Reinforcement Learning Exercise

Joshua Glaspey

EEL6938: AI for Autonomous Systems

Due Date: March 21, 2025

# Table of Contents

Contents

[Table of Contents 2](#_Toc192961270)

[Introduction 3](#_Toc192961271)

[Code Modifications 3](#_Toc192961272)

[Models 3](#_Toc192961273)

[Hyperparameters 3](#_Toc192961274)

[Episode Lengths 5](#_Toc192961275)

[Reward Structures 5](#_Toc192961276)

[Training Procedure 7](#_Toc192961277)

[Running the Training Script 7](#_Toc192961278)

[Performance Metrics 8](#_Toc192961279)

[Key Performance Metrics 8](#_Toc192961280)

[Visuals and Tabulations 9](#_Toc192961281)

[Results 11](#_Toc192961282)

[Learning Rate Analysis 11](#_Toc192961283)

[Batch Size Analysis 11](#_Toc192961284)

[Buffer Size Analysis 12](#_Toc192961285)

[Tau Analysis 13](#_Toc192961286)

[Gamma Analysis 13](#_Toc192961287)

[Entropy Coefficient Analysis 14](#_Toc192961288)

[Network Architecture Analysis 14](#_Toc192961289)

[Episode Length Analysis 15](#_Toc192961290)

[Reward Function Analysis 15](#_Toc192961291)

[Best Configurable Results 16](#_Toc192961292)

[Discussion 17](#_Toc192961293)

# Introduction

In this assignment, the goal was to train an RL agent to follow a reference speed profile. Given a pre-existing codebase, I modified the model, experimented with different hyperparameters, and adjusted the episode length to study its impact on learning. Additionally, I explored different reward structures and evaluated how they influenced performance. To assess the effectiveness of these changes, I implemented various quantitative metrics and visualized the results through plots comparing predicted and reference speeds. This report walks through the modifications, experimental findings, and key results gained from the process.

# Code Modifications

To improve the reinforcement model’s performance, I added several changes. These focused on addressing each point provided in the assignment’s description, namely model types, hyperparameters, episode length, reward structures, and performance metrics. This section will cover all these changes.

## Models

The provided code utilizes a Soft Actor Critic (SAC) model. To evaluate the impact of different RL algorithms, three additional models were implemented alongside SAC: Proximal Policy Optimization (PPO), Twin Delayed DDPG (TD3), and Deep Deterministic Policy Gradient (DDPG). Each model has its own respective configuration of hyperparameters, but for consistency, no additional parameters outside of those provided in the SAC model were added. The code is shown below for each additional model.

model = PPO(policy="MlpPolicy", env=train\_env, verbose=1, policy\_kwargs=policy\_kwargs, learning\_rate=args.learning\_rate, batch\_size=args.batch\_size, gamma=args.gamma, ent\_coef=ent\_coef\_param, device=device)

model = TD3(policy="MlpPolicy", env=train\_env, verbose=1, policy\_kwargs=policy\_kwargs, learning\_rate=args.learning\_rate, batch\_size=args.batch\_size, buffer\_size=args.buffer\_size, tau=args.tau, gamma=args.gamma, device=device)

model = DDPG(policy="MlpPolicy", env=train\_env, verbose=1, policy\_kwargs=policy\_kwargs, learning\_rate=args.learning\_rate, batch\_size=args.batch\_size, buffer\_size=args.buffer\_size, tau=args.tau, gamma=args.gamma, device=device)

## Hyperparameters

Each of the models has a set of configurable hyperparameters. Below is the list of modifiable hyperparameters for each model.

1. SAC: Learning rate, Batch size, Buffer size, Tau, Gamma, Entropy Coefficient, Net Arch.
2. PPO: Learning rate, Batch size, Gamma, Entropy Coefficient, Net Arch.
3. TD3: Learning rate, Batch size, Buffer size, Tau, Gamma, Net Arch.
4. DDPG: Learning rate, Batch size, Buffer size, Tau, Gamma, Net Arch

Each hyperparameter was tested using a comprehensive range of values. For each training iteration, all hyperparameters were set to their default value except the one being tested to isolate functionality. These ranges are provided below (**bold** indicates the default value). For example, if currently testing the effects of “batch\_size,” all other hyperparameters are set to their default, and batch size iterates across its five values. Additionally, this procedure was repeated for each model (where hyperparameters are applicable).

