# REINFORCEMENT LEARNING EXERCISE 2

By Alexander Green

EEL6938: Artificial Intelligence for Autonomous Systems

## Table of Contents

# Setup

This assignment setup has two separate python files to run. The first file is my modified version of the provided **RL_assignmnet.py**. I modified this file to perform the experiment with the hyperparameters that I pass through the command terminal. The second file is **Main.py**. This file was created to intelligently run all the required experiments I need for this assignment. **Main.py** handles commands with variations in hyperparameters, data storage, and basic data comparison.

## RL_Assignment.py

In this section, we outline the updates to **RL_assignment.py** from its initial state, detailing specific modifications with corresponding line numbers. We then provide an overview of its functionality, explaining how it processes data, trains reinforcement learning models, and evaluates performance using custom environments, multiple RL algorithms, and detailed performance metrics.

### Modifications to Original Code

1. **Command-Line Arguments for Hyperparameters**
   The script has been enhanced with comprehensive command-line argument support, allowing users to flexibly customize key reinforcement learning parameters without modifying the code. The **--output_dir** argument specifies the destination for logs and trained models, while **--chunk_size** controls training episode length by determining how the dataset is segmented. Algorithm selection is handled through the **--model** parameter, which supports multiple reinforcement learning approaches including **SAC**, **PPO**, **TD3**, and **DDPG**.

   Core learning parameters can be fine-tuned through arguments like **--learning_rate**, which regulates optimization speed, **--batch_size** for controlling data processing scale, and **--buffer_size** for memory capacity. Additional nuanced control is available through --tau for target network update rates, --gamma for future reward discounting, and **--ent_coef** for entropy regularization configuration. The neural network's structure is customizable via **--net_arch**. Training duration is controlled through the **--total_timesteps** argument. Each parameter has a sensible default value and includes helpful documentation to guide users.

2. **Multiple RL Algorithms**
   In the first reinforcement learning assignment, I expanded the original code to

support not only the **SAC** algorithm but also **PPO**, **TD3**, and **DDPG**. These modifications remain intact and continue to serve as the foundation for this second exercise. The implementation defines a set of common parameters shared across all four models, ensuring consistency in policy selection, environment setup, and key hyperparameters. This standardization simplifies model initialization and eliminates redundant code across different algorithms.

The implementation maintains individual parameter configurations unique to each reinforcement learning algorithm. These include specific adjustments for batch size, buffer size, entropy coefficient, and other algorithm-specific settings that ensure each model is configured appropriately for training. All these parameters continue to be imported from command-line arguments passed to **RL_assignment.py**, preserving the flexibility established in the first exercise while extending its application to the current adaptive cruise control scenario.

3. **Reward Function**

The reward function for this adaptive cruise control (ACC) exercise was carefully designed to balance multiple competing objectives of real-world autonomous driving. Unlike the previous exercise which explored multiple reward formulations, this implementation uses a single, comprehensive reward function that addresses several key aspects of ACC behavior simultaneously.

The reward function is mathematically expressed as:

$$reward = -speed\_error - 2.0 * distance\_error - 0.1 * abs(jerk) - 0.05 * abs(accel)$$

This formulation integrates four critical components:

First, the -speed_error term penalizes deviations from the reference speed, encouraging the vehicle to maintain the desired velocity when possible. Second, the -2.0 * distance_error term, with its higher coefficient, prioritizes safe distance maintenance above other concerns. This distance error is calculated using a tiered approach - applying a significant penalty (5.0 multiplier) when following too closely (less than 5m), a moderate penalty (1.0 multiplier) when exceeding the maximum desired distance (30m), and no penalty when within the safe range.

The third component, -0.1 * abs(jerk), promotes driving comfort by discouraging rapid changes in acceleration that passengers would experience as unpleasant jerking motions. Finally, the -0.05 * abs(accel) term encourages energy efficiency by slightly penalizing unnecessary acceleration or braking.

This multifaceted reward function guides the agent toward behavior that balances safety, comfort, and efficiency - maintaining appropriate following distances while achieving smooth speed transitions that closely track the lead vehicle's movement patterns.

## 4. Model Performance Metrics

The following metrics quantify the reinforcement learning model's performance:

**Speed Tracking Metrics**

- **Mean Absolute Error (MAE_Speed):** Average absolute difference between predicted and reference speeds
- **Mean Squared Error (MSE_Speed):** Average of squared differences, emphasizing larger deviations
- **Root Mean Squared Error (RMSE_Speed):** Square root of MSE, providing error in original units

**Distance Maintenance Metrics**

- **MAE_Distance:** Average deviation from the safe distance range (5-30m)
- **Distance_In_Range_Percent:** Percentage of time spent within the safe distance range

**Ride Comfort Metrics**

- **Mean_Absolute_Jerk:** Average magnitude of acceleration changes per second
- **Jerk_Variance:** Consistency of acceleration changes, with lower values indicating smoother rides

**Comparative Performance Metrics**

- **Mean_Speed_Diff:** Average speed difference between ego and lead vehicles, measuring how well the controller matches lead vehicle behavior

These metrics balance the competing objectives of speed tracking accuracy, safety through proper distance maintenance, and passenger comfort through smooth acceleration profiles.

5. **Visualization and Data Analysis**
   The **speed tracking plot** compares three key speed metrics: the reference speed profile, the ego vehicle's speed achieved by the agent, and the lead vehicle's speed. This visualization demonstrates how well the ACC system maintains desired speeds while adapting to the lead vehicle.

   The **distance plot** shows the following distance between the ego vehicle and lead vehicle over time, with highlighted safe distance range (5-30m). This critical ACC metric visualizes collision avoidance and proper spacing maintenance.

   The **jerk plot** analyzes ride comfort by tracking the rate of change in acceleration (jerk), with lower values indicating smoother driving. The plot includes the mean absolute jerk as a reference line for comfort assessment.

   The **speed difference plot** illustrates the difference between ego vehicle and lead vehicle speeds over time, showing how effectively the ACC system maintains appropriate relative velocities for safe following.

   Each visualization is automatically saved to the specified output directory with systematic naming (e.g., "1_acc_speed_tracking_plot.png"). These plots, along with comprehensive metrics like MAE, RMSE, distance-in-range percentage, and jerk statistics, provide a complete performance assessment of the ACC reinforcement learning agent.

## General Functionality of Final Code

The script begins by generating a 1200-step speed dataset, adding noise to a sinusoidal speed profile, and saving it as a CSV file. This dataset is split into chunks for episodic training, ensuring consistent episode lengths while handling any remaining data.

Two custom Gym environments manage training and testing. **TrainEnv** selects random data chunks per episode, where the agent adjusts acceleration to minimize speed error under various reward functions. **TestEnv** evaluates the trained model on the full dataset in a single run, assessing generalization.

For training, the script supports **SAC**, **PPO**, **TD3**, and **DDPG**, with customizable hyperparameters set via command-line arguments. The model is trained using stable-baselines3, logging progress and dynamically selecting GPU or CPU.

During testing, the trained model runs through all 1200-steps, tracking speed accuracy and calculating key performance metrics like average reward and speed error.

## Main.py

The **Main.py** program functions as an experiment runner, automating the testing of various hyperparameter configurations for **RL_assignment.py**. It systematically varies parameters such as **learning rate**, **batch size**, **chunk size**, **reward functions**, **RL algorithms**, **network architectures**, **discount factors**, and **entropy coefficients**. To optimize efficiency, the script first checks whether results for a specific configuration already exist. If the output directory is found, that experiment is skipped, preventing redundant computations and significantly reducing runtime. This ensures that previously calculated results remain intact while allowing new experiments to be conducted without repeating unnecessary steps.

Each hyperparameter category is executed independently, meaning that specifying ***--experiment learning_rate*** runs only learning rate variations, while ***--experiment batch_size*** runs only batch size experiments, and so on. This modular design allows for recalculating results for specific hyperparameters without rerunning all previous experiments. Users can define the total training timesteps and choose to test a single parameter category or run all experiments sequentially.

If all experiments are executed using ***--experiment all***, the script refactors the final results, compiling updated comparative metrics and generating new **final_results** figures. These figures highlight the best-performing hyperparameters and provide a visual comparison of different configurations, ensuring that the latest and most accurate results are always reflected in the final summary.

## Metrics

To evaluate the performance of the reinforcement learning model for Adaptive Cruise Control (ACC), several metrics are used to quantify different aspects of the system's behavior:

1. **Speed Tracking Metrics**
    - **Mean Absolute Error (MAE):** Measures the average absolute difference between the ego vehicle's speed and the reference speed, providing a direct measure of speed tracking accuracy.
    - **Mean Squared Error (MSE):** Weighs larger deviations more heavily, making the model more sensitive to significant speed tracking errors.
    - **Root Mean Squared Error (RMSE):** Provides speed error in the original units while still penalizing larger errors more heavily than MAE.
2. **Distance Maintenance Metrics**
    - **Distance_MAE:** Measures the average absolute deviation from the desired following distance.
    - **Safety_Violations_Percent:** Percentage of time steps where the following distance falls outside the safe range of 5-30 meters.
    - **Distance_In_Range_Percent:** Percentage of time steps where the following distance remains within the safe range.
3. **Ride Comfort Metrics**
    - **Jerk_Mean:** Average rate of change of acceleration, measuring ride smoothness.
    - **Jerk_Variance:** Measures consistency of acceleration changes; lower values indicate smoother driving.
4. **Comparative Performance Metrics**
    - **Speed_Difference_MAE:** Average absolute difference between ego vehicle and lead vehicle speeds.
    - **Reward:** Overall performance metric combining speed tracking, distance maintenance, and comfort factors.

These metrics provide a comprehensive evaluation of the ACC system, balancing the sometimes-competing objectives of speed tracking accuracy, safe distance maintenance, and passenger comfort.

## Visualizations

There are several visualizations generated by our software implementation. This section explains how to interpret each visualization within the results section.

1. **Comparative Metrics Bar Charts**
   These charts display performance metrics for different hyperparameter values. Each bar represents a different configuration, with the height indicating the metric value. Lower values generally indicate better performance for error metrics. The actual value is displayed above each bar for precise comparison.
2. **Performance Profile Plots**
   The top row shows speed profiles for each configuration:
   - The dashed line represents the reference speed
   - The dash-dot line shows the lead vehicle speed
   - The solid line displays the ego vehicle speed under the tested configuration

   The bottom row shows the following distance over time:
   - The solid line represents the actual following distance
   - Red and green horizontal lines mark the minimum (5m) and maximum (30m) safe distance boundaries
3. **Learning Curve Plots**
   These four-panel plots track the model's learning progress:
   - Top-left: Average reward over training timesteps
   - Top-right: Average speed error over training timesteps
   - Bottom-left: Average distance error over training timesteps
   - Bottom-right: Average jerk over training timesteps

   Each line represents a different configuration of the parameter being tested, allowing for comparison of learning stability and convergence rates.

4. **Combined Metrics Comparison**
   This visualization appears in the final results section, showing the best configuration for each hyperparameter type across multiple metrics. Lower values indicate better performance for all metrics except Distance_In_Range_Percent, where higher values are better.

By examining these visualizations together, we can comprehensively assess how each hyperparameter affects the ACC system's ability to maintain proper speed and distance while providing a comfortable ride.

# Results

For this exercise, I modified the **Algorithm Type**, **Batch size**, **Chunk Size**, **Entropy Coefficient**, **Gamma** value, **Learning Rate**, and **Network Architecture** dimensions. Each individual section will have the 4 figures shown in the visualization section. I will make an analysis of the 3 figures to highlight which individual generated the best performance. These findings for all categories will be summarized in the **Conclusion** section.

