A. Normalizing Flow Models for Trajectory Forecasting

In this section, we review some preliminaries on normalizing flow based trajectory forecasting models. We refer readers to [29] for a comprehensive review of normalizing flows.

Normalizing flows learn a bijective mapping between a simple base distribution (e.g. Gaussian) and complex target data distribution through a series of learnable invertible functions. In this work, we denote the flow model as f_θ, where θ represents its learnable parameters. The base distribution is a multivariate Gaussian $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^{T\times 2}$, which factorizes across timesteps and (x, y) coordinates. Then, the bijective relationship between $\mathbf{Z}$ and $\mathbf{S}$ is captured by the following forward and inverse computations of f_θ:

$$\mathbf{S} = f_\theta(\mathbf{Z}; \mathbf{o}) \sim p_\theta(\mathbf{S} \mid \mathbf{o}), \quad \mathbf{Z} = f_\theta^{-1}(\mathbf{S}; \mathbf{o}) \sim P_\mathbf{Z} \tag{6}$$

We further impose the structural dependency between $\mathbf{S}$ and $\mathbf{Z}$ to be an invertible autoregressive function, τ_θ, between the stepwise relative *offset* of the trajectory and the corresponding $\mathbf{z}$ sample [35, 13]:

$$\mathbf{s}_t - \mathbf{s}_{t-1} = \tau_\theta(\mathbf{z}_t; \mathbf{z}_{<t})$$

The flow model can be trained using maximum likelihood. Because τ_θ is autoregressive ($\mathbf{z}_t$ does not depend on any $\mathbf{z}_k$ where $k > t$), its Jacobian is a lower-triangular matrix, which admits a simple log-absolute-determinant form [29]. The negative log-likelihood (NLL) objective is

$$\begin{aligned} & -\log p_\theta(\mathbf{S}; \mathbf{o}) \\ =& -\log\left(p(\mathbf{Z})\left|\det \frac{\mathrm{d}f_\theta}{\mathrm{d}\mathbf{Z}}|_{\mathbf{Z}=f_\theta^{-1}(\mathbf{S};\mathbf{o})}\right|^{-1}\right) \\ =& -\left(\sum^{T}\sum^{D}\log p(\mathbf{Z}_{t,d}) - \sum^{T}\sum^{D}\log\left|\frac{\partial \tau_\theta}{\partial \mathbf{Z}_{t,d}}\right|\right) \end{aligned} \tag{7}$$

Once the model is trained, both sampling and *exact* inference are simple. To draw one trajectory sample $\mathbf{S}$, we sample $\mathbf{Z} \sim P_\mathbf{Z}$ and compute $\mathbf{S} = f_\theta(\mathbf{Z}; \mathbf{o})$ Additionally, the exact likelihood of *any* trajectory $\mathbf{S}$ under the model f_θ can be computed by first inverting $\mathbf{Z} = f_\theta^{-1}(\mathbf{S}; \mathbf{o})$ and then computing its transformed log probability via the change of variable formula, as in the second line of Equation 7.

B. Toy Example Details

In the example given in Figure 1(a), we simulate a vehicle either going straight or turning right at the intersection. The vehicle obeys the Bicycle dynamics [23] and uses a MPC-based controller with intermediate waypoints to guide it to its goal. In each simulation, the vehicle starts behind the bottom entrance of the intersection with a fixed initial position perturbed by white noise, and the simulation ends when the vehicle reaches its goal. We collect 100 simulations where the vehicle's final goal is the top exit of the intersection) and 900 simulations where the vehicle's final goal is the right exit of the intersection. They combine into a training dataset of 1000 trajectories. Each trajectory is of 100 simulation steps. We downsample it by a factor of 10 and use the first two steps as the history input to the flow model and the task is to forecast the next 8 steps. With the first 2 steps "burned" in, the vehicle is exactly at the bottom entrance of the intersection, creating a bi-modal dataset that proves to be difficult to model for a normalizing flow model.

After the data collection, we train an autoregressive affine flow as in Appendix F.3 using the entire dataset until the log likelihood converges; Then, we sample 100 times from the flow model using i.i.d unit Gaussian inputs to create trajectories as in Figure 1(b). We train a LDS model with $K = 2$ on top of this flow model and generate 2 samples using one Gaussian noise ϵ to generate the trajectories in Figure 1(c).

C. LDS-CVAE Objective

As shown in our experiments, LDS can also be applied to CVAE models. Doing so requires changing the NLL term in the LDS objective (Equation 3 to ELBO. Because ELBO is a lower bound of the true likelihood, optimizing for this modified objective would still achieve optimizing for the original objective. Formally, let $q_\phi(\mathbf{Z}|\mathbf{o})$ be the approximate latent posterior computed from the encoder and $p_\theta(\mathbf{S}|\mathbf{Z})$ be the likelihood computed from the decoder, we have

$$\mathrm{L}_{\text{LDS-CVAE}}(\psi) \coloneqq \mathrm{ELBO}(\psi) - \lambda_d \mathrm{L_d}(\psi) \tag{8}$$

where $\mathrm{ELBO}(\psi) = \mathbb{E}_{\mathbf{Z}\sim q_\phi(\mathbf{Z}|\mathbf{o})}[\log p_\theta(\mathbf{S}|\mathbf{Z})] - \mathrm{KL}(q_\phi(\mathbf{Z}|\mathbf{o})|p(\mathbf{Z}))$. The $\mathbf{Z}$s fed into CVAE are generated from $r_\psi(\epsilon; \mathbf{o})$, while the prior distribution is $p(\mathbf{Z}) \sim \mathcal{N}(0, \mathbf{I})$.

D. Transductive LDS Algorithms

Algorithm 2 LDS-TD-NN Training

Input: Flow f_θ, Context $\mathbf{o}$
Initialize LDS model r_ψ
for i=1,... **do**
  Sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
  Compute $\mathbf{Z}_1, ..., \mathbf{Z}_K = r_\psi(\epsilon; \mathbf{o})$
  Transform $f_\theta(\mathbf{Z}_1), .., f_\theta(\mathbf{Z}_K)$
  Compute losses using Equations 4 & 5
  Update ψ w.r.t gradient of Equation 3
end for
Compute $\mathbf{Z}_1, ..., \mathbf{Z}_K = r_\psi(\epsilon; \mathbf{o})$
Output: $\mathcal{S} = f_\theta(\mathbf{Z}_1), .., f_\theta(\mathbf{Z}_K)$

Algorithm 3 LDS-TD-P Training

Input: Flow f_θ, Context $\mathbf{o}$

Randomly initialize $\mathbf{Z}_1, ..., \mathbf{Z}_K \sim \mathcal{N}(0, \mathbf{I})$
for i=1,... **do**
  Transform $f_\theta(\mathbf{Z}_1), .., f_\theta(\mathbf{Z}_K)$
  Compute losses using Equations 4 & 5
  Update $\mathbf{Z}_1, ..., \mathbf{Z}_K$ w.r.t gradient of Equation 3
end for

Output: $\mathcal{S} = f_\theta(\mathbf{Z}_1), .., f_\theta(\mathbf{Z}_K)$

E. DLow & DSF Details

Our implementations of DLow and DSF utilizes the same architecture as LDS. The main difference is the loss functions of the two methods. The DLow objective includes three terms:

$$\text{Reconstruction Loss}: E_r(\hat{\mathbf{s}}) = \min_{k \in K} \|\hat{\mathbf{s}}_k - \mathbf{s}\|^2$$

$$\text{Diversity Loss}: E_d(\hat{\mathbf{s}}) = \frac{1}{K(K-1)} \sum_{i \neq j \in K} \exp\Big(- \frac{\|\hat{\mathbf{s}}_i - \hat{\mathbf{s}}_j\|^2}{\sigma_d} \Big)$$

$$\text{KL Loss}: L_{\text{KL}}(\mathbf{z}) = \sum_{k=1}^{K} \text{KL}(p_\psi(\mathbf{z}_k|\mathbf{o}) || p(\mathbf{z}_k)) \tag{9}$$

and the whole objective is:

$$L_{\text{DLow}}(\psi) = \lambda_r E_r + \lambda_d E_d + \lambda_{\text{KL}} L_{\text{KL}}$$

We tune the hyperparameters of DLow and find the following setting to work the best: $\lambda_d = 0.5, \lambda_r = 1, \lambda_{\text{KL}} = 1$, and $\sigma_d = 1$.

DSF is a bit more involved. First, suppose that given input $\mathbf{o}$, DSF network ψ learns a set of latent samples $\mathbf{z}_1, ..., \mathbf{z}_K$, and the flow model decodes them to $\mathbf{s}_1, ..., \mathbf{s}_K$. DSF constructs a deterministic point process (DPP) kernel $\mathbf{L} = \text{Diag}(\mathbf{r}) \cdot \mathbf{S} \cdot \mathbf{Diag}(\mathbf{r})$, where $\mathbf{S}_{ij} = \exp\big(-k \cdot \|\mathbf{s}_i - \mathbf{s}_j\|^2\big)$ for some $k > 0$. Each entry in the quality vector $\mathbf{r}$ is defined as $r_i = \omega \exp\big(- \mathbf{z}_i^\top \mathbf{z}_i + R^2\big)$ if $\|\mathbf{z}_i\| \leq R$; otherwise, $r_i = \omega$. R is chosen as the x-th quantile of the chi-squared distribution with degree of freedom equal to the dimension of $\mathbf{z}_i$. Finally, DSF objective is $L_{\text{DSF}}(\psi) = -\text{trace}(\mathbf{I} - (\mathbf{L}(\psi) + \mathbf{I})^{-1})$, and stochastic gradient is back-propagated with respect to DSF parameters ψ. We choose $k = 1$ and $x = 90$ to be consistent with the original work; however, we find DSF to be sensitive to these hyperparameter choices and fail to scale to large-dimension tasks (e.g. Forking Paths).

F. NuScenes Experimental Details

F.1. Dataset Details

NuScenes [4] is a large urban autonomous driving dataset. The dataset consists of instances of vehicle trajectories coupled with their sensor readings, such as front camera images and lidar scans. The instances are further collected from 1000 distinct traffic scenes, testing forecasting models' ability to generalize. Following the official dataset split provided by the nuScenes development kit, we use 32186 instance for training, 8560 instances for validation, and report results on the 9041 instances in the test set.

F.2. Model Inputs

Model Inputs. All models we implement (AF, CVAE baseline models and MTP) accept the same set of contextual information

$$\mathbf{o} = \{\text{Lidar scans, velocity, acceleration, yaw}\}$$

of the predicting vehicle at time $t = 0$. Below we visualize an example Lidar scan and its histogram version [34] that is fed into the models.

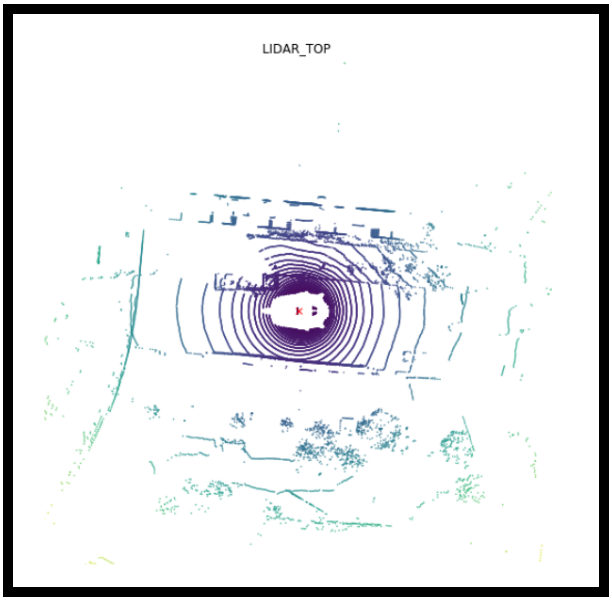

Figure 5: LiDAR inputs in nuScenes.

The Lidar scans are first processed by a pre-trained MobileNet-v128 [17] to produce visual features. These features, concatenated with the rest of the raw inputs, are passed through a neural network to produce input features for the models.

F.3. Autoregressive Affine Flow Details

Our architecture is adapted from the implementation[2] provided in [13]. Here, we describe it in high level and leave the details to the architecture table provided below.

[2] https://github.com/OATML/oatomobile/blob/alpha/oatomobile/torch/networks/sequence.py

AF consists of first a visual module that transforms the observation information $\mathbf{o}$ into a feature vector $\mathbf{h}_0$. Then, $\mathbf{h}_0$ is processed sequentially through a GRU network [8] to produce the per-step *conditioner* $\mathbf{h}_t$ of the affine transformation: $\mathbf{h}_t = \text{GRU}(\mathbf{s}_t, \mathbf{h}_{t-1})$. Finally, we train a neural network (MLP) on top of $\mathbf{h}_t$ to produce the modulators μ, σ of the affine transformation:

$$\begin{aligned}\mathbf{s}_t - \mathbf{s}_{t-1} &= \tau_\theta(\mathbf{z}_t; \mathbf{z}_{<t}) \\ &= \underbrace{\mu_\theta(\mathbf{s}_{1:t-1}, \phi)}_{\text{MLP}_1(\mathbf{h}_t)} + \underbrace{\sigma_\theta(\mathbf{s}_{1:t-1}, \phi)}_{\text{EXP}\left(\text{MLP}_2(\mathbf{h}_t)\right)} \mathbf{z}_t \end{aligned} \tag{10}$$

Table 3: AF Architecture Overview

Attributes	Values
Visual Module	MobileNet(200 × 200 × 3, 128) Linear(128+3,64) Linear(64,64) Linear(64,64)
Autoregressive Module	GRUCell(64)
MLP Module	ReLU ∘ Linear(64,32) Linear(32, 4)
Base Distribution	$\mathcal{N}(0, \mathbf{I})$

F.4. CVAE Details

Our CVAE implementation is adapted from the implementation[3] in [41]. It takes the same set of inputs as our AF model except the addition of a one-frame history input. The history is encoded using a GRU network of hidden size 64 to produce h_0, which is then concatenated with the rest of the inputs. This concatenated vector is then encoded through a 2-layer fully-connected network. To encode the future, our CVAE model uses a GRU network of the same architecture as the GRU encoder for the history. Finally, the encoded input and output (i.e. future) is concatenated and passed through another 2-layer network to give the mean and the variance of the approximate posterior distribution. For the decoder, we first sample a latent vector z using the reparameterization trick. Then, z is concatenated with the encoded inputs to condition the per-step GRU roll-out of the reconstructed future. The model is trained to maximize ELBO.

[3] https://github.com/Khrylx/DLow/blob/master/models/motion_pred.py

Table 4: CVAE Architecture Overview

Attributes	Values
History Encoder	GRU(2, 64)
Visual Module	MobileNet(200 × 200 × 3, 128)
Full Input Encoder	Linear(128+64+3, 64) Linear(64, 64) Linear(64, 64)
Full Output Encoder	GRU(2, 64)
Input Output Merger	Linear(64+64, 64) Linear(64, 32+32)
μ, σ	$\mathbb{R}^{32}, \mathbb{R}^{32}$
Decoder	GRU(2+32+64, 64) Linear(64, 64) Linear(64, 32, 2)

F.5. LDS Architecture Details

LDS r_ψ is a single multi-layer neural network with K heads, the number of modes pre-specified. To ensure stable training, we clip the diversity loss to be between $[0, 40]$ for $K = 5$ and $[0, 30]$ for $K = 10$. DSF and DLow use the same architecture as LDS.

Table 5: LDS Architecture and Hyperparameters Overview

Attributes	Values
LDS Architecture	Linear(Input Size, 64) Linear(64,32) Linear(32, 2 × T × K)
Learning Rate	0.001
λ_d	1
Diversity function clip value	40/30

F.6. Training Details

We train the "backbone" forecasting models AF, CVAE, MTP-Lidar for 20 epochs with learning rate 10^{-3} using Adam [20] optimizer and batch size 64. LDS-AF iterates through the full training set once, while LDS-AF-TD directly optimizes on the test set with a minibatch size of 64 and 400 adaptation iterations for every minibatch. For all LDS models, we set $\lambda_d = 1$ and do not experiment with further hyperparameter tuning. LDS training also uses Adam. For all models, we train 5 separate models using random seeds and report the average and standard deviations in our results.

F.7. Additional Results

Comparison with Trajectron++ To test the limit of our approach, we additionally compare against Trajectron++ [36]. Different from other baselines we include in the main paper, Trajectron++ utilizes additional inputs such as spatio-temporal graph and vehicle/pedestrian dynamics and evaluates on a shorter horizon (i.e. 8 frames). Here, we re-train an AF model by truncating all trajectories to 8 frames, and then train a LDS model on top, giving us LDS-AF-8. Trajectron++ only reports minFDE_{10} and obtains 2.24. LDS-AF-8 obtains $\text{minFDE}_{10} : 2.28, \text{minADE}_{10} : 1.09, \text{minASD}_{10} : 2.18, \text{minFSD}_{10} : 5.80$, where we include the other three metrics for completeness. As shown, LDS-AF-8 is slightly worse than Trajectron++ on minFDE_{10}; however, giving that the backbone model LDS utilizes here is a much weaker model than Trajectron++. this result is encouraging and suggests that even a weak model can produce competitive predictions when it is augmented with an effective sampling mechanism.

minASD vs. meanASD. Here, we illustrate the robustness of minASD compared to meanASD. Consider the following two sets of 10 predictions. In the first set, the pairwise distance between all pairs is exactly 1. In the second set, 9 predictions are identical, but their distance to the remaining one is 100. The first set achieves identical diversity value 1 under the two metrics, whereas the second set achieves 0 minASD but $\frac{20}{9}$ meanASD. Therefore, we might incorrectly conclude that the second set is more diverse if we were to solely rely on mean metrics for diversity.

Mean diversity results. Here, we report the meanASD and meanFSD metrics for diversity. As shown, LDS still achieves the highest diversity on these metrics and provide greater diversity boost than DLow; however, the relative differences among models are much smaller. Additionally, by examining the tables from $K = 5$ to $K = 10$, we no longer find the pattern that LDS being the only model whose diversity does not deteriorate as we did in Table 1 (Right). This is because the mean metric only captures the average behavior and not the worst case behavior. Thus, our hypothesis that the min metrics are more informative than the mean metrics are supported by the following results.

Method	Samples	meanASD (↑)	meanFSD (↑)
MTP-Lidar-5	5	5.74 ± 0.79	13.80 ± 2.00
CVAE	5	5.38 ± 0.09	12.28 ± 0.16
DLow-CVAE	5	6.66 ± 0.21	15.43 ± 0.44
AF	5	6.21 ± 0.02	14.48 ± 0.04
DLow-AF-5	5	**7.41** ± 0.29	**17.90** ± 0.67
LDS-AF-5	5	**7.89** ± 0.29	**19.06** ± 0.58

Method	Samples	meanASD (↑)	meanFSD (↑)
MTP-Lidar-10	10	5.24 ± 0.30	12.71 ± 0.59
CVAE	10	5.38 ± 0.10	12.28 ± 0.18
DLow-CVAE	10	6.96 ± 0.20	16.16 ± 0.45
AF	10	6.21 ± 0.01	14.48 ± 0.04
DLow-AF-10	10	**7.88** ± 0.57	**19.49** ± 1.36
LDS-AF-10	10	**7.90** ± 0.28	**19.71** ± 0.74

Table 6: NuScenes prediction mean diversity results.

F.8. Additional Ablation Results

In this section, we provide some additional ablation studies to further understand the effectiveness of LDS. First, we aim to understand: **how sensitive is LDS to the quality of the underlying flow model?** To answer this question, we train an additional AF model with half the number of epochs as the original one (**AF**$^-$), and then train LDS as before. The comparisons are shown in Table 7. With only half the training time, AF$^-$ performs considerably worse than AF, yet LDS is still able to provide a significant performance boost, achieving **33**% reduction in both minADE and minFDE. This reduction is greater than that of LDS applied to the stronger AF model (27%), suggesting the utility of LDS is greater for weaker pre-trained model and highlighting that its overall effectiveness is robust to the quality of the underlying flow.

Method	S	mADE_5	↓%	mFDE_5	↓%	minASD	↑%	minFSD	↑%
AF$^-$	5	3.49 ± 0.16	-	7.79 ± 0.41	-	1.99 ± 0.15	-	4.58 ± 0.46	-
LDS-AF$^-$	5	2.31 ± 0.19	**34**%	5.17 ± 0.39	**34**%	2.91 ± 0.05	146%	8.25 ± 0.32	180%
LDS-AF$^-$-TD	5	2.35 ± 0.16	**33**%	5.24 ± 0.33	**33**%	3.00 ± 0.16	151%	8.36 ± 0.14	183%
AF	5	2.86 ± 0.01	-	6.26 ± 0.05	-	1.58 ± 0.02	-	3.75 ± 0.04	-
LDS-AF	5	2.06 ± 0.09	28%	4.67 ± 0.25	25%	3.13 ± 0.18	98%	8.19 ± 0.26	118%
LDS-AF-TD	5	2.06 ± 0.02	28%	4.62 ± 0.07	26%	3.09 ± 0.07	95%	8.15 ± 0.17	117%

Table 7: LDS ablations on the pre-trained flow models.

LDS training stability. We track the state of minADE and minASD on the mini nuScenes validation set over the course of LDS training. On the mini version of the nuScenes dataset, we train a LDS-AF ($K = 5$) model, and for every training iteration, we compute the current LDS-AF model's minADE and minASD on the mini validation set. The mini version is a much smaller dataset with only 1000 instances, making this procedure manageable. The sequence of (ASD, ADE) pairs are traced as a trajectory on a 2D plane, where the x-axis corresponds to minASD_5 and the y-axis corresponds to minADE_5.

The entire trajectory is visualized in Figure 6. We additionally visualize both minADE and minASD individually over the course of training in the middle and the right panels. In all three plots, the initial AF model's (ASD, ADE) point is colored in red, and the final LDS-AF's (ASD,ADE) point is colored in blue. Focusing on the left panel, we

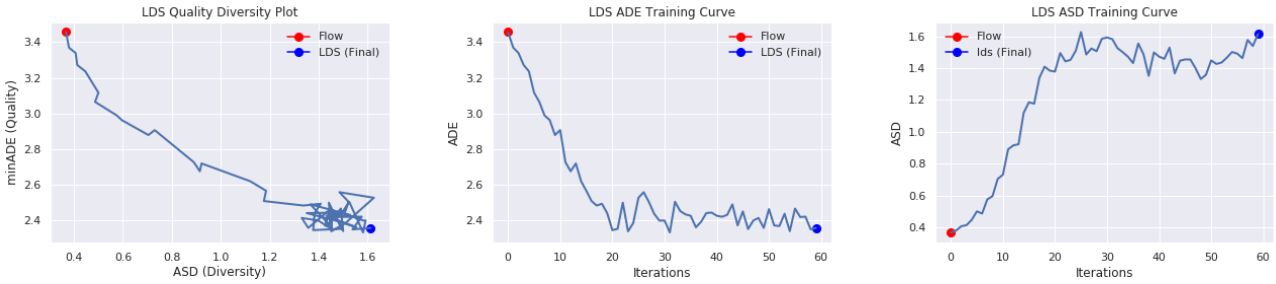

Figure 6: **Left**: LDS (ASD, ADE) plot on nuScenes mini. LDS offers stable and fast improvements to flow outputs in both accuracy and diversity during its training. **Middle**: LDS ADE over the course of training. **Right**: LDS minASD over the course of training.

note that the initial point at the top left corner represents the ADE/ASD of the pre-trained flow backbone, which has relatively high error and low diversity. However, LDS quickly discovers good sampling distribution and offers fast and near "monotonic" improvements along both axis. This provides evidence that the joint objective is effective and helps avoiding potential local minima in the loss landscape. The rapid improvement early on in training also explains why LDS's transductive adaptation procedure can be effective. Finally, an early stopping mechanism may be effective given the fast convergence to the optimum (i.e. the bottom right corner); we leave it to future work to investigate the precise stopping criterion.

Dependence on ϵ. One may eliminate the dependence on ϵ by making the mapping procedure deterministic: $\{\mathbf{Z}_1, ..., \mathbf{Z}_K\} = r_\psi(\mathbf{o})$. In theory, this simplified formulation should optimize for the same objective as the objective itself does not explicitly depend on ϵ. However, in practice, we find this ablation reduces performance as the trained LDS model may be overfitting to some deterministic set of trajectories to multiple different inputs. We report the average results over 5 seeds on LDS-AF for $K = 10$ where we do not utilize the ϵ sampling procedure (Step 3 and 4 in Algorithm 1): $\text{minADE}_{10} : 1.74, \text{minFDE}_{10} : 3.73, \text{minASD}_{10} : 2.06, \text{minFSD}_{10} : 6.03$. These results are slightly worse than the original LDS-AF results reported in Table 1. Therefore, we confirm that adding innate stochasticity to the training procedure with ϵ boosts performance, validating our original formulation.

Mismatched K. Here, we perform a controlled experiment analyzing LDS outputs when the number of samples K it learns to output mismatches the number of modes in the underlying distribution. To do this, we return to our toy intersection environment as in Figure 1(a). Instead of training a LDS with $K = 2$ as done in Figure 1(c), we train one with $K = 5$. This LDS's set of 5 samples are shown in Figure 7. Among the 5 trajectories, the two modes are still captured. Importantly, the major mode (turning right) is captured twice. However, the remaining two trajectories represent the average of the two modes and correspond to potentially unrealistic behaviors. Note that this set of five trajectories would achieve very low minADE errors in both single and multiple-future evaluations, since at least one trajectory in the predictions is close to both modes. However, for planning settings, this set of predictions may not be optimal. Hence, choosing K carefully is important in practice, and we leave it to future work for further investigations.

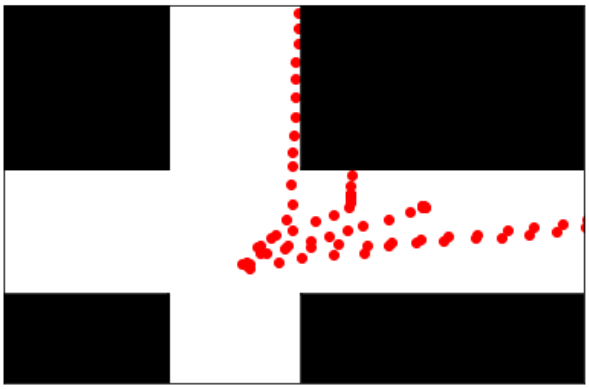

Figure 7: LDS outputs for $K = 5$.

F.9. Additional Visualizations

LDS-AF vs. AF vs. MTP-Lidar. Additional visualizations of LDS-AF, AF, and MTP-Lidar outputs (Figure 10).

LDS-AF vs. LDS-AF-TD-NN. Visualizations of LDS-AF (red) vs. LDS-AF-TD-NN (blue) are provided in Figure 8. As shown, LDS-AF-TD-NN generally adapt its predictions to the current observation better and produce more realistic trajectories with respect to the ground truth.

Varying ϵ. Finally, we visualize different sets of LDS-AF

trajectories by randomly sampling different $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. The visualizations are in 9.

G. Forking Paths Experimental Details

G.1. Dataset Details

Here, we briefly describe the Forking Paths evaluation dataset [25]. FP semi-automatically reconstruct static scenes and their dynamic elements (e.g. pedestrians) from real-world videos in ActEV/VIRAT and ETH/UCY in the CARLA simulator. To do so, it converts the ground truth trajectory annotations from the real-world videos to coordinates in the simulator using the provided homography matrices of the datasets. Then, 127 "controlled agents" (CA) are selected from the 7 reconstructed scenes. For each CA, there are on average 5.9 human annotators to control the pedestrian to the pre-defined destinations in a "natural" way that mimics real pedestrian behaviors. The scenes are also rendered to the annotators in different camera angles ranging from 'Top-Down' to '45-Degree'. The annotation can last up to 10.4 seconds, which is far longer than the 4.8 seconds prediction window in the original datasets, making the forecasting problem considerably harder. In total, there are 750 trajectories after data cleaning.

G.2. CAM-NF Model Details

Compared to the perception-based flow model we use in nuScenes, CAