving on a real-world track, the maximum speed and steering angle of the vehicle must be calibrated to match those used in the simulation training.

To ensure that the real-world experiences match the simulated experiences, you should calibrate your vehicle to match the maximum speed and maximum steering angles between the simulation and the real world. In general, there are two ways to do this calibration:

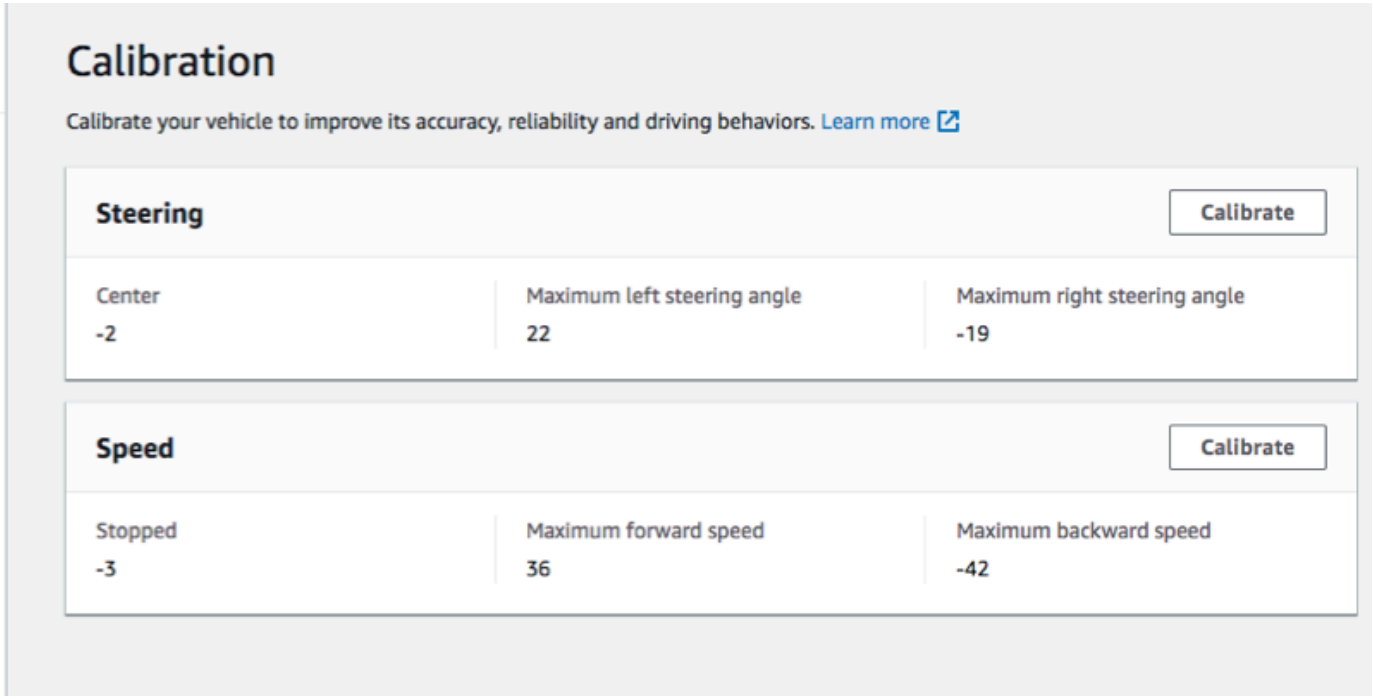
- Define the action space in training and calibrate the physical vehicle to match the settings.
- Measure the actual performance of your vehicle and change the settings of the action space in the simulation.

A robust model can handle certain differences between the simulation and the real world. However, you should experiment with either approach and iterate to find the best results.

Before starting the calibration, turn on the compute module. After it's started and the power LED has turned solid blue, turn on the vehicle battery. After you hear two short beeps and one long beep, you're ready to proceed with the calibration.

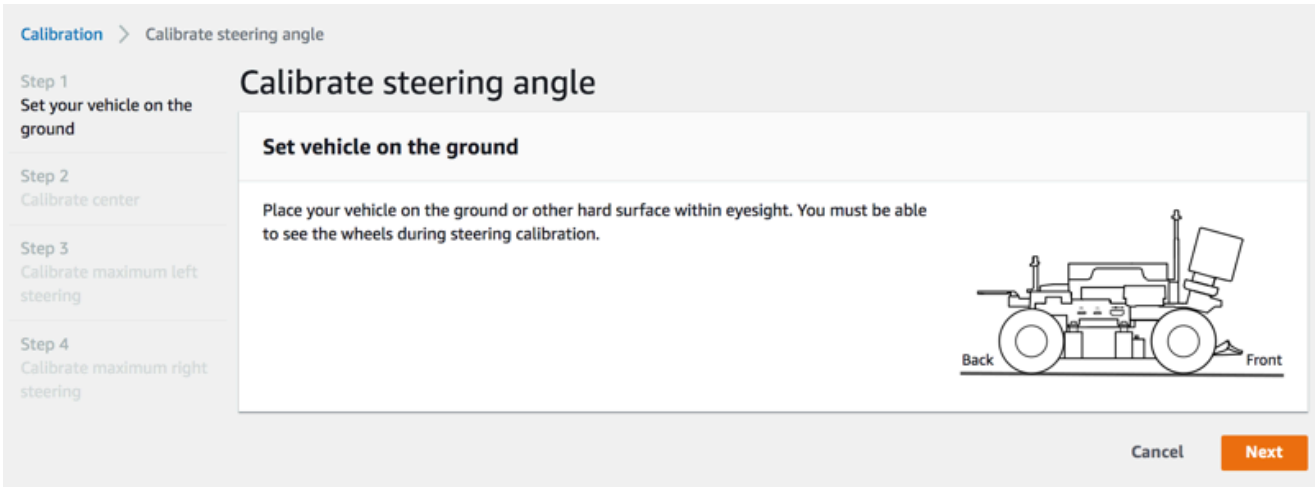
To calibrate your AWS DeepRacer vehicle to match the training settings:

1. Follow these instructions to access your vehicle and open the device control console.
2. Choose **Calibration** from the main navigation pane.



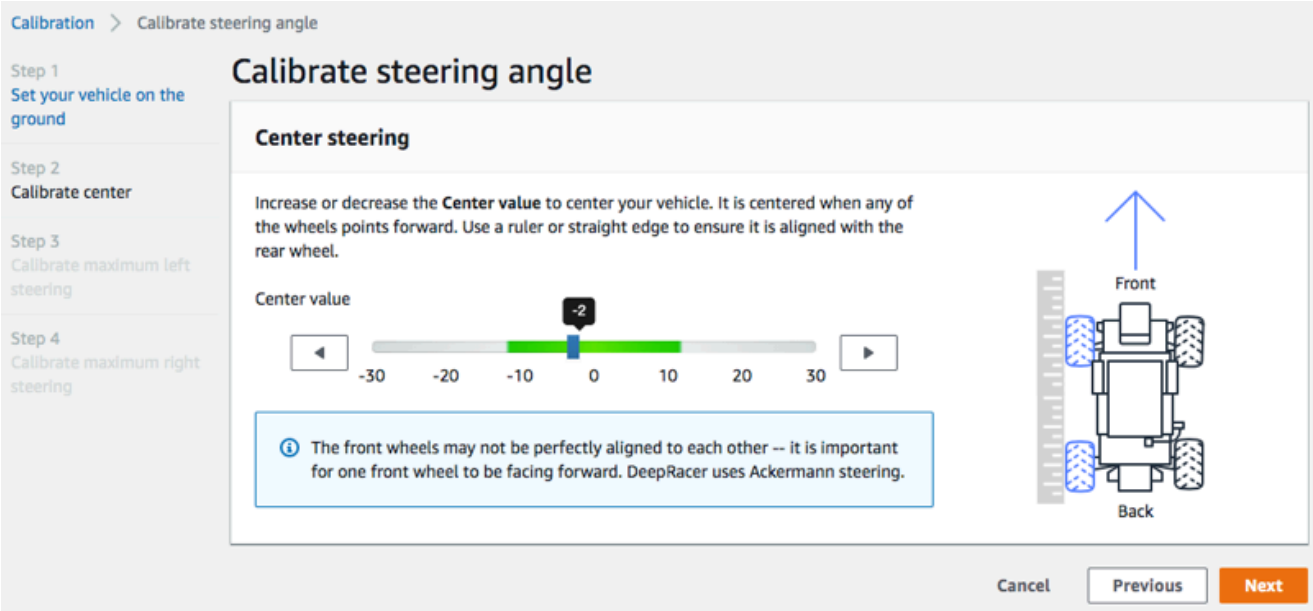
3. On the **Calibration** page, choose **Calibrate** in **Steering** and then follow the steps below to calibrate the vehicle's maximum steering angles.

   a. Set the vehicle on the ground or another hard surface where you can see the wheels during the steering calibration. Choose **Next**.



Steering a vehicle on a track requires the much smaller steering angles than turning wheels in the air. To measure the actual steering angles of the wheels, it's important that you place the vehicle down on the track surface.

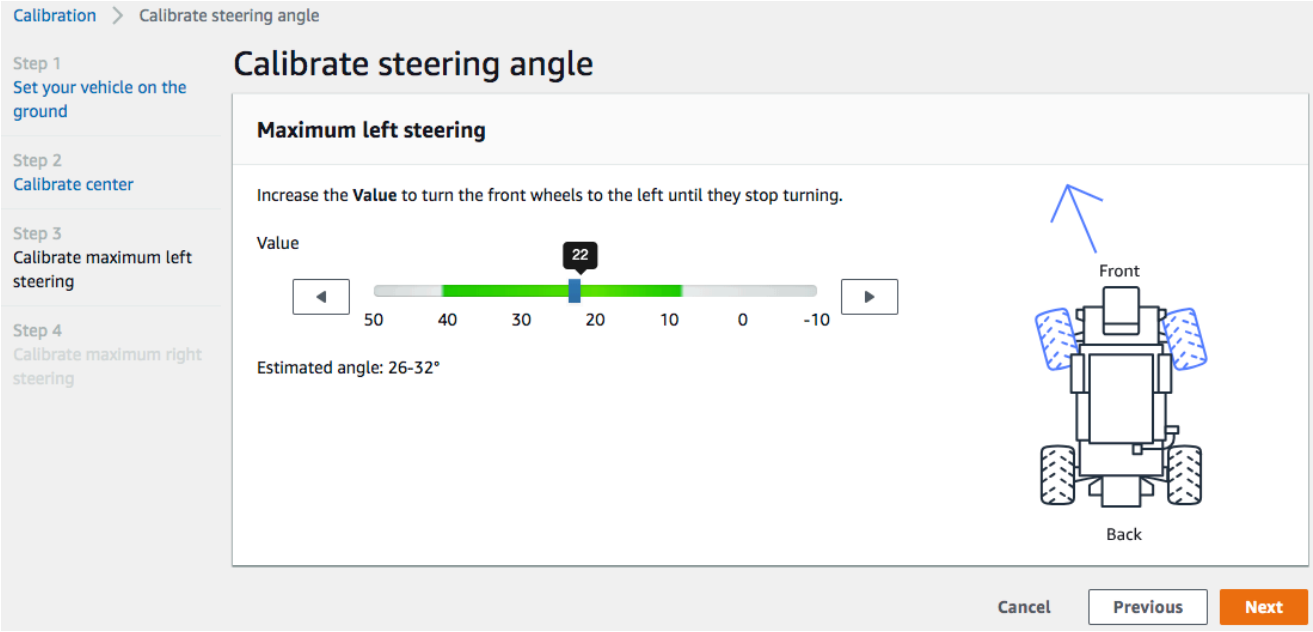
b. Under **Center steering**, gradually move the slider or press the left or right arrow to the position where at least one of the front wheels is aligned with the rear wheel on the same side. Choose **Next**.



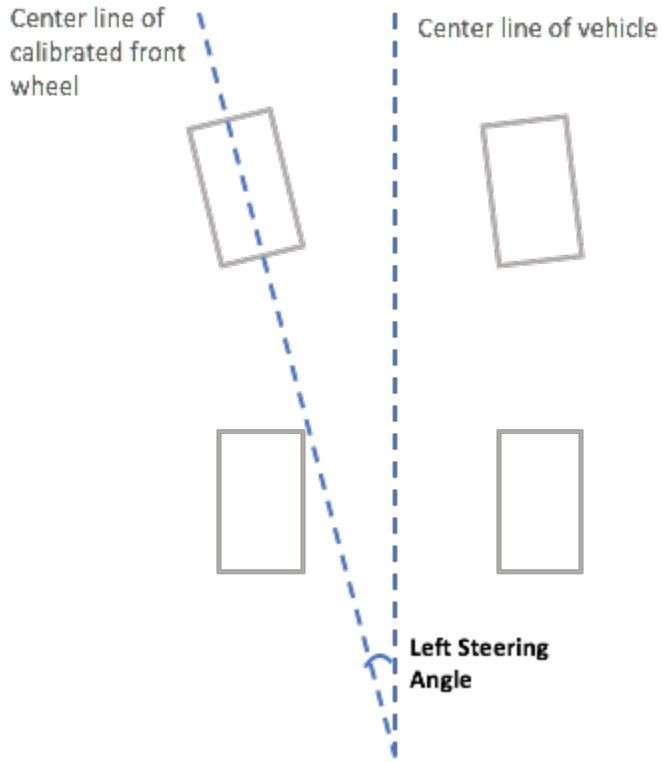
AWS DeepRacer uses Ackermann front-wheel steering to turn wheels on the inside and outside of a turn. This mean that the left and right front wheels generally turn at different angles. In AWS DeepRacer, the calibration is done on the center value. Therefore, you need to adjust the wheels on the selected side to be aligned in a straight line.

> **Note**
>
> Make sure to calibrate your AWS DeepRacer vehicle well so that it can maintain center steering as straight as possible. You can test this by manually pushing the vehicle to verify it follows a straight path.

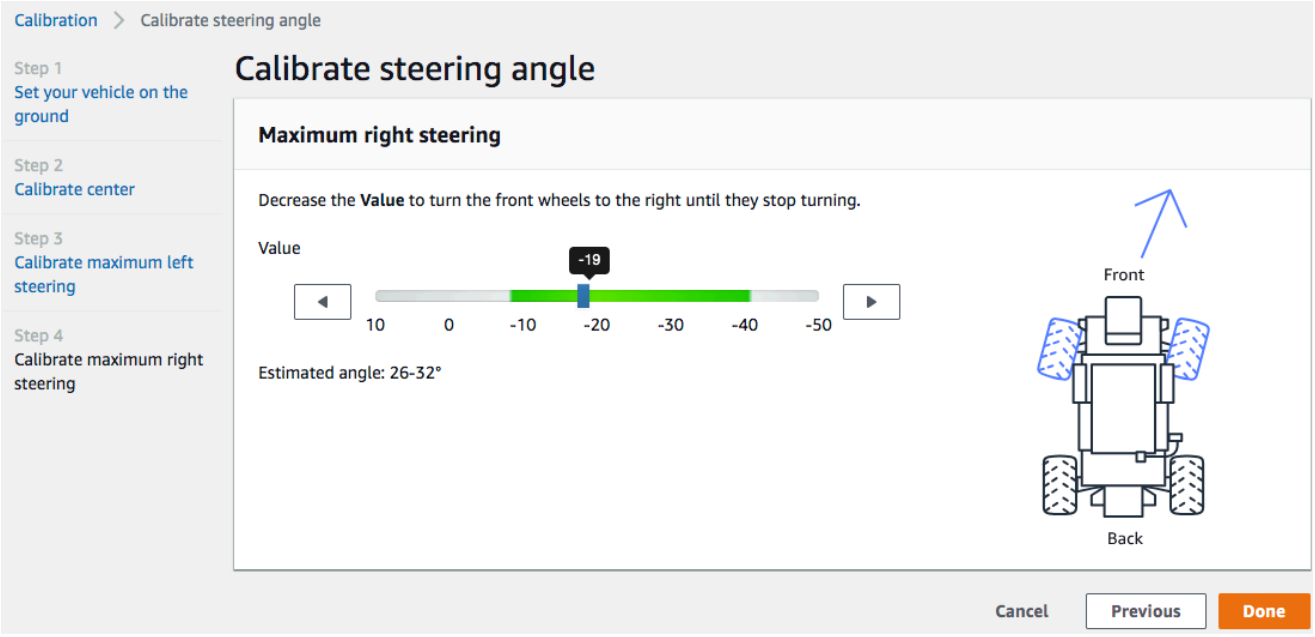
c. Under **Maximum left steering**, gradually move the slider to the left or press the left arrow until the vehicle front wheels stops turning left. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum left steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Next**.



To measure the actual maximum left steering angle, draw a center line for the vehicle, mark the two edge points of the selected front wheel for calibration, and draw the center line of this front wheel until it crosses over the center line of the vehicle. Use a protractor to measure the angle. See the figure below. If you want to match the actual angle in your training, you can set the same value in the action space in your next training job.



d. Under **Maximum right steering**, gradually move the slider to the right until the selected front wheels stop turning right. There will be a quiet noise. If you hear a loud noise, you have gone too far. The position corresponds to the maximum right steering angle. If you have limited your steering angle in the simulated action space, match the corresponding value here. Choose **Done**.

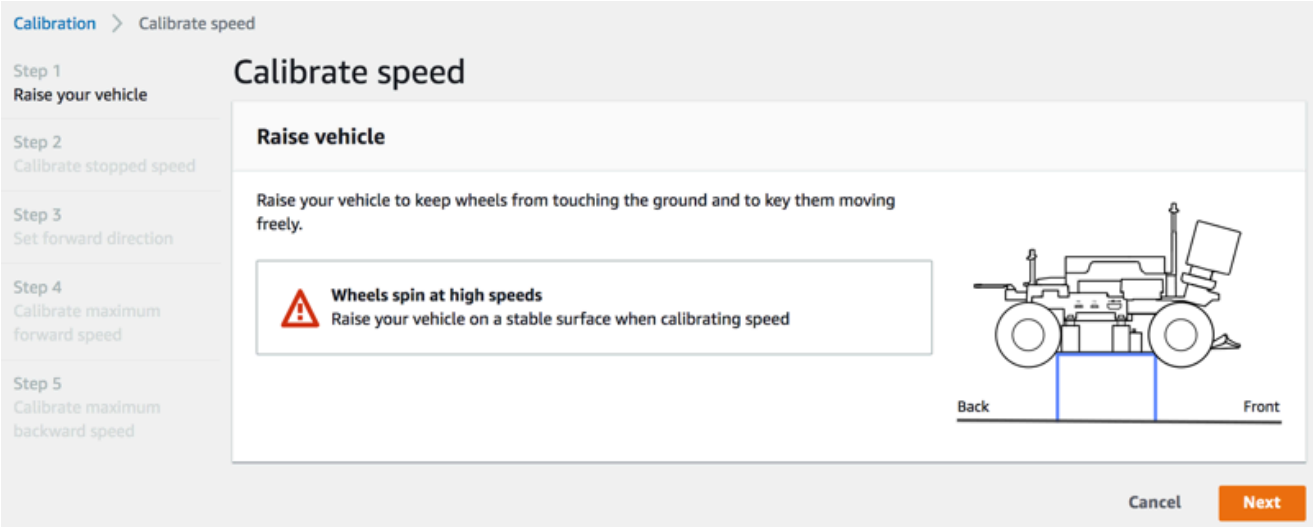


To measure the actual maximum right steering angle, follow the steps similar to those used to measure the maximum left steering angle.

This concludes the steering calibration for your AWS DeepRacer vehicle.

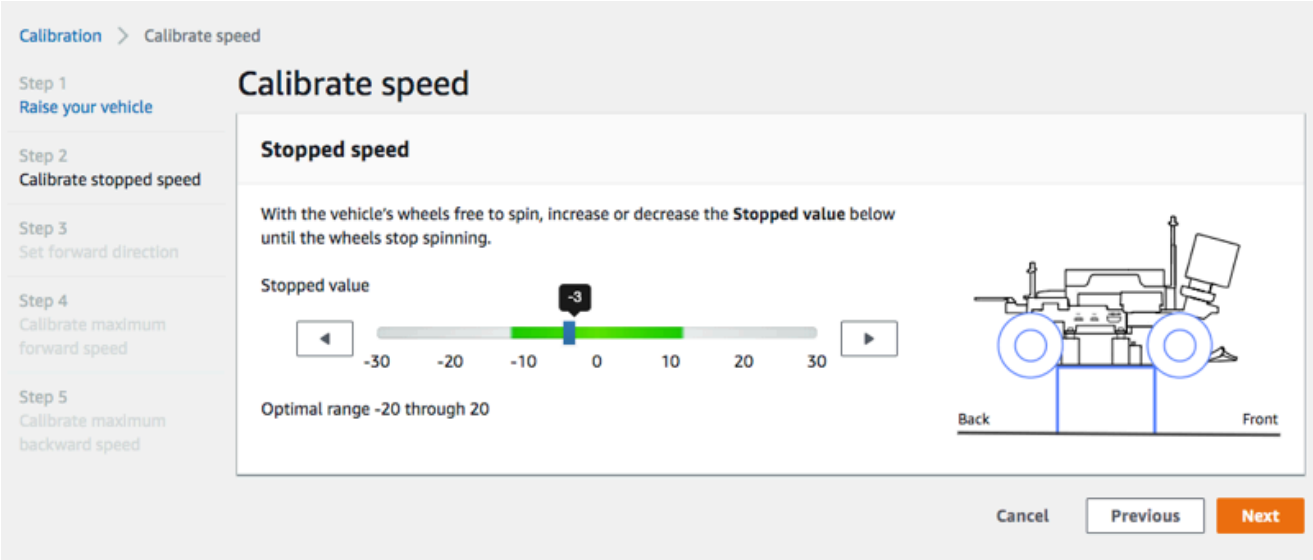
4. To calibrate the vehicle's maximum speed, choose **Calibrate** in **Speed** on the **Calibration** page and then follow the steps below.

   a. Raise the vehicle so that the wheels are free to turn. Choose **Next** on the device control console.



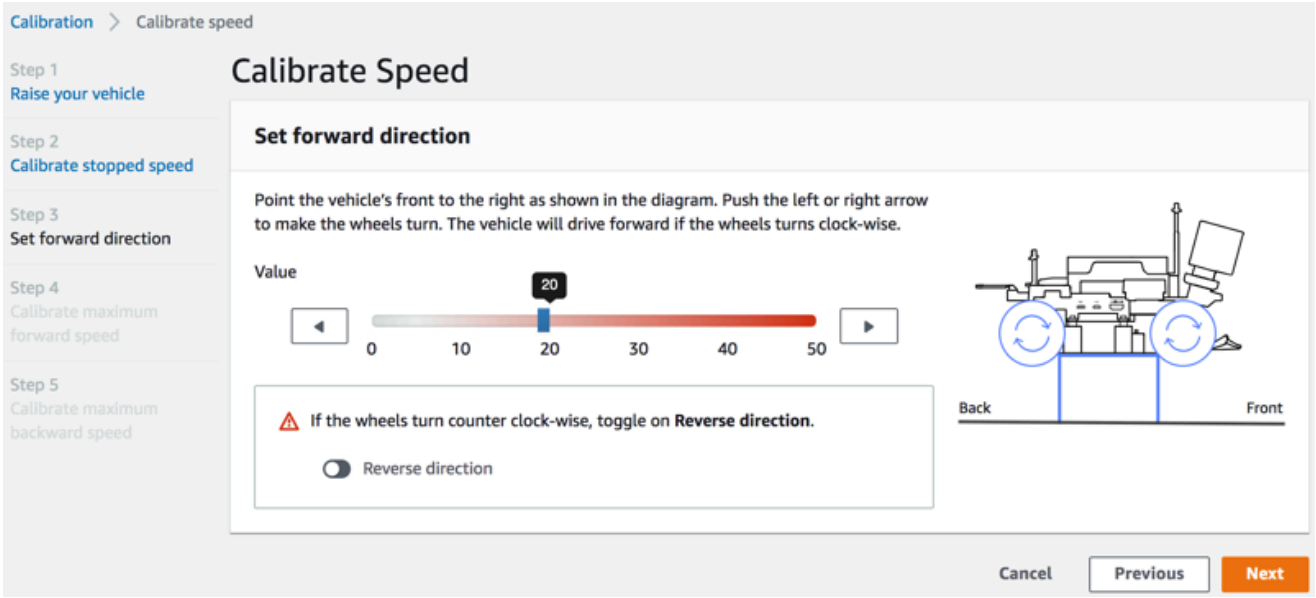
> **Note**
>
> If the vehicle's speed has been set too high, it may run too fast during calibration and cause damage to the environment, the vehicle, or others nearby. You should raise the vehicle, as instructed here, but not hold it in your hands.

b. To calibrate the stopped speed, press the left or right arrow to gradually change **Stopped value** under **Stopped speed** on the device control console until the wheels stop turning. Choose **Next**.



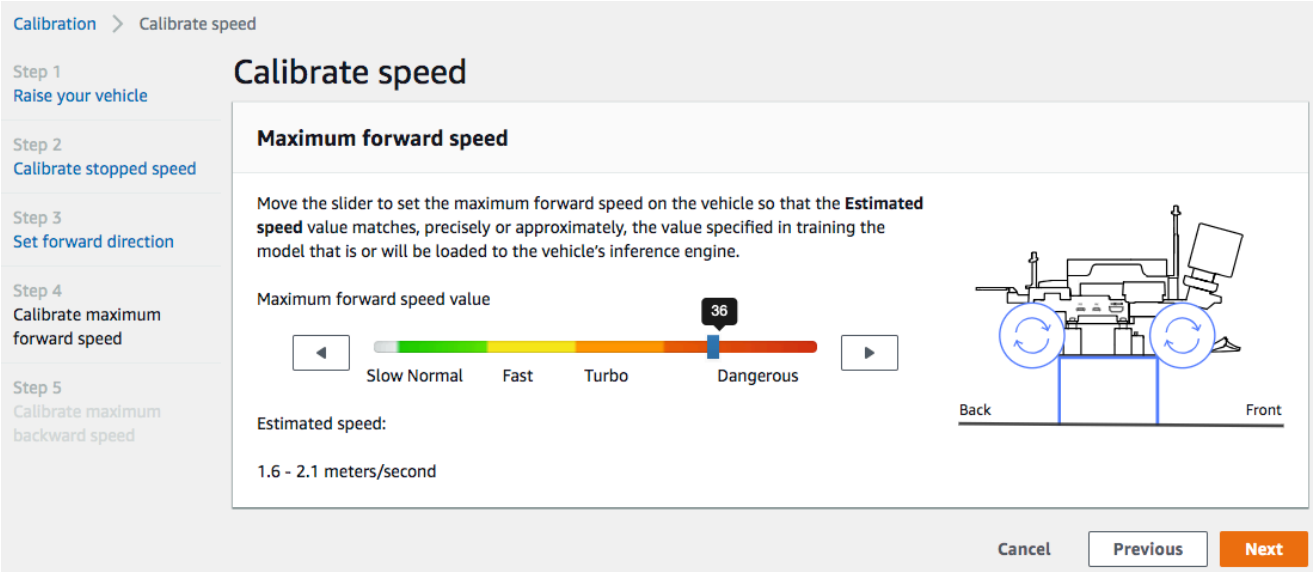
> **Note**
>
> When pressing the **Stopped value** further left or further right to the value when you start hearing noises, the wheels are about to move. The ideal zero-throttle point is the middle of the two values. For example, if you start hearing a noise at 16 on the left and at -4 on the right, the optimal stopped value should be 10.

c. To set the vehicle's forward direction, place the vehicle as shown on the screen and the image here, and then press the left or right arrow to make the wheels turn. If the wheels turn clock-wise, the forward direction is set. If not, toggle **Reverse direction**. Choose **Next**.



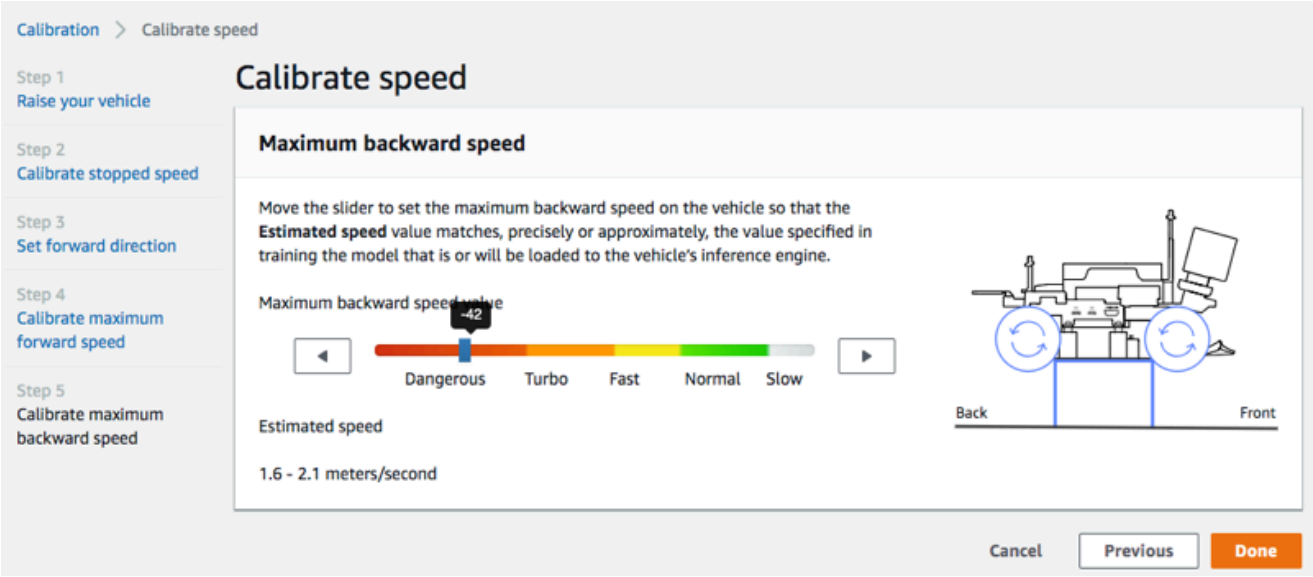
> **Note**
>
> Vehicles distributed at AWS re:Invent 2018 might have their forward direction set in reverse. In such a case, make sure to toggle **Reverse direction**.

d. To calibrate the maximum forward speed, under **Maximum forward speed**, gently move the slider left or right to adjust the **Maximum forward speed value** number gradually to such a positive value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Next**.



> **Note**
>
> The actual maximum speed your vehicle depends on the friction of the track surface as well as the vehicle battery level. To make it flexible, you can set the vehicle's throttle limit to be 20-30 percent higher than the maximum speed specified for training in the simulation. Generally speaking, you should set the maximum speed value within the green area. Above that, your vehicle is likely to drive too fast at increased risk of breaking. Additionally, the action space for training doesn't support the maximum speed of more than 2 m/s.

e. To calibrate the maximum backward speed, under **Maximum backward speed**, gently move the slider left or right to adjust the **Maximum backward speed value** number gradually to such a negative value that the **Estimated speed** value is equal or similar to the maximum speed specified in the simulation. Choose **Done**.



> **Note**
>
> The AWS DeepRacer vehicle doesn't use backward speed in the autonomous driving mode. You can set the backward speed to any value with which you can comfortably control the vehicle's manual driving mode.

This concludes calibrating your AWS DeepRacer vehicle's maximum speed.

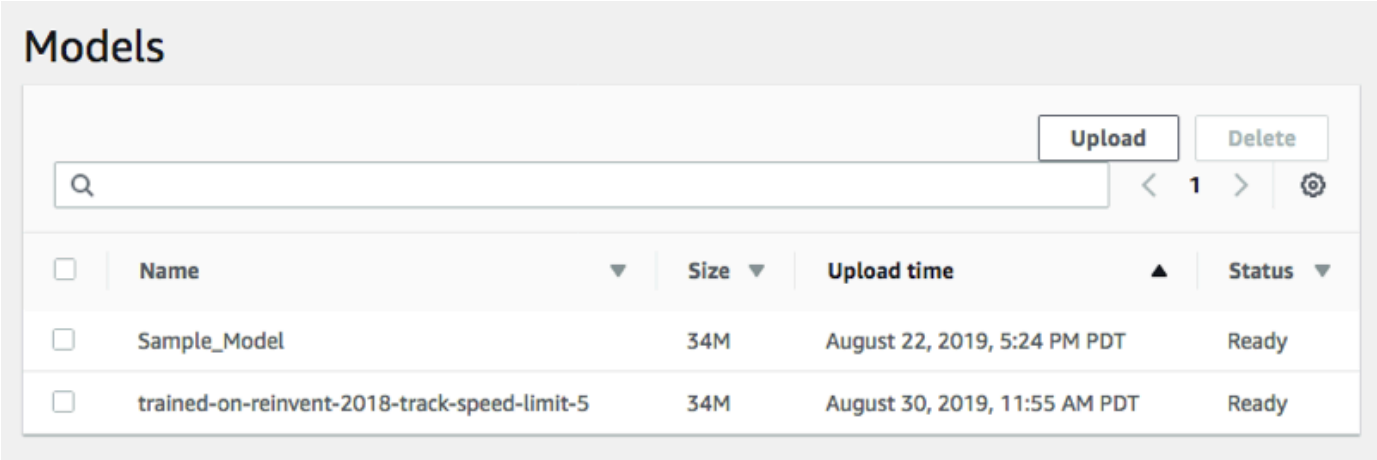
Upload a model to your AWS DeepRacer vehicle

To start your AWS DeepRacer vehicle on autonomous driving, you must have uploaded at least one AWS DeepRacer model to your AWS DeepRacer vehicle.

To upload a model, you must have trained and evaluated the model. You can train the model using the AWS DeepRacer console. After that, you need to download the model artifacts from its Amazon S3 storage to a (local or network) drive that can be accessed by your computer.

To upload a trained model to your vehicle

1. Choose **Models** from the device console's main navigation pane.



2. On the **Models** page, choose **Upload** above the **Models** list.
3. From the file picker, navigate to the drive or share where you've downloaded your model artifacts and choose the the compressed model file (of the `*.tar.gz` extension) to upload.

   Only a successfully uploaded model will be added to the **Models** list and can be available for you to load it into the vehicle's inference engine in the autonomous driving mode. For the instructions on how to load a model into your vehicle's inference engine, see Drive your AWS DeepRacer vehicle autonomously.

Drive your AWS DeepRacer vehicle

After setting up your AWS DeepRacer vehicle, you can start to drive your vehicle manually or let it drive autonomously, using the vehicle's device console.

For autonomous driving, you must have trained an AWS DeepRacer model and have the trained model artifacts deployed to the vehicle. In the autonomous racing mode, the model running in the inference engine controls the vehicle's driving directions and speed. Without a trained model downloaded to the vehicle, you can use the vehicle's device console to drive the vehicle manually.

Many factors affect the vehicle's performance in autonomous driving. They include the trained model, vehicle calibration, track conditions, such as surface frictions, color contrasts and light reflections, etc. For your vehicle to achieve an optimal performance, you must make sure that the model transfer from the simulation to the real world is as accurate, relevant and meaningful. For more information, see the section called “Optimize training for real environments”.

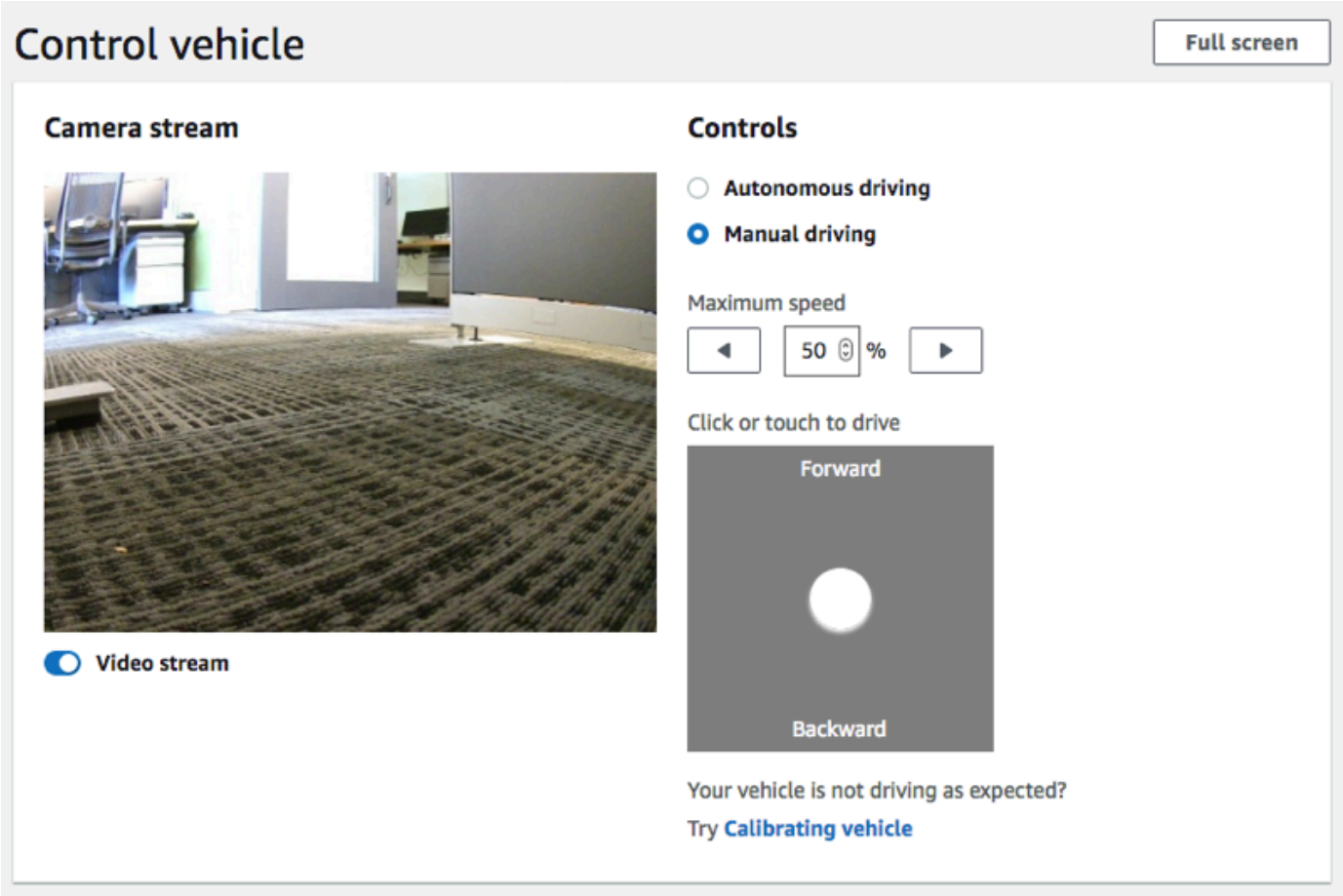
Drive your AWS DeepRacer vehicle manually

If you have not trained any model or have not deployed any trained model to your AWS DeepRacer vehicle, you can't let it drive itself. But you can drive it manually.

To drive a AWS DeepRacer vehicle manually, follow the steps below.

To drive your AWS DeepRacer vehicle manually

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign in to the vehicle's device control console.
2. On the **Control vehicle** page, choose **Manual driving** under **Controls**.



3. Under **Click or touch to drive**, click or touch a position within the driving pad to drive the vehicle. Images captured from the vehicle's front camera are displayed in the video player under **Camera stream**.
4. To turn video stream on or off on the device console while you drive the vehicle, toggle the **Video stream** option under the **Camera stream** display.

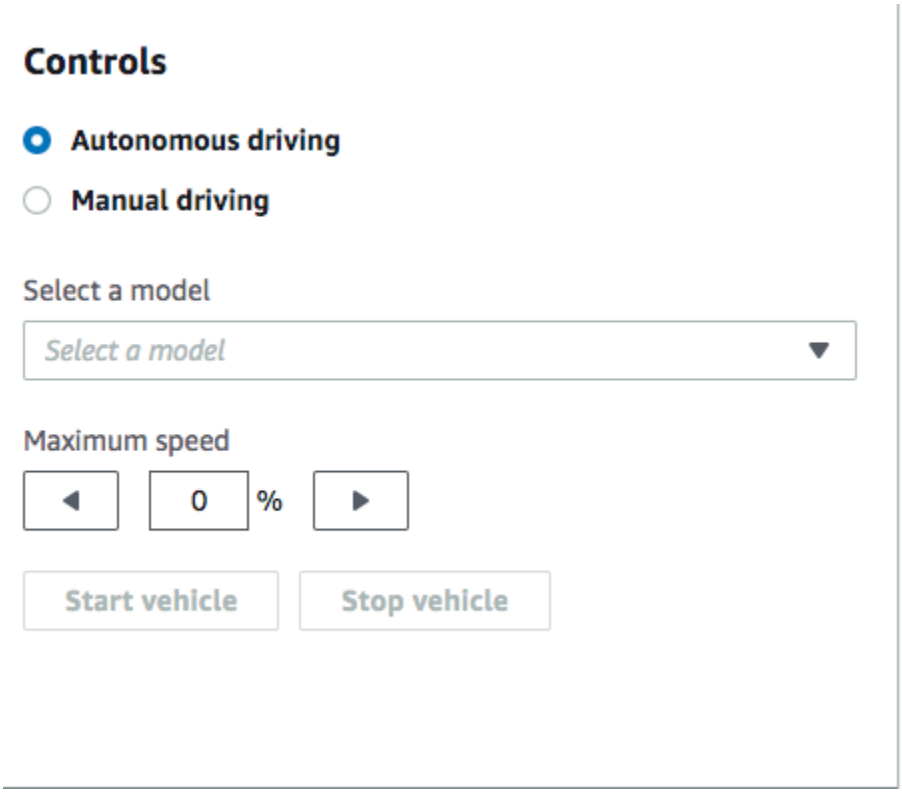
5. Repeat from **Step 3** to drive the vehicle to different locations.

Drive your AWS DeepRacer vehicle autonomously

To start autonomous driving, place the vehicle on a physical track and do the following:

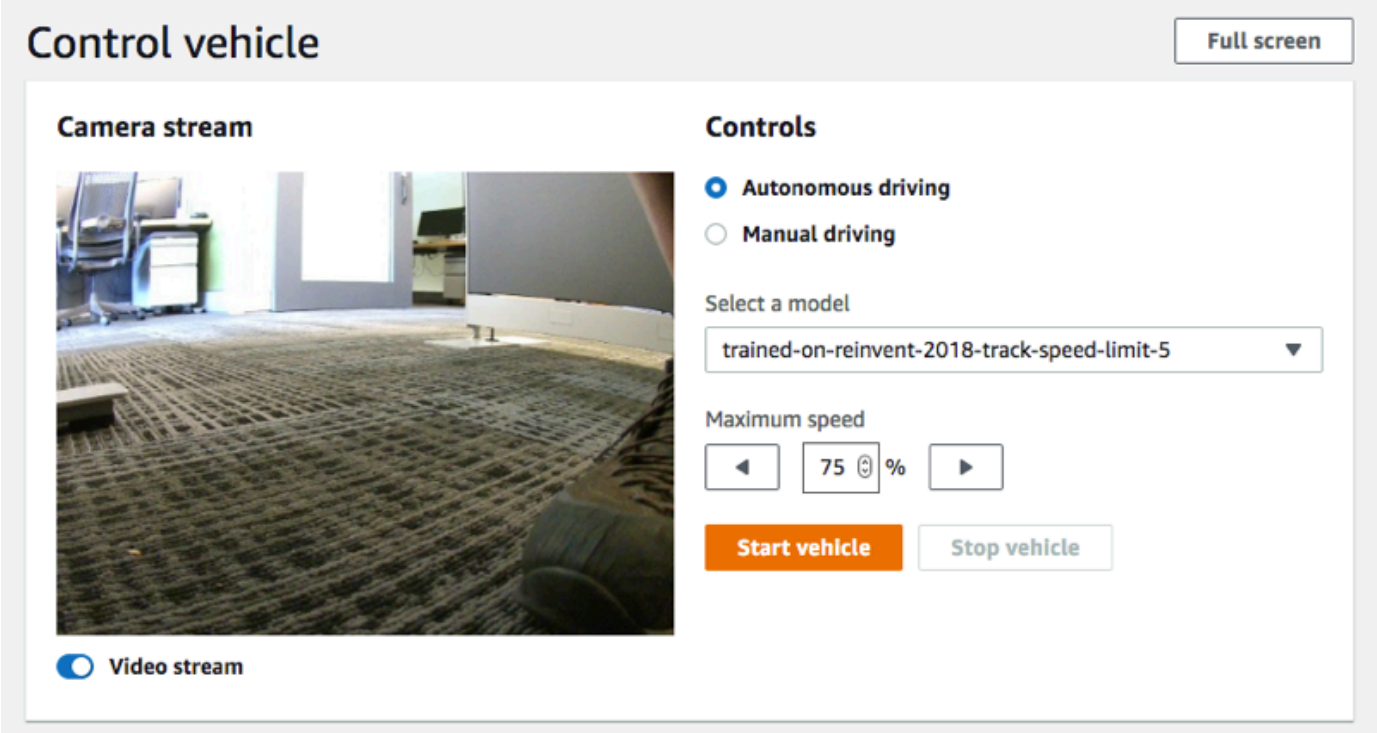
To drive your AWS DeepRacer vehicle autonomously

1. Follow the instructions to sign in to the vehicle's device console, and then do the following for autonomous driving:
2. On the **Control vehicle** page, choose **Autonomous driving** under **Controls**.



3. From the **Select a model** drop-down list, choose an uploaded model. Then choose **Load model**. This will start loading the model into the inference engine. The process takes about 10 seconds to complete.
4. Adjust the **Maximum speed** setting of the vehicle to be a percentage of the maximum speed used in training the model.

   Certain factors, such as surface friction of the real track, can reduce the maximum speed of the vehicle from the maximum speed used in the training. You'll need to experiment to find the optimal setting.



5. Choose **Start vehicle** to set the vehicle to drive autonomously.
6. To turn video stream on or off on the device console while you drive the vehicle, toggle the **Video stream** option under the **Camera stream** display.
7. Watch the vehicle drive on the physical track or the streaming video player on the device console.
8. To stop the vehicle, choose **Stop vehicle**.

Repeat from Step 3 for another run with the same or a different model.

Inspect and manage your AWS DeepRacer vehicle settings

After the initial setup, you can use the AWS DeepRacer device control console to manage your vehicle's settings. The tasks include the following:

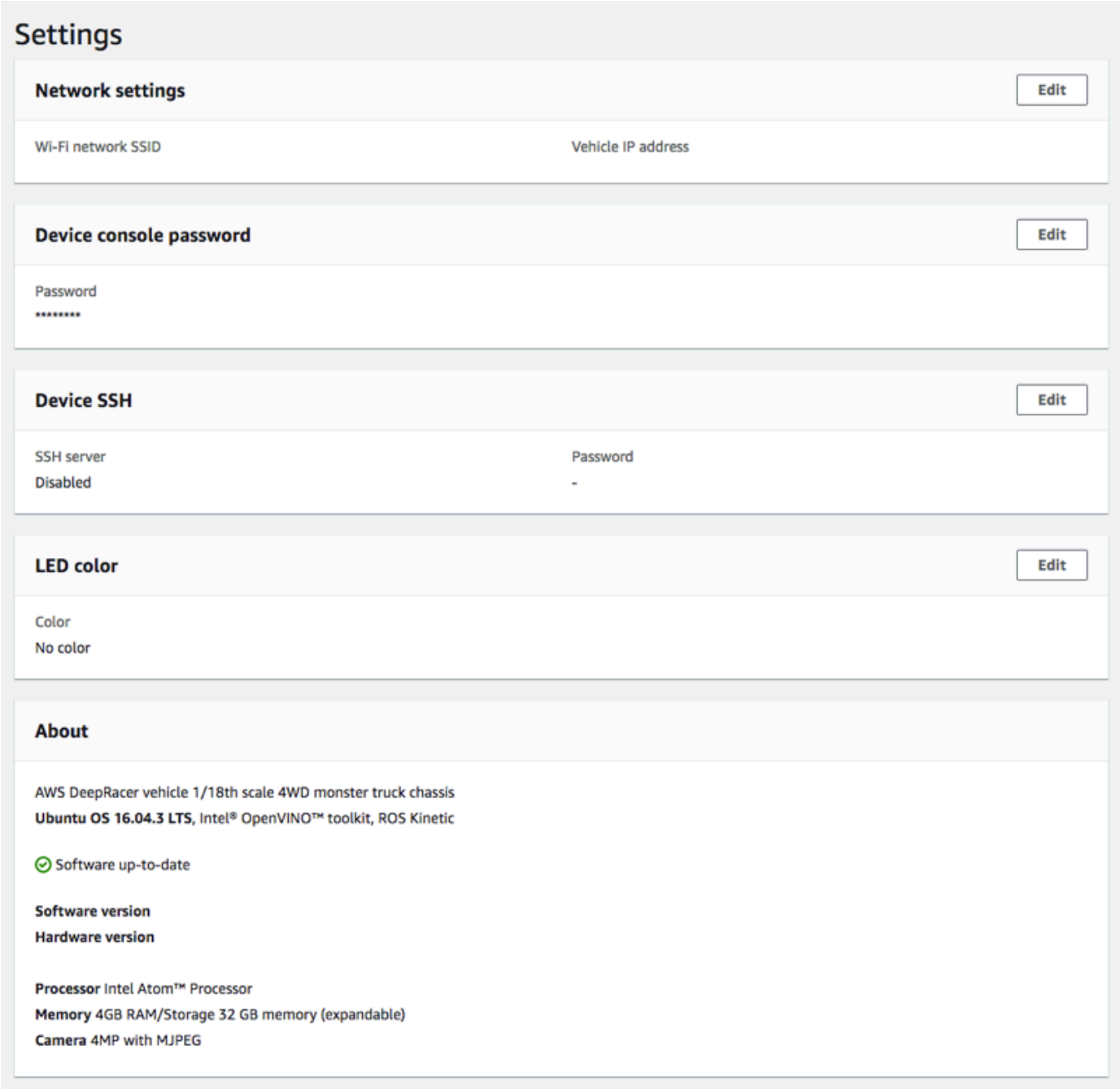
- choosing another Wi-Fi network,
- resetting the device console password,
- enabling or disabling the device SSH settings,
- configuring the vehicle's trail light LED color,
- inspecting the device software and hardware versions,

- checking the vehicle battery level.

The procedure below walks you through these tasks.