## Algorithm Type

**RL_assignment.py** has the capability to use any one of four learning algorithms: **SAC**, **PPO**, **TD3**, and **DDPG**. Each of these algorithms employs a different approach to learning optimal policies, balancing exploration and exploitation in distinct ways. **SAC (Soft Actor-Critic)** emphasizes entropy maximization for more stable learning, **PPO (Proximal Policy Optimization)** improves training efficiency through constrained policy updates, **TD3 (Twin Delayed Deep Deterministic Policy Gradient)** reduces overestimation bias in continuous control tasks, and **DDPG (Deep Deterministic Policy Gradient)** leverages actor-critic methods for deterministic policy learning. The resulting performance metrics for each algorithm are shown below.

| Alg. Type | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Abs. Jerk | Jerk Var. | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| SAC | 2.02 | 6.43 | 2.54 | 0.20 | 1.96 | 0.46 | 0.33 | 91.75 |
| PPO | 8.51 | 95.30 | 9.76 | 436.47 | 8.50 | 0.00 | 0.00 | 1.33 |
| TD3 | 2.46 | 9.49 | 3.08 | 0.54 | 2.44 | 1.01 | 2.04 | 79.08 |
| DDPG | 2.06 | 6.92 | 2.63 | 0.42 | 2.05 | 1.73 | 4.43 | 85.83 |

***Table 1: Algorithm Type***

Based on the metrics data, the **SAC algorithm** significantly outperforms all other algorithms across most key performance indicators. With an MAE_Speed of 2.02, it achieves competitive speed tracking accuracy compared to DDPG while maintaining the best distance control with an MAE_Distance of just 0.20. The Distance_In_Range_Percent of 91.75% for SAC indicates that it maintains safe following distances for the vast majority of the simulation, outperforming all other algorithms. While SAC's Mean_Absolute_Jerk value of 0.46 is higher than PPO's near-zero value, this represents a reasonable trade-off for the dramatically improved tracking accuracy and safety performance. PPO shows extremely poor performance in distance maintenance and overall tracking accuracy, making it unsuitable for this application despite its smooth acceleration profiles.

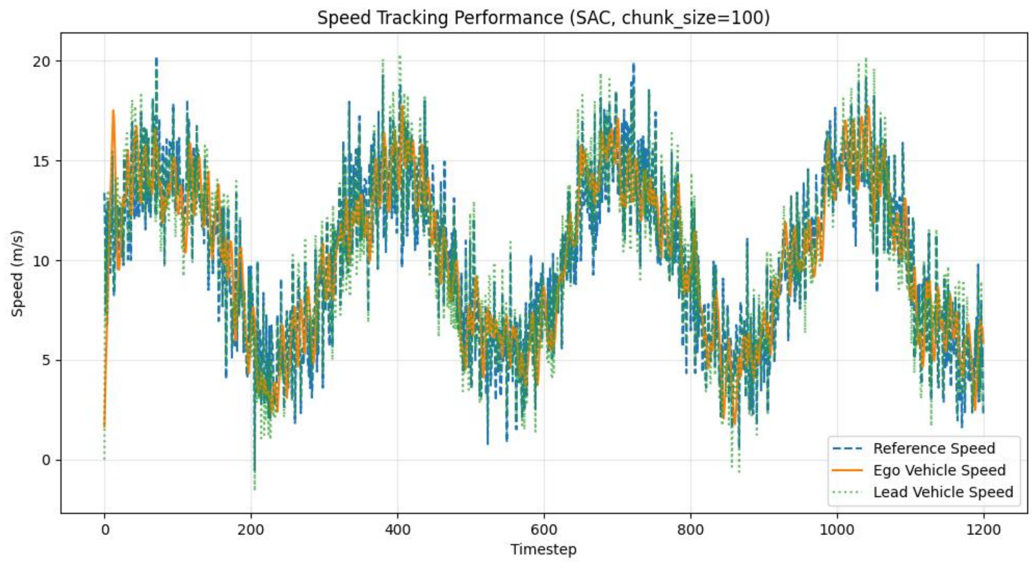


***Figure 1: SAC Speed Tracking Performance***

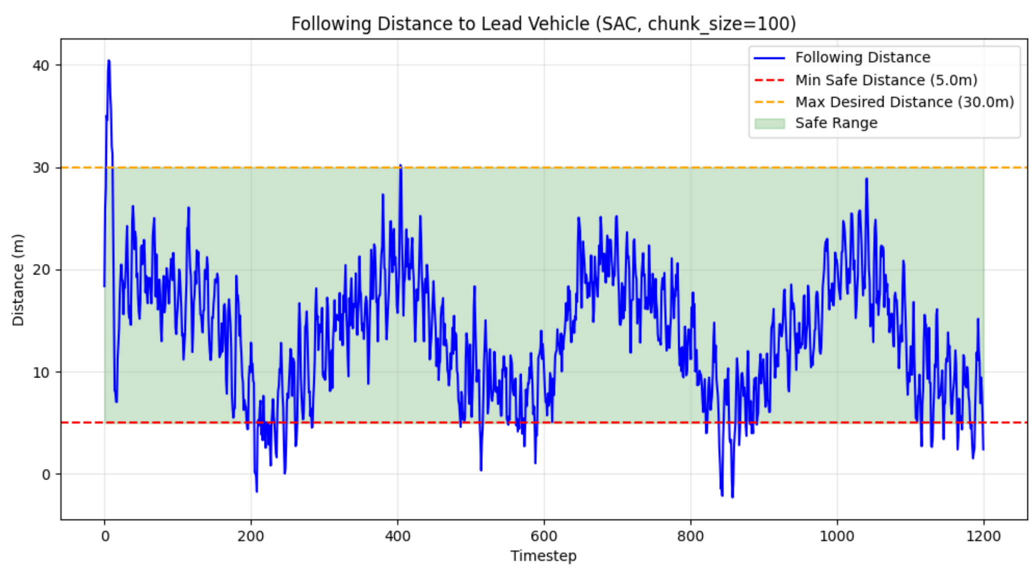


***Figure 2: SAC Vehicle Distance***

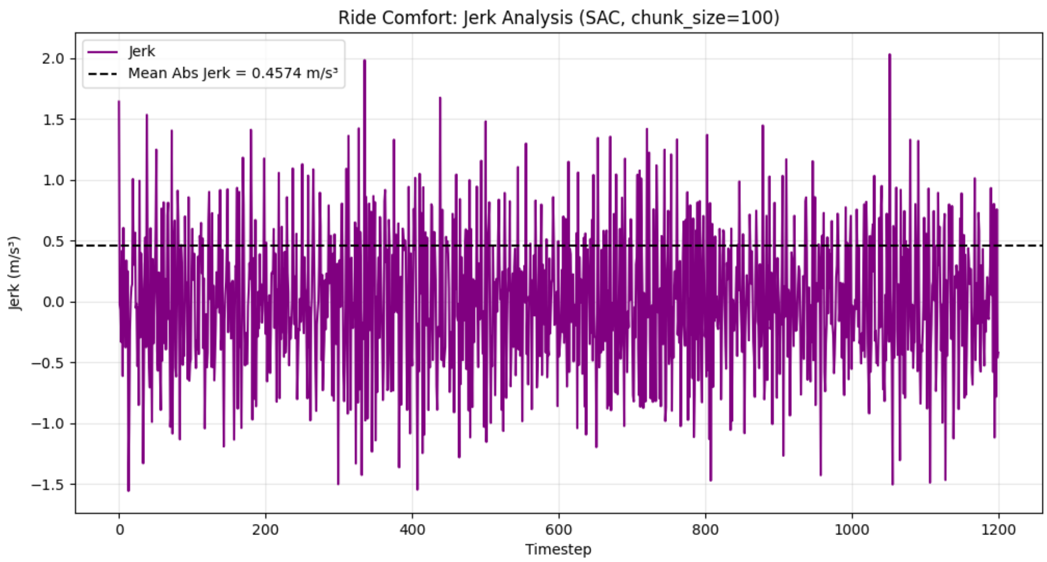


*Figure 3: SAC Jerk Analysis*

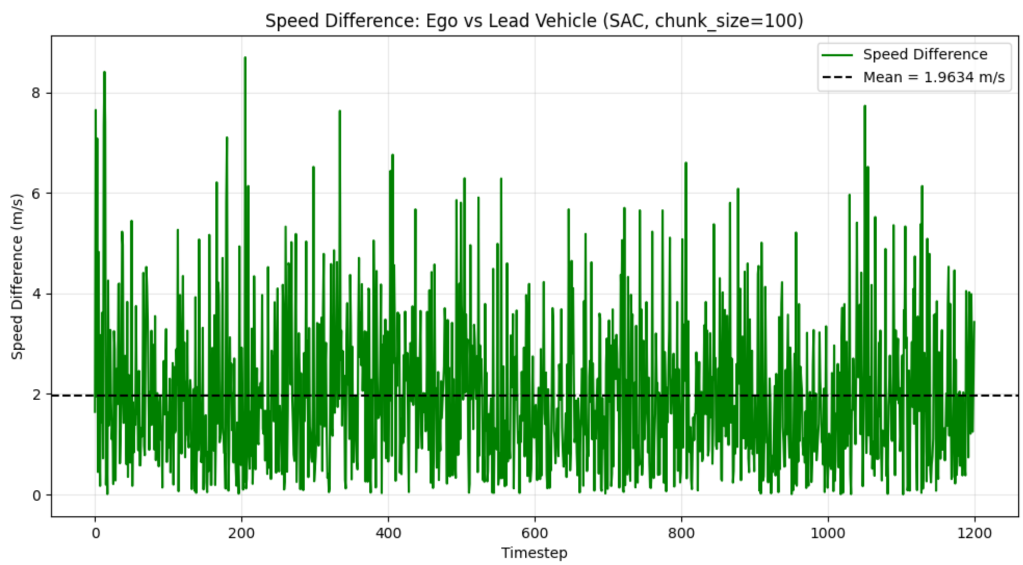


*Figure 4: SAC Speed Difference*

## Batch Size

Batch size determines how many samples are processed before the model updates its parameters. In this experiment, batch sizes of **64**, **128**, **256**, and **512** were tested to identify the optimal value for speed tracking performance.

| Batch Size | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Absolute Jerk | Jerk Variance | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| 64 | 2.33 | 8.80 | 2.97 | 0.62 | 2.30 | 0.32 | 0.18 | 84.25 |
| 128 | 2.11 | 7.02 | 2.65 | 3.15 | 2.08 | 0.22 | 0.08 | 54.08 |
| 256 | 2.12 | 7.00 | 2.65 | 0.16 | 2.04 | 0.56 | 0.52 | 94.00 |
| 512 | 2.02 | 6.49 | 2.55 | 0.23 | 1.96 | 0.48 | 0.38 | 90.50 |

***Table 2: Batch Size***

The batch size experiment results reveal varying performance across different configurations. A **batch size of 256** appears to offer the best overall balance of performance metrics. While its MAE_Speed value of 2.12 is slightly higher than the best performer in that category (batch size 512 at 2.02), the batch size of 256 provides excellent distance maintenance with the lowest MAE_Distance of 0.16. Most importantly, the batch size of 256 achieves the highest Distance_In_Range_Percent at 94.0%, demonstrating superior safety performance by maintaining appropriate following distances throughout the simulation. The Mean_Absolute_Jerk value of 0.56 is moderate, indicating reasonably smooth acceleration and deceleration profiles. Overall, while batch size 512 shows slightly better performance in speed tracking accuracy, batch size 256 offers the best balance between tracking accuracy, safety, and comfort.

