Using a Pretrained DDIM for Roll-out Scene Generation for Latent Imagination

Courtney Amm, Fei Yu Guan, Miguel Mallari, Jun Hao Zhong
University of Toronto
courtney.amm@mail.utoronto.ca
vi.guan@mail.utoronto.ca
miguel.mallari@mail.utoronto.ca
andy.zhong@mail.utoronto.ca

Abstract: The usage of automated vehicles is foreseen to have many practical applications, including but not limited to self-driving cars that can operate with minimal or absence of interference from the user; this technology would prove useful in emergency cases such as inability for the driver to continue operation. However, this form of automation is still in development and is subject to error-prone accidents pertaining to misjudgements in the user-independent control algorithm. While this project is still a heavy simplification of real-life cases of automated cars, it mainly serves as an investigation on the usage of latent imagination to refine the decision-making process of a car agent.

Evidently, a driving system must be prepared for unexpected road conditions such as sudden twists or turns, as not every road is expected to be naturally ordered and without obstructions. The consequences of decision failure and their disastrous impact in the real world are reasons for training a car model to anticipate its movement not only relative to its current environment, but from past accumulated experience as a means of "learning" and acting in advance towards sudden scenarios before they occur. Furthermore, rather than training models being traditionally focused on formatted edge cases constructed manually, it would be more beneficial to induce randomly generated scenarios to handle cases that would be deemed unforeseen by both the vehicle and the programmers of the algorithm. Introducing randomness into the scenario-generating process offers diversity in training the model as well as posing as a demo for the unpredictable road patterns in real-world applications.

Thus, in this paper, we investigate the performance and robustness of learning from generated data for latent imagination in the pixel space of a multi-car racing simulator.

Keywords: Autonomous Driving, Diffusion, Reinforcement Learning

1 Problem

In this project, we utilize a pre-trained Denoising Diffusion Implicit Model (DDIM) [1] to generate diverse, realistic driving scenarios for training an actor model. The generated image is used as an initial input to rollout the GRU [2] world to predict future outcomes, accumulate corresponding rewards signals, and optimize the policy. This method is designed to enhance autonomous driving system's adaptability and decision-making under rare or unforeseen circumstances, as such events would most likely not be accounted for in a standard training dataset. Our research explores the feasibility and effectiveness of augmenting training with these DDIM-generated scenes, focusing on their influence on the vehicle's ability to anticipate and react according to dynamic environments

Submitted to the 4th Conference on Robot Learning (CoRL 2020). Do not distribute.

of any variety. We use DDIM-integrated data into the training regimen in hopes of improving the model's overall performance and robustness, particularly in complex, multi-agent settings such as the multi-car racing simulator used in our experiments.

DreamerV3 [3], a state-of-the-art model in deep reinforcement learning, is used to optimize policy learning through predicted outcomes within complex environments. Concurrently, DDIMs are employed to synthetically generate high-quality driving scenarios. Ideally, this approach will enhance the autonomous system's ability to navigate unforeseen events by expanding the diversity of driving scenarios observed in the newly generated dataset. As these diffusion generated scenarios are given to DreamerV3, it will then be able to create a larger diversity of imagined rollouts, leading to a better training signal for the policy. This could be expanded to real life driving systems, enhancing existing autonomous driving technology.

2 Related Works

2.1 World Models and Latent Imagination:

There is a history of large neural networks being used to create abstract world model representations for machine learning algorithms [3]. The world model provide predictions of environmental and dynamic changes based on a latent representation learned from real data. In our case, this corresponding real data pertains to recordings taken from the driving simulator. Using this abstract world model, it has been shown that effective control policies can be learned using latent imagination [4], using the world model to train the control policy instead of actual data.

These control networks can then be reapplied to the actual task, as has been done in previous work on autonomous racing. The project extends on this work heavily, and uses the same simulation environment as [5].

2.2 Diffusion and Learning from Generated Data:

Diffusion-based models are currently the most advanced generative neural networks, allowing for the creation of realistic but novel data points. Collecting data for training can be time consuming, and it can be difficult to gather a large diversity of data for optimal training. Thus, there has been work on using diffusion generated data for training control policies [6]. This generated data provides additional variability to the dataset, making the algorithm more robust to unforeseen states. While our work is in a relatively simple simulation, similar networks have been trained for more complex real-world driving tasks, such as GAIA-1 [7].

