TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A

PROJECT REPORT

ON

LATENT WORLDS: A LATENT SPACE APPROACH TO AUTONOMOUS DRIVING TRAINING

SUBMITTED BY:

DEEPAK THAPA (PUL078BEI014)

MANISH ADHIKARI (PUL078BEI019)

SUBMITTED TO:

DEPARTMENT OF ELECTRONICS & COMPUTER ENGINEERING

April, 2025

Page of Approval

TRIBHUVAN UNIVERSIY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **"LATENT WORLDS: A LATENT SPACE APPROACH TO AUTONOMOUS DRIVING TRAINING"** submitted by **Deepak Thapa**, **Manish Adhikari** in partial fulfillment of the requirements for the Bachelor's degree in Electronics & Computer Engineering.

.....
External examiner
Department of Electronics and Computer Engineering,
Pulchowk Campus, IOE, TU.

Date of approval:

Copyright

Acknowledgments

We would like to express our sincere gratitude to the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering, for their continuous support and guidance throughout the course of this project. We are particularly grateful to the Head of the Department, Deputy Head, and the members of the Project Management Team, including IC Chair **Asst. Prof. Anku Jaiswal** and **Asst. Prof. Anila Kansakar**, for their valuable insights and encouragement.

Our heartfelt thanks go to our respected seniors, whose advice and experience have been invaluable in shaping the direction of this project. We also appreciate the support of our friends, whose encouragement and contributions have been instrumental in our progress.

We extend our deepest gratitude to our family members for their constant motivation and support, which have inspired us throughout this project. Lastly, we acknowledge the contributions of everyone who has supported us during the course of this project.

Abstract

This project implements a world model architecture for autonomous driving in simulated environments. We used a three-component system consisting of a Variational Autoencoder (VAE) for observation encoding, a Mixture Density Network with Recurrent Neural Network (MDN-RNN) for predicting future states, and policy networks trained using Deep Q-Networks (DQN) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES). Experiments in OpenAI Gymnasium's CarRacing-v3 and the CARLA simulator demonstrated that policies trained with world model architecture learn significantly faster and achieve higher rewards than those trained through direct interaction. The VAE effectively compressed high-dimensional visual inputs into meaningful latent representations, while the MDN-RNN successfully predicted future states. Our findings confirm that world models provide an efficient framework for developing autonomous driving policies, reducing simulation requirements while maintaining effective performance.

Keywords: *Model-Based Reinforcement Learning, Latent Environments, Generative Models, Self-Driving Cars, Simulation Training*

Contents

Page of Approval **ii**

Copyright **iii**

Acknowledgements **iv**

Abstract **v**

Contents **vii**

List of Figures **viii**

List of Abbreviations **ix**

1 Introduction **1**

1.1 Background 1

1.2 Problem Statement 1

1.3 Objectives 1

1.4 Scope 2

2 Literature Review **3**

2.1 Related work 3

2.2 Related Theory 4

2.2.1 World Model Architecture 4

2.2.2 Reinforcement Learning Algorithms 6

2.2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES) 8

3 Methodology **11**

3.1 Feasibility Study 11

3.2 Reward System 11

3.3 Data Collection 12

3.4 System Design 13

4 Results & Discussion **15**
4.1 Results 15
4.1.1 Experiments in CarRacing-v3 15
4.1.2 V Model Encoding Performance 17
4.1.3 MDN-RNN Training Results 19
4.1.4 Policy Training Results 19
4.1.5 Algorithm Comparison in CarRacing-v3 21

5 Discussion **22**

6 Limitations and Future Enhancements **23**
6.1 Limitations 23
6.2 Attempts at Improvement 23
6.3 Future Enhancements 23

7 Conclusions **25**

List of Figures

2.1 VMC architecture of world model 4
2.2 Flow diagram of Variational Autoencoder 5
2.3 RNN with a Mixture Density Network output 6
2.4 Structure of DQN 7
2.5 Optimization run with covariance matrix adaptation 9

3.1 System architecture 14

4.1 Comparison of policy learning with (left) and without (right) the V model in CarRacing-v3. 15
4.2 Bar graph comparing the average rewards of policies trained with and without the V model in CarRacing-v3. 16
4.3 Reward vs. time graph for policies trained with and without the V model in CarRacing-v3, showing faster convergence and higher rewards with the V model. 16
4.4 VAE encoded representation of CarRacing-v3. 17
4.5 VAE encoded representation of CARLA. 17
4.6 VAE encoded representation of highway-env. 17
4.7 Total VAE loss in CARLA. 18
4.8 Reconstruction loss in CARLA. 18
4.9 KL divergence loss in CARLA. 18
4.10 Training loss of VAE in highway-env. 18
4.11 Training loss of MDN-RNN. 19
4.12 Generation vs. reward for CarRacing-v3 using CMA-ES. 19
4.13 Epoch vs. reward for CarRacing-v3 using DQN. 20
4.14 Generation vs. reward for CARLA using CMA-ES. 20
4.15 Episode vs. reward for highway-env using DQN. 21
4.16 Comparison of max reward in CarRacing-v3 using DQN and CMA-ES. . . . 21

6.1 Possible improved architecture. 24

List of Abbreviations

AI	Artificial Intelligence
AV	Autonomous Vehicles
RL	Reinforcement Learning
MDP	Markov Decision Process
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
A2C	Advantage Actor Critic
VAE	Variational Autoencoder
MDN	Mixture Density Network
RNN	Recurrent Neural Network
FOV	Field Of Vision
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
BEV	Bird's Eye View
VMC	Vision-Memory-Controller
SOTA	State-Of-The-Art
LWM	Latent World Model
CARLA	Car Learning to Act

1. Introduction

1.1 Background

Artificial intelligence has long been used to develop systems that perform human-like tasks. Imitation learning, which trains models to mimic expert behavior, was an early approach in many domains. However, research has shown that imitation learning is inherently limited—it can only replicate human performance, not surpass it.

The breakthrough came with reinforcement learning (RL), where agents learn through trial and error, often exceeding human capabilities. Notable successes, such as DeepMind's AlphaGo [1] and StarCraft II agents[2], demonstrated RL's potential to outperform even the best human players.

A similar trend is emerging in autonomous vehicle (AV) research. While early AV systems relied on imitation learning, researchers are now turning to RL-based methods to handle complex driving scenarios. However, training RL agents directly in high-dimensional environments is computationally expensive. Recent advancements in deep RL, particularly learning latent representations and dynamics, have introduced a more efficient way to train agents—by operating in a compressed latent space instead of raw sensory inputs. This approach holds promise for making RL-based AV training more scalable and effective.

1.2 Problem Statement

Applying standard RL algorithms like DQN [3] to AV is impractical as it operates only in discrete action spaces. Defining discrete actions explicitly can limit the agent's control flexibility. While PPO [4] and A2C [5] support continuous action spaces and generalize better than DQN, PPO is computationally expensive, and A2C is highly sensitive to hyperparameter tuning, making training difficult.

There is a need for a simple yet effective architecture that enables fast generalization with lower computational resources.

1.3 Objectives

- Design a simple architecture using world models to learn policies for environments.
- Evaluate the architecture for autonomous driving.

1.4 Scope

This project aims to evaluate the effectiveness of world models in autonomous vehicle training. It includes an analysis of world models combined with different reinforcement learning algorithms to understand their impact on training efficiency and generalization.

