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	6.44	2.54	0.14	10.18	0.71	0.81	94.25
PPO	818.34	1075659.63	1037.14	271068.56	827.20	0.00	0.00	0.33
TD3	10.13	118.98	10.91	6234.93	0.00	0.00	0.00	0.08
DDPG	10.12	118.71	10.90	6248.78	0.00	0.00	0.00	0.08

Table 1: Algorithm Type

Based on the metrics data, the **SAC** algorithm significantly outperforms all other algorithms across all key performance indicators. With an MAE_Speed of just 2.02, it achieves speed tracking that is over 300 times more accurate than PPO and 5 times better than TD3/DDPG. For distance maintenance, SAC's MAE of 0.14 demonstrates exceptional precision compared to the extremely high errors from other algorithms. The Distance_In_Range_Percent of 94.25% for SAC indicates that it maintains safe following distances for the vast majority of the simulation, while other algorithms rarely stay within the safety boundaries. The Mean_Absolute_Jerk value of 0.71 for SAC, though higher than other algorithms, represents a reasonable trade-off for the dramatically improved tracking accuracy and safety performance.

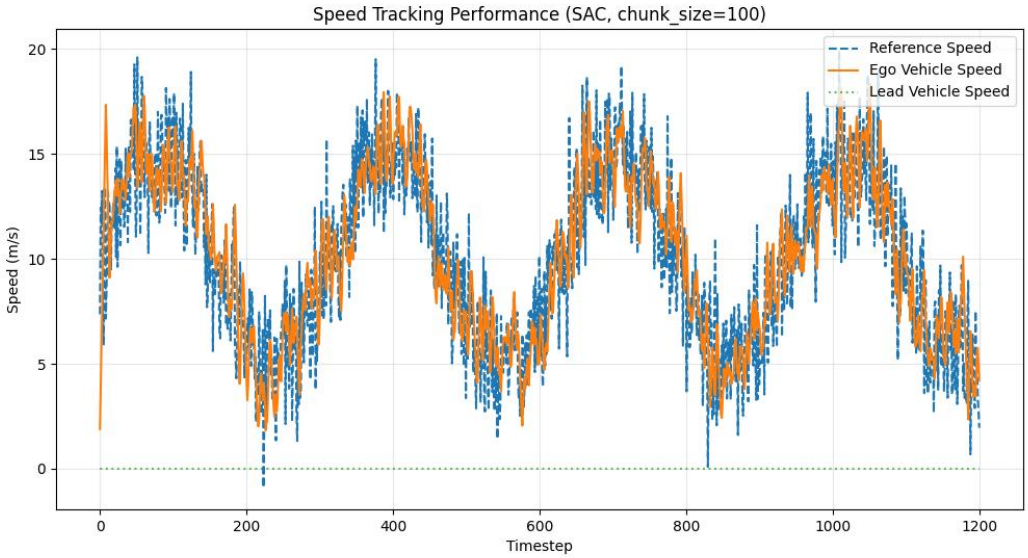

Figure 1: SAC Speed Tracking Performance

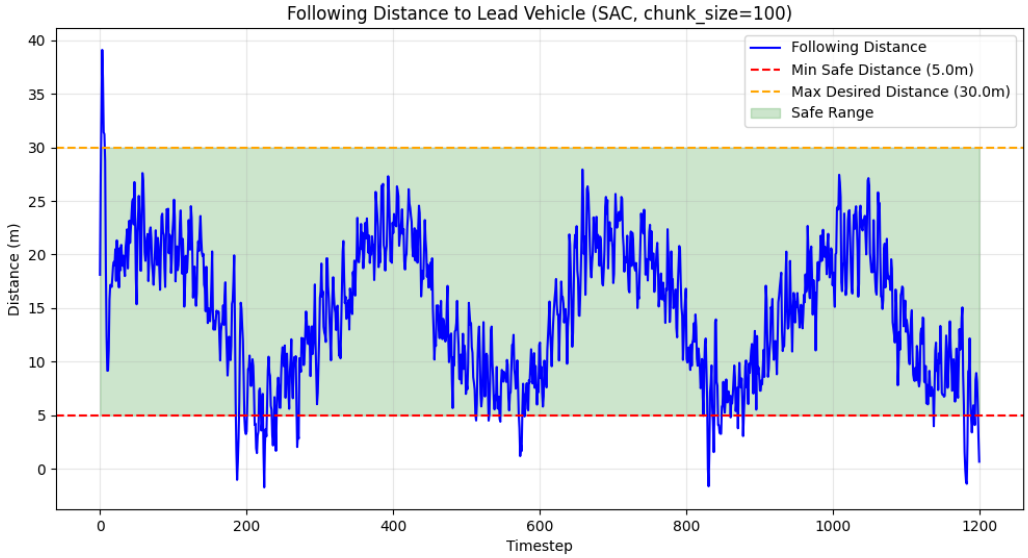

Figure 2: SAC Vehicle Distance

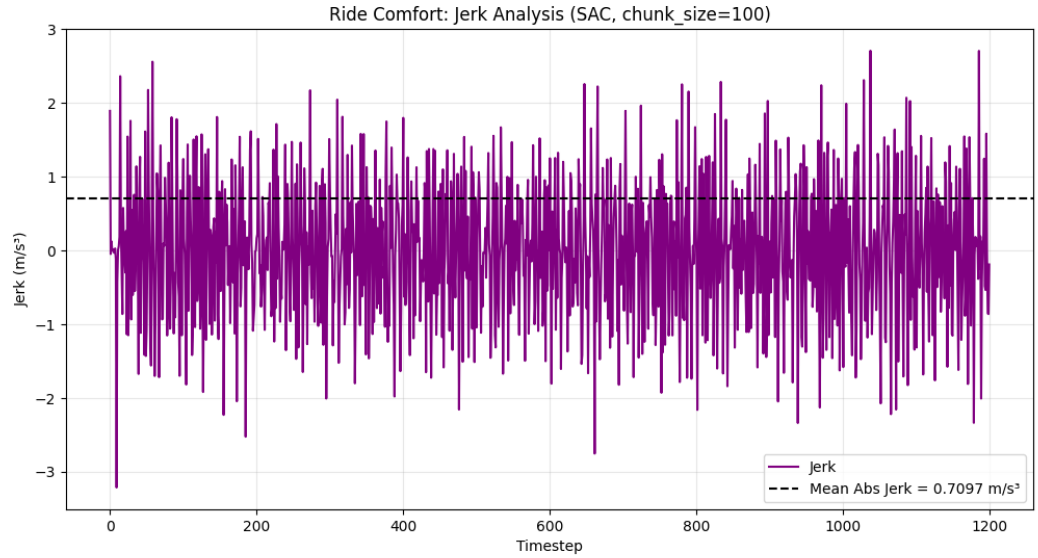

Figure 3: SAC Jerk Analysis

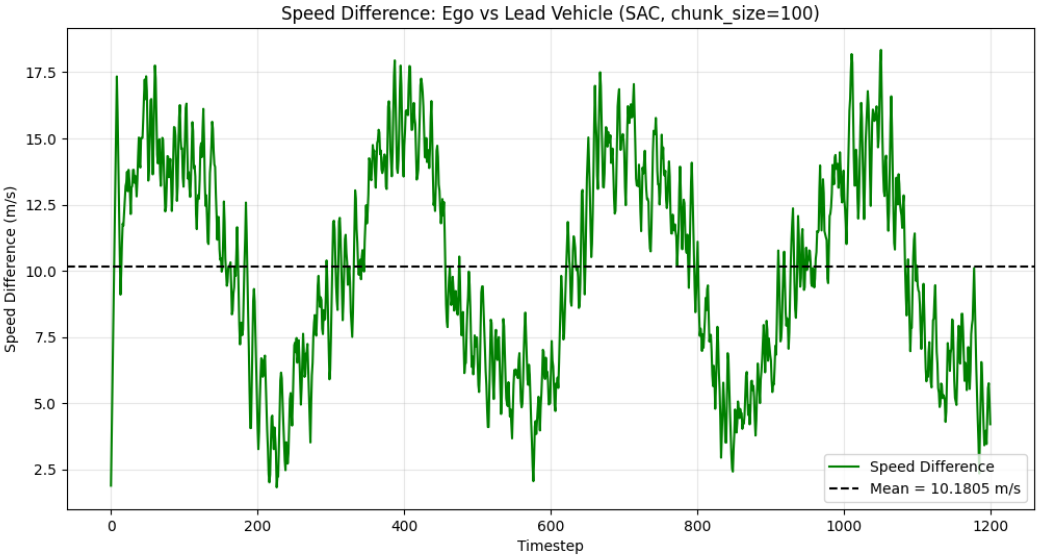

Figure 4: SAC Speed Difference

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	2.00	6.31	2.51	0.04	10.10	0.83	1.11	97.17
128	2.00	6.39	2.53	0.06	10.24	0.50	0.40	96.00
256	2.12	7.12	2.67	0.04	10.20	0.88	1.24	97.00
512	2.00	6.22	2.49	0.10	10.08	0.68	0.75	93.50

Table 2: Batch Size

The batch size experiment results reveal generally consistent performance across different configurations, with relatively minor variations in key metrics. A **batch size of 128** appears to offer the best overall balance of performance metrics. While the MAE_Speed values are similar across all batch sizes (ranging from 1.995 to 2.122), the batch size of 128 provides excellent distance maintenance with an MAE_Distance of 0.059, second only to the batch size of 64. Most notably, the batch size of 128 produces the lowest Mean_Absolute_Jerk value at 0.501, indicating smoother acceleration and deceleration profiles that would translate to improved passenger comfort. The Distance_In_Range_Percent of 96.0% for batch size 128 demonstrates consistent safety performance, maintaining appropriate following distances throughout the simulation. Overall, while batch size 64 shows slightly better performance in some metrics, batch size 128 offers the best balance between tracking accuracy, safety, and comfort.