* Learning Rates: 1e-5, 3e-5, 1e-4**, 3e-4**, 1e-3, 3e-3, 1e-2, 3e-2, 1e-1
* Batch Sizes: 32, 64, 128, **256**, 512
* Buffer Sizes: 50000, 100000, **200000**, 500000, 1000000
* Tau: 0.0001, 0.001, **0.005**, 0.01, 0.02
* Gamma: 0.90, 0.95, **0.99**, 0.999, 1.0
* Entropy Coefficients: **‘auto’**, 0.01, 0.05, 0.1, 0.2
* Net Archs: [64x64], [128x128], **[256x256]**, [512x512]

Each hyperparameter range was carefully selected to explore different trade-offs in model performance:

1. **Learning Rate (1e-5 to 1e-1)** – The values span several orders of magnitude to compare slow convergence with rapid learning. This broad range helps identify the optimal balance between learning speed and stability.
2. **Batch Size (32 to 512)** – The batch sizes are selected in powers of 2 to examine the effect of variance in updates. Smaller batches introduce more variability, which can enhance exploration, while larger batches provide more stable but computationally expensive updates.
3. **Buffer Size (50,000 to 1,000,000)** – This range tests the impact of memory capacity on training. Smaller buffers prioritize recent experiences, while larger buffers retain extensive history at the cost of increased resource consumption.
4. **Tau (0.0001 to 0.02)** – This parameter controls the rate of network updates. Lower values ensure smoother, incremental updates, while higher values enable faster adaptation at the risk of increased variability.
5. **Gamma (0.9 to 1.0)** – The discount factor determines the weighting of future rewards. Lower values emphasize short-term gains, whereas higher values favor long-term rewards, influencing overall policy behavior.
6. **Entropy Coefficient (‘auto’ / 0.0, 0.01 to 0.2)** – This regulates the exploration-exploitation balance. Higher values encourage more exploration, while the ‘auto’ setting dynamically adjusts based on policy confidence. For the SAC model, the default value is ‘auto’ and for the PPO model, the default value is 0.0.
7. **Network Architecture ([64×64] to [512×512])** – The varying sizes of the neural network test the trade-off between computational efficiency and representational power. Smaller architectures train faster, while larger networks capture more complex patterns.

The hyperparameters were added to the skeleton code using the argument parser. Following the same structure as the “—chunk\_size” parameter, each of the provided hyperparameters were added such that they can be manually set per run via the terminal. The code to scan each hyperparameter is provided below – this is in the beginning of the main function. Note that, for the entropy coefficient, “-1.0” is a temporary value used to be masked with a value of “auto” for the variable.

parser.add\_argument("--model", type=int, default=0)

parser.add\_argument("--learning\_rate", type=float, default=3e-4)

parser.add\_argument("--batch\_size", type=int, default=256)

parser.add\_argument("--buffer\_size", type=int, default=200000)

parser.add\_argument("--tau", type=float, default=0.005)

parser.add\_argument("--gamma", type=float, default=0.99)

parser.add\_argument("--ent\_coef", type=float, default=-1.0)

parser.add\_argument("--net\_arch", type=str, default="256,256")

Moreover, the processing of these hyperparameters is provided below. Many of these values can be inserted into the models in their raw state, but a small amount of preprocessing is required. Note that there is a unique log directory created for this configuration of hyperparameters. Also, ignore the “args.reward” variable, as this is another inputted variable representative of the reward function and will be explained in a later section.

# Define the entropy coefficient and network architecture variables

args = parser.parse\_args()

ent\_coef\_param = 'auto' if args.ent\_coef == -1.0 else args.ent\_coef

net\_arch\_param = list(map(int, args.net\_arch.split(",")))

# Create a unique log directory for this configuration

log\_dir = os.path.join(

    args.output\_dir,

    f"model-{args.model}\_chunk-{args.chunk\_size}\_lr-{args.learning\_rate}\_batch-{args.batch\_size}\_buffer-{args.buffer\_size}\_tau-{args.tau}\_gamma-{args.gamma}\_ent-{ent\_coef\_param}\_arch-{'-'.join(map(str, net\_arch\_param))}\_reward-{args.reward}"

)

os.makedirs(log\_dir, exist\_ok=True)

# Assign the network architecture variable in policy\_kwargs

policy\_kwargs = dict(net\_arch=net\_arch\_param, activation\_fn=nn.ReLU)

## Episode Lengths

Originally, the data was split into episodes of 100 steps. This is determined by the chunk size variable. Therefore, a range of episode lengths can be tested by simply passing in a new chunk size through the terminal (“—chunk\_size <number here>”). For this experiment, the effect of the episode length was tested using the SAC model and default hyperparameters. In theory, shorter episodes should provide more frequent updates but overall higher variance, where longer episodes should improve stability but require more training time. The range of chunk sizes is provided below.

* Chunk sizes: 1, 10, 20, 50, **100**, 200, 600

## Reward Structures

The reward function operates by favoring larger values. For this assignment, the default reward was calculated as the negative of the absolute error between the current and reference speeds. Therefore, the smaller the difference, the larger the negative equivalent of the value. Specifically, it follows the equation below, given that is the reference speed is the current speed, and is the reward.