2.3 Generated data in Autonomous Driving

Recent works on learning from generated data have significantly contributed to advancements in autonomous driving technologies. Specifically, the study by [8] showcases the utilization of synthetic data to enhance the learning processes of autonomous driving systems. This approach leverages generated datasets to expose the driving models to a wider range of scenarios, including rare or hazardous situations not commonly found in real-world data, thereby improving the robustness and adaptability of these systems in dynamic environments.

3 Methodology

3.1 Pipeline

For our pipeline, Dreamer-v3 and DDIM are combined for enhancing MBRL (Model-Based Reinforcement Learning) training performance. As introduced in [5], MBRL involves latent imagination which unrolls and extracts training signals from the data generated by DDIM. The structure of the Representation and Dynamics layer in the Dreamer is shown in Figure 1. Additionally, the Actor-

Critic update used for latent imagination and training the control network is shown in Figure 2. Full model training information can be seen in the appendix.

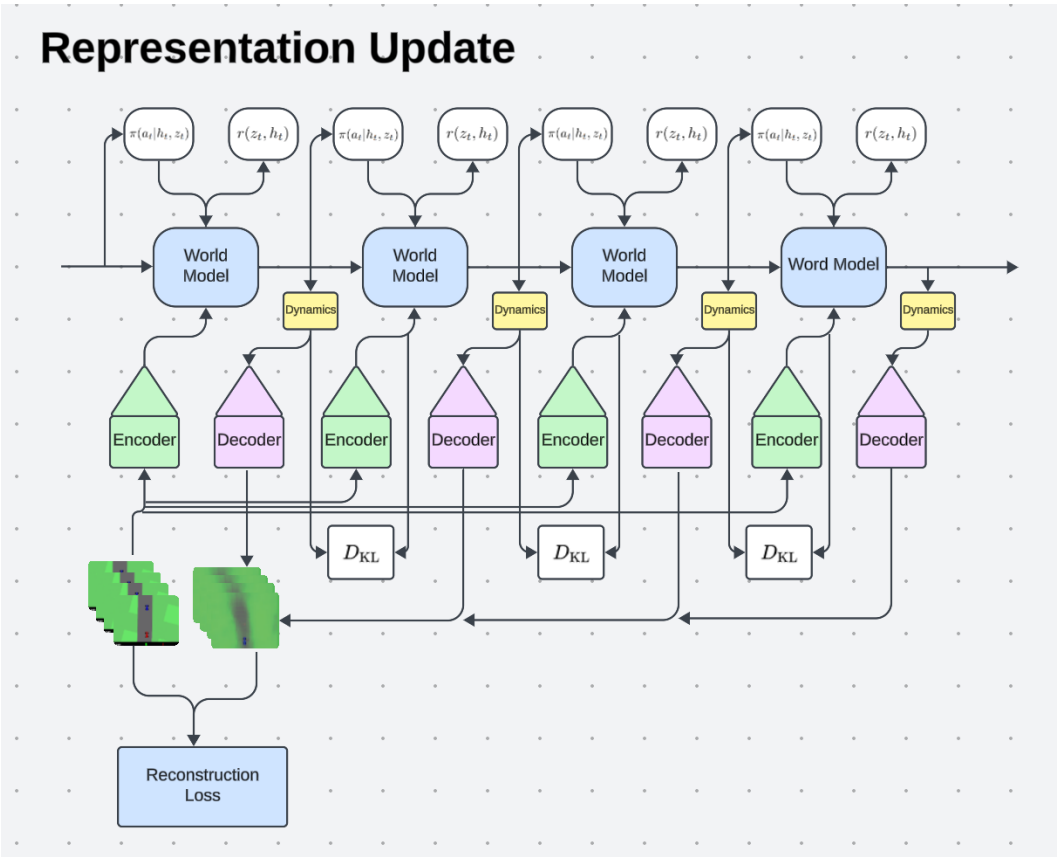

Figure 1: Representation/Dynamics Update

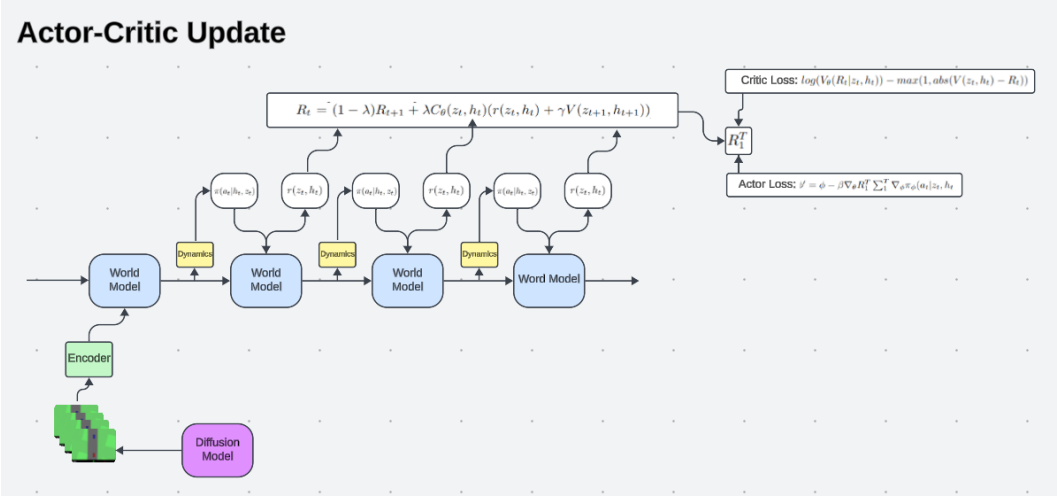

Figure 2: Latent Imagination

3.2 World Model

The world model captures the underlying changes and predicts the upcoming changes in the latent state. In this project, it is implemented as an GRU, and the information generated is used for learning the representations for itself, and for updating the actor and predicting the reward, through latent imagination.

3.3 Encoder and Decoder Model

The encoder model captures observations from the environment and feeds them into the world model. The decoder reconstructs the embeddings so that reconstruction loss can be calculated and minimized to ensure the embedding and hidden state are meaningful.

3.4 Actor Model

The actor model converts a meaningful, embedding sampled from the encoder, as well as the hidden state generated by the world model, to an action that maximizes the long-term return to the best of its knowledge. Our actor is stochastic, meaning it outputs a Gaussian distribution rather than a vector from the NN.

3.5 Critic Model

The critic model estimates the long-term value from the hidden state and the embedding. We use the values of the critic model and the policy of the reward model to update the policy.

3.6 Dynamics Model

The dynamics model converts a hidden state generated by the world model into an embedding. This embedding should contain meaningful information so that it resembles the input at the current timestep.

3.7 Reward Model

The reward model uses the world model and its state to determine the predicted reward it will receive.

3.8 Termination Predictor

The termination predictor outputs 0 if the next state of the current state is terminal, and 1 otherwise. We get this information from the replay buffer.

3.9 Update Rules

3.10 Baseline Architecture

$$\text{World Model: } h_t = f(z_{t-1}, a_{t-1}, h_{t-1})$$
$$\text{Encoder Model: } z_t \sim e(x_t, h_t)$$
$$\text{Decoder Model: } \hat{x_t} \sim d(z_t, h_t)$$
$$\text{Actor Model: } a_t \sim \pi(a_t | z_t, h_t)$$
$$\text{Critic Model: } v_t \sim V(z_t, h_t)$$
$$\text{Dynamics Model: } \hat{z_t} \sim m(h_t)$$
$$\text{Reward Model: } \hat{r_t} \sim r(z_t, h_t)$$

where z_t denotes the embedding of the encoder at time t.
h_t is the hidden states of the GRU.
$\hat{r_t}$ is the predicted reward at time t.
a_t is the action taken at time t.
x_t is the pixel input at time t.

We define the parameters of the world model as: θ. Then

$$\mathcal{L}_\theta = \sum_{t=1}^{T} \mathcal{L}_{\theta;pred} + \mathcal{L}_{\theta;dyn} + \mathcal{L}_{\theta;rep}$$

where

$$\mathcal{L}_{\theta;pred} = MSE(\hat{x_t}, x_t) + MSE(\hat{r_t}, symlog(r_t)) - log(C_\theta(c_t|z_t, h_t))$$

$$\mathcal{L}_{\theta;dyn} = max(1, D_{KL}(sg(e_\theta(z_t|x_t, h_t))||m_\theta(\hat{z_t}|h_t))$$

$$\mathcal{L}_{\theta;rep} = max(1, D_{KL}(e_\theta(\hat{z_t}|x_t, h_t)||sg(m_\theta(\hat{z_t}|h_t))))$$

$$\mathcal{L}_{\theta;crit} = MSE(V_\theta(v_t|z_t, h_t), symlog(R_t))$$

$sg(\cdot)$ is the stop-gradient operator and here it means $\hat{H}$ where its parameters $\hat{\psi}_t = \rho\psi_t + (1-\rho)\hat{\psi}_t$, with ρ being a float number close to 0. $sg(H)$ is also called the "target network" of H. We employ $symlog$ function as mentioned in the original DreamerV3 paper, to enable better representation learning and mitigate overfitting. However, in some other places, such as the reconstruction loss, $\mathcal{L}_{\theta;pred}$, we did not use $symlog$ as the latent representation would update too aggressively due to the rapidly decreasing reconstruction loss, and our alignment mechanism cannot sync up.

For updating the actor and critic, we let the world model imagine 15 steps, only giving it the initial h, and the action input for each step, and let it unroll. This is called Latent Imagination. Specifically, we define $R_t = (1-\lambda)R_{t+1} + \lambda C_\theta(z_t, h_t)(r(z_t, h_t) + \gamma V(z_{t+1}, h_{t+1})) - \alpha H(\pi(a_t|z_t, h_t))$ where $\lambda = 0.95$, and $R_T = R_{T-1}$ whereas $T > 0$ is the terminal step and $H(\cdot)$ is the entropy regularizer described in the next subsection. In this paper specifically, we also train the actor half of the times the model is trained to avoid being misled by the model error as discussed in TD3 paper[9], giving the critic, the model and the alignment mechanism more time to refine its update signals to the actor.

3.11 Training of diffusion model

Due to the slow evaluation speed of DDIM, we lowered the inference steps of the diffuser to 30 and changed the downsampling blocks to sizes of (128, 256, 256, 256). For the DDIM, we used the algorithms in [10], where p_θ is our latent forward model and x_t is our image at step t:

Algorithm 2 DDIM Forward Process

Require: Original data x_0, total timesteps T, variance schedule $\{\beta_1, \ldots, \beta_T\}$
Ensure: Noisy data x_T
for $t = 1$ to T **do**
  Sample noise $\epsilon_t \sim \mathcal{N}(0, I)$
  $\alpha_t = 1 - \beta_t$
  $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$
  $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_t$
end for

Algorithm 4 DDIM Backward Process (Sampling)

Require: Noisy data x_T, total timesteps T, model $\mathcal{M}$, variance schedule $\{\beta_1, \ldots, \beta_T\}$
Ensure: Reconstructed data x_0
for $t = T$ downto 1 **do**
  Estimate $\hat{\epsilon}_t = \mathcal{M}(x_t, t)$
  Compute intermediate values for reconstruction
  if $t > 1$ **then**
    x_{t-1} = some function of x_t, $\hat{\epsilon}_t$ and other parameters
  else
    x_0 = final reconstruction from x_1 and $\hat{\epsilon}_1$
  end if
end for

Algorithm 1 DLC-V1

Initialize parameters θ for model, ϕ for policy, $\tau = 0.005$, $\lambda = 0.95$, $\alpha = 0.0006$, $\beta = 0.0001$
Hard update $\hat{\theta_{targ}} \leftarrow \theta$
for i = 1, 2,..., num_episodes **do**
  while not $Terminate$ **do**
    Predict z_t, h_t from the world model for each agent
    $action \leftarrow a_t \sim \pi_a(z_t, h_t)$
    Add $(O_t, a_t, r_t, term_t)$ to agent's replay buffer
    $O_{t+1}, r_{t+1}, term_{t+1} \leftarrow$ *env.step(action)*
  end while
  for y = 1,2,..., Training_step = 20 **do**
    Sample sequences τ of batch size $B = 15$ and length $L = 50$ from replay buffer
    Use the encoder for each agent to predict embeddings $z_t, \hat{z}_t$
    Calculate $\mathcal{L}_\theta$ using the **Update Rule**
    Update the model by minimizing $\mathcal{L}_\theta$ with learning rate α
    for imagination_step = 1,...,T=15 **do**
      Sample the stochastic embedding $\hat{z}_t \sim m(\cdot|h_t)$
      Choose $a_t \sim \pi(a_t|z_t, h_t)$
      Propagate the deterministic component of state h_{t+1} with a_t, $\hat{z}_t$ and h_t
      Accumulate λ-return in **Update Rule**, $r_t \sim r(z_t, h_t)$ and $V_t \sim V(z_t, h_t)$
    end for
    When $y\%2$, Update the actor through gradient ascent,

$$\phi' = \phi + \beta \nabla_a R_1^T \nabla_\phi \pi_\phi(a_t|z_t, h_t) - \alpha \nabla H_t$$