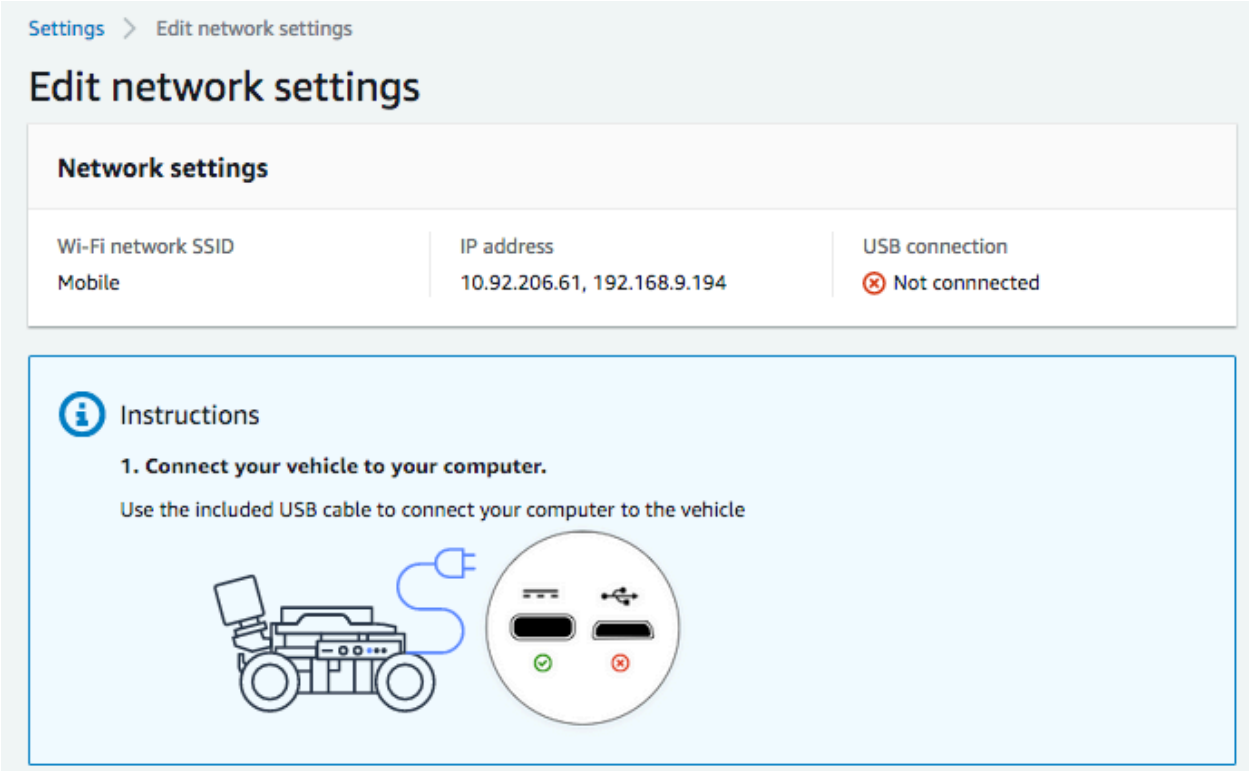
To inspect and manage your vehicle's settings

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign in to the vehicle's device control console.
2. Choose **Settings** from the main navigation pane.
3. On the **Settings** page, perform one or more of the following tasks of your choosing.



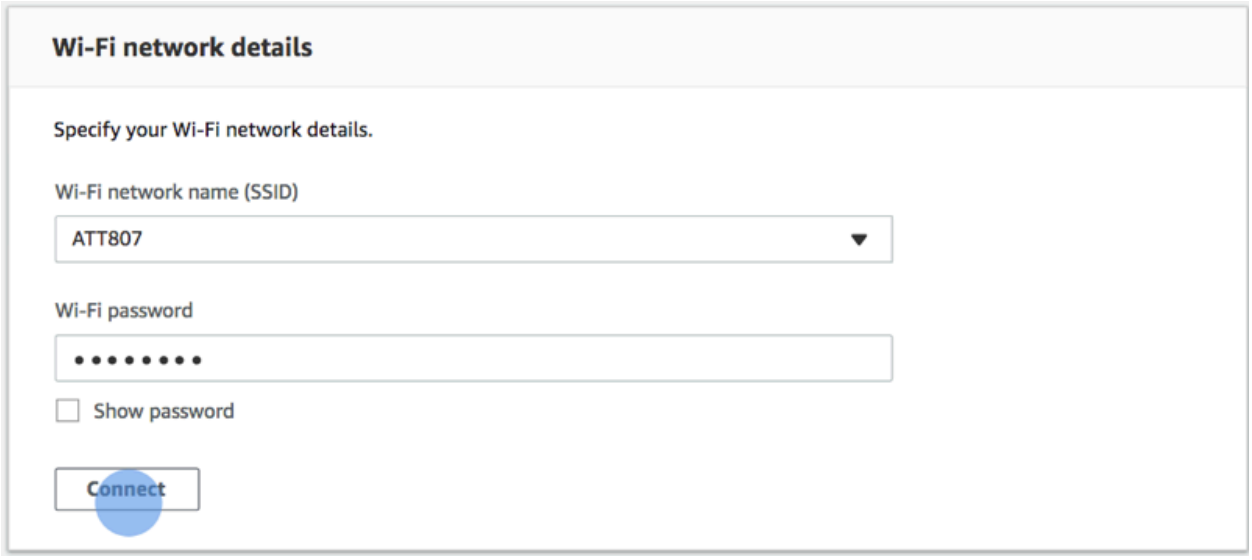
a. To choose another Wi-Fi network, choose **Edit** for **Network settings** and then follow the steps below.

   i. Follow the instructions, shown on **Edit network settings**, to connect your vehicle to your computer using the USB-to-USB-C cable. After the **USB connection** status becomes **Connected**, choose the **Go to deepracer.aws** button to open the device console login page.



ii. On the device console login page, type the password printed on the bottom of your vehicle and then choose **Access vehicle**.

iii. Under **Wi-Fi network details**, choose a Wi-Fi network from the drop-down list, type the password of the chosen network, and then choose **Connect**.



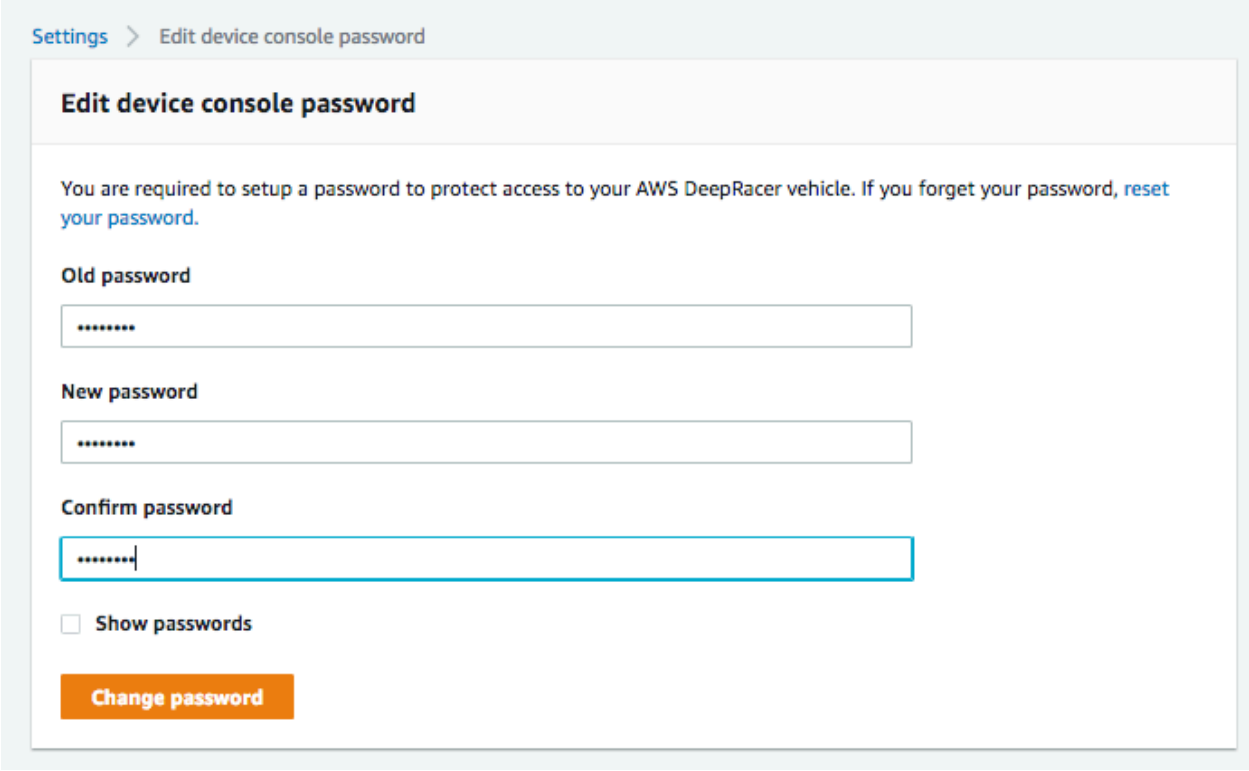
iv. After the **Vehicle status** for the Wi-Fi connection becomes **Connected**, choose **Next** to return to the **Settings** page of the device console, where you'll see a new IP address of the vehicle.

b. To reset the password for signing in to the device console, choose **Edit** for **Device console password** and then follow the steps below.

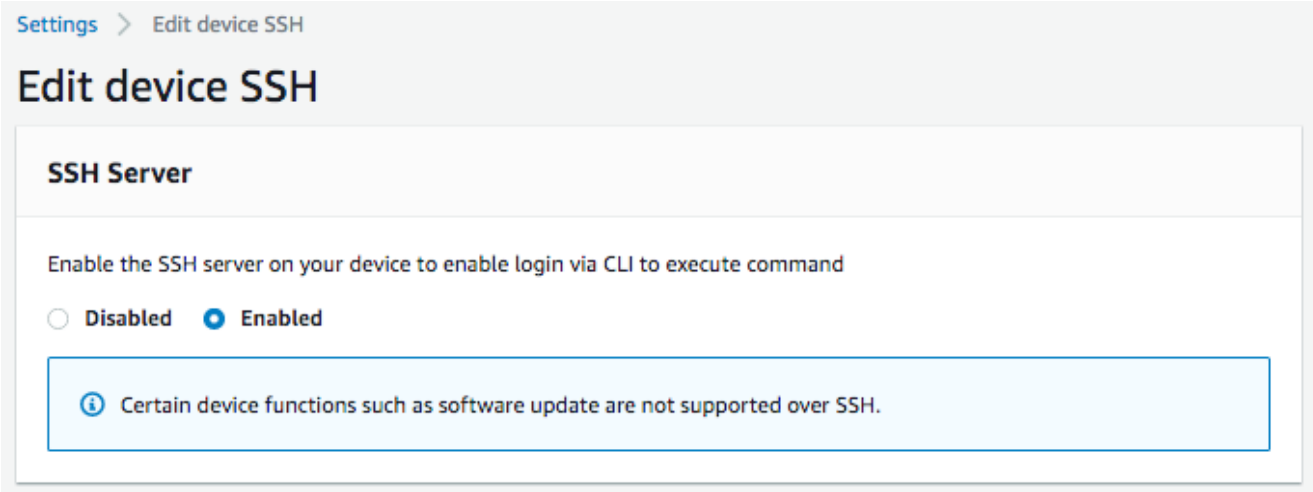
   i. On **Edit device console password** page, type a new password in **New password**.

   ii. Retype the new password in **Confirm password** to confirm your intention for the change. The password value must be the same before you can move on.

   iii. Choose **Change password** to complete the task. This option is activated only if you have entered and confirmed a valid password value in the steps above.

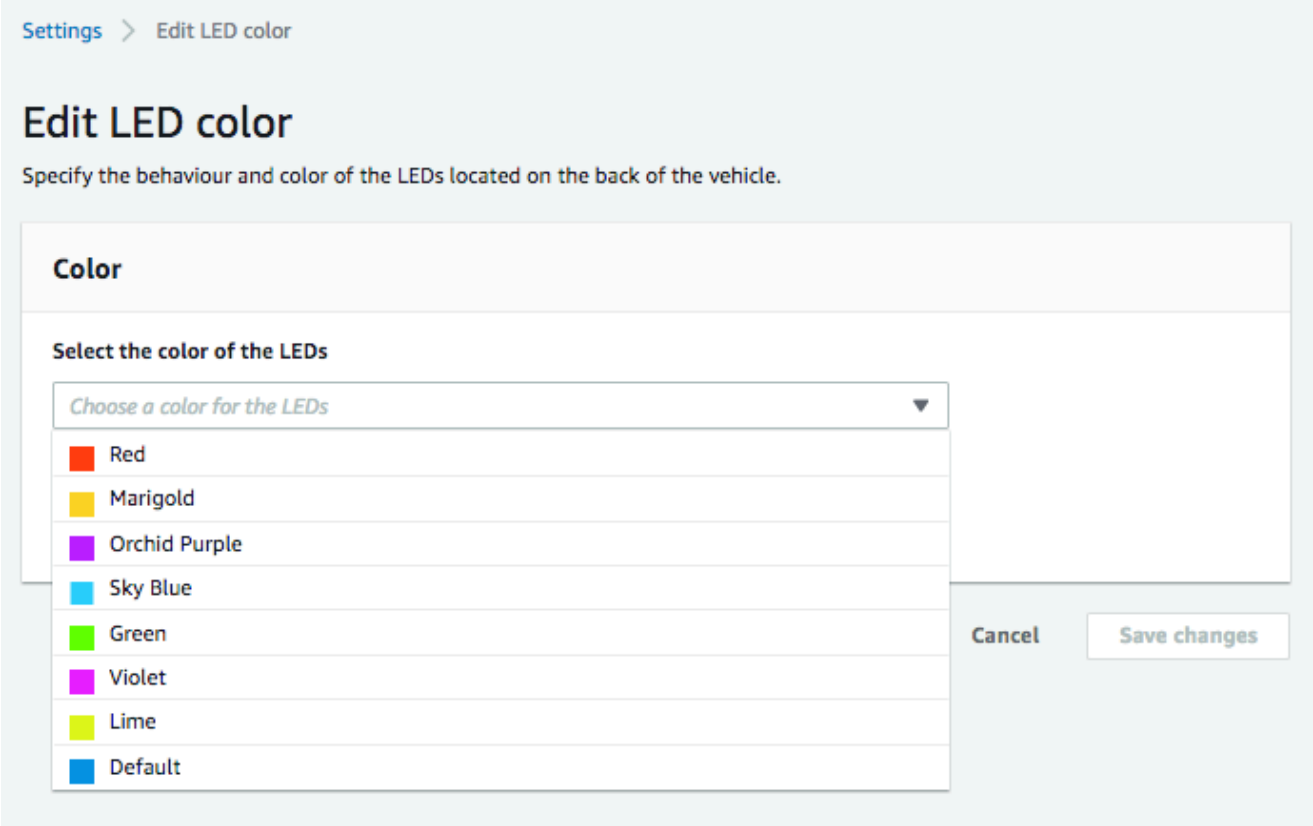


c. To enable or disable SSH connection to the vehicle, choose **Edit** for **Device SSH** and then choose **Enable** or **Disable**.



4. To change the vehicle's trail light LED color to distinguish your vehicle on a track, choose **Edit** for **LED color** on the **Settings** page and do the following.

   a. Choose an available color from the **Select the color of the LEDs** drop-down list on the **Edit LED color** page.



   You should choose a color that can help identify your vehicle from other vehicles sharing the track at the same time.

b. Choose **Save changes** to complete the task.

   The **Save changes** functionality becomes active only after you have chosen a color.

5. To inspect the device software and hardware versions and to find out the system and camera configurations, check the **About** section under **Settings**.
6. To inspect the vehicle battery's charge level, check the lower part of the primary navigation pane.

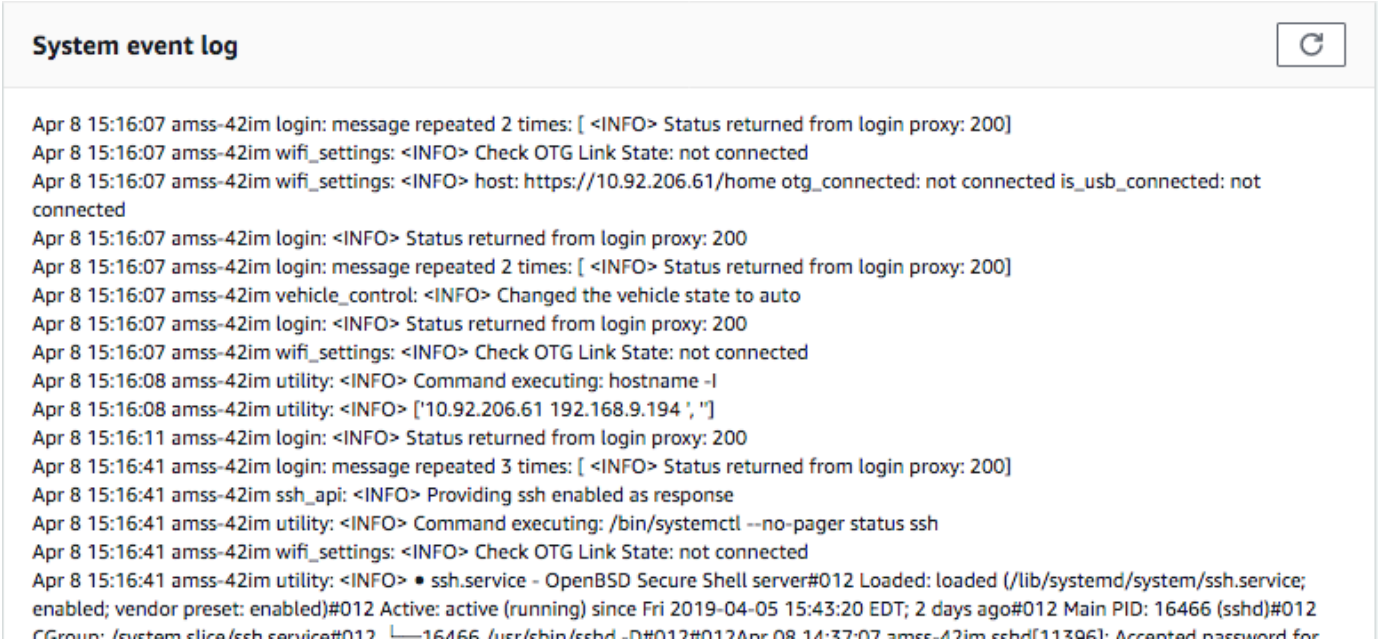
View your AWS DeepRacer vehicle logs

Your AWS DeepRacer vehicle logs operational events that can be helpful for troubleshooting issues encountered in running your vehicle. There are two types of AWS DeepRacer vehicle logs:

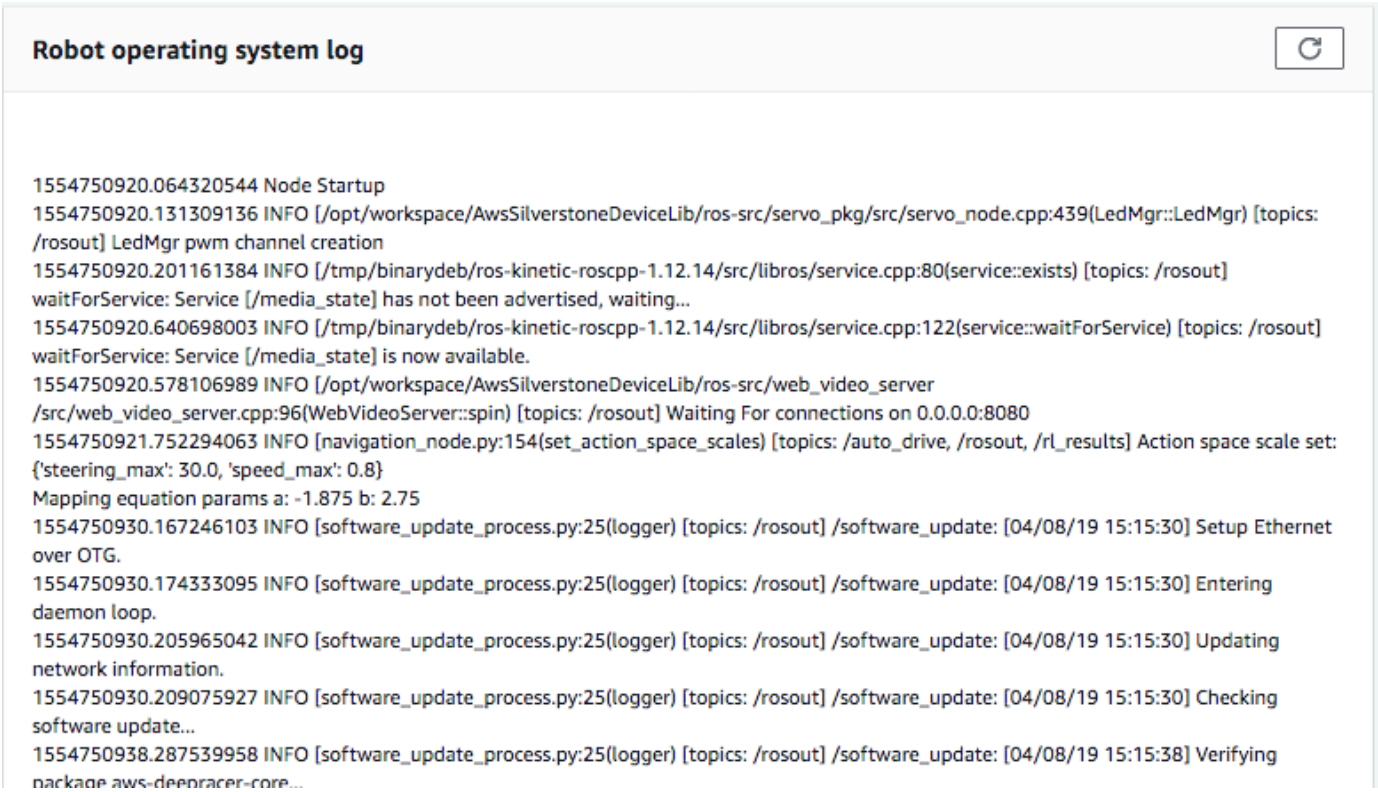
- The system event log keeps track of operations taking place in the vehicle's computer operating system, such as process managing, Wi-Fi connecting or password reset events.
- The robot operating system logs record statuses of operations taking place in the vehicle's operating system node for robotic operations, including vehicle driving, video streaming and policy inferencing operations.

To view the device logs, follow the steps below.

1. With your AWS DeepRacer vehicle connected to the Wi-Fi network, follow the instructions to sign in to the vehicle's device control console.
2. Choose **Logs** from the device console's main navigation pane.
3. To view the system events, scroll down the event list under **System event log**.



4. To view the robot operating system events, scroll down the event list under **Robot operating system log**.



Update and restore your AWS DeepRacer device

Update your AWS DeepRacer device to the latest software stack including Ubuntu 20.04 Focal Fossa, Intel® OpenVINO™ toolkit 2021.1.110, ROS2 Foxy Fitzroy, and Python 3.8. This update is required to run AWS DeepRacer open-source projects but is otherwise optional. AWS DeepRacer only supports Ubuntu 20.04 Focal Fossa and ROS2 Foxy Fitzroy.

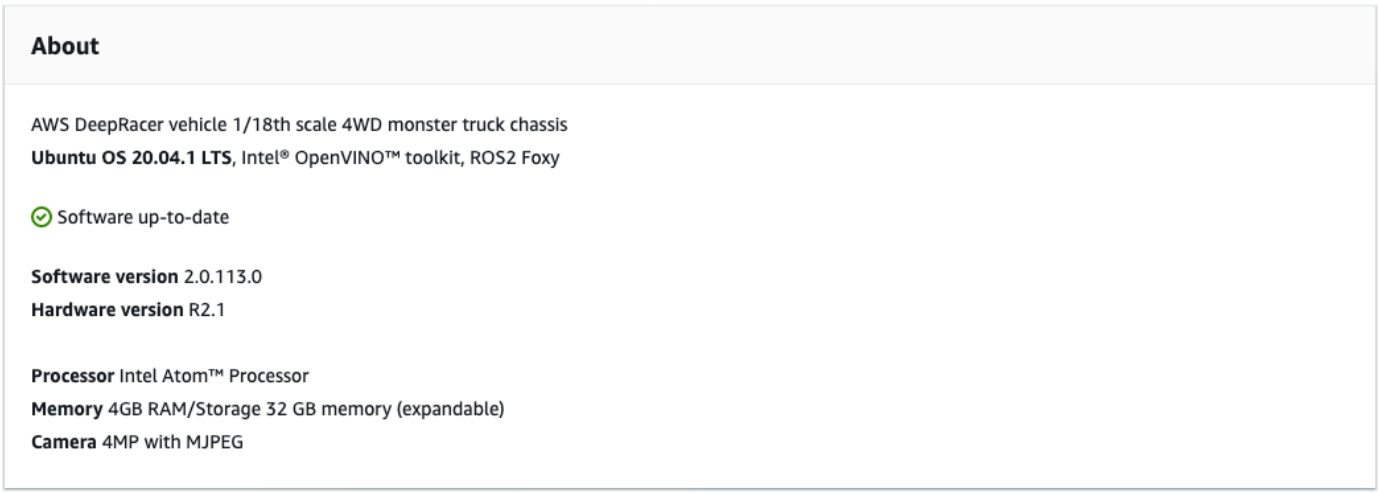
> **Important**
>
> Updating to the new AWS DeepRacer software stack will wipe all data on your AWS DeepRacer device.

Topics

Check which software version your AWS DeepRacer device is currently running

To check which software version your AWS DeepRacer device is currently running

1. Log in to the AWS DeepRacer device console. To learn how, follow the steps in the section called “Launch device console”.
2. Choose **Settings** on the navigation pane.
3. Check the **About** section to verify which software version your AWS DeepRacer Vehicle is currently running.



Prepare to update your AWS DeepRacer device to the Ubuntu 20.04 software stack

This topic walks you through the process to create the AWS DeepRacer Ubuntu instalation media. Preparing the bootable USB drive requires additional hardware.

Prerequisites

Before you get started, make sure you have the following items ready:

- An AWS DeepRacer device
- A USB flash drive (32 GB or larger)
- A custom AWS DeepRacer Ubuntu ISO image .
- The latest AWS DeepRacer software update package .
- A copy of UNetbootin compatible with your operating system.
- A computer running Ubuntu, Windows, or macOS to prepare the USB instalation media. You can also use the compute module on your AWS DeepRacer device as a Linux computer by connecting a mouse, keyboard, and monitor with an HDMI type A cable.

Preparation

To prepare the AWS DeepRacer update media, you will perform the following tasks:

- Format the USB drive into the following two partitions:
  - A 4 GB, FAT32 boot partition
  - An NTFS data partition of at least 18 GB
- Make the USB drive bootable to start the update on reboot:
  - Burn the required custom Ubuntu ISO image to the boot partition
  - Copy the required update files to the data partition of the USB drive

Prepare a bootable USB drive

Follow these instructions to prepare your AWS DeepRacer update media on Ubuntu (Linux), Windows, or macOS. Depending on the computer you use, specific tasks may differ from one operating system to another. Choose the tab corresponding to your operating system.

Ubuntu

Follow the instructions here to use an Ubuntu computer, including your AWS DeepRacer device's compute module, to prepare the update media for your AWS DeepRacer device. If you are using a different Linux distribution, replace the `apt-get *` commands with those compatible with your operating system's package manager.

To erase and partition the USB drive

1. Run the following commands to install and launch GParted.

   ```
   sudo apt-get update; sudo apt-get install gparted
   sudo gparted
   ```

2. To erase your USB drive, you will need its device path. To find it on the GParted console and erase the USB drive, do the following:

   a. On the menu bar, choose **View**, then choose **Device Information**. A sidebar showing the selected disk's **Model**, **Size**, and **Path** will appear.

   b. Select your USB drive by going to **GParted** on the menu bar, then **Devices**, finally, select your USB drive from the list. Match the **Size** and **Model** shown in the **Device Description** with your USB drive.

   c. Once you are sure that you've selected the correct disk, delete all its existing partitions.

      If the partitions are locked, open the context (right-click) menu and choose **unmount**.

3. To create the FAT32 boot partition with a 4 GB capacity, select the file icon on the top-left, set the following parameters, and choose **Add**.

   Free space preceding: `1`

   New size: `4096`

   Free space following: `<remaining size>`

   Align to: `MiB`

   Create as: `Primary Partition`

   Partition name:

   File system: `fat32`

   Label: `BOOT`

4. To create the NTFS data partition with a minimum 18 GB capacity, select the file icon, set the following parameters, and choose **Add**.

   Free space preceding: `0`

   New size: `<remaining size>`

   Free space following: `0`

   Align to: `MiB`

   Create as: `Primary Partition`

   Partition name:

   File system: `nfts`

   Label: `Data`

5. On the menu bar, choose **Edit**, then **Apply All Operations**. A warning prompt will appear asking if you want to apply the changes. Choose **Apply**.
6. After the FAT32 and NTFS partitions are created, the USB drive's partition information will appear in the GParted console. Make note of the B00T partition's drive path, you will need it to complete the next step.

To make the USB drive bootable from the FAT32 partition

1. Make sure you downloaded the custom Ubuntu ISO image from the pre-requisites section.
2. If you're using Ubuntu 20.04, you need to run UNetbootin using its binary file. To do this:

   a. Download the latest UNetbootin binary file to your Downloads folder. In our example, we use `unetbootin-linux64-702.bin`.

   b. Press **Ctrl+Alt+T** to open a new terminal window. Alternatively, choose **Activities** on the menu bar, enter **`terminal`** in the search bar, then select the **Terminal** icon.

   c. Use the following commands to navigate to the binary file location, give the file execute permission, and run UNetbootin. Make sure to adjust the file name in the commands if the version doesn't match the one on your downloaded binary file.

   ```
   cd Downloads
   sudo chmod +x ./unetbootin-linux64-702.bin
   sudo ./unetbootin-linux64-702.bin
   ```

   If you're using an older version of Ubuntu, install UNetbootin from its repository by running the following commands:

   ```
   sudo add-apt-repository ppa:gezakovacs/ppa
   sudo apt-get update; sudo apt-get install unetbootin
   sudo unetbootin
   ```

3. On the **UNetbootin** console, do the following:

   a. Select the **Disk image** radio button.

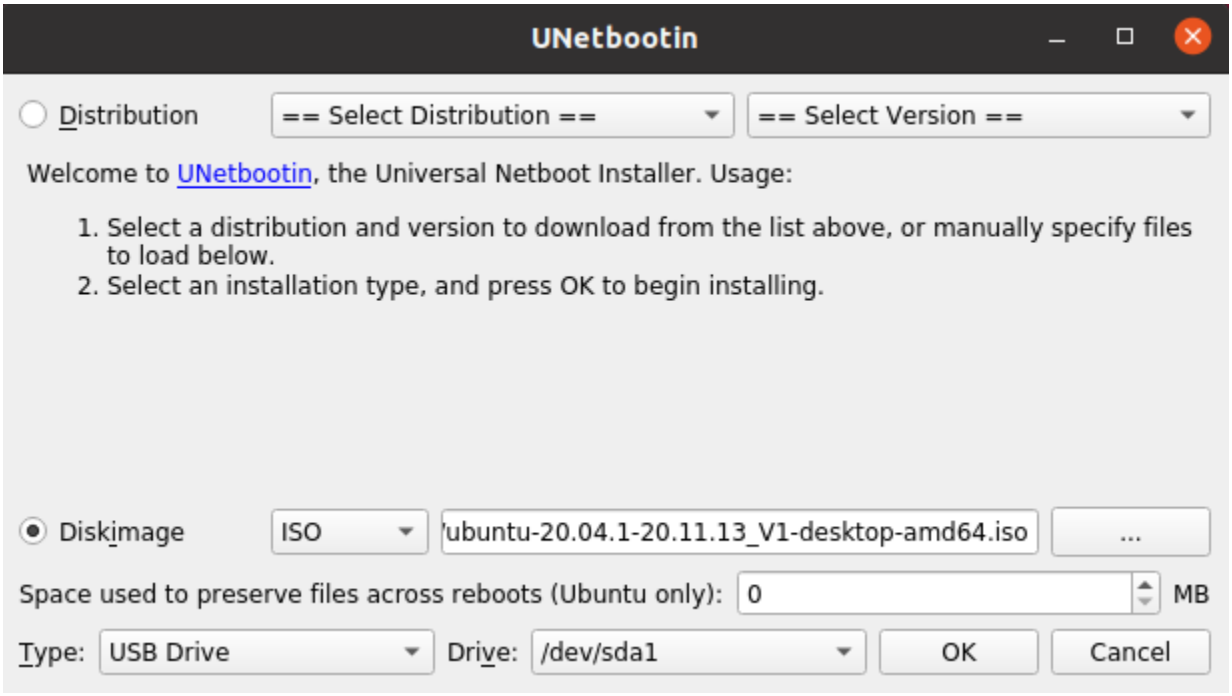
   b. For the disk image type, choose **ISO** from the drop-down list.

   c. Open the file selector and choose the Ubuntu ISO provided in the pre-requisites section.

   d. For **Type**, choose **USB Drive**.

   e. For **Drive**, choose the drive path for your B00T partition, in our case **`/dev/sda1`**.

   f. Choose **OK**.



> **Tip**
>
> If you get a **/dev/sda1 not mounted** alert message, choose **OK** to close the message, unplug the USB drive, plug in the drive again, and then follow the preceding steps to create the Ubuntu ISO image.

To extract the AWS DeepRacer update files to the NTFS partition

1. Unzip the software update package you downloaded from the prerequisites section.
2. Extract the contents of the update package to the root of your USB drive's Data (NTFS) partition.

Windows

Follow the instructions here to use a Windows computer to prepare the update media for your AWS DeepRacer device.

To erase the USB drive

1. Open the Windows command prompt, enter `diskpart`, and choose **OK** to launch Windows DiskPart.
2. Once the terminal for Microsoft DiskPart opens, list the available disks to find the USB drive you want to clean by entering `list disk` after the **DISKPART>** prompt.
3. Select the disk corresponding to your USB drive. For example, we entered `select Disk 2` after the **DISKPART>** prompt. Read the output carefully to verify that you have chosen the disk you want to clean because the next step is irreversible.
4. Once you are sure that you've selected the correct disk, enter `Clean` after the **DISKPART>** prompt.
5. Enter `list disk` after the **DISKPART>** prompt again. Find the disk you cleaned on the table and compare the disk size to the free disk space. If the two values match, the cleaning was successful.
6. Exit the Windows **DiskPart** console by entering `Exit` after the **DISKPART>** prompt.

To partition the USB drive

1. Open the Windows command prompt, enter `diskmgmt.msc`, and choose **OK** to launch the **Disk Management** console.
2. From the **Disk Management** console, select your USB drive.
3. To create the FAT32 partition with a 4 GB capacity, open the context (right-click) menu on your USB drive's **Unallocated** space and choose **New Simple Volume**. The New Simple Volume Wizard will appear.
4. Once the New Simple Volume Wizard appears, do the following:

    a. On the **Specify Volume Size** page, set the following parameter and then choose **Next**.

    Simple volume size in MB: 4096

    b. On the **Assign Drive Letter or Path** page, check the **Assign the following drive letter:** radio button and select a drive letter from the drop down list, then choose **Next**. Make note of the assigned drive letter, you will need it later to make the FAT32 partition bootable.

    c. On the **Format Partition** page, check the **Format this volume with the following settings** radio button and set the following parameters, then choose **Next**.

File system: `FAT32`

Allocation unit size: `Default`

Volume label: `BOOT`

Leave **Perform a quick format** checked.

5. To create the NTFS partition with the remaining disk capacity, open the context (right-click) menu on your USB drive's remaining **Unallocated** space and choose **New Simple Volume**. The New Simple Volume Wizard will appear.
6. Once the New Simple Volume Wizard appears, do the following:

   a. On the **Specify Volume Size** page, set the **Simple volume size in MB** to match the **Maximum disk space in MB**, then choose **Next**.

   b. On the **Assign Drive Letter or Path** page, check the **Assign the following drive letter:** radio button and select a drive letter from the drop down list, then choose **Next**.

   c. On the **Format Partition** page, check the **Format this volume with the following settings** radio button and set the following parameters, then choose **Next**.

   File system: `NTFS`

   Allocation unit size: `Default`

   Volume label: `Data`

   Leave **Perform a quick format** checked.

To make the USB drive bootable from the FAT32 partition

1. Make sure you've downloaded the customized Ubuntu ISO image from the prerequisites section.
2. After downloading UNetbootin, start the **UNetbootin** console.
3. On the UNetbootin console, do the following:

   a. Check the **Disk image** radio button.

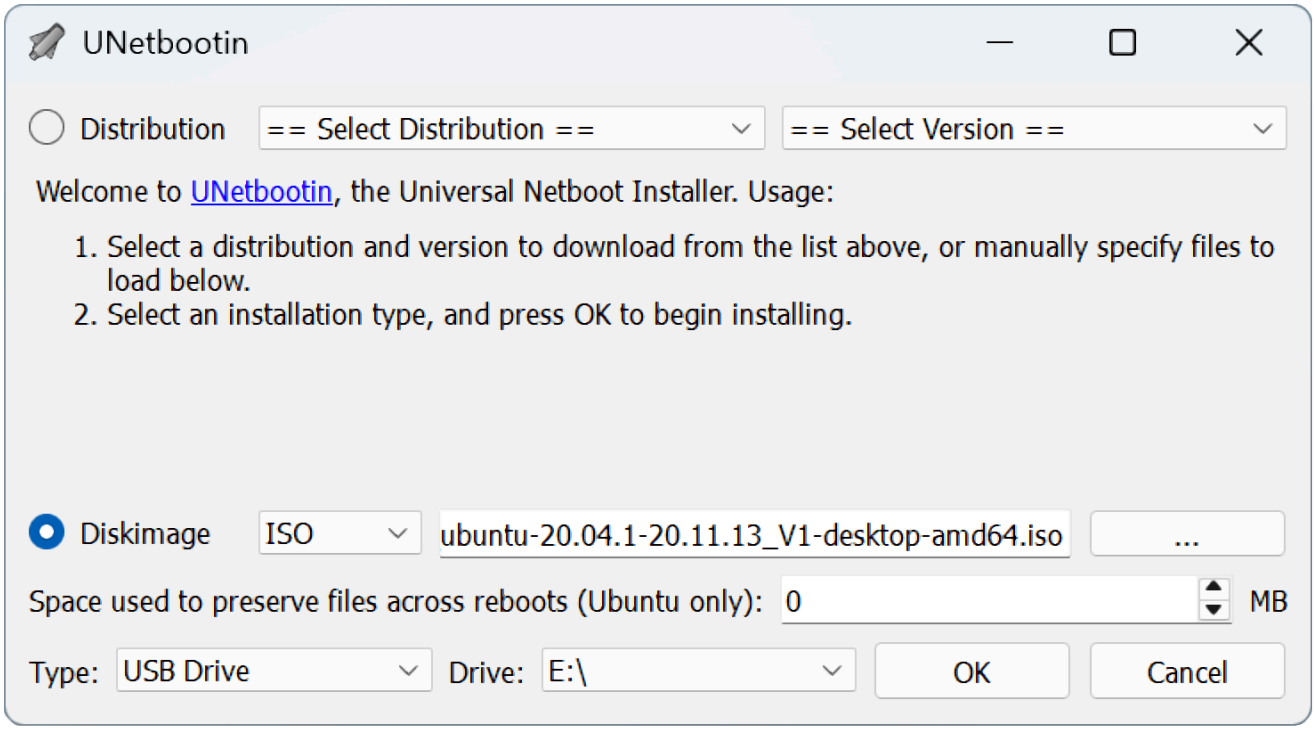
   b. For **disk image**, choose **ISO** from the drop-down list.

   c. Open the file picker and choose the custom Ubuntu ISO file.

d. For **Type**, choose **USB Drive**.

e. For **Drive**, choose the drive letter corresponding to the FAT32 partition you created. In our case, it's `E:\`.

f. Choose **OK**.



To extract the AWS DeepRacer update files to the NTFS partition

1. Unzip the software update package you downloaded from the prerequisites section.

> **Tip**
>
> If your favorite tool can't unzip the file successfully, try using the PowerShell Expand-Archive command.

2. Extract the contents of the update package to the root of your USB drive's Data (NTFS) partition.

macOS

Follow the instructions here to use a Mac to prepare the update media for your AWS DeepRacer device.

To erase and partition the USB drive

1. Plug in the USB drive to your Mac.
2. Press **Command+Space bar** to open the **Spotlight** search field, then enter `Disk Utility`.

   Alternatively, you can choose **Finder** > **Applications** > **Utilities** > **Disk Utility** to open Disk Utility.
3. On the menu bar, choose **View**, then **Show All Devices**.
4. In the sidebar, under **External**, select the USB drive that you want to format and then choose **Erase**.
5. A new window will ask you to confirm that you want to erase your USB drive and will allow you to change its **Name**, **Format**, and **Partition Scheme**. You don't need to change the name yet, for **Format** and **Scheme**, select the following options and choose **Erase**.

   - **Format:** Mac OS Extended (Journaled)
   - **Scheme:** GUID Partition Map

   Once the erase process is complete, choose **Done** on the dialog window.
6. On the main Disk Utility window, select your USB drive from the sidebar, choose **Partition** from the toolbar on the top. A window titled **Partition device "*YOUR-USB-DRIVE*"?** will pop up. Select the **add (+)** button to create a new partition.
7. Once you create the new partition, under **Partition Information**, choose and enter the following:

   - **Name: B00T**
   - **Format:** MS-DOS (FAT)
   - **Size: 4** GB

> **Tip**
>
> If the **Size** input box is grayed out after choosing MS-DOS (FAT) as the format, you can drag the resize control on the partition graph until the **B00T** partition is 4 GB.

Do not choose **Apply** yet.

8. Select the other **Untitled** partition, choose and enter the following options under **Partition Information**:

   - **Name: `Data`**
   - **Format:** ExFAT
   - **Size:** the remaining space of the USB drive (in GB)

   Choose **Apply**.

9. A new window will pop up and show you the changes that will be made to the USB Drive. Verify that these changes are correct. To confirm and begin the creation of the new partitions, choose **Partition**.
10. On the Disk Utility console, choose the **BOOT** partition from the Sidebar, then select **Info** from the Toolbar. Make note of the **BSD device node** value, it might be different from the one used in this tutorial. In our case, the value assigned is `disk4s2`. You need to supply this path when making the USB drive bootable from the FAT32 partition.

To make the USB drive bootable from the FAT32 partition

1. Make sure you've downloaded the customized Ubuntu ISO image from the prerequisites section.
2. After downloading UNetbootin, select **open** from the context (right-click) menu. A security prompt will appear asking if you want to open the application, select **open** to start the UNetbootin console.

   If you are using a Mac with Apple Silicon, and the UNetbootin console does not show after selecting **open**, make sure that Rosetta 2 is installed by following these steps:

   a. Open a terminal window by choosing **Finder** > **Applications** > **Utilities** > **Terminal**.

b. Enter the following command to install Rosetta 2:

```
softwareupdate --install-rosetta
```

c. Retry opening UNetbootin.

3. On the UNetbootin console, do the following:

a. Check the **Disk image** radio button.

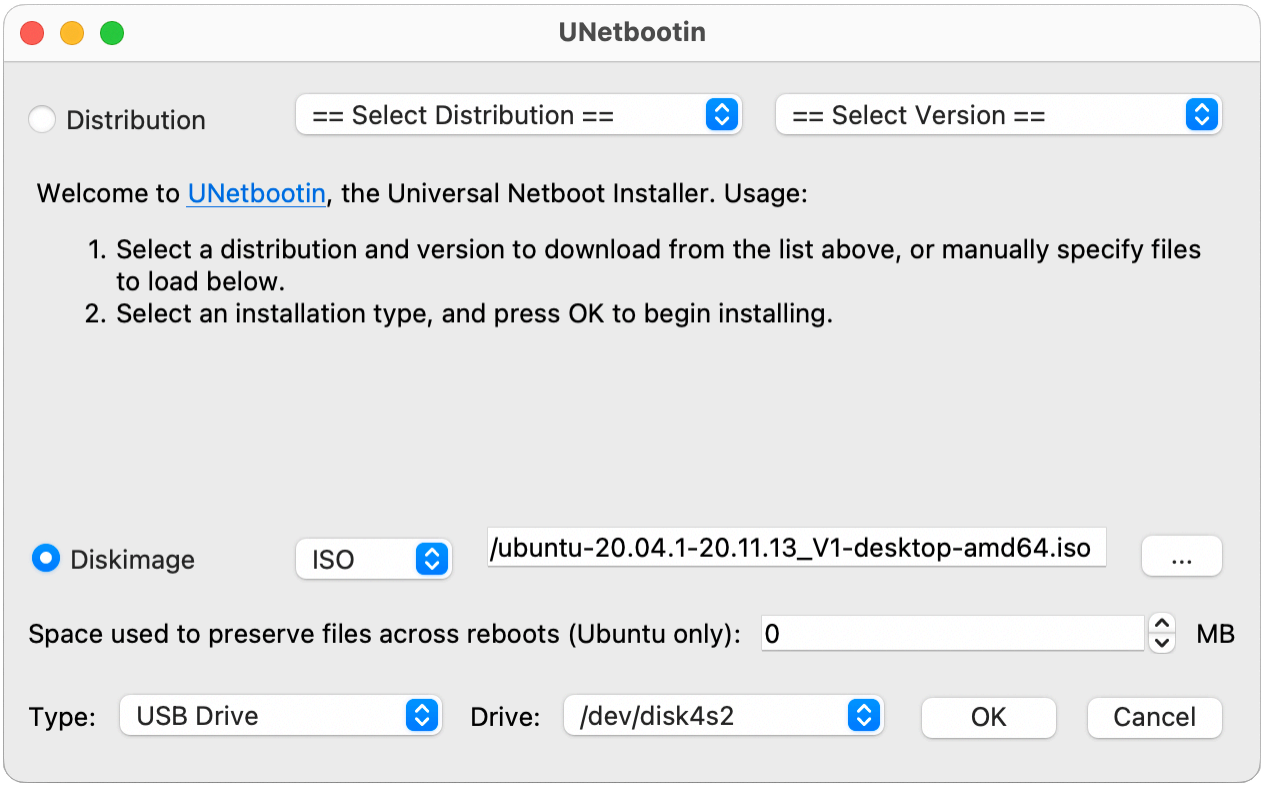
b. For **disk image**, choose **ISO** from the drop-down list.

c. Open the file picker and choose the custom Ubuntu ISO file.

d. For **Type**, choose **USB Drive**.

e. For **Drive**, choose the BSD device node for your BOOT partition, in our case, `/dev/disk4s2`.

f. Choose **OK**.



> **Tip**
>
> If you get a **/dev/disk4s2 not mounted** alert message, choose **OK** to close the message, unplug the USB drive, replug the drive, and then follow the steps above create the Ubuntu ISO image.

To extract the AWS DeepRacer update files to the ExFAT partition

1. Unzip the software update package you downloaded from the prerequisites section.
2. Extract the contents of the update package to the root of your USB drive's Data (ExFAT) partition.

Update your AWS DeepRacer device to the Ubuntu 20.04 software stack

Once you create the USB update media as described in the previous steps, you can update your AWS DeepRacer device to the latest software stack including Ubuntu 20.04 Focal Fossa, Intel® OpenVINO™ toolkit 2021.1.110, ROS2 Foxy Fitzroy, and Python 3.8.

> **Important**
>
> Updating to the new AWS DeepRacer software stack will wipe all data on your AWS DeepRacer device.

To update your AWS DeepRacer device software to the Ubuntu 20.04 stack

1. Connect your AWS DeepRacer device to a monitor. You'll need an HDMI-to-HDMI, HDMI-to-DVI, or similar cable. Insert the HDMI end of the cable into the compute module's HDMI port and plug the other end into a compatible port on the monitor.
2. Connect a USB keyboard and mouse. The AWS DeepRacer device's compute module has three USB ports in the front of the vehicle, on either side of and including the port into which the camera is plugged. A fourth USB port is found at the back of the vehicle, in the space between the compute battery and the LED tail light.

3. Insert the USB update media into an available USB port on your compute module. Turn on the power or reset your AWS DeepRacer device and repeatedly press the **ESC** key to enter the BIOS.
4. From the BIOS window, choose **Boot From File**, then select the option with your boot partition's name, in our case it's named **BOOT** , then select **<EFI>**, then **<BOOT>**, and finally **BOOTx64.EFI**.
5. After the compute module has booted, a terminal window will appear on the desktop to display the progress. The AWS DeepRacer device will automatically begin the update process after ten seconds. You don't need to provide any input at this stage.

   If an error occurs and the update fails, restart the procedure from **Step 1**. For detailed error messages, see the `result.log` file generated on the USB drive's data partition.
6. Wait for the update to complete. When the factory reset is complete the terminal window will close automatically.
7. After the device software is updated, disconnect the USB drive from the compute module. You can now reboot or shut down your AWS DeepRacer device.
8. The AWS DeepRacer device defaults to the following user credentials after update. You will be prompted to change your password on your first login.

   User: Deepracer

   Password: `deepracer`

Build your physical track for AWS DeepRacer

This section describes how you can build a physical track for a AWS DeepRacer model. To drive your AWS DeepRacer autonomously and to test your reinforcement learning model in a physical environment, you need a physical track. Your track resembles the simulated track used in training and replicates the environment used to train the deployed AWS DeepRacer model.

For the best experience, we recommend using pre-printed tracks and track barriers. Using pre-printed tracks and barriers facilitates the smooth set up and installation of the AWS DeepRacer track environment. Instead of building a track from scratch, you assemble pre-printed sections of track and track barriers. When your event is over, you can disassemble and store and reuse the pre-printed tracks and barriers for future events. Pre-printed tracks and barriers as well as details for estimating space and other requirements for events are available at AWS DeepRacer Storefront.

Topics

Track materials and build tools

Before you start to construct you track, get the following materials and tools ready.

Topics

Materials you may need

To build a track, you need the following materials:

- For track borders:

  You can create a track with tape that is about 2-inches wide and white or off-white color against the dark-colored track surface. For a dark surface, use a white or off-white tape. For example, 1.88 inch width, pearl white duct tape or 1.88 inch (less sticky) masking tape.

- For track surface:

  You can create a track on a dark-colored hard floor such as hardwood, carpet, concrete, or asphalt felt. The latter mimics the real-world road surface with minimal reflection. Interlocked foam or rubber pads are also good options.

Tools you may need

The following tools are either required or helpful to design and build your track:

- Tape measure and scissors

  A good tape measure and a pair of scissors are essential for building your track. If you don't already have one, you can order a tape measure here or scissors here.

- Optional design tools

  To design your own track, you might need a protractor, a ruler, a pencil, a knife and a compass.

Lay your track for AWS DeepRacer

When you build your track, it's a good practice to start with a simple design, such as a straight or single-turn track. Next you can move on to looped tracks. Here, we use a single-turn track as an example to walk you through the steps to construct your own track. First let's review dimensional requirements of a track.

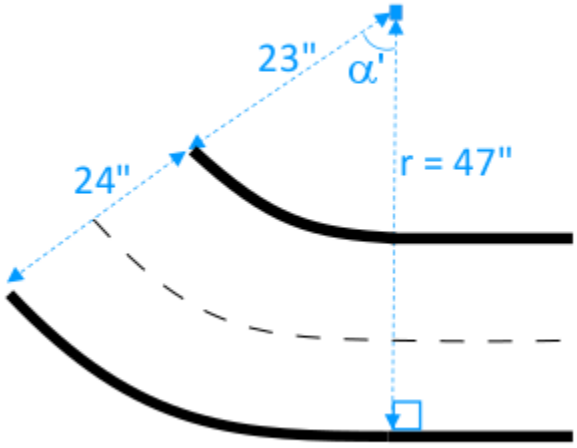
Topics

Dimensional Requirements

You can build a track of any shape as long as it meets the following requirements:

- **Minimum turning radius**:

On a curved track, the turning radius (`r`) measures from the circle center to the outside border, as illustrated below.



The minimum turning radius (r_{min}) depends on the track turning angle (α) at a corner and should comply to the following limits:

- If the track's turning angle is $\alpha \leq 90$ degrees,

  $r_{min} \geq 25$ inches

  We recommend 30 inches.

- If the track's turning angle is $\alpha > 90$ degrees, α

  $r_{min} \geq 30$ inches.

  We recommend 35 inches.

- **Track width**,

  The track width (w_{track}) should comply to the following limit:

  $w_{track} \geq 24 \pm 3$ inches.