To determine the effect of this reward structure, I introduced four new reward functions. These reward functions are tested against the SAC model and default parameters (including chunk size of 100). This section provides a definition of each function and the justification of its selection.

First, a squared error function can be used to penalize large deviations in speed differences. The default error uses a linear difference, so this reward function analyzes the effect of increasing the magnitude of larger differences. Theoretically, this should force the model to minimize significant speed differences. The reward function is provided below.

Second, an exponential error equation is tested. This harshly penalizes errors of any kind to force the model to be as precise as possible. However, it may make the learning process unstable. The reward function is provided below.

Third, a thresholded absolute error is used. This provides a small tolerance for minor errors to encourage the model to continue finding near-perfect speeds while discouraging larger deviations. The goal is to reduce the exploration of the policy after finding a close enough speed to the reference value. The equation is provided below. In this case, the threshold is selected to be an absolute difference of 0.5.

Fourth, a cubed error equation is tested. This should strongly discourage large errors while magnifying the difference between smaller errors, which can potentially lead to smoother corrections. This is a glorified version of the squared error equation. The reward function is provided below.

The code implementation for each reward function is provided below. Note that the equation is selected depending on an input flag “ERROR\_SELECTION.”

# 1: Absolute error

if ERROR\_SELECTION == 0:

    error = abs(self.current\_speed - self.ref\_speed)

    reward = -error

# 2: Squared error

elif ERROR\_SELECTION == 1:

    error = self.current\_speed - self.ref\_speed

    reward = -error\*\*2

# 3: Exponential error

elif ERROR\_SELECTION == 2:

    error = abs(self.current\_speed - self.ref\_speed)

    reward = -np.exp(error)

# 4. Thresholded absolute error

elif ERROR\_SELECTION == 3:

    error = abs(self.current\_speed - self.ref\_speed)

    reward = 1.0 if error < 0.5 else -error

# 5. Cubed error

elif ERROR\_SELECTION == 4:

    error = abs(self.current\_speed - self.ref\_speed)

    reward = -error\*\*3

## Training Procedure

The RL model for speed following was trained using a batch script () that automates the execution of many different training runs with different hyperparameter configurations. The training process involves defining a reference speed profile, training an RL agent to follow it, and evaluating its performance using one of the provided reward functions and model architectures. For consistency, once one reference speed profile is created, it is stored locally and reused for all subsequent model trainings. This is done through the code below.

# Force-generate a 1200-step sinusoidal + noise speed profile

if not os.path.exists(CSV\_FILE):

    speeds = 10 + 5 \* np.sin(0.02 \* np.arange(DATA\_LEN)) + 2 \* np.random.randn(DATA\_LEN)

    df\_fake = pd.DataFrame({"speed": speeds})

    df\_fake.to\_csv(CSV\_FILE, index=False)

    print(f"Created {CSV\_FILE} with {DATA\_LEN} steps.")

else:

    print(f"{CSV\_FILE} already exists. Skipping creation.")

Each line in executes with a unique combination of parameters that affect model learning. The required terminal flags are listed below.

* **--model [int]:** Maps the integer range to the respective models .
* **--chunk\_size [int]:** Defines the episode length for training.
* **--learning\_rate [float]:** Defines the scale of parameter updating for the model.
* **--batch\_size [int]:** Controls the number of training samples used per update.
* **--buffer\_size [int]:** Defines the memory capacity for experience replay.
* **--tau [float]:** Influences how smoothly the target network updates.
* **--gamma [float]:** Sets the discount factor for future rewards.
* **--ent\_coef [float]:** Manages the balance between exploration and exploitation.
  + A value of -1.0 maps to ‘auto’ – a dynamic variable controlled by model confidence.
* **--net\_arch [str]:** Specifies the neural network size for function approximation.
  + Provided as a string of 2 comma-separated numbers (ex. “64,64”).
* **--reward [int]:** Maps the integer range to the respective reward functions Absolute error, Squared error, Exponential error, Thresholded absolute error, Cubed error.

The batch script systematically varies these hyperparameters to analyze their effects on model performance. These results will be explored in the **Results** section

## Running the Training Script

To execute the training script, perform the following steps.

1. Open a command prompt and navigate to the project directory (containing both and ).
2. Type the following command:

# Performance Metrics

This section outlines the key performance metrics used to evaluate the speed-following RL model. These metrics quantify the model’s ability to track reference speeds accurately and efficiently. By analyzing these measures, we can assess the model performance, compare different approaches, and optimize a final training strategy.

## Key Performance Metrics

**1. Mean Absolute Error (MAE)**

MAE measures the average magnitude of the errors between predicted speed and reference speed, regardless of direction. It is defined by the equation below, where is the current speed, is the reference speed, and is the number of steps taken.