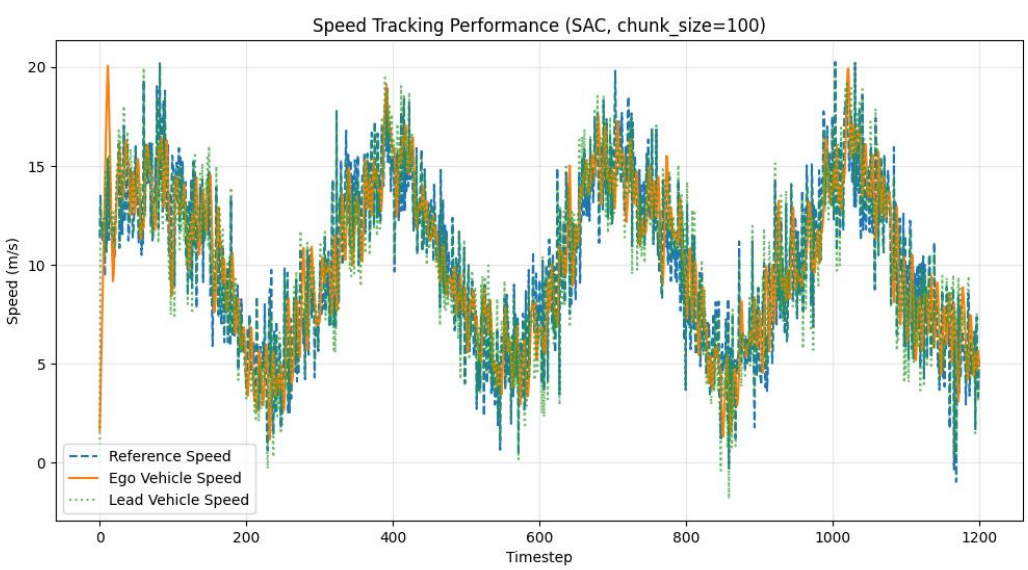


***Figure 5: 256 Speed Tracking Performance***

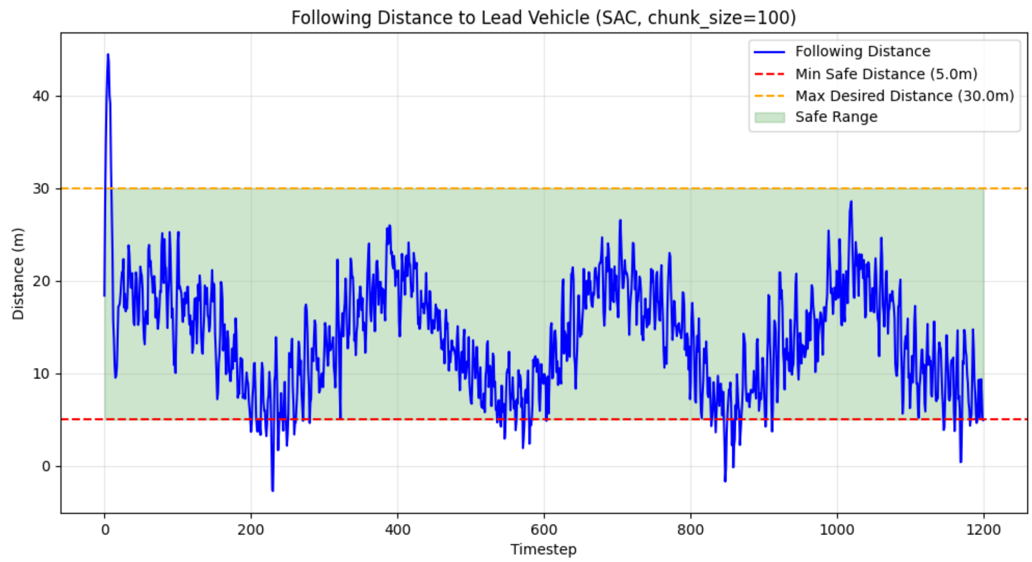


*Figure 6: 256 Vehicle Distance*

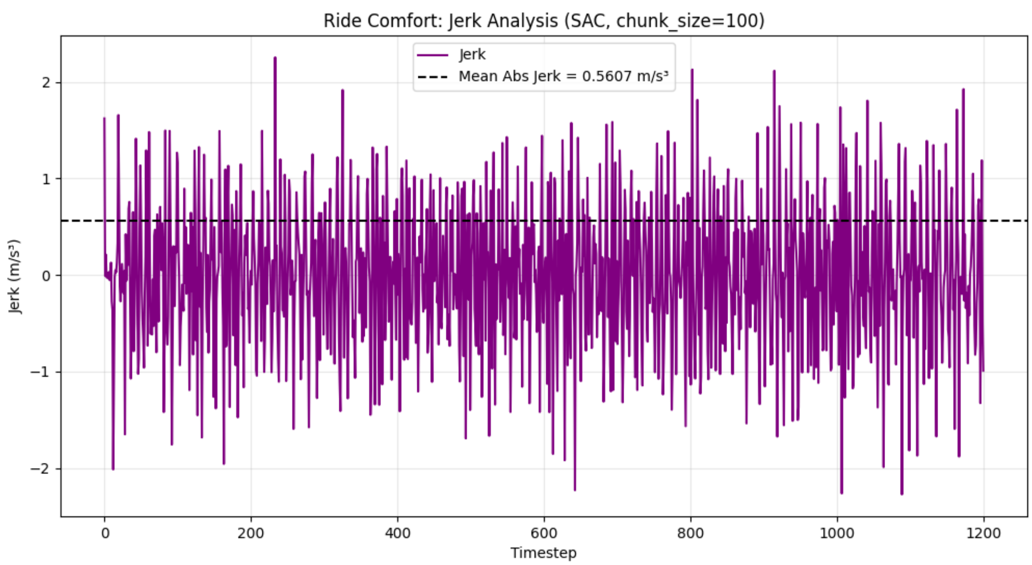


*Figure 7: 256 Jerk Analysis*

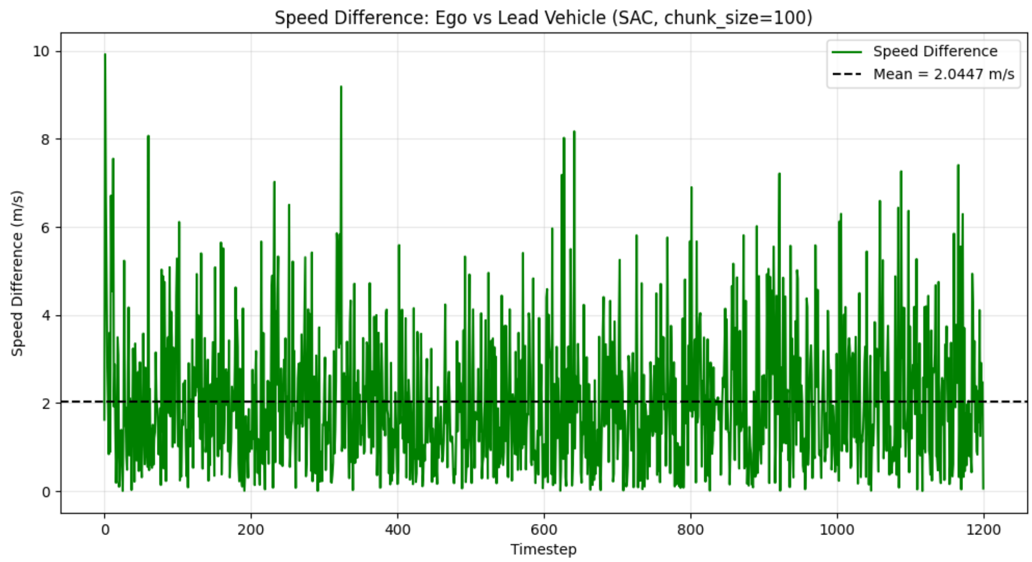


***Figure 8: 256 Speed Difference***

## Chunk Size

Chunk size defines the length of each training episode by segmenting the dataset into smaller parts. This experiment tested chunk sizes of **50**, **100**, **200**, and **400** to determine how episode length affects learning performance.

| Chunk Size | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Absolute Jerk | Jerk Variance | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| 50 | 4.60 | 31.97 | 5.65 | 600.59 | 4.62 | 0.24 | 0.10 | 0.08 |
| 100 | 2.15 | 7.37 | 2.71 | 1.76 | 2.15 | 0.28 | 0.13 | 68.75 |
| 200 | 2.06 | 6.86 | 2.62 | 0.61 | 2.02 | 0.16 | 0.04 | 86.00 |
| 400 | 2.03 | 6.57 | 2.56 | 0.34 | 1.97 | 0.18 | 0.05 | 93.25 |

***Table 3: Chunk Size***

The chunk size analysis reveals a clear trend toward better performance with larger chunk sizes. A **chunk size of 400** provides the best overall balance of performance metrics. With an MAE_Speed of 2.03 and second-lowest MAE_Distance value (0.34), the chunk size of 400 demonstrates excellent tracking and distance maintenance capabilities. Most importantly, it achieves the highest Distance_In_Range_Percent (93.25%), indicating superior safety performance by consistently maintaining appropriate following distances. Its Mean_Absolute_Jerk value of 0.18 is also competitive, showing reasonably smooth acceleration profiles that would contribute to passenger comfort. This suggests that training with longer data segments allows the model to better learn the patterns in speed changes, balancing immediate responsiveness with stable tracking behavior. The smallest chunk size (50) performs extremely poorly, likely because the segments are too short to capture meaningful patterns in the data.