- **Track surface**:

  The track surface should be smooth and of a uniform dark color. The minimum enclosing area should be `30 inches x 60 inches` in size.

Carpeted and wood floors work well. Interlocked foam or rubber pads match the simulated environment better than wood, but this is not required. Concrete floors can be problematic due to light reflection on the surface.

- **Track barrier**

  Though not required, we recommended that you encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

Considerations for model performance

How you build a track can affect the reliability and performance of a trained model. The following are factors you should consider when building your own tracks.

1. Do not place any white objects on or near your track. If necessary, remove any white object from the track or its vicinity. This is because training in the simulated environment assumes that only the track borders are white.
2. Use clean and continuous tape to mark the track borders. Broken or creased track borders can affect the trained model performance.
3. Avoid using a reflective surface as the track floor. Reduce glare from bright lights. The glare from straight edges can be misinterpreted as objects or borders.
4. Do not use a track floor with line markings other than the track lines. The model might interpret the non-track lines as part of the track.
5. Place barriers around the track to help reduce distractions from background objects.

Steps to build the track

As an illustration, we use the most basic single-turn track. You can modify the instructions to create a more complex track such as an S-curve, a loop, or the AWS re:invent 2018 track.

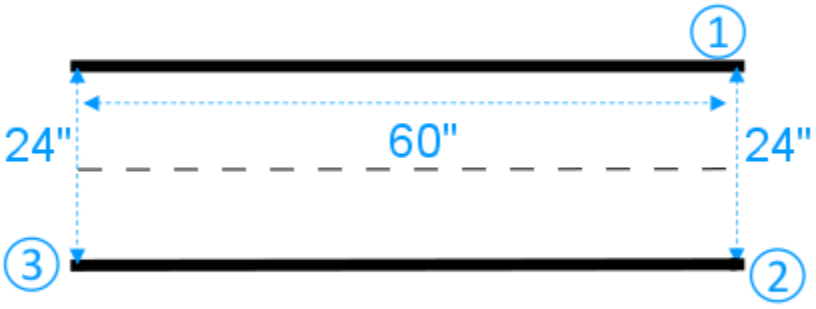
To build an AWS DeepRacer single-turn track

1. To construct the straight portion of the track, follow the steps below and refer to the diagram.

   a. Put a 60-inch long piece of tape on the floor to lay down the first border in a straight line (1).

b. Use a tape measure to locate the second border's two end points, (2) and (3). Put them 24 inches apart from the first border's two ends.

c. Put another 60-inch long piece of tape on the floor to lay down the second boarder to connect the two endpoints (2) and (3).

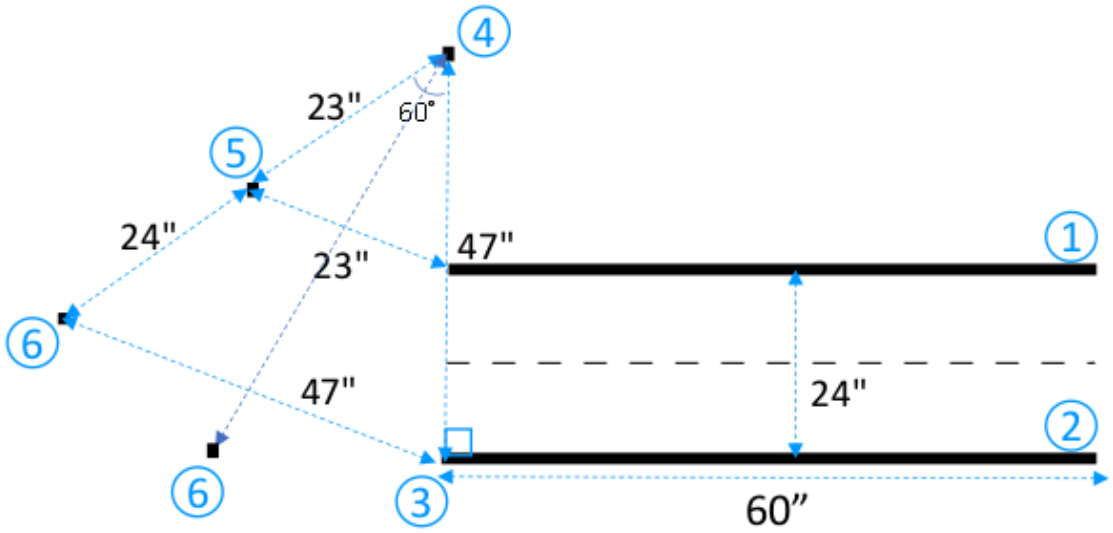


We assume the straight track segment is 60-inches long and 24-inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met.

2. To make the track to turn at a 60-degree angle, do the following and refer to the diagram:

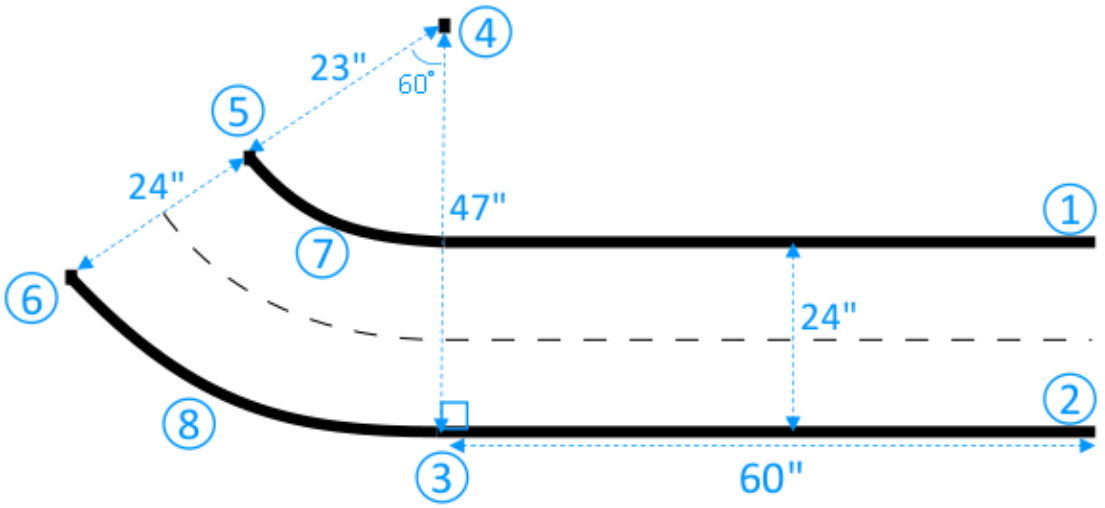
   a. Use the tape measure to locate the center (4) of the turning radius (4-3 or 4-6). Mark the center with a piece of tape.

   b. Draw an equilateral triangle. The three sides are (3-4), (4-6), and (6-3).



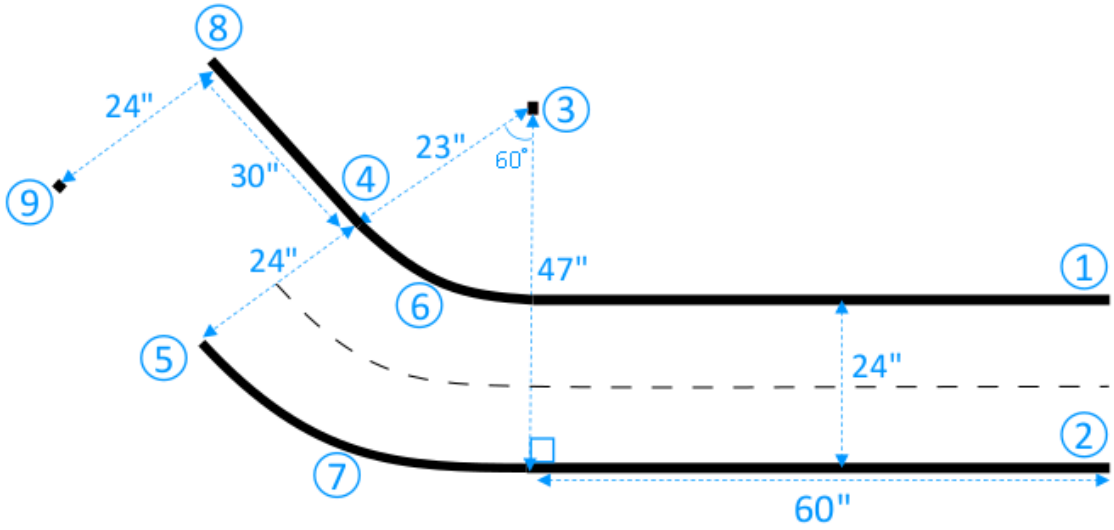
To make a 60-degree turn along the track, use the equilateral triangle (3-4-6) to determine the locations of the two final end points (5) and (6) for the curved track segment. For turns at a different angle, you can use a protractor (or a protractor app) to locate the two final ends (5) and (6) of the curved track segment. Turning radius variations are acceptable as long as the minimum turning radius requirement in Step 2 is met.

c. Put small tape segments, e.g. 4-inches each, on the floor to lay the curved border segments (7) and (8) and connect them with the straight-line borders. The two curved borders don't need to be parallel.



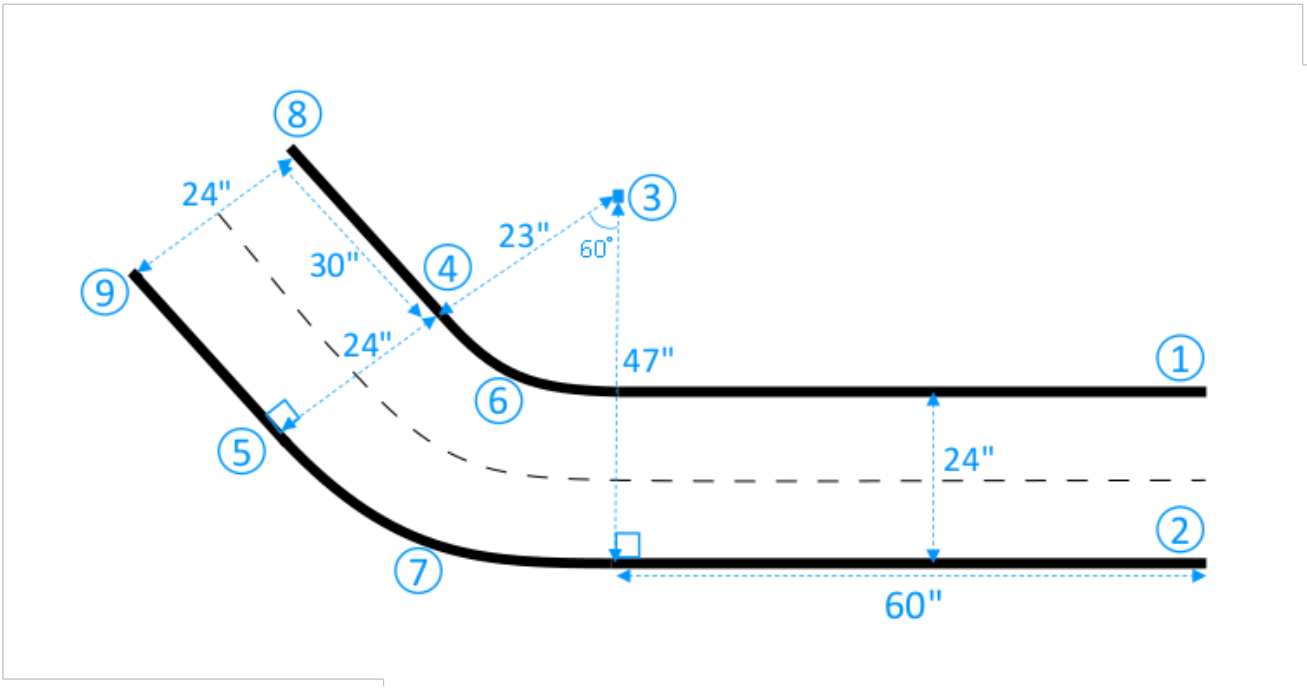
3. To extend the track with the next straight segment of 30 inches long and 24 inches wide, do the following:

   a. Put a 30-inch long piece of tape on the floor to lay down the first border (4-8) perpendicular to the edge (3-5).



b. Use the tape measure to locate the ending point of the second border (9). You can customize the length of the straight lines to fit to the space you have.

c. Put another 30-inch long piece of tape on the floor to lay down the second border (5-9) perpendicular to the edge (3-5).



We assume the second straight track segment is 30 inches long and 24 inches wide. You can adjust the length and width to fit to your space, provided that the dimensional requirements are met and the dimensions are consistent with other track segments.

4. Optionally, cut tape segments of 4 inches long and then place the tape segments 2 inches apart along the track center to lay the dashed center line.

You've now finished building the single-turn track. To help your vehicle to better distinguish the drivable surfaces from non-drivable surfaces, you should paint the off-track surface a color of sufficient contrast with respect to the on-track surface color. To ensure safety, you could encircle the track with uniform-colored barriers that are at least 2.5 feet tall and 2 feet away from the track at all points.

You can apply the instructions to extend the track to more complex shapes.

AWS DeepRacer track design templates

The following track design templates show AWS DeepRacer tracks that you can build by following the instructions presented in this section.

> **Note**
>
> Templates for tracks that are available pre-printed are also presented in this section. The assembly of pre-printed tracks requires less time and is a simpler process than constructing tracks with your own materials. We recommend using pre-printed tracks and barriers. To purchase pre-printed tracks, see AWS DeepRacer storefront.

For all tracks, to reproduce the same color production, use the following specifications:

- Green: PMS 3395C
- Orange: PMS 137C
- Black: PMS 432C
- White: CMYK 0-0-2-9

These tracks were tested with the following materials for their surfaces:

- Vinyl

  The tracks were printed on 13-ounce scrim vinyl with a matte finish to reduce glare. Vinyl is typically cheaper than carpet and provides good performance. Vinyl is not as durable as carpet.
- Carpet

  The tracks were printed on 8-ounce, dye-sublimated, polyester-faced carpet with latex rubberized backing. Carpet is durable and provides great performance, but is expensive.

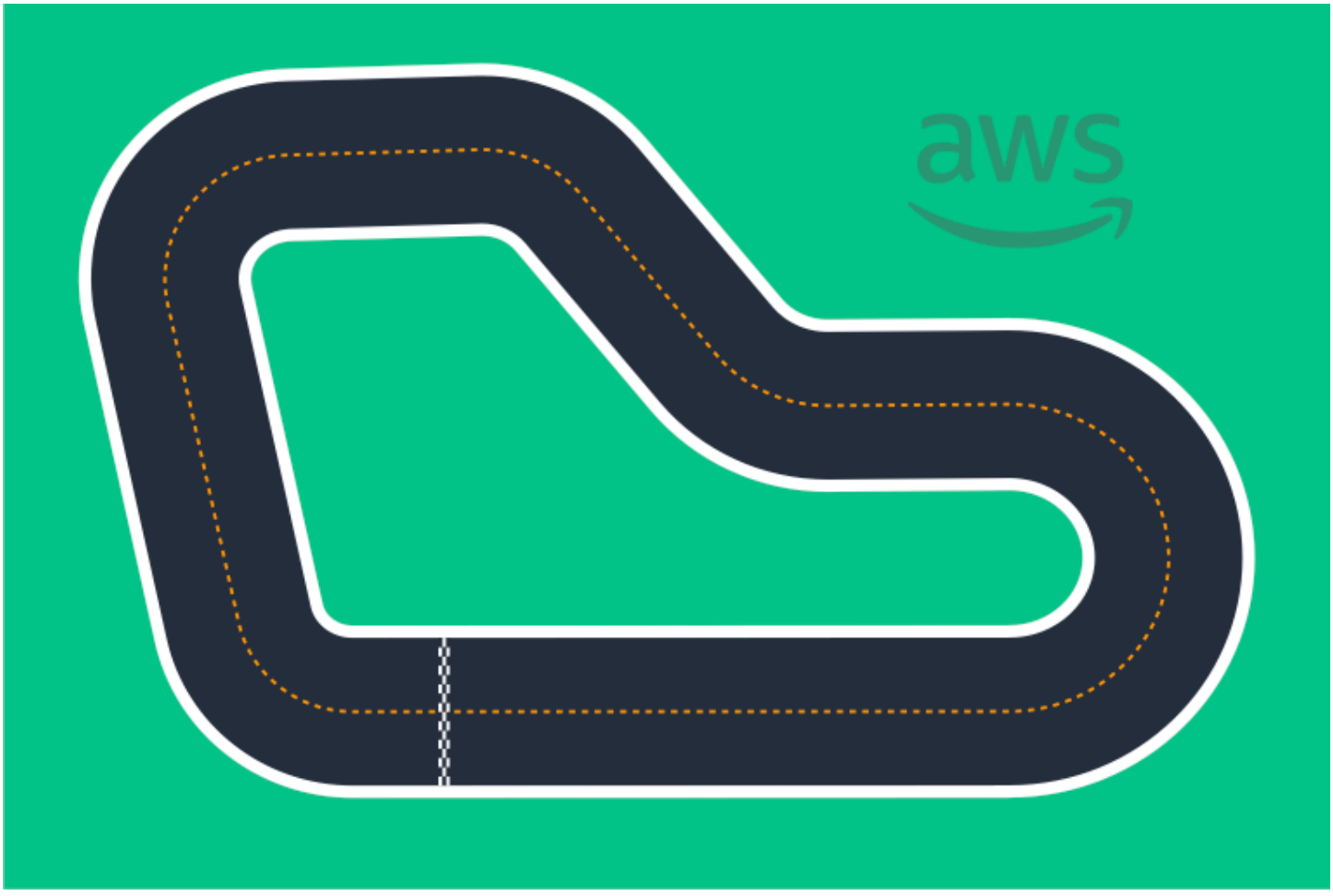
Due to their large size, the tracks cannot be easily printed on a single piece of material. Align track lines well when connecting pieces together.

Topics

- AWS DeepRacer RL Speedway (Advanced) track template
- AWS DeepRacer Single-turn track template
- AWS DeepRacer S-curve track template
- AWS DeepRacer Loop track template

AWS DeepRacer A to Z Speedway (Basic) track template

The AWS DeepRacer A to Z Speedway (Basic) track is the most popular physical competition track in AWS DeepRacer history. It was originally released at AWS re:invent 2018 and has the smallest footprint of all the AWS DeepRacer physical competition tracks. It's available pre-printed for purchase at AWS DeepRacer Storefront.



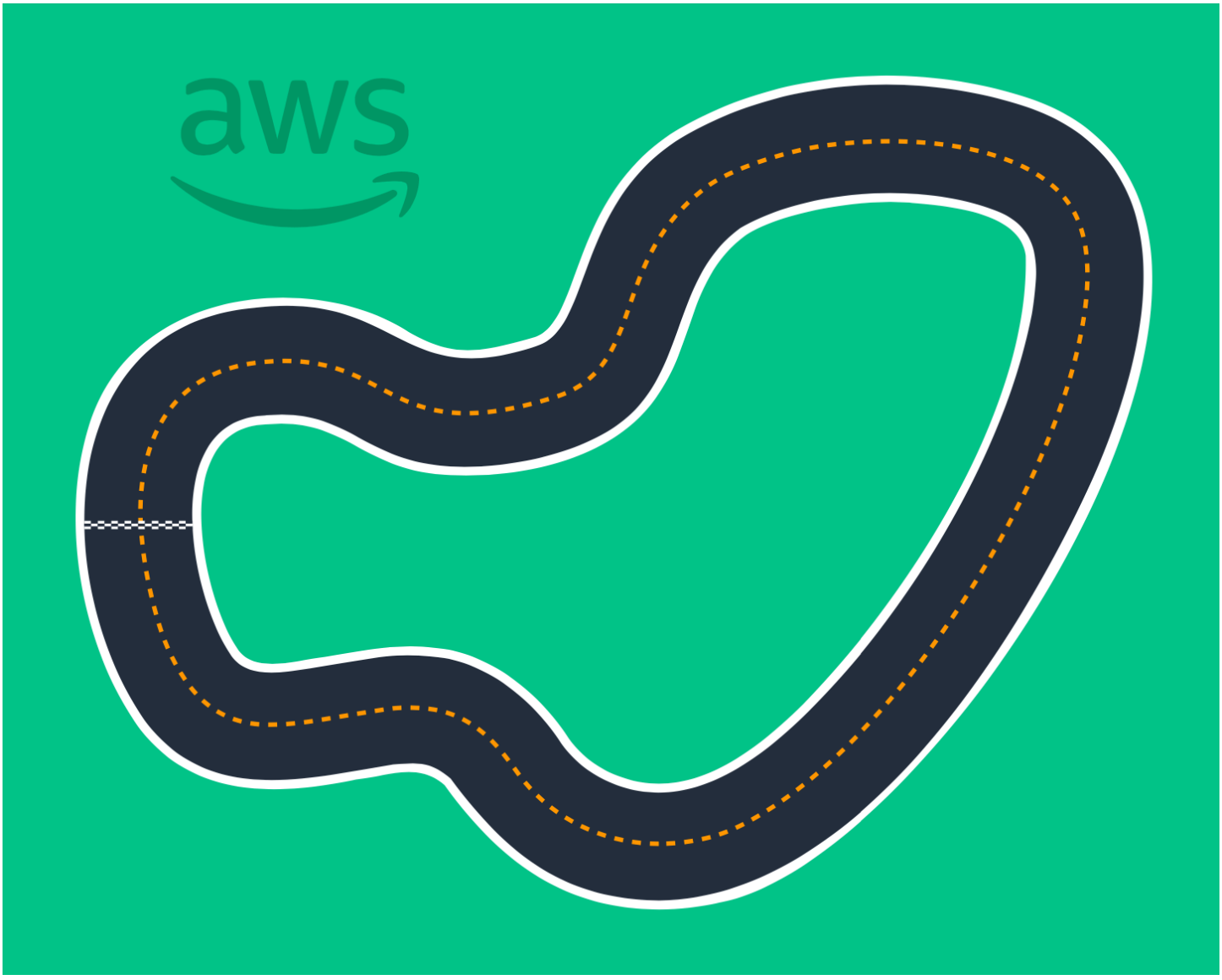
We recommend this track for beginner events and first-time racers. With a variety of runs and straightaways, it offers a compelling challenge for both first-time and experienced racers. The AWS DeepRacer A to Z Speedway (Basic) track is a 1:1 physical reproduction of the virtual track available in the console. It provides racers the opportunity to train a model in a virtual environment and

then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physical track.

To print or create your own A to Z Speedway (Basic) track, download this AWS DeepRacer A to Z Speedway (Basic) file.

AWS DeepRacer Smile Speedway (Intermediate) track template

The AWS DeepRacer Smile Speedway track was originally released as the AWS DeepRacer Championship 2019 track. It's available pre-printed for purchase at AWS DeepRacer Storefront.



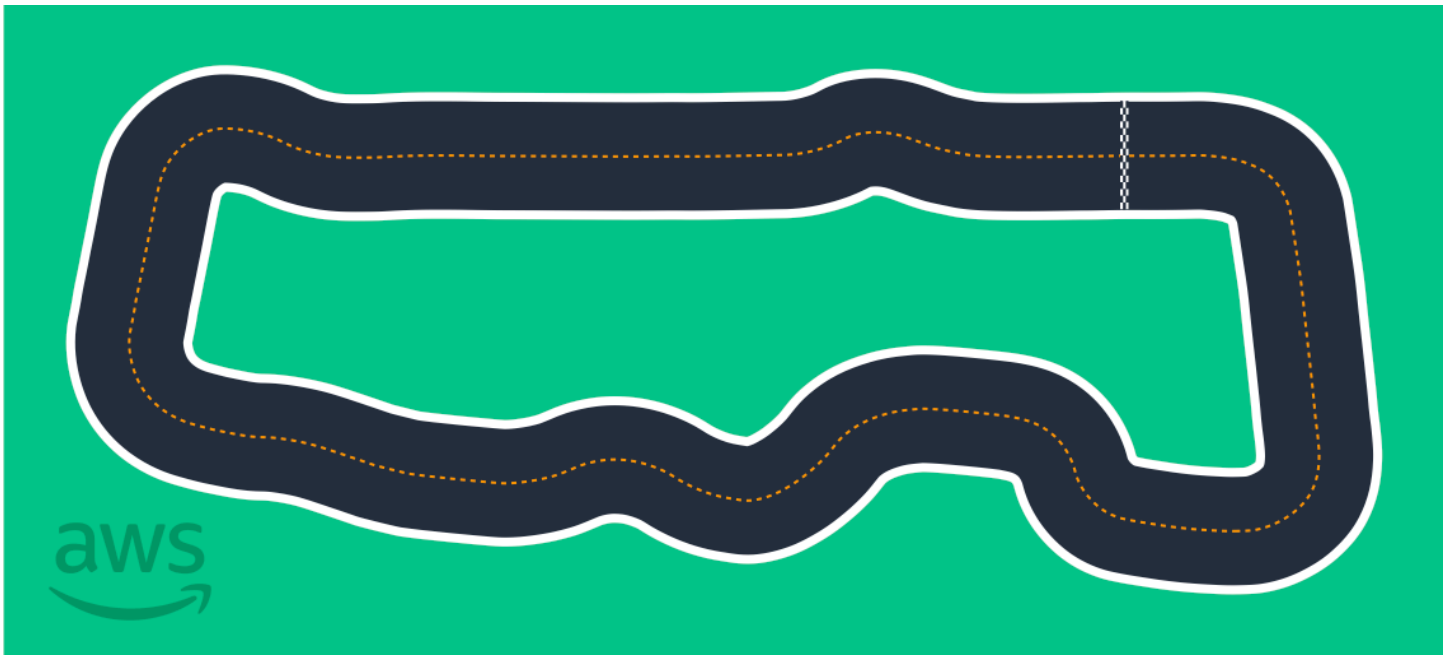
We recommend this intermediate track for events with experienced racers and larger physical spaces. It's a 1:1 physical reproduction of the virtual track available in the console. It provides

racers the opportunity to train a model in a virtual environment and then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physical track.

To print or create your own AWS DeepRacer Smile Speedway (Intermediate) track, download this AWS DeepRacer Smile Speedway (Intermediate) track file.

AWS DeepRacer RL Speedway (Advanced) track template

The AWS DeepRacer RL Speedway (Advanced) track (aka AWS DeepRacer Summit Speedway) was originally released for AWS DeepRacer summits in 2022 and is the longest physical track in AWS DeepRacer history. It's available pre-printed for purchase at AWS DeepRacer Storefront.

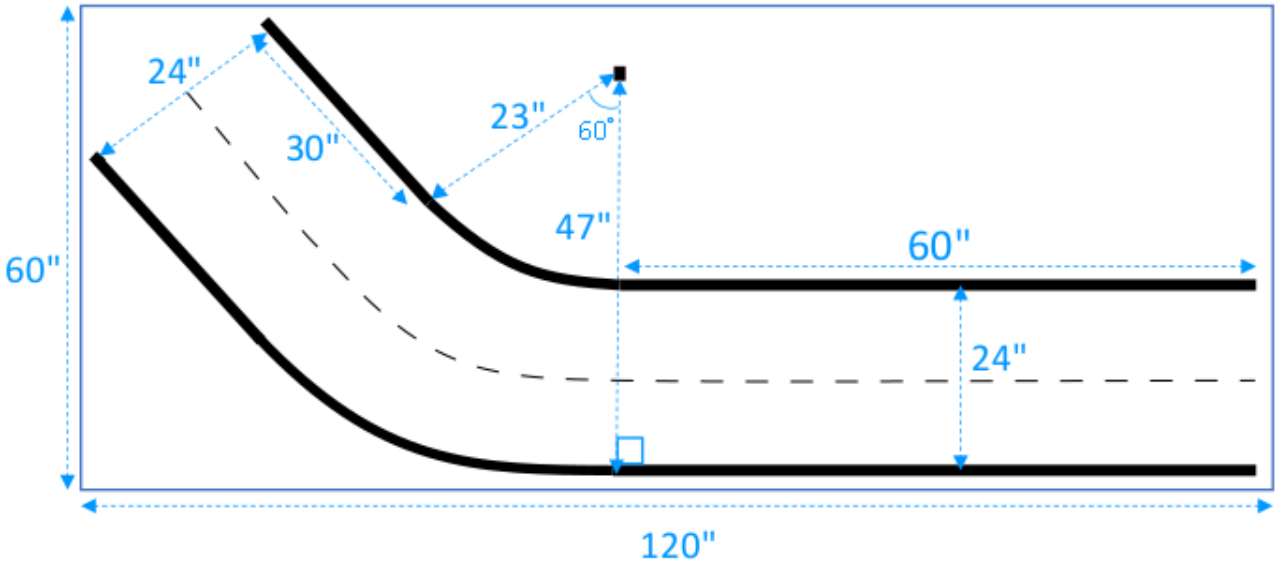


We recommend the AWS DeepRacer RL Speedway (Advanced) track for events with experienced racers. It offers a compelling challenge for racers who enjoy going fast on straightaways. The AWS DeepRacer RL Speedway (Advanced) track is a 1:1 physical reproduction of the virtual track available in the console. It provides the opportunity for racers to train a model in a virtual environment and then deploy the model to a physical AWS DeepRacer device for autonomous racing on a physcial track.

To print or create your own AWS RL Speedway (Advanced) track, download this AWS DeepRacer RL Speedway (Advanced) track file.

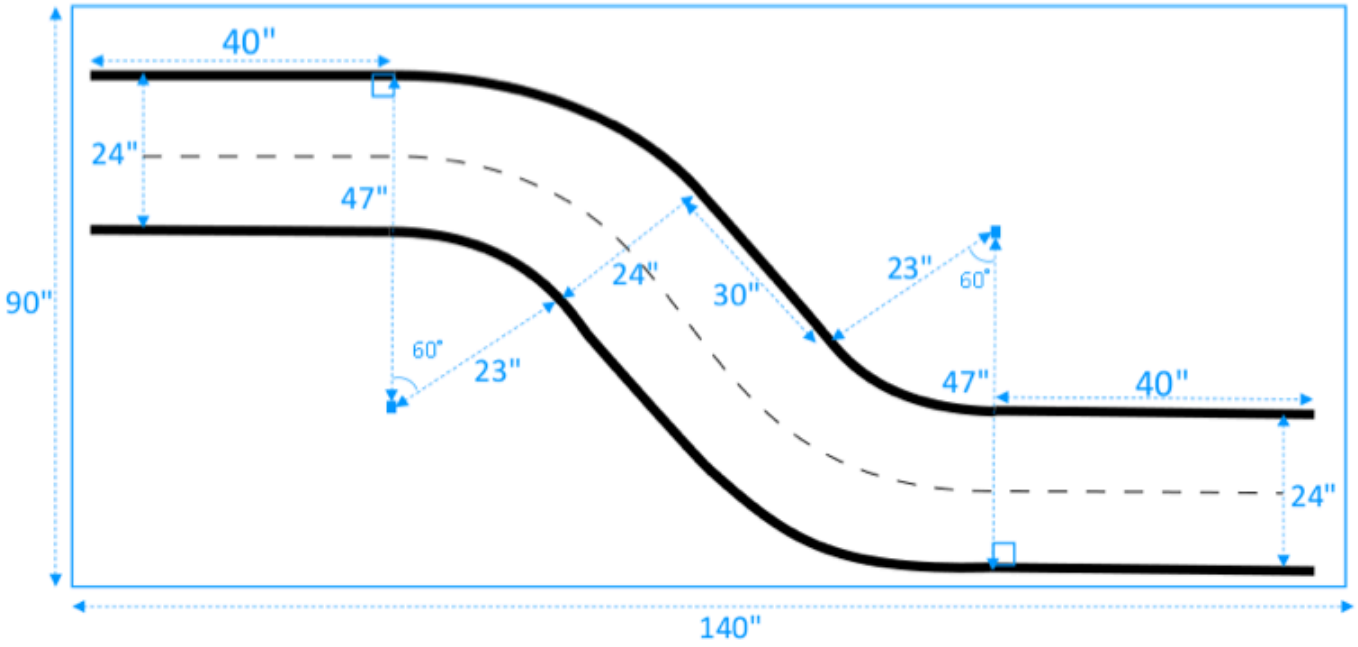
AWS DeepRacer Single-turn track template

This basic track template consists of two straight track segments connected by a curved track segment. Models trained with this track should make your AWS DeepRacer vehicle drive in straight line or make turns in one direction.



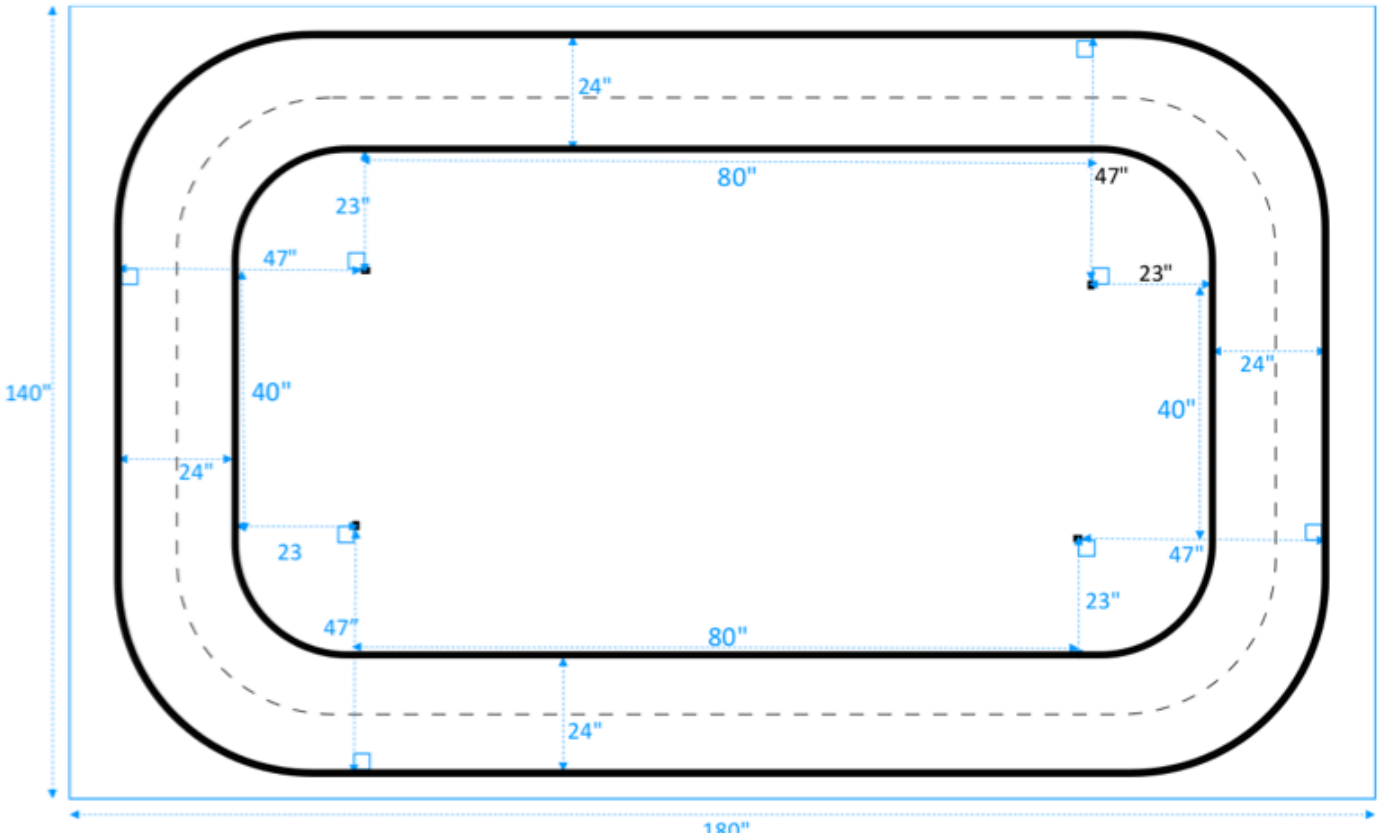
AWS DeepRacer S-curve track template

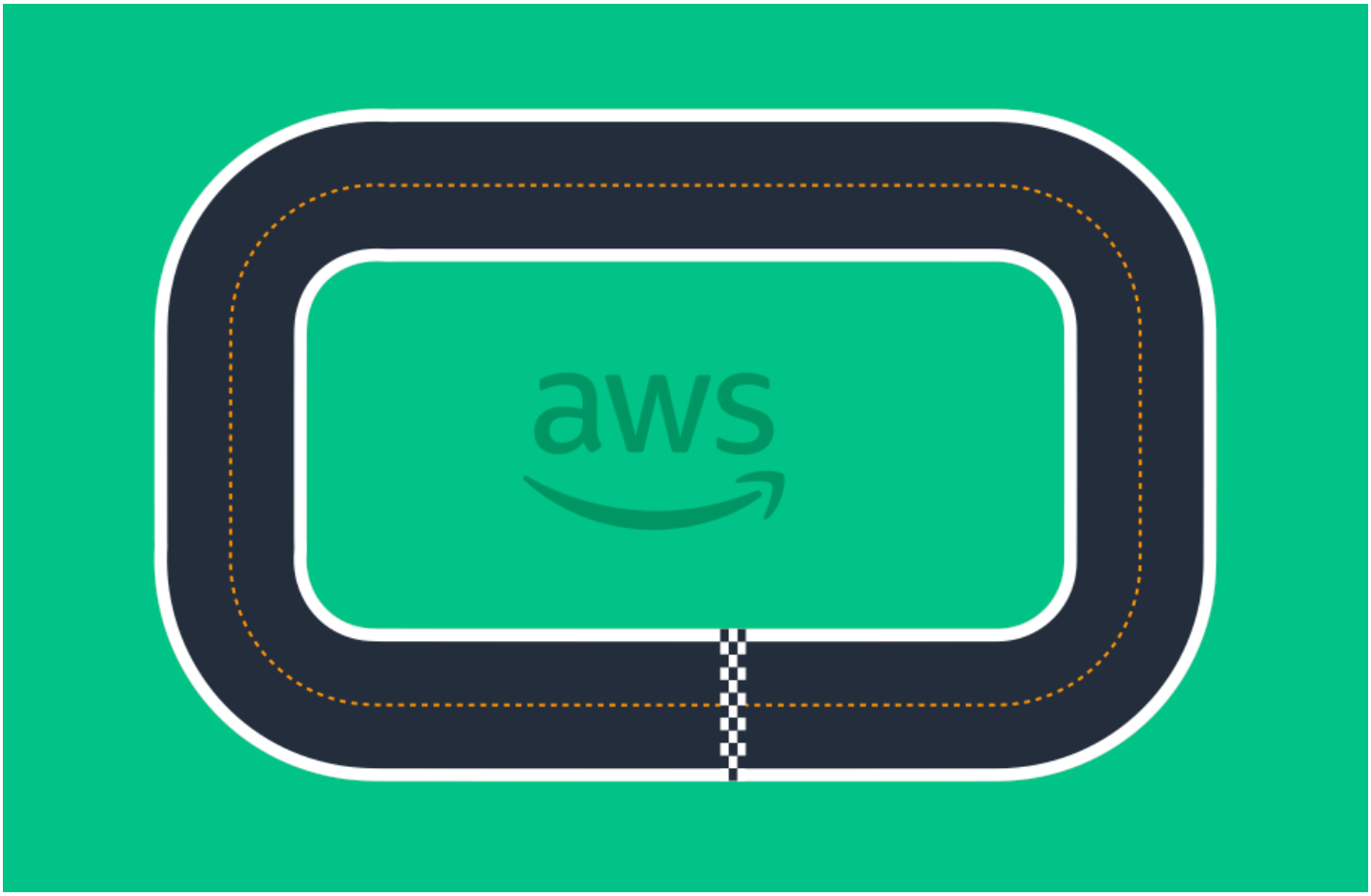
The track is more complex than the single-turn track because the model needs to learn to make turns in two directions. You can easily extend the single-turn track construction instructions to this track by turning it in the opposite direction after the first turn.



AWS DeepRacer Loop track template

This regular loop track is a repeating, 90-degree, single-turn track. It requires a larger enclosing area for laying the entire track.


24"
80"
47"
23"
47"
23"
140"
40"
24"
40"
24"
23
47"
23"
47"
80"
24"
180"


aws

Join an AWS DeepRacer race

After successfully training and evaluating your model in simulation, compare your model's performance to other racers' models by participating in a race. Racing is a fun way to get feedback about your model, win awards and prizes, meet other AWS DeepRacer community members, hear about opportunities to learn and improve your skills, and have fun.

Races can be in-person or online (virtual) and virtual races can be formatted synchronously as *LIVE* races or asynchronously as *classic* races. LIVE and classic virtual races can be broadcast privately or publicly.

This section discusses how to participate in an AWS DeepRacer League Virtual Circuit race or a community-based virtual race and your different options for formatting.

AWS DeepRacer racing event types

An event can be categorized by its sponsor or organizer. Both AWS DeepRacer League and community racing events can take place in person on a physical track or online on a virtual track.

- **AWS-sponsored racing events** – Racing events sponsored by AWS are referred to as AWS DeepRacer League events and are open to any AWS DeepRacer users. First-time racers can start their league journey by joining a monthly virtual race. Once a racer has submitted a model to the race, they earn points and will receive their national and regional season standings.

- **Community-sponsored racing events** – Racing events created by AWS DeepRacer users are called community racing events.

Joining an online AWS-sponsored or community-sponsored race

You can use the AWS DeepRacer console to enter an AWS DeepRacer League Virtual Circuit event or a community-based online race.

- Any AWS DeepRacer user can join a AWS DeepRacer League Virtual Circuit online race.

- Only invited users can access or participate in community racing virtual events. Users are invited when they receive an invitation link sent by the race organizer or forwarded by another race participant.

Topics

Join an AWS DeepRacer League Virtual Circuit race

In this section, learn how to use the AWS DeepRacer console to submit your trained model to a Virtual Circuit race.

To enter the AWS DeepRacer League Virtual Circuit

1. Sign in to the AWS DeepRacer console.
2. From the main navigation pane, choose **AWS Virtual Circuit**.
3. On the **AWS Virtual Circuit** page, under the **Open races** section, choose **Enter race**.
4. If this is your first time participating in an AWS DeepRacer League racing event, set your alias in **Racer name** under **AWS DeepRacer League racer name**.
5. Under **Choose model**, select the model you want to use from the **Model** list. Ensure that your model has been trained to handle the track shape.
6. If this is your first time participating in an AWS DeepRacer League event, under **League requirements**, select your **Country of residence**. Once you select your country of residence and submit your first model, it is locked in for the racing season and will be verified when prizes are awarded. Then, accept the terms and conditions by selecting the checkbox.
7. Choose **Enter race** to complete the submission. The submission quota for each race is 50.

   After your model is submitted, the AWS DeepRacer console starts its evaluation. The process can take up to 10 minutes.
8. On the race page, review the race details.

9. On the race page, note your submission status under your racer name.
10. On the race page, view the ranking list on the leaderboard to see how your model compares with others.

    If your model doesn't finish the track in three consecutive trials, it is not included in the ranking list on the leaderboard. Your leaderboard ranking reflects your best performing submission. You also receive a national and regional season standings to gauge where you rank amongst other racers in your country and region.

    After you submit a model, try improving its performance by refining your reward function and iterating on your model. You can also train a new model with a different algorithm or action space. Learn, adjust, and race again to increase your chances for rewards.

To join a AWS DeepRacer community race

> **Note**
>
> To join an AWS DeepRacer community race, you first need to receive a link to the race from the race organizer.

When you receive an invitation to join an AWS DeepRacer race, find out whether it's a *LIVE* or *classic* race.

Classic race

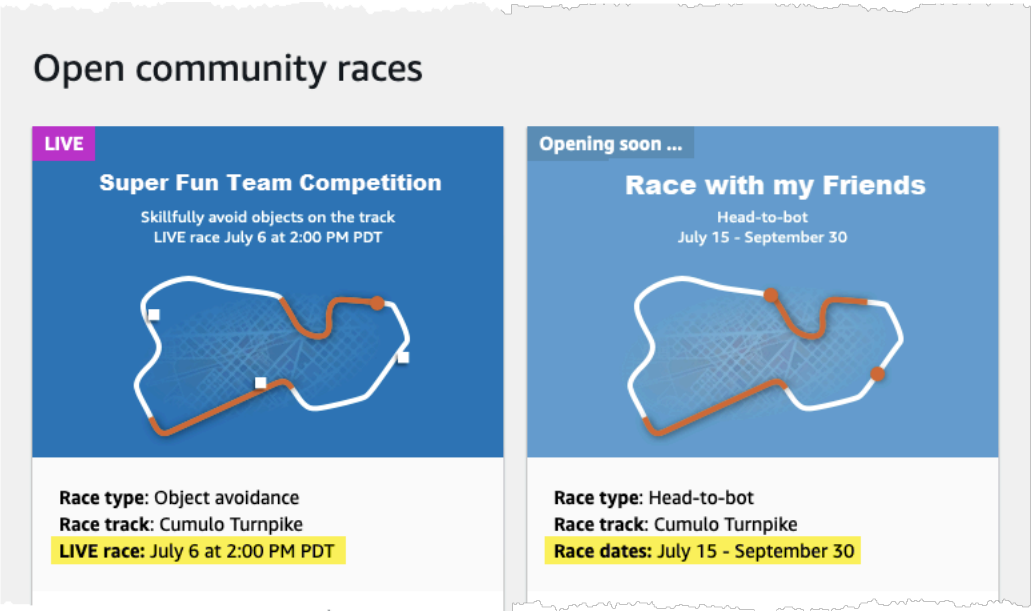
Classic races are asynchronous events that do not require real-time interaction. Your invitation link gives you access to submit a model to the race and view the leaderboard. You can submit unlimited models at any time within the race's opening and closing dates to achieve your best standing on the leaderboard. Results and videos for classic races are viewable for submitted models on the Leaderboard page as soon as the race is initiated. All classic races are private events.

LIVE race

LIVE races are real-time racing events where you gather virtually with other racers taking turns to compete for the fastest time on the leaderboard. You can enter multiple models, but only the last model you submit before the submission window closes will be used. During your race, you have the option to try interactive speed controls, which temporarily override your model's

speed parameters giving you the opportunity to make strategic, real-time adjustments. LIVE races can be broadcast privately among invited racers or publicly for anyone to see.

If the competition format is not specified in your invitation, check your race card. LIVE races say, "LIVE," and tell you the date and time of the synchronous event. Classic races give you the date range for the asynchronous competition.



Join an AWS DeepRacer community race as a race participant

If you're new to AWS and receive an invitation to join an AWS DeepRacer community race, follow the steps in **To join as a new user**. If you're invited to an active community race and you've entered an AWS DeepRacer race before, follow the steps below in **To join a Classic race** or **To join a LIVE race** as is appropriate for your competition format.

To join as a new user

If you're new to AWS and receive an invitation link to join an AWS DeepRacer community race, choose the link to go to the AWS DeepRacer console and then sign up for an AWS account before proceeding to join the race.

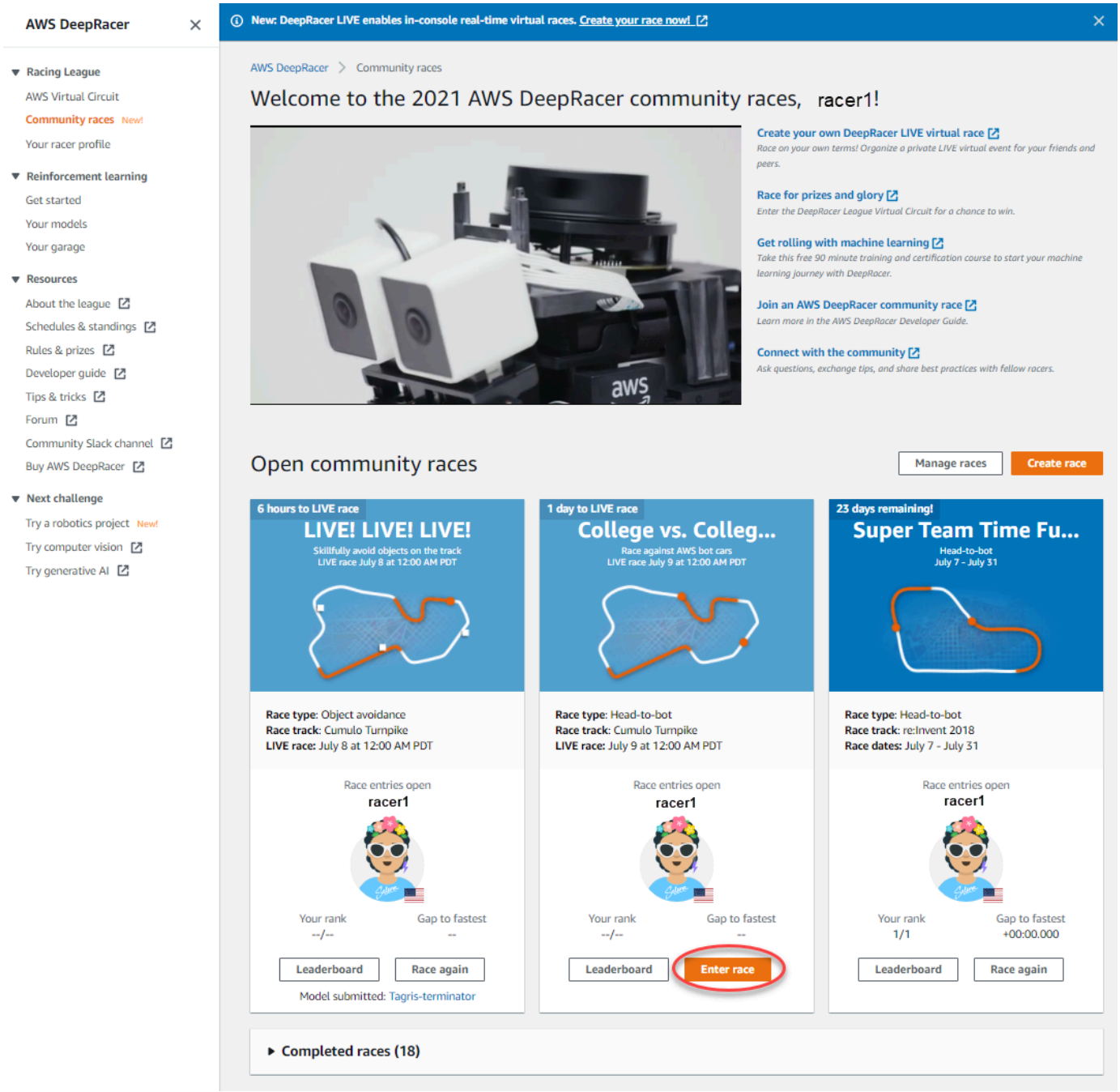
As a new AWS DeepRacer user or a first-time participant to any AWS DeepRacer race, follow the steps to join a community race in the AWS DeepRacer console.

To join a race as a new user

1. Create an AWS account in the AWS DeepRacer console.

2. Once you are set up and signed in, choose the link shared with you by the race organizer to open the Race page.
3. When prompted to create an **AWS DeepRacer racer name**, enter a name that you will use as identification across all the AWS DeepRacer leaderboards. Once you choose a racer name, you cannot change it.
4. On the Race details page, expand **Get started racing**.
5. Choose **Get started with RL** to get a quick introduction to training an AWS DeepRacer model for autonomous driving.
6. Train and evaluate your model for the race in the AWS DeepRacer console.

   For more information on training your model, see Train your first AWS DeepRacer model.
7. Navigate to **Community races**.
8. Find the race you're invited to. Choose **Enter race** on the race card.