2. Literature Review

2.1 Related work

End-to-end learning for autonomous driving has been widely explored, primarily using imitation learning. Imitation learning approaches, such as Conditional Imitation Learning (CIL) [6], train a policy by mimicking expert demonstrations. However, these methods struggle in unseen scenarios due to their dependence on expert trajectories. Variational End-to-End Navigation and Localization [7] improves generalization by learning a latent representation of navigation but still relies on imitation signals. End-to-End Differentiable Adversarial Imitation Learning [8] introduces adversarial training to improve robustness, but it remains sensitive to distributional shifts. Query-Efficient Imitation Learning [9] minimizes the number of expert queries, yet it still inherits the limitations of imitation-based approaches, particularly in long-horizon tasks.

Pure reinforcement learning (RL) methods address the limitations of imitation learning by training policies from scratch through trial and error. Techniques like Efficient Reinforcement Learning for Autonomous Driving [10] and Confidence-Aware Reinforcement Learning [11] attempt to improve sample efficiency and robustness. However, these methods require a large number of interactions with the environment, making them computationally expensive. Other works, such as Tackling Real-World Autonomous Driving using Deep Reinforcement Learning [12] and Improving the Performance of Autonomous Driving through Deep Reinforcement Learning [13], demonstrate the feasibility of RL-based approaches but highlight the need for better sample efficiency.

To address the computational challenges of reinforcement learning, world models have been proposed as an efficient alternative. The World Models framework [14] learns a latent representation of the environment using a Variational Autoencoder (VAE) and models its dynamics with a recurrent neural network. DreamerV1 [15] extends this by enabling gradient-based policy learning in the latent space, reducing sample complexity. DreamerV2 [16] further stabilizes training with improved latent dynamics learning, while DreamerV3 [17] introduces architectural improvements for better scalability.

Applying world models to autonomous driving has led to promising results. LAW: Enhancing End-to-End Autonomous Driving with Latent World Models [18] leverages a learned latent space for efficient policy training. Think2Drive [19] explores reinforcement learning within a latent world model, allowing for quasi-realistic autonomous driving. Multiview

Visual Forecasting and Planning with World Models [20] improves future prediction capabilities for planning, while BEVWorld [21] unifies multimodal inputs in a BEV latent space for robust autonomous control.

2.2 Related Theory

2.2.1 World Model Architecture

The world model consists of three main components: **Vision (V), Memory (M), and Controller (C)**, each serving a distinct role in learning and decision-making.

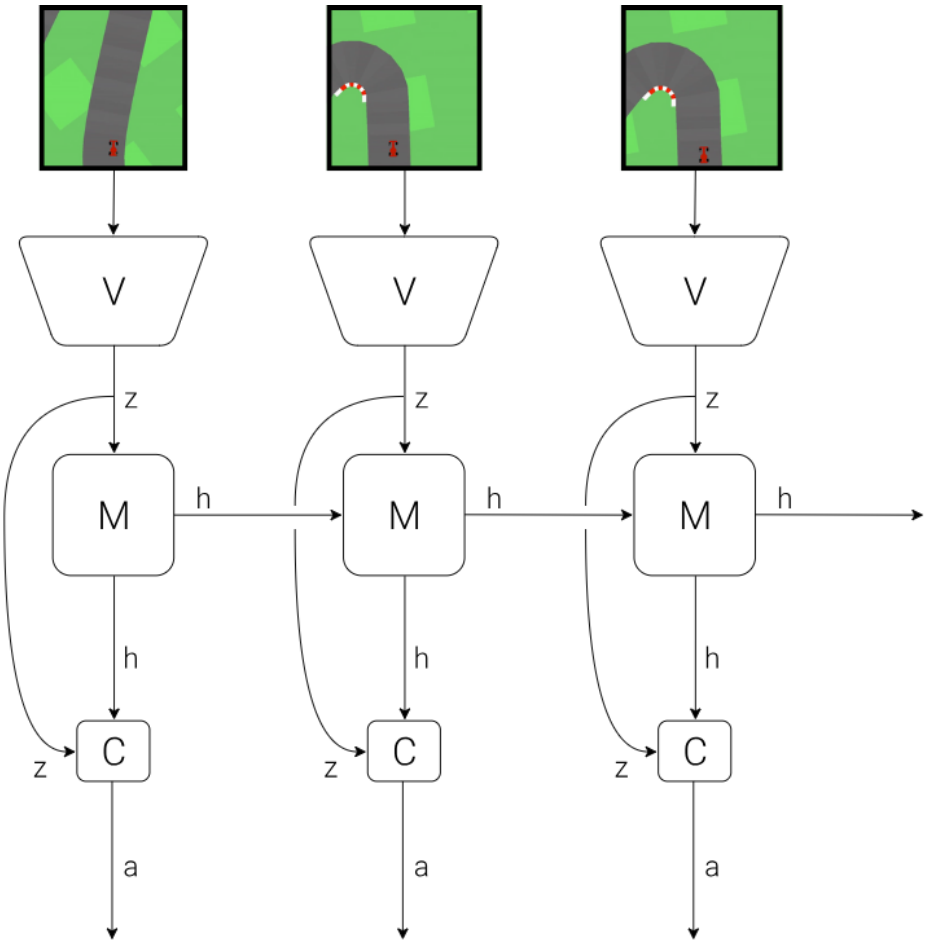

Figure 2.1: VMC architecture of world model

Vision (V): The visual sensory component compresses raw observations into a lower-dimensional, informative representation. This helps in reducing complexity while preserving essential details required for decision-making.

Memory (M): The memory component models temporal dependencies by predicting future states based on past observations. It enables the system to anticipate and reason

about future events, improving long-term planning.

Controller (C): The controller is responsible for decision-making. It utilizes the representations generated by the vision and memory components to determine optimal actions, allowing the agent to interact effectively with the environment.

Variational Autoencoder (V) Model

The environment provides our agent with a high dimensional input observation at each time step. This input is a 2D top view image frame that is part of a video sequence. The role of the V model is to learn an abstract, compressed representation of each observed input frame.

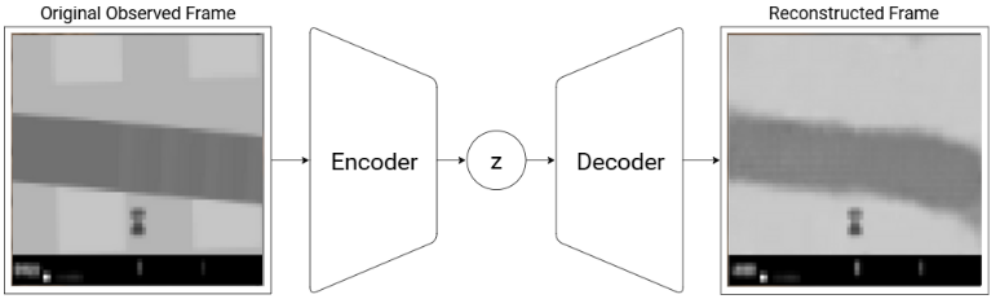

Figure 2.2: Flow diagram of Variational Autoencoder

We use a Variational Autoencoder (VAE) [22] as the V model in our experiment.

MDN-RNN (M) Model

The role of the V model is to compress visual information from each time frame. However, capturing temporal dependencies is equally important. The M model addresses this by predicting the future latent representations produced by the V model. Since many real-world environments exhibit stochastic behavior, the M model does not generate deterministic predictions but instead models a probability distribution over possible future latent states.

A Recurrent Neural Network (RNN) with a Mixture Density Network (MDN) output layer is used for this purpose [23, 24].. The MDN outputs the parameters of a Gaussian mixture distribution, from which the next latent vector is sampled. In this approach, the probability distribution $p(z)$ is approximated as a mixture of Gaussians, and the RNN is trained to output the probability distribution of the next latent vector z_{t+1}, given the current and past information.

More specifically, the RNN models:

$$P(z_{t+1} \mid a_t, z_t, h_t)$$

where a_t represents the action taken at time t, and h_t denotes the hidden state of the RNN at time t.

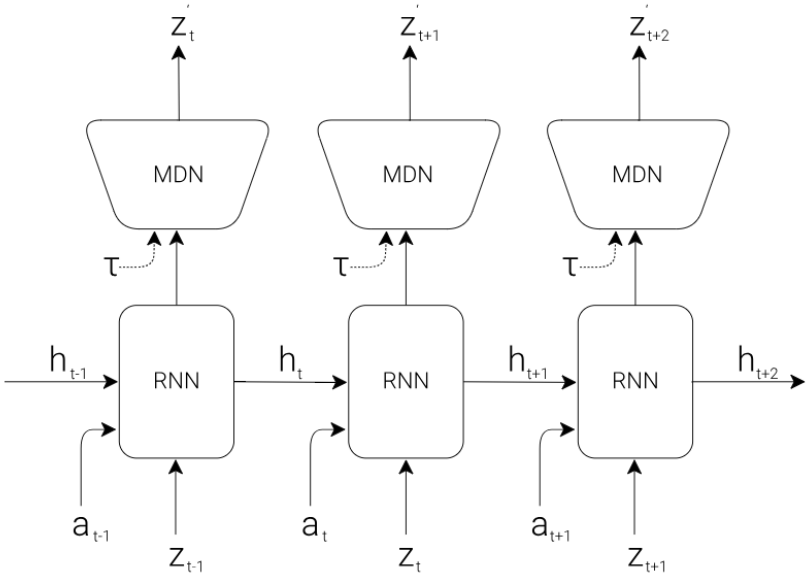

Figure 2.3: RNN with a Mixture Density Network output

Controller

The controller C is a simple single-layer linear model that maps the latent state z_t and the hidden state h_t directly to the action a_t at each time step:

$$a_t = W_c[z_t, h_t] + b_c$$

In this linear model, W_c and b_c are the weight matrix and bias vector that map the concatenated input vector $[z_t, h_t]$ to the output action vector a_t.

2.2.2 Reinforcement Learning Algorithms

Several reinforcement learning methods can be used in training agents within the world model framework.

Deep Q-Network (DQN)

Traditional Q-Learning [25] works well for environments with a small and finite number of states, but it struggles with large or continuous state spaces due to the size of the Q-table.

Deep Q-Learning or Deep Q-Network (DQN) is an extension of the basic Q-Learning algorithm, which uses deep neural networks to approximate the Q-values. It overcomes the limitation of traditional Q-Learning by replacing the Q-table with a neural network that can generalize across state-action pairs.

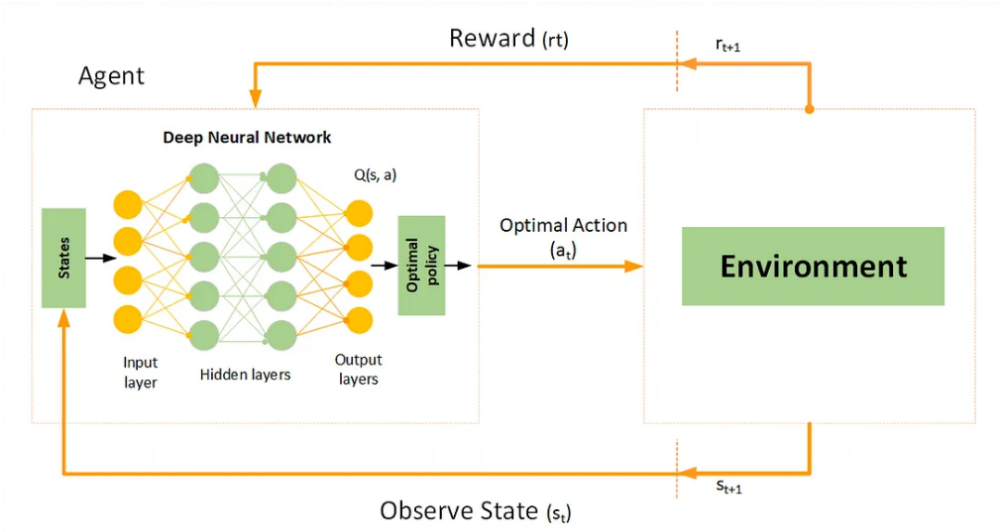

Figure 2.4: Structure of DQN

To stabilize training, DQN utilizes *experience replay*, which stores past experiences and samples a batch of them for training. This process breaks the correlation between consecutive experiences and improves sample efficiency.

The Q-learning update rule in DQN is given by:

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a')$$

where: - s and a are the current state and action, - s' is the next state, - r is the reward received, - γ is the discount factor.

The loss function used to train the Q-network is:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

where θ represents the parameters of the Q-network, and θ^- represents the parameters of a target network that is updated periodically.

2.2.3 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

Evolution Strategies (ES) [26] follow a general approach:

- Generate λ individuals as a population by sampling from a multivariate normal distribution:

$$x_k \sim \mathcal{N}(m, \Sigma) \tag{2.1}$$

  where m is the mean vector and Σ is the covariance matrix.

- Sort the population based on fitness.
- Select μ individuals with the highest fitness as parents.
- Update the mean vector m to be the average of the selected parents.
- Repeat the process iteratively.

Individuals are generated from elite parents using normal distributions. This can be interpreted as modeling the "distribution of minds" of the parents, meaning that sampling from this distribution provides likely actions for a given state.

Probability Density Function

The probability density function (PDF) of a univariate normal distribution is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.2}$$

For complex policies, we need a multivariate normal distribution:

$$f(x) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(x-m)^T \Sigma^{-1}(x-m)\right) \tag{2.3}$$

Here, the data points and means are vectors because they represent multiple correlated variables instead of a single scalar value. The covariance matrix Σ replaces the scalar variance to account for dependencies between different variables.

Adapting the Covariance Matrix

In simple ES, keeping covariance matrix constant results in a static search of distribution. However, the optimal search distribution should be an ellipse like shape biased towards promising directions. Hence, CMA-ES adapt covariance matrix over time.