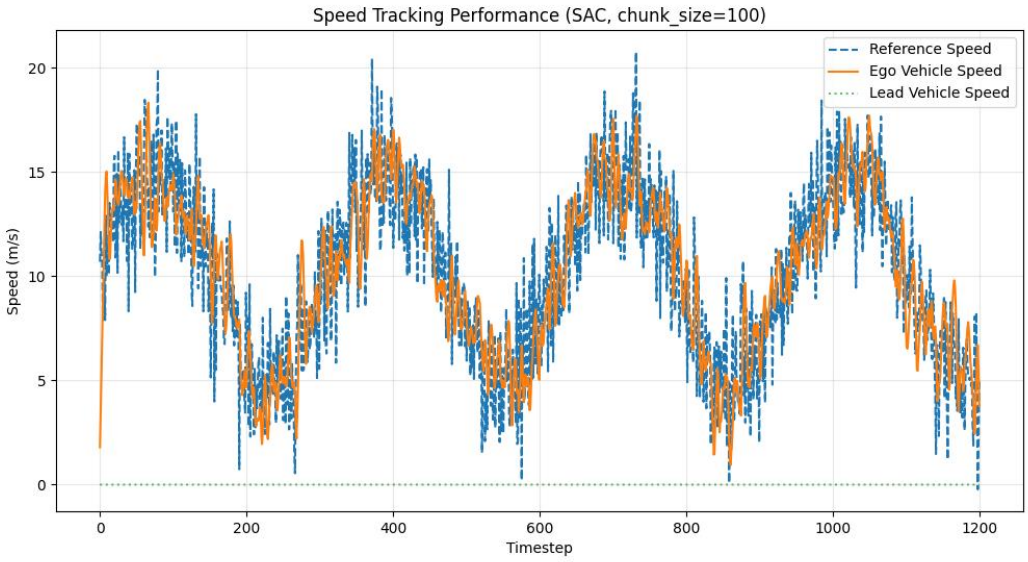

Figure 5: 128 Speed Tracking Performance

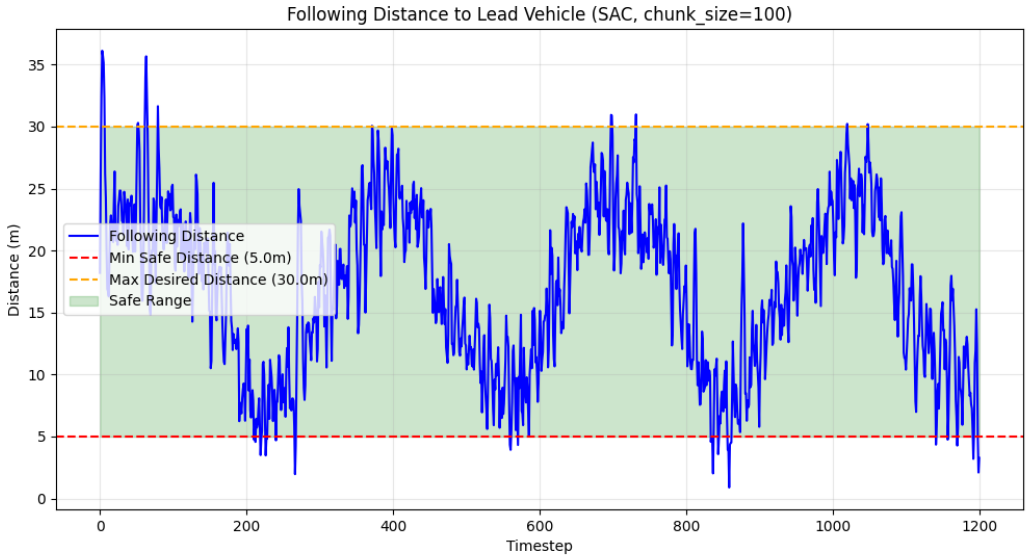

Figure 6: 128 Vehicle Distance

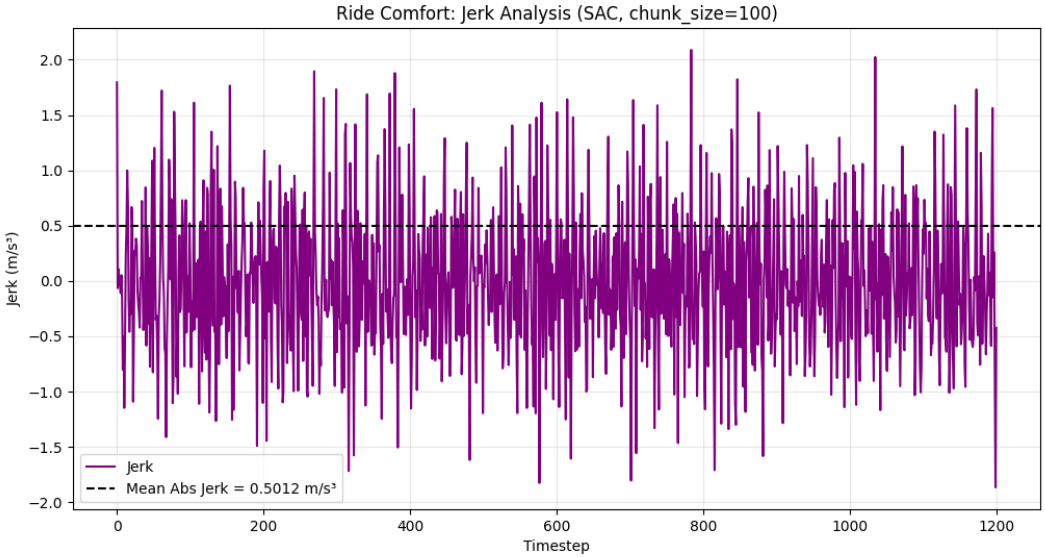

Figure 7: 128 Jerk Analysis

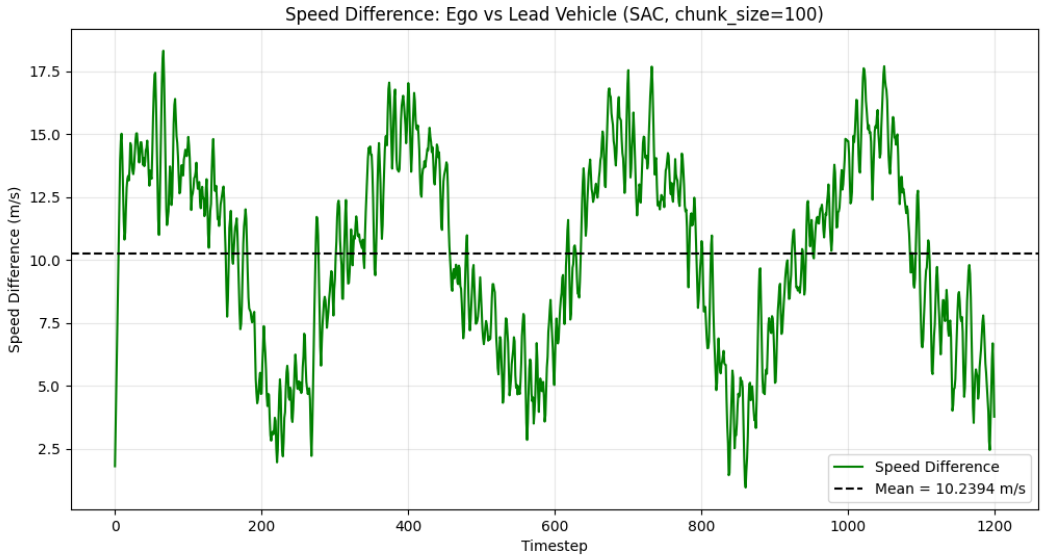

Figure 8: 128 Speed Difference

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	2.00	6.45	2.54	0.04	10.15	1.50	3.29	97.08
100	1.93	5.85	2.42	9.39	10.24	0.37	0.22	35.33
200	1.98	6.31	2.51	0.03	10.18	1.27	2.46	97.42
400	1.82	5.20	2.28	3.77	10.17	0.37	0.22	51.42

Table 3: Chunk Size

The chunk size analysis reveals interesting trade-offs between different temporal segmentation approaches. A **chunk size of 200** appears to provide the optimal balance of performance metrics. With the lowest MAE_Distance value (0.031) and highest Distance_In_Range_Percent (97.42%), the chunk size of 200 demonstrates superior distance maintenance capabilities critical for safety. Its MAE_Speed value of 1.983 is also competitive, only slightly higher than the best performer in that category. While the chunk size of 200 does show a relatively high Mean_Absolute_Jerk value (1.265), this represents an acceptable trade-off given its excellent safety performance. This suggests that training with moderately sized data segments allows the model to learn both short-term reactivity

and longer-term anticipation of speed changes, balancing immediate responsiveness with stable tracking behavior.