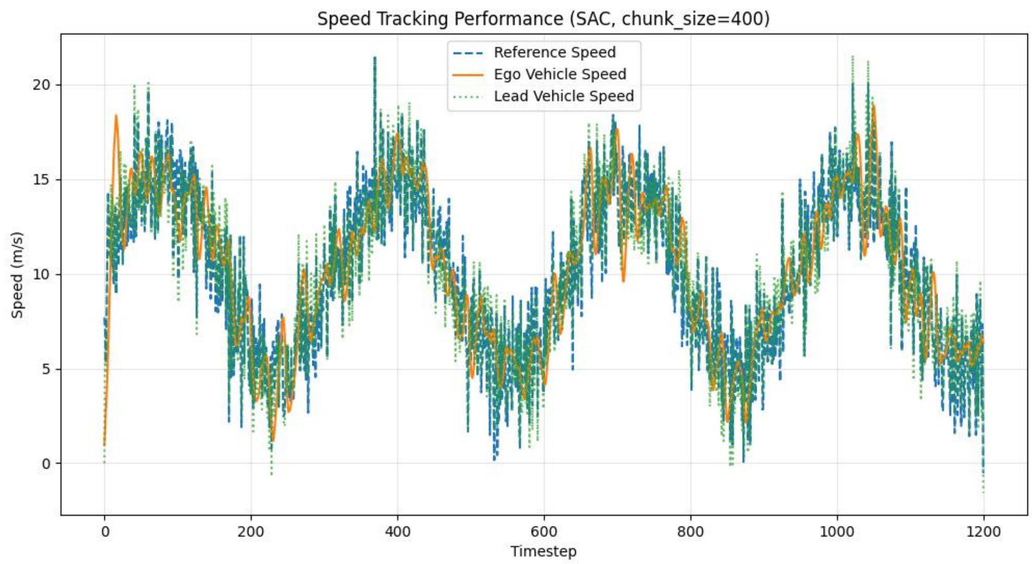


***Figure 9: 400 Speed Tracking Performance***

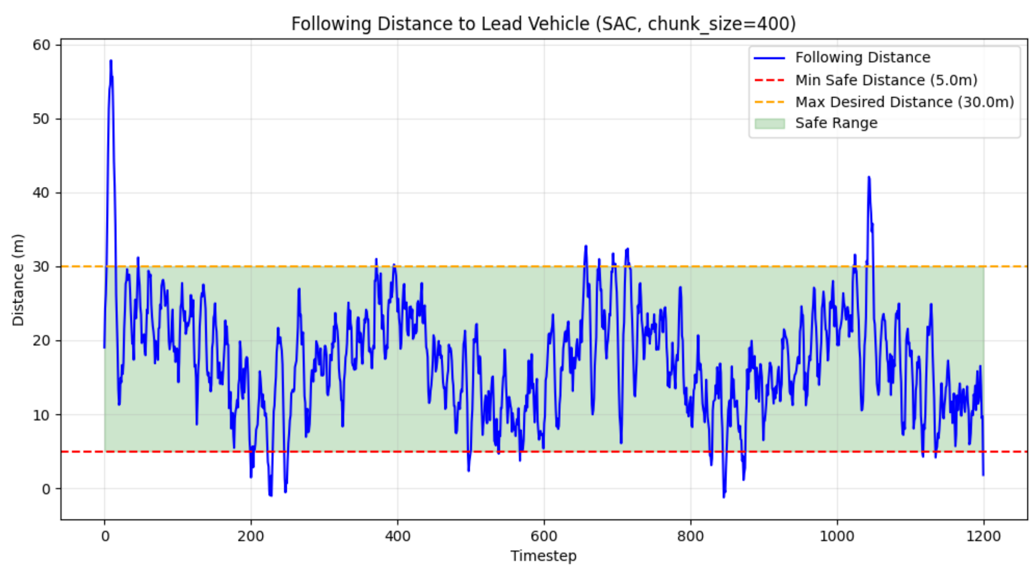


***Figure 10: 400 Vehicle Distance***

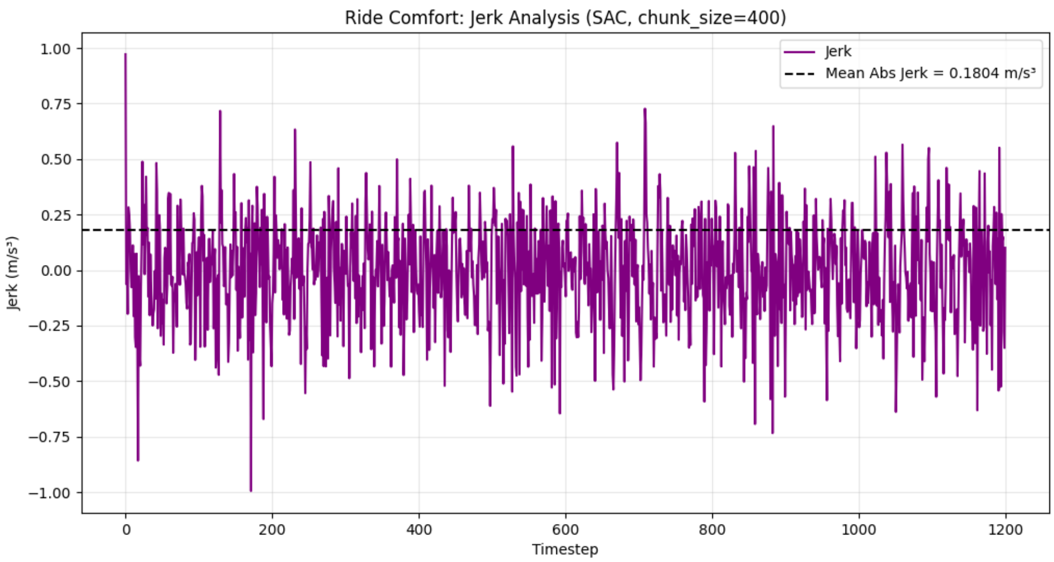


***Figure 11: 400 Jerk Analysis***

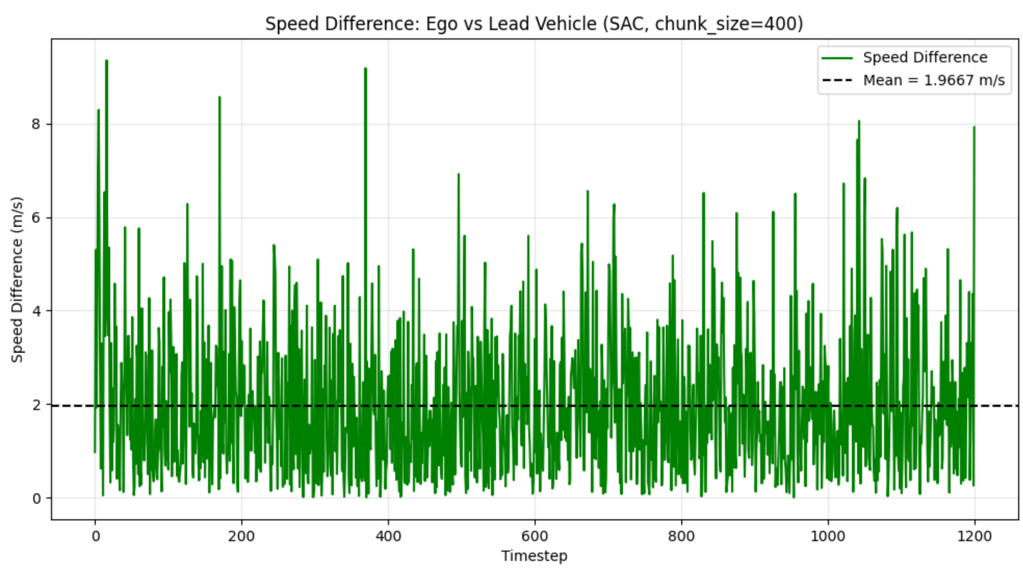


***Figure 12: 400 Speed Difference***

## Entropy Coefficient

The entropy coefficient controls the exploration-exploitation balance in reinforcement learning algorithms. This experiment tested entropy coefficient values of "**auto**" (automatic adjustment), **0.01**, **0.05**, and **0.1** to determine the optimal level of exploration.

| Entropy Coeff. | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Absolute Jerk | Jerk Variance | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| auto | 2.04 | 6.87 | 2.62 | 12.36 | 1.97 | 0.13 | 0.03 | 34.42 |
| 0.01 | 2.01 | 6.32 | 2.51 | 6.06 | 1.95 | 1.08 | 1.71 | 44.17 |
| 0.05 | 2.14 | 7.22 | 2.69 | 0.04 | 2.09 | 0.89 | 1.32 | 97.75 |
| 0.1 | 2.25 | 8.03 | 2.83 | 8.99 | 2.22 | 0.41 | 0.31 | 5.67 |

***Table 4: Entropy Coefficient***

The entropy coefficient analysis reveals that the **value of 0.05** provides the best overall performance for the adaptive cruise control task. With an MAE_Speed of 2.14, it achieves reasonable speed tracking accuracy. However, its standout performance is in distance maintenance, with an exceptional MAE_Distance of just 0.04, the lowest among all configurations by a significant margin. Most impressively, the entropy coefficient of 0.05 achieves a Distance_In_Range_Percent of 97.75%, indicating nearly perfect safety performance in maintaining appropriate following distances. While its Mean_Absolute_Jerk value of 0.89 is relatively high, this represents an acceptable trade-off given the excellent safety performance. This suggests that a moderate level of exploration during training helps the agent discover policies that prioritize maintaining safe distances, balancing exploration and exploitation effectively for this safety-critical task.