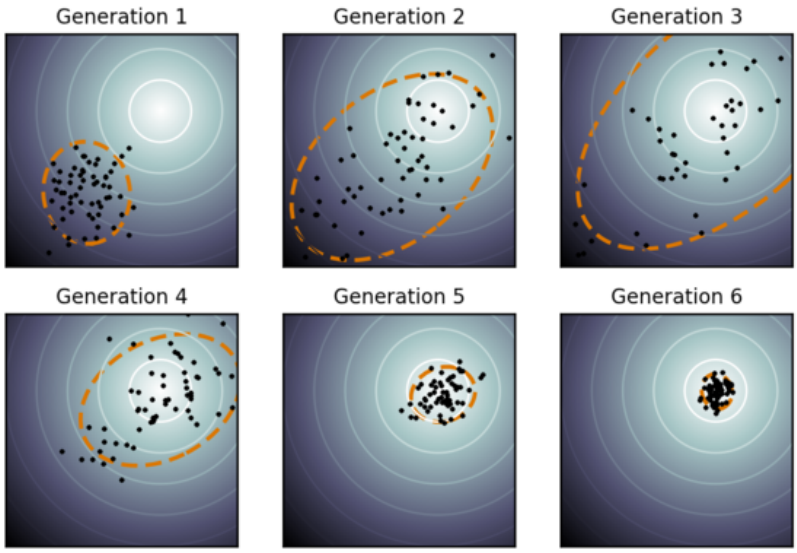

Figure 2.5: Optimization run with covariance matrix adaptation

The covariance matrix adaptation rule is:

$$m^{(g+1)} = \frac{1}{\mu} \sum_{i=1}^{\mu} x_i \tag{2.4}$$

$$\Sigma^{(g+1)} = \frac{1}{\mu} \sum_{i=1}^{\mu} (x_i - m^{(g)})(x_i - m^{(g)})^T \tag{2.5}$$

where $m^{(g)}$ is the mean of the previous generation.

Algorithm 1 CMA-ES (Covariance Matrix Adaptation Evolution Strategy)

Initialize mean $\mathbf{m}$ and covariance matrix $\mathbf{\Sigma}$
while not converged **do**
    for $k = 1$ to λ **do**
        Sample individuals: $\mathbf{x}_k \sim \mathcal{N}(\mathbf{m}, \mathbf{\Sigma})$
    end for
    Sort all $\mathbf{x}_k$ by fitness
    Update covariance matrix:

$$\mathbf{\Sigma} \leftarrow \frac{1}{\mu} \sum_{i=0}^{\mu} (\mathbf{x}_i - \mathbf{m})(\mathbf{x}_i - \mathbf{m})^T$$

    Update mean:

$$\mathbf{m} \leftarrow \frac{1}{\mu} \sum_{i=0}^{\mu} \mathbf{x}_i$$

end while

3. Methodology

3.1 Feasibility Study

The feasibility of using a world model approach was evaluated based on selected environments:

- **CarRacing-v3** [27]: A simple 2D car racing environment, making it suitable for rapid experimentation.
- **Carla Simulator** [28]: A stable and efficient autonomous driving simulator, ensuring feasibility for world model training.
- **Highway-env** [29]: A lightweight environment focused on highway driving scenarios, ideal for testing lane changes, overtaking, and multi-agent interactions.

3.2 Reward System

Each environment employs a distinct reward structure that shapes the agent's learning behavior. Below, we formally define the reward functions for each environment:

- **CarRacing-v3**: CarRacing-v3 is a continuous control environment where the agent navigates a procedurally generated 2D track composed of discrete square tiles. The agent earns rewards for visiting new tiles and incurs a penalty for time spent.

  The total reward for an episode is:

  $$R = \left(\frac{1000}{N}\right) \cdot T - 0.1 \cdot F$$

  where:

  - N: Total number of tiles in the track.
  - T: Number of unique tiles visited during the episode.
  - F: Total number of frames (timesteps) taken.

  The term $\frac{1000}{N}$ scales the reward such that visiting all tiles yields a maximum of approximately 1000. The penalty $-0.1 \cdot F$ encourages the agent to complete the track efficiently, discouraging idle or overly cautious behavior.

- **Carla Simulator**: The Carla simulator lacks a default reward function, so we designed a custom reward to promote safe and efficient driving:

$$R_t = w_v \cdot v_t - w_c \cdot \mathbb{I}\{C_t\} - w_o \cdot \mathbb{I}\{O_t\}$$

where:

  - v_t: Vehicle speed at time t (in m/s).
  - $\mathbb{I}\{C_t\}$: Indicator function, equal to 1 if a collision (with obstacles or other vehicles) occurs at time t, otherwise 0.
  - $\mathbb{I}\{O_t\}$: Indicator function, equal to 1 if the vehicle deviates from the designated lane at time t, otherwise 0.
  - w_v, w_c, w_o: Weights to balance speed, collision avoidance, and lane discipline.

  This reward encourages high speed while imposing significant penalties for collisions and lane departures.

- **Highway-env**: Highway-env is a discrete multi-agent driving simulator focused on highway scenarios. Its reward function promotes high-speed, collision-free driving with a preference for the rightmost lane:

$$R_t = \alpha \cdot v_t - \beta \cdot \mathbb{I}\{C_t\} + \gamma \cdot l_t$$

where:

  - v_t: Normalized vehicle speed at time t (scaled to [0, 1]).
  - $\mathbb{I}\{C_t\}$: Indicator function, equal to 1 if a collision occurs at time t, otherwise 0.
  - l_t: Lane reward, positive for staying in the rightmost lane and reduced (or negative) for other lanes or lane changes.
  - α, β, γ: Tunable coefficients to balance speed, safety, and lane preference.

  This reward aligns with real-world driving norms, prioritizing speed, collision avoidance, and adherence to lane discipline.

3.3 Data Collection

Data were collected from the simulators in the form of *state-action-reward-next state-done* tuples:

$$(s_t, a_t, r_t, s_{t+1}, d_t) \tag{3.1}$$

where:

- s_t is the current state (bird's eye view observation).
- a_t is the action taken at time t.
- r_t is the reward provided by the environment.
- s_{t+1} is the next state after taking a_t.
- d_t is a boolean indicating if the episode has ended.

3.4 System Design

First, the system learns the environment's representation using the extbfV model, then models environment dynamics using the extbfM model, and finally, trains/tests the policy using different algorithms like CMA-ES or DQN.

At each time step t, the system follows these steps:

1. The raw observation o_t is encoded into a latent representation by the extbfV model:

$$z_t \sim p(z_t|o_t) \tag{3.2}$$

2. The extbfcontroller takes the latent representation z_t and the hidden state h_t from the extbfM model to produce an action:

$$a_t = C(z_t, h_t) \tag{3.3}$$

3. The environment executes the action a_t, returning reward r_t and next observation o_{t+1}. The extbfM model updates its hidden state:

$$h_{t+1} = M(h_t, z_t, a_t) \tag{3.4}$$

4. The new observation o_{t+1} is encoded into z_{t+1}, and the process repeats.

This interaction between components enables the system to learn a compact environment representation, predict future dynamics, and make decisions efficiently.