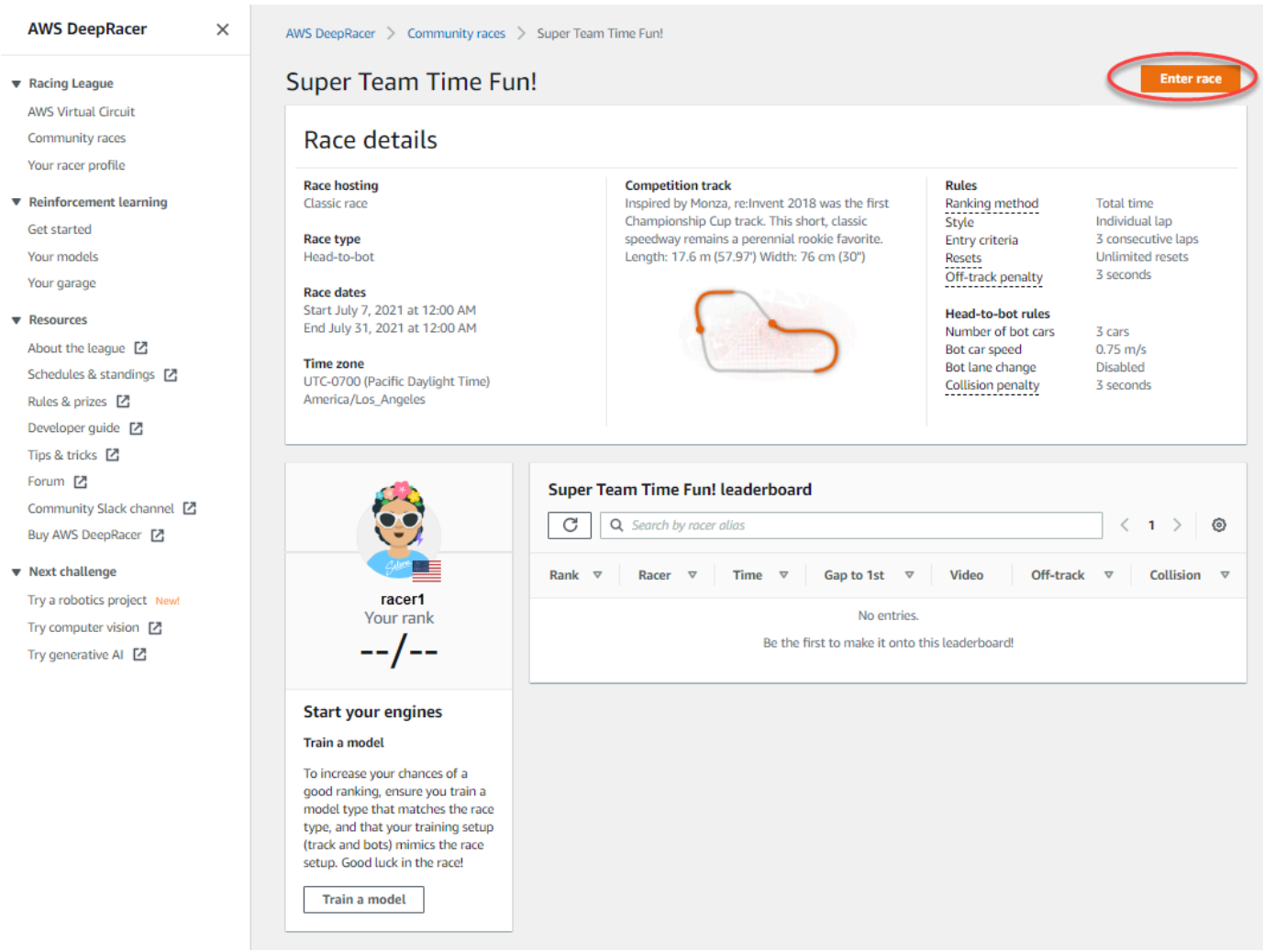


9. Follow the steps in **To join a Classic race** or **To join a LIVE race** as is appropriate for your race's competition format.

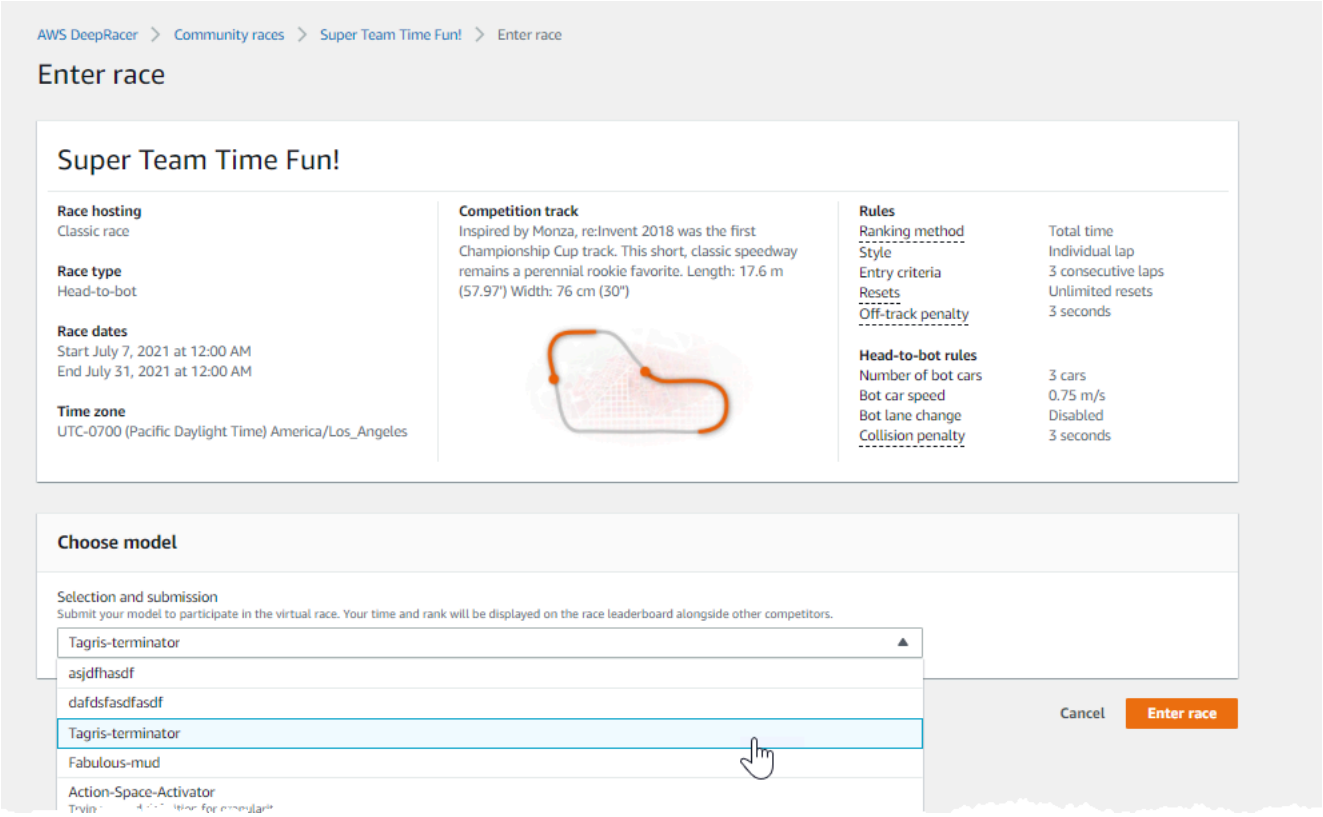
To join a classic race

1. Select the link you received from the race organizer. If you're not already signed in to your account in the AWS DeepRacer console, you'll be prompted to sign in.

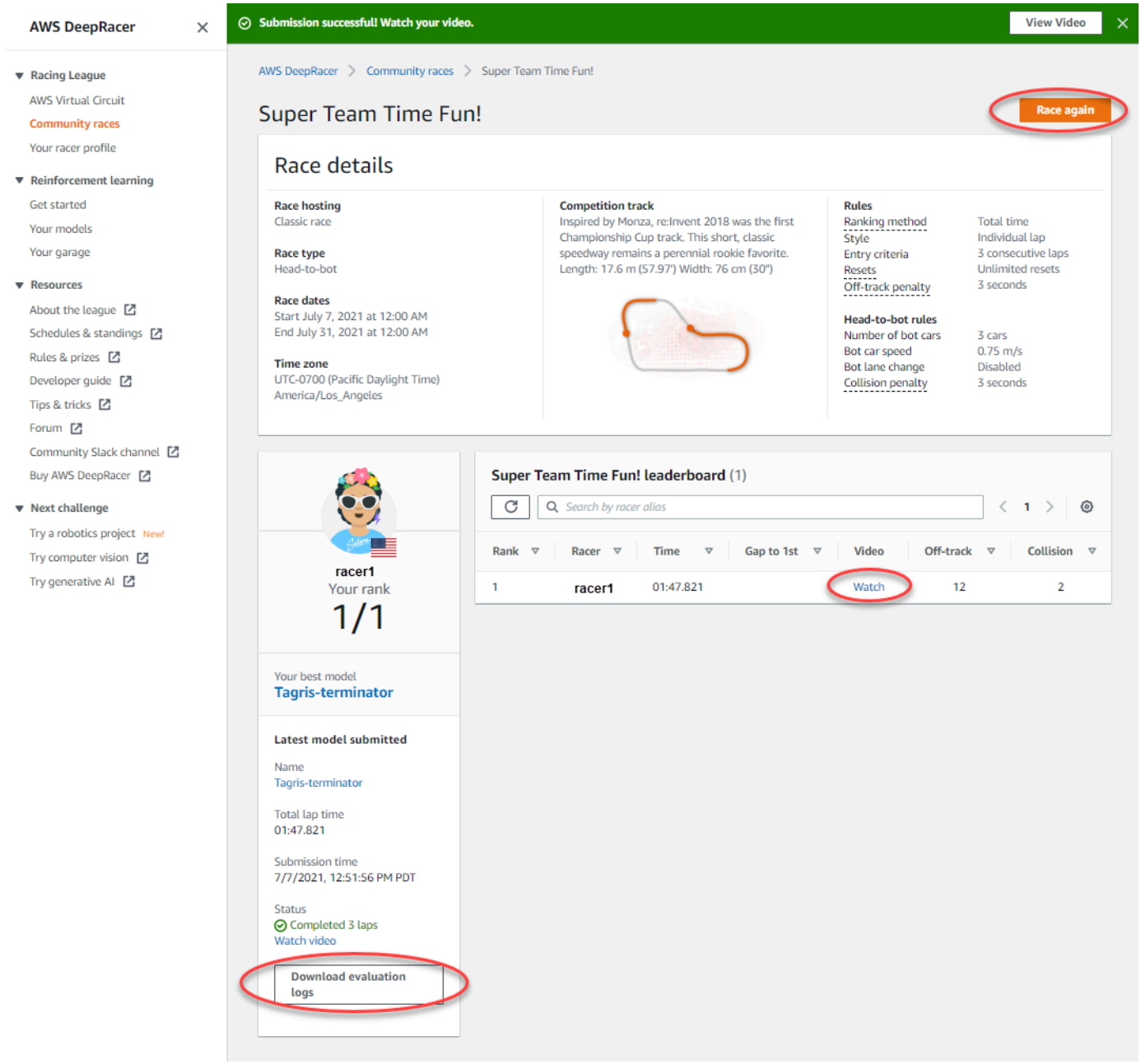
2. Once signed in to the AWS DeepRacer console, the link will take you to the Race page. The Race page displays the race details, leaderboard, and your racer info. Choose **Enter race**.



3. On the **Enter race** page, under **Choose model**, choose a trained model and then choose **Enter race**.



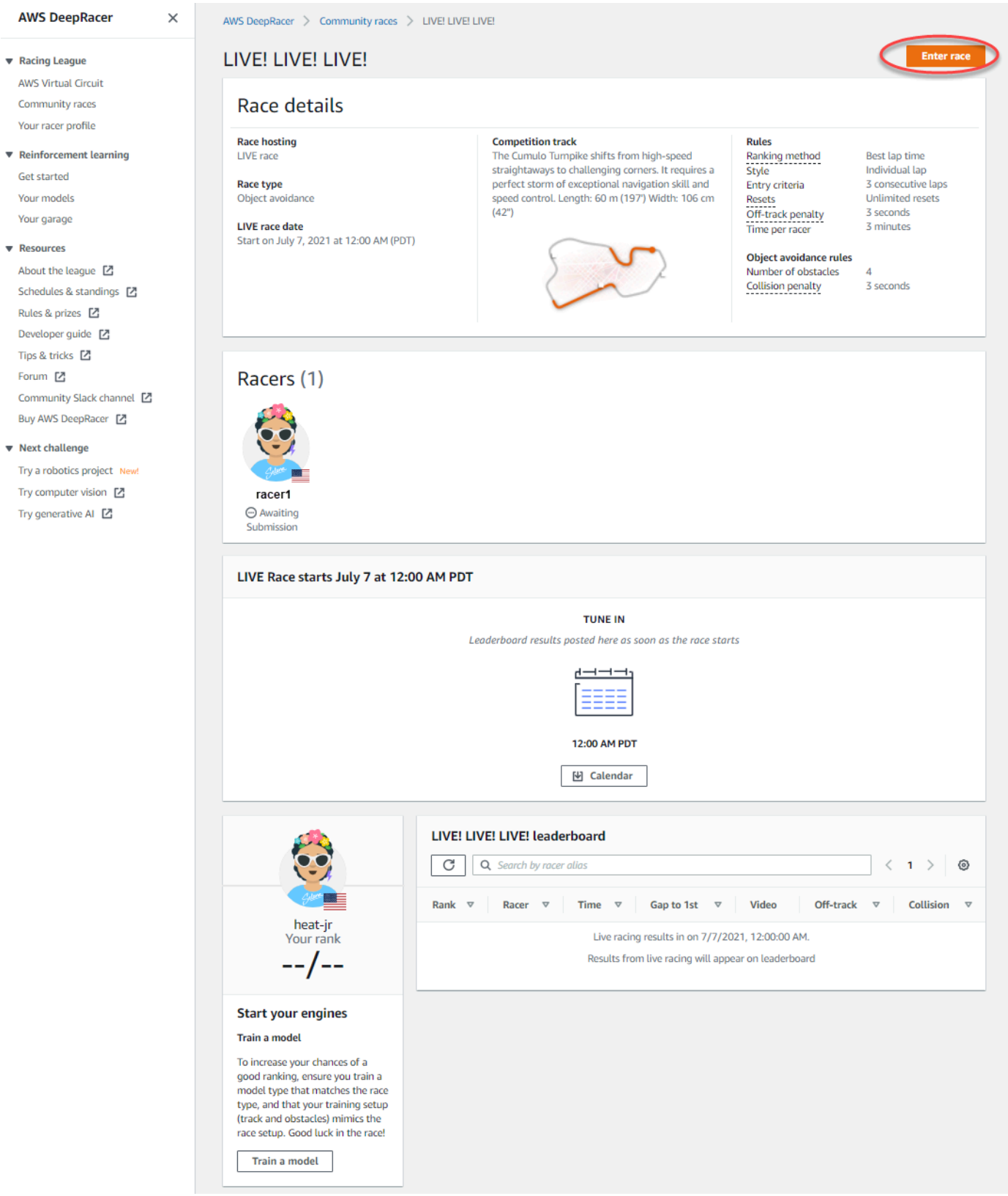
4. If your model is evaluated successfully against the racing criteria, watch the event's leaderboard to see how your model ranks against other participants.
5. Optionally, choose **Watch** to view a video of your vehicle's performance or choose **Download evaluation logs** to review a detailed look at the outputs produced.



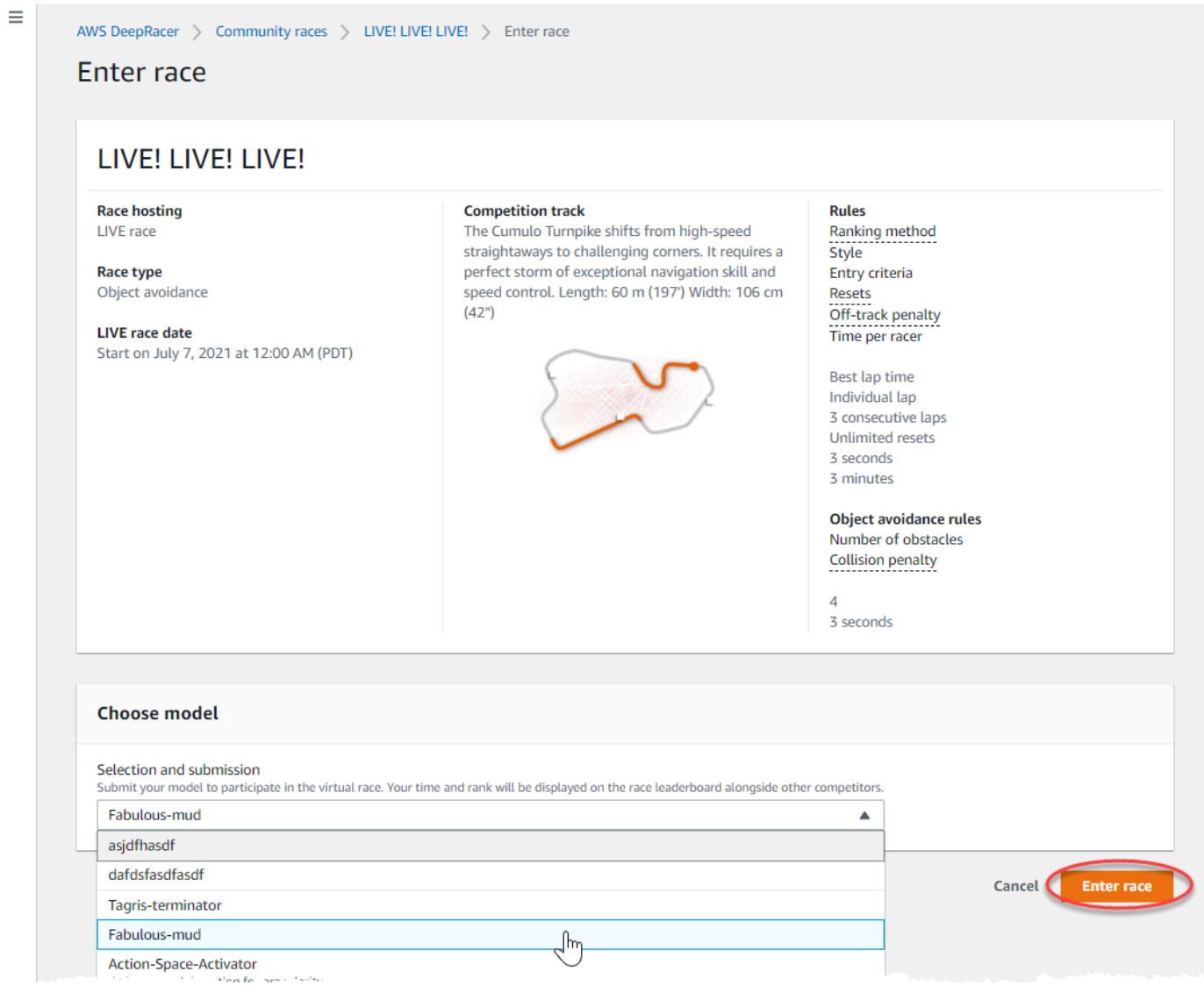
6. Choose **Race again** to enter another model. You can submit unlimited models at any time within the race's opening and closing dates to achieve your best standing on the leaderboard.

To join a LIVE race

1. Select the link you received from the race organizer. If you're not already signed in to your account in the AWS DeepRacer console, you'll be prompted to sign in.
2. Once signed in to the AWS DeepRacer console, the link will take you to the Race page. The Race page displays the race details and leaderboard. Choose **Enter race**.



3. On the **Enter race** page, under **Choose model**, choose a trained model and then choose **Enter race**.



4. If your model is evaluated successfully against the racing criteria, watch the event's leaderboard to see how your model ranks against other participants.
5. Optionally, for LIVE races, select **Calendar** to add the LIVE racing event to your calendar.
6. Choose **Race again** to enter another model. You can enter multiple models, but only the last model you submit before the submission window closes will be used.

Participate in an AWS DeepRacer LIVE race

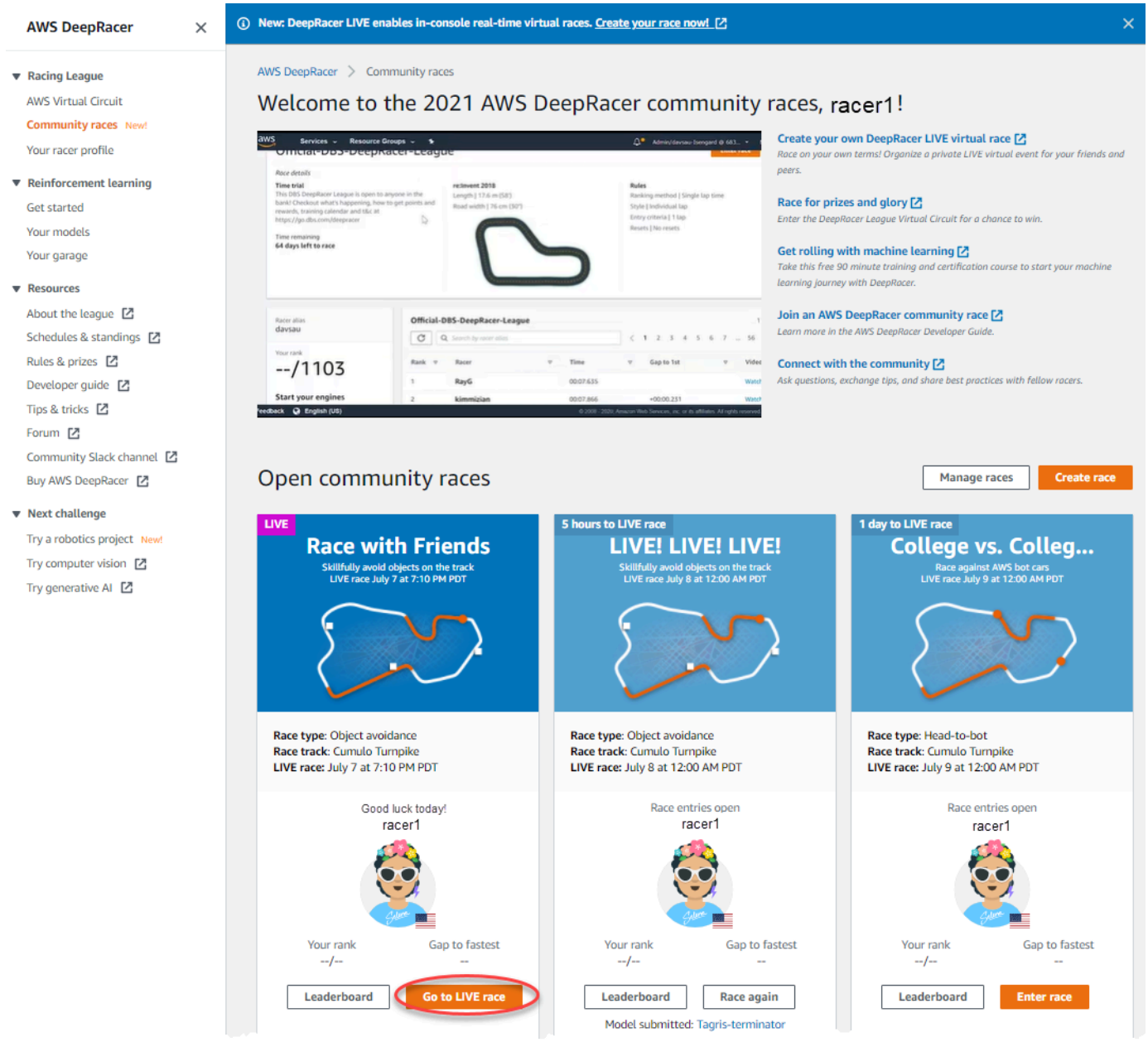
> **Note**
>
> Submit your model at least one hour prior to the LIVE race start time. You can enter multiple models, but only the last model you submit before the submission window closes will be used.

Before you start

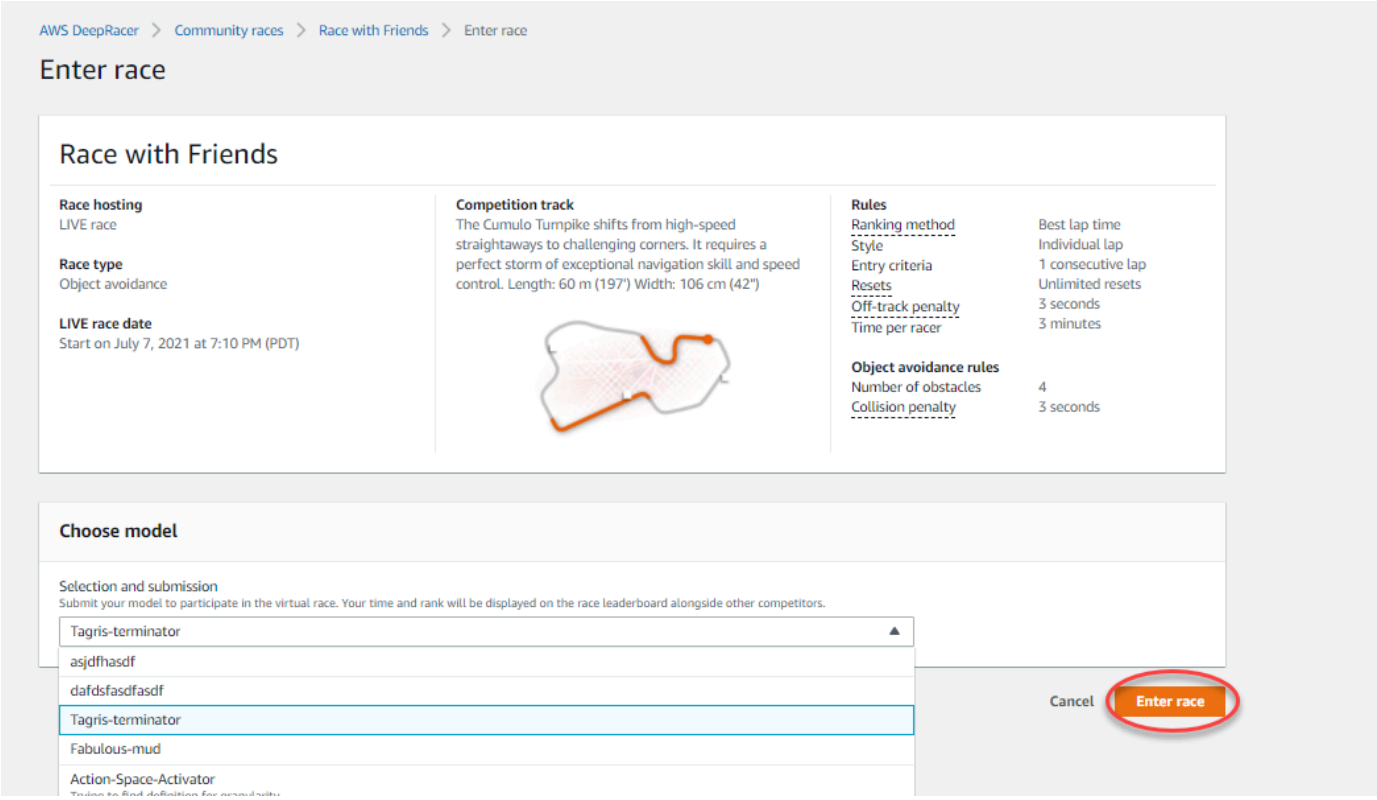
- Use a Chrome or Firefox browser (Check that your browser is up to date).
- Disconnect virtual private network (VPN) if you're using one.
- Close all extra tabs.

To participate in a LIVE race

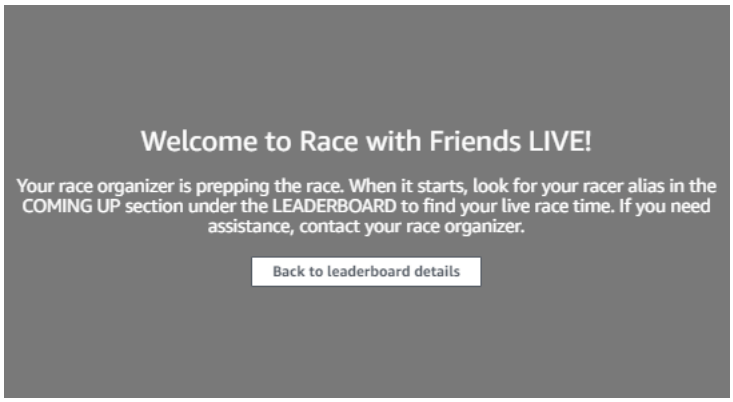
1. Sign in to the AWS DeepRacer console.
2. If you haven't submitted a model, find the race card for the race you want to participate in and select **Go to LIVE race**.



3. On the **Race** page, select **Enter race**.
4. On the **Enter race** page, under **Choose model**, select the model you want to submit from the drop down menu and choose **Enter race**.



5. On the **Race** page, choose **Go to LIVE race**.
6. On the **LIVE race** page, you'll see a wait message. Navigate to the conference bridge provided to you by your race organizer.

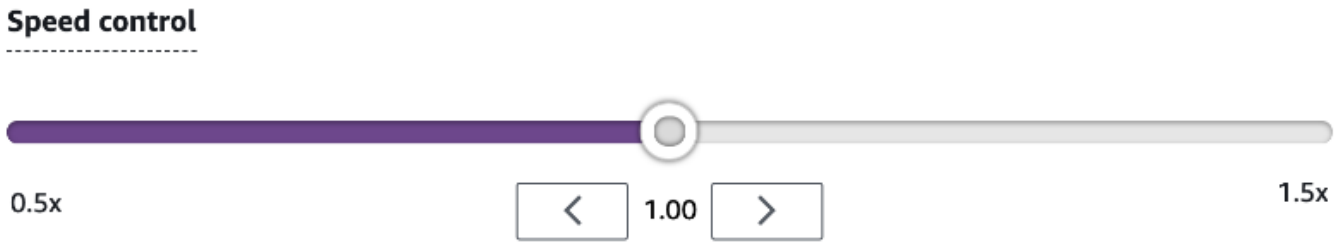


7. Check in with your race organizer, who will review the race rules and answer racer questions.
8. Check the **COMING UP** section under **LEADERBOARD** for your live race time and be ready when the race organizer announces that you are up next.
9. On your turn, there will be a **10, 9, 8, 7, 6...** countdown animated in the console when the race organizer launches your race. On **Go!** you will have access to the optional speed control. To choose key moments to boost or slow down your model's speed. There are three ways to operate the Speed control feature:

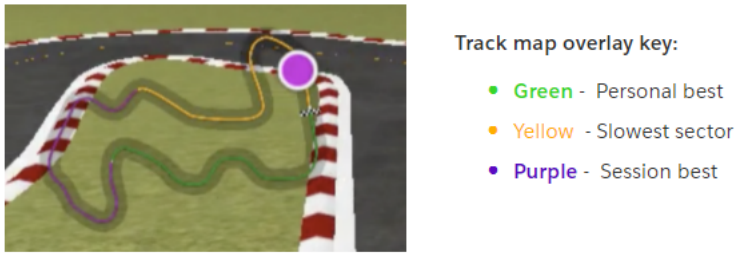
a. Drag the slider with your computer's mouse.

b. Alternatively, choose the **< / >** arrow buttons in console.

c. You can also select the slider knob to activate the slider and then use your **#** and **#** keyboard arrow keys.



10. Reset the multiplier to 1 to return to using your model's speed parameters.
11. As you race, check the video overlay of your LIVE race to help optimize your performance. The track map overlay is divided into three sectors that change color depending on your pace. Green indicates the section of the tack where you clocked a personal best, yellow denotes the slowest sector driven, and purple signifies a session best. You can also find statistics detailing your best lap time, time remaining speed in m/s, resets, and current lap time.



12. The race ends when you see the checkered flag icon in the console. The Speed control is disabled and a replay of your race launches on the video screen. You are ranked on the leaderboard by your single best lap time.

Organize an AWS DeepRacer community race

Community races are races organized by AWS DeepRacer users who are not officially sponsored by AWS.

You can create your own community race and invite your colleagues, classmates, or friends by sharing a race invitation link.

If you want to organize a race for students, see the Educator tools for AWS DeepRacer Student.

Topics

Create a virtual community race: a quick start guide

You can set up a virtual race quickly using the default community race settings. When you're ready to learn about all of your options, go to the section called “Customize a race”.

Before creating any virtual race, consider whether a *Classic* or *LIVE* race will be the best fit for your group and, if you choose a LIVE race, whether you are sharing it privately or publicly?

Classic race

Classic races are asynchronous events that do not require real-time interaction. Participants must receive an invitation link to submit a model to the race and view the leaderboard. Racers can submit unlimited models any time within a date range to climb the leaderboard. Speed controls are not available. Results and videos for classic races are viewable for submitted models on the Leaderboard page as soon as the race is initiated. All classic races are private events.

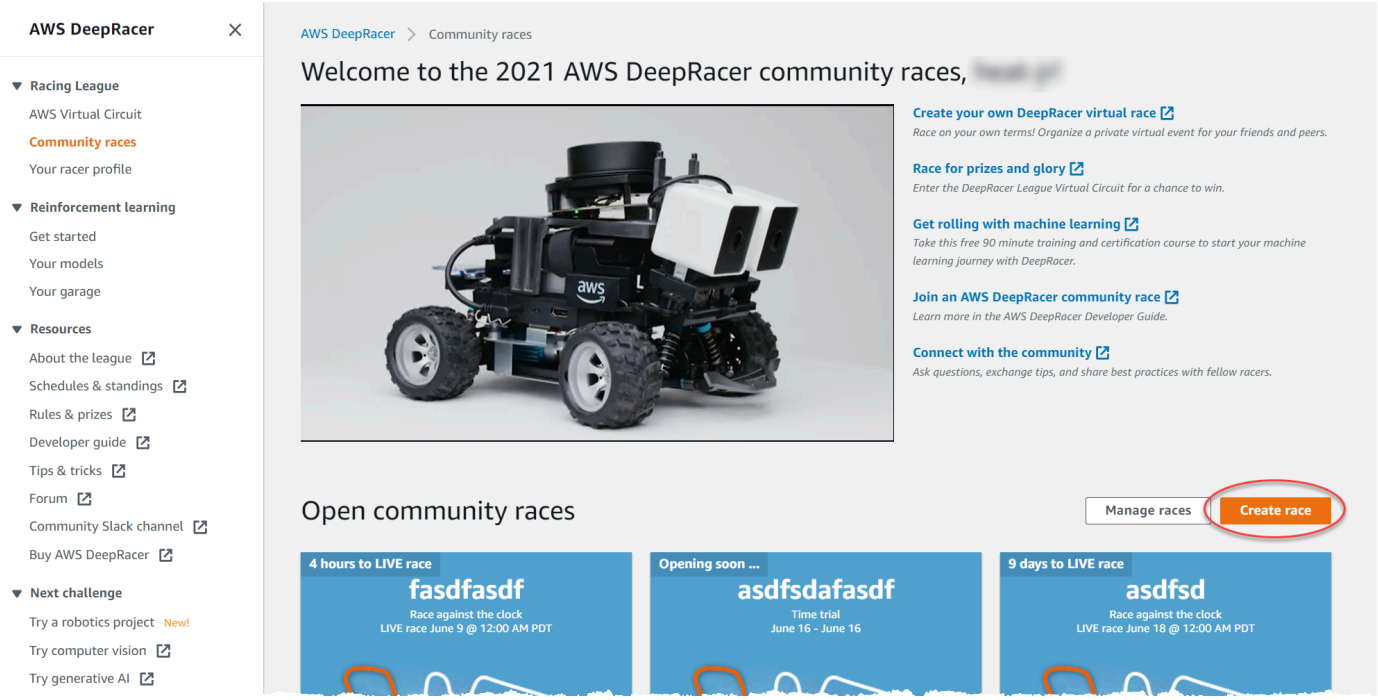
LIVE race

LIVE races are synchronous events that occur at a set time and range in scope from small events with one race organizer facilitating one private video conference to large events broadcast

publicly by a small team of organizers, commentators, and broadcasters. You can open and close the door for model submission at any time, so let racers know the deadline. Participants can submit multiple models, but only the last model they submit before you close the door can race during the event. During LIVE races, queued participants have the option of using interactive speed controls to give their model a competitive edge on their turn. Participants in LIVE races must also receive an invitation link to submit a model to the race, but you can choose to broadcast the event privately to invited participants only or publicly using a LIVE streaming service like Twitch. See the section called “Broadcast a LIVE race” to learn more.

To begin creating a community race

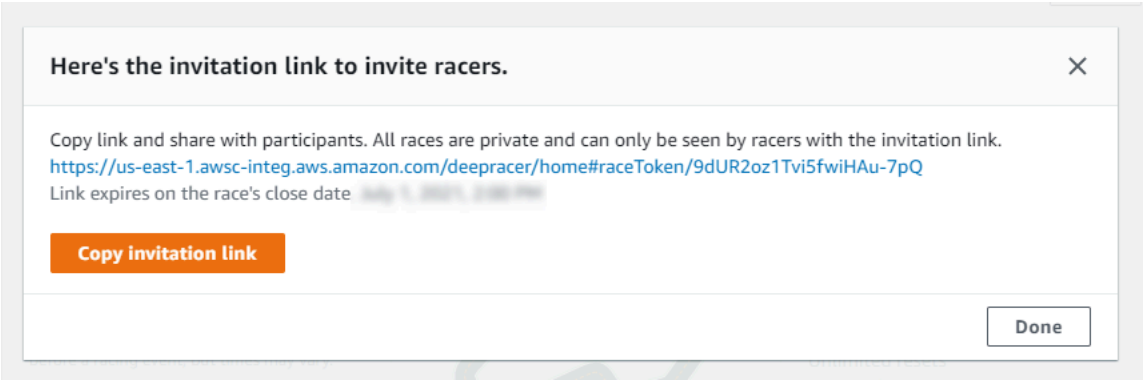
1. Open the AWS DeepRacer console.
2. Choose **Community races**.
3. On the **Community races** page, choose **Create race**.



4. On the **Race details** page, choose a competition format: a **Classic race**, which your guests can participate on their own schedule within the time frame you set, or a **LIVE race**, which can be broadcast privately or publicly as a real-time event.

To continue creating a classic race

1. Choose a race type. Race types increase in complexity from **Time trial** to **Object avoidance** to **Head-to-bot**. For first time racers, we recommend **Time trial**. Time trial races require only one camera, so the sensor configuration is simpler, and reinforcement learning (RL) models trained for this type of race converge faster. For more information about race types, see Tailor AWS DeepRacer Training for Time Trials, Object Avoidance, and Head-to-Bot Races.
2. Enter an original, descriptive name for the race.
3. Specify the start date and time of the event in 24-hour format. The AWS DeepRacer console automatically recognizes your time zone. For classic races, also enter an end date and time. LIVE races have a default duration of four hours. Contact customer support to schedule a longer race. There is no action to take if your race LIVE ends early.
4. To use the default race settings, choose **Next**.
5. On the **Review race details** page, check the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.
6. To share your race, choose **Copy invitation link** on the modal and paste it into emails, text messages, and your favourite social media applications. All classic races are private and can be seen only by racers with the invitation link. The link expires on the race's close date.



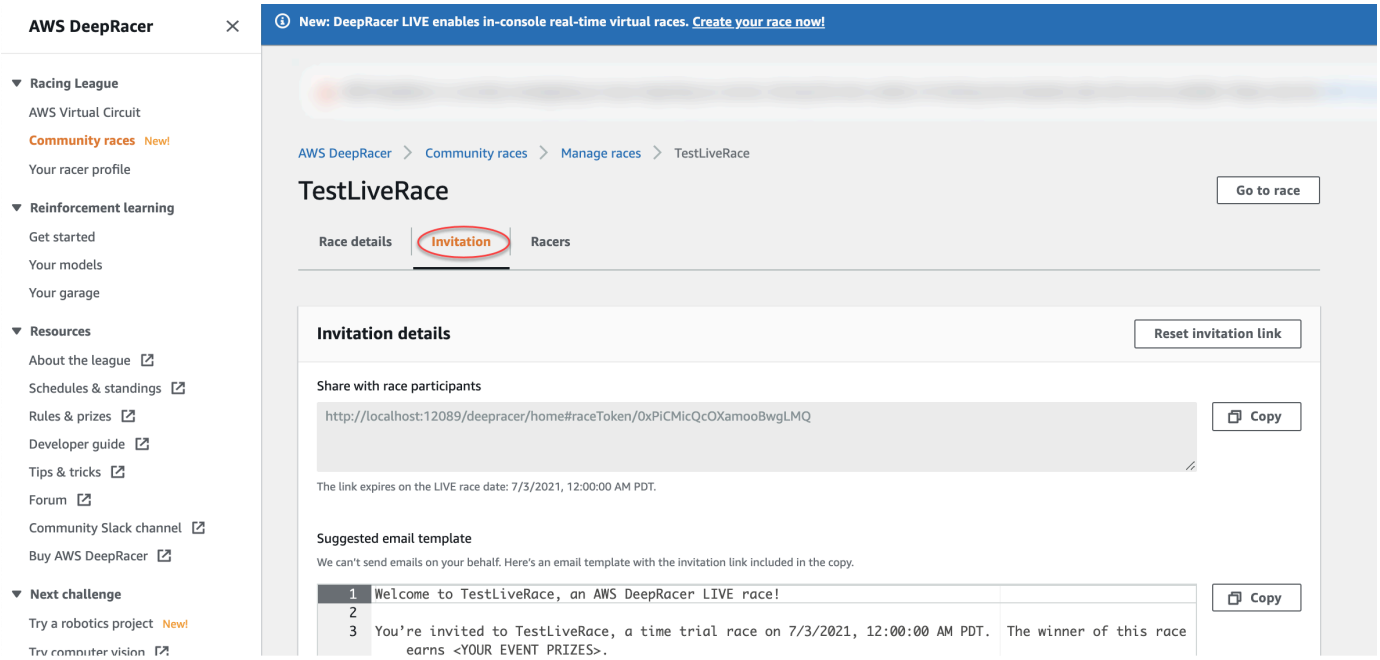
7. Choose **Done**. The **Manage races** page is displayed.
8. As your classic race time-frame comes to a close, take note of who has entered a model and who still needs to do so under **Racers** on the **Leaderboard details** page.

To continue creating a LIVE race

1. Choose a race type. Race types increase in complexity from **Time trial** to **Object avoidance** to **Head-to-bot**. For first time racers, we recommend **Time trial**. Time trial races require only one

camera, so the sensor configuration is simpler, and reinforcement learning (RL) models trained for this type of race converge faster. For more information about race types, see Tailor AWS DeepRacer Training for Time Trials, Object Avoidance, and Head-to-Bot Races.

2. Enter an original, descriptive name for the race.
3. Specify the start date and time of the event in 24-hour format. The AWS DeepRacer console automatically recognizes your time zone. For classic races, also enter an end date and time. LIVE races have a default duration of four hours. Contact customer support to schedule a longer race. There is no action to take if your race LIVE ends early.
4. To use the default race settings, choose **Next**.
5. On the **Review race details** page, check the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.
6. On the **<Your Race Name>** page, choose the **Invitation tab** to share your race.



7. Under **Invitation details**, choose **Copy** to paste the invitation link into emails, text messages, and your favorite social media applications.
8. Optionally, choose **Copy** next to the suggested email template and fill in your prizes, model submission time frame and the conference bridge link where your racers will meet to queue up and prepare for the race.

LIVE races are private and can be seen only by racers with the invitation link unless you choose to broadcast publicly. See the section called “Broadcast a LIVE race” to learn more. The link expires at 12:00 AM PDT on the race's close date.

9. Choose the **Race details** tab.
10. Under **Race details**, note the options for broadcasting your LIVE race. Once you've decided whether to broadcast your race publicly or privately, use playbooks created by the AWS DeepRacer League team to get started. The **View broadcast mode** button allows you to see the LIVE race event page formatted so that it can be used with branded graphic overlays that include cut outs for commentator streams.
11. As your LIVE race date approaches, take note of who has entered a model and who still needs to do so under the **Invitation** tab on the **<Your Race Name>** page.

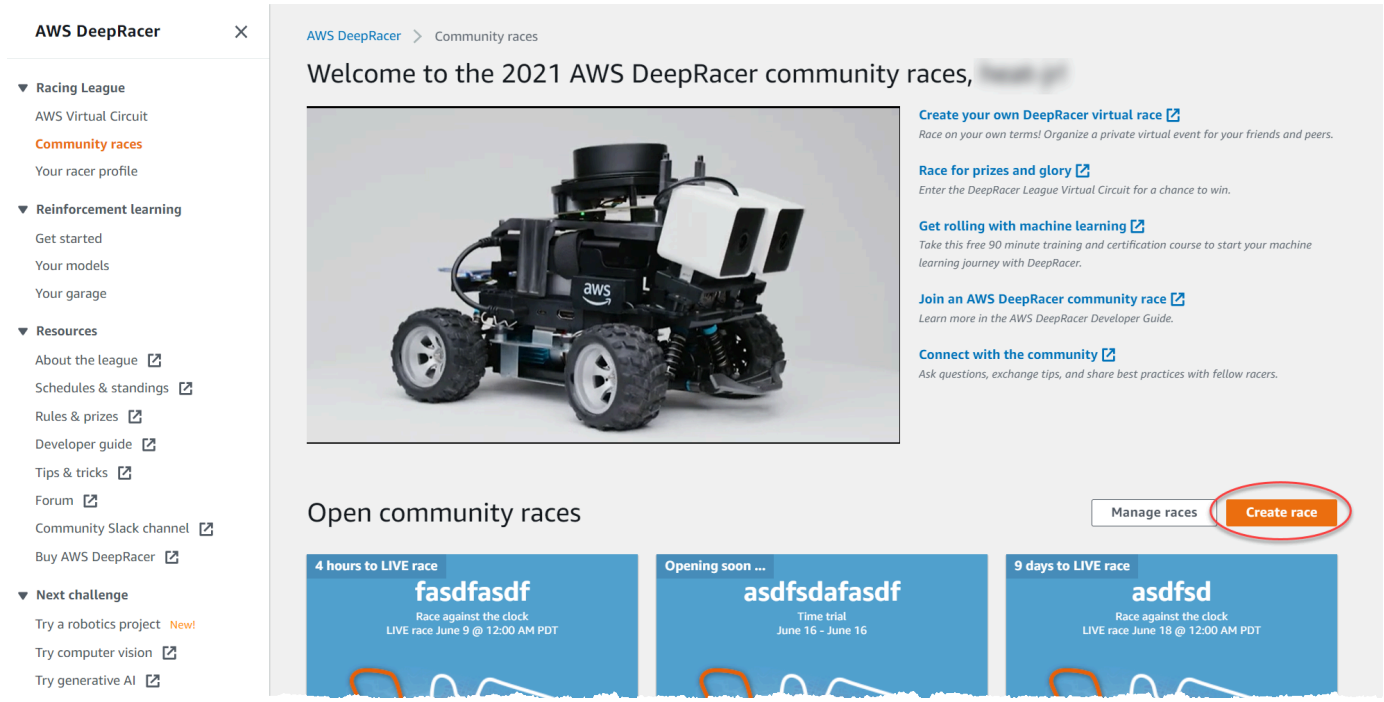
To change the race track selected, add a race description, pick a ranking method, decide how many resets racers are allowed, determine the minimum number of laps an RL model must complete to qualify for your race, set the off-track penalty, and customize other race details, choose **Edit race details** in Manage Community Races.

Customize a race

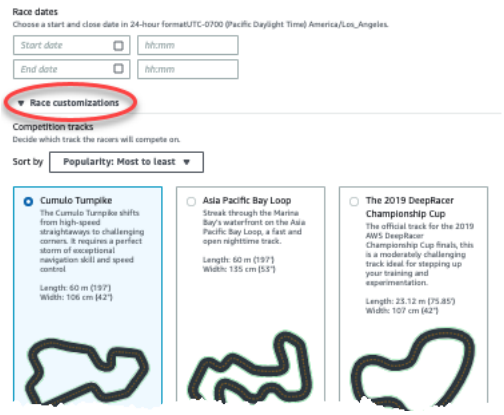
To create a race that is tailored for your group, expand **Race customizations** on the **Race details** page. The settings for a time trial race also apply to object avoidance and head-to-bot races, but object avoidance and head-to-bot race types have additional settings that give you the control to create race environments specially tuned to your event goals.

To customize a race

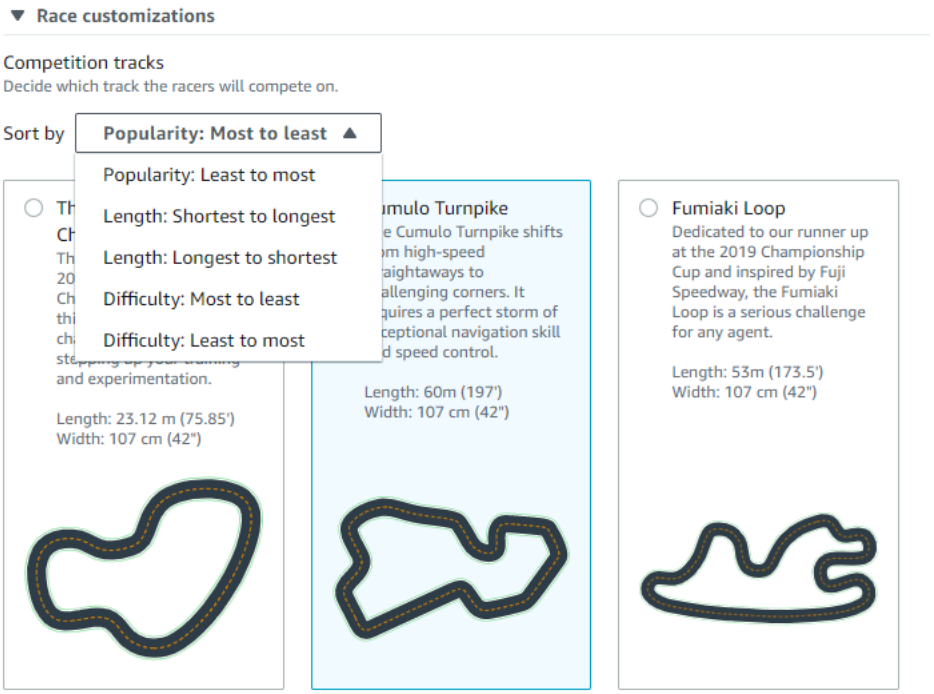
1. Open the AWS DeepRacer console.
2. Choose **Community races**.
3. On the **Community races** page, choose **Create race**.



4. On the **Race details** page, choose a competition format: a **Classic race**, in which your guests can participate on their own schedule within the time frame you set, or a **LIVE race**, which can be broadcast privately or publicly as a real-time event.
5. Based on your competition format choice, follow steps 1-3 of **To continue creating a Classic race** or **To continue creating a LIVE race** in the section called "Create a race quick start".
6. After choosing your **Race dates**, expand **Race customizations**.

