Supplementary Materials for
DriveGAN: Towards a Controllable High-Quality Neural Simulation

1. Model Architecture and Training

We provide detailed descriptions of the model architecture and training process for pre-trained image encoder-decoder (Sec. 1.1) and dynamics engine (Sec. 1.2). Unless noted otherwise, we denote tensor dimensions by $H \times W \times D$ where H and W are the spatial height and width of a feature map, and D is the number of channels.

1.1. Pre-trained Latent Space

The latent space is pretrained with an encoder, generator and discrminator. Figure 1 shows the overview of the pretraining model.

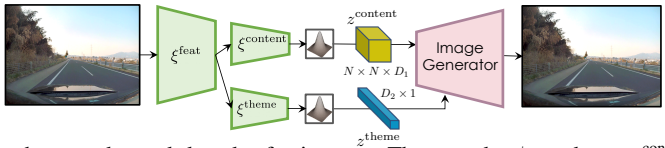

Figure 1. Pretraining stage learns the encoder and decoder for images. The encoder ξ produces z^{content} and z^{theme} which comprise the disentangled latent space that the dynamics engine trains on. The gaussian blocks represent reparameterization steps [9].

1.1.1 Encoder

Encoder ξ takes an RGB image $x \in \mathbb{R}^{256\times256\times3}$ as input and produces disentangled latent codes $z = \{z^{\text{theme}}, z^{\text{content}}\}$ where $z^{\text{theme}} \in \mathbb{R}^{128}$ and $z^{\text{content}} \in \mathbb{R}^{4\times4\times64}$. ξ is composed of a feature extractor ξ^{feat} and two encoding heads ξ^{content} and ξ^{theme}.

Layer	Output dimension
Conv2d 3×3	256×256×128
ResBlock	128×128×256
ResBlock	64×64×512
ResBlock	32×32×512

Table 1. ξ^{feat} architecture

Layer	Output dimension
ResBlock	16×16×512
ResBlock	8×8×512
ResBlock	4×4×512
Conv2d 3×3	4×4×512
Conv2d 3×3	4×4×128

Table 2. ξ^{content} architecture

Layer	Output dimension
Conv2d 3×3	32×32×512
AvgPool2d 32×32	512
Linear	256

Table 3. ξ^{theme} architecture

The above tables show the architecture for each component. ξ^{feat} takes x as input and consists of several convolution layers whose output is passed to the two heads. Conv2d 3×3 denotes a 2D convolution layer with 3×3 filters and padding of 1 to produce the same spatial dimension as input. ResBlock denotes a residual block [3] with downsampling which is composed of two 3×3 convolution layers and a skip connection layer. After each layer, we put the leaky ReLU [11] activation function, except for the last layer of ξ^{content} and ξ^{theme}. The outputs of ξ^{content} and ξ^{theme} are equally split into two chunks by the channel dimension, and used as μ and σ for the reparameterization steps:

$$z = \mu + \epsilon\sigma, \quad \epsilon \sim N(0, I) \tag{1}$$

producing $z^{\text{theme}} \in \mathbb{R}^{128}$ and $z^{\text{content}} \in \mathbb{R}^{4\times4\times64}$.

1.1.2 Generator

The generator architecture closely follows the generator of StyleGAN [7]. Here, we discuss a few differences. z^{content} goes through a 3×3 convolution layer to make it a 4×4×512 tensor. StyleGAN takes a constant tensor as an input to the first layer. We concatenate the constant tensor with z^{content} channel-wise and pass it to the first layer. z^{theme} goes through 8 linear layers, each outputting a 1024-dimensional vector, and the output is used for the adaptive instance normalization layers in the same way *style* vectors are used in StyleGAN. The generator outputs a 256×256×3 image.

1.1.3 Discriminator

Dicriminator takes the real and output images (256×256×3) as input. We use multi-scale multi-patch discriminators [17, 6, 15], which results in higher quality images for complex scenes.

Layer	Output dimension
Conv2d 3×3	256×256×128
ResBlock	128×128×256
ResBlock	64×64×512
ResBlock	32×32×512
ResBlock	16×16×156
ResBlock	8×8×512
ResBlock	4×4×512
Conv2d 3×3	4×4×512
Linear	512
Linear	1

Table 4. D_1 architecture

Layer	Output dimension
Conv2d 3×3	256×256×128
ResBlock	128×128×256
ResBlock	64×64×512
ResBlock	32×32×512
ResBlock	16×16×512
Conv2d 3×3	16×16×1

Table 5. D_2 architecture

Layer	Output dimension
Conv2d 3×3	128×128×128
ResBlock	64×64×256
ResBlock	32×32×512
ResBlock	16×16×512
ResBlock	8×8×512
Conv2d 3×3	8×8×1