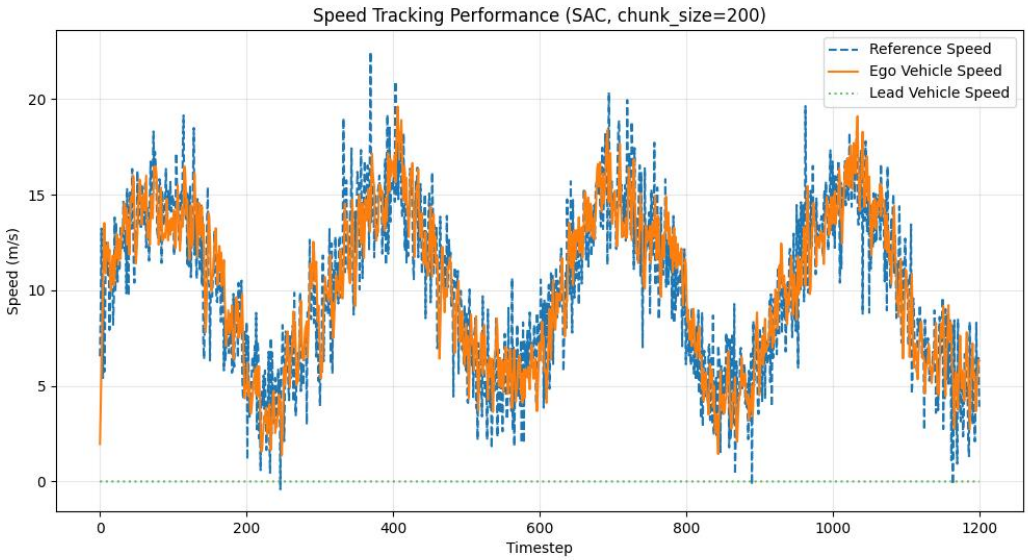

Figure 9: 200 Speed Tracking Performance

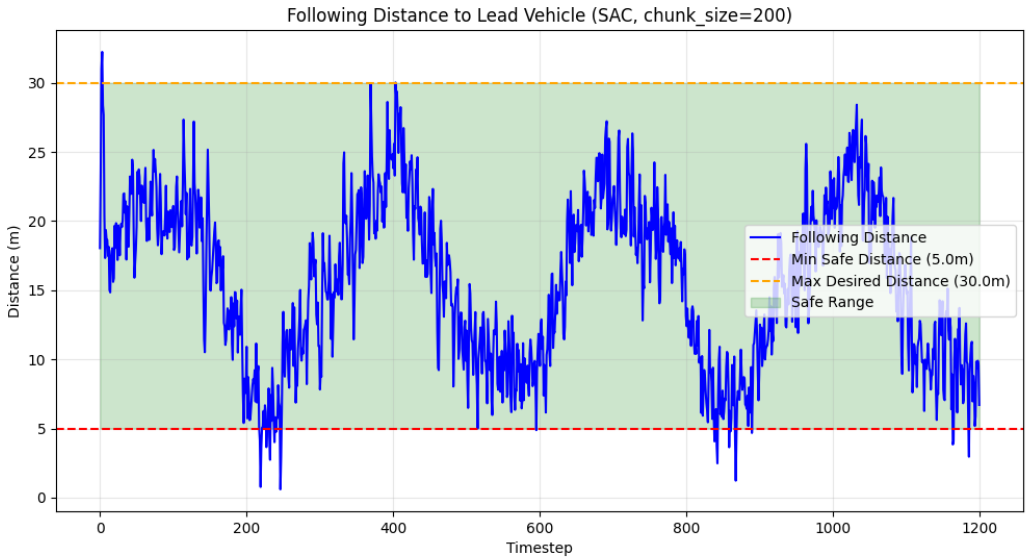

Figure 10: 200 Vehicle Distance

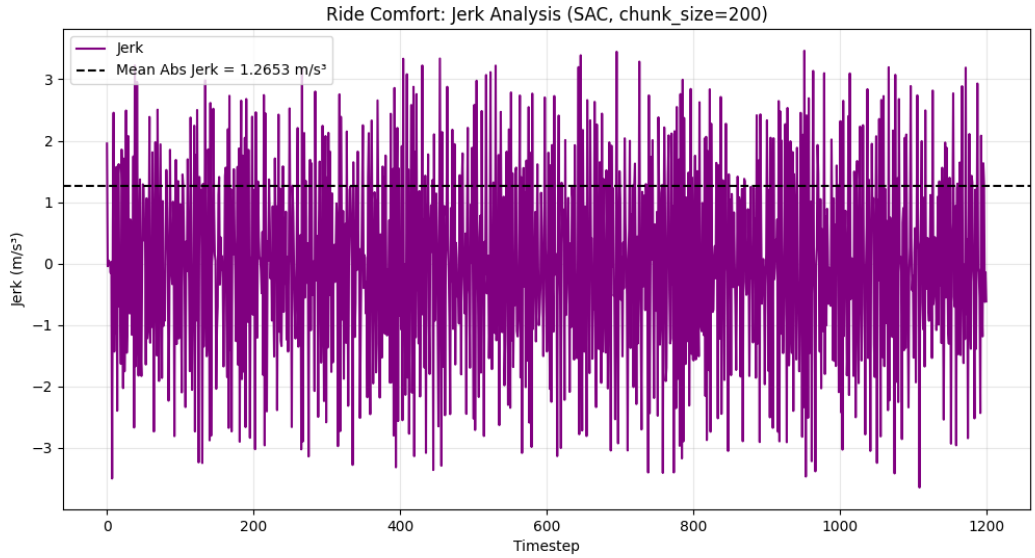

Figure 11: 200 Jerk Analysis

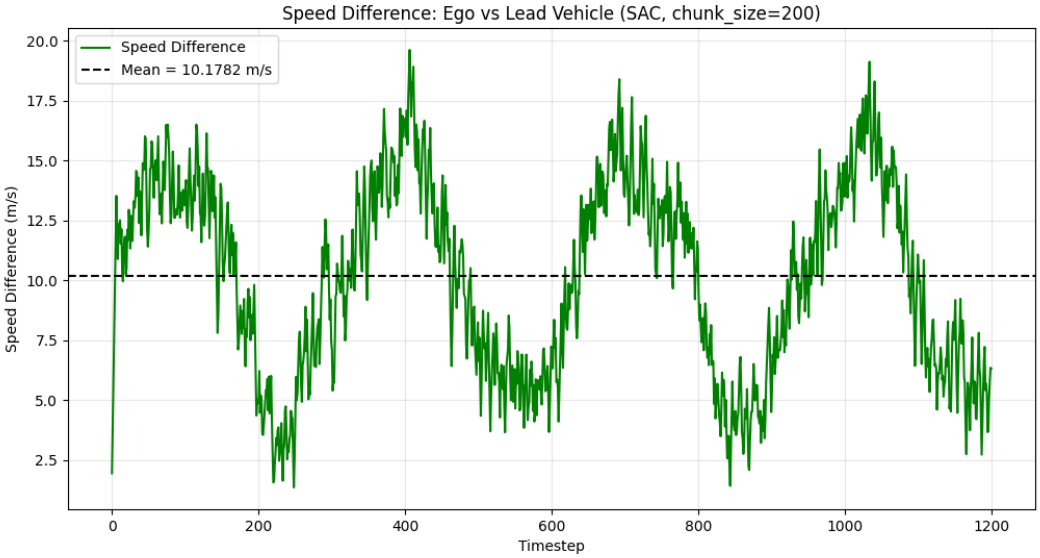

Figure 12: 200 Speed Difference

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	1.96	6.16	2.48	1.05	10.08	0.63	0.65	73.25
0.01	2.10	6.95	2.64	0.29	10.16	2.14	6.30	85.33
0.05	2.18	7.46	2.73	4.66	10.09	1.65	4.25	50.83
0.1	1.88	5.76	2.40	6.34	10.12	0.61	0.59	44.08

Table 4: Entropy Coefficient

The entropy coefficient analysis reveals that the **"auto" setting**, which dynamically adjusts exploration during training, provides the best overall performance. With an MAE_Speed of 1.961, it achieves the best speed tracking accuracy among all tested entropy coefficient values. While its distance maintenance performance (MAE_Distance of 1.050) is not the best in the group, it maintains a reasonable balance with a moderate Mean_Absolute_Jerk value of 0.633, indicating relatively smooth acceleration profiles. The Distance_In_Range_Percent of 73.25% suggests room for improvement in consistently maintaining safe following distances. The data indicates that allowing the algorithm to automatically balance exploration and exploitation based on training progress is more effective than fixed entropy values, likely because it can adapt its exploration strategy as the policy improves. This adaptive approach appears particularly beneficial for the complex, dynamic nature of speed control tasks.