MAE provides an intuitive measure of error, making it useful for evaluating how well the model follows the reference speed. A lower MAE value indicates better performance. It is particularly useful when small deviations are equally significant.

**2. Mean Squared Error (MSE)**

MSE calculates the average squared differences between predicted and reference speeds. This penalizes larger errors more significantly. The formula is shown below.

MSE is particularly useful when larger errors should be more heavily penalized, as applying the square amplifies the impact of significant deviations. Lower MSE values indicate better performance, but due to squaring, it may exaggerate the effect of outliers.

**3. Root Mean Squared Error (RMSE)**

RMSE is the square root of the MSE, providing an error measure in the same unit as the predicted and reference speeds. The formula is shown below.

RMSE is useful for interpreting errors on the same scale as the original data, making it more intuitive for understanding deviations. Lower RMSE values indicate better performance, similar to MSE, but without the exaggerated effect of squaring large errors.

**4. Score**

The score indicates how well the predicted speeds fit the reference speeds in terms of distribution. The formula is shown below, where is the mean of the reference speeds.

measures the proportion of variance explained by the model. A higher value suggests a better fit. Specifically, the score falls into these ranges:

* : Perfect prediction.
* : Model performs as poorly as a constant mean predictor.
* : Model performs worse than a constant mean predictor.

**5. Average Reward**

The average reward measures the RL agent’s performance by averaging the cumulative reward over all time steps. The formula is shown below, where is the reward at time step .

The higher an average reward value is, the better the model performance is at achieving the task efficiently. Specifically, a higher reward value implies that the model successfully follows the reference speed while optimizing its policy.

## Visuals and Tabulations

To better compare the performance of different model configurations, the results will present graphical and tabular representations of these metrics. These files will be in the log directory after final execution of each model configuration. There are three structures that will be used for comparison:

1. **Performance Summary Table**

A performance summary table is provided for every result. It provides the final values for each performance metric defined above and can allow for direct comparison of the final policies of each model. An example table is provided below. These values are located at the bottom of .

----------------------------------------

[TEST] Average reward over 1200-step test: -19.514

[TEST] Mean Absolute Error (MAE): 19.514

[TEST] Mean Squared Error (MSE): 404.408

[TEST] Root Mean Squared Error (RMSE): 20.110

[TEST] R2 Score (R2): -23.595

1. **Predicted Speed vs. Reference Speed Visualization**

A visualization of the reference speed versus predicted speed for all time steps is provided. It helps visualize when and where models deviate the most in terms of speed from the reference profile. The X-axis represents the time steps, and the Y-axis represents speed. The blue dotted line will be the reference speed, and the orange solid line will be the predicted speed. After a successful program run, this will be saved as .

1. **Training Convergence Visualization**

A visualization of the training convergence over time is provided here. It helps visualize areas during the model training where it struggles to determine the reference speed, highlighting problematic areas and training behavior. The X-axis represents the time steps, and the Y-axis represents the average reward. Note that the reward is determined by the selection of the reward function, which may explain larger deviations in rewards. After a successful program run, this will be saved as .

# Results

This section presents the results of training and evaluating the RL models on the speed-following task. Performance is assessed using the key metrics, including MAE, MSE, RMSE, Score, and Average Reward. Additionally, the best configuration for each hyperparameter’s training progress and speed-tracking accuracy will be visualized to provide an insight into model behavior.

The following sections will explore every configurable variable’s contribution to model learning and present a comparison of all results.

## Learning Rate Analysis

The results of using various learning rates are shown below. Table 1 compares the key metrics for each model. Note that, for SAC, using learning rates of 1e-1 or 3e-2 threw exceptions, signifying its sensitivity to high learning rates.