, where

$$H_t = \lambda log(\pi(a_t|z_t, h_t)) + (1 - \lambda) H_{t+1}, H_{t+14} = log(\pi(a_{t+14}|z_{t+14}, h_{t+14}))$$

    Update the critic by minimizing the critic loss using learning rate α:

$$\hat{R}_t \sim V(\cdot|z_t, h_t)$$

$$\mathcal{L}_{\theta;crit} = MSE(\hat{R}_t, sg(symlog(R_t)))$$

    Update the target critic by soft update:

$$\theta_{crit;targ} \leftarrow (1 - \tau)\theta_{crit;targ} + \tau\theta_{crit}$$

  end for
end for

3.12 Latent Space Alignment Layer

During training, we observed some artifacts in the images that likely reduced our method's effectiveness. In order to combat this, two additional layers forming a bottle neck shape (let's call it a "mini-Unet") were added to the model when evaluating the latent state from diffusion generated images. We use a UNet architecture for filtering out information that are hard to predict through a 0.5 size narrower layer, outputting the difference between the "fake"(generated images latent) versus the ground truth latent. This network exists between the embedded states (z) of the diffusion generated image and the ground truth image, and then through a skip connection between the original state embedding and output of the mini dense Unet. It has been proven from time to time that predicting the difference instead of the whole latent is quicker to learn, preserves rich meaningful information, and typically outperform not using such a method(see [11]). It is trained as a generator, through taking the chain rule from the loss of the discriminator, to discriminate between latent states from the simulator images and the latent state from the diffusion-generated images. The discriminator utilizes cross entropy loss, as is shown below.

$$L(t, x) = -tlog(x) - (1 - t)log(1 - x)$$

However, the mini Unet uses the following to confuse the discriminator (the discriminator outputs 0 for ground truth and 1 for the supposedly "fake" latent inputs); the loss is taken W.R.T. the mini-Unet's parameters:

$$-log(1 - MiniUnet(x))$$

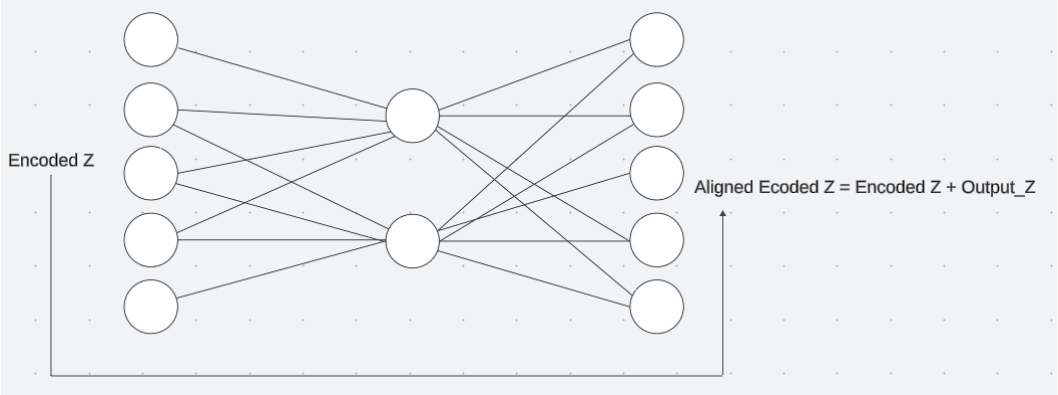

Figure 3: Skip Mini-Unet

4 Evaluation

4.1 Training Performance

Results from our method, which are the episodes vs. unsmoothed episodic returned can be observed below. Overall, the model is proven to be effective. We are unable to benchmark as many seeds/episodes as we proposed due to time constraints. Most of our times were used to benchmark, trying various ideas and it usually take hours before getting a positive or negative result. And, despite reaching the peak(a successful full lap completion) sooner than the baseline, our training curve seems to be more stochastic. This is often due to the car driving off the road in the session. Please notice that the track is randomly generated and the car is seeing the layout for the very first time. If trained in repetition on the same track, the rate of error would be significantly lower since the world model literally "memorizes" the track's features instead of trying to generalize across unseen tracks. This may also be partially caused by visual artifacts in images produced by the diffusion model, which may include not meaningful information the world model and thus undermining the quality of actor critic training.