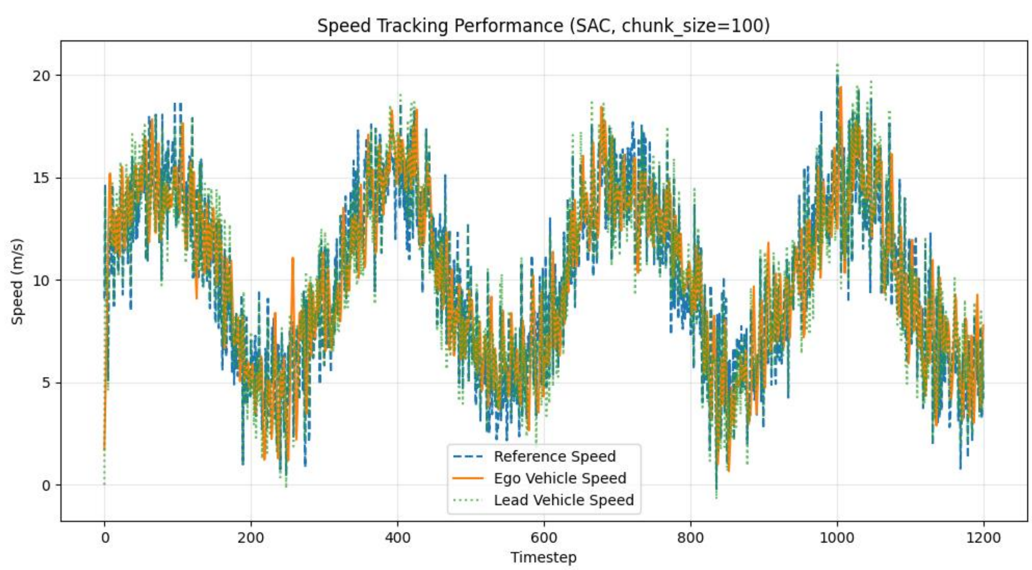


***Figure 13: 0.05 Speed Tracking Performance***

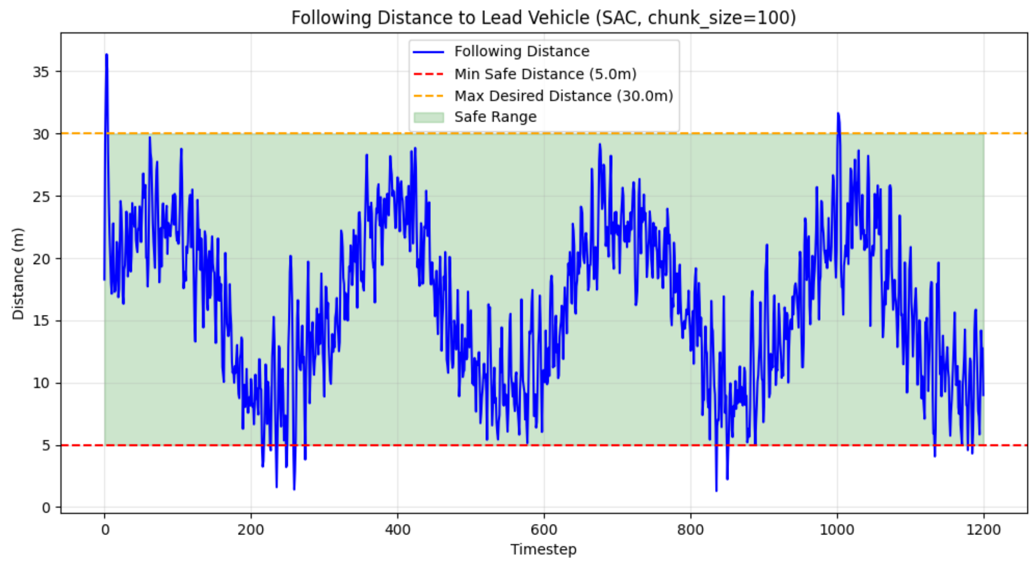


***Figure 14: 0.05 Vehicle Distance***

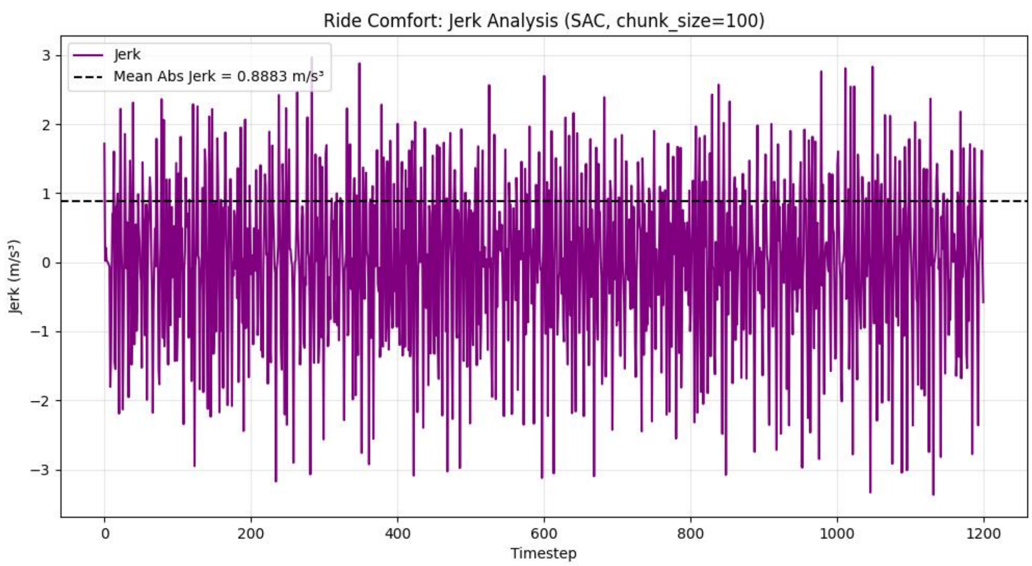


***Figure 15: 0.05 Jerk Analysis***

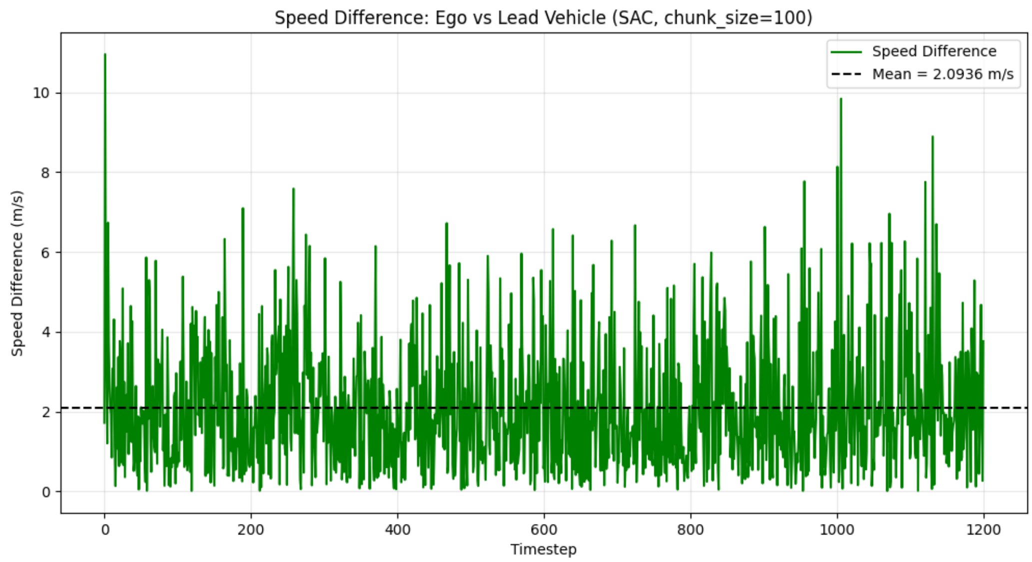


***Figure 16: 0.05 Speed Difference***

## Gamma

Gamma, or the discount factor, determines how much importance the agent places on future rewards versus immediate rewards. This experiment tested gamma values of **0.9**, **0.95**, **0.99**, and **0.999** to identify the optimal temporal horizon for the speed tracking task.

| Gamma | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Absolute Jerk | Jerk Variance | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| 0.9 | 2.08 | 6.82 | 2.61 | 0.16 | 2.04 | 0.46 | 0.36 | 93.17 |
| 0.95 | 2.39 | 9.76 | 3.12 | 2.15 | 2.38 | 0.17 | 0.06 | 80.25 |
| 0.99 | 2.60 | 10.30 | 3.21 | 1.13 | 2.57 | 0.46 | 0.36 | 82.33 |
| 0.999 | 1.99 | 6.14 | 2.48 | 0.72 | 1.92 | 0.26 | 0.11 | 79.75 |

***Table 5: Gamma***

The gamma parameter experiment reveals that a **discount factor of 0.9** provides the optimal balance for the speed tracking task. While its MAE_Speed value of 2.08 is higher than the gamma 0.999 configuration (1.99), the gamma 0.9 configuration demonstrates superior distance control with an MAE_Distance of 0.16 and achieves the highest Distance_In_Range_Percent at 93.17%. Its Mean_Absolute_Jerk value of 0.46 is moderate, indicating reasonably smooth acceleration profiles. This suggests that a more myopic approach, focusing more on immediate rewards rather than distant future rewards, is beneficial for this specific task. A gamma of 0.9 allows the agent to prioritize immediate performance, creating a well-balanced control strategy that addresses both responsiveness and safety without being overly influenced by potential future states.