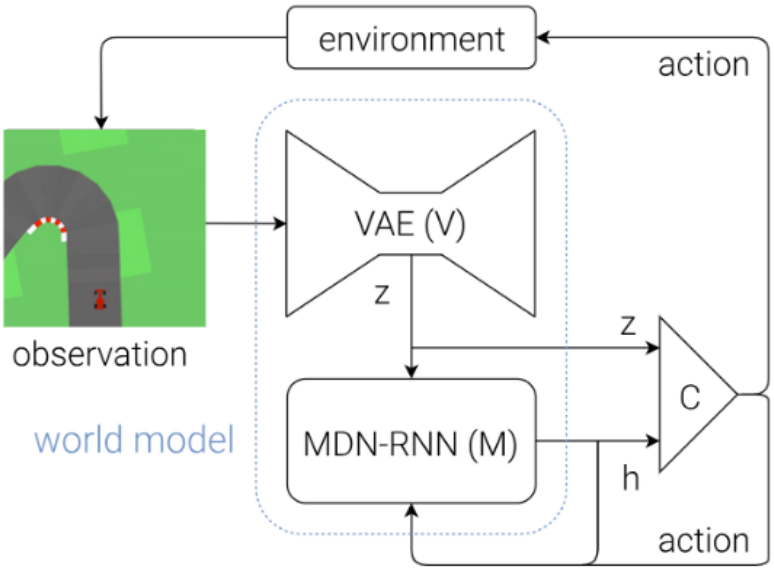

Figure 3.1: System architecture

4. Results & Discussion

4.1 Results

4.1.1 Experiments in CarRacing-v3

We first experimented with our model in the OpenAI Gymnasium CarRacing-v3 environment. This benchmark is challenging due to the randomly generated track tiles in each episode. Losing control can cause the car to drift off the track, making policy learning difficult.

We trained the policy using the DQN algorithm, both with and without the V model, to evaluate its effectiveness. The comparison is shown in Figure 4.1.

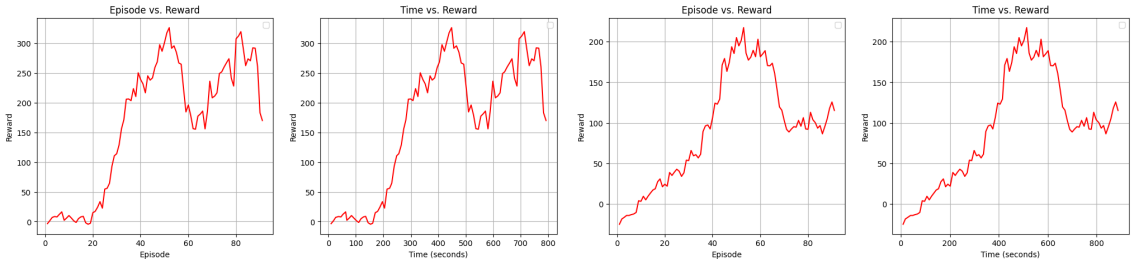

Figure 4.1: Comparison of policy learning with (left) and without (right) the V model in CarRacing-v3.

From the images, we observe that the policy trained with the V model learns much faster and converges more quickly to a higher reward compared to training without it.

To further quantify the performance, we present a bar graph comparing the average rewards achieved by the two models after training. Figure 4.2 illustrates the superior performance of the policy trained with the V model, which achieves a significantly higher average reward.

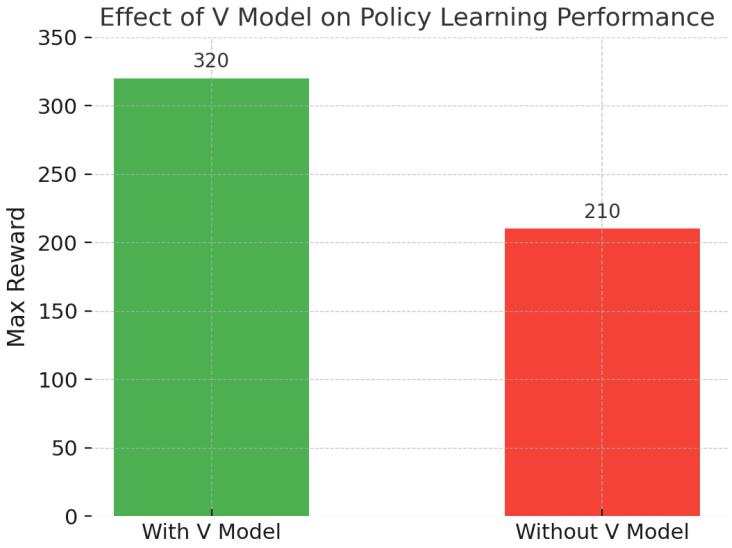

Figure 4.2: Bar graph comparing the average rewards of policies trained with and without the V model in CarRacing-v3.

Additionally, we analyzed the training dynamics by plotting the reward over time for both models. Figure 4.3 shows the reward curves, highlighting that the V model not only achieves higher rewards but also demonstrates faster convergence and greater stability during training.

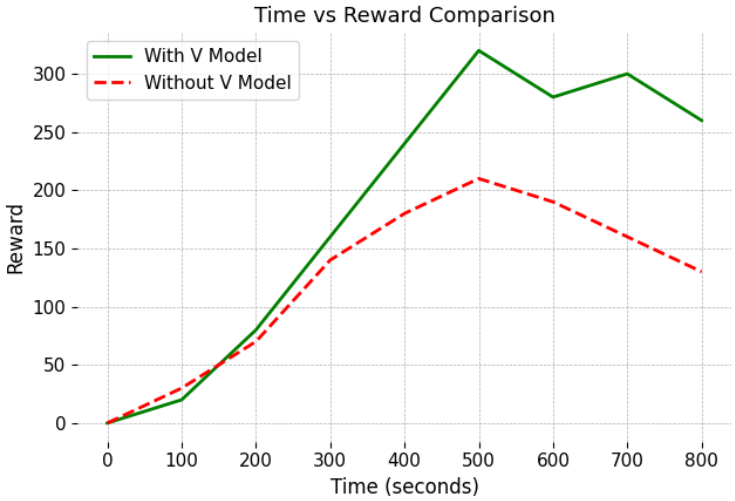

Figure 4.3: Reward vs. time graph for policies trained with and without the V model in CarRacing-v3, showing faster convergence and higher rewards with the V model.

These results collectively demonstrate that incorporating the V model into the DQN training process significantly enhances policy learning in the CarRacing-v3 environment, leading to improved performance and more efficient training.

4.1.2 V Model Encoding Performance

To evaluate how well the V model encodes observations, we visualize the latent representations for CarRacing-v3, CARLA simulator, and highway-env.

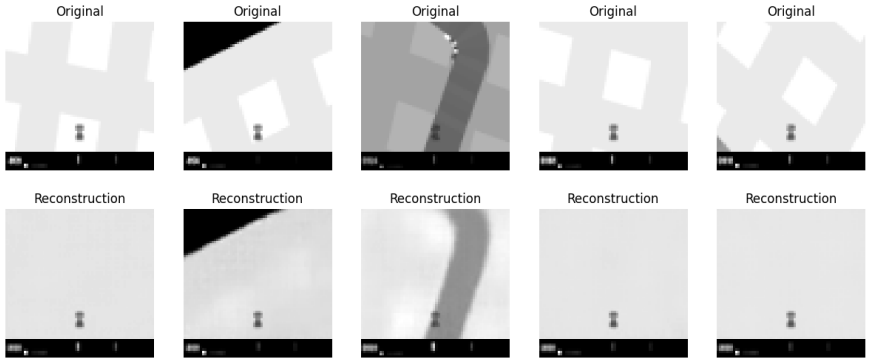

Figure 4.4: VAE encoded representation of CarRacing-v3.