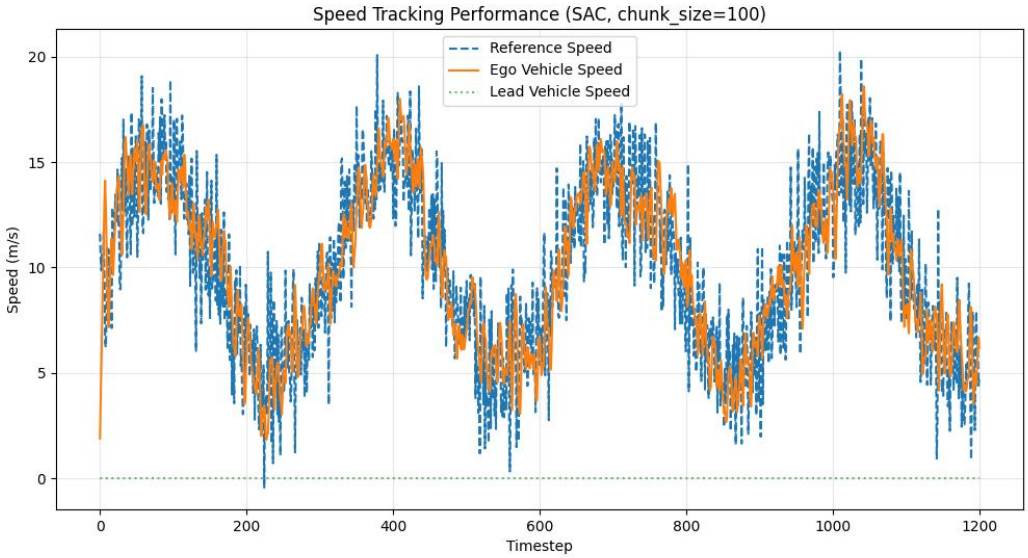

Figure 13: Auto Speed Tracking Performance

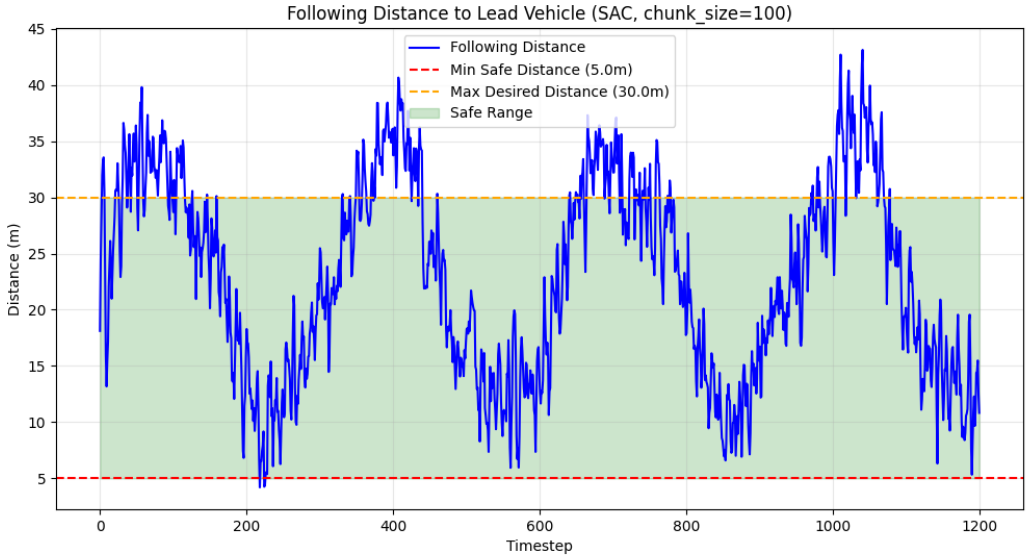

Figure 14: Auto Vehicle Distance

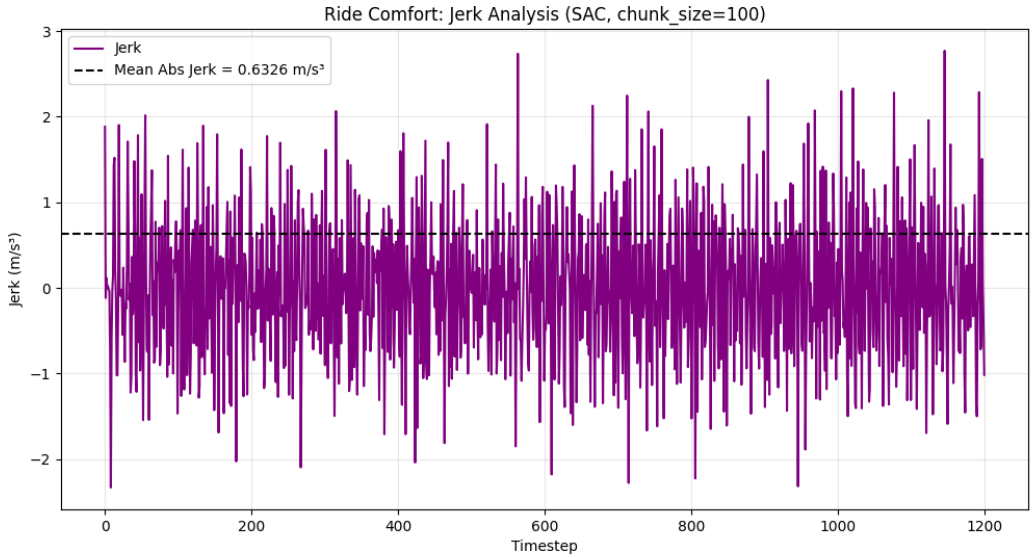

Figure 15: Auto Jerk Analysis

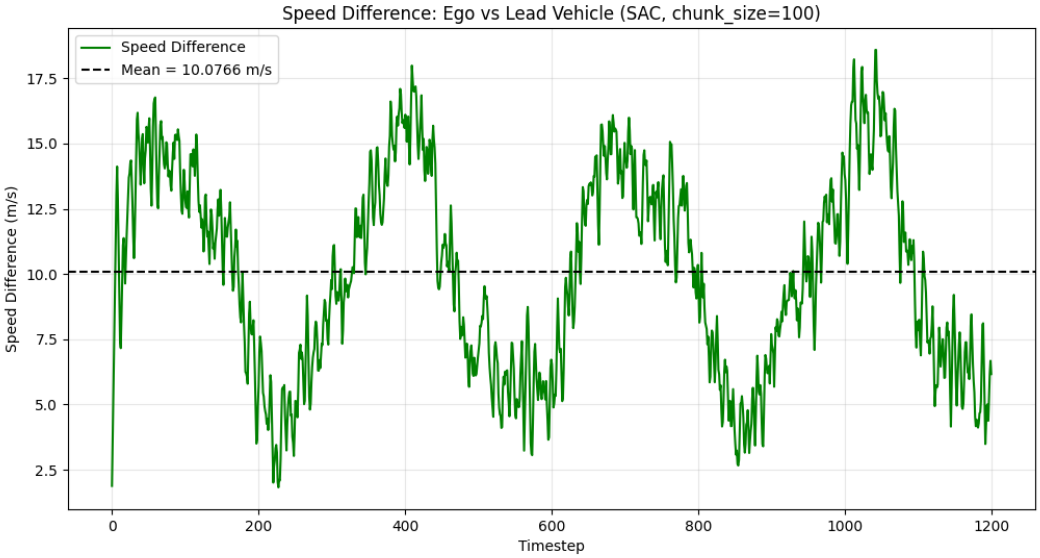

Figure 16: Auto Speed Difference

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	1.90	5.71	2.39	1.07	10.07	0.37	0.23	72.33
0.95	2.10	7.05	2.66	0.16	10.31	1.40	3.09	89.25
0.99	1.92	5.64	2.37	0.31	10.13	0.35	0.19	88.08
0.999	1.89	5.76	2.40	13.98	10.26	0.14	0.03	31.75

Table 5: Gamma

The gamma parameter experiment reveals that a **discount factor of 0.99** provides the optimal balance for the speed tracking task. With the lowest MAE_Speed value (1.915) among all tested configurations, gamma 0.99 demonstrates superior speed tracking accuracy. It also maintains reasonable distance control with an MAE_Distance of 0.308 and achieves smooth acceleration profiles with a Mean_Absolute_Jerk value of 0.352. The Distance_In_Range_Percent of 88.08% indicates good safety performance, keeping appropriate following distances for most of the simulation. This suggests that considering future rewards with a moderate discount (neither too myopic nor too focused on the distant future) is ideal for the speed control task. A gamma of 0.99 allows the agent to anticipate upcoming speed changes while still prioritizing immediate performance, creating a well-balanced control strategy that addresses both short-term responsiveness and longer-term stability.