**Table 1:** Testing various learning rates against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

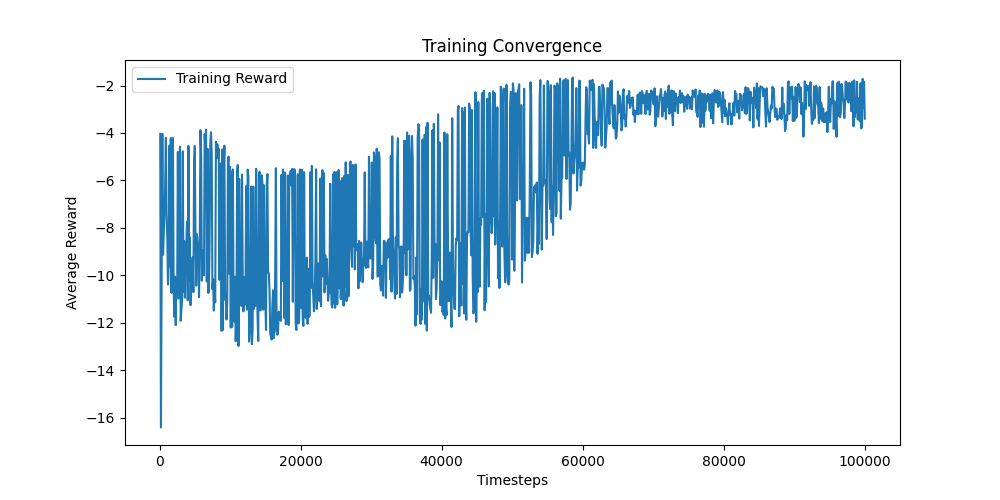
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Learning Rate | | | | | | | | |
|  |  | **1e-5** | **3e-5** | **1e-4** | **3e-4** | **1e-3** | **3e-3** | **1e-2** | **3e-2** | **1e-1** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.721  2.721  11.274  3.358  0.314 | -1.972  1.972  6.127  2.475  0.62 | -2.186  2.186  7.625  2.761  0.536 | -1.958  1.958  6.010  2.452  0.634 | -2.260  2.260  7.927  2.816  0.518 | -2.118  2.118  7.062  2.658  0.570 | -1.904  1.904  5.661  2.379  0.656 | N/A  N/A  N/A  N/A  N/A | N/A  N/A  N/A  N/A  N/A |
| PPO | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.906  1.906  5.698  2.387  0.653 | -1.766  1.766  4.831  2.198  0.706 | -1.756  1.756  4.743  2.178  0.712 | -1.754  1.754  4.743  2.178  0.712 | -1.718  1.718  4.549  2.133  0.723 | -1.783  1.783  4.877  2.208  0.703 | **-1.712**  **1.712**  **4.523**  **2.127**  **0.725** | -1787.1  1787.1  4274623  2067.52  -259973 | -1791.42  1791.42  4290175  2071.27  -260919 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.790  2.790  11.954  3.458  0.273 | -2.571  2.571  10.177  3.190  0.381 | -2.985  2.985  13.265  3.642  0.193 | -2.244  2.244  7.747  2.783  0.529 | -3.450  3.450  17.465  4.179  -0.062 | -1791.42  1791.42  4290175  2071.27  -260919 | -1791.42  1791.42  4290175  2071.27  -260919 | -1791.42  1791.42  4290175  2071.27  -260919 | -1791.42  1791.42  4290175  2071.27  -260919 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -9.011  9.011  104.38  10.216  -5.348 | -2.758  2.758  11.104  3.332  0.325 | -2.274  2.274  8.060  2.839  0.510 | -2.370  2.370  8.780  2.963  0.466 | -2.426  2.426  9.127  3.021  0.445 | -2.040  2.040  6.565  2.562  0.601 | -1791.42  1791.42  4290175  2071.27  -260919 | -10.097  10.097  118.400  10.881  -6.201 | -10.097  10.097  118.400  10.881  -6.201 |

**Analysis**

The learning rate significantly impacts the stability and convergence of reinforcement learning models. A lower learning rate, such as 1e-5 or 3e-5, resulted in slow but stable learning. These rates often led to suboptimal performance as the model failed to update weights effectively, resulting in higher Mean Absolute Error (MAE) and Mean Squared Error (MSE). The slow weight updates prevented the agent from adapting quickly, leading to prolonged training times and suboptimal final policies.

A demonstration of the slow improvement of low learning rates is shown below. Figure 1 shows the convergence of the average reward for a SAC model with a learning rate of 1e-5 across all the timesteps. As visualized, it takes around 60,000 time steps to converge to a smaller reward density, with high fluctuations while the timesteps were fewer than 60,000.

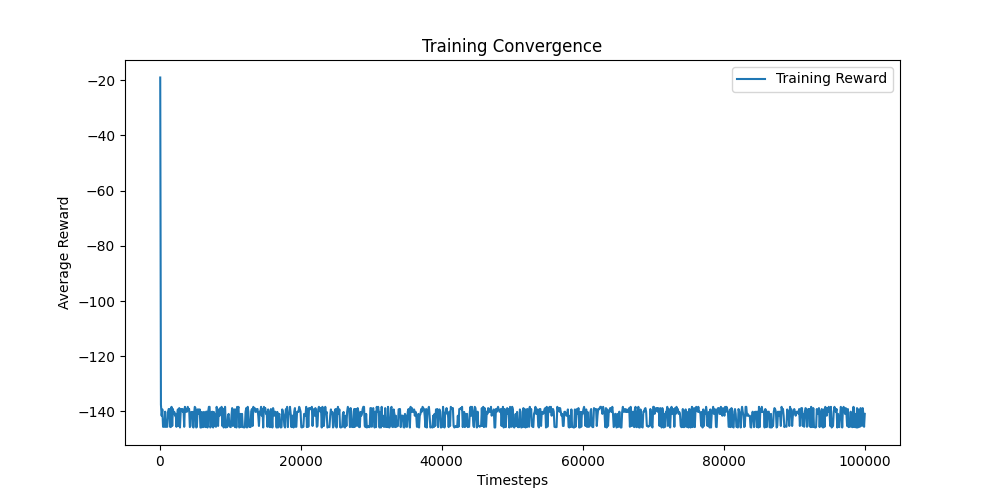
**Figure 1**: The training convergence of a SAC model using a learning rate of 1e-5.