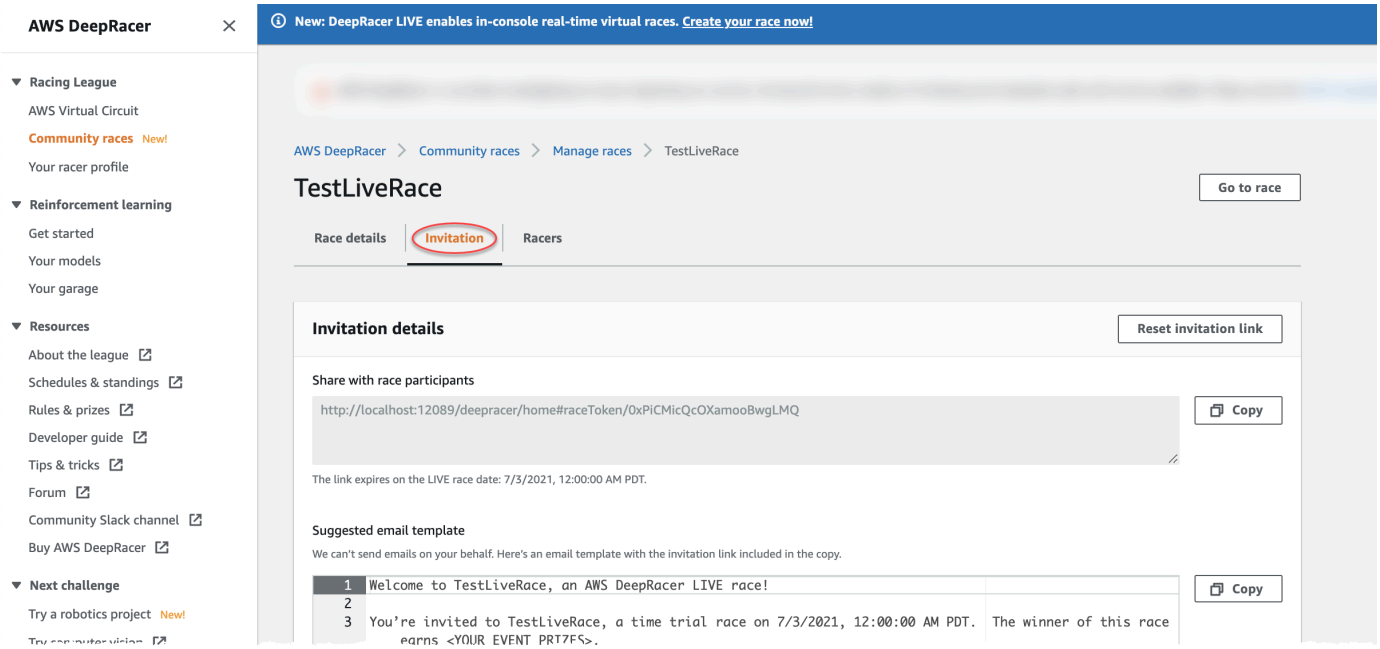


7. Choose a competition track. You can sort tracks by **Popularity: Most to least/Least to most**, **Difficulty: Most to least/Least to most**, and **Length: Longest to shortest/shortest to longest**. To see all tracks in each category, choose **View more race track options**. To close the expanded menu, choose **View fewer race track options**.



8. Optionally, write a description for your race that summarizes the goals and rules of the event for participants. For LIVE races, add the link for your event's video conference or LIVE stream. The description appears in your leaderboard details.
9. For **Ranking method** for a classic race, choose between the **Best lap time**, where the winner is the racer who posts the fastest lap; **average time** where, after multiple attempts within the time-frame of the event, the winner is the racer with the best average time; or **Total time**, where the winner is the racer with the fastest overall average. Leaderboard standings for all LIVE races are ranked by best lap time so this field does not appear.
10. For classic races, choose a value for **Minimum laps**, which is the number of consecutive laps a racer must complete to qualify for submission of the result to the race's leaderboard. For a beginners' race, choose a smaller number. For advanced users, choose a larger number. This customization is not available for LIVE races because the default is one lap.
11. For **Off-track penalty**, choose the number of seconds to add to a racer's time when their RL model drives off track.
12. You have now completed all the customization options for a **Time trial** race. If you chose a **Time trial** race format, choose **Next** to review race details. If you chose an Object Avoidance or Head-to-bot race format, skip to the appropriate procedure to finish customizing your race.
13. On the **Review race details** page, review the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.

14. To share your race, choose **Copy invitation link** on the modal to your clipboard and paste it into emails, text messages, and your favorite social media applications. You can also choose the **Invitation tab** to share your race on the **<Your Race Name>** page. The link expires on the race's close date.

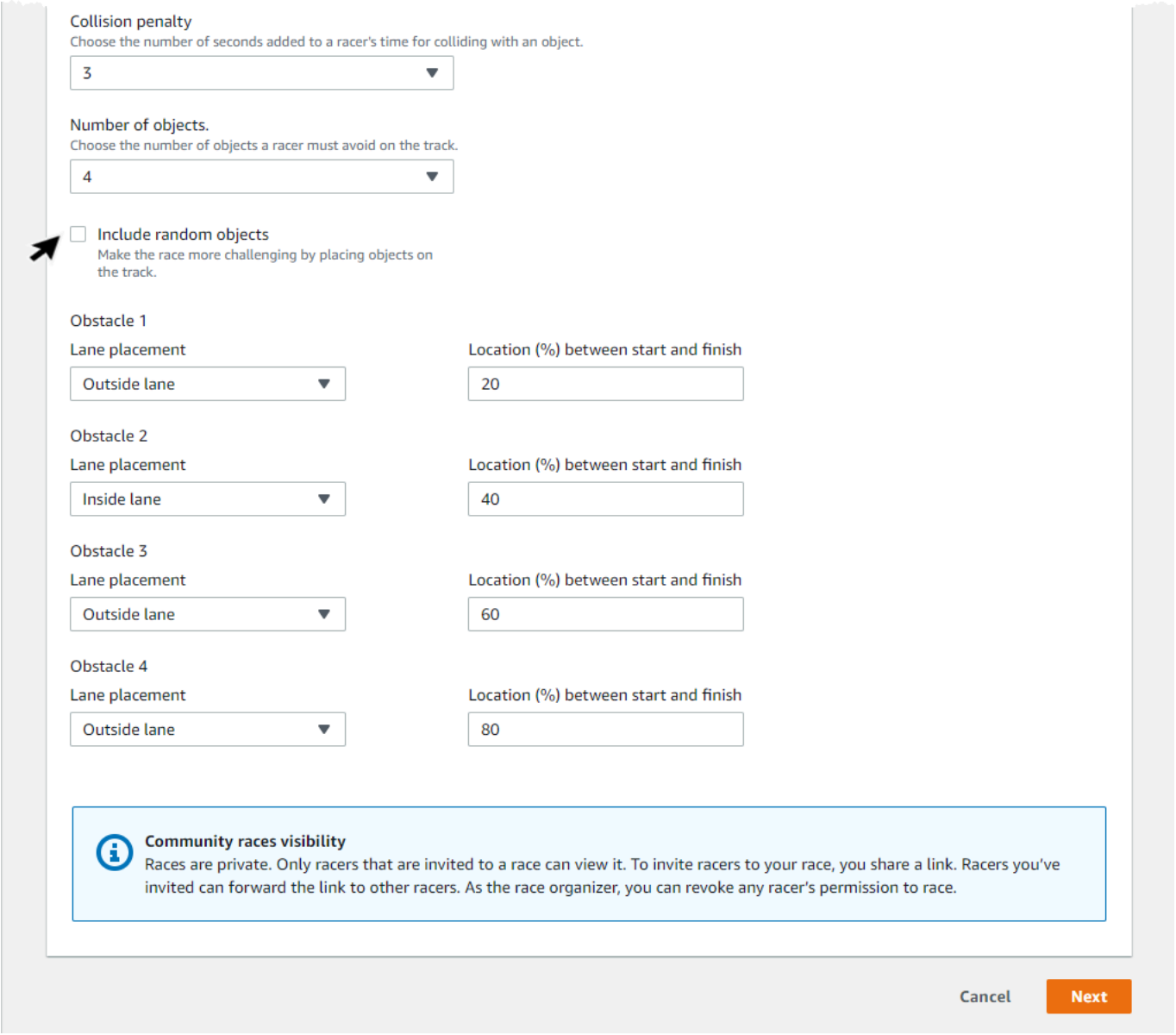


15. Choose **Done**. The **Manage races** page is displayed.

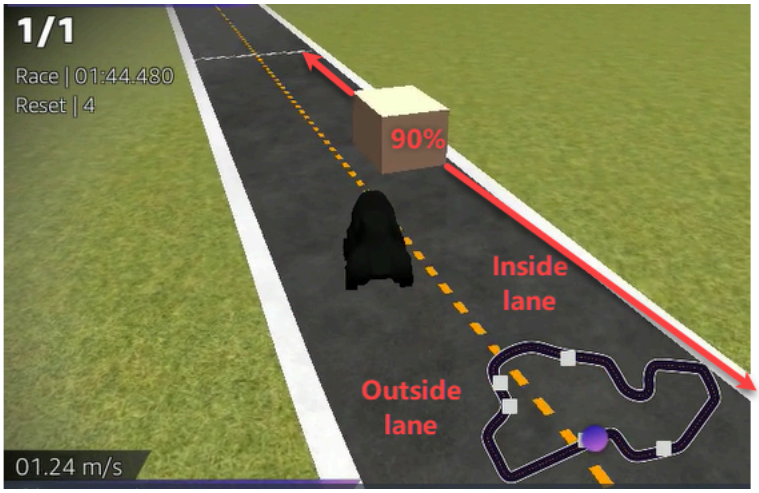
To learn how to use our email template to invite new racers, remove racers from your race, check on racers' model submission status and more, see Manage Community Races.

To finish customizing an object avoidance race

1. For **Collision penalty**, choose the number of seconds added to a racer's time for colliding with an object or bot. The more seconds added the greater the challenge.



2. For **Number of objects**, choose how many obstacles a racer must avoid on the track. The more objects, the more difficult the race.
3. To add random objects to the race track which will populate in different places for each racer, choose **Include random objects**. This is more challenging for participants, because it takes training for longer periods of time and reward function trial and error to create RL models that generalize well to random events like unexpected objects on a race track.
4. Choose where to place each object by choosing a lane number or object location for **Lane placement**. The track is divided in half at the center line, creating inside and outside lanes. You can place an object on either the inside or outside lane.

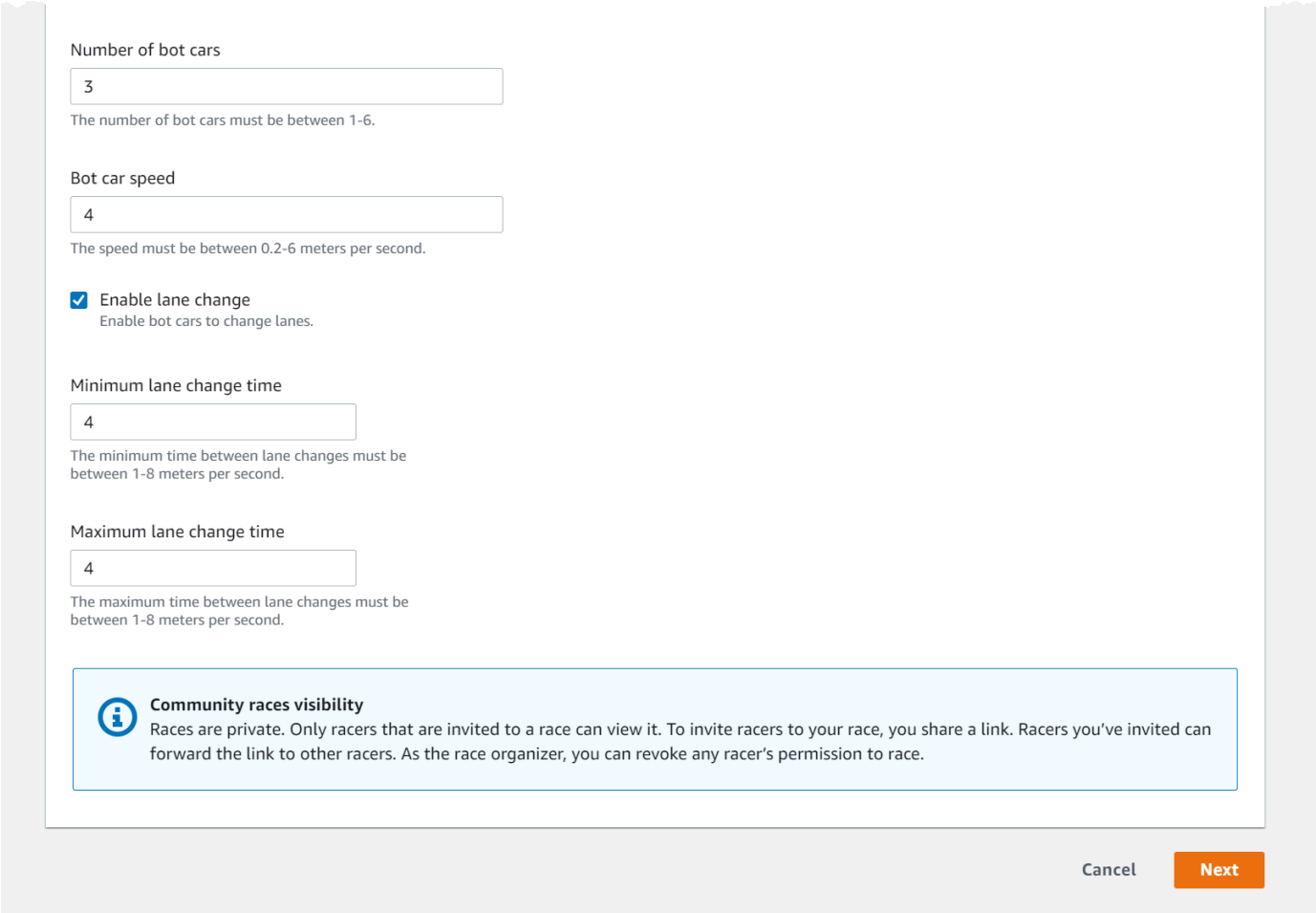


5. For each object, choose a value for **Location (%) between start and finish**. The number represents the location, represented as a percentage, between the starting and finish lines of your track where you want to place the object.
6. You have now completed all the unique customization options for an object avoidance race. Choose **Next**.
7. On the **Review race details** page, review the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.
8. To share your race, choose **Copy invitation link** and paste it into emails, text messages, and your favorite social media applications. All races are private and can be seen only by racers with the invitation link. The link expires on the race's close date.
9. Choose **Done**. The **Manage races** page is displayed.

To learn about what you can do with your race, see Manage Community Races.

To finish customizing a head-to-bot race

1. For **Number of bot cars**, choose the number of cars you want to race against your participants' AWS DeepRacer RL models. Bot cars are similar to video game AI vehicles. They are random objects that move, so they are a step up in complexity from stationary objects. The more bots on the track, the more challenging the race. Choose up to six.



2. For **Bot car speed**, choose how fast you want the bot cars to move around the track. Speed is measured in meters per second. The speed must be between 0.2 – 6 meters per second.
3. If you want to allow bots to change lanes, which adds further complexity to the challenge for your racers' AWS DeepRacer RL models, choose **Enable lane change**.
4. For **Minimum lane change time** , choose the minimum number of seconds that pass between instances where the bot cars change lanes.
5. For **Maximum lane change time**, choose the maximum number of seconds that pass between instances where the bot cars change lanes.
6. You have now completed all the unique customization options for a head-to-bot race. Choose **Next**.
7. On the **Review race details** page, review the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.

8. To share your race, choose **Copy invitation link** and paste it into emails, text messages, and your favorite social media applications. All races are private and can be seen only by racers with the invitation link. The link expires on the race's close date.
9. Choose **Done**. The **Manage races** page is displayed.

To learn about how you can edit and erase your race, see Manage Community Races.

Run a LIVE AWS DeepRacer community race

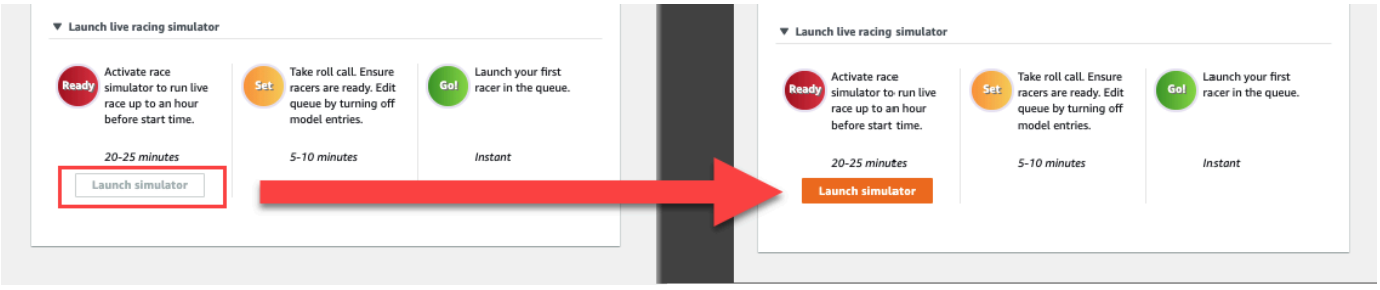
You've created a LIVE race and invited racers. You've decided whether to broadcast your event privately or publicly with support from the section called “Broadcast a LIVE race”. Now, learn how to manage the queue, set up the race simulator and launch your racers.

Before you start

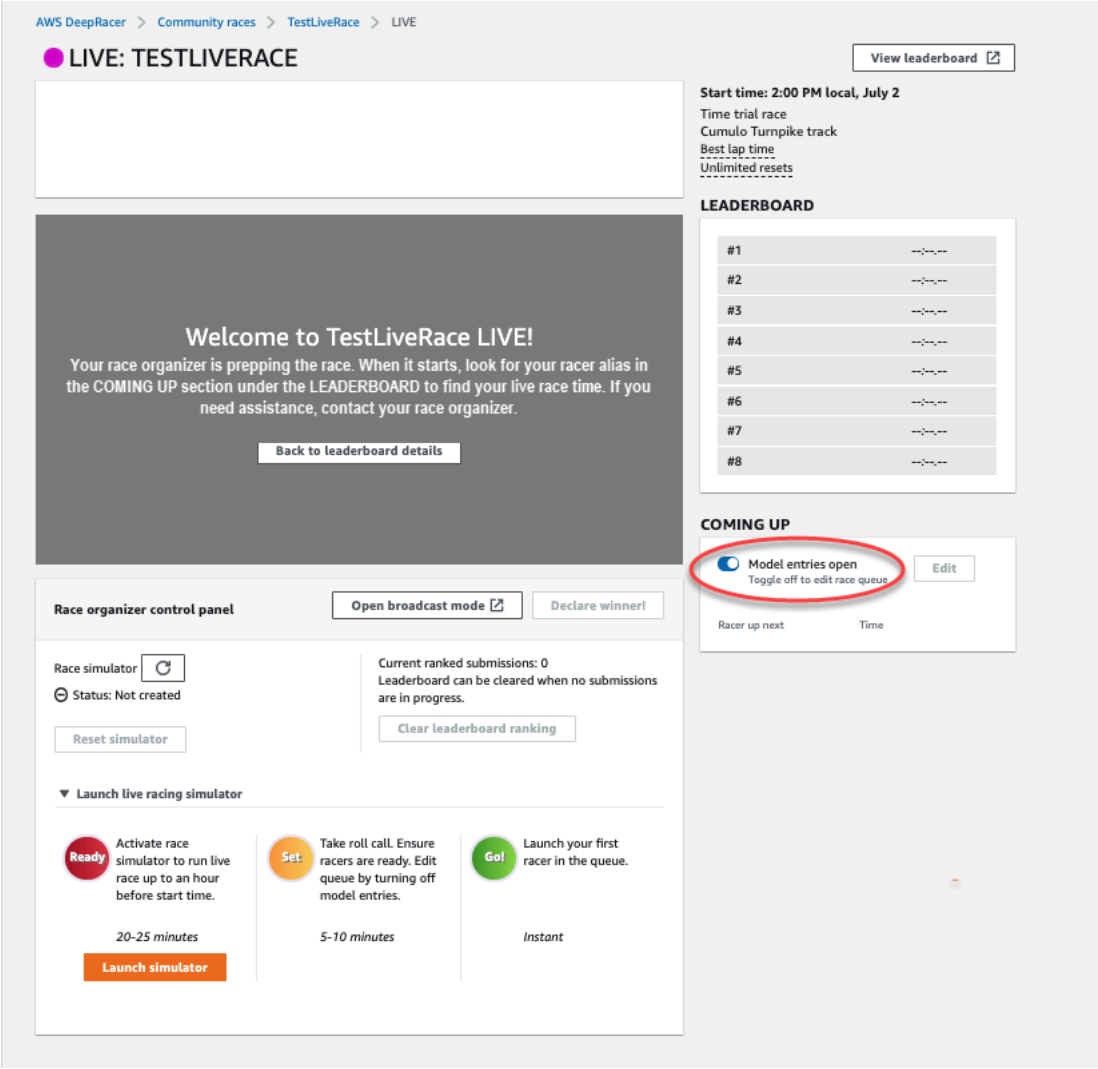
- Use a Chrome or Firefox browser (Check that your browser is up to date).
- Disconnect virtual private network (VPN) if you're using one.
- Close all extra tabs.

To run a LIVE virtual race

1. On the **Community races** page, find the race card for the race you want to moderate and choose **Join now** to view the race.
2. On the **LIVE: <Your Race Name>** page, under **Race organizer control panel** choose **Launch simulator**. This button becomes usable one hour before your race start time. You can hide this section of the race organizer control panel by selecting the **Launch LIVE racing simulator** header.

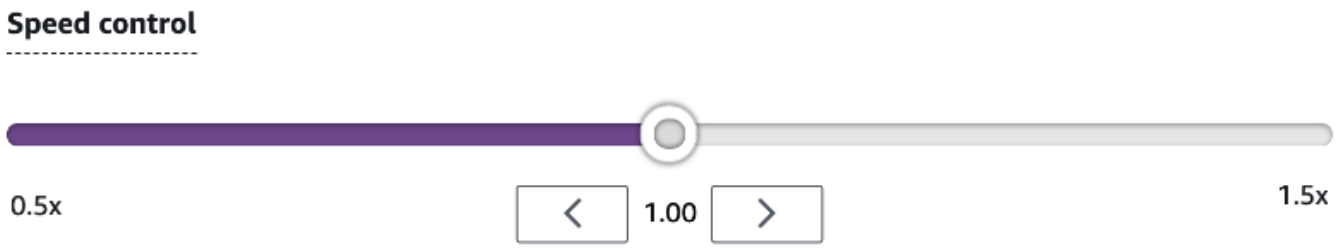


3. Under **COMING UP**, toggle off **Model entries open** to close submissions. This closes model submissions and creates an editable racer queue below the toggle. You can't launch racers until the toggle is switched off.

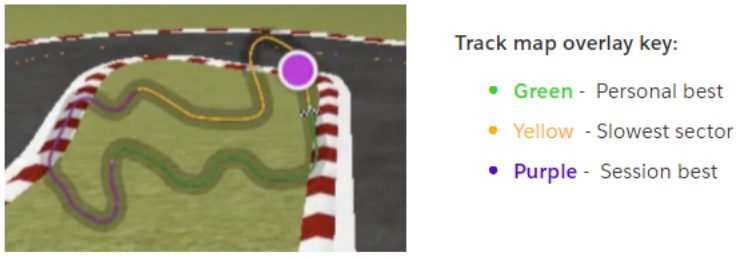


4. Open the video conference you created to gather your racers.
5. Initiate a racer roll call:
   a. Check with the racers to ensure they can hear you clearly.
   b. Use video at first to introduce yourself. You may want to shut it off later to optimize bandwidth.
   c. Check that the list of people on the call matches the list of racers in your group.
6. Initiate a model roll call:
   a. Check that the list of aliases in the racer queue matches those of your racers and that none of them are highlighted in red, which means that their model did not successfully submit.
   b. Check in with your racers to see if they're having any issues submitting their models.

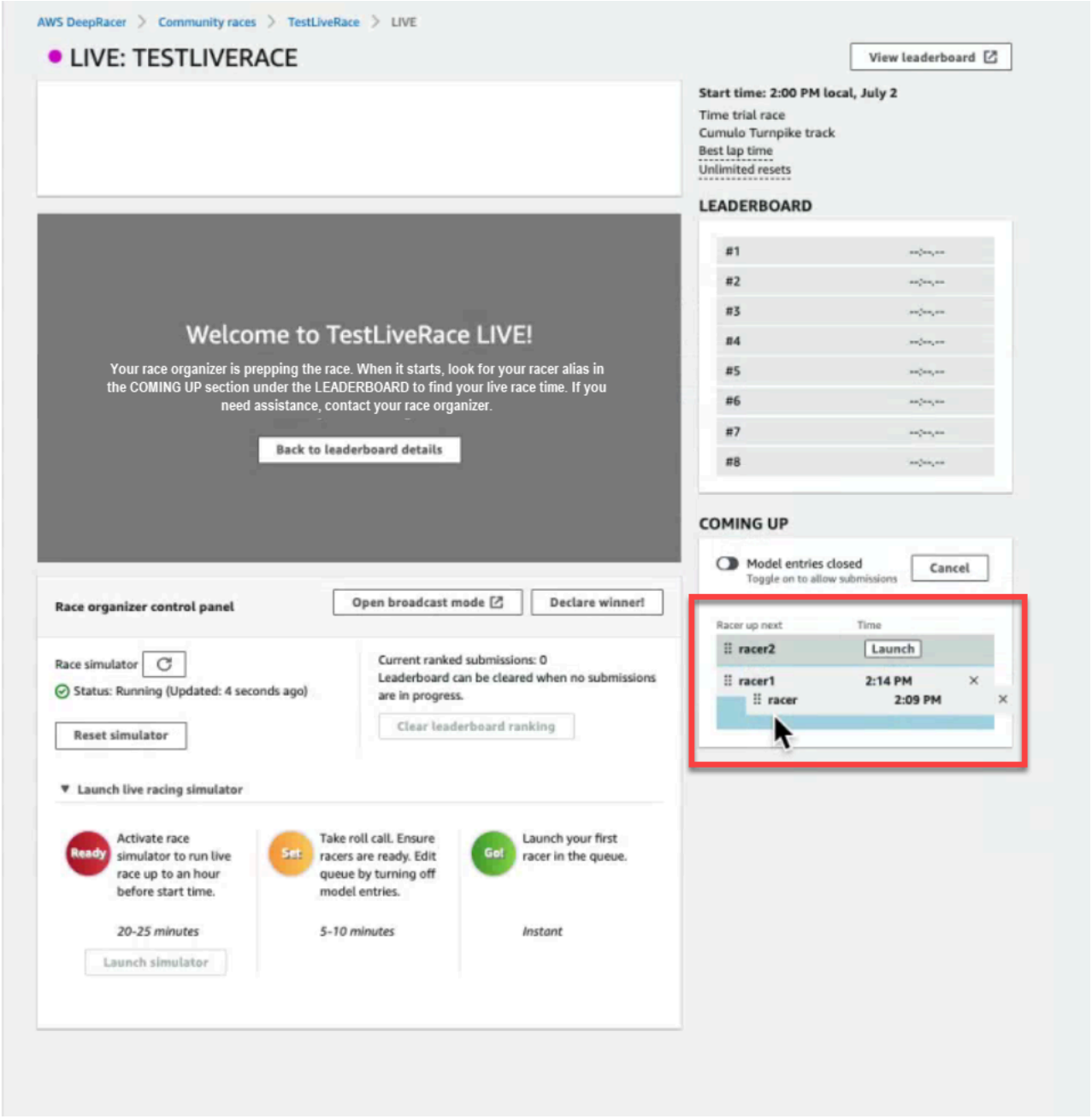
7. Review the race schedule and rules. Tell racers how much time they have to race on their turn, and remind them that the leaderboard standings are determined by their single fastest lap during that timeframe.
8. Explain that by using the **Speed control** feature, which is only visible to the racer during their race, they can manually set the maximum speed using the speed control slider, which temporarily overrides their model's speed parameters, but not the steering angle. The model still steers, but racers can now choose key moments to increase or decrease their car's speed by multiplying its rate. To return to using the model's speed parameters, racers can reset the multiplier to 1. Remind racers that the speed control slider is not the gas pedal; it's an opportunity for a strategic real-time adjustment.



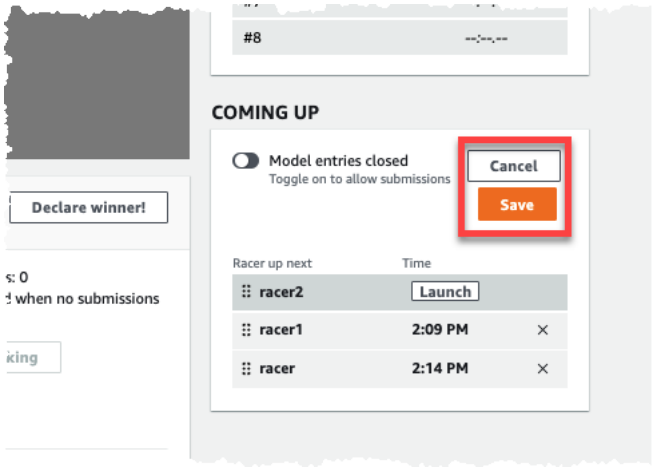
9. Next, explain that the video overlay of the race window features information to help optimize a racer's performance. The track map overlay is divided into three sectors that change color depending on a racer's pace. Green indicates the section of the tack where a racer clocked a personal best, yellow denotes the slowest sector driven, and purple signifies a session best. Racers can also find statistics detailing their best lap time, time remaining speed in m/s, resets, and current lap time.



10. Answer racer questions.
11. Optionally, under **COMING UP**, choose **Edit** to reorder your race queue by grabbing and dropping racer names.



12. If you make changes to your racer queue, select **Save** to keep your edits or **Cancel** to discard them.



13. Launch the first racer in your queue:

    a. Launch each racer manually by choosing the **Launch** button next to the top racer queue name. On each racer's turn, there is a **10, 9, 8, 7, 6...** countdown animated in the console after you launch.

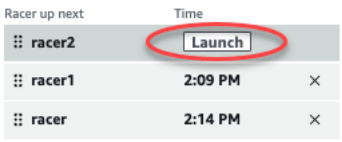
    b. On **Go!**, the model runs for your chosen amount of time while being evaluated in real time.

    c. In the case of a model failing in the middle of the race, you need to relaunch the racer using the **Launch** button next to their alias in the Racer queue.

    d. About 2 minutes before the current racer finishes, contact the next 2 racers in the queue through your conference bridge and confirm that they are ready to race.

    e. 30 seconds before the current racer finishes, give the next racer a 30-second warning.

    f. Launch the next racer as soon as you see that the current racer has finished. The end of the race is indicated by a checkered flag icon in the console. The racer's speed control is deactivated and a replay of the race launches on the video screen.



14. Optionally, choose **Reset simulator** if you are experiencing issues with the simulator.

15. You can also choose **Clear leaderboard ranking** if for any reason you'd like to reset the leaderboard, which clears all entries.

16. At the end of your race, choose the **Declare winner!** button, make final remarks to racers, explain how prizes are be distributed, answer questions, and close the video conference.

Broadcast a LIVE community race using AWS DeepRacer League production playbooks

LIVE races are real-time events that occur at a designated date and time. They range in scope from small events with one race organizer facilitating one private video conference to large events broadcast publicly by a small team of organizers, commentators, and broadcasters using a LIVE streaming service like Twitch.

Organizer roles

The following are suggested roles organizers can play during an AWS DeepRacer LIVE event. The more complex the event you plan, the more help you may need to enlist.

Organizers

Race organizers set up the race and associated video conference to organize and guide the racers. During a LIVE race, organizers use the organizer controls to queue, launch racers, and call a winner. Organizers do not appear on the LIVE channel.

Commentators

Commentators discuss the race while it's happening, providing a play-by-play of events, additional information, and inside knowledge of the event and its participants. Commentators are the main speakers of the public event.

Broadcasters

Broadcasters use streaming software to create scenes ahead of time and transition through them during the LIVE race. A broadcaster also manages the video feeds. The broadcasters do not appear on the LIVE channel. They act as producer of content during the event.

Broadcaster scenes

The LIVE stream of an AWS DeepRacer event tells the story of your race. To promote engagement throughout the beginning, middle, and end of your event, use *scenes*. These are animations and

layouts composed of graphic overlays and video streams that punctuate the different segments of your event.

An *overlay* is a graphic (usually a transparent PNG file) that sits on top of the broadcaster mode window of your race and the (optional) webcam streams or your commentators. It's like a mask for your stream. Position your content underneath it so everything lines up seamlessly to create one unified layout.

Use streaming software, such as OBS, to set up your scenes before your broadcast. Smoothly transition through them during the event to create dynamic pacing and audience delight. For example, use an intro animation scene to kickoff the event. Then transition to your primary content scene (PCS), which is the main layout containing the race view and one or two windows for commentators. Cut to a full screen dual commentator or commentator and interviewee scene to keep things lively, and end with a leaderboard scene. Optionally, create commercial scenes to cut in between races.

AWS DeepRacer scene templates

The AWS DeepRacer League Virtual Circuit team has created a collection of template files for you to use for your LIVE community races. Download the AWS DeepRacer Scene Templates and use them to broadcast a professional-looking event.

Scene types and how to use them

1. Intro AWS DeepRacer shield animation
2. Console share only view:
   - Base layer - screen share of the broadcaster mode url of your race. Resize it to fit frames of scene.
3. Single commentator view (1up):
   - Base layer - screen share of the broadcaster mode url of your race. Resize it to fit frames of scene.
   - Next layer - OBS Ninja or local webcam if commentator you are filming is in the same room. Pull in and resize under scene frame in upper right picture in picture (PIP) window.
4. Commentator plus interviewee or dual commentator (2up):
   - Base layer - screen share of the broadcaster mode url of your race. Resize it to fit frames of scene.
   - Next layer - OBS Ninja or local webcam if commentator you are filming is in the same room. Pull in and resize under scene frame in upper right picture in picture (PIP) window.
   - Pull in dual webcam feeds or ninja feeds into upper right windows resizing to fit (in setup a week before your event - AV check all your feeds and assign cameras in OBS)
5. Dual commentator full screen (no racing view; interview only):
   - No base layer console; only two camera feeds.
6. Ending leaderboards:
   - In real time, manually enter leaderboard results over scene layer.

AWS DeepRacer scene template file tips

- 34 - Configure your titles for commentators (prebuild scenes with names in PIPs)
- 234 - Racing views
  - Consider replacing the AWS DeepRacer League logo in the upper left with your company logo.
  - Replace the text in the lower left with your race name and your info in the vertical text.

To produce a LIVE private broadcast - 5 minute set up

An AWS DeepRacer LIVE Community Race Private Broadcast is a good fit for a small, informal race.

Organizer roles

- For a standard race you only need one organizer.

Hardware

- Recommended hardware - minimum 16 GB of ram
- (Optional) Quality microphones, headsets, or AirPods
- (Optional) LED ring light - To avoid seeing the ring light reflected on eyeglasses, position it at an angle to wearer's face.
- (Optional) Webcams and GoPros - to diversify footage

Tips

- Use a Chrome or Firefox browser (Check that your browser is up to date)
- Disconnect from VPN if using
- Close all extra tabs

To run a private LIVE AWS DeepRacer event

1. Open the AWS DeepRacer console.
2. Choose **Community races.**
3. On the Community races page, choose **Create race**.
4. Decide which date and time you would like to host a standard LIVE community race.
5. Before following the steps to create a LIVE community race, under Race date, check to see that this time frame is available. LIVE community races can be as long as four hours. Contact customer support to schedule a longer race.
6. When you settle on an available date time, create a corresponding video conference for race organizers and participants. If you are running a small race with little to no audience, one video conference is all you need. If you'd like to run a larger private race, create another video conference for broadcasting your race to an audience.
7. Follow the steps in the section called "Create a race quick start" and select **To continue creating a LIVE race**.

- Optionally, on step 8, choose **Copy** next to the **Suggested email template** and create an email for racers and race organizers. Fill in your prizes, model submission time frame, and the conference bridge link where your racers will meet to queue up and prepare for the race.

8. On race day, follow instructions to the section called “Run a LIVE race”.
9. Distribute prizes, if any, to race participants.

To produce a LIVE public broadcast - 2 hour set up

An AWS DeepRacer LIVE community race premium broadcast uses multiple broadcast scenes, a crew of three or more to broadcast a race on a global streaming platform. The following instructions use Twitch as an example.

Organizer roles

- Organizers
- Commentators/MC
- Broadcasters
- Twitch moderator - optional

Hardware

- Recommended hardware: You should have a minimum of 16 GB of RAM
- (Optional) Quality microphones, headsets, or AirPods
- (Optional) LED ring light: To avoid seeing the ring light reflected on eyeglasses, position it at an angle to wearer’s face.
- (Optional) Webcams and GoPros: Use these to diversify footage.

Tips

- Use a Chrome or Firefox browser (Check that your browser is up to date).
- Disconnect from VPN if you're using one.
- Close all extra tabs.

Prerequisites

- Twitch account - LIVE video streaming service.
- Twitch stream key - lets the software know where to send your video.
- Open Broadcaster Software (OBS) - Free and open source software for video recording and LIVE streaming.
- (Optional) VDO Ninja (formerly OBS Ninja) - Tool for adding and switching to and from additional video feeds if you opt to include commentators and interviewees.

To run a public LIVE AWS DeepRacer event

1. Set up a Twitch account by following the steps in How to sign up for a Twitch account.
2. Locate your Twitch stream key. Learn how to find your Twitch Steam key.
3. Download Open Broadcaster Software (OBS).
4. Learn how to use OBS to manage your scenes. Set them up ahead of time. We recommend preparing your assets at least one week before your race:
    a. Download the included AWS DeepRacer scene templates.
    b. Load scenes and modify them.
    c. Update the source with your race URL.
    d. Check your cameras.
    e. Assign people to their feeds.
5. Optionally, if commentators and interviewee are part of your broadcast event, use VDO Ninja (formerly OBS Ninja) to manage multiple video feeds. Learn how to use OBS Ninja .
6. Navigate to the AWS DeepRacer console to create a race.
7. Choose **Community races**.
8. On the **Community races** page, choose **Create race**.
9. Decide on which date and time you would like to host a public LIVE community race.
10. Before following the steps to create a LIVE community race, under **Race date**, check to see that this time frame is available. LIVE community races have a default duration of four hours. Contact customer support to schedule a longer race. There is no action to take if your LIVE race is shorter than four-hours.
11. When you settle on an available date and time, create a corresponding video conference for race organizers and participants.

12. Next, create another video conference for your broadcasters.
13. Follow the steps to set up a LIVE community race.

    a. Optionally, on step 8, under Description of race, add the link for your LIVE stream for racers to share with their families and friends. You may also include the racer room conference bridge for racers. The description will appear in your leaderboard details providing easy access to the links.

    b. Optionally, on step 12, choose **Copy** next to the **Suggested email template** and create an email for racers and race organizers. Fill in your prizes, model submission time frame, and the conference bridge link where your racers will meet to queue up and prepare for the race.

    c. Create another email or chat for your team of organizers.
14. On the race day, follow instructions to the section called “Run a LIVE race”
15. Celebrate winners and participants, distribute prizes, write blogs, tweet, post, and proliferate.

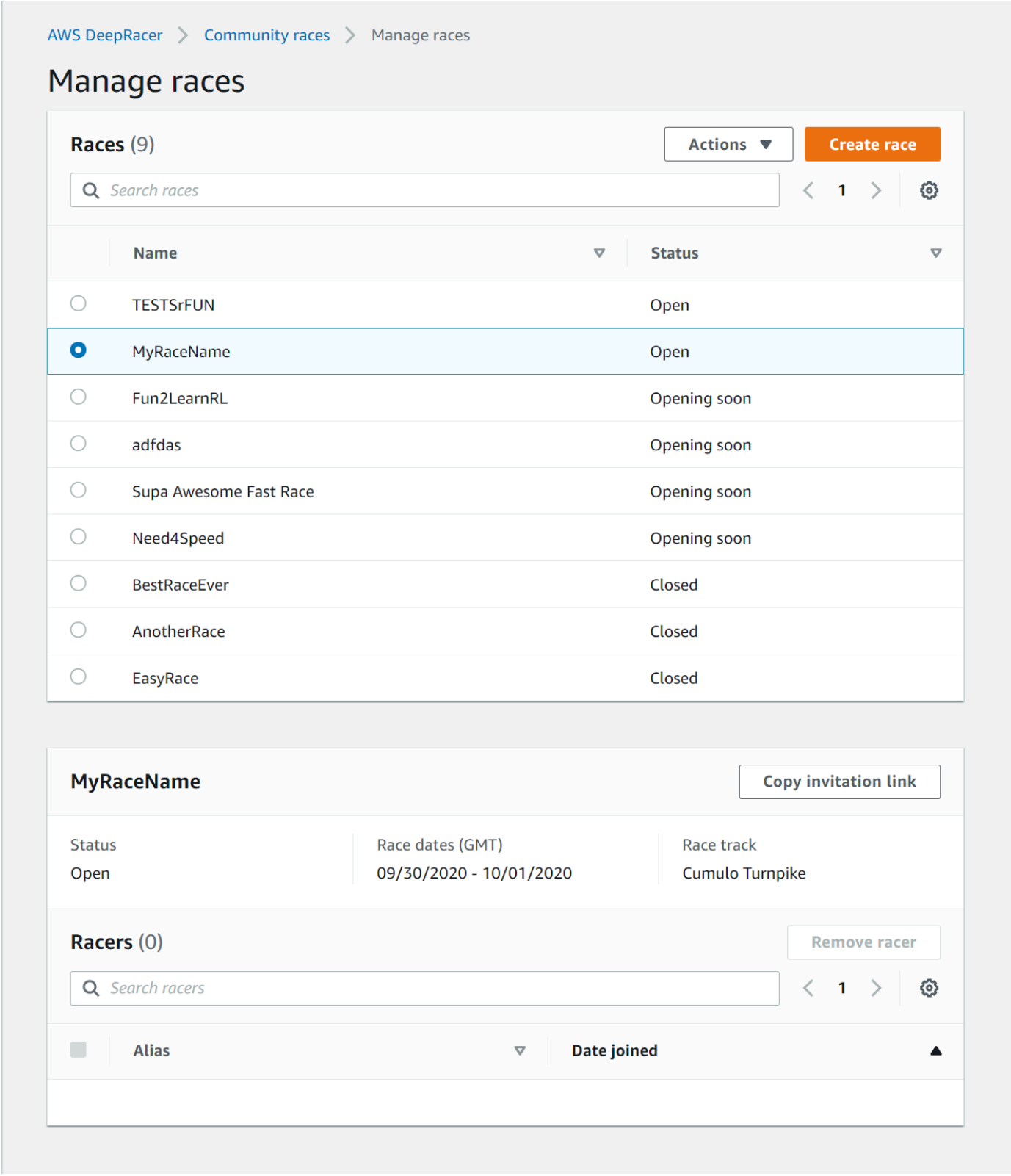
Manage an AWS DeepRacer community race

All community races are private. They are visible only to individuals who have an invitation link. Participants can freely forward invitation links. However, to join a race, participants need an AWS account. First-time users must complete the account creation process before they can join the race.

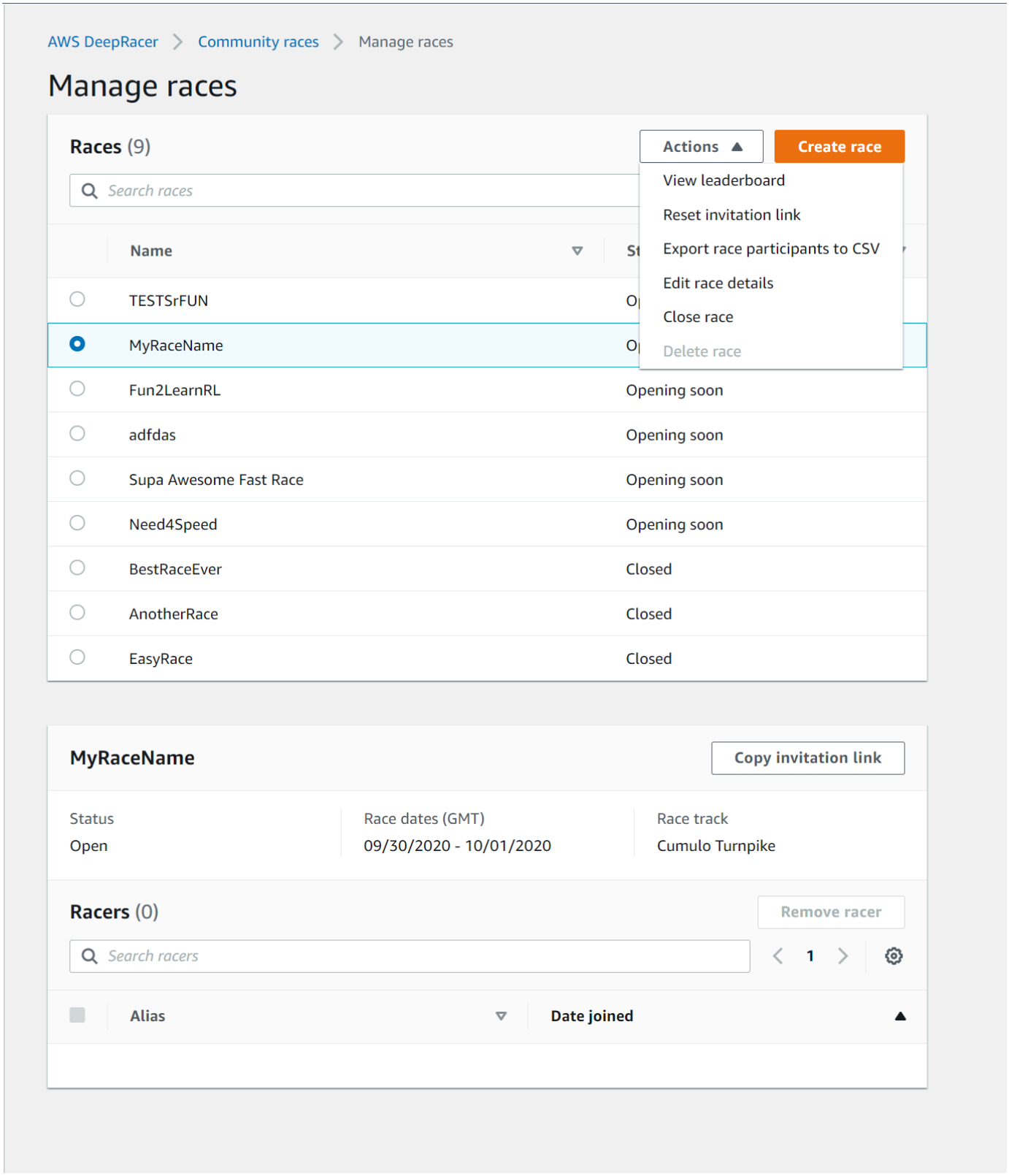
As the race organizer, you can edit race details, including the start and end dates, and remove participants.

To manage an AWS DeepRacer community race

1. Sign in to the AWS DeepRacer console.
2. Choose **Community races**.
3. On the **Manage races** page, for **Races**, choose the race that you want to manage. The chosen race's details, including the list of participants is displayed.



4. To edit the race details, in **Actions**, choose **Edit race details**.



Follow the on-screen instructions to finish editing.

5. To view the event's leaderboard, from **Actions**, choose **View leaderboard**.

6. To reset the event's invitation link, from **Actions**, choose **Reset invitation link**. Resetting the invitation link prevents anyone who has not yet chosen the original link from accessing the race. All users who have already clicked the link and submitted a model remain in the race.

   You can also copy the link to share it with invited participants.
7. To end an open race, from **Actions**, choose **Close race**. This ends the race immediately, before the specified closing date.
8. To delete the event, from **Actions**, choose **Delete race**. This permanently removes this race and details from all participants' community races.
9. To remove a participant, choose one or more race participants, choose **Remove participants**, and then confirm to remove the participant.

   Removing a participant from an event revokes the user's permissions to access the racing event.

Organize an AWS DeepRacer event

What is an AWS DeepRacer event?

AWS DeepRacer is an educational service that provides a fun way to get hands-on learning with artificial intelligence and machine learning (AI/ML). AWS DeepRacer can help bridge the AI/ML talent gap for your organization and apply AI/ML to your business needs.

AWS DeepRacer not only introduces AI/ML skills to your team, it also allows you to host events to encourage team building and friendly competition. These events help both technical and non-technical participants learn the fundamentals of machine learning by providing hands-on experience with creating reinforcement learning models to race AWS DeepRacer cars in-person or virtually in the AWS DeepRacer League. AWS DeepRacer events also help leaders engage their teams to reach their organization's AI/ML visions and goals.

This guide provides resources, tools, and examples to help you start planning and hosting your own virtual or in-person AWS DeepRacer events. If you want to plan your AWS DeepRacer event with 50 participants or fewer, jump to What to consider before getting started. If you're planning a larger event (with more than 50 participants), we recommend working with your AWS account team and requesting an event.

To learn more about the benefits of AWS DeepRacer events and view customer testimonials, see AWS DeepRacer enterprise events.

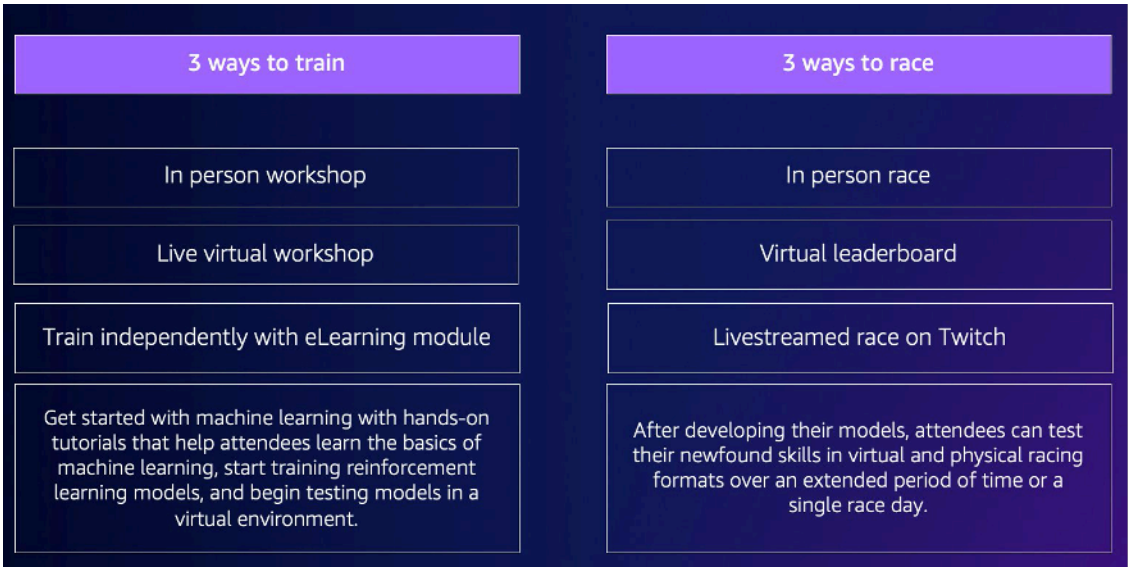
How AWS DeepRacer events work and what to expect

Whether you want to provide education and hands-on practice with reinforcement learning for your team, promote your organization to attract new talent, or a combination of both, this guide provides the tools and resources to help you create and customize your own AWS DeepRacer event.

AWS DeepRacer events are flexible to suit your needs and objectives, but the basic formula for an AWS DeepRacer event includes:

- An educational component, such as an online AWS DeepRacer reinforcement learning course or in-person workshop with an AWS DeepRacer Pit Crew expert.
- Hands-on model training in the AWS console and time for participants to create and train their models.

- An AWS DeepRacer race, such as a single in-person race or multiple virtual races, so that participants can see their trained machine learning models in action. For more information about the types of races you can host, see Types of AWS DeepRacer races.
- A post-event recap or next steps communication for event participants.