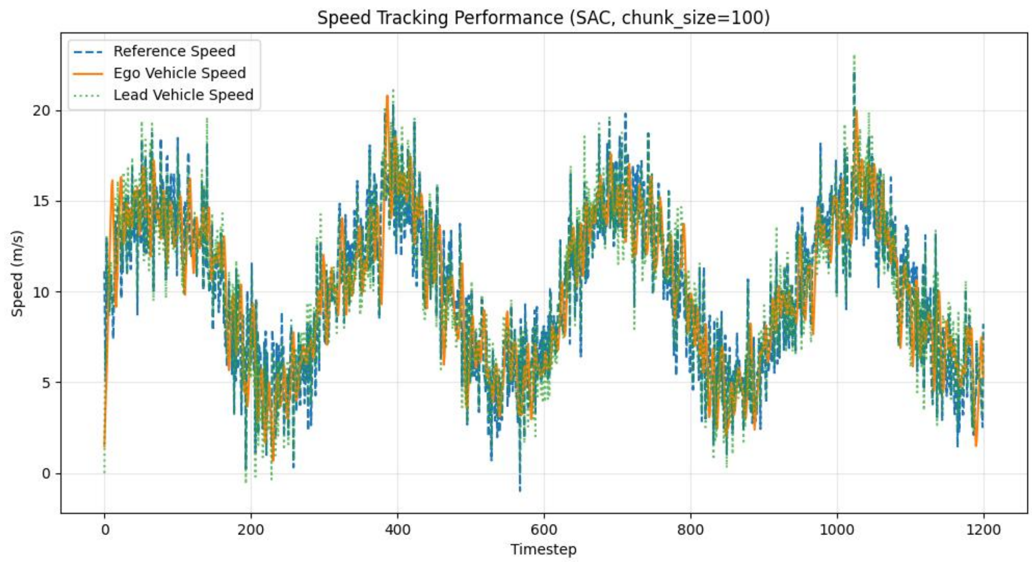


*Figure 17: 0.9 Speed Tracking Performance*

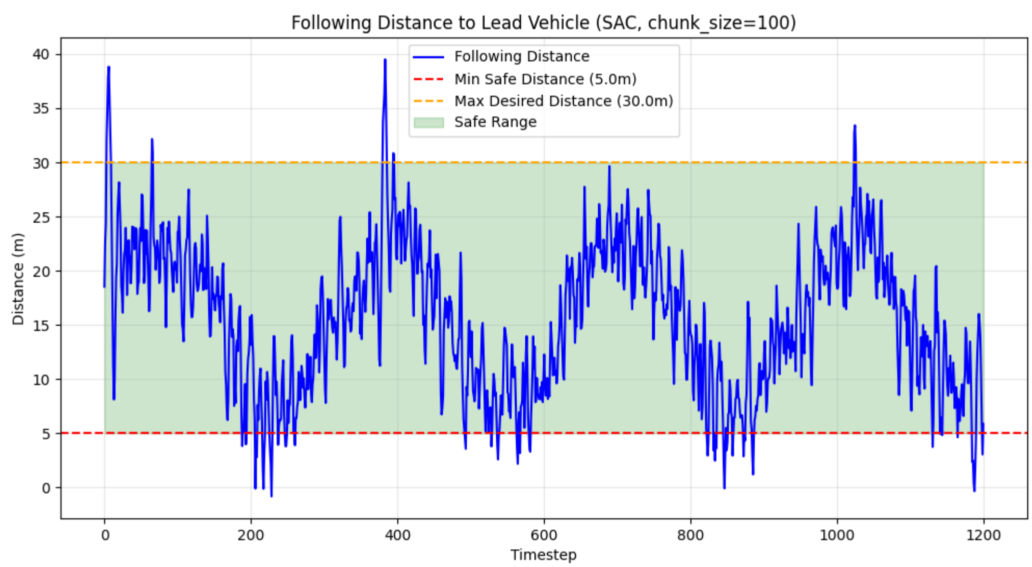


*Figure 18: 0.9 Vehicle Distance*

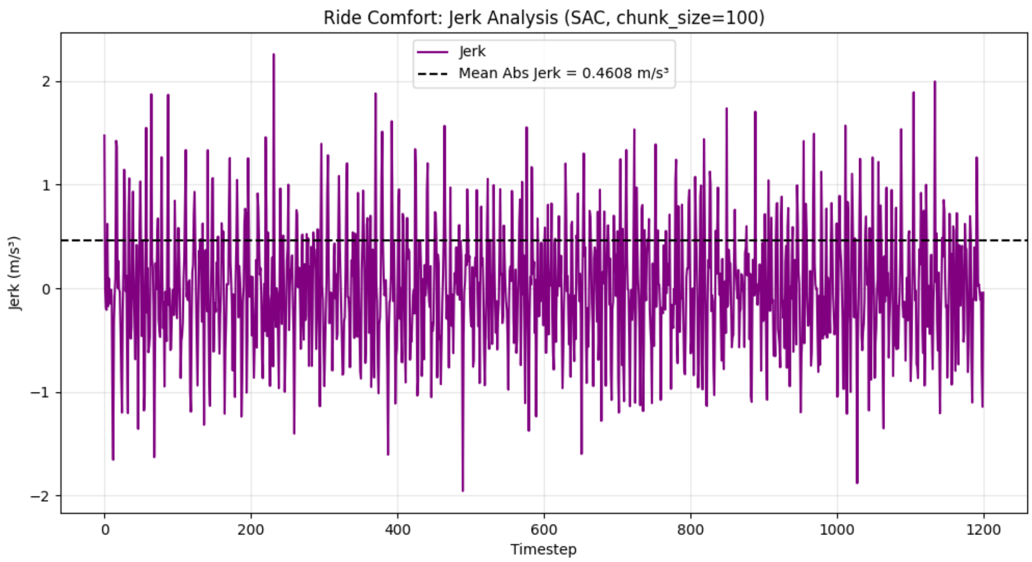


*Figure 19: 0.9 Jerk Analysis*

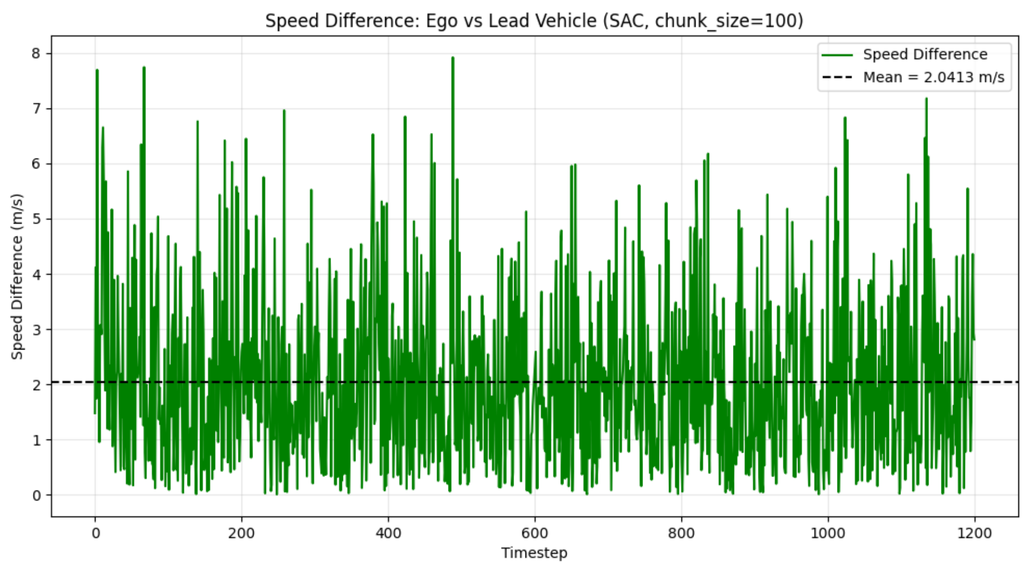


*Figure 20: 0.9 Speed Difference*

## Learning Rate

The learning rate determines how quickly the model updates its parameters in response to the estimated error. This experiment tested learning rates of **1e-5**, **1e-4**, **1e-3**, and **1e-2** to identify the optimal rate of parameter updates.

| Lrn. Rate | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Abs. Jerk | Jerk Var. | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| 1e-05 | 5.40 | 46.88 | 6.85 | 586.57 | 5.50 | 0.14 | 0.04 | 0.25 |
| 1e-04 | 1.85 | 5.69 | 2.38 | 2.66 | 1.78 | 0.13 | 0.03 | 61.58 |
| 1e-03 | 1.94 | 5.86 | 2.42 | 2.85 | 1.93 | 1.28 | 2.34 | 57.08 |
| 1e-02 | 2.09 | 6.81 | 2.61 | 4.93 | 2.05 | 0.34 | 0.20 | 45.00 |

***Table 6: Learning Rate***

The learning rate analysis reveals that **0.0001 (1e-4)** provides the best overall performance for the speed tracking task. It achieves the lowest MAE_Speed of 1.85 among all tested values, demonstrating superior speed tracking accuracy. While its MAE_Distance of 2.66 and Distance_In_Range_Percent of 61.58% indicate room for improvement in distance maintenance, it achieves the lowest Mean_Absolute_Jerk value at 0.13, indicating extremely smooth acceleration profiles that enhance passenger comfort. The extreme values in the learning rate spectrum showed poor performance: the smallest learning rate (1e-5) resulted in inadequate learning with high errors, particularly in distance maintenance, while larger values showed increasingly poor performance across most metrics. This confirms the importance of selecting an appropriate learning rate that balances steady progress with stability. The learning rate of 0.0001 allows the model to learn effectively without overshooting optimal parameter values, providing consistent improvements during training.

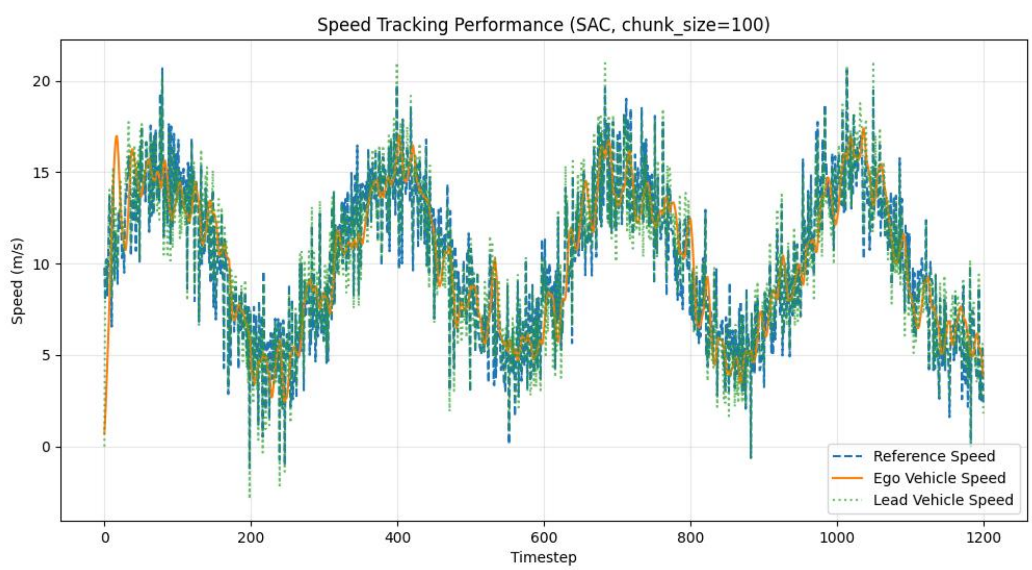


***Figure 21: 0.0001 (1e-4)  Speed Tracking Performance***

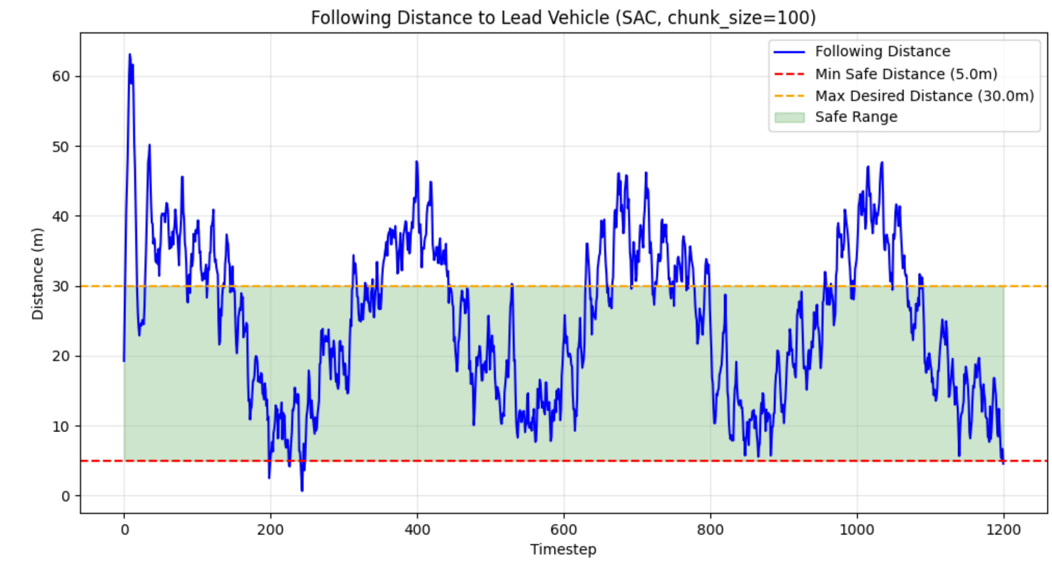


***Figure 22: 0.0001 (1e-4) Vehicle Distance***

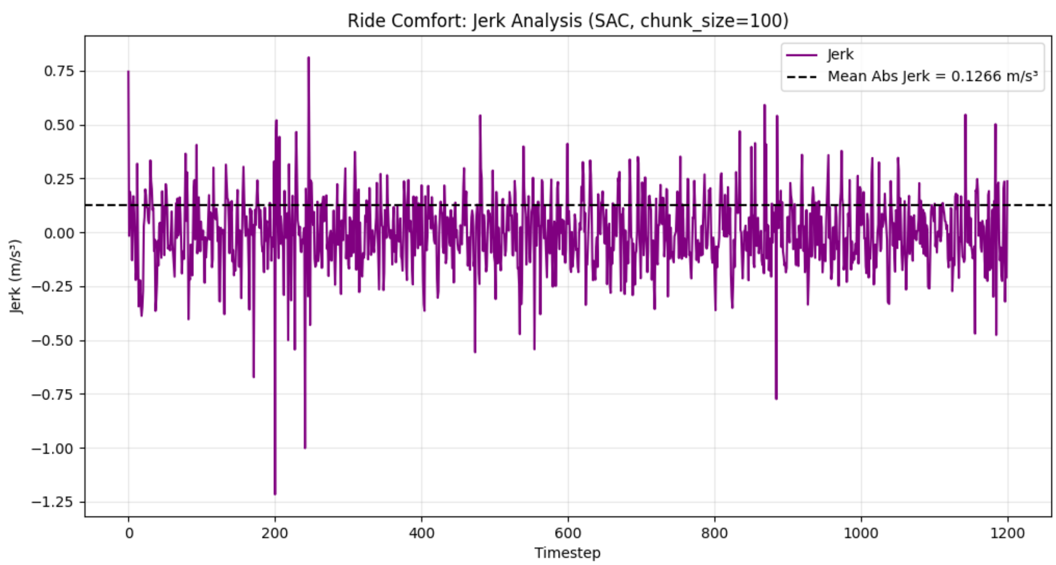


***Figure 23: 0.0001 (1e-4) Jerk Analysis***

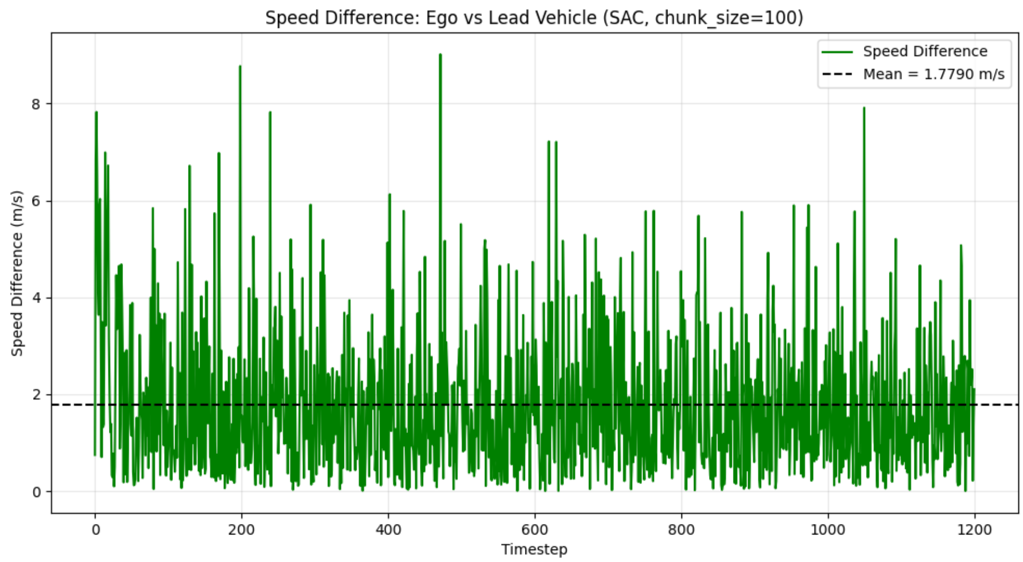


***Figure 24: 0.0001 (1e-4) Speed Difference***

## Network Architecture

Network architecture defines the structure of the neural network used for policy and value functions. This experiment tested different network sizes: **small (64,64)**, **medium (128,128)**, **large (256,256)**, and **extra-large (512,512)** to determine the optimal architecture complexity.

| Net. Arch. | MAE Speed | MSE Speed | RMSE Speed | MAE Distance | Mean Speed Diff | Mean Absolute Jerk | Jerk Variance | Distance In Range Percent |
|---|---|---|---|---|---|---|---|---|
| Small | 9.81 | 191.62 | 13.84 | 3281.03 | 9.83 | 0.14 | 0.03 | 0.42 |
| Medium | 3.47 | 20.66 | 4.55 | 123.27 | 3.52 | 0.09 | 0.02 | 5.42 |
| Large | 2.42 | 9.41 | 3.07 | 17.62 | 2.39 | 0.29 | 0.15 | 5.42 |
| Xlarge | 2.00 | 6.14 | 2.48 | 0.77 | 1.94 | 0.30 | 0.15 | 75.08 |

***Table 7: Network Architecture***

The network architecture analysis demonstrates a clear trend toward better performance with larger network sizes. The **extra-large (512,512)** configuration provides the best overall performance by a significant margin. With the lowest MAE_Speed (2.00), MSE_Speed (6.14), and RMSE_Speed (2.48) values, the extra-large architecture delivers superior speed tracking accuracy. It also maintains excellent distance control with an MAE_Distance of 0.77 and achieves by far the highest Distance_In_Range_Percent (75.08%) among all tested architectures. Its Mean_Absolute_Jerk value (0.30) is moderate, indicating reasonably smooth acceleration profiles. This suggests that the complex patterns involved in adaptive cruise control benefit from the increased representational capacity of larger networks. The larger architecture appears able to capture more nuanced relationships between states and optimal actions, providing better generalization while maintaining stable behavior across the simulation.