Conversely, extremely high learning rates, such as 1e-1 or 3e-2, led to instability, as shown by erratic training curves and high variance in rewards. Large updates to policy parameters often caused the model to diverge instead of converging, leading to poor generalization. The high variance in error metrics across training runs further supports this instability.

Moreover, a demonstration of the divergent behavior of high learning rates is shown below. Figure 2 shows the divergence of the average reward for a TD3 model with a learning rate of 1e-1. It rapidly grows, on average, to a value of 140, and continues to remain high for the remainder of the training despite the large learning rate.

**Figure 2**: The training convergence of a TD3 model using a learning rate of 0.1.

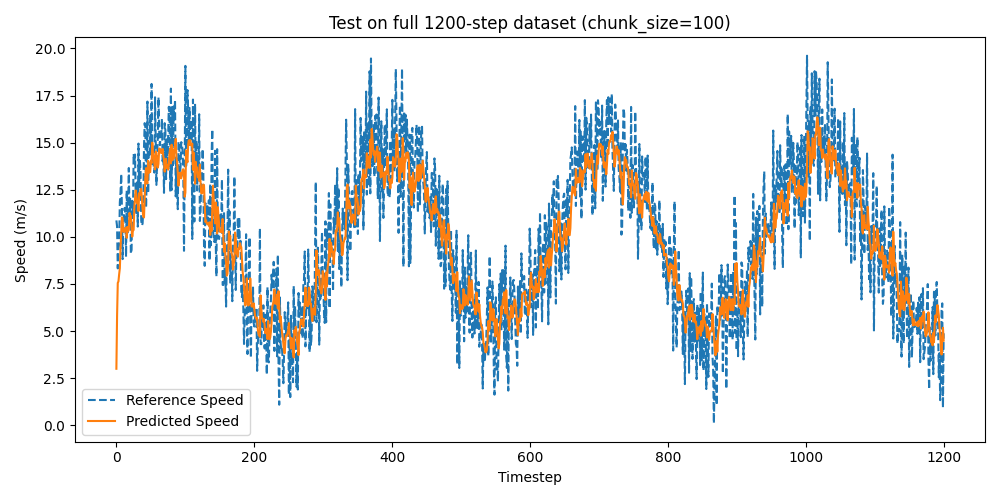


While the learning rate of 1e-2 appeared to yield the lowest MAE and MSE in some cases, it was not chosen as the best due to instability. Higher learning rates can cause the model to overshoot optimal policy updates, leading to oscillations and divergence. This is particularly relevant for SAC and PPO, where policy optimization is sensitive to large updates. Empirically, 3e-4 provided the best trade-off between fast learning and stable convergence, as SAC achieved an MAE of 1.958, while PPO reached 1.754 at this rate.

However, the best combination from this hyperparameter is using the PPO model with a learning rate of 1e-2. Therefore, for comparison, the convergence rates for PPO using a learning rate of 1e-2 and 3e-4 are presented against each other in Figure 3. As visualized, using a learning rate of 3e-4 presents more stability in convergence, which is a better selection for this model given the insignificant benefits that using a learning rate of 1e-2 brings. The higher learning rate shows an early oscillation, which is more unpredictable in its ability to converge to an optimized policy.

<Insert a graph here of the convergence rates of PPO with 3e-4 vs 1e-2>

**Figure 4:** Speed comparison for testing PPO with a learning rate of 3e-4 against the reference speeds.



<Insert explanation here>

## Batch Size Analysis

The results of using various batch sizes are shown below. Table 2 compares the key metrics for each model.

**Table 2:** Testing various batch sizes against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Batch Size | | | | |
|  |  | **32** | **64** | **128** | **256** | **512** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.149  2.149  7.348  2.711  0.553 | -1.882  1.882  5.547  2.355  0.663 | -2.149  2.149  7.272  2.697  0.558 | -1.958  1.958  6.010  2.452  0.634 | -2.301  2.301  8.433  2.904  0.487 |
| PPO | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.754  1.754  4.785  2.187  0.709 | -1.828  1.828  5.205  2.281  0.683 | **-1.751**  **1.751**  4.778  2.186  0.709 | -1.754  1.754  **4.743**  **2.178**  **0.712** | -1.921  1.921  5.673  2.382  0.655 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -3.071  3.071  13.717  3.704  0.166 | -2.232  2.232  7.991  2.827  0.514 | -1.971  1.971  6.100  2.470  0.629 | -2.244  2.244  7.747  2.783  0.529 | -2.135  2.135  7.091  2.663  0.569 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.092  2.092  6.779  2.604  0.588 | -2.045  2.045  6.546  2.558  0.602 | -5.674  5.674  46.216  6.798  -1.811 | -2.370  2.370  8.780  2.963  0.466 | -2.830  2.830  12.368  3.517  0.248 |