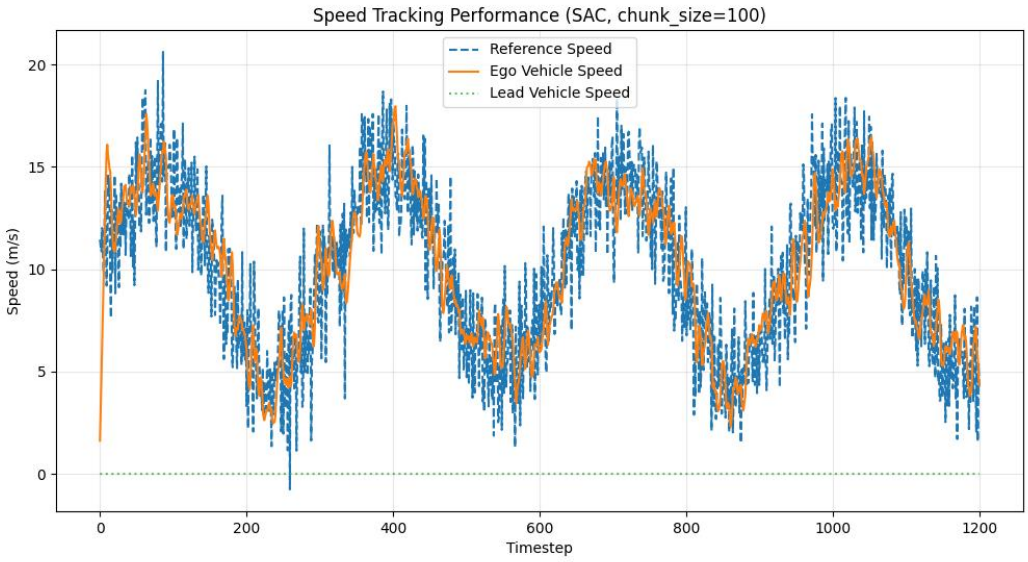

Figure 17: 0.99 Speed Tracking Performance

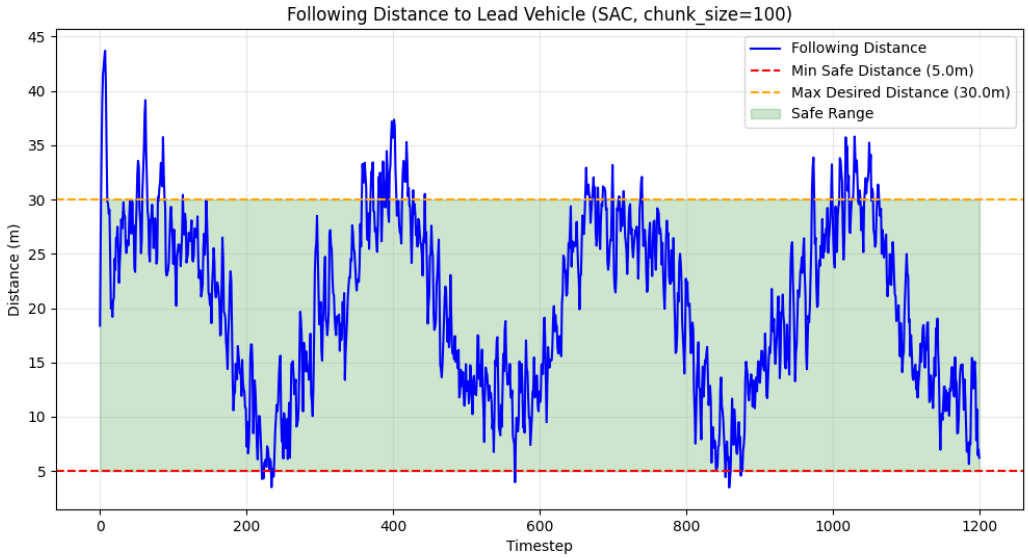

Figure 18: 0.99 Vehicle Distance

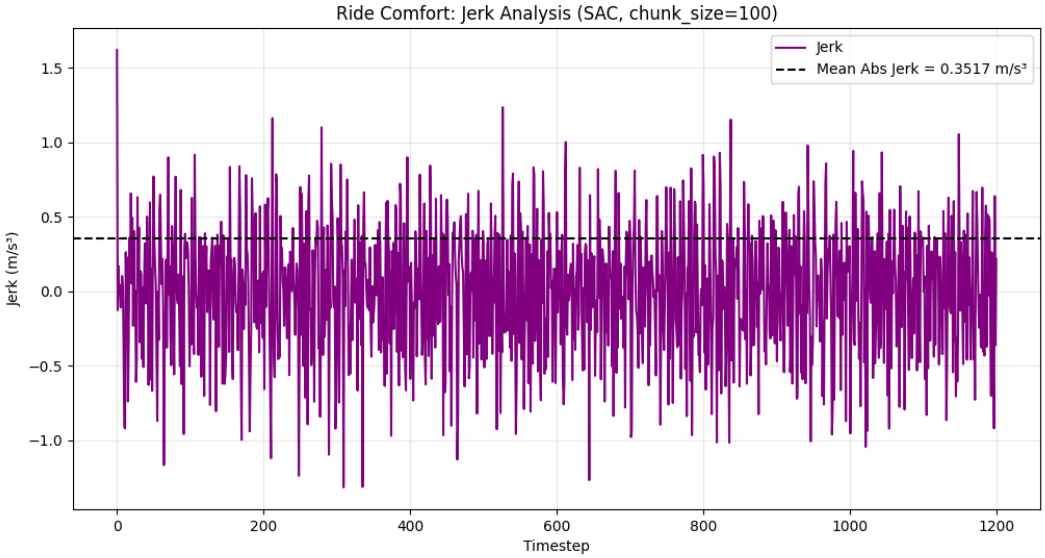

Figure 19: 0.99 Jerk Analysis

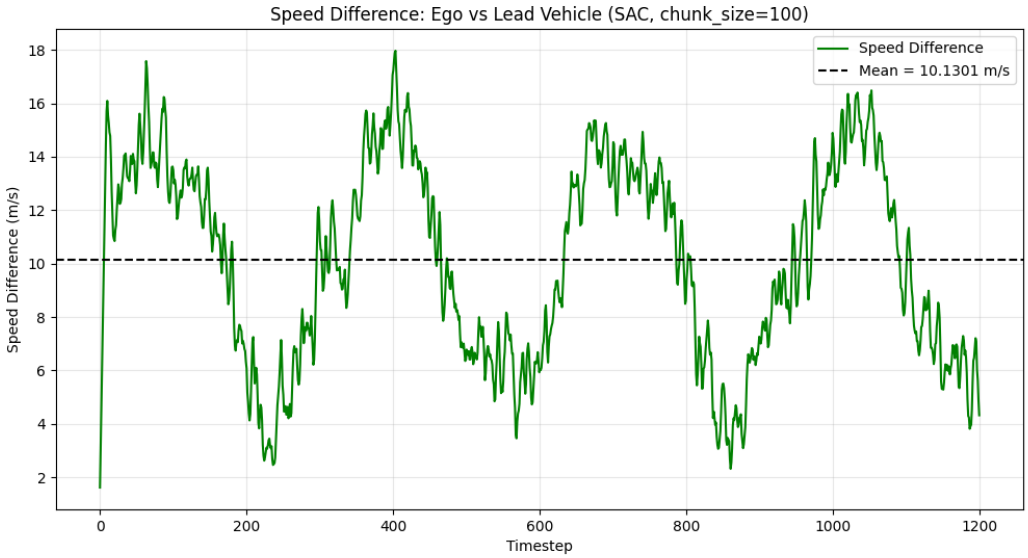

Figure 20: 0.99 Speed Difference

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	10.12	119.07	10.91	6259.55	0.00	0.00	0.00	0.08
1e-04	2.18	7.63	2.76	0.72	10.17	0.33	0.19	85.75
1e-03	2.07	6.68	2.59	0.06	10.22	1.72	4.22	95.92
1e-02	1190.99	1899143.63	1378.09	474961.87	1201.00	0.00	0.00	0.58

Table 6: Learning Rate

The learning rate analysis reveals that **0.0001 (1e-4)** provides the best overall performance for the speed tracking task. While its MAE_Speed of 2.185 is not the lowest among the tested values, it maintains good distance control with an MAE_Distance of 0.716 and achieves the second-best Mean_Absolute_Jerk value at 0.332, indicating smooth acceleration profiles that enhance passenger comfort. Its Distance_In_Range_Percent of 85.75% demonstrates good safety performance. The extreme values in the learning rate spectrum showed poor performance: the smallest learning rate (1e-5) resulted in inadequate learning with high errors, while the largest (1e-2) led to unstable learning with extremely high error values. 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 while avoiding convergence to poor local optima.

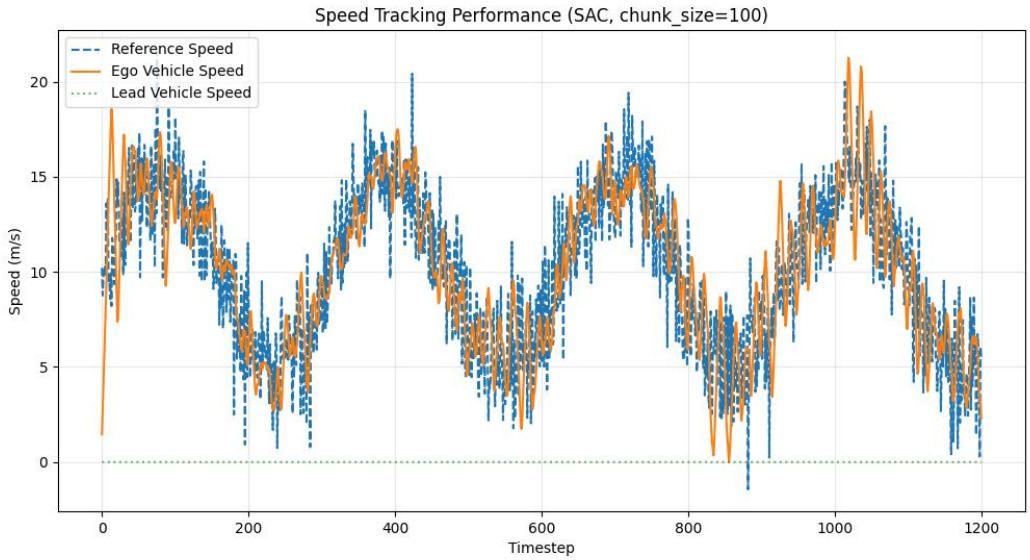

Figure 21: 0.0001 (1e-4) Speed Tracking Performance

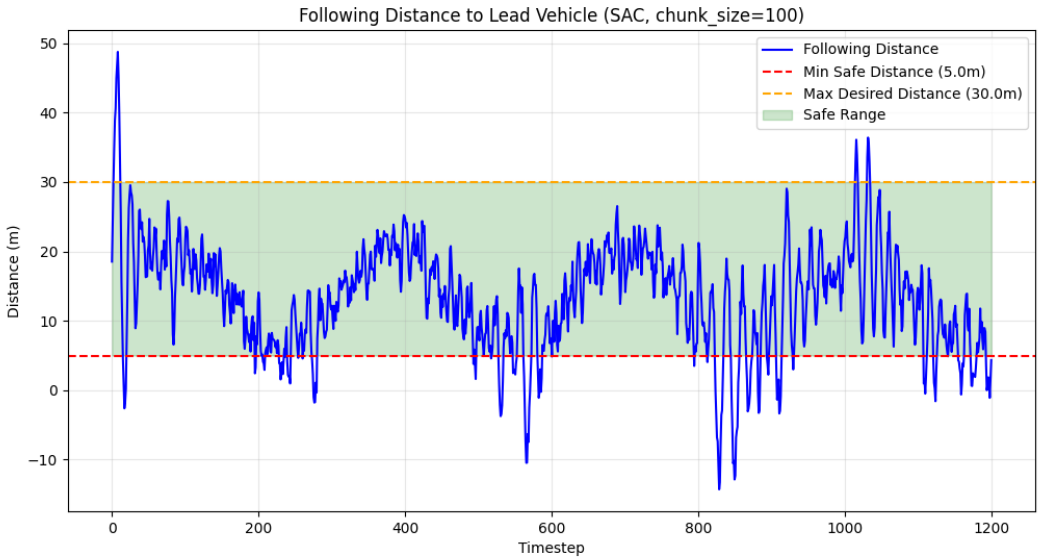

Figure 22: 0.0001 (1e-4) Vehicle Distance

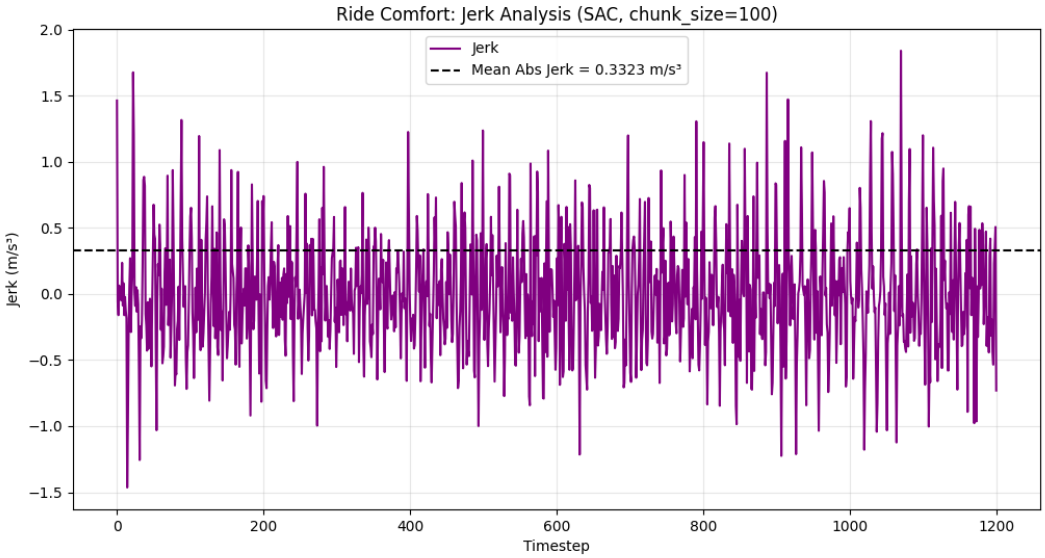

Figure 23: 0.0001 (1e-4) Jerk Analysis

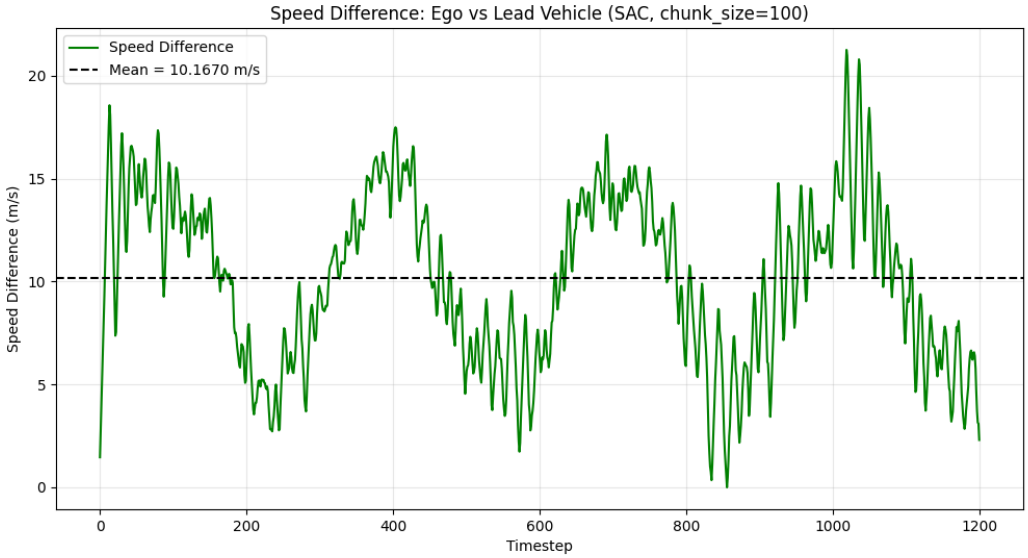

Figure 24: 0.0001 (1e-4) Speed Difference

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	16.39	607.21	24.64	2123.22	11.43	0.06	0.03	2.08
Medium	1.90	5.85	2.42	2.06	10.18	0.24	0.10	66.17
Large	2.07	6.73	2.59	0.34	10.16	0.92	1.34	89.83
Xlarge	1.93	5.81	2.41	7.03	10.26	0.51	0.40	43.25

Table 7: Network Architecture

The network architecture analysis demonstrates that the **medium (128,128)** configuration provides the best overall performance. With the second-lowest MAE_Speed value (1.905) and the best Mean_Absolute_Jerk performance (0.243), the medium architecture delivers accurate speed tracking with exceptionally smooth acceleration profiles. While its MAE_Distance value (2.060) and Distance_In_Range_Percent (66.17%) indicate room for improvement in distance maintenance, the medium architecture still outperforms the small architecture significantly. This suggests that the medium-sized network offers sufficient complexity to capture the necessary patterns in the speed control task without the potential overfitting or training difficulties that might occur with larger architectures. The balance between representational capacity and trainability makes the medium architecture particularly well-suited for this reinforcement learning application, providing good generalization while maintaining efficient training characteristics.