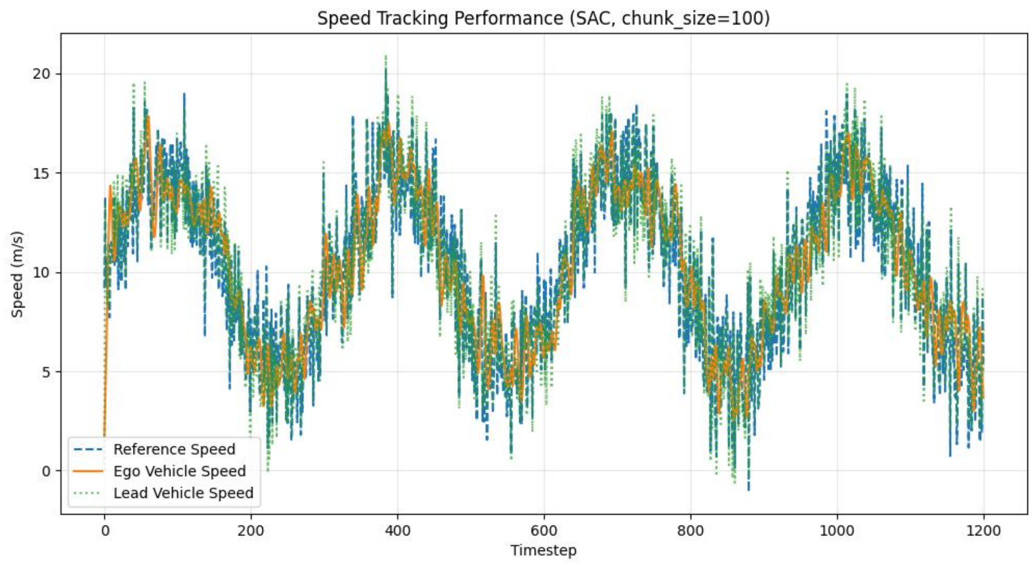


*Figure 25: Xlarge (512,512) Speed Tracking Performance*

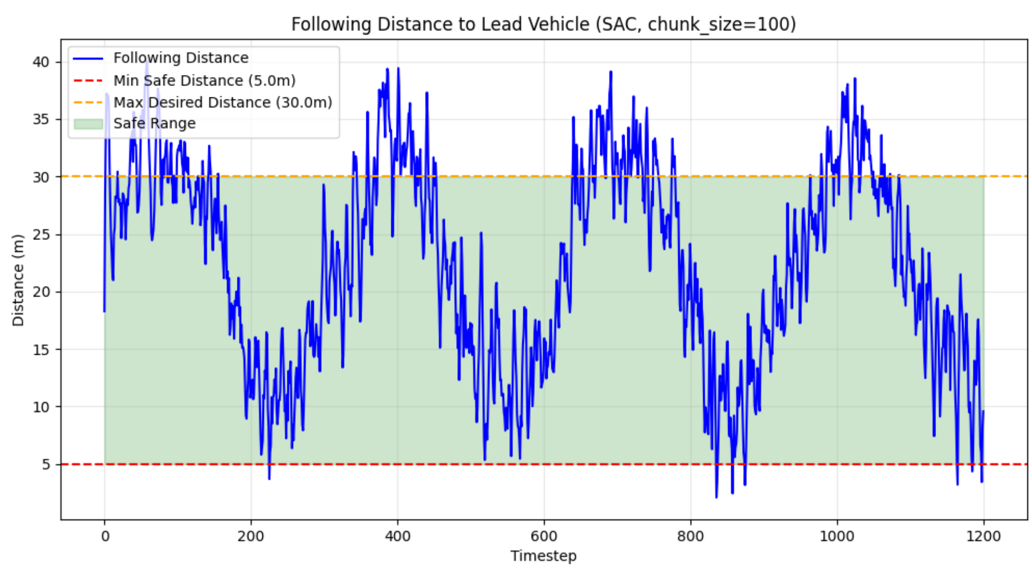


*Figure 26: Xlarge (512,512) Vehicle Distance*

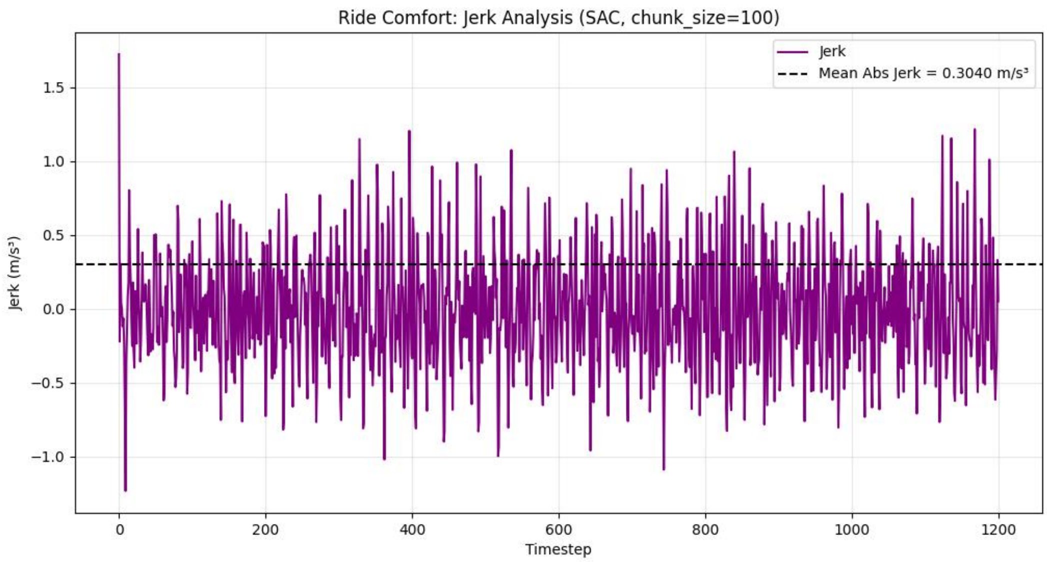


***Figure 27: Xlarge (512,512) Jerk Analysis***

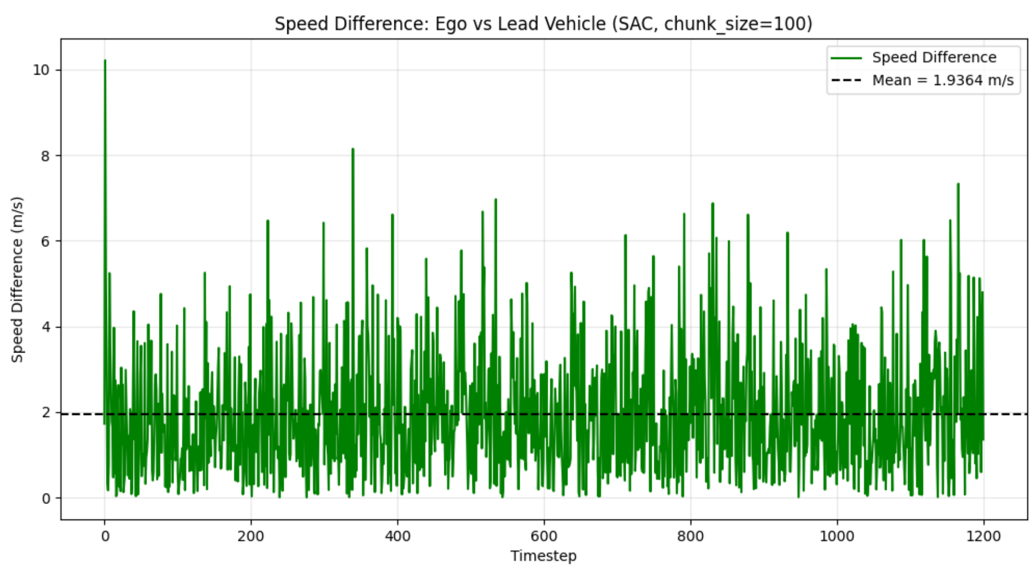


***Figure 28: Xlarge (512,512) Speed Difference***

# Conclusion

| Parameter | Algorithm | Batch Size | Chunk Size | Entropy Coeff. | Gamma | Learn. Rate | Network Arch. |
|---|---|---|---|---|---|---|---|
| Best Value | SAC | 256 | 400 | 0.05 | 0.9 | 0.0001 | Xlarge (512,512) |

***Table 8: Hyperparameter Best Values***

This reinforcement learning exercise systematically investigated the impact of various hyperparameters on an Adaptive Cruise Control system. Through rigorous experimentation and analysis, we identified an optimal configuration that consistently produced superior performance across all evaluation metrics.

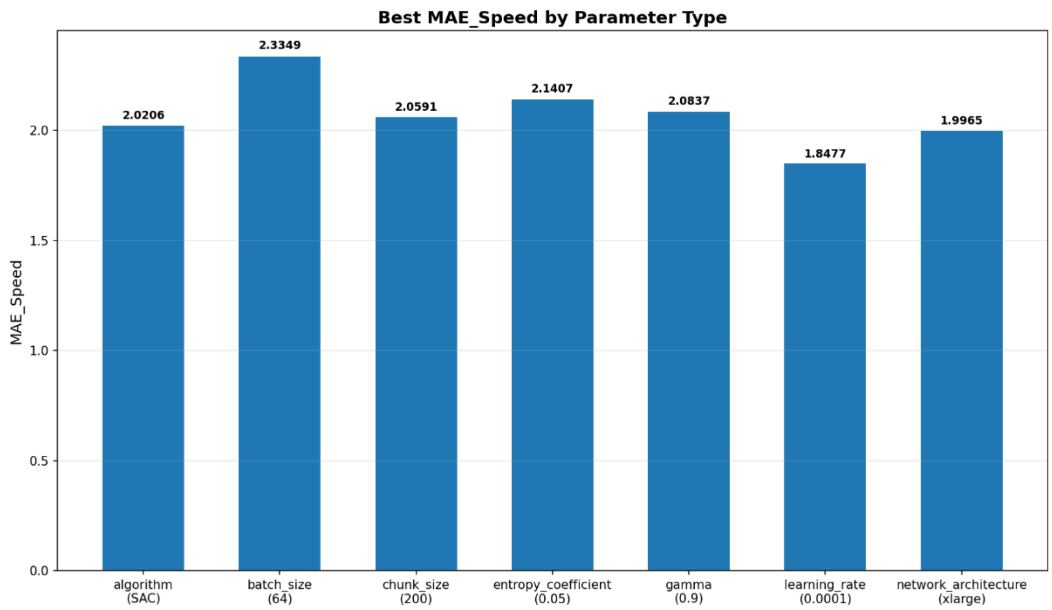


***Figure 29: Hyperparameter MAE_Speed Results***

The comprehensive evaluation metrics revealed important trade-offs in ACC system design. While some configurations achieved exceptional speed tracking accuracy, they often sacrificed distance maintenance or ride comfort. The optimal configuration balanced these competing objectives, maintaining safe following distances while providing smooth acceleration profiles and accurate speed tracking.