Results are still found to be replicable when using the same setup and hyperparameters.

In order to evaluate our diffusion generated images, we test the realness of the generated images by simply comparing them to the images observed directly from the simulation. As can be seen in the figure, artifacts appear in the diffusion generated images, especially in tightly detailed areas such as text. Noises are visibly also present in the images which likely makes it more difficult for the encoder to successfully encode the state from the generated images, causing adversarial effect, undermining the prediction/meaningful interpretation of the image.

4.2 Comparison

The effectiveness of the control model was tested using the following method. We will train in the environment for 50 episodes. While running the simulation to train the world model, we will train

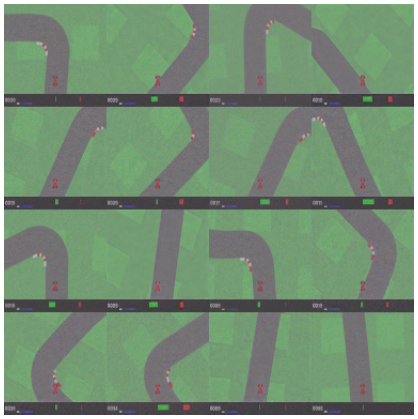

Figure 4: The output of our DDIM trained for 2 hours with 30 inference steps.

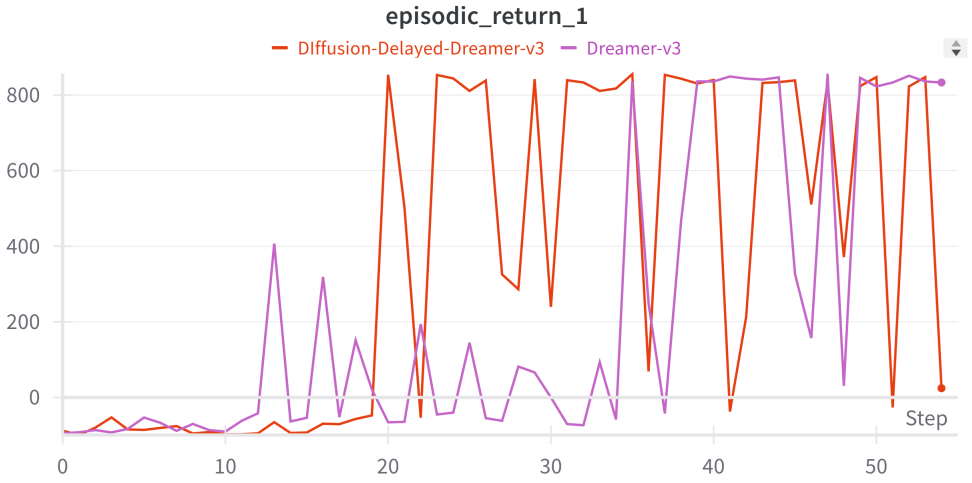

Figure 5: A comparison of the episodic return of our DDIM-augmented model versus the baseline Dreamer v3 using regular simulation data.

for 20 iterations per episode offline. We will use a batch size of 50 (sampling from 50 sequences in parallel) and batch length of 50. To train the policy, we will use 300 steps of imagination(20 iterations, 15 imagination steps each) per episode of diffusion generated data for 50 episodes.

A comparison of our method versus the baseline can be seen in Figure 3. As shown, our method significantly out preforms the baseline over the 50 test episodes. It reaches peak at the 20th episode while the baseline reaches peak at around 35th. Additionally, there is slightly higher stochasticity in the data indicated by drops in the data of the diffusion model.

5 Limitations

5.1 Stochatiscity

We observe significantly more frequent performance drops in our method, which suggests even with the latent alignment mechanism, there are still noisy images being generated just by 30 steps of DDIM being trained for 2 hours. Additionally, the alignment mechanism - since it involves skipping frame - still preserves content from the originally encoded latent. Thus, its meaningful information is capped at those in the original encoded latent, affecting the rollout imagination quality.

5.2 Sensitivity to Hyperparameters Setup

We have found that the setup is very sensitive to the choices of hyperparameters, layer size, and the choices of activation. The parameter space seems to be sparse and have low error tolerance. For example, the model may reach the peak in 20 runs if having a 3-layer representation model but 100 runs if having a 2-layer representation model. Additionally, the choice of ϵ in ADAM optimizer(intended to address zero division issues, but also used as a form of dampening in the optimization step so that the gradient is less/more sensitive to numerator, the moment of the gradient when the denominator is close to 0) is also very sensitive. Having a doubled or tripled value of epsilon would typically affect the training outcome by a lot, and require a lot more higher entropy coefficient to compensate for the aggressive or overly deterministic actions. In our case, for model learning, our epislon is 1e-8(the default) which means more aggressive update, but for the actor and the critic, the epsilon is 1e-5. Those are the results from the Dreamer v3 paper that we avoid to risk changing by ourselves.

5.3 Real-world applications

The basic idea behind this method could be expanded to more advanced autonomous driving systems. As mentioned previously, generative models have a greater potential to increase richness in more complex real-life data versus our simulator data. Additionally, autonomous driving data is more expensive to collect in real life or in more computationally advanced and thus more expensive simulators. This makes generative augmentation more attractive, as it may allow for improved training without the need to collect more data. However, there are certain extensions to our project needed to make this feasible. Our project is currently only working with image data, while many autonomous vehicles use multi-modal data from various sensors. This requires that a more advanced system be able to account for this multi-modal data, either through additional generative models or through some type of data fusion. Further expansions of this method using a real-world environment instead of a simulation would also require the addition and handling of more control variables such as road friction, inertia, and drag when calculating a new control step.

6 Conclusion

6.1 Project Summary

Overall, our project successfully implements a novel method for training a simplified autonomous racing system using latent imagination (Dreamer-V3) with diffusion-generated data outperforming the baseline. We believe that the use of generated data will continue to be relevant as methods of data augmentation continue to become more advanced. We hope that this project adds to that body of work, providing a jumping off point for more advanced systems.

References

[1] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. URL https://arxiv.org/abs/2010.02502.

[2] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[3] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. URL https://arxiv.org/abs/2301.04104.

[4] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

[5] W. Schwarting, T. Seyde, I. Gilitschenski, L. Liebenwein, R. Sander, S. Karaman, and D. Rus. Deep latent competition: Learning to race using visual control policies in latent space. In *Conference on Robot Learning*, 2020.

[6] E. Alonso, A. Jelley, A. Kanervisto, and T. Pearce. Diffusion world models, 2024. URL https://openreview.net/forum?id=bAXmvOLtjA.

[7] A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton, G. Corrado, et al. Gaia-1: A generative world model for autonomous driving, 2023.

[8] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots. Agile autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

[9] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018. URL https://arxiv.org/abs/1802.09477.

[10] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. 2020.

[11] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, N. Johnston, and E. P. Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1802.01436*, 2018.

Appendices

A Appendix

This work extends on work from a previous course project. This project only adds the DDIM component which generates images feeding into the latent imagination of Fei's CSC413 project (Figure 1 and 2).

B Group Member Contributions

Courtney	Code refactoring, Related work, limitations, general editing of all sections of final paper.
Fei	Dreamer v3 baseline implementation, any benchmarks, hyperparameter tuning, model design, debug, solving every problem, idea generation, algorithm design, proofreading the paper and rewrite if necessary
Miguel	Abstract, Real-World applications
Andy	None

C Network Architecture

World Model	
Network Type: GRU	**Embedding Dim:** 512
Initial h: Zeros	**Output Dim:** 500
Representation Model (Sub-part of the encoder model)	
Network Type: MLP	**Embedding Dim:** 1000
Activation: SiLU	**Hidden Dim:** 1000
Output Type: Gaussian	**Output Dim:** $\mu(30)$, $\sigma(30)$
Num Layers: 2	**Layer Norm:** Every Layer Before Activation
Transition Model	
Network Type: MLP	**Input Dim:** 1000
Activation: SiLU	**Hidden Dim:** 1000
Output Type: Gaussian	**Output Dim:** $\mu(30)$, $\sigma(30)$
Num Layers: 3	**Layer Norm:** Every Layer
Encoder	
Network Type: CNN	**1st Layer:** Conv2D(in=3, out=32, k=4, s=3)
2nd Layer: Conv2D(in=32, out=64, k=4, s=2)	**3rd Layer:** Conv2D(in=64, out=128, k=4, s=2)
4th Layer: Conv2D(in=128, out=256, k=4, s=2)	**5th Layer:** Linear(in=1024, out=512)
Activation Type: SiLU	**Output Type:** Deterministic
Layer Norm: Every Layer Before Activation	
Decoder	
Network Type: Transpose CNN	**1st Layer:** Linear(230, 1024)
2nd Layer: Unflatten((1024, 1, 1))	**3rd Layer:** in=1024, out=128, k=5, s=2
4th Layer: in=128, out=64, k=6, s=2	**5th Layer:** in=64, out=3, k=6, s=3
Activation Type: SiLU (Sigmoid for the last layer)	**Output Type:** Deterministic
Output Dims: $(3, 96, 96)$	**Layer Norm:** Before Every SiLU Activation
Reward	
Network Type: MLP	**Num Layers:** 3
Activation Type: SiLU and None for last layer	**Output Type:** Gaussian
Layer Norm: Every Layer Before Activation Except Final Layer	**Hidden Size:** 400
Output Dim: $\mu(1)$, $\sigma(1)$	
Continue Predictor	
Network Type: MLP	**Num Layers:** 3
Activation Type: SiLU and None for last layer	**Output Type:** Categorical
Layer Norm: Every Layer Before Activation Except Final Layer	**Hidden Size:** 400
Output Dim: 1	
Actor	
Network Type: MLP	**Num Layers:** 4
Activation Type: SiLU and None for last layer	**Output Type:** Gaussian
Layer Norm: Every Layer Before Activation Except Final Layer	**Hidden Size:** 400
Output Dim: $\mu(2)$, $\sigma(2)$	
Critic	
Network Type: MLP	**Num Layers:** 3
Activation Type: SiLU and None for last layer	**Output Type:** Gaussian
Layer Norm: Every Layer Before Activation Except Final Layer	**Hidden Size:** 400
Output Size: $\mu(1)$, $\sigma(1)$	
Mini-UNET	
Network Type: MLP	**Num Layers:** 2
Activation Type: SiLU	**Output Type:** Deterministic
Layer Norm: Every Layer Before Activation	**Hidden Size:** 256
Output Size: 512	**Input Size:** 512

D Network Architecture (Continued)

Mini-UNET Disc
Network Type: MLP
Num Layers: 2
Activation Type: SiLU
Output Type: Deterministic
Layer Norm: Every Layer Before Activation
Hidden Size: 256, 128
Output Size: 1
Input Size: 512

DDIM-UNET
Network Type: UNET
Block Out Channels: 128, 128, 256, 256, 256
Attention Head Dim: 16
Downblock Types: DownBlk2D, DownBlk2D, DownBlk2D, AttnDownBlk2D, DownBlk2D
Upblock Types: UpBlk2D, AttnDownBlk2D, UpBlk2D, UpBlk2D, UpBlk2D
Activation: SiLU

E Hyperparameters:

Actor Learning Rate: 0.0001
Critic Learning Rate: 0.0006
Model Learning Rate: 0.0006
Discriminator Learning Rate: 0.001
Mini-UNET Learning Rate: 0.0001
DDIM Learning Rate: 0.0002
DDIM Training Steps: 1000
DDIM Inference Steps: 30
World Model Grad Norm Clip: 1000
Other Grad Norm Clip: 100
Optimizers Type: ADAM
Actor-Critic ϵ: 1e-5
World Model ϵ: 1e-8
Replay Buffer Size: 100000
Transition Model Minimum σ: 0.12
Representation Model Minimum σ: 0.12
Actor Minimum σ: 0.1
Entropy Coefficient α: 0.1
Discount-γ: 0.99
Return-λ: 0.95
Critic Decay: 0.98
Actor Update Interval: 2
Deterministic Size: 200
Stochastic Size: 30
Embedding Size: 512
DDIM Batch Size: 100