**Analysis**

Batch size determines how many samples are used per update, impacting learning stability. Smaller batch sizes (e.g., 32) led to higher variance in updates, causing unstable training with fluctuating rewards. This increased variance makes it difficult for the policy to settle on an optimal trajectory, as updates are based on noisier gradient estimates. While small batch sizes can enhance exploration due to their randomness, they can also lead to suboptimal convergence when stability is required.

Conversely, excessively large batch sizes (e.g., 512) resulted in slower updates, limiting the agent’s ability to adapt quickly to new data. Large batch sizes reduce variance in gradient updates, leading to more stable convergence. However, they also reduce the model’s ability to learn from diverse experiences, potentially leading to overfitting and slower policy updates.

## Buffer Size Analysis

The results of using various buffer sizes are shown below. Table 3 compares the key metrics for each model.

**Table 3:** Testing various buffer sizes against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Buffer Size | | | | |
|  |  | **50,000** | **100,000** | **200,000** | **500,000** | **1,000,000** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.049  2.049  6.601  2.569  0.599 | **-1.879**  **1.879**  **5.550**  **2.356**  **0.662** | -1.958  1.958  6.010  2.452  0.634 | -2.031  2.031  6.513  2.552  0.604 | -2.129  2.129  7.191  2.682  0.563 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.380  2.380  8.784  2.964  0.466 | -2.529  2.529  9.954  3.155  0.395 | -2.244  2.244  7.747  2.783  0.529 | -4.377  4.377  27.945  5.286  -0.700 | -2.126  2.126  7.051  2.655  0.571 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.941  1.941  5.872  2.423  0.643 | -4.468  4.468  29.354  5.418  -0.785 | -2.370  2.370  8.780  2.963  0.466 | -2.581  2.581  10.407  3.226  0.367 | -2.121  2.121  7.061  2.657  0.571 |

**Analysis**

<Fill section here>

## Tau Analysis

The results of using various tau values are shown below. Table 4 compares the key metrics for each model.

**Table 4:** Testing various taus against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Tau | | | | |
|  |  | **0.0001** | **0.001** | **0.005** | **0.01** | **0.02** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.817  1.817  5.131  2.265  0.688 | -2.079  2.079  6.976  2.641  0.576 | -1.958  1.958  6.010  2.452  0.634 | -1.844  1.844  5.269  2.295  0.680 | -1.970  1.970  6.051  2.460  0.632 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.920  1.920  5.782  2.405  0.648 | -2.613  2.613  10.582  3.253  0.356 | -2.244  2.244  7.747  2.783  0.529 | -2.075  2.075  6.741  2.596  0.590 | -2.111  2.111  6.976  2.641  0.576 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | **-1.793**  **1.793**  **5.092**  **2.256**  **0.690** | -2.108  2.108  6.975  2.641  0.576 | -2.370  2.370  8.780  2.963  0.466 | -2.231  2.231  7.742  2.782  0.529 | -2.408  2.408  9.086  3.014  0.447 |

**Analysis**

<Fill section here>

## Gamma Analysis

The results of using various gamma values are shown below. Table 5 compares the key metrics for each model.

**Table 5:** Testing various gammas against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Gamma | | | | |
|  |  | **0.90** | **0.95** | **0.99** | **0.999** | **1.0** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.744  1.744  4.770  2.184  0.710 | -1.927  1.927  5.808  2.410  0.647 | -1.958  1.958  6.010  2.452  0.634 | -2.146  2.146  7.332  2.708  0.554 | -2.464  2.464  9.542  3.089  0.420 |
| PPO | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | **-1.735**  **1.735**  **4.656**  **2.158**  **0.717** | -1.744  1.744  4.734  2.176  0.712 | -1.754  1.754  4.743  2.178  0.712 | -1.781  1.781  4.909  2.216  0.701 | -1.773  1.773  4.830  2.198  0.706 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.260  2.260  7.908  2.812  0.519 | -3.102  3.102  14.395  3.794  0.124 | -2.244  2.244  7.747  2.783  0.529 | -2.480  2.480  9.359  3.059  0.431 | -2.374  2.374  9.002  3.000  0.453 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.871  1.871  5.484  2.342  0.666 | -1.829  1.829  5.201  2.281  0.684 | -2.370  2.370  8.780  2.963  0.466 | -4.492  4.492  28.572  5.345  -0.738 | -2.900  2.900  12.826  3.581  0.220 |

**Analysis**

<Fill section here>

## Entropy Coefficient Analysis

The results of using various entropy coefficients are shown below. Table 6 compares the key metrics for each model. Note that SAC uses a default entropy coefficient of ‘auto’ and PPO uses a default entropy coefficient of 0.0.