For more context about what to expect from AWS DeepRacer events and customer highlights, see the AWS DeepRacer Accelerate YouTube video.

What to consider before getting started

The first step in planning your event is to define your business objectives and goals for your organization and then develop a project plan. An example of a goal is, “I want to encourage team building in a fun and educational way in my organization.”

Your project plan should answer the following questions:

- **Who are your event leaders?**

  Identify who in your organization is helping lead the planning and execution of this event. Event leaders might include teams like Facilities, Human Resources, PR/Marketing, an Executive sponsor, or AWS account team
- **Who are the event owners?**

  Identify who in your organization owns the event. This person or team should be the primary organizer and should be part of all decisions when planning this event.

- **What is the event date?**

  Identify when you want to hold the event. If you are considering a large event (with more than 100 participants), you need to schedule your event date further out to provide enough lead time to plan and facilitate the event. For an example timeline, refer to AWS DeepRacer event examples.

- **What is the estimated budget?**

  Estimate a budget for your event. There are two cost considerations you should plan for with an event.

  - **The event costs**: If you're hosting an in-person event, this can include everything from the event location, track, and device purchases to food, beverages, and event prizes.
  - **Model training costs**: This cost is incurred by your employees training reinforcement learning models on the AWS Management Console using the AWS DeepRacer service. For example, each participant needs approximately 10 hours to train one or multiple models in the AWS DeepRacer console. For more details about costs, see AWS DeepRacer Pricing.

- **What kind of location is necessary for the event?**

  If you are holding an in-person event, you need to procure a physical location for the workshop and race. The type of physical track you select also needs to fit in the location.

- **What is the estimated number of participants?**

  Estimate the number of participants attending your event. The number of participants also factors into your estimated budget.

- **Do you plan to have a retrospective session after the event to gauge success?**

  To encourage participants to continue growing their ML skills and keep collaborating, consider communicating next steps and asking for participant feedback. For example, send a survey to participants to gauge interest and recruit AI/ML leaders within the organization. This may also determine who is involved with future AWS DeepRacer events.

- **What is the communications strategy to promote the event in your organization?**

  Communicating this event within your organization can be as simple as an internal email or calendar invite.

- **Who are the executive stakeholders?**

Identify the executives that are sponsoring the event and encouraging thought leadership and collaboration within your organization.

Types of AWS DeepRacer races

After answering the main questions for your project plan, decide what type of races you want at your event. The type of race you host should be based on your estimated budget and number of participants for the event. You can host the following two types of races.

Virtual racing

Virtual racing is a great way for multi-region or remote teams to collaborate and race their trained models in a virtual environment. Since virtual racing doesn't require physical devices and tracks, it provides flexibility in group size and it's a great solution for organizations that have location and budget constraints.

There are two types of virtual racing: classic racing and live racing. The main difference between classic racing and live racing is that classic racing participants can train their models and submit them on their own time. Live Racing involves participants racing their models one after another within a set amount of time. Both types of races can be shared privately or publicly and use leaderboards to determine participant ranking. AWS also provides tools so you can broadcast your races on Twitch.

Virtual racing is the fastest way to get started with AWS DeepRacer races. Participants can go through the getting started process, which includes an introduction to machine learning and reinforcement learning in the console. The console guides the participants through creating their first model. For more information about how to set up a virtual race, see Create a Race.

In-person racing

Host an in-person race to provide an engaging event that your team can attend in-person. In-person racing includes a physical track and AWS DeepRacer vehicle devices. AWS provides options for building your track and procuring vehicles for your event.

For more information about AWS DeepRacer devices, see Operate your AWS DeepRacer vehicle. For more information about constructing and ordering your physical track and barriers, see Build your physical track.

In addition to these two types of races, you can also include educational online training or workshops and livestream your race events on Twitch.

Best practices

To make your AWS DeepRacer event planning even more efficient, we also recommend the following best practices.

- Gain internal leader support. Support from organization leaders encourages team participation and increases engagement and overall participant satisfaction.
- If you have an AWS Account Manager, contact them to inform them of your event and discuss whether you need any support.
- If you're planning a large event (with more than 50 participants), engage with your AWS Account Managers early and often. Submit a request if you aren't sure who to contact on your AWS account team. To submit a request, see Request an event. For more resources from the AWS DeepRacer community about events and training, see the AWS DeepRacer Slack channel.
- Establish a budget that meets your AWS DeepRacer event goals. Your budget might affect the type of event you're planning, but AWS DeepRacer provides tools, such as multi-user mode, to help limit and manage costs.
- Provide prizes for your participants. Whether your team is interested in the traditional championship cup or NFTs, make sure your prizes are appealing to encourage participation.

Getting started with your AWS DeepRacer event

Once you've defined your organization's goals, you can use your project plan to begin narrowing down the type of event you want to hold. The following example goals demonstrate how you can set up an event based on your requirements and the benefits you want to gain from AWS DeepRacer.

Team building

If you want to host a one-time local event that encourages team building for smaller groups, consider an in-person or virtual event. For an example of the type of event that meets this goal, see Virtual event examples.

Investing in AI/ML education

If you want your technical and non-technical employees to become more familiar with machine learning and apply these skills, you should consider asking for more leadership support and think about making your event a cross-team event, which includes more participants to create a larger organizational impact. This event can include both in-person and virtual event components to allow for participant scaling. For an example of the type of event that meets this goal, see In-Person event examples.

Promote and generate organization awareness

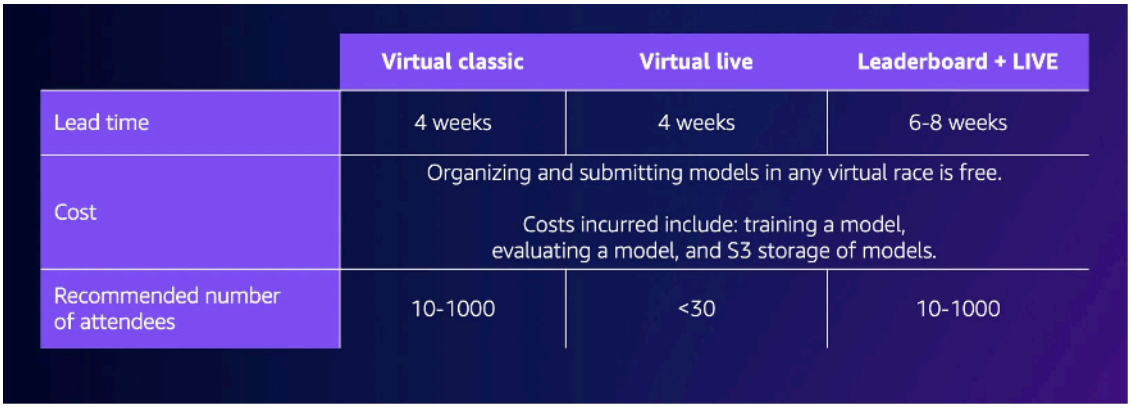
If you want to position your organization as innovative and thought leaders to attract more talent and encourage more general awareness within your organization, consider livestreaming your virtual or in-person event, or creating a custom event. For an example of the type of event that meets this goal, see Custom event example.

AWS DeepRacer event examples

The following sections provide some examples of the different types of events you can create based on your goals and project plan requirements. These event timelines are scheduled based on the education and race components. However, you can customize your event timeline for any number of event components based on your organizational needs.

Virtual event examples

Virtual events are a great way for organizations across multiple locations or multi-regional teams to gather in a convenient and cost-effective way. Virtual events are more convenient and cost-effective because they have fewer dependencies. For example, you don't need to consider physical tracks, devices, or a location space like you would for an in-person event. The following virtual event examples focus on different project plan items, such as timeline and number of participants.



	Virtual classic	Virtual live	Leaderboard + LIVE
Lead time	4 weeks	4 weeks	6-8 weeks
Cost	Organizing and submitting models in any virtual race is free. Costs incurred include: training a model, evaluating a model, and S3 storage of models.		
Recommended number of attendees	10-1000	<30	10-1000

- **Virtual classic race two-week event**:

The following example of a two-week event schedule is a great option for organizations looking for a simple way to host an event with many participants and teams planning to join the race, since there is no limit to classic race submissions and no Live race time constraints. All participants can submit their models at any time, even simultaneously, within the race window. This event example uses the community races section in the AWS console to create a private classic race and schedules the training and races in two separate weeks. To learn how to set up a community race, see Manage your races.

- Day 1: Participants attend a free, on-demand virtual workshop for all attendees. For more information about the online workshop resource, see AWS DeepRacer: Driven by Reinforcement Learning. You can also schedule an AWS DeepRacer Pit Crew expert to deliver a virtual workshop.
- Days 1 - 5: Participants train, update, and test their models in the AWS DeepRacer console. They submit their models to compete in a private virtual race throughout days 1-5. To learn how to set up the AWS console for multiple participants under one AWS account, see Multi-user Mode.
- Days 5 - 10: The top 10 winners are identified on day 5 and provided with access to a new private virtual race on a new track in the following week.
- Day 10: The top three winners are identified on day 10 and the race and event concludes.

- **Virtual live race one-week event:**

  The following Virtual live race event example is a great way to bring the excitement and fun of racing in real time to smaller teams. This type of event is low-budget and allows everyone to race in real time. For more information about how to run a live race, see Run a LIVE race.

  - Day 1: Participants attend a virtual workshop with an AWS DeepRacer Pit Crew expert. For more information about workshops, see AWS DeepRacer Events.
  - Days 1 - 5: Participants train, update, and test their models in the AWS DeepRacer console from days 1-5.
  - Day 5: Participants gather for 1-2 hours online to submit their models and participate in a live virtual race on day 5.

- **Virtual classic race and live race event:**

  The following event example is a great option for organizations that want to unite many multi-regional participants or participants working remotely across multiple offices online. In this type of event, your participants have more opportunities to practice training their models and race since the event timeline is spread out across two weeks. We also recommend

having an announcer at your live race events to make your event more exciting for participants watching the livestream. Check out the Pro Division Finale Twitch stream to see how the finale broadcasters make the race more exciting.

- Day 1: Participants attend virtual workshop with an AWS DeepRacer Pit Crew expert. For more information about workshops, see AWS DeepRacer Events.
- Days 1-5: Participants train, update, and test their models in the AWS DeepRacer console and then submit models to compete in a private league race from days 1-5.
- Day 5: The top 10 winners are identified and provided with access to a new private league race on a new track.
- Days 5 - 10: On the following week, the top 10 winners are identified and race while live streaming the event on Twitch. For more information, see Broadcast a LIVE race.

In-person event examples

For organizations that are able to gather their participants in a single physical space, there is no better way to experience AWS DeepRacer than an in-person event. Nothing beats experiencing the thrill and excitement of standing trackside and seeing your model perform on a physical car. In general, in-person events require more resources and are more expensive than virtual events. For any organization that has more than 50 participants and the necessary budget, we highly recommend taking advantage of the in-person AWS DeepRacer experience. We also recommend having an announcer at these in-person events to make your races even more exciting as competition results are announced in real time for participants watching from the trackside.

	One-day workshop + race	Two-day workshop + race	Custom format
Lead time	6+ weeks	6+ weeks	10+ weeks
Recommended number of attendees	51–100	100–500	500–1,000

- **In-person one-day event:**

  The following example of an in-person event is great for local teams and ensures that all participants can train their first model in the AWS Management Console and compete in one day. A typical one-day in-person event consists of an in-person workshop and race for all attendees. The workshop is usually led by an AWS DeepRacer Pit Crew expert who goes through the

fundamentals of reinforcement learning and gives participants an opportunity to train their first model. Following the workshop, participants can submit their models and upload them to an AWS DeepRacer device and race on the same day.

- Participants attend an in-person AWS DeepRacer workshop delivered by an AWS DeepRacer Pit Crew expert (90-120 minutes). For more information on how to request AWS DeepRacer workshops, see Request an event.
- In-person race following the workshop (120 minutes or ~5 minutes per race).

- **In-person two-day event:**

  The following two-day in-person event example is similar to the one-day event, except that spreading out the workshop and actual race on separate days gives participants more time to train and update their models as they prepare for the race. It's common for organizations to host these events over a few days or even weeks apart to give participants more time to train and refine their models. Providing more time in between workshops and races allows participants to have a more competitive race day.

  - Day 1: Participants attend an in-person AWS DeepRacer workshop delivered by an AWS DeepRacer Pit Crew expert (90-120 minutes). For more information on how to request in-person workshops, see Request an event.
  - Day 2: Participants attend an in-person race following the workshop (120 minutes or approxiamtely 5 minutes per race).

Custom event example

As with virtual events, custom events are a great option for larger organizations (100 participants or more) that need to host an event for teams across multiple locations. Custom events allow you to be more flexible with training, workshops, and races because there are no limits on time and race formats. You can include both virtual and in-person races in these events and these races can span multiple weeks to allow global participants to spend more time training their models and collaborating. This type of custom event is more successful when you first run a few smaller events beforehand to prepare for any potential logistical issues. This type of event or series of events also helps cultivate a team of machine learning evangelists in your own organization.

- In the following example, the custom event is spread across three months to accommodate employees across multiple regions.
  - Month 1: Global workshops in multiple locations.
  - Month 2: Month-long virtual league qualifier races available to participants in multiple regions.

- Month 3: In-person race and virtual Championship Cup race. This in-person race can be livestreamed so your global teams can watch.

If you're interested in hosting a custom event, contact AWS to get event support. See Request an event.

Additional resources

For more resources related to AWS DeepRacer events, refer to the following list:

- AWS DeepRacer Blog
- AWS DeepRacer League
- AWS DeepRacer community Slack channel
- Machine learning training
- Machine learning certification
- AWS DeepRacer training
- AWS DeepRacer GitHub repository
- Racing tips
- AWS DeepRacer YouTube channel
- AWS DeepRacer storefront

Multi-user mode

Multi-user mode account setup provides an exciting way for organizations to sponsor multiple AWS DeepRacer participants under one AWS account. Sponsored participants do not incur any of their own expenses; instead, their training hours and storage costs are billed to the sponsoring AWS account. With a multi-user mode account setup, AWS DeepRacer event organizers can set budgets and monitor and control spending by updating default quotas on training hours and models for individual participants, groups, or across all participants.

The following sections describe how to get rolling with AWS DeepRacer multi-user mode, as either an admin or a participant.

> **Note**
>
> Multi-user mode with account sponsoring is only available in the AWS DeepRacer service.

Topics

Set up multi-user mode (admin)

With a multi-user account setup, organizers (such as account administrators) can provide participants access to AWS DeepRacer service under their account ID. They can also set usage quotas on participants' training hours, monitor spending on training and storage, start and stop training, and view and manage models for every user in their account from the AWS DeepRacer console.

Multi-user mode is particularly useful for large events with multiple participants who don't have individual AWS accounts. Instead of creating and managing accounts for each participant in an event, an AWS DeepRacer administrator can host all of their sponsored participants through a single AWS account.

In multi-user mode, sponsored participants can compete and train without incurring any of their own costs. Their training and storage charges are billed to the sponsoring multi-user AWS account

billing. If an administrator stops sponsoring participants' usage, participants keep their racer aliases and profiles.

Multi-user stakeholders

This walkthrough refers to the following typical multi-user stakeholders for setting up and using multi-user mode.

- **AWS administrator for IAM/SSO configuration.** The AWS administrator for IAM/SSO configuration sets up IAM or SSO for the AWS DeepRacer administrator and for participants to use multi-user mode. The AWS administrator for IAM/SSO has IAM and SSO administrator permissions. For information about creating IAM users, see Creating an IAM user in your AWS account.
- **AWS DeepRacer administrator.** The AWS DeepRacer administrator manages AWS DeepRacer participants' sponsorship and can pause and resume sponsorship, delete models and artifacts, configure and host virtual races, and enable and disable multi-user mode. The AWS DeepRacer administrator has AWSDeepRacerAccountAdminAccess permissions.
- **AWS DeepRacer participant.** AWS DeepRacer participants are invited to participate in events under an administrator's AWS account in multi-user mode. Participants have AWSDeepRacerDefaultMultiUserAccess permissions to train, evaluate, and store models in the sponsor's account. Participants also configure their racer profile, enter virtual races, and download their models for deploying on a physical AWS DeepRacer vehicle.

In this walkthrough, you perform the following steps:

- Step 1. Perform prerequisites.
- Step 2. Activate multi-user mode on your AWS DeepRacer account.
- Step 3. Invite participants.
- Step 4. Set usage quotas.
- Step 5. Monitor usage of your sponsored participants.

Step 1. Prerequisites for AWS DeepRacer multi-user mode

Complete the following prerequisites for multi-user mode

- Set up your account with AWS DeepRacer admin permissions for multi-user. If you are organizing a race with multi-user mode and performing typical AWS DeepRacer administrator tasks, you need to set up your account as an AWS DeepRacer admin with AWSDeepRacerAccountAdminAccess permission.
- Provide AWS console access and racer policy permission to the participants you want to sponsor.

Set up your account with AWS DeepRacer admin permissions for multi-user

To set up as an AWS DeepRacer admin for multi-user mode, you need to have the IAM AWS DeepRacer administrator policy, AWSDeepRacerAccountAdminAccess, attached to your user, group, or role. Depending on your organization, you may set yourself up with the administrator policy by using the console to create a user or role and attaching the required IAM policy, or you may have your IT administrator provide it. For information about the required administrator policy, see AWSDeepRacerAccountAdminAccess. For more information about IAM policies, see Access Management in the *IAM User Guide*.

Provide AWS console access to your sponsored participants

To provide racers you sponsor with access to the AWS DeepRacer console, we recommend using standard AWS authorization protocols such as AWS IAM Identity Center or AWS Identity and Access Management. You can also provide access through your organization's preexisting SSO. When participants log in to the AWS DeepRacer console using the credentials you provide, they are prompted to create an AWS player account to log in and access the AWS DeepRacer console under your AWS account. For more information about AWS Player accounts, see AWS Player accounts.

Provide AWS console access to sponsored participants using IAM

1. Create an IAM username and password for each participant. See Creating an IAM user in your AWS account .
2. Grant each participant the permissions in AWSDeepRacerDefaultMultiUserAccess. For more information, see AWS managed policies for AWS DeepRacer.
3. Email participants with IAM usernames and passwords as well as a link to the console. Using the provided link and entering their IAM usernames and passwords, participants can access the console. For information about creating IAM users, see Creating an IAM user in your AWS account.

Provide AWS console access to sponsored participants using IAM Identity Center

1. Open the IAM Identity Center console at https://console.aws.amazon.com/singlesignon/, create a custom permission set, and assign users to the account. For more information, see Permission sets.
2. When creating the custom permission set, provide the following values:

- **Relay state:** `https://console.aws.amazon.com/deepracer/home?region=us-east-1#getStarted`

> **Note**
>
> The **Relay state** redirects participants within the account to a specified URL; in this case, it directs them to the AWS DeepRacer console.

- **AWS managed policies:** `AWSDeepRacerDefaultMultiUserAccess`

After you have fulfilled the prerequisites, you are ready to activate multi-user mode and invite participants to race through your account.

Step 2: Activate multi-user account mode

After you have set up your AWS DeepRacer admin account and granted console access and permissions to your sponsored participants, you can activate multi-user mode on your AWS DeepRacer account.

> **Note**
>
> By default, there are quotas on accounts sponsoring participants in multi-user mode. For more information, see the section on account quotas in Monitor usage.

1. In the left navigation pane, navigate to **Multi-user management** and the **Setup** page.
2. In **Enable multi-user account mode**, turn on **Enable multi-user mode**.
3. In the **Enable multi-user mode** dialog box, select the checkboxes to confirm that your sponsored participants have required access and permissions.
4. Choose **Enable multi-user mode**.

When you meet the prerequisites and activate multi-user mode, each of your sponsored participants can create races and train models with all training and storage charges billed to the administrator's AWS account. By default, a participant has a quota of 3 concurrent models and can manage up to 10 open or future races at a time (includes LIVE, Classic, and Student races).

Disable multi-user account mode

Disabling multi-user mode ensures that no new profiles can be created under your administrator account and the profiles of previously sponsored participants are no longer visible on the administrator's account. Participants are no longer prompted to log in to their AWS player accounts and can't access or train models created under the administrator's account.

The administrator can download, save, and import sponsored participants' models.

1. Navigate to **Multi-user management** and the **Setup** page.
2. In **Disable multi-user account mode**, choose **Disable multi-user mode**.
3. In the **Disabling multi-user mode** dialog box, select the checkbox to confirm that you want to disable multi-user mode. Choose **Disable multi-user mode**.

   Multi-user mode is disabled.

> **Note**
>
> All models created under a sponsoring AWS multi-user account persist and model storage costs continue on the AWS account until models are deleted.

Step 3: Invite participants to be sponsored

You can invite participants to train and race as sponsored participants by using the provided email template.

To invite participants

1. In the left navigation pane, navigate to **Multi-user management** and the **Setup** page. Under **Set up multi-user mode** in the **Invite users** section, choose **View invite template**.

2. Copy the email template that appears into your email client application and use it to craft an email to send to the participants you want to invite to be sponsored. If you are using your company's existing SSO, you can include a SSO URL for your participants to use. Alternatively, you can provide IAM credentials for the participants to use to log in to the AWS console.

Step 4: Set usage quotas

After your sponsored participants have received their invitation email and have created their profiles under your account, they appear in the **Sponsored users** list in the **Monitor Usage** screen. In this screen, you can then set usage quotas on the number of available training hours and models for sponsored participants. By setting quotas, you can control costs per participant under your account and ensure that participants can't exceed their usage quota. You can also increase or decrease usage quotas as needed to provide sponsored participants with the hours they need to effectively train an AWS DeepRacer model.

> **Note**
>
> By default, sponsored participants in multi-user mode receive 5 hours of training time.

To edit usage quotas for sponsored racers

1. In the left navigation pane, navigate to **Multi-user management** and the **Monitor usage** screen. In the **Monitor usage** screen in **Sponsored users**, select the participants for whom you want to set quotas. Choose **Actions** to open the dropdown list and choose **Set usage quotas**.
2. In the **Set usage quotas** pop-up, enter the **Maximum training hours** and **Maximum model count** for the participants you selected. Choose **Confirm** to keep your changes or **Cancel** to discard them.

Step 5: Monitor usage

You can monitor the usage of your sponsored participants, including estimated spending and training model hours. You can also pause sponsorship of participants, delete models, and view summaries of usage. You perform all tasks related to monitoring usage in AWS DeepRacer **Multi-user management** in the **Monitor usage** page.

All information about expenses for sponsored racers is an estimate only and should not be used for budgeting or cost accounting purposes. Estimates are in USD and do not reflect any special pricing. For more information about pricing, see Pricing.

Account quotas for multi-user mode

By default, a sponsoring account in multi-user mode has the following quotas which are shared among all sponsored profiles:

- 100 concurrent training jobs
- 100 concurrent evaluation jobs
- 100 open or future races (includes LIVE, Classic, and Student races)
- 1000 cars
- 50 private leaderboards

To adjust these quotas, contact Customer Service.

To view an estimate of spending

On the **Monitor usage** page, under **Monitor usage**, you can view an estimated summary of your participants' usage.

To set up billing alerts

You can set up billing alerts for your account. Billing alerts help you to keep up to date on spending. For more information, see Billing.

To pause sponsorship

You can pause sponsorship of a single participant, multiple participants, or all participants. When you pause sponsorship, your sponsored participants can't create new models or train models under your account. Training that is in progress runs through to completion and is included in estimates for spending. You can resume sponsorship at any time. Participants whose multi-user access has been paused can still view their models and post models to leaderboards, but they can't perform any cost-generating activities.

1. On the **Monitor usage** page, under **Monitor usage**, in the **Sponsored users** section, select the users for whom you want to pause sponsorship.

2. Choose **Pause sponsorship**.
3. In the **Pause sponsorship** dialog box, choose **Pause sponsorship** to pause sponsorship. Choose **Cancel** if you decide that you don't want to pause sponsorship.

To resume sponsorship

You can resume sponsorship of racers for whom you have paused sponsorship.

1. On the **Monitor usage** page, under **Monitor usage**, in the **Sponsored users** section, select the racers for whom you want to resume sponsorship.
2. Choose **Resume sponsorship**.

To view racers' models

- On the **Your Models** page, under **Models**, you can view your models and your users' models.

Next steps

After you have set up and activated multi-user mode, you can take the following steps:

- Create a community race.
- Request an AWS DeepRacer workshop.

Create a community race

Community races provide an exciting way for your sponsored participants to experience reinforcement learning.

You can create community races and invite your sponsored participants.

For more information, see the section called “Create a race quick start”.

Request a workshop

You can request a workshop to learn more about AWS DeepRacer with a 60-minute online or in-person workshop.

For more information, see Workshop.

AWS DeepRacer multi-user experience (participant)

This walkthrough demonstrates the experience of an individual participant whose profile is sponsored by an organization's account in multi-user mode.

AWS DeepRacer provides an exciting way for you to experience reinforcement learning (RL) by training and racing AWS DeepRacer models. Your organization may offer you the opportunity to have your profile sponsored under their AWS account. All charges you generate, including training, evaluating, and storing models, are billed to the AWS account you used to log in. The administrator of the AWS account that sponsors your profile can view your models, cars, and leaderboards; pause your training hours; adjust your training hours and storage quotas; and stop sponsoring your profile.

As part of your sponsored racer sign up process, you create an AWS Player account. The account is a portable profile that you retain and can use with a number of other AWS services. For more information, see AWS Player accounts.

Prerequisites

Your organization's event coordinator shares an invitation to join AWS DeepRacer, which includes login credentials for the AWS console. Use these credentials to log in to the console. You also create a racer profile and an AWS Player account as part of your setup.

This walkthrough covers the following steps:

- Log in to the AWS console using the sponsoring account's credentials.
- Create or log in to an AWS Player account.
- Customize your profile.
- Train models.
- View sponsored usage.
- (Optional) Request additional sponsored hours.

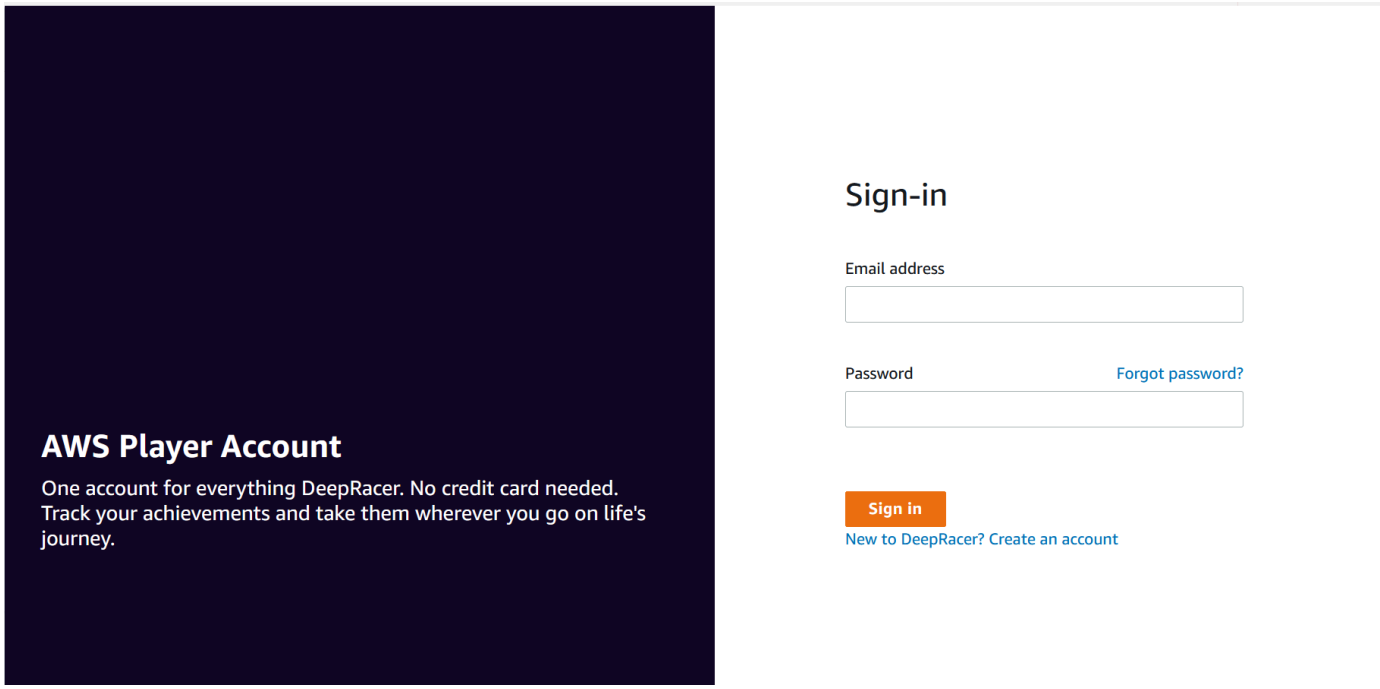
Step 1. Log in to the AWS console using the sponsoring account's credentials

To get started with AWS DeepRacer as a sponsored participant, you log in to the console using the credentials provided in the invitation you received from the event coordinator.

To log in to the AWS console as a sponsored participant

1. Use the credentials provided in the invitation you received from the event coordinator.
2. In the console, navigate to AWS DeepRacer.

   The AWS Player account page appears.



Step 2. Create or log in to an AWS Player account

1. In the **AWS Player account** page, create or log in to an existing AWS Player account.
   - If you do not already have an account, choose **Create account**, enter your email address and a password, then choose **Create your account**.
   - If you already have an AWS Player account, enter your email and password and choose **Sign in**.
2. A message is sent to the email address you specified to validate the account setup.
3. In the **Verification code** box, enter the code you received in the email and choose **Confirm registration**.

> **Note**
>
> Stay on the current page until you have entered your verification code.

You are now logged in to AWS DeepRacer console as a sponsored participant.

4. Proceed to Step 3 to customize your racer profile.

Step 3. Customize your profile

Customize your profile by editing your profile image and adding a racer name. You can update and change your racer profile at any time. You can also add your country of residence and a contact email for receiving communications about prizes earned in the AWS DeepRacer League. Additionally, if you receive achievements for your performance in the AWS DeepRacer League, you can share them on social media from the **Your racer profile** page.

> **Note**
>
> To join in AWS DeepRacer League racing events and train models, you need to create a racer name and add your country of residence. Your racer name must be globally unique. Once you select your country of residence, it is locked in for the racing season.

To customize your racer profile image

1. In the left navigation pane, navigate to the **Your racer profile** page.
2. In the **Your racer profile** page, choose **Edit**.
3. In the **Your racer profile** dialog box, customize your racer profile image by choosing items from the dropdown lists.
4. Choose **Save**.

To customize your racer name

1. In the left navigation pane, navigate to the **Your racer profile** page.
2. In the **Your racer profile** page, choose **Edit**.
3. In the **Your racer profile** dialog box, choose **Change your racer name** and enter a name for your profile.
4. Choose **Save**.

Step 4. Train models

When you have customized your profile, you are ready to start training models. For more information, see Train and evaluate AWS DeepRacer models.

Step 5. View sponsored usage

You'll want to keep track of your sponsored hours and models so you can get the most out of them.

To view sponsored hours usage and stored models

- In **Your racer profile** page, see **Sponsored usage** for total hours used and number of stored models.

Step 6. (Optional) Request additional sponsored hours

As a sponsored participant, you receive five hours of free training time. If you run out of your free sponsored hours, you can request additional hours from your account administrator or event organizer. Alternatively, if you don't have access to additional sponsored hours, you can continue your journey with AWS DeepRacer by creating your own AWS DeepRacer account. For information about training and storage costs, see Pricing.

Educator tools for AWS DeepRacer Student

This section provides you with information and resources to integrate the AWS DeepRacer Machine Learning curriculum in the classroom, hold AWS DeepRacer Student hands-on labs, and create student community races.

Integrate AWS DeepRacer Student in the classroom

If you're an educator that's just getting started with AWS DeepRacer, we recommend that you read the AWS DeepRacer Student educator playbooks.

Curriculum Playbook

The AWS DeepRacer Student Curriculum Playbook outlines each AWS DeepRacer Student module's overview, learning objectives, learning outcomes, key concepts, support material, and assessment and activity suggestions.

Student Labs Playbook

The AWS DeepRacer Student Labs Playbook provides the information and resources for educators to hold AWS DeepRacer Student hands-on labs. Hands-on labs consist of virtual events like AWS DeepRacer Student League races, Private Community Races, Live Virtual Racing, and in person events with a physical track and AWS DeepRacer device.

Create student community races

After you get started with the educator playbooks, use **Community races** in the AWS DeepRacer console to create races for students in AWS DeepRacer Student League. Share a race invitation link to invite student race participants.

Educators need an AWS account to sign into the AWS DeepRacer console to create and organize races, but students only need an email address to log in to AWS DeepRacer Student League, update their profile, start taking free courses, and create AWS DeepRacer models. Educators can also use an email address to create an account in AWS DeepRacer Student League to preview the curriculum, try out the race experience, and monitor your students' progress.

Continue to one of the following topics to create or manage an AWS DeepRacer Student virtual race.

Topics

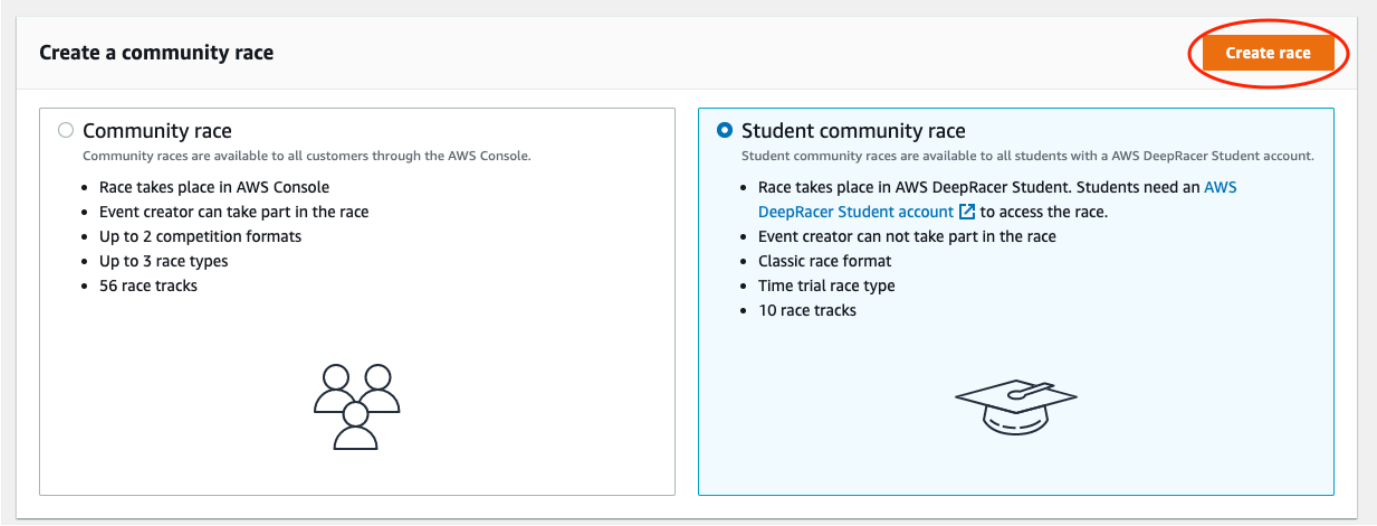
Create an AWS DeepRacer Student community race

You can set up a virtual race quickly using the default student community race settings.

Student community races are asynchronous events that do not require real-time interaction. Participants must receive an invitation link to submit a model to the race and view the leaderboard. Racers can submit unlimited models at any time within a date range to climb the leaderboard. Results and videos for classic races are viewable for submitted models on the leaderboard page as soon as the race is initiated.

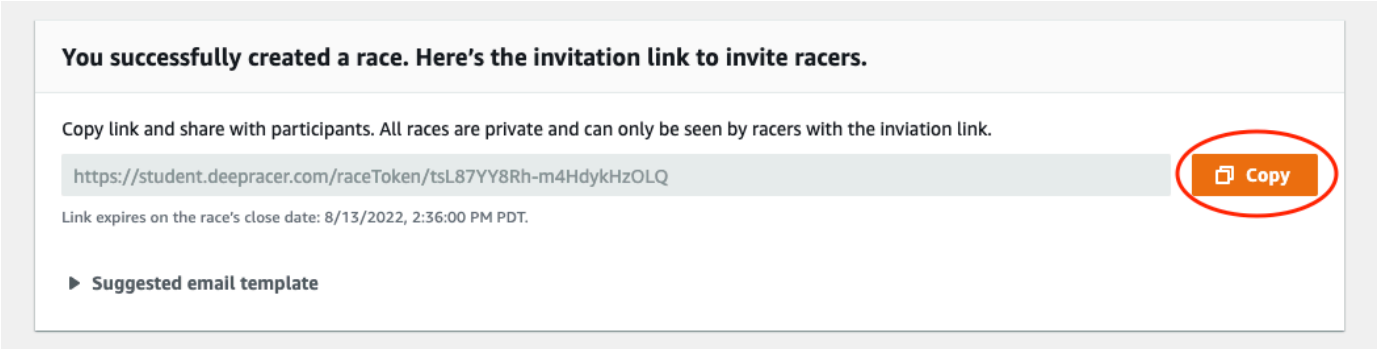
To begin creating a student community race

1. Open the AWS DeepRacer console.
2. On the **Community races** page, choose **Student community race**.
3. Select **Create race**.



4. Enter an original, descriptive name for the race.
5. Specify the start date and time of the event in 24-hour format. The AWS DeepRacer console automatically recognizes your time zone. Also enter an end date and time.

6. To use the default race settings, choose **Next**. When you're ready to learn about all of your options, go to the section called “Customize a student race”.
7. On the **Review race details** page, check the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.
8. To share your race, choose **Copy** and paste the link into the suggested email template, text messages, and your favorite social media applications. All races can be seen only by Only racers with an invitation link can see a race. The link expires on the race's close date.



9. As your student race time frame comes to a close, take note of who has entered a model and who still needs to do so under **Racers** on the **Manage races** page.

Choose Manage races to change the race track selected, add a race description, pick a ranking method, decide how many resets racers are allowed, determine the minimum number of laps an RL model must complete to qualify for your race, set the off-track penalty, and customize other race details.

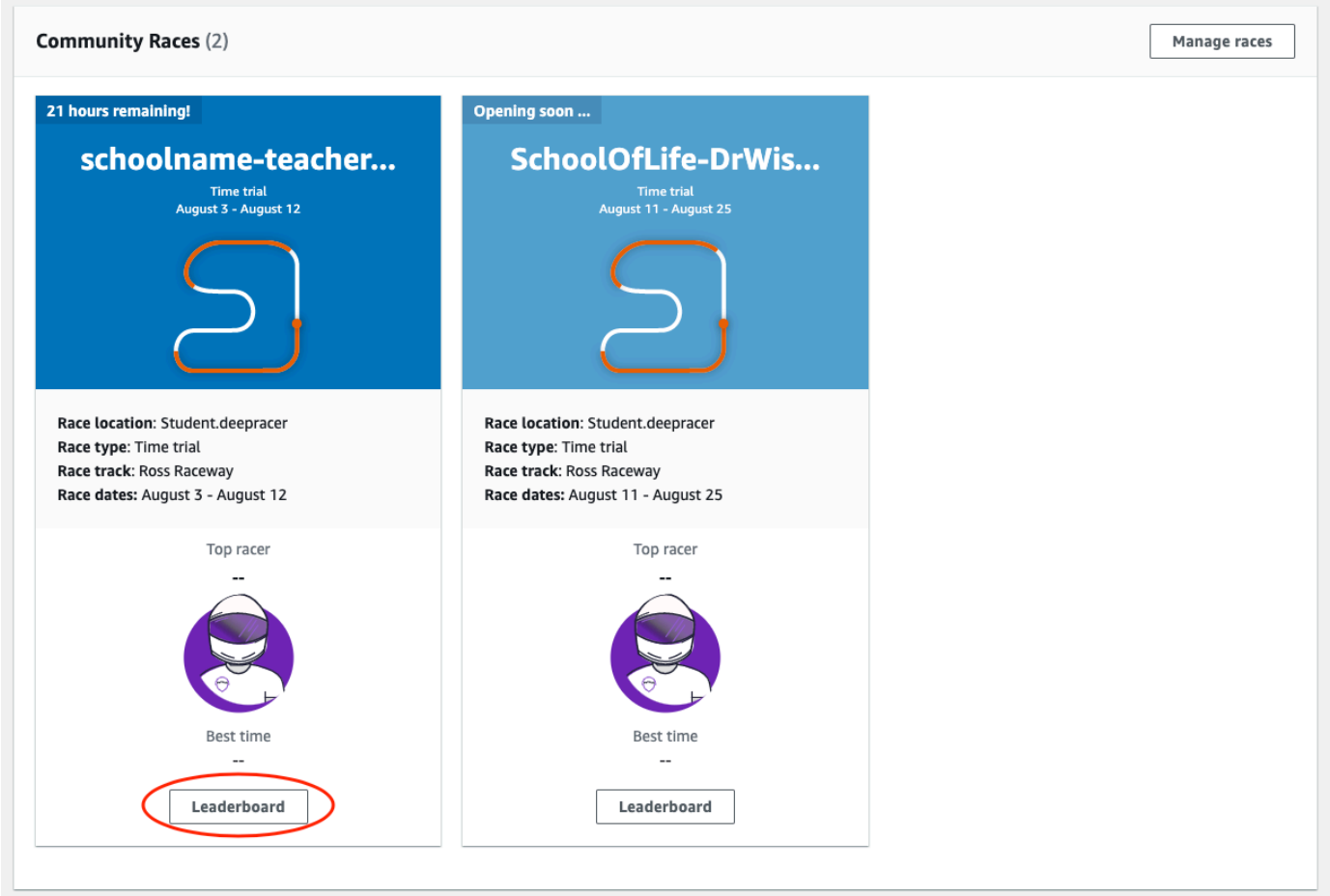
> **Note**
>
> You will only see your students' aliases in the **Racers** tab and on the **Leaderboard**, so make note of which alias is associated with which student.

Customize an AWS DeepRacer Student community race

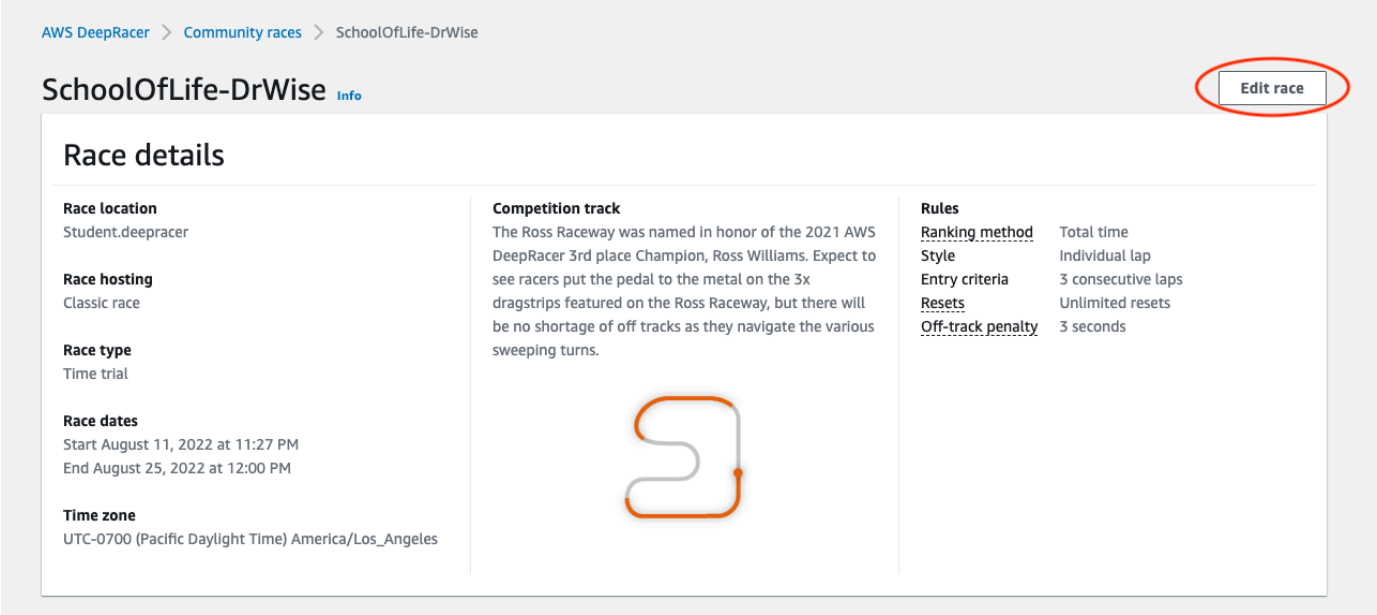
To create a race that is tailored for your group, add customizations that increase or decrease a race's complexity and challenge.

To customize a student race

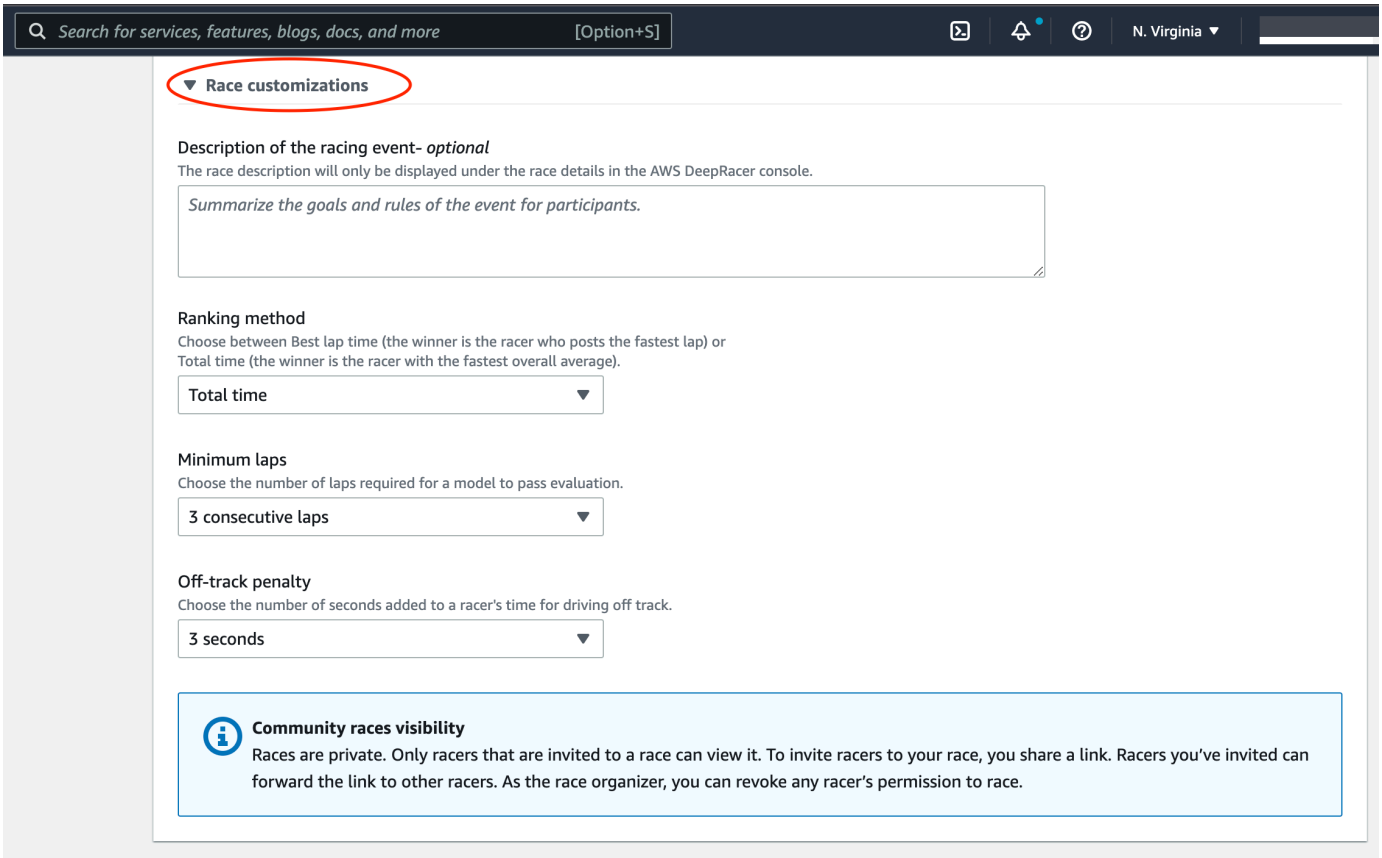
1. Open the AWS DeepRacer console.
2. Choose **Community races**.
3. On the **Community races** page, choose the **Leaderboard** for the race you want to customize.



4. On the **Race details** page, choose **Edit race**.



5. Expand **Race customizations**.



6. Optionally, write a description for your race that summarizes the goals and rules of the event for participants. The description will appear in your leaderboard details.

7. For **Ranking method** for a classic race, choose between the **Best lap time**, where the winner is the racer who posts the fastest lap; **average time**, where, after multiple attempts within the time-frame of the event, the winner is the racer with the best average time; or **Total time**, where the winner is the racer with the fastest overall average.
8. Choose a value for **Minimum laps**, which is the number of consecutive laps a racer must complete to qualify for submission of the result to the race's leaderboard. For a beginners' race, choose a smaller number. For advanced users, choose a larger number.
9. For **Off-track penalty**, choose the number of seconds to add to a racer's time when their RL model drives off track.
10. You have now completed all the customization options for your student community race. Choose **Next** to review the race details.
11. On the **Review race details** page, review the race specifications. To make changes, choose **Edit** or **Previous** to return to the **Race details** page. When you're ready to get the invitation link, choose **Submit**.
12. Choose **Done**. The **Manage races** page is displayed.

To learn how to use our email template to invite new racers, remove racers from your race, check on racers' model submission status and more, see Manage Community Races.

Manage an AWS DeepRacer Student community race

All student community races are only visible to individuals who have received an invitation link. Participants can freely forward invitation links. However, to join a race, participants need an AWS DeepRacer Student account. First-time users must complete the account creation process before they can join the race. Students only need an email address to set up an account.