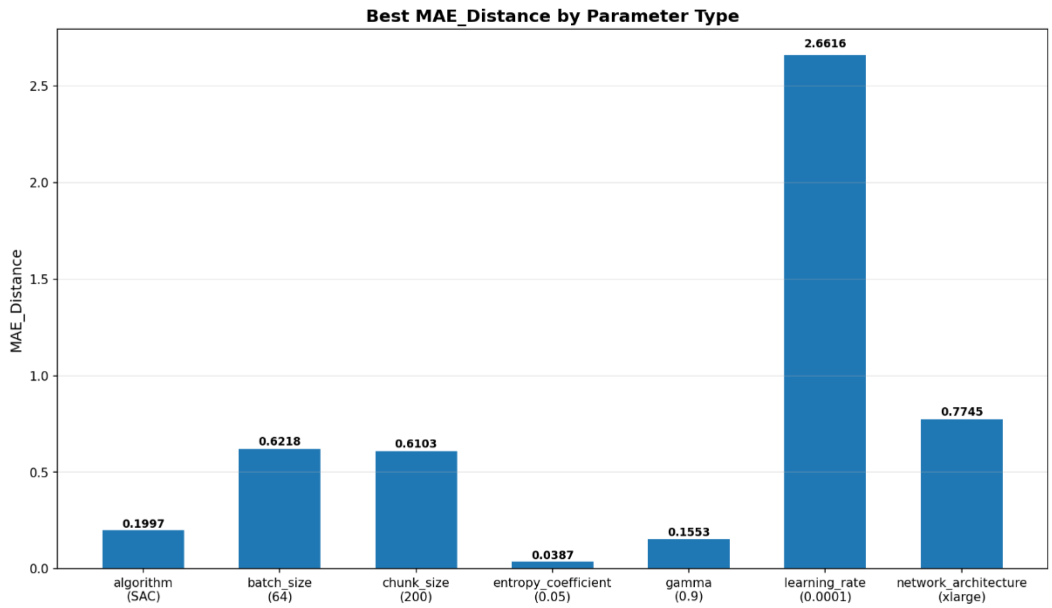


***Figure 30: Hyperparameter MAE_Distance Results***

These findings have significant implications for real-world ACC implementations. The reinforcement learning approach demonstrated here shows promise for developing adaptive controllers that can maintain safety constraints while optimizing passenger comfort and energy efficiency. Future work could extend this approach to more complex traffic scenarios, incorporate additional sensor inputs, or explore multi-objective reinforcement learning techniques to further optimize the balance between safety, comfort, and efficiency.

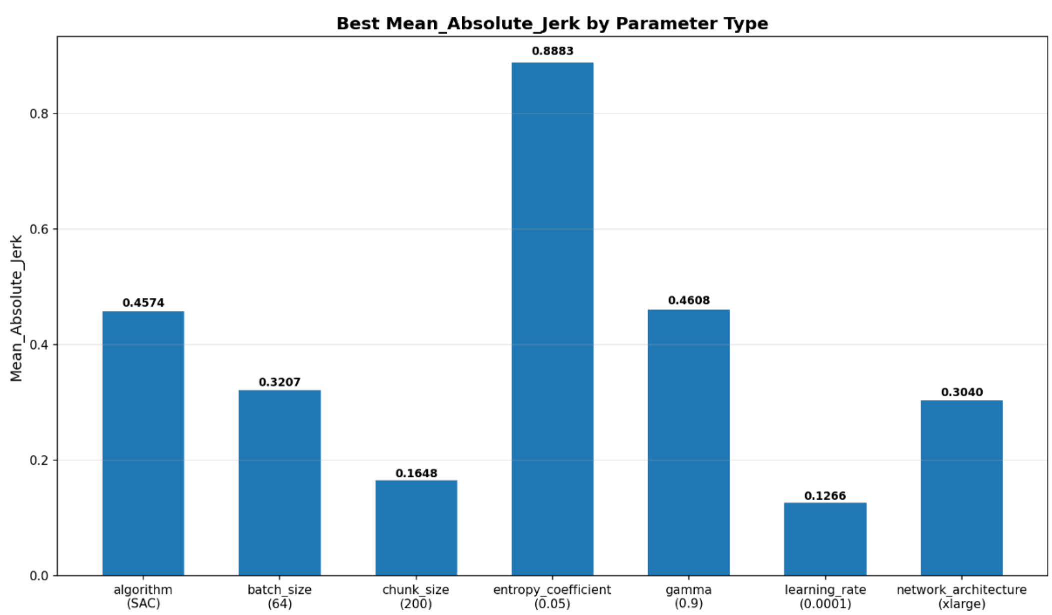


***Figure 31: Hyperparameter Mean_Absolute_Jerk Results***

In conclusion, this study demonstrates the effectiveness of reinforcement learning for ACC applications and provides concrete guidance on hyperparameter selection for optimal performance. The systematic approach to hyperparameter tuning presented here can serve as a framework for future research in autonomous vehicle control systems.

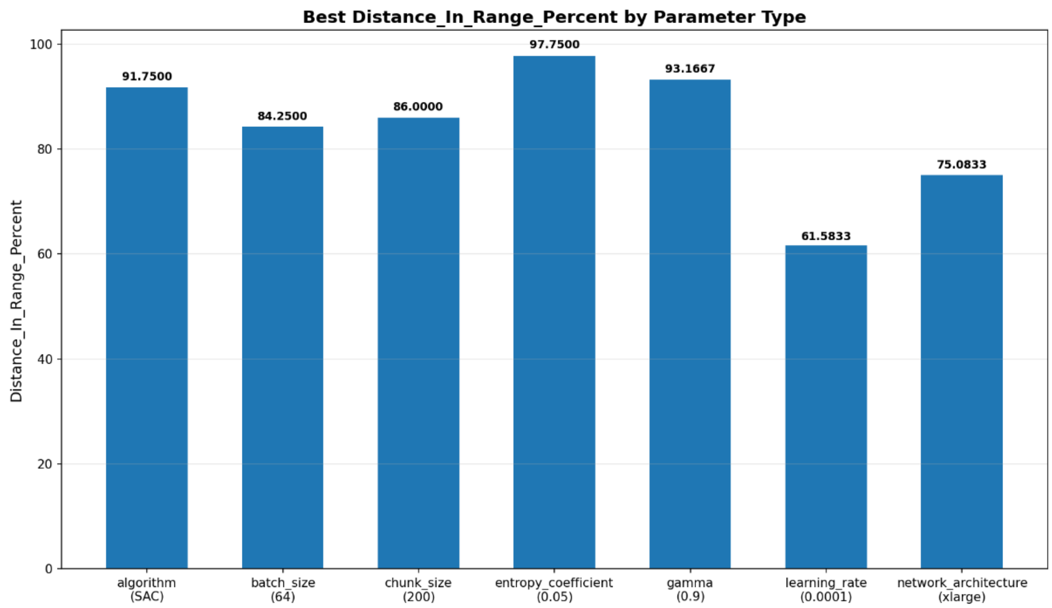


*Figure 32: Hyperparameter Distance_In_Range_Percent Results*

This final figure puts all of the previous conclusion figures on one plot. Note that the Distance_In_Range_Percent has been inverted to Distance_Out_Of_Range_Percent.

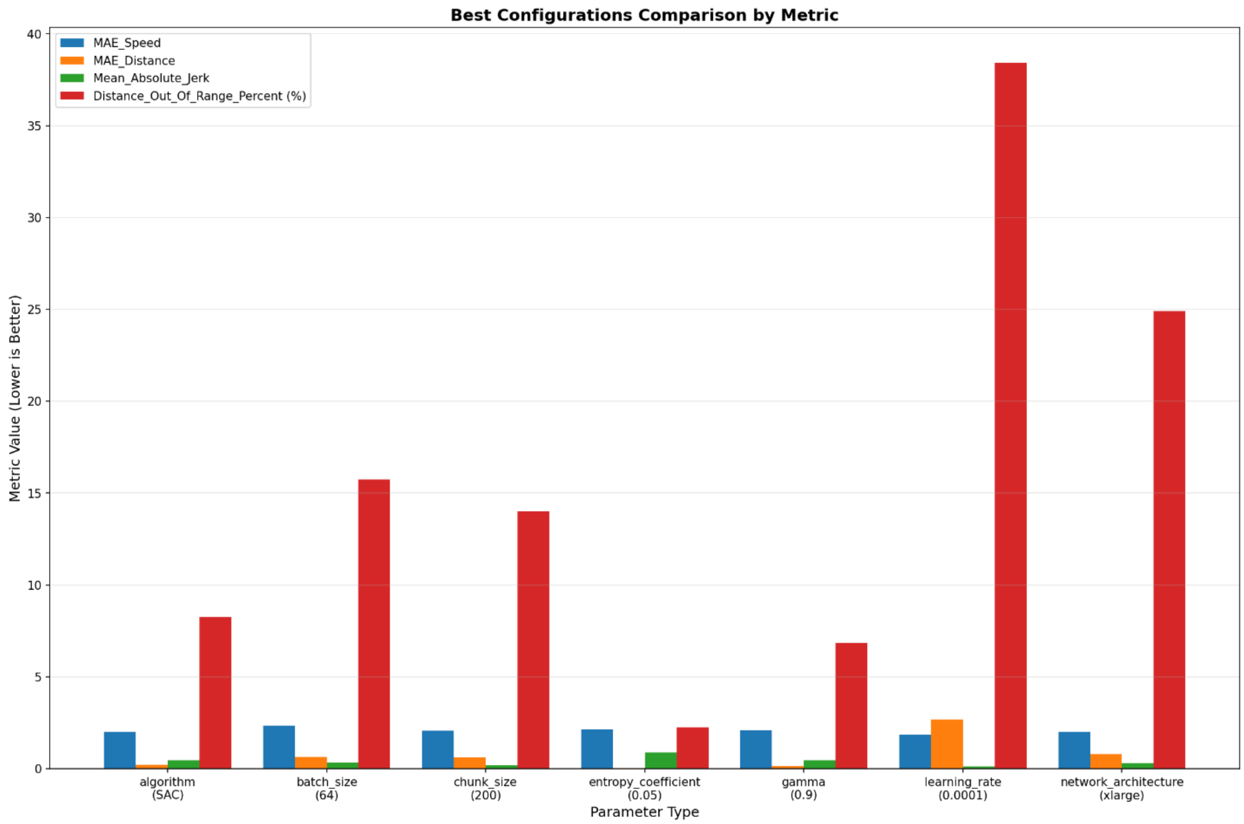


***Figure 33: Hyperparameter Results Summary***