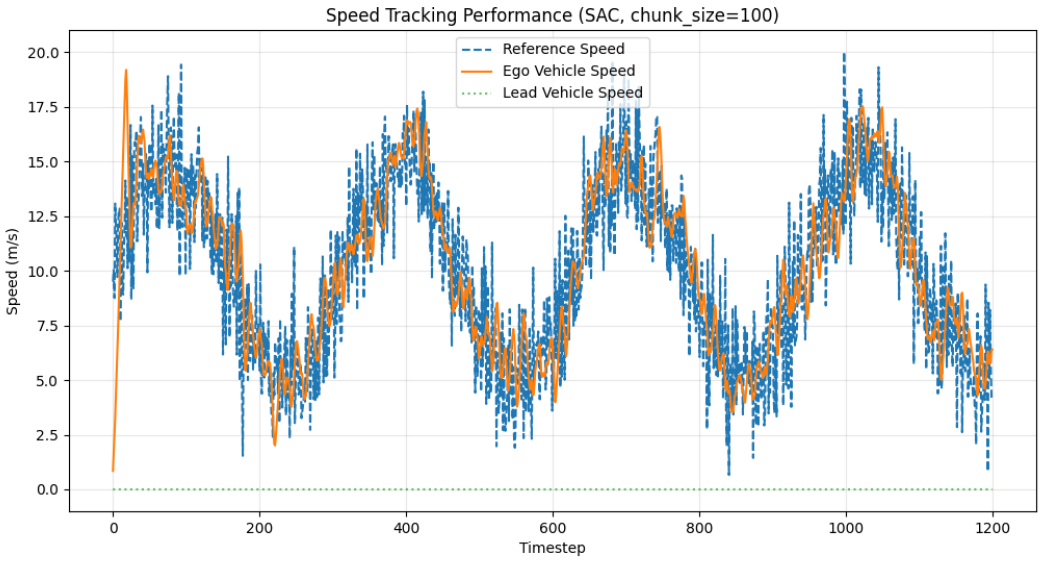

Figure 25: Medium (128,128) Speed Tracking Performance

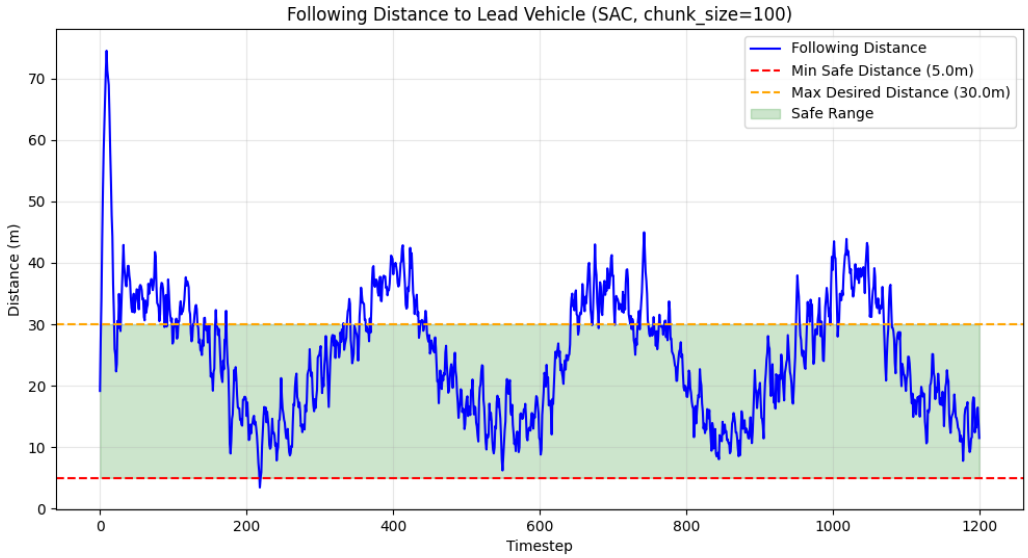

Figure 26: Medium (128,128) Vehicle Distance

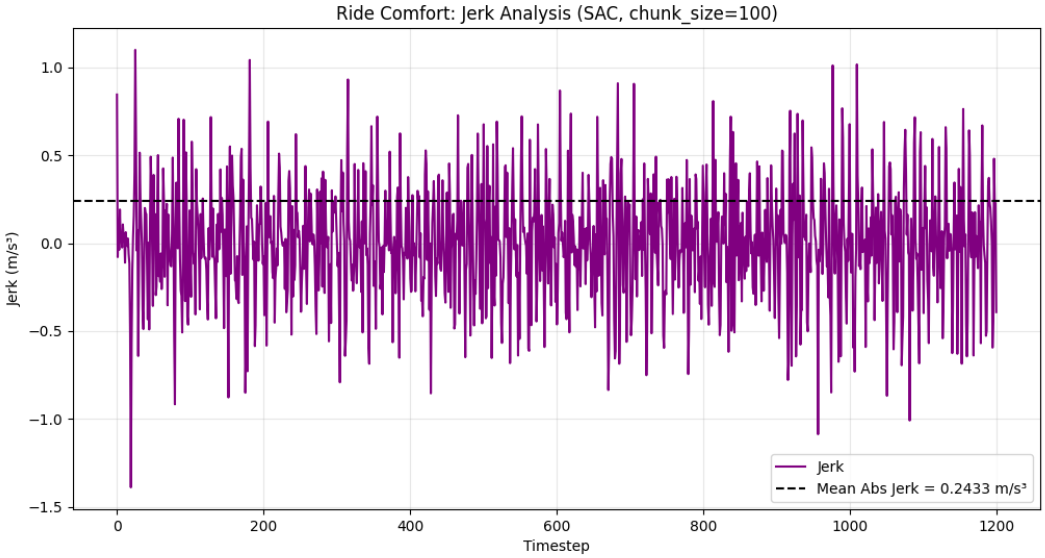

Figure 27: Medium (128,128) Jerk Analysis

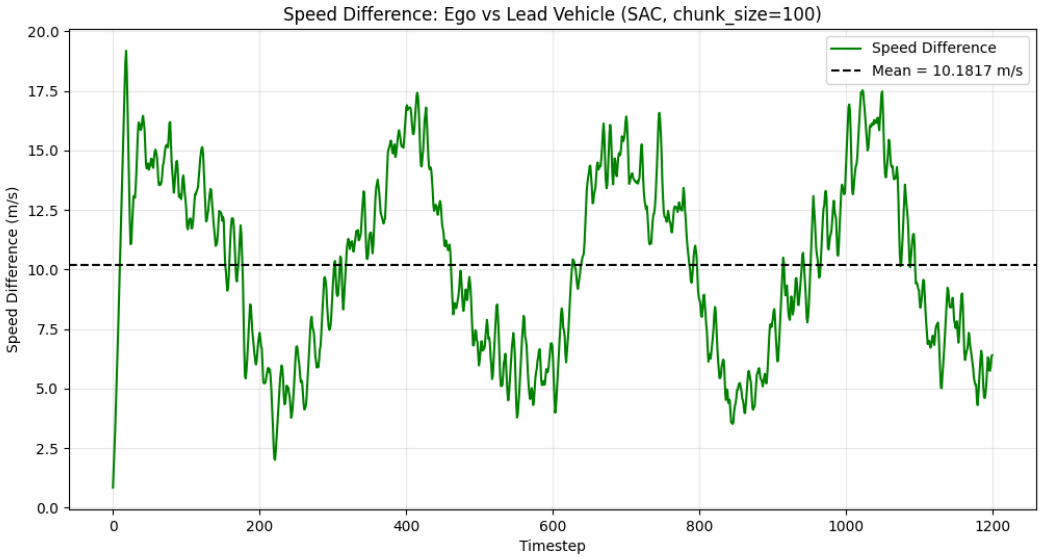

Figure 28: Medium (128,128) Speed Difference

Conclusion

Parameter	*Algorithm*	*Batch Size*	*Chunk Size*	*Entropy Coeff.*	*Gamma*	*Learn. Rate*	*Network Arch.*
Best Value	SAC	128	200	Auto	0.99	0.0001	Medium (128, 128)

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.

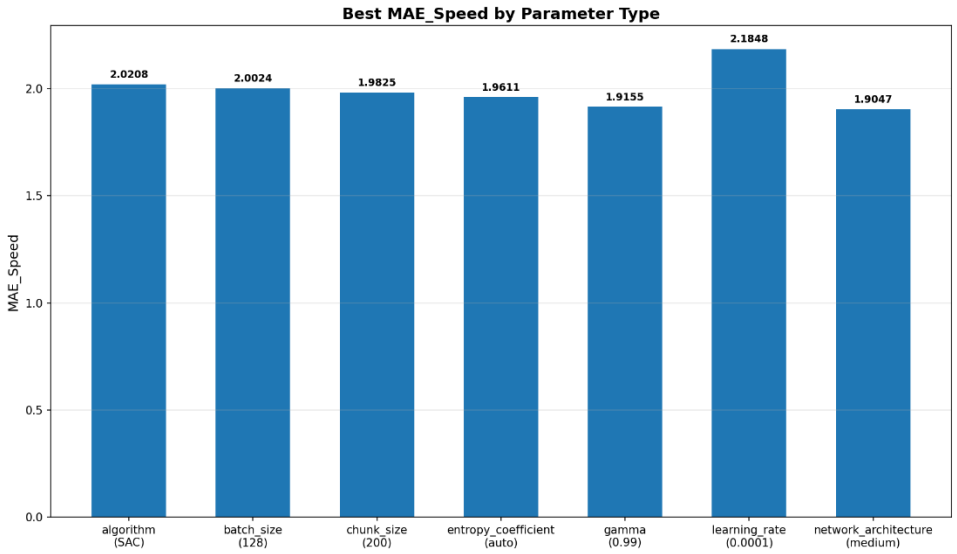

Figure 29: Hyperparameter MAE_Speed Results

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.