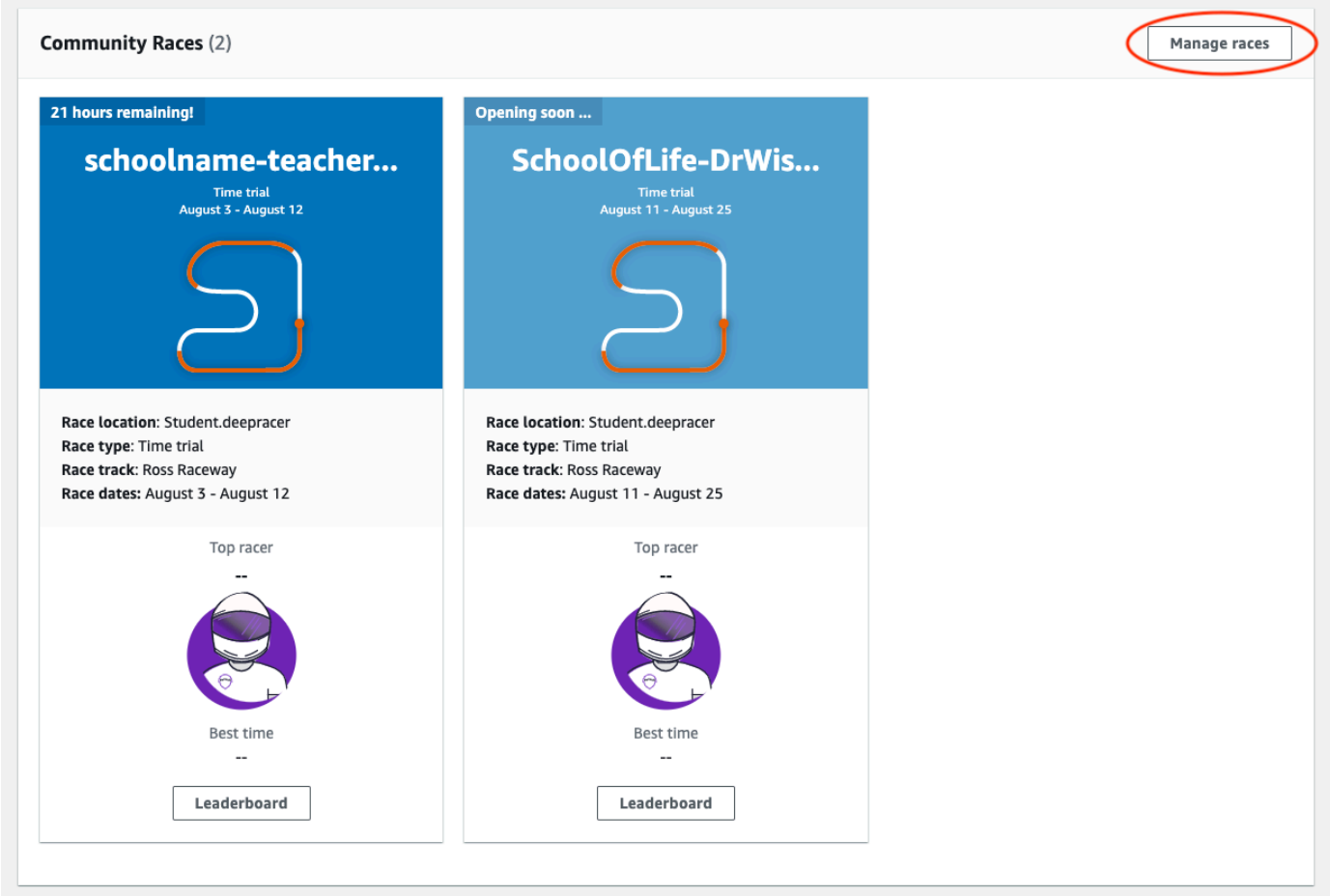
As the race organizer, you can:

- Edit race details (including the start and end dates)
- Remove participants
- End races
- Delete races

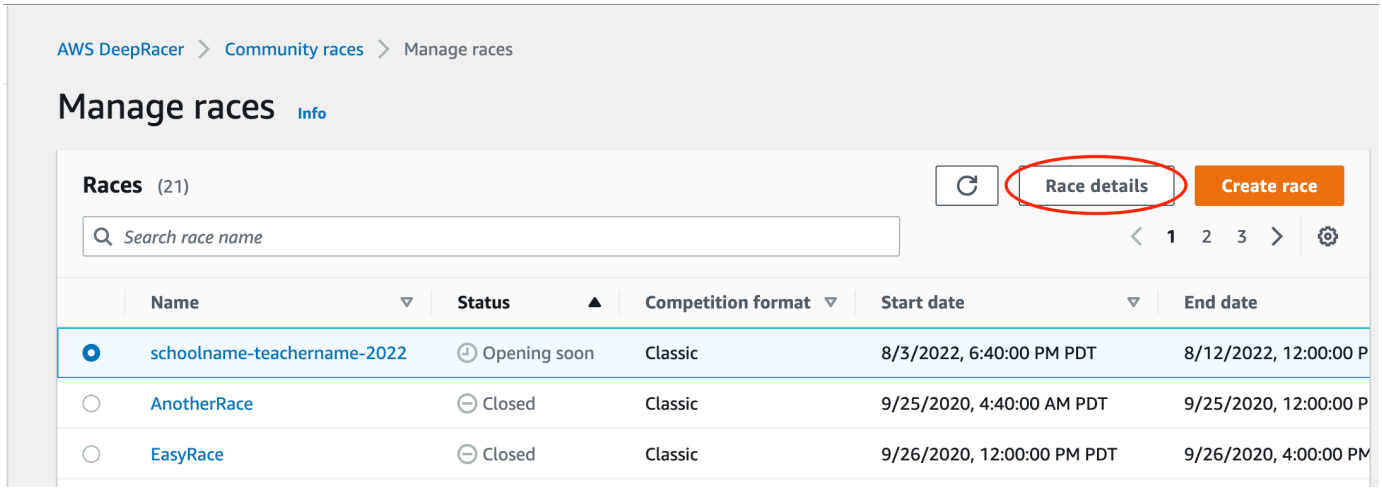
> **Note**
>
> You will only see your students' aliases in the **Racers** tab and on the **Leaderboard**, so make note of which alias is associated with which student.

To manage an AWS DeepRacer Student community race

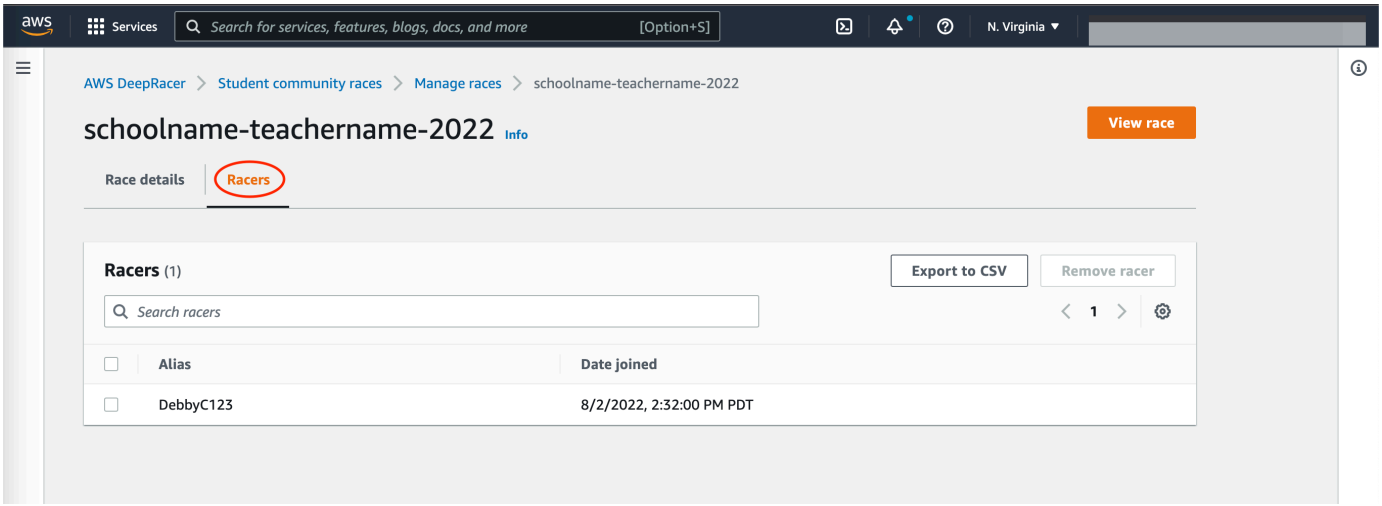
1. Sign in to the AWS DeepRacer console.
2. Choose **Community races**.
3. Select **Manage races**.



4. On the **Manage races** page, choose the race that you want to manage.
5. Choose **Race details** and select **Edit**.



6. To view the event's leaderboard, choose **View race**.
7. To reset the event's invitation link, choose **Reset invitation link**. Resetting the invitation link prevents anyone who has not yet chosen the original link from the race. Resetting the invitation link does not affect existing participants in the race.
8. To end a race, choose **End race**. This ends the race immediately.
9. To delete the event, choose **Delete race**. This permanently removes this race from the AWS console and AWS DeepRacer Student.
10. To remove a participant, choose the **Racers** tab, select one or more participants and select **Remove racer**. Removing a participant from an event prevents them from joining the race.



Security for AWS DeepRacer

To use AWS DeepRacer to train and evaluate reinforcement learning, your AWS account must have appropriate security permissions to access dependent AWS resources, including Amazon VPC to run training jobs and an Amazon S3 bucket to store trained model artifacts.

The AWS DeepRacer console provides a way for you to have the required security settings set up for the dependent services. This section documents the AWS services AWS DeepRacer depends as well as the the IAM roles and policy defining the required permissions to access the dependent services.

Topics

Data protection in AWS DeepRacer

AWS DeepRacer conforms to the AWS shared responsibility model, which includes regulations and guidelines for data protection. AWS is responsible for protecting the global infrastructure that runs all the AWS services. AWS maintains control over data hosted on this infrastructure, including the security configuration controls for handling customer content and personal data. AWS customers and APN partners, acting either as data controllers or data processors, are responsible for any personal data that they put in the AWS Cloud.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM), so that each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.

- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with AWS DeepRacer or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into AWS DeepRacer or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

For more information about data protection, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

AWS DeepRacer-Dependent AWS Services

AWS DeepRacer uses the following AWS services to manage required resources:

Amazon Simple Storage Service

To store trained model artifacts in an Amazon S3 bucket.

AWS Lambda

To create and run the reward functions.

AWS CloudFormation

To create training jobs for AWS DeepRacer models.

SageMaker AI

To train the AWS DeepRacer models.

The dependent AWS Lambda, AWS CloudFormation, and SageMaker AI in turn use other AWS services including Amazon CloudWatch and Amazon CloudWatch Logs.

The following table shows AWS services used by AWS DeepRacer, directly or indirectly.

AWS Services that AWS DeepRacer uses directly or indirectly

AWS service principal	Comments
application-autoscaling	• Indirectly called by SageMaker AI to automatically scale its operations.
cloudformation	• Directly called by AWS DeepRacer to create account resources.
cloudwatch	• Directly called by AWS DeepRacer to log its operations. • Indirectly called by SageMaker AI to log its operations.
ec2	• Indirectly called by AWS CloudFormation and SageMaker AI to create and run training jobs.
kinesisvideo	• Directly called by AWS DeepRacer to view cached training streams.
lambda	• Directly called by AWS DeepRacer to create and run the reward functions.
logs	• Directly called by AWS DeepRacer to log its operations. • Indirectly called by AWS Lambda to log its operations.
s3	• Indirectly called by SageMaker AI to perform SageMaker AI-specific storage operations. • Directly called by AWS DeepRacer to create, list, and delete buckets that have names starting with "`deepracer-`." Also called to download objects from the buckets, upload objects to the buckets, or delete objects from the buckets.

AWS service principal	Comments
sagemaker	• Directly called by AWS DeepRacer to train reinforcement learning models.

To use AWS DeepRacer to call these services, you must have appropriate IAM roles with required policies attached to them. Learn the details about these policies and roles in Required IAM roles for AWS DeepRacer to call dependent AWS Services.

Required IAM roles for AWS DeepRacer to call dependent AWS Services

Before you create a model, use the AWS DeepRacer console to set up resources for your account. As you do this, the AWS DeepRacer console creates the following IAM roles:

AWSDeepRacerServiceRole

Allows AWS DeepRacer to create required resources and call AWS services on your behalf.

AWSDeepRacerSageMakerAccessRole

Allows Amazon SageMaker AI to create required resources and call AWS services on your behalf.

AWSDeepRacerLambdaAccessRole

Allows AWS Lambda functions to call AWS services on your behalf.

AWSDeepRacerCloudFormationAccessRole

Allows AWS CloudFormation to create and manage AWS stacks and resources on your behalf.

Follow the links to view detailed access permissions in the AWS IAM console.

AWS Identity and Access Management for AWS DeepRacer

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use DeepRacer resources. IAM is an AWS service that you can use with no additional charge.

Topics

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting AWS DeepRacer identity and access)
- **Service administrator** - determine user access and submit permission requests (see How AWS DeepRacer works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for AWS DeepRacer)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

IAM Users and Groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM Roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing cccess using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other Policy Types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.

- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS DeepRacer works with IAM

Before you use IAM to manage access to DeepRacer, learn what IAM features are available to use with DeepRacer.

IAM features you can use with AWS DeepRacer

IAM feature	DeepRacer support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Yes

IAM feature	DeepRacer support
Temporary credentials	Yes
Principal permissions	Yes
Service roles	Yes
Service-linked roles	No

To get a high-level view of how DeepRacer and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for DeepRacer

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for DeepRacer

To view examples of DeepRacer identity-based policies, see Identity-based policy examples for AWS DeepRacer.

Resource-based policies within DeepRacer

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified

principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for DeepRacer

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of DeepRacer actions, see Actions defined by AWS DeepRacer in the *Service Authorization Reference*.

Policy actions in DeepRacer use the following prefix before the action:

```
deepracer
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "deepracer:action1",
      "deepracer:action2"
         ]
```

To view examples of DeepRacer identity-based policies, see Identity-based policy examples for AWS DeepRacer.

Policy resources for DeepRacer

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

To see a list of DeepRacer resource types and their ARNs, see Resources defined by AWS DeepRacer in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by AWS DeepRacer.

To view examples of DeepRacer identity-based policies, see Identity-based policy examples for AWS DeepRacer.

Policy condition keys for DeepRacer

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of DeepRacer condition keys, see Condition keys for AWS DeepRacer in the *IAM User Guide* in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by AWS DeepRacer.

To view examples of DeepRacer identity-based policies, see Identity-based policy examples for AWS DeepRacer.

Access control lists (ACLs) in DeepRacer

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with DeepRacer

Supports ABAC (tags in policies): Yes

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using Temporary credentials with DeepRacer

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for DeepRacer

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for DeepRacer

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

> **Warning**
>
> Changing the permissions for a service role might break DeepRacer functionality. Edit service roles only when DeepRacer provides guidance to do so.

Service-linked roles for DeepRacer

Supports service-linked roles: No

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing service-linked roles, see AWS services that work with IAM. Find a service in the table that includes a `Yes` in the **Service-linked role** column. Choose the **Yes** link to view the service-linked role documentation for that service.

Identity-based policy examples for AWS DeepRacer

By default, users and roles don't have permission to create or modify DeepRacer resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by DeepRacer, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for AWS DeepRacer in the *Service Authorization Reference*.

Topics

Policy best practices

Identity-based policies determine whether someone can create, access, or delete DeepRacer resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API

operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

Using the DeepRacer console

To access the AWS DeepRacer console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the DeepRacer resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the DeepRacer console, also attach the DeepRacer `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
```

```
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS managed policies for AWS DeepRacer

An AWS managed policy is a standalone policy that is created and administered by AWS. AWS managed policies are designed to provide permissions for many common use cases so that you can start assigning permissions to users, groups, and roles.

Keep in mind that AWS managed policies might not grant least-privilege permissions for your specific use cases because they're available for all AWS customers to use. We recommend that you reduce permissions further by defining customer managed policies that are specific to your use cases.

You cannot change the permissions defined in AWS managed policies. If AWS updates the permissions defined in an AWS managed policy, the update affects all principal identities (users, groups, and roles) that the policy is attached to. AWS is most likely to update an AWS managed policy when a new AWS service is launched or new API operations become available for existing services.

For more information, see AWS managed policies in the *IAM User Guide*.

The following AWS-managed policies are specific to using AWS DeepRacer multi-user mode to sponsor multiple participants under your AWS account.

- `AWSDeepRacerAccountAdminAccess` Grants required AWS DeepRacer permissions for multi-user account admin.
- `AWSDeepRacerDefaultMultiUserAccess` Grants required AWS DeepRacer permissions to use the AWS DeepRacer console.

Topics

AWSDeepRacerAccountAdminAccess managed policy for AWS DeepRacer administrators

To enable multiple profiles to use your AWS account ID and billing information with AWS DeepRacer, attach the `AWSDeepRacerAccountAdminAccess` policy.

You can attach the `AWSDeepRacerAccountAdminAccess` policy to the IAM identity you want to use for sponsoring other racers.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeepRacerAdminAccessStatement",
            "Effect": "Allow",
            "Action": [
                "deepracer:*"
            ],
            "Resource": [
                "*"
            ],
```

```
            "Condition":
            {
                "Null": {
                    "deepracer:UserToken": "true"
                }
            }
        }
    ]
}
```

AWSDeepRacerDefaultMultiUserAccess managed policy for AWS DeepRacer multi-user racers

The policy, `AWSDeepRacerDefaultMultiUserAccess`, gives AWS DeepRacer racers access to all AWS DeepRacer actions except multi-user account admin actions.

You can attach the `AWSDeepRacerDefaultMultiUserAccess` policy to the IAM identities of participants you want to sponsor under your account.

JSON

```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "deepracer:Add*",
    "deepracer:Remove*",
    "deepracer:Create*",
    "deepracer:Perform*",
    "deepracer:Clone*",
    "deepracer:Get*",
    "deepracer:List*",
    "deepracer:Edit*",
    "deepracer:Start*",
    "deepracer:Set*",
    "deepracer:Update*",
    "deepracer:Delete*",
    "deepracer:Stop*",
    "deepracer:Import*",
```

```
    "deepracer:Tag*",
    "deepracer:Untag*"
   ],
   "Resource": [
    "*"
   ],
   "Condition": {
    "Null": {
     "deepracer:UserToken": "false"
    },
    "Bool": {
     "deepracer:MultiUser": "true"
    }
   }
  },
  {
   "Effect": "Allow",
   "Action": [
    "deepracer:GetAccountConfig",
    "deepracer:GetTrack",
    "deepracer:ListTracks",
    "deepracer:TestRewardFunction"
   ],
   "Resource": [
    "*"
   ]
  },
  {
   "Effect": "Deny",
   "Action": [
    "deepracer:Admin*"
   ],
   "Resource": [
    "*"
   ]
  }
 ]
}
```

AWS DeepRacer updates to AWS managed policies

View details about updates to AWS managed policies for AWS DeepRacer since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the AWS DeepRacer Document history page.

Change	Description	Date
`AWSDeepRacerAccountAdminAccess` and `AWSDeepRacerDefaultMultiUserAccess` policies added	New managed policies added so you can sponsor multiple participants under one AWS DeepRacer account using multi-user mode.	October 26, 2021
AWS DeepRacer started tracking policy changes.	AWS DeepRacer started tracking changes for its AWS managed policies.	October 26, 2021

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in resource policies to limit the permissions that AWSDeepRacerLong gives another service to the resource. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The value of `aws:SourceArn` must be s3:::your-bucket-name.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know

the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename::123456789012:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in AWSDeepRacer to prevent the confused deputy problem.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1586917903457",
            "Effect": "Allow",
            "Principal": {
                "Service": "deepracer.amazonaws.com"
                },
            "Action": [
                "s3:GetObjectAcl",
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": [
               "arn:aws:s3:::your-bucket-name",
               "arn:aws:s3:::your-bucket-name/*"
            ],
            "Condition": {
                "StringEquals": {
                    "aws:SourceArn": "arn:${Partition}:deepracer:$us-east-1:
${Account}:model/reinforcement_learning/${ResourceId}"
                }
            }
        }
    ]
}
```

If you use a custom AWS Key Management Service (KMS) resource for this bucket, include the AWS KMS resource policy:

Troubleshooting AWS DeepRacer identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with DeepRacer and IAM.

Topics

I get an authorization error in DeepRacer multi-user account mode

If you are an Administrator with an AWSDeepRacerAccountAdminAccess policy, you may get an authorization error if there is a user-token associated with your session. Administrators should not have any user-tokens associated with a session. To resolve this, clear your cookies.

If the account is in multi-user mode and you are a racer with an AWSDeepRacerDefaultMultiUserAccess policy, you may get an authorization error if there is no user-token associated with your policy. To resolve this, you need to authenticate to your AWS Player profile before continuing to use AWS DeepRacer.

If the account is in single-user mode and you are a racer with an `AWSDeepRacerDefaultMultiUserAccess` policy, you may get an authorization error. To resolve this, check with your AWS account admin because in single-user mode a user with an `AWSDeepRacerDefaultMultiUserAccess` policy cannot use AWS DeepRacer.

I am not authorized to perform an action in DeepRacer

If the AWS Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but does not have the fictional `deepracer:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 deepracer:GetWidget on resource: my-example-widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the *my-example-widget* resource using the `deepracer:GetWidget` action.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to DeepRacer.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in DeepRacer. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to view my access keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

> **Important**
>
> Do not provide your access keys to a third party, even to help find your canonical user ID. By doing this, you might give someone permanent access to your AWS account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see Managing access keys in the *IAM User Guide*.

I'm an administrator and want to allow others to access DeepRacer

To allow others to access DeepRacer, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in DeepRacer. After the permissions are granted, provide the credentials to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my DeepRacer resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether DeepRacer supports these features, see How AWS DeepRacer works with IAM.

- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Tagging

A tag is a custom attribute label that you or AWS assigns to an AWS resource. Each AWS tag has two parts:

- A tag key (for example, `companyname`, `costcenter`, `environment`, `project`, or `secret`). Tag keys are case sensitive.
- An optional field known as a tag value. Omitting the tag value is the same as using an empty string. Like tag keys, tag values are case sensitive.

Together these are known as key-value pairs.

In the AWS DeepRacer service, you can assign tags to cars, RL models, and community races leaderboards. Tag these and other AWS resources that support tagging to indicate that the resources are related. In addition to identifying and organizing your models and leaderboards with tags, you can also use tags to track cost allocation and in IAM policies to help control who can view and interact with your resources. Use the AWS DeepRacer console or the AWS CLI to add, manage, and remove tags.

For more information about using tags, see the Tagging best practices whitepaper.

Tag to Track Cost Allocation

AWS Cost Explorer and Cost and Usage Report support the ability to break down AWS costs by tag. Business tags such as `cost center`, `businessunit`, or `project` can be used to associate AWS costs with an organization's typical financial reporting categories. However, a cost allocation report can include any tag allowing you to easily associate costs with technical or security categories, such as specific applications, environments, or compliance programs. Only a management account in an organization and single accounts that aren't members of an organization have access to the Cost Allocation Tags manager in the Billing and Cost Management console. For more information on using tags to track cost allocation see User-Defined Cost Allocation Tags.

Tag to Manage Access

You can also tag IAM users and roles to manage access to your models and community races leaderboards. To learn how to tag IAM users and roles, see Tagging IAM users and roles. To view a tutorial for creating and testing a policy that allows IAM roles with principal tags to access resources with matching tags, see IAM Tutorial: Define permissions to access AWS resources based

on tags. For more information on using tags to control access to your AWS resources that support tagging see Controlling access to AWS resources using resource tags.

Topics

Add, view, and edit tags for a new resource

Adding tags to a new car, RL model, or community races leaderboard can help you identify, organize, track cost allocation, and manage access to these resources. Add one or more tags (key-value pairs) to a model or leaderboard. For each resource, each tag key must be unique, and each tag key can have only one value, but one resource may have up to 50 tags.

Create and apply the tags one resource at a time in the AWS DeepRacer console or use the Tag Editor to add, edit, or delete multiple resources at once.

> **Important**
>
> Editing tags for an RL model or community races leaderboard can impact access to those resources. Before you edit the name (key) or value of a tag, make sure to review any IAM policies that might use the key or value for a tag to control access to those resources.

To add, view, and edit tags for a new RL model

Use the AWS DeepRacer console to add, view, and edit tags to a new RL model.

1. In **Your models**, choose **Create model**.
2. On the **Create model** page, after filling out the **Training details**, expand the **Tags** heading.
3. Under the **Tags** heading, choose **Add new tag**.
4. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**. For more information about naming tags, see the Best Practices for Naming Tags and Resources topic in the Tagging best practices whitepaper.
5. (Optional) To add another tag, choose **Add new tag** again.
6. (Optional) To remove an individual key or value, select the **X** next to it.
7. (Optional) To remove a key-value pair, choose **Remove**.

8. When you have finished adding tags, choose a track under **Environment simulation** and select **Next**.

After tagging and submitting a new model for training, you can manage its tags during or after training and evaluation under the **Tags** heading at the bottom of the page.

1. Choose **Manage tags**.
2. In the **Manage tags** pop up box, you can remove a tag you have created by selecting the **Remove** button next to the tag you want to remove or choose **Add a new tag** to add a new tag.
3. If you choose to add a new tag, in **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**. For more information about naming tags, see the Best Practices for Naming Tags and Resources topic in the Tagging best practices whitepaper.
4. When you have finished removing and adding tags, choose **Submit**.

To add, view, and edit Tags for a new community races leaderboard

Use the AWS DeepRacer console to add, view, and edit tags to a new community races leaderboard.

1. In **Community races**, choose **Create race**.
2. On the **Race details** page, expand the **Tags** heading.
3. Under the **Tags** heading, choose **Add new tag**.
4. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**. For more information about naming tags, see the Best Practices for Naming Tags and Resources topic in the Tagging best practices whitepaper.
5. (Optional) To add another tag, choose **Add new tag** again.
6. (Optional) To remove an individual key or value, select the **X** next to it.
7. (Optional) To remove a key-value pair, choose **Remove**.
8. When you have finished adding tags, choose a track under **Environment simulation** and select **Next**.

Add, view, and edit tags for an existing resource

Adding tags to an existing AWS DeepRacer RL model or community races leaderboard can help you identify, organize, track cost allocation, and manage access to these resources. Add one or more

tags (key-value pairs) to a model or leaderboard. For each resource, each tag key must be unique, and each tag key can have only one value, but one resource may have up to 50 tags.

Create and apply the tags one resource at a time in the AWS DeepRacer console or use the Tag Editor to add, edit, or delete multiple resources at once.

> **Important**
>
> Editing tags for an RL model or community races leaderboard can impact access to those resources. Before you edit the name (key) or value of a tag, make sure to review any IAM policies that might use the key or value for a tag to control access to those resources.

To add, view, and edit tags for an existing RL model

You can use the AWS DeepRacer console to add, view, or edit tags for an existing RL model.

1. In **Your models**, select a model from the list by choosing its name.
2. Select **Actions**.
3. Choose **Manage tags** from the drop down list.
4. In the **Manage tags** pop up box, you can view, add, or remove a tags:
    a. To add a tag, choose **Add new tag**. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**. For more information about naming tags, see the Best Practices for Naming Tags and Resources topic in the Tagging best practices whitepaper.
    b. To add another tag, choose **Add new tag** again.
    c. To remove an individual key or value, select the **X** next to it.
    d. To remove a key-value pair, choose **Remove**.
5. When you have finished viewing, adding, and removing tags, choose **Submit**.

To add, view, and edit tags for an existing community races leaderboard

1. In **Community races**, choose **Manage races**.
2. On the **Manage races** page, select a race.
3. Select **Actions**.
4. Choose **Manage tags** from the drop down list.

5. In the **Manage tags** pop up box, you can view, add, or remove a tags:

   a. To add a tag, choose **Add new tag**. In **Key**, enter a name for the tag. You can add an optional value for the tag in **Value**. For more information about naming tags, see the Best Practices for Naming Tags and Resources topic in the Tagging best practices whitepaper.

   b. To add another tag, choose **Add new tag** again.

   c. To remove an individual key or value, select the **X** next to it.

   d. To remove a key-value pair, choose **Remove**.

6. When you have finished viewing, adding, and removing tags, choose **Submit**.

Troubleshoot common AWS DeepRacer issues

Here you'll find troubleshooting tips for frequently asked questions as well as late-coming bug fixes.

Topics

How to resolve common AWS DeepRacer LIVE issues

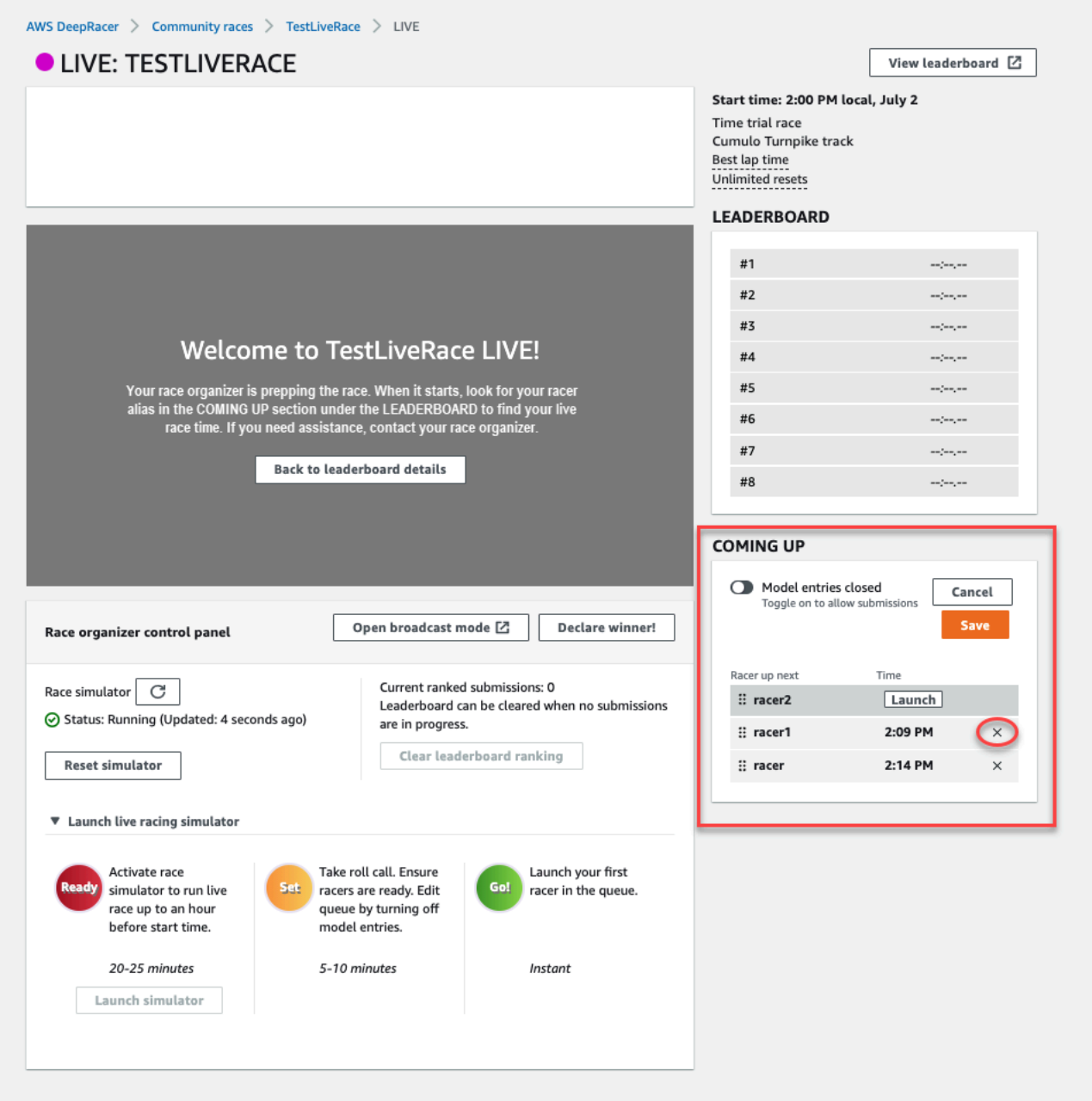
I can't see the race video on the LIVE race page

- If you are using a virtual private network (VPN), verify that it's disconnected during the racing event.
- If your device runs an ad blocker, verify that it's disconnected during the racing event.
- If your home network is running an ad blocker, verify that it's disconnected during the racing event.

A racer's name in the race queue is red

When a racer's name in the coming up section of the **LIVE: <Your Race Name>** page is highlighted in red, it means that something went wrong with the racer's model submission.

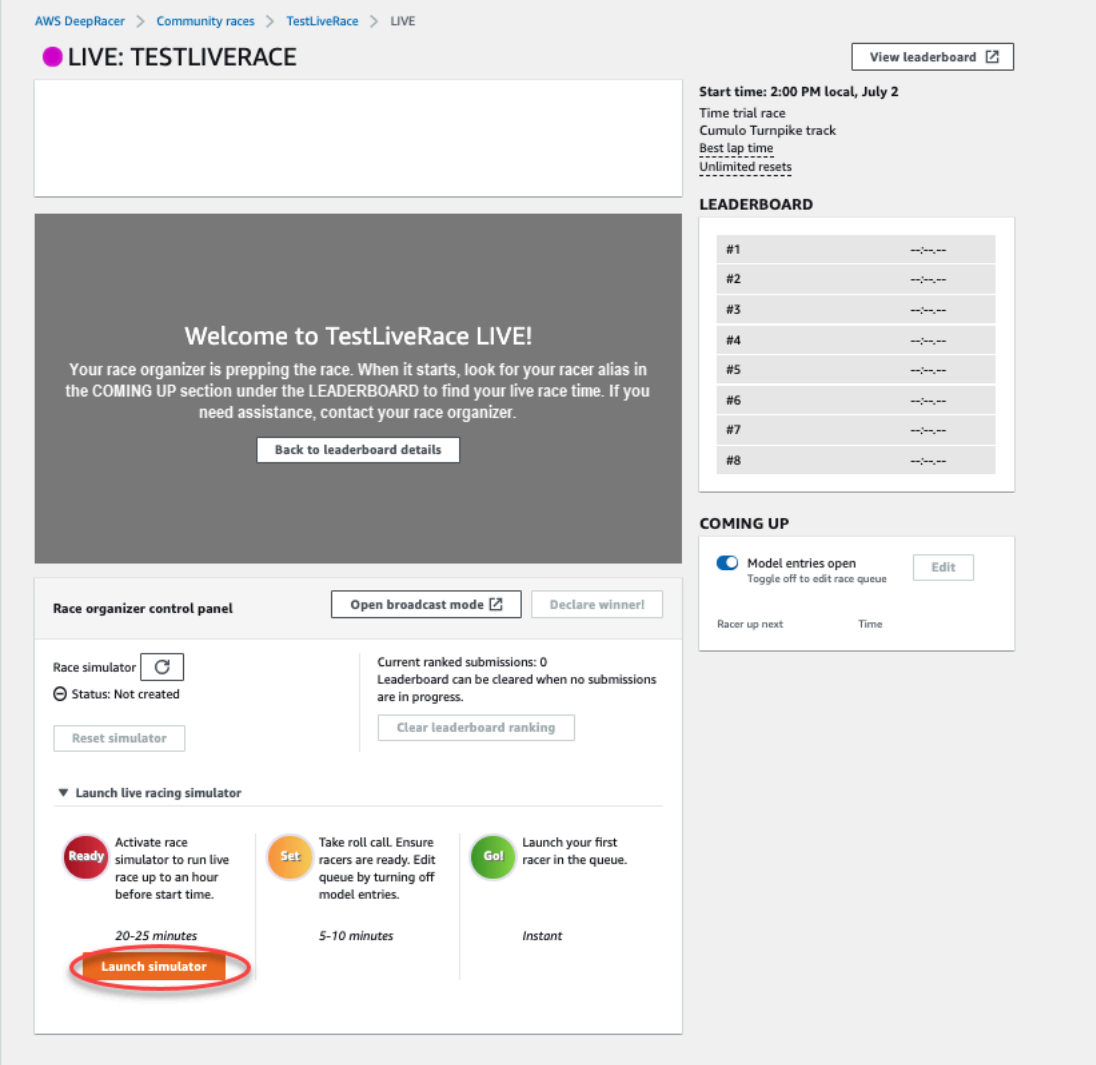
- If you are a race organizer, in the coming up section of the **LIVE: <Your Race Name>** page, choose **Edit** to delete the racer's model submission by selecting **X** on the row containing that racer's name. Next, choose **Save**. See step 11 of the section called “Run a LIVE race” for help reordering your queue.



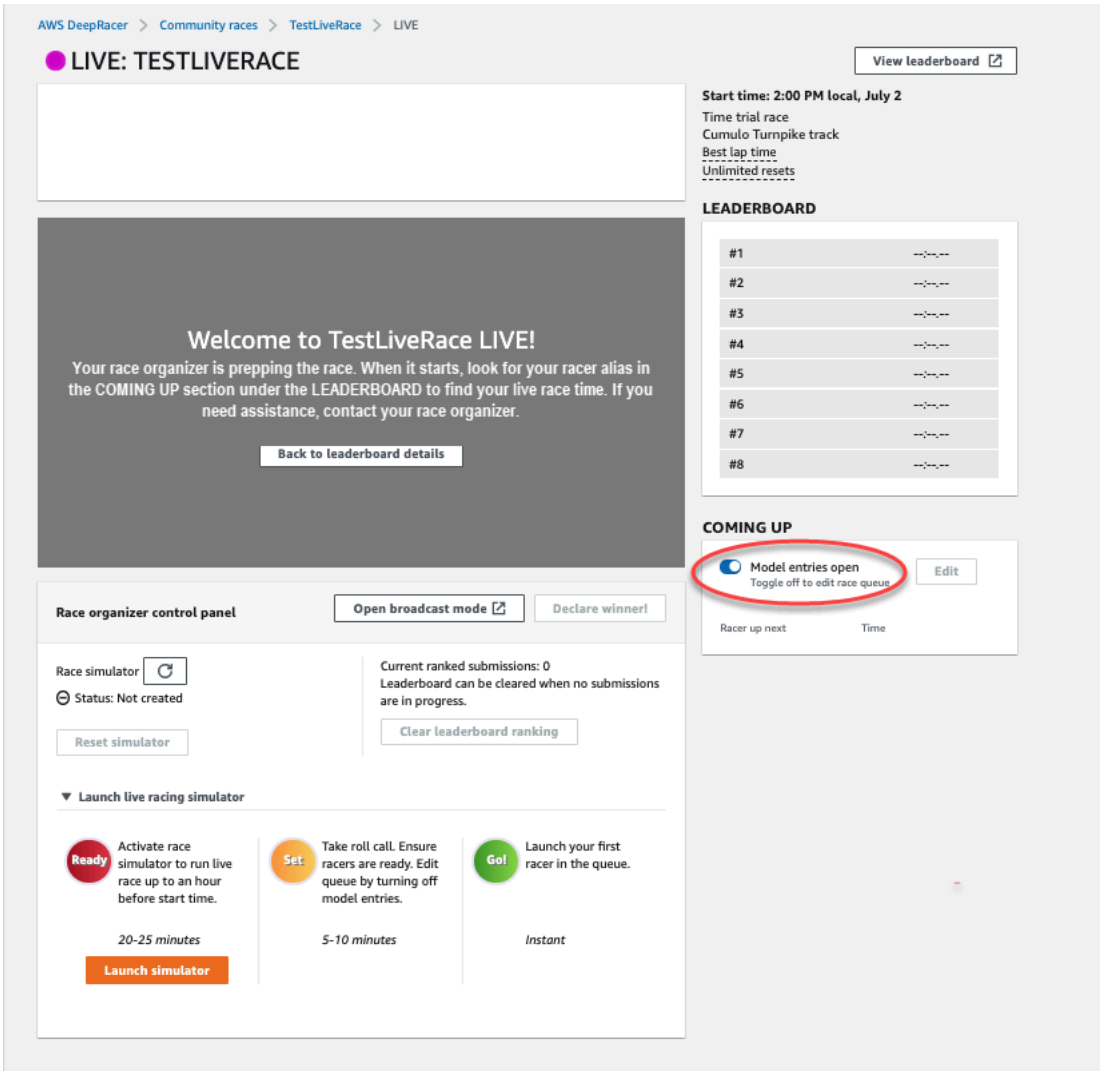
- If you are a racer, resubmit your model to the race. Go to the section called “Run a LIVE race” and choose **To join a LIVE race** for help.

I'm running a LIVE Race and I can't launch the racers

- Verify that you have selected **Launch simulator** under the **Launch live racing simulator** section of the **LIVE: <Your Race Name>** page. For more help see step two of the section called “Run a LIVE race”.



- Verify that you have toggled off **Model entries open** to close submissions under **COMING UP** on the **LIVE: <Your Race Name>** page. For more help see step three of the section called “Run a LIVE race”.



I'm using a Chrome or Firefox browser but I'm still having issues seeing the LIVE race

- Verify that you have the most recent version of the Chrome or Firefox browser. If not, update your browser to the latest version and try viewing the race again.
- If you are using a virtual private network (VPN), verify that it's disconnected.
- If your device runs an ad blocker verify that its disconnected during the racing event.
- If your home network is running an ad blocker, verify that it's disconnected during the racing event.
- If WebRTC is turned off in your internet browser, turn it on during the racing event.

Why can't I connect to the device console with USB connection between my computer and vehicle?

When setting up your vehicle for the first time, you might find it unable to open the device console (also known as the device web server, `https://deepracer.aws`, hosted on the vehicle) after connecting your AWS DeepRacer vehicle to your computer with a micro-USB/USB cable (USB is also referred to as USB-A).

Multiple causes may be behind this. Typically, you can resolve the issue with the following simple remedy.

To activate your device's USB-over-Ethernet network

1. Turn off Wi-Fi on your computer and unplug any Ethernet cable connected to it.
2. Press the **RESET** button on the vehicle to reboot the device.
3. Open the device console by navigating to `https://deepracer.aws` from a web browser on your computer.

If the previous procedure doesn't work, you can check your computer's network preferences to verify that they're properly configured to let the computer connect to the device's network, whose network name is `Deepracer`. To do this, follow the steps in the following procedure.

> **Note**
>
> The instructions below assume you're working with a MacOS computer. For other computer systems, consult with the network preferences documentation for the respective operating system and use the below instructions as a general guide.

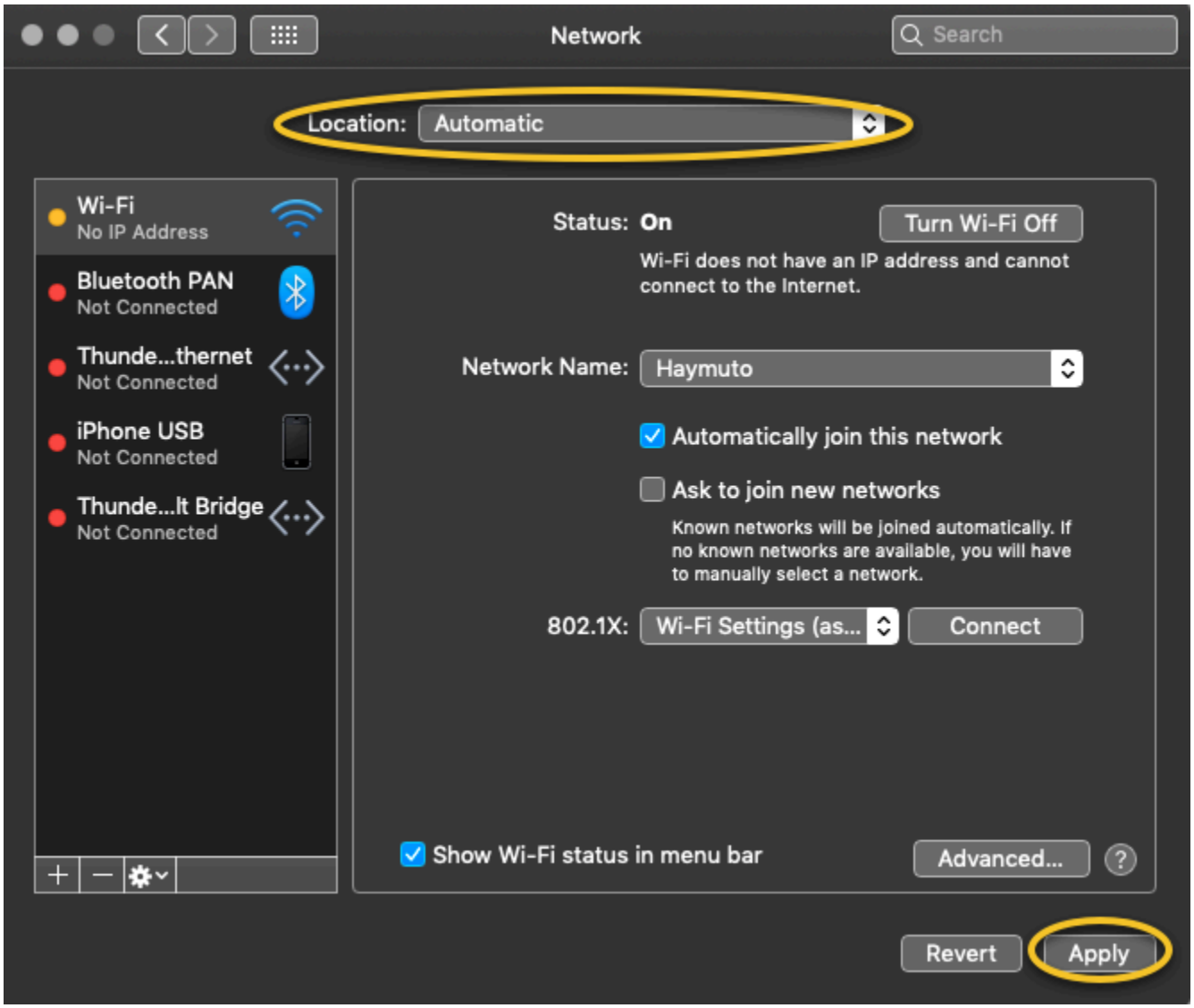
To activate the device's USB-over-ethernet network on your MacOS computer

1. Choose the network icon (on the top-right corner of the display) to open **Network preferences**.

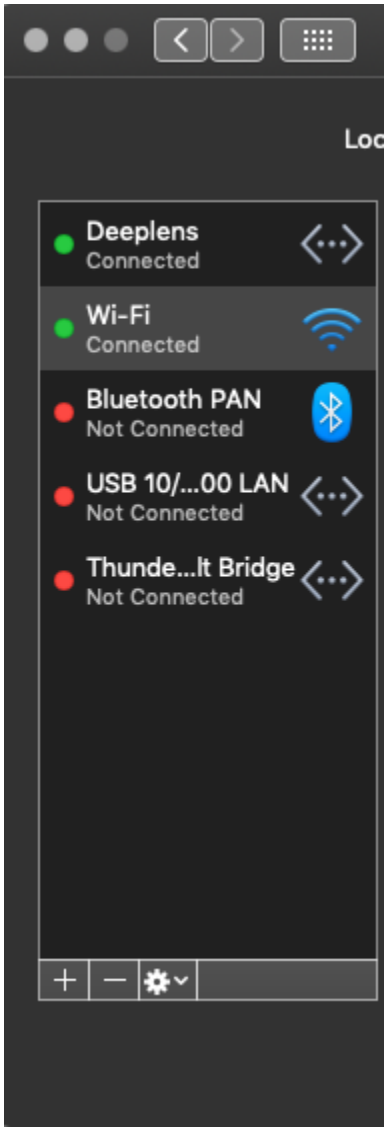
Alternatively, choose **Command+space**, type **Network**, and then choose **Network System Preferences**.

2. Check if **Deepracer** is listed as **Connected**. If **DeepRacer** is listed but not connected, make sure the micro-USB/USB cable is tightly plugged in between the vehicle and your computer.

3. If the **Deepracer** network is not listed there or is listed but not connected when the USB cable is plugged in, choose **Automatic** from the **Location** preference and then choose **Apply**.



4. Verify that the AWS DeepRacer network is up and running as **Connected**.



5. When your computer is connected to the **Deepracer** network, refresh the `https://deepracer.aws` page on the browser, and continue with the rest of **Get Started Guide** instructions of **Connect to Wi-Fi**.
6. If the **Deepracer** network is not connected, disconnect your computer from the AWS DeepRacer vehicle and then reconnect it. When the **Deepracer** network becomes **Connected**, continue with the **Get Started Guide**d instructions.
7. If the **Deepracer** network on the device is still not connected, reboot your computer and AWS DeepRacer vehicle and repeat from **Step 1** of this procedure, if necessary.

If the above remedy still doesn't resolve the issue, the device certificate might have been corrupted. Follow the steps below to generate a new certificate for your AWS DeepRacer vehicle to repair the corrupted file.

To generate a new certificate on the AWS DeepRacer vehicle

1. Terminate the USB connection between your computer and your AWS DeepRacer vehicle by unplugging the micro-USB/USB cable.
2. Connect your AWS DeepRacer vehicle to a monitor (with a HDMI-to-HDMI cable) and to USB keyboard and mouse.
3. Log in to the AWS DeepRacer operating system. If this is the first login to the device operating system, use `deepracer` for the password, when asked for, and then proceed to change the password, as required, and use the updated password for subsequent logins.
4. Open a terminal window and type the following Shell command. You can choose the **Terminal** shortcut from **Applications -> System Tools** on the desktop to open a terminal window. Or you can use the file browser, navigate to the `/usr/bin` folder, and choose **gnome-terminal** to open it.

   ```
   sudo /opt/aws/deepracer/nginx/nginx_install_certs.sh && sudo reboot
   ```

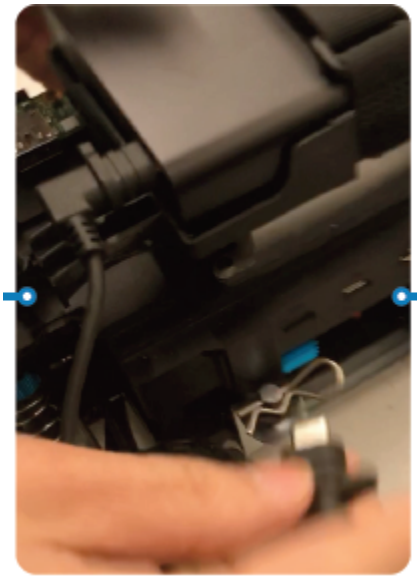
   Enter the password, which you used or updated in the previous step, when prompted.

   The above command installs a new certificate and reboots the device. It also reverts the device console's password to the default value printed at the bottom of the AWS DeepRacer vehicle.
5. Disconnect the monitor, keyboard and mouse from the vehicle and reconnect it to your computer with the micro-USB/USB cable.
6. Follow the second procedure in this topic to verify your computer is indeed connected to the device network before opening the device console (`https://deepracer.aws`) again and, then, continue with the **Connect to Wi-Fi** instructions in **Get Started Guide**.

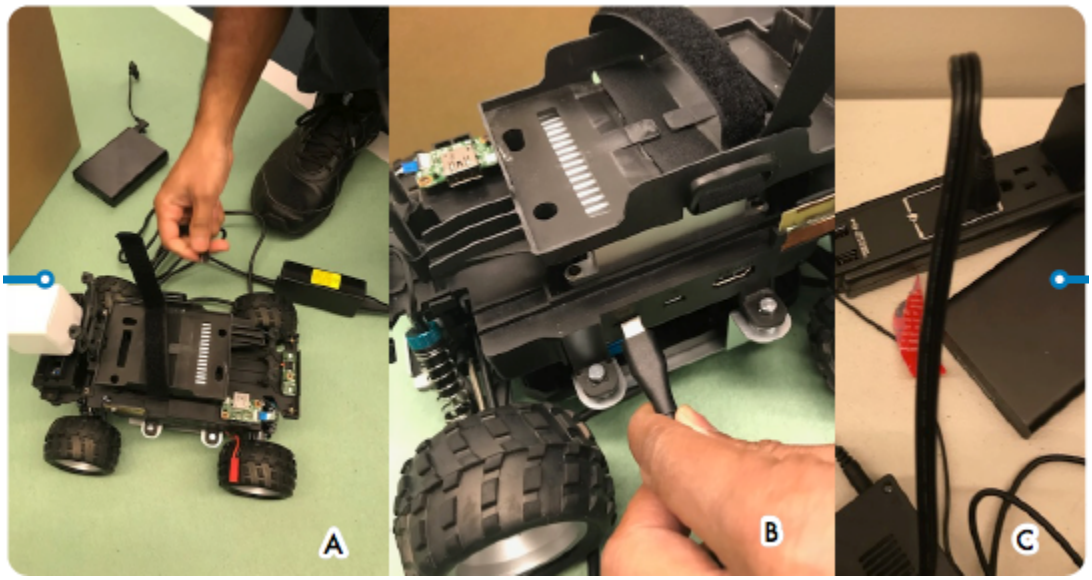
How to switch AWS DeepRacer compute module power source from battery to a power outlet

If the compute module battery level is low when you set up your AWS DeepRacer for the first time, follow the steps below to switch the compute power supply from the battery to a power outlet:

1. Unplug the USB-C cable from the vehicle's compute power port.



2. Attach the AC power cord and the USB-C cable to the computer module power adapter (A). Plug the power cord to a power outlet (C) and plug the USB-C cable the vehicle's computer module power port (B).