**Table 6:** Testing various entropy coefficients against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Entropy Coefficient | | | | |
|  |  | **auto / 0.0** | **0.01** | **0.05** | **0.1** | **0.2** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.958  1.958  6.010  2.452  0.634 | -3.616  3.616  19.749  4.444  -0.201 | -2.227  2.227  7.823  2.797  0.524 | -2.111  2.111  6.901  2.627  0.580 | -2.495  2.495  9.982  3.159  0.393 |
| PPO | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | **-1.754**  **1.754**  **4.743**  **2.178**  **0.712** | -1.812  1.812  5.050  2.247  0.693 | -1.784  1.784  4.930  2.220  0.700 | -1.957  1.957  5.936  2.436  0.639 | -1.950  1.950  5.922  2.433  0.640 |

**Analysis**

<Fill section here>

## Network Architecture Analysis

The results of using various network architectures are shown below. Table 7 compares the key metrics for each model.

**Table 7:** Testing various network architectures against the default hyperparameters for each model. The best metric result for each hyperparameter value across all models is highlighted in red, while the best overall result for each model across all hyperparameter values is underlined. The best overall model-hyperparameter combination for each metric is **bold.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Metric | Network Architecture | | | |
|  |  | **64 x 64** | **128 x 128** | **256 x 256** | **512 x 512** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.086  2.086  6.809  2.609  0.586 | -1.984  1.984  6.145  2.479  0.626 | -1.958  1.958  6.010  2.452  0.634 | -1.898  1.898  5.615  2.370  0.659 |
| PPO | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | **-1.749**  **1.749**  **4.731**  **2.175**  **0.712** | -1.795  1.795  4.980  2.232  0.697 | -1.754  1.754  4.743  2.178  **0.712** | -1.800  1.800  5.023  2.241  0.695 |
| TD3 | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -2.507  2.507  9.816  3.133  0.403 | -1.904  1.904  5.622  2.371  0.658 | -2.244  2.244  7.747  2.783  0.529 | -2.315  2.315  8.206  2.865  0.501 |
| DDPG | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.847  1.847  5.360  2.315  0.674 | -2.761  2.761  12.169  3.488  0.260 | -2.370  2.370  8.780  2.963  0.466 | -2.001  2.001  6.284  2.507  0.618 |

**Analysis**

<Fill section here>

## Episode Length Analysis

The results of using various episode lengths are shown below. Table 8 compares the key metrics for each model. 1, 10, 20, 50, 100, 200, 600

**Table 8:** Testing various episode lengths against the default hyperparameters for each model. The best result using all hyperparameter values for the SAC model is **underlined and bold.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Episode Length | | | | | | |
|  |  | **1** | **10** | **20** | **50** | **100** | **200** | **600** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -19.514  19.514  404.408  20.110  -23.595 | -3.703  3.703  18.623  4.315  -0.133 | -2.325  2.325  8.302  2.881  0.495 | -2.429  2.429  9.111  3.018  0.446 | -1.958  1.958  6.010  2.452  0.634 | **-1.832**  **1.832**  **5.316**  **2.316**  **0.674** | -2.018  2.018  6.632  2.575  0.597 |

**Analysis**

<Fill section here>

## Reward Function Analysis

The results of using various reward functions are shown below. Table 9 compares the key metrics for each model.

**Table 9:** Testing various reward functions against the default hyperparameters for each model. The best result using all hyperparameter values for the SAC model is **underlined and bold.** However, the average reward scores are not considered due to calculation differences.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Model | Metric | Reward Function | | | | |
|  |  | **Absolute Error** | **Squared Error** | **Exponential Error** | **Thresholded Error** | **Cubed Error** |
| SAC | **Avg. Reward**  **MAE**  **MSE**  **RMSE** | -1.958  1.958  **6.010**  **2.452**  **0.634** | -2.032  2.032  6.719  2.592  0.591 | -2.813  2.813  7.497  2.738  0.544 | -2.072  2.072  6.641  2.577  0.596 | -1.958  1.958  **6.010**  **2.452**  **0.634** |

**Analysis**

<Fill section here>

## Best Configurable Results

Using the best hyperparameters:

* Model = PPO
* Learning Rate = 3e-4
* Batch Size = 128
* Buffer Size = N/A
* Tau = N/A
* Gamma = 0.9
* Net Arch = 64 x 64
* Chunk Size = 200
* Reward = 4

<Table for these results>

# Discussion

<Fill section here>