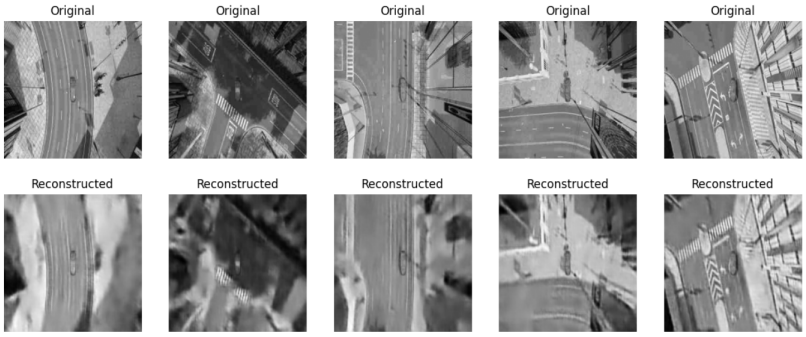

Figure 4.5: VAE encoded representation of CARLA.

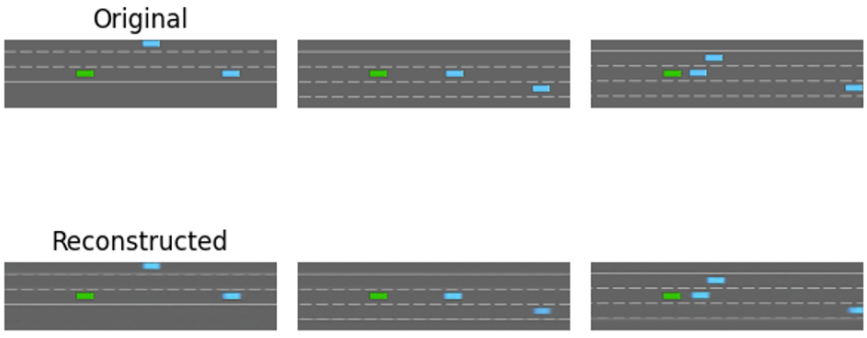

Figure 4.6: VAE encoded representation of highway-env.

We further analyze the VAE training process in CARLA and highway-env by plotting their respective total losses, KL divergence losses (for CARLA), and reconstruction losses.

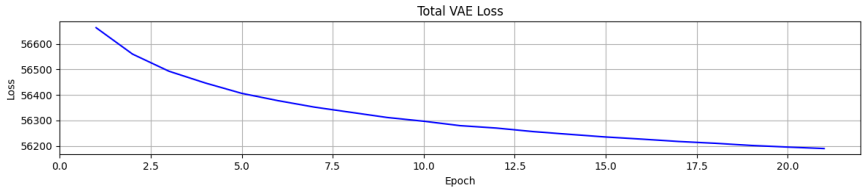

Figure 4.7: Total VAE loss in CARLA.

Figure 4.8: Reconstruction loss in CARLA.

Figure 4.9: KL divergence loss in CARLA.

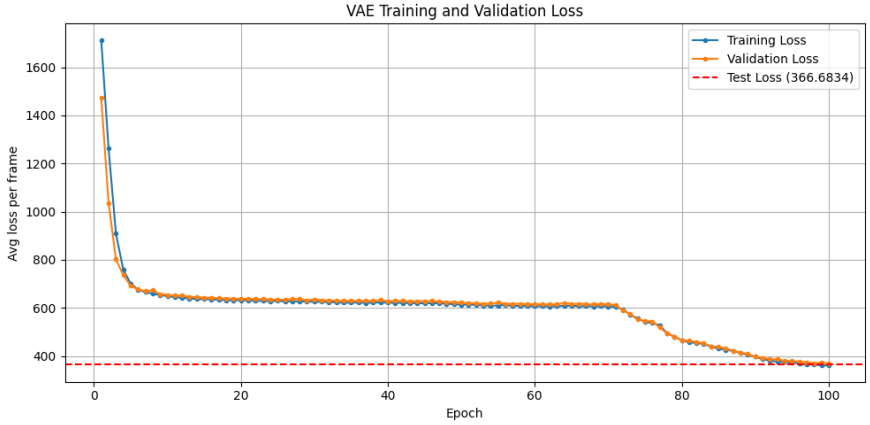

Figure 4.10: Training loss of VAE in highway-env.

These plots demonstrate how the VAE effectively minimizes the loss while encoding meaningful representations.

4.1.3 MDN-RNN Training Results

The M model (MDN-RNN) was trained to predict future latent states for CARLA. The training loss curve is shown in Figure 4.11.

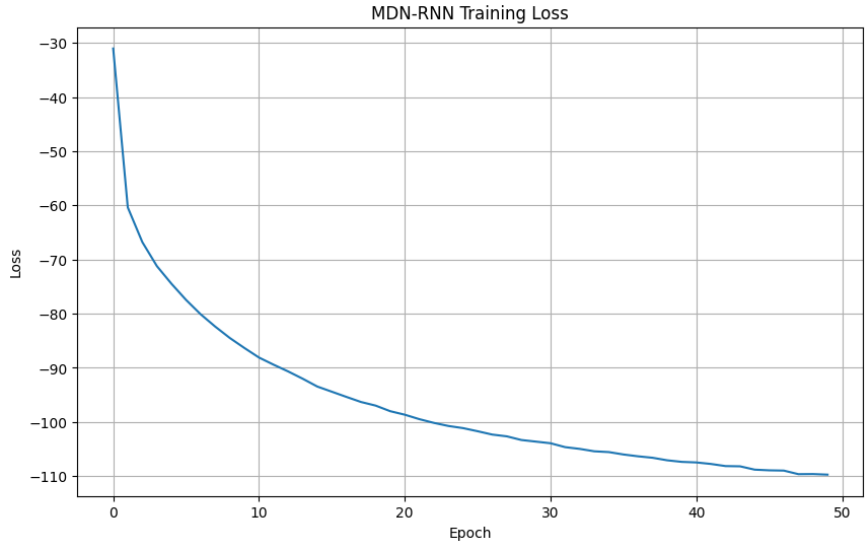

Figure 4.11: Training loss of MDN-RNN.

4.1.4 Policy Training Results

We trained the policy combining both V and M model using CMA-ES and DQN. The results are shown below.

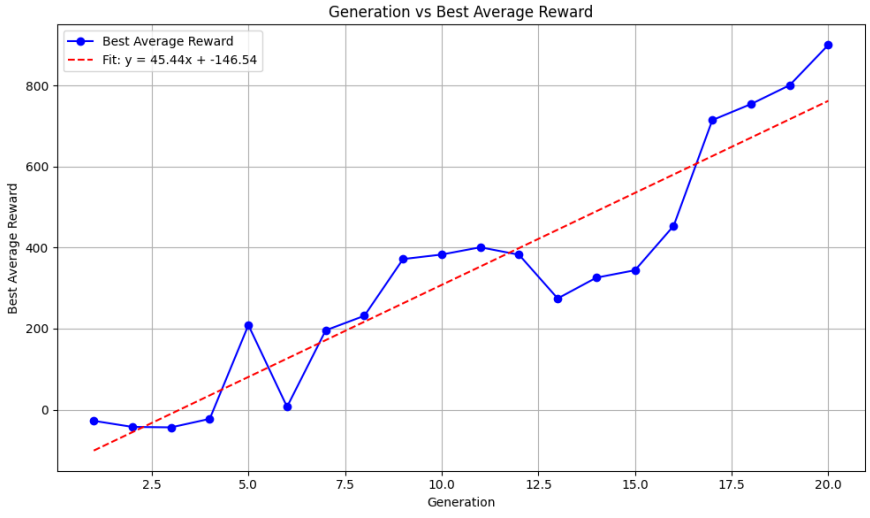

Figure 4.12: Generation vs. reward for CarRacing-v3 using CMA-ES.