Table 6. D_3 architecture

We use three discrminators D_1, D_2, and D_3. D_1 takes a 256×256×3 image as input and produces a single number. D_2 takes a 256×256×3 image as input and produces 16×16 patches each with a single number. D_3 takes a 128×128×3 image as input and produces 8×8 patches each with a single number. The inputs to D_1, D_2, D_3 are the real and generated images, except that the input to D_3 is downsampled by 2×. The model architectures are described in the above tables, and we use the same convolution layer and residual blocks from the previous sections. Each layer is followed by a leaky ReLU activation function except for the last layer.

1.1.4 Training

We combine the loss functions of VAE [9] and GAN [2], and let $L_{pretrain} = L_{VAE} + L_{GAN}$. We use the same loss function for the adversarial loss L_{GAN} from StyleGAN [7], except that we have three terms for each discriminator. L_{VAE} is defined as:

$$L_{VAE} = E_{z\sim q(z|x)}[log(p(x|z))] + \beta KL(q(z|x)||p(z))$$

where $p(z)$ is the standard normal prior distribution, $q(z|x)$ is the approximate posterior from the encoder ξ, and KL is the Kullback-Leibler divergence. For the reconstruction term, we reduce the perceptual distance [18] between the input and output images rather than the pixel-wise distance, and this term is weighted by 25.0. We use separate β values β^{theme} and β^{content} for z^{content} and z^{theme}. We also found different β values work better for different environments. We use $\beta^{\text{theme}} = 1.0, \beta^{\text{content}} = 2.0$ for Carla, $\beta^{\text{theme}} = 1.0, \beta^{\text{content}} = 4.0$ for Gibson, and $\beta^{\text{theme}} = 1.0, \beta^{\text{content}} = 1.0$ for RWD. Adam [8] optimizer is employed with learning rate of 0.002 for 310,000 optimization steps. We use a batch size of 16.

1.2. Dynamics Engine

With the pre-trained encoder and decoder, the Dynamics Engine learns the transition between latent codes from one time step to the next given an action a_t. We first pre-extract the latent codes for each image in the training data, and only learn the transition between the latent codes. All neural network layers described below are followed by a leaky ReLU activation function, except for the outputs of discriminators, the outputs for μ, σ variables used for reparameterization steps, and the outputs for the AdaIN parameters.

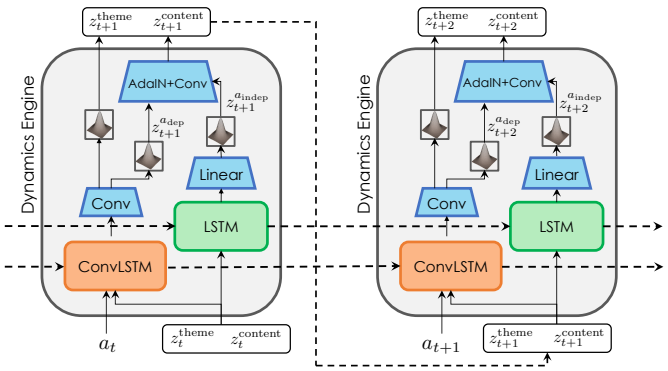

Figure 2. Dynamics Engine produces the next latent codes, given an action and previous latent codes. It disentangles content information into action-dependent and action-independent features with its two separate LSTMs. Dashed lines correspond to temporal connections. Gaussian blocks indicate reparameterization steps.

The major components of the Dynamics Engine are its two LSTM modules. The first one learns the spatial transition between the latent codes and is implemented as a convolutional LSTM module (Figure 2).

$$v_t = \mathcal{F}(\mathcal{H}(h_{t-1}^{\text{conv}}, a_t, z_t^{\text{content}}, z_t^{\text{theme}})) \tag{2}$$

$$i_t, f_t, o_t = \sigma(v_t^i), \sigma(v_t^f), \sigma(v_t^o) \tag{3}$$

$$c_t^{\text{conv}} = f_t \odot c_{t-1}^{\text{conv}} + i_t \odot \tanh(v_t^g) \tag{4}$$