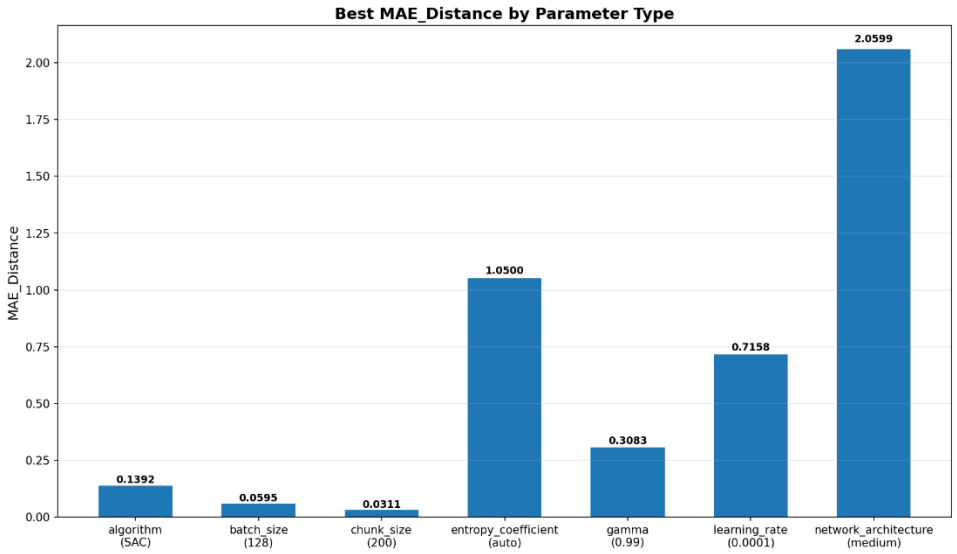

Figure 30: Hyperparameter MAE_Distance Results

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.

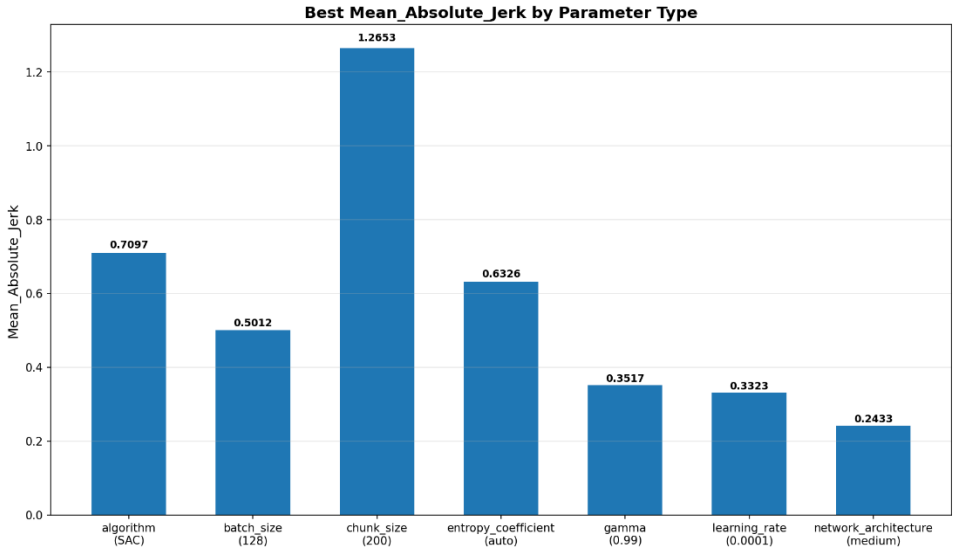

Figure 31: Hyperparameter Mean_Absolute_Jerk Results

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.

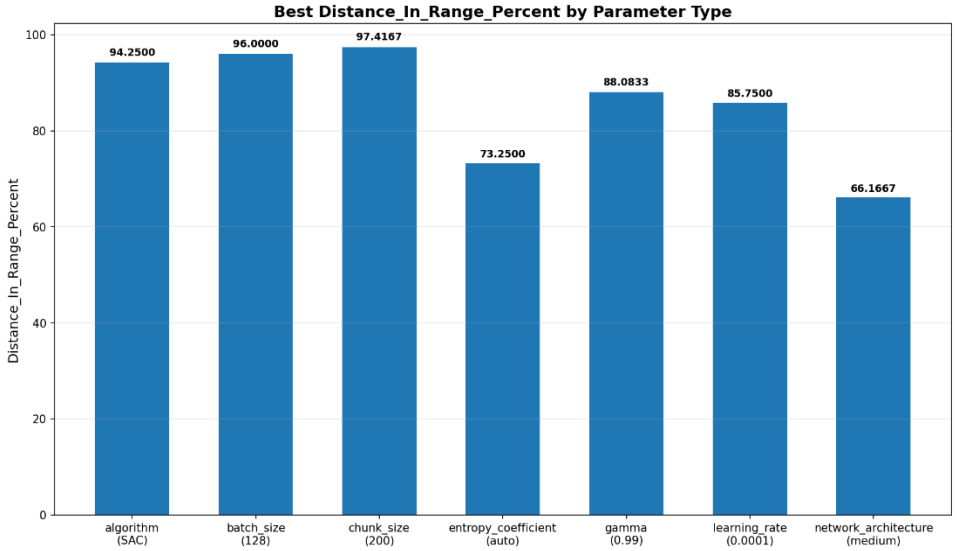

Figure 32: Hyperparameter Distance_In_Range_Percent Results

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.

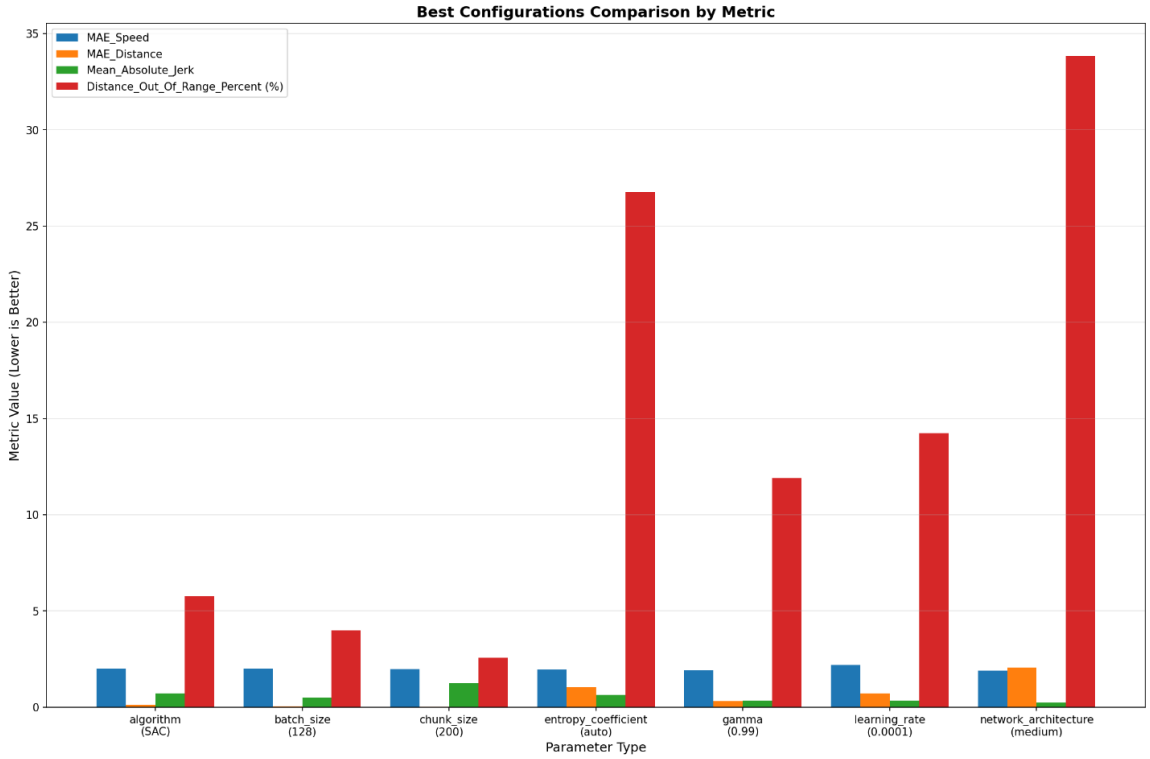

Figure 33: Hyperparameter Results Summary