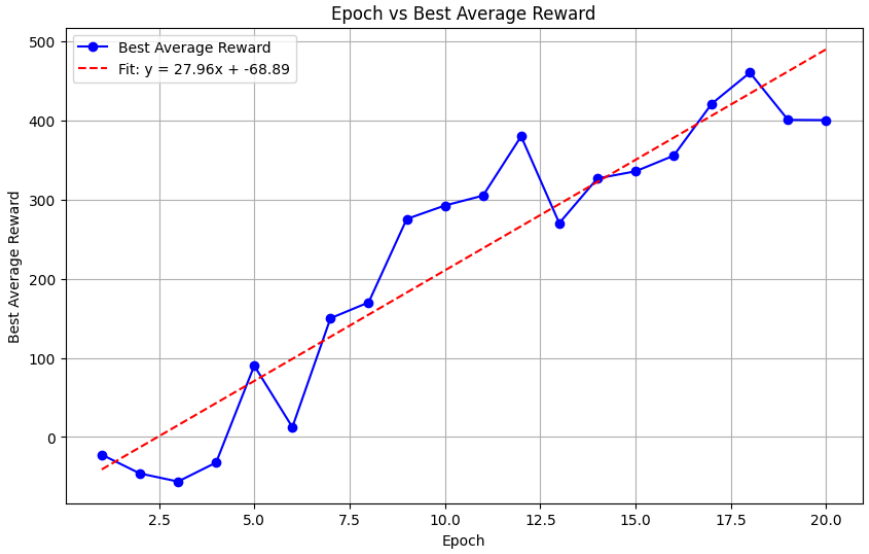

Figure 4.13: Epoch vs. reward for CarRacing-v3 using DQN.

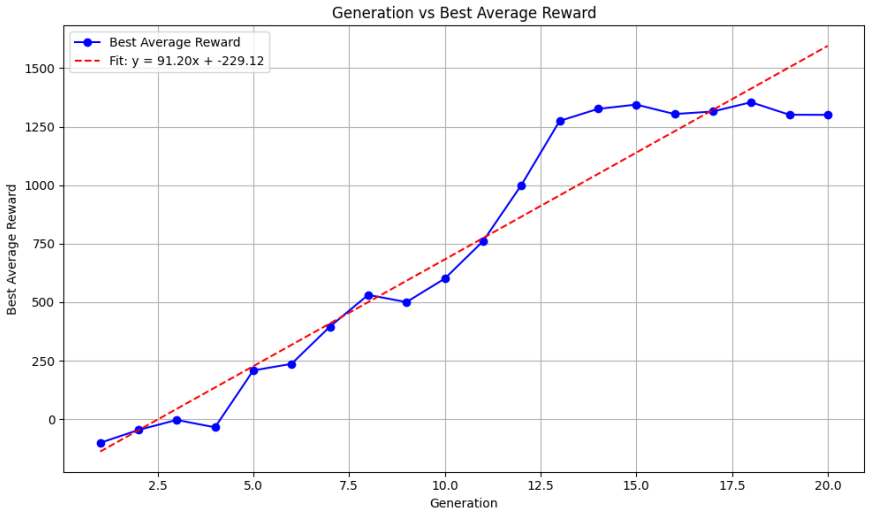

Figure 4.14: Generation vs. reward for CARLA using CMA-ES.

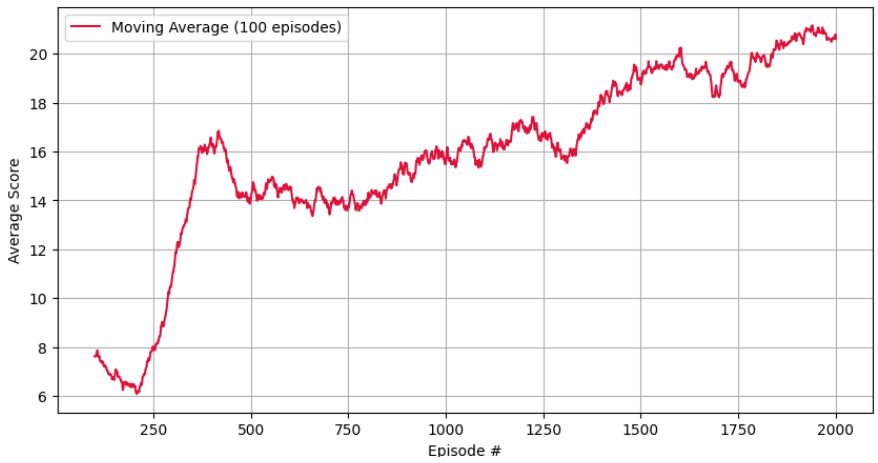

Figure 4.15: Episode vs. reward for highway-env using DQN.

4.1.5 Algorithm Comparison in CarRacing-v3

We compared the performance of two reinforcement learning algorithms—Deep Q-Network (DQN) and Covariance Matrix Adaptation Evolution Strategy (CMA-ES)—on the CarRacing-v3 environment.

The DQN-based controller achieved an average reward of approximately 450 after convergence. In contrast, the CMA-ES controller reached rewards up to 850 and was still improving, showing no clear signs of convergence.

Figure 4.16 provides a bar chart comparing the max rewards for each algorithm.

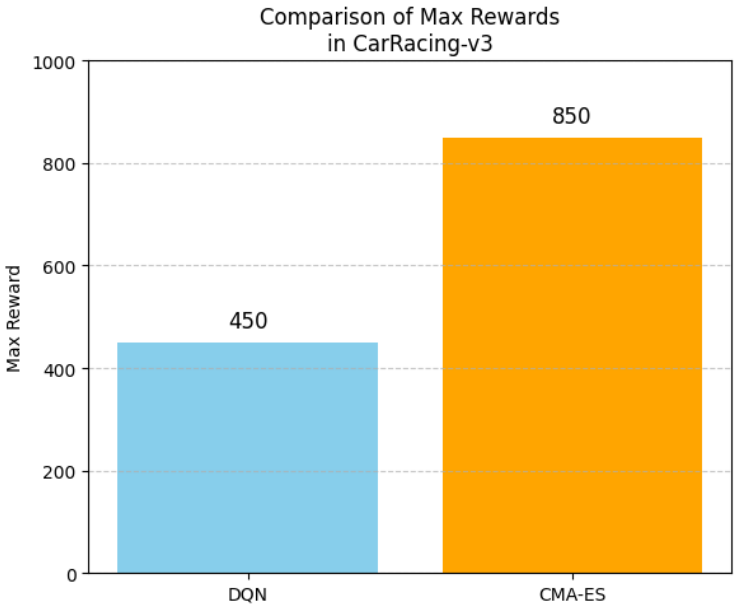

Figure 4.16: Comparison of max reward in CarRacing-v3 using DQN and CMA-ES.

5. Discussion

Our experiments demonstrate the importance of using a world model for policy learning. The V model provides a structured latent representation that accelerates policy learning, as seen in the DQN results. The M model enables understanding of environment dynamics. Training with CMA-ES further refines policy optimization in the learned world model. These results highlight the efficiency of world models in reinforcement learning, reducing direct simulation requirements while maintaining effective policy learning.