$$h_t^{\text{conv}} = o_t \odot \tanh(c_t^{\text{conv}}) \tag{5}$$

where $h_t^{\text{conv}}, c_t^{\text{conv}}$ are the hidden and cell state of the convLSTM module, and i_t, f_t, o_t are the input, forget, output gates, respectively. $\mathcal{H}$ replicates a_t and z_t^{theme} spatially to match the 4×4 spatial dimension of z_t^{content}. It fuses all inputs by concatenating and running through a 1×1 convolution layer, resulting in a 4×4×48 tensor. $\mathcal{F}$ is composed of two 3×3 convolution layers with a padding of 1, and produces $v_t \in \mathbb{R}^{4\times4\times512}$. v_t is split channel-wise into intermediate variables $v_t^i, v_t^f, v_t^o, v_t^g$. All state and intermediate variables have the same size $\mathbb{R}^{4\times4\times128}$. The hidden state h_t^{conv} goes through two separate convolution layers: 1) 1×1 Conv2d layer that produces 4×4×128 tensor which is split into two chuncks with equal size 4×4×64 and used for the reparameterization step (Eq. 1) to produce $z_{t+1}^{a_{\text{dep}}} \in \mathbb{R}^{4\times4\times64}$, and 2) 4×4 conv2d layer with no padding that produces a 256 dimensional vector; this is also split into two chunks and reparameterized to produce $z_{t+1}^{\text{theme}} \in \mathbb{R}^{128}$.

The second one is a plain LSTM [4] module that only takes z_t as input. Therefore, this module is responsible for information that does not depend on the action a_t. The input z_t is flattened into a vector $\in \mathbb{R}^{1152}$ and goes through five linear layers each outputting 1024-dimensional vectors. The encoded z_t is fed to the LSTM module and all variables inside this module have size $\mathbb{R}^{1024}$. We experimented with both LSTM and GRU [1] but did not observe much difference. The hidden state goes through a linear layer that outputs a 2048-dimensional vector. This vector is split into two chunks for reparmetrization and produces $z_{t+1}^{a_{\text{indep}}} \in \mathbb{R}^{1024}$.

Finally, $z_{t+1}^{a_{\text{dep}}}$ and $z_{t+1}^{a_{\text{indep}}}$ are used as inputs to two $AdaIN$ + Conv blocks.

$$\boldsymbol{\alpha}, \boldsymbol{\beta} = MLP(z_{t+1}^{a_{\text{indep}}}) \tag{6}$$

$$z_{t+1}^{\text{content}} = \mathcal{C}(\mathcal{A}(\mathcal{C}(\mathcal{A}(z_{t+1}^{a_{\text{dep}}}, \boldsymbol{\alpha}, \boldsymbol{\beta})), \boldsymbol{\alpha}, \boldsymbol{\beta})) \tag{7}$$

where we denote convolution and $AdaIN$ layers as $\mathcal{C}$ and $\mathcal{A}$, respectively. The two MLPs (for each block) consist of two linear layers. They produce 64 and 256 dimensional $\boldsymbol{\alpha}, \boldsymbol{\beta}$, respectively. The first 3×3 conv2d layer $\mathcal{C}$ produces 4×4×256 tensor, and the second 3×3 conv2d layer produces $z_{t+1}^{\text{content}} \in \mathbb{R}^{4\times4\times64}$.

Layer	Output dimension
SNLinear + BN	1024
SNLinear + BN	1024
SNLinear + BN	1024
SNLinear + BN	1024
SNLinear + BN	1024
SNLinear	1

Table 7. D_{single} architecture

Layer	Input dimension	Output dimension
SNConv1d	2048×31	128×15
SNConv1d	128×15	256×13
SNConv1d	256×13	512×6

Table 8. $D_{temporal}$ architecture. Input and output dimensions contain two numbers, the first one for the number of channels or vector dimension, and the second one for the temporal dimension. Note that Conv1d is applied on the temporal dimension.

1.2.1 Discriminator

We use disciminators on the flattened 1152 dimensional latent codes z (concatenation of z^{theme} and flattened z^{content}). There are two discriminators 1) single latent discriminator D_{single}, and 2) temporal action-conditioned discriminator $D_{temporal}$.

We denote SNLinear and SNConv as linear and convolution layers with Spectral Normalization [13] applied, and BN as 1D Batch Normalization layers [5]. D_{single} is a 6-layer MLP that tries to discriminate generated z from the real latent codes. It takes a single z as input and produces a single number. For the temporal action-conditioned discriminator $D_{temporal}$, we first reuse the 1024-dimensional feature representation from the fourth layer of D_{single} for each z_t. The represenations for z_t and z_{t-1} are concatenated and go through a SNLinear layer to produce the 1024-dimensional temporal discriminator feature. Let us denote the temporal discriminator feature as $z_{t,t-1}$. The action a_t also goes through a SNLinear layer to produce the 1024-dimensional action embedding. $z_{t,t-1}$ and the action embedding are concatenated and used as the input to $D_{temporal}$. We use 32 time-steps to train DriveGAN, so the input to $D_{temporal}$ has size 2048×31 where 31 is the temporal dimension. Table 8 shows the architecture of $D_{temporal}$. After each layer of $D_{temporal}$, we put a 3-timestep wide convolution layer that produces a single number for each resulting time dimension. Therefore, there are three outputs of $D_{temporal}$ with sizes 14, 11, and 4 which can be thought of as *patches* in the temporal dimension. We also sample negative actions $\bar{a}_t$, and the job of $D_{temporal}$ is to figure out if the given sequence of latent codes is realistic and faithful to the given action sequences. $\bar{a}_t$ is sampled randomly from the training dataset.