How to use a USB flash drive to connect AWS DeepRacer to your Wi-Fi network

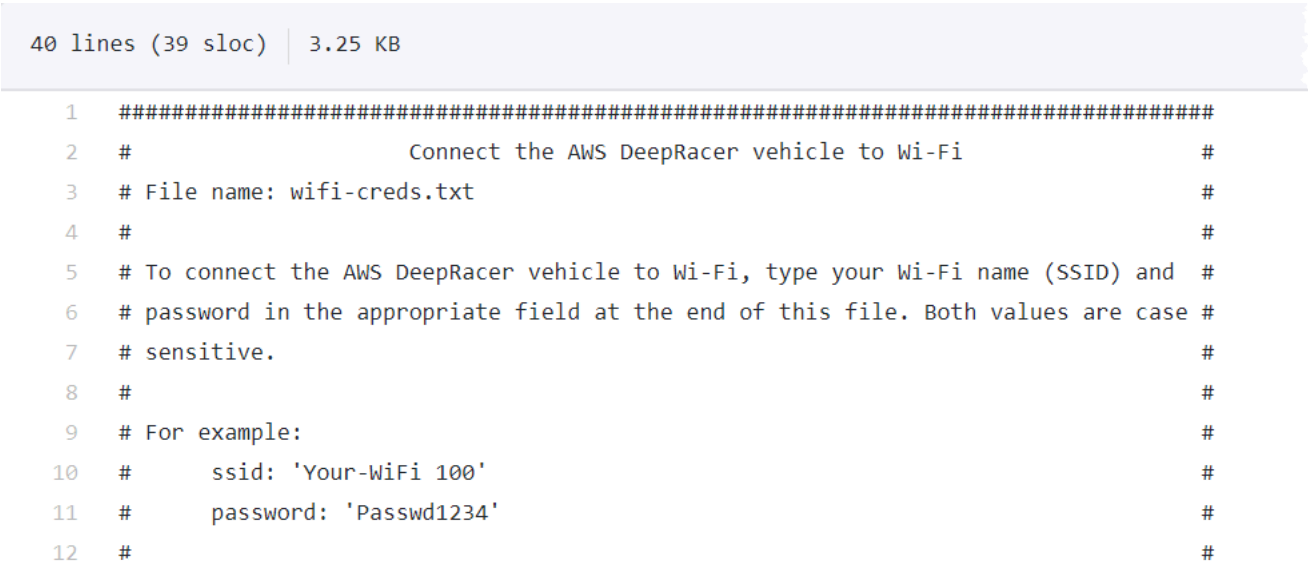
To connect an AWS DeepRacer vehicle to your home or office Wi-Fi network using a USB flash drive, you need the following:

- A USB flash drive
- The name (SSID) and password for the Wi-Fi network that you want to join

> **Note**
>
> AWS DeepRacer does not support Wi-Fi networks that require active captcha verification for user sign-in.

To connect an AWS DeepRacer vehicle to a Wi-Fi network using a USB flash drive

1. Plug the USB flash drive into your computer.
2. Open a web browser on your computer and navigate to https://aws.amazon.com/deepracer/usbwifi. This link opens a text file named `wifi-creds.txt` hosted on GitHub.



3. Save `wifi-creds.txt` to your USB flash drive. Depending on which web browser you use, the text file might download to your computer and open in your default code editor automatically.

If `wifi-creds.txt` doesn't download automatically, open the context (right-click) menu and choose **Save as** to save the text file to your USB flash drive.

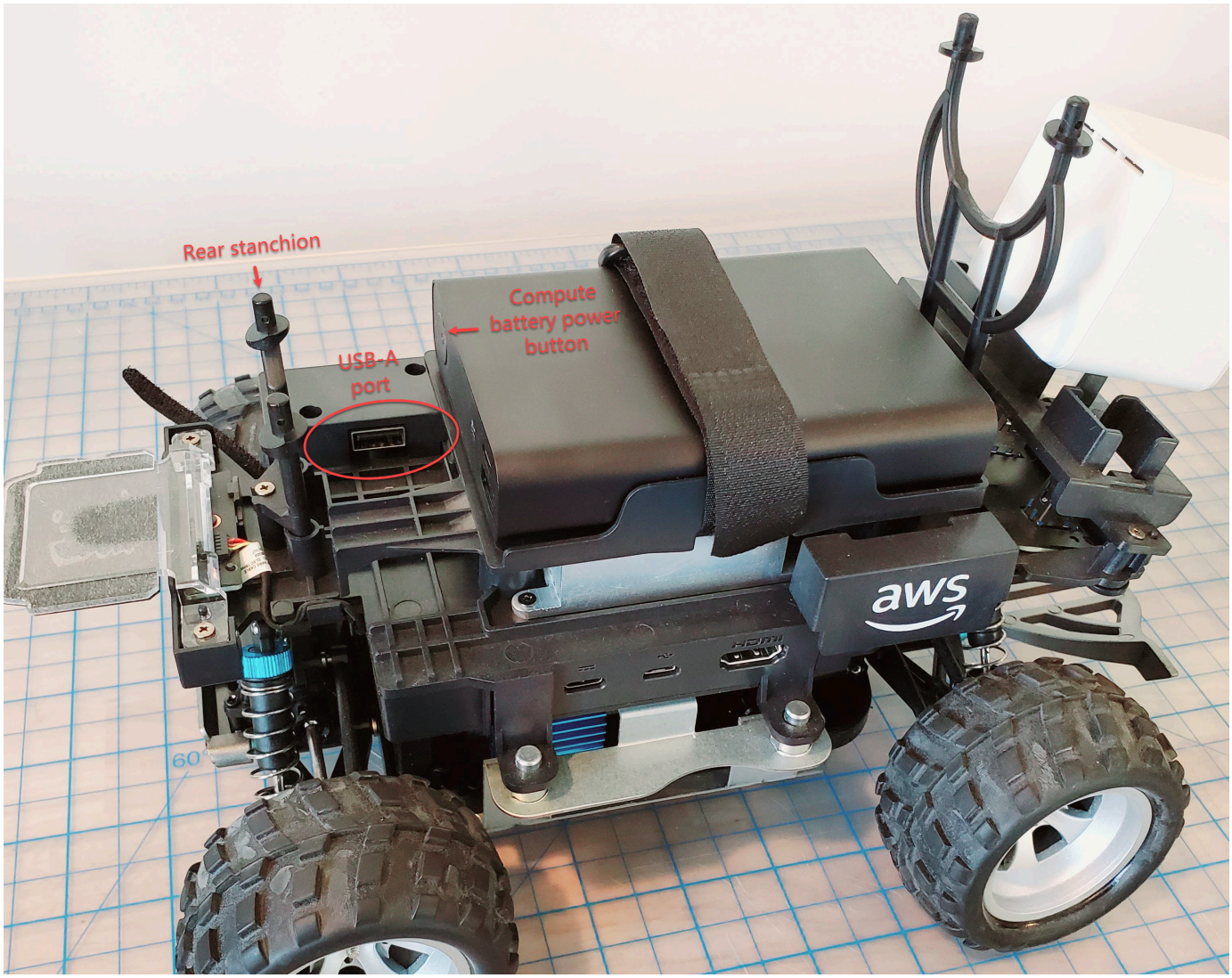
> **Warning**
>
> Do not change the file name.

4. If `wifi-creds.txt` isn't already open, open it in a code editor in plain text mode. Some text editors default to rich text (.rtf) instead of plain text (.txt) when the file type isn't specified, so if you are having trouble editing the file, check your settings. If you are using Windows, you can also try to open the file using the Sublime Text application, which you can download for free, or, if you use a Mac, try the TextEdit application, which is pre-installed on most Mac devices and defaults to plain text.
5. In between the single quotation marks at the bottom of the file, enter the name (SSID) and password of the Wi-Fi network that you want to use. SSID stands for "Service Set Identifier." It is the technical term for the name of your Wi-Fi network.

> **Note**
>
> If the network name (SSID) or password contains a space, such as in *Your-Wi-Fi 100*, enter the name exactly, including the space, inside the quotation marks ('). If there is no space, using quotation marks is optional. For example, the Wi-Fi password, *Passwd1234* doesn't contain a space, so using single quotation marks works but isn't necessary. Both SSID and password are case sensitive.

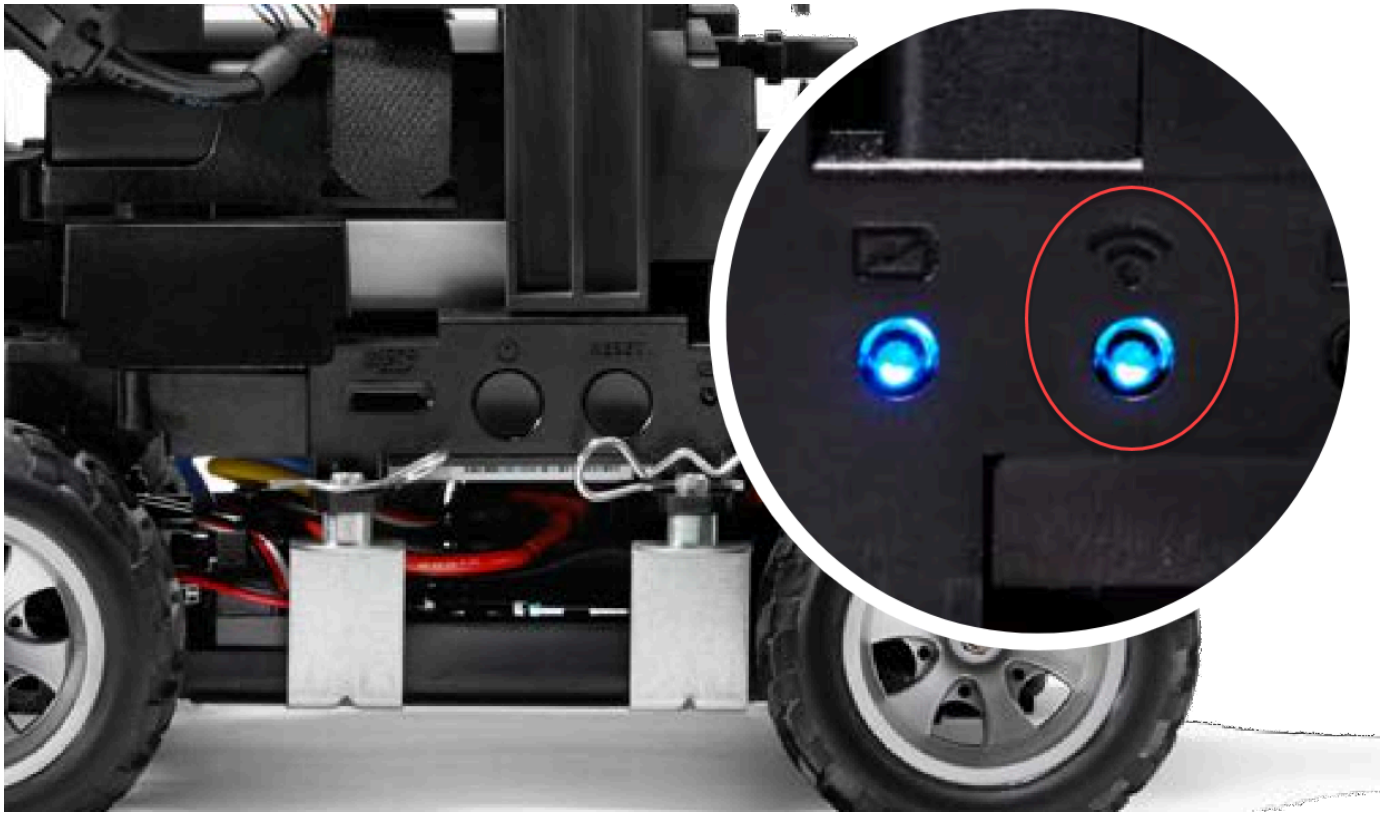
```
# If you have validated the Wi-Fi credentials but the Wi-Fi LED doesn't          #
# turn solid blue, try restarting the vehicle by pressing the reset button.      #
# When the power LED turns blue, plug the USB drive in again.                    #
#                                                                                #
# To finish setting up, follow the instructions on https://docs.aws.amazon.com/  #
# deepracer/latest/developerguide/deepracer-troubleshooting-wifi-connection-first #
# -time.                                                                         #
##################################################################################

# Enter your Wi-Fi name (SSID) and password:
ssid: ''
password: ''
```

6. Save the file on your USB flash drive.
7. Eject the USB drive from your computer and plug it into the USB-A port on the back of the AWS DeepRacer vehicle between the compute battery power button and the rear stanchion.



8. Ensure that the AWS DeepRacer is powered on.
9. Watch the Wi-Fi LED on the vehicle. If it blinks and then changes from white to blue, the vehicle is connected to the Wi-Fi network. Unplug the USB drive and skip to step 11.

> **Note**
>
> If the USB drive was plugged into the vehicle before you attempted to connect the vehicle to a Wi-Fi network, a list of available Wi-Fi networks will be automatically displayed in `wifi-creds.txt` file on your flash drive. Uncomment the one that you want to connect to by removing the pound sign.



10. If the Wi-Fi LED turns red after blinking, unplug the USB drive from the vehicle and plug it back into your computer. Check the Wi-Fi name and password that you entered in the text file for typos, errors in spacing, incorrect sentence casing, or missing or misused single quotation marks. Correct mistakes, and re-save the file, and repeat Steps 7-9.
11. After the vehicle Wi-Fi LED turns blue, unplug the USB drive from the vehicle and plug it into your computer.
12. Open the `wifi-creds.txt` file. Find your vehicle's IP address at the bottom of the text file and copy it.
13. Make sure your computer is in the same network as the vehicle, then paste the IP address into your web browser.

> **Note**
>
> If you are using macOS Catalina, use the Firefox web browser. Chrome is not supported.

14. When prompted with a message that the connection is not private or secure, accept the security warning and proceed to the host page.

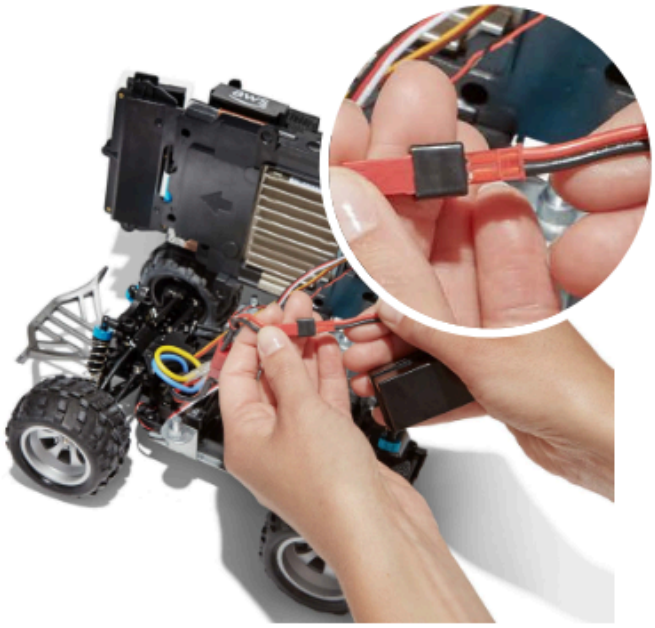
Your AWS DeepRacer is now connected to Wi-Fi.

How to charge the AWS DeepRacer drive module battery

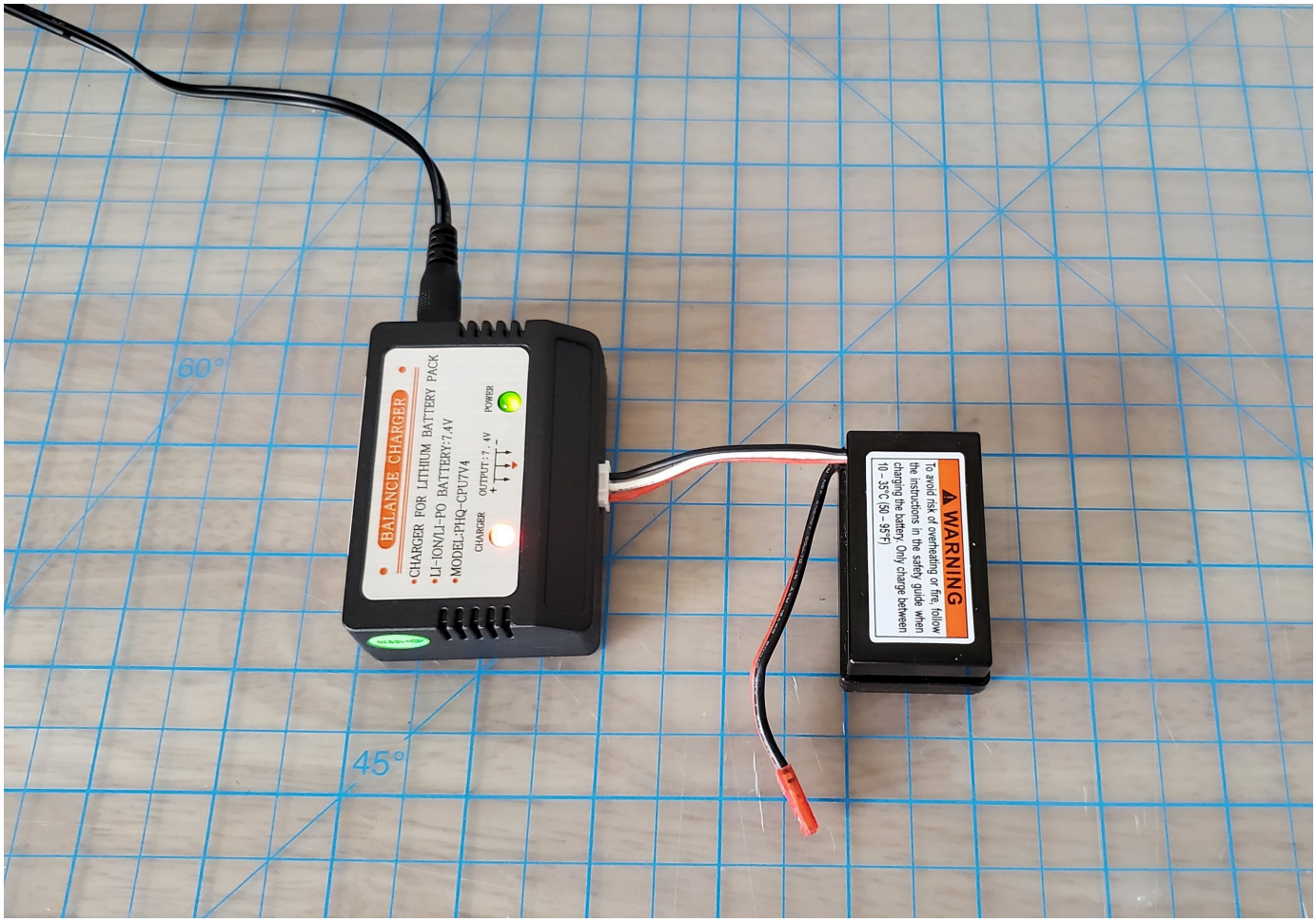
The AWS DeepRacer drive module battery has two sets of cables with two different color JST connectors, white and red. The white 3-pin connector, at the end of the black, red, and white cables, connects the vehicle module battery to its battery charger. The red 2-pin connector, at the end of the black and red cables, connects the battery to the vehicle drive train.

Follow the steps below to charge your AWS DeepRacer drive module battery:

1. To access the drive module battery if it is connected to the vehicle, lift the compute module, being careful not to loosen the wires connecting it to the drive train.



2. Optionally, to remove the drive module battery from the vehicle, disconnect the red 2-pin battery connector from the black and red drive train connector and unstrap the Velcro strap.
3. Attach the battery to the battery charger by connecting the battery's white 3-pin connector to the charger port.



Red light + green light = not fully charged

4. Plug the power cord of the battery charger into a power outlet. When only the green light is illuminated, your battery is fully charged.
5. Disconnect the charged vehicle battery's white 3-pin connector from the charge adapter. If you removed the battery to charge it (optional) make sure to reconnect its red 2-pin connector to the vehicle drive train connector and secure the battery to the vehicle with the Velcro strap.
6. Turn on the vehicle drive train by pushing its switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm a successful charge. If you don't hear two beeps, try unlocking your vehicle battery

Your AWS DeepRacer drive module battery is now ready for use.

How to charge the AWS DeepRacer compute module battery

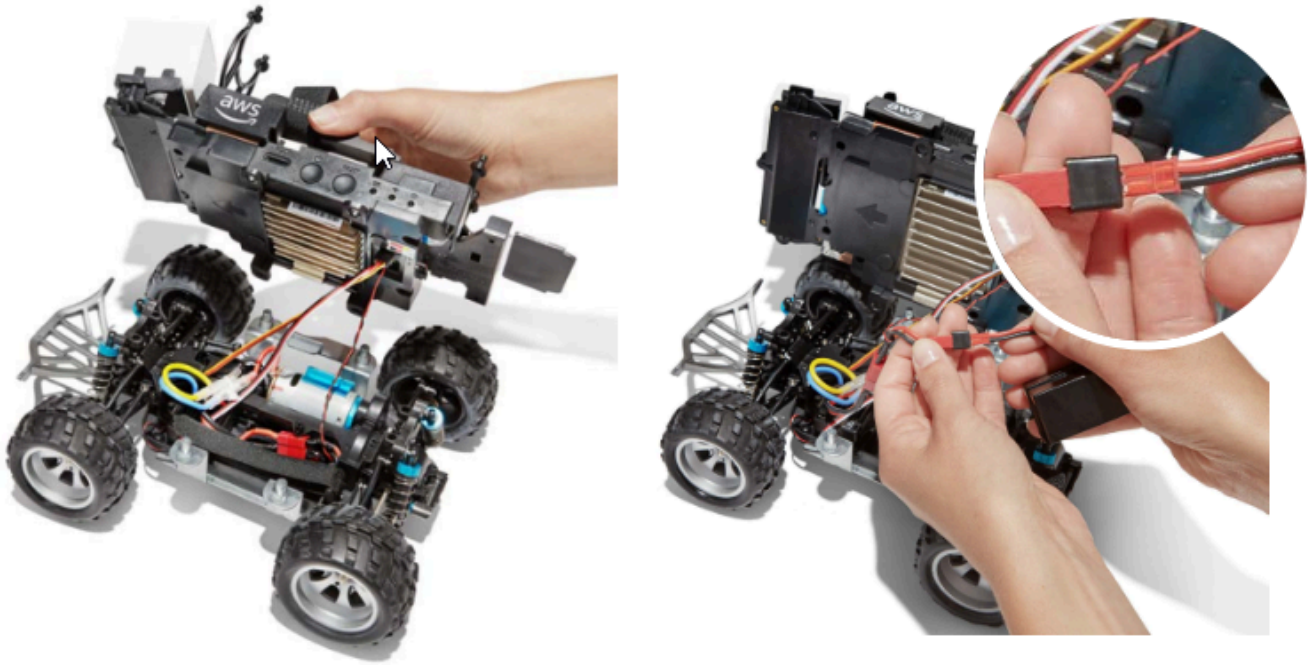
Follow the steps below to charge your AWS DeepRacer compute module battery:

1. Optionally remove the compute module battery from the vehicle.
2. Attach the compute power charger to the compute module battery.
3. Plug the power cord of the compute battery charger into a power outlet.

My battery is charged but my AWS DeepRacer vehicle doesn't move

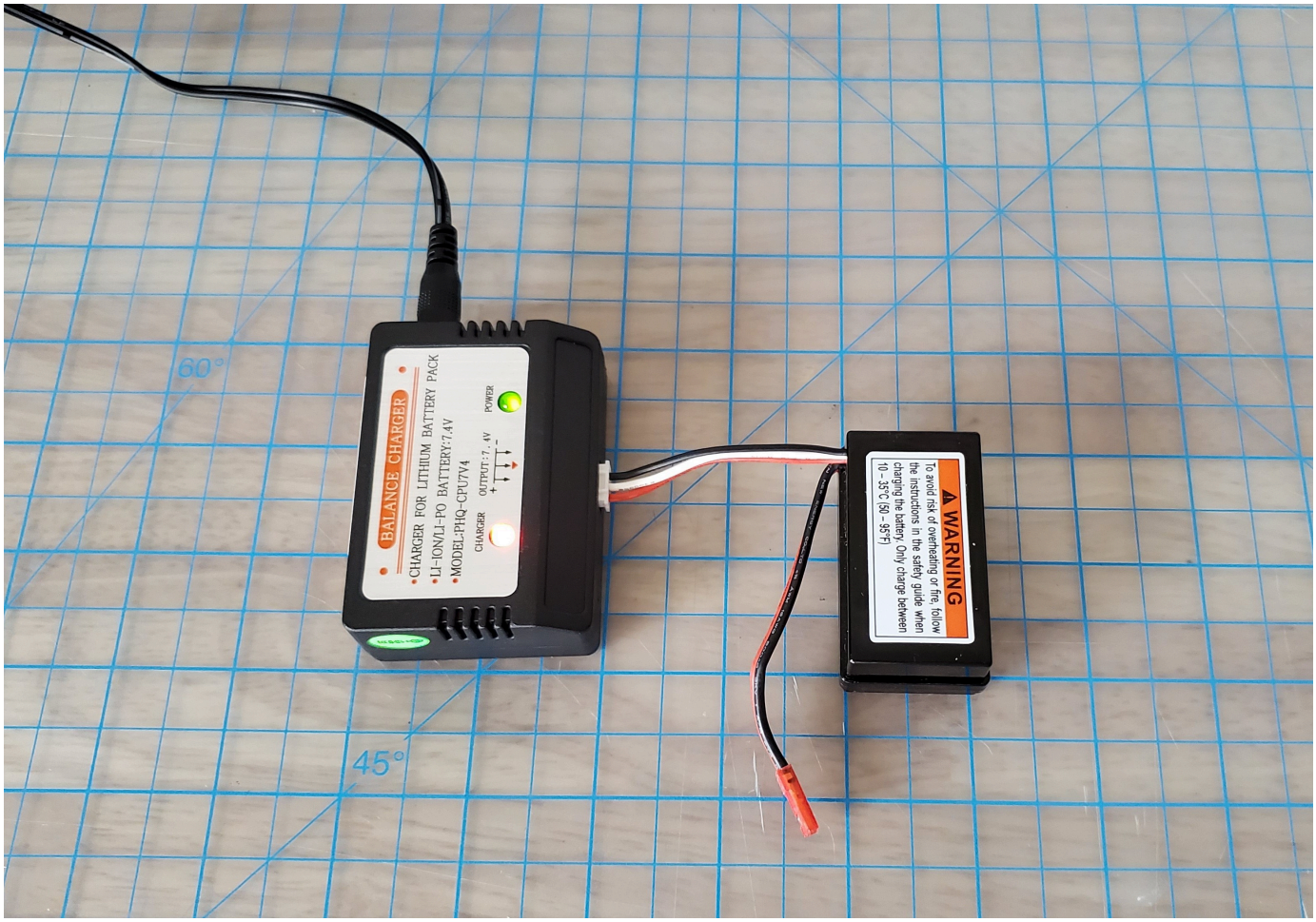
Follow these steps if your AWS DeepRacer console is set up, your compute battery is charged, and your Wi-Fi is connected, but your vehicle still doesn't move:

1. Lift the compute module, being careful not to loosen the wires connecting it to the drive train. Make sure the vehicle battery underneath is correctly connected, red 2-pin connector to black and red drive train connector.



2. Turn on the vehicle drive train by pushing the switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm that the vehicle has charge. If the vehicle powers on successfully, skip to step 4.

3. If you do not hear two beeps when you switch on your vehicle battery, ensure that the battery is fully charged. Plug the vehicle battery's white connector cable into its charge adapter, which can be differentiated from the compute module's adapter by its red and green LED indicator lights. Connect the adapter to its charge cable and plug it into a power outlet. When both red and green lights on the vehicle battery charge adapter are lit, it indicates that the battery still needs charging.



Red light + green light = not fully charged

When only the green light is illuminated, your battery is fully charged and ready to use. Disconnect the car battery's white connector from the charge adapter, and reconnect its red connector to the vehicle. If you removed the battery to charge it (optional) make sure to once again secure it to the drive train with the Velcro strap. Turn on the vehicle drive train by pushing its switch to the "on" position. If you still don't hear two beeps, try unlocking your vehicle battery.

4. Connect your vehicle to Wi-Fi and open the AWS DeepRacer console in your browser. Manually drive your vehicle with the touch joystick to confirm that it can move.

REMINDER: To get the most millage out of your vehicle battery, make sure to switch off the vehicle drive train or disconnect its battery when you are not using your AWS DeepRacer.

If your vehicle still does not move, contact AWSDeepRacer-Help@amazon.com.

Troubleshoot AWS DeepRacer vehicle battery lockout

> **Important**
>
> This battery is only for use with the DeepRacer Car. This battery must be handled properly to avoid risk of fire, explosion, or other safety concerns. Follow all instructions and heed all warnings included in the AWS DeepRacer Device Safety Guide.

AWS DeepRacer Device Terms, Warranties, and Notices

- AWS DeepRacer Device Terms of Use
- One-Year Limited Warranty for AWS DeepRacer Device
- AWS DeepRacer Device Safety Guide

Topics

How to prevent AWS DeepRacer vehicle battery lockout

Learn how to prevent AWS DeepRacer vehicle battery lockout.

To preserve battery health, the AWS DeepRacer vehicle battery goes into lockout state. When this happens, the battery won't power your vehicle even if it's still partially charged. To prevent your car battery from entering lockout state, do the following:

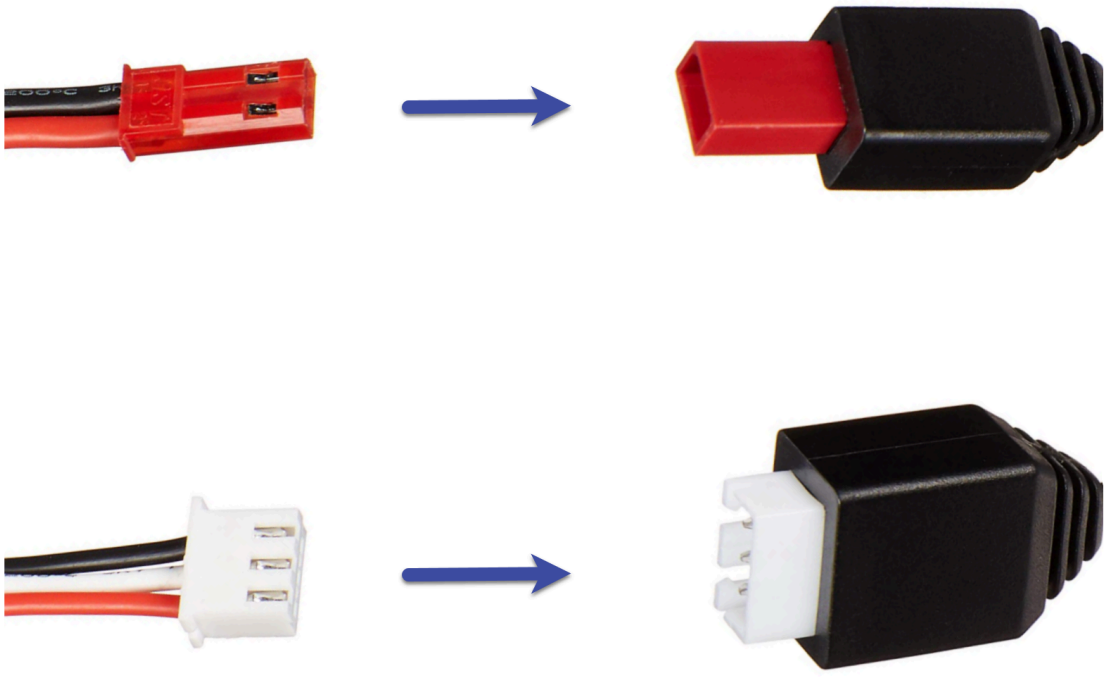
- When you finish using your AWS DeepRacer, turn off the vehicle to preserve the battery's charge.
- When the device console alerts you that your vehicle battery's power level is low, charge it as soon as possible.
- When you think you won't use AWS DeepRacer for a while, disconnect the battery from the vehicle and fully charge it. We suggest you charge your vehicle battery at least once a year to protect it and prevent lockout.

> **Note**
>
> All lithium polymer (LiPo) batteries slowly discharge over time, even when not in use.

How to unlock an AWS DeepRacer vehicle battery after lockout

To unlock your AWS DeepRacer battery after lockout, use the unlock cable:

1. Insert battery connectors into to the matching colored cable connectors, red to red and white to white.



2. Disconnect the battery from the cable.

3. Your AWS DeepRacer vehicle battery is immediately ready for use. Reconnect its red 2-pin connector to the vehicle drive train connector and secure the battery to the vehicle with the Velcro strap.
4. Turn on the vehicle drive train by pushing its switch to the "on" position. Listen for the indicator sound (two short beeps) to confirm that the battery has been successfully unlocked.

How to wrap a Dell battery connector cable when installing a LiDAR sensor

Fitting the Evo shell over a LiDAR sensor connected to an AWS DeepRacer vehicle using the extra long Dell USB-C to angle USB-C connector cable requires a specific cable wrapping technique.

To watch a video of this process, see AWS DeepRacer: Install LiDAR Sensor and wrap Dell compute battery connector cable on YouTube. The video starts with the installation of the LiDAR sensor on the AWS DeepRacer vehicle. The Dell battery wrapping technique begins at 00:01:27 seconds.

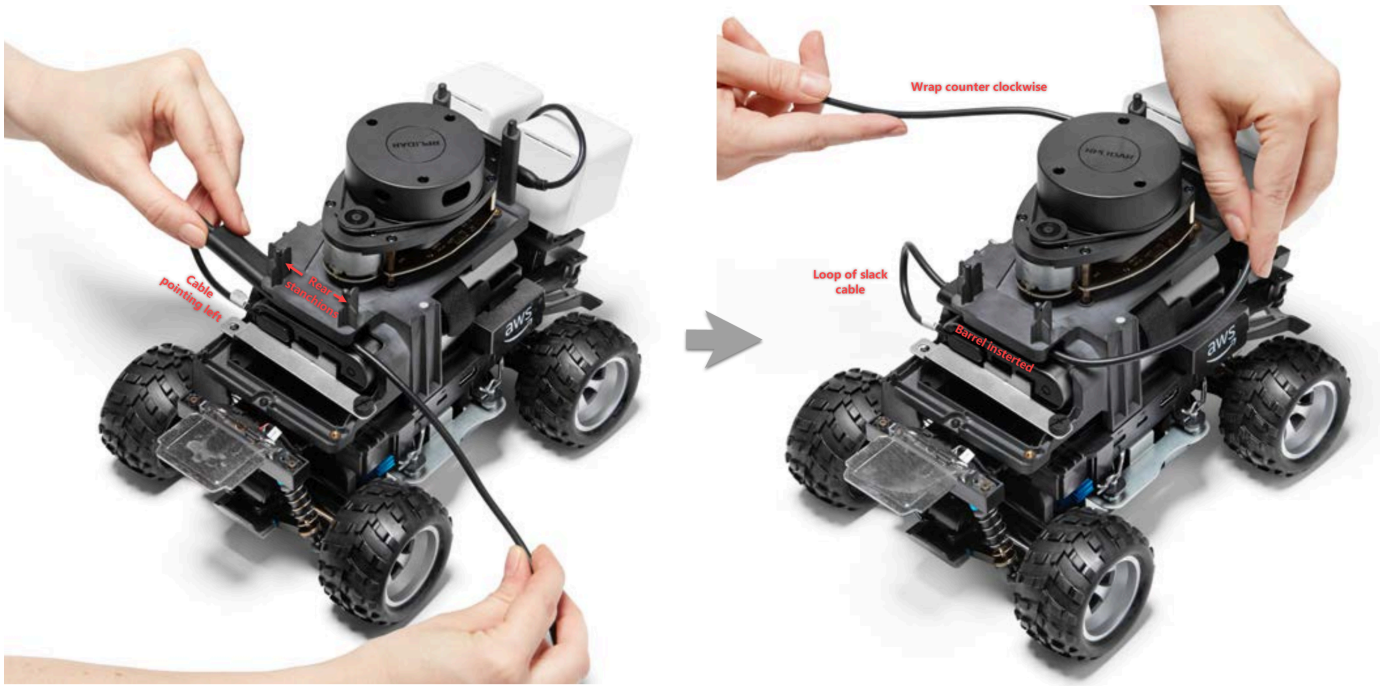
Note

The Dell compute battery connector cable has a barrel, a standard USB-C end, and an angle USB-C end.

To wrap a Dell battery cable around a LiDAR sensor to accommodate the Evo shell

1. Facing the rear of the AWS DeepRacer vehicle, plug the angle end of the compute battery connector cable into the compute battery USB-C port with the connector cable pointing to the left.

2. Turning the vehicle slightly to the left, find the opening to the space in between the LiDAR holder and the compute battery just below the rear stanchions and lace the cable through. Stop pulling the cable through when the barrel is inserted into this space. There should be a loop of slack cable to the left of the USB-C port.



3. Facing the rear of the AWS DeepRacer vehicle, wrap the cable counterclockwise around the base of the LiDAR sensor, using the cable clips to secure the cable to itself to ensure a snug fit.
4. Turn the vehicle slightly to the right and plug the standard USB-C end of the cable into the USB-C port.

5. Place the Evo shell on your AWS DeepRacer vehicle and fasten it with pins to test the fit. When the shell fits correctly, the LiDAR sensor is fully visible through the cutout in the shell, and you have access to the pin holes on the top of the stanchions. Remove the shell and adjust your cable as necessary.

Your LiDAR sensor is connected. You are ready to turn on your vehicle, drive, and experiment.

How to maintain your vehicle's Wi-Fi connection

The following troubleshooting guide provides you tips for maintaining your vehicle's connection.

How to troubleshoot Wi-Fi connection if your vehicle's Wi-Fi LED indicator flashes blue, then turns red for two seconds, and finally off

Check the following to verify you have the valid Wi-Fi connection settings.

- Verify that the USB drive has only one disk partition with only one *wifi-creds.txt* file on it. If multiple *wifi-creds.txt* files are found, all of them will be processed in the order they were found, which may lead to unpredictable behavior.

- Verify the Wi-Fi network's SSID and password are correctly specified in *wifi-creds.txt* file. An example of this file is shown as follows:

```
##################################################################################
#                                  AWS DeepRacer                                 #
# File name: wifi-creds.txt                                                      #
#                                                                                #
# ...                                                                            #
##################################################################################

# Provide your SSID and password below
ssid: ' MyHomeWi-Fi'
password: myWiFiPassword
```

- Verify both the field names of `ssid` and `password` in the *wifi-creds.txt* file are in lower case.
- Verify that each of the field name and value is separated by one colon (:). For example. `ssid : ' MyHomeWi-Fi'`
- Verify that the field value containing a space is enclosed by a pair of single quotes. On Mac, TextEdit or some other text editor shows single quotes as of the '...' form, but not of ‘...’. If the field value does not contain spaces, the value can be without single quotes.

What does it mean when the vehicle's Wi-Fi or power LED indicator flashes blue?

If the USB drive contains *wifi-creds.txt* file, the Wi-Fi LED indicator flashes blue while the vehicle is attempting to connect to the Wi-Fi network specified in the file.

If the USB drive has the `models` directory, the Power LED flashes blue while the vehicle is attempting to load the model files inside the directory.

If the USB drive has both the *wifi-creds.txt* file and the `models` directory, the vehicle will process the two sequentially, starting with an attempt to connect to Wi-Fi and then loading models.

The Wi-Fi LED might also turn red for two seconds if the Wi-Fi connection attempt fails.

How can I connect to the vehicle's device console using its hostname?

When connecting to the vehicle's device console using its hostname, make sure you type: `https://`*`hostname`*`.local` in the browser, where *`hostname`* value (of the `AMSS-`*`1234`* format) is printed on the bottom of the AWS DeepRacer vehicle.)

How to connect to vehicle's device console using its IP address

To connect to the device console using IP address as shown in the *device-status.txt* file (found on the USB drive), make sure the following conditions are met.

- Check your laptop or mobile devices are in the same network as the AWS DeepRacer vehicle.
- Check if you have connected to any VPN, if so, disconnect first.
- Try a different Wi-Fi network. For example, turn on personal hotspot on your phone.

How to get the Mac address of your AWS DeepRacer device

Follow the instructions below to get the Mac address of your AWS DeepRacer device:

1. Make sure that your AWS DeepRacer device is only connected to a Wi-Fi network.
2. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
3. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.
4. Type `deepracer` in the **Username** input field.
5. Type the device SSH password in the **Password** input field.

   If this is your first time to log in to the device, type `deepracer` in the **Password** input field. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.
6. After logged in, open a Terminal window.

   You can use the Search button for the Terminal application.

7. Type the following Ubuntu shell command in the Terminal window:

```
ifconfig | grep HWaddr
```

The command produces an output similar to the following:

```
mlan0    Link encap:Ethernet   HWaddr   01:2a:34:b5:c6:de
```

The hexadecimal numbers are the device's MAC address.

How to recover your AWS DeepRacer device console default password

Recovering your AWS DeepRacer device console default password involves retrieving or resetting the default password. The default password is printed at the bottom of the device, as shown in the following image.

Follow the instructions in the following procedure to recover the password for your AWS DeepRacer device web server using an Ubuntu terminal window.

1. Connect your AWS DeepRacer device to a monitor. You'll need a HDMI-to-HDMI, HDMI-to-DVI or similar cable and insert one end of the cable into the HDMI port on the vehicle's chassis and plug the other end into a supported display port on the monitor.
2. Connect a USB keyboard to your AWS DeepRacer using the USB port on the device's compute module, after the compute module is booted.
3. In the **Username**, enter `deepracer`.
4. In **Password**, enter the device SSH password.

   If this is your first time to log in to the device, enter `deepracer` in **Password**. Reset the password, as required, before moving to the next step. You'll use the new password for future logins. For security reasons, use a complex or strong password phrase for the new password.
5. After you're logged in, open a terminal window.

   You can use the search button to find the terminal window application.
6. To get the default device console password, type the following command in the terminal window:

   ```
   $cat /sys/class/dmi/id/chassis_asset_tag
   ```

   The command outputs the default password as its result.

7. To reset the device console password to the default, run the following Python script in the terminal window:

   ```
   sudo python /opt/aws/deepracer/nginx/reset_default_password.py
   ```

How to manually update your AWS DeepRacer device

Recent changes in the AWS DeepRacer service has made certain legacy devices, such as those distributed at AWS re:Invent 2018, unable to update automatically. Follow the steps below to manually update such a device.

To manually update an AWS DeepRacer device

1. Download to your computer and unzip this manually update a AWS DeepRacer device script.

The default name of the uncompressed file for this script is `deepracer-device-manual-update.sh`. In this topic, we'll assume you use this default script file name.

2. Copy the downloaded and uncompressed the script file (`deepracer-device-manual-update.sh`) from your computer to a USB drive.
3. Connect the device to a monitor using a HDMI-HDMI cable, to a USB keyboard, and to a USB mouse.
4. Power on the device and sign into the OS after the device is booted up.

   You'll need to set the new OS password, if this is your first sign-in to the device.
5. Plug in the USB drive into the device and copy the script file to a folder (for example, ~/Desktop) on the device.
6. From a terminal on the device, type the following command to go to the script file's folder and to add execution permission to the script file:

   ```
   cd ~/Desktop
   chmod +x deepracer-device-manual-update.sh
   ```
7. Type the following shell command to run the script:

   ```
   sudo -H ./deepracer-device-manual-update.sh
   ```
8. When done with updating the device, open a web browser on your computer or a mobile device and navigate to the device IP address, e.g., 192.168.1.11 in a home network or 10.56.101.13 in an office network.

   Make sure that your device is connected to your Wi-Fi network and use a browser in the same network without tunneling through a VPN.
9. On the device console, type the password for the device console to sign in. Wait for the update screen to show up. When prompted for further updates, follow the instructions therein.

How to diagnose and resolve common AWS DeepRacer operational issues

As you explore reinforcement learning with your AWS DeepRacer vehicle, the device may become non functional. The following troubleshooting topics help you diagnose the problems and resolve the issues.

Topics

Why doesn't the video player on the device console show the video stream from my vehicle's camera?

After logging into the AWS DeepRacer device console, you don't see any live video streamed from the camera mounted on the AWS DeepRacer vehicle in the video player in **Device Controls**. The following could cause this issue:

- The camera might have a loose connection to the USB port. Unplug the camera module from the vehicle, replug it into the USB port, power off the device, and then power on the device to restart it.
- The camera might be defective. Use a known working camera from another AWS DeepRacer vehicle, if available, to test whether this is the cause.

Why Doesn't my AWS DeepRacer vehicle move?

You powered on your AWS DeepRacer vehicle, but you can't make it to move. The following could cause this issue:

- The vehicle's power bank is not turned on or the power bank is not connected to the vehicle. Make sure to connect the provided USB-C-to-USB C cable between the USB-C port on the power bank and the USB-C port on the vehicle chassis. Verify that the LED indicators light up, which indicates the charge levels of the power bank. If not, push the power button on the power bank, and then push the power button on the vehicle's chassis to boot up the device. The device is booted up when its tail lights light up.
- If the power bank is on and the vehicle is booted up, but the vehicle does not move in either manual or autonomous driving mode, check if the vehicle's battery under the vehicle chassis is charged and turned on. If not, recharge the vehicle battery and then turn it on after the battery is fully charged.
- The vehicle battery cable connectors are not fully plugged into the device driving module power cable connector. Make sure the cable connectors are tightly coupled.
- The battery cables are defective. Test this battery on another working vehicle, if possible, to test whether this is the cause.
- The power switch of the vehicle battery is not turned on. Turn on the power switch and make sure you hear two beeps followed by a long beep.

Why don't I see the latest device update? How do I get the latest update?

Why is my AWS DeepRacer vehicle's software outdated?

- No automatic update is performed on the device lately. You may need to perform a manual update.
- The vehicle is not connected to the Internet. Make sure the vehicle is connected to a Wi-Fi or Ethernet network with internet access.

Why isn't my AWS DeepRacer vehicle connected to my Wi-Fi network?

When I check the network status on the vehicle's OS, I don't see the AWS DeepRacer vehicle connected to any Wi-Fi network. This could happen because of the following issues:

- No Wi-Fi has been configured for the AWS DeepRacer vehicle. Follow this setup instruction to set up the Wi-Fi network for your vehicle.

- The vehicle is out of the active network signal range. Make sure to operate the vehicle within the chosen Wi-Fi network range.
- The vehicle's pre-configured Wi-Fi network doesn't match the available Wi-Fi network. Follow the setup instruction to set up the Wi-Fi network that does not require active CAPTCHA.

Why does the AWS DeepRacer device console page take a long time to load?

When I tried to open the device console of my AWS DeepRacer vehicle, the device console page appears to take a long time to load.

- Your vehicle is down or off. Make sure the vehicle is powered on when the tail lights are on.
- The IP address of you vehicle has been changed, most likely by your network's DHCP server. To find out the vehicle's new IP address, follow these setup instructions to sign in to the device console with the USB-US cable connection between your computer and the vehicle. View the new IP address in **Settings**. Alternatively, you can examine the list of devices attached to your network to discover the new IP address. If you're not a network administrator, ask the administrator to investigate this for you.

Why does a model fail to perform well when deployed to an AWS DeepRacer vehicle?

After training a model and deploying its artifacts to your AWS DeepRacer vehicle, sometimes the vehicle doesn't perform as expected. What went wrong?

In general, optimizing a trained model for transfer to a physical AWS DeepRacer vehicle is a challenging learning process. It often requires iterations through trial and error. For general guidelines on best practices, see Optimize training AWS DeepRacer models for real environments.

The following are some likely common factors affecting the model performance in your AWS DeepRacer vehicle:

- Your model has not converged in training. Clone the model to continue the training or retrain the model for a longer period of time. Make sure the agent continuously finishes laps in the simulation (that is, 100% process towards the end of the training).

- Your model was over-trained (that is, over-fitted). It fits too well to the training data, but doesn't generalize to unknown situations. Retrain the model with a more flexible or accommodating reward function or increase the granularities of the action space. You should also evaluate a trained model on different tracks to see if the model generalizes well.
- Your AWS DeepRacer vehicle might not have been calibrated properly. To test whether this is true, switch to manual driving and see if the vehicle drives as expected. If it doesn't, calibrate the vehicle.
- You are running the vehicle autonomously on a track that doesn't meet the requirements. For track requirements, see Build your physical track for AWS DeepRacer.
- There are too many objects close to the physical track, making the track significantly different from the simulated environment. Clear the track surroundings to make the physical track as close to the simulated one as possible.
- Reflection from the track surface or a near-by object can create glare to confuse the camera. Adjust lighting and avoid making the track on smooth-surfaced concrete floors or with other shiny materials.

Document history for the AWS DeepRacer Developer Guide

The following table describes the important changes to the documentation since the last release of AWS DeepRacer.

Change	Description	Date
Updates for the 2023 AWS DeepRacer League	Updated multiple topics that reference the AWS DeepRacer League. For more information about the 2023 AWS DeepRacer League season, see the terms and conditions.	March 1, 2023
Train and Evaluate AWS DeepRacer Models Using SageMaker AI Notebooks topic removed temporarily	The topic, Train and Evaluate AWS DeepRacer Models Using SageMaker AI Notebooks, was removed from *Train and evaluate models*. Currently, the procedures for using an AWS SageMaker AI notebook with AWS DeepRacer are being updated.	November 1, 2022
Updates to IAM managed policies for multi-user feature	New managed policies, `AWSDeepRacerAccountAdminAccess` and `AWSDeepRacerDefaultMultiUserAccess`, added so you can sponsor multiple participants under one AWS DeepRacer account using multi-user mode, see the section called "AWSDeepRacerAccountAdminAccess".	October 26, 2021

Updates for multi-user feature	AWS DeepRacer now supports the multi-user feature which allows one AWS account to sponsor multiple participants to race and train. For more information, see *Multi-user mode*.	October 26, 2021
Updates for multi-vehicle racing and obstacle avoidance	AWS DeepRacer now supports new sensor types of stereo camera and LIDAR that allows multi-vehicle racing and obstacle avoidance. For more information, see the section called "Understanding racing types and enabling sensors".	December 2, 2019
Updates for community races	AWS DeepRacer now allows AWS DeepRacer users to organize their own racing events, known as community races, with private leaderboards open to invited users only. For more information, see *Join a race*.	December 2, 2019
Updates for general availability	AWS DeepRacer now features more robust methods to train and evaluate deep learning models. The user interface is updated and explained . More options and precise data is available to build your own physical tracks. Troubleshooting information is available.	April 29, 2019

Initial release AWS DeepRacer Developer Guide	Initial release of the documentation to help the AWS DeepRacer user to learn reinforcement learning and explore its applications for autonomous racing, using the AWS DeepRacer console, the AWS DeepRacer simulator, and a AWS DeepRacer scale model vehicle.	November 28, 2018

AWS Glossary

For the latest AWS terminology, see the AWS glossary in the *AWS Glossary Reference*.