In the CarRacing-v3 environment, we observed that CMA-ES consistently achieves higher rewards than DQN. With a simple single-layer neural network as the controller, DQN converged around 450 reward, while CMA-ES reached 850 and continued improving. This aligns with existing research showing that evolutionary strategies like CMA-ES perform better in low-complexity neural network settings where gradient information is noisy or hard to estimate [30].

CMA-ES is more robust to noisy gradients and handles exploration in parameter space effectively. This makes it a good fit for our architecture, where the controller is small and direct policy gradients struggle to capture meaningful improvements.

6. Limitations and Future Enhancements

6.1 Limitations

The current approach has two primary limitations. First, the Variational Autoencoder (VAE) encodes observations solely based on reconstruction loss and Kullback-Leibler (KL) divergence. In complex environments, this method may fail to prioritize observation regions critical to reward outcomes, limiting the VAE's ability to capture essential state features.

Second, the state collection relies on a purely random policy, which is inefficient for exploration. This approach often misses critical states in complex environments, impeding effective learning and policy improvement.

6.2 Attempts at Improvement

To address the first limitation, we experimented with incorporating reward information into the VAE loss function, as shown below:

$$\text{Loss} = \alpha \cdot \text{BCE} + \beta \cdot \text{KL} + \gamma \cdot \text{Reward}$$

While this modification yielded minor improvements, the impact was not substantial. Time constraints prevented further exploration, but we hypothesize a modification of the architecture is needed. The ideal architecture may identify important regions for reward for each time step and then encode it.

6.3 Future Enhancements

To overcome the inefficiencies of random state collection, we propose an iterative state-gathering strategy. The process begins with collecting an initial dataset using a random policy. Vision (V) and Memory (M) models are then trained on this data, followed by training the agent. After initial training, the learned policy is used to explore the environment and collect new states. These states are then incorporated into the training dataset, and the V model, M model, and agent are retrained. This iterative cycle enables the discovery of previously unseen, high-value states, enhancing learning in complex environments.

However, this iterative approach is computationally expensive, posing a trade-off between exploration quality and resource demands.

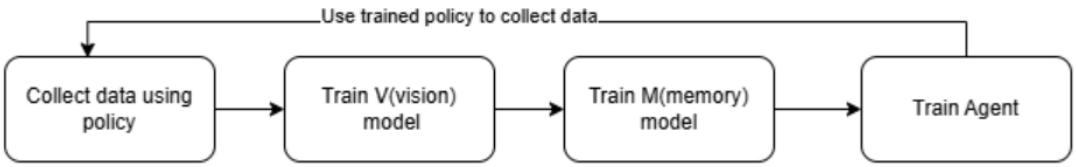

Figure 6.1: Possible improved architecture.

7. Conclusions

In this project, we successfully implemented and evaluated world models for autonomous driving in CarRacing-v3, CARLA and Highway-env environments. Our three-component approach—VAE for observation encoding, MDN-RNN for state prediction, and policy networks trained with DQN and CMA-ES—demonstrated significant advantages over traditional methods. The results clearly showed faster learning and better performance when using world models compared to direct policy learning. While our current implementation has limitations in how the VAE encodes observations, it provides a solid foundation for future extensions. This project demonstrates that world models offer an efficient approach to autonomous driving challenges even with limited computational resources.

References

[1] DeepMind. Alphago: The story so far. *DeepMind Research*, 2016. https://deepmind.google/research/breakthroughs/alphago/, Accessed: 2025-02-26.

[2] DeepMind. Alphastar: Grandmaster level in starcraft ii using multi-agent reinforcement learning. *DeepMind Blog*, 2019. https://deepmind.google/discover/blog/alphastar-grandmaster-level-in-starcraft-ii-using-multi-agent-reinforcement-learning/, Accessed: 2025-02-26.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *International conference on machine learning*, pages 1928–1937, 2016.

[6] Felipe Codevilla, Eder Miiller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[7] Anuj Agarwal, Antonio Loquercio, René Ranftl, and Davide Scaramuzza. Variational end-to-end navigation and localization. *IEEE Robotics and Automation Letters*, 7(2):2578–2585, 2022.

[8] Fumihiro Sasaki, Issei Sato, and Masashi Sugiyama. End-to-end differentiable adversarial imitation learning. *International Conference on Machine Learning (ICML)*, 2018.

[9] Peide Sun, Xinjing Wang, Yanning Yao, Lihua Xie, and Wei Liu. Query-efficient imitation learning for end-to-end autonomous driving. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021.

[10] David Isele, Akansel Cosgun, Kalpesh Subramanian, Jonathan Bohren, and Homayoun Wang. Efficient reinforcement learning for autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 3(1):1–12, 2018.

[11] Angelos Filos, Elias Tschernutter, Rowan McAllister, Yarin Gal, and Ingmar Posner. Confidence-aware reinforcement learning for self-driving cars. *International Conference on Machine Learning (ICML)*, 2020.

[12] Alex Kendall, James Hawke, Daniel Janz, Pawel Mazur, John Allen, Veronika Lam, Alex Bewley, and Amar Shah. Tackling real-world autonomous driving using deep reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[13] Siyuan Zhang, Yongsheng Zhao, Xiangyu Zhou, Jiabin Wu, and Jianhua Wang. Improving the performance of autonomous driving through deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 22(11):7335–7345, 2021.

[14] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[15] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations (ICLR)*, 2020.

[16] Danijar Hafner, Venkatraman Srinivasan, Timothy Lillicrap, and Corentin Tallec. Mastering atari with discrete world models. *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[17] Danijar Hafner, Deepak Pathak, and Timothy Lillicrap. Dreamerv3: Scaling world models for reinforcement learning. *arXiv preprint arXiv:2301.04104*, 2023.

[18] Haotian Yin, Ding Zhou, Zizhang Liu, Yuxiang Wang, and Zehuan Yuan. Law: Enhancing end-to-end autonomous driving with latent world model. *IEEE Transactions on Intelligent Vehicles*, 2023.

[19] Haotian Zhang, Wei Chen, Wenxi Xu, Jie Wu, and Haifeng Wang. Think2drive: Efficient reinforcement learning by thinking in latent world model for quasi-realistic autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[20] Jin Xu, Heng Ma, Yan Wang, Jianye He, and Jian Wang. Driving into the future: Multiview visual forecasting and planning with world model for autonomous driving. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[21] Shilin Li, Hang Zhou, Xiaoqing Zhou, and Changhu Wang. Bevworld: A multimodal world model for autonomous driving via unified bev latent space. *arXiv preprint arXiv:2310.12983*, 2023.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[24] Christopher M Bishop. Mixture density networks. In *Technical Report*, 1994.

[25] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[26] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[27] OpenAI Gym. Carracing-v3, 2021. Available at https://www.gymlibrary.dev/environments/box2d/car$_r$*acing*/.

[28] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *Proceedings of the 1st Annual Conference on Robot Learning (CoRL)*, pages 1–16, 2017.

[29] Edouard Leurent. An environment for autonomous driving decision-making. *GitHub repository*, 2018.

[30] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017. https://arxiv.org/abs/1703.03864, Accessed: 2025-04-19.