1.2.2 Training

We use Adam optimizer with learning rate of 0.0001 for 400,000 optimization steps. We use batch size of 128 each with 32 time-steps and train with a warm-up phase. In the warm-up phase, we feed in the ground-truth latent codes as input for the first 18 time-steps and linearly decay the number to 1 at 100-th epoch, which corresponds to completely autoregressive training at that point. We use the loss $L_{DE} = L_{adv} + L_{latent} + L_{action} + L_{KL}$. L_{adv} is the adversarial losses, and we use the hinge loss [10, 16]. We also add a R_1 gradient regularizer [12] to L_{adv} that penalizes the gradients of discriminators on true data . L_{action} is the action reconstruction loss (implemented as a mean squared error loss) which we obtain by running the temporal discriminator features $z_{t,t-1}$ through a linear layer to reconstruct the input action a_{t-1}. Finally, we add the latent code reconstruction loss L_{latent} (implemented as a mean squared error loss) so that the generated z_t matches the input latent codes, and reduce the KL penalty L_{KL} for $z_t^{a_{\text{dep}}}$,$z_t^{a_{\text{indep}}}$, z_t^{theme}. L_{latent} is weighted by 10.0 and we use different β for the KL penalty terms. We use $\beta^{a_{\text{dep}}} = 0.1, \beta^{a_{\text{indep}}} = 0.1, \beta^{\text{theme}} = 1.0$ for Carla, and $\beta^{a_{\text{dep}}} = 0.5, \beta^{a_{\text{indep}}} = 0.25, \beta^{\text{theme}} = 1.0$ for Gibson and RWD.

2. Additional Analysis on Experiments

LiftSplat [14] proposed a model for producing the Bird's-Eye-View (BEV) representation of a scene from camera images. Section 4.3 in the main text shows how we can leverage LiftSplat to get BEV lane predictions from a simulated sequence from DriveGAN. We can further analyze the qualitative result by comparing how the perception model (LiftSplat) perceives the ground truth and generated sequences differently. We fit a quadratic function to the LiftSplat BEV lane prediction for each image in the ground-truth sequence, and compare the distance between the fitted quadratic and the predicted lanes.

We show results on different look-ahead distances, which denote how far from the ego-car we are making the BEV predictions for. *Random* denotes comparing the distance between the fitted quadratic and the BEV prediction for a randomly sampled RWD sequence. *DriveGAN* denotes the distance for the BEV prediction for the optimized sequence with *differentiable simulation* of DriveGAN. *Ground-Truth* denotes the distance for the BEV prediction for the ground-truth image. Note that *Ground-Truth* is not 0 since the fitted quadratic does not necessarily follow the lane predction from the ground-truth

Model	BEV Prediction Look-ahead Distance			
	25m	50m	75m	100m
Random	0.91m	1.78m	2.95m	4.74m
DriveGAN	0.58m	1.00m	1.70m	2.99m
Ground-Truth	0.31m	0.37m	0.88m	2.07m

Table 9. Mean distance from the BEV lane predictions and fitted quadratic in meters.

image exactly. We can see that DriveGAN-optimized sequences produce lanes that follow the ground-truth lanes. We could find the underlying actions and stochastic variables from a real video through differentiable simulation.

References

[1] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 3

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[6] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017. 2

[7] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020. 2

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

[9] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014. 1, 2

[10] Jae Hyun Lim and Jong Chul Ye. Geometric gan. *arXiv preprint arXiv:1705.02894*, 2017. 4

[11] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013. 1

[12] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018. 4

[13] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018. 4

[14] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. *arXiv preprint arXiv:2008.05711*, 2020. 4

[15] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4570–4580, 2019. 2

[16] Dustin Tran, Rajesh Ranganath, and David Blei. Hierarchical implicit models and likelihood-free variational inference. In *Advances in Neural Information Processing Systems*, pages 5523–5533, 2017. 4

[17] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *CoRR*, abs/1808.06601, 2018. 2

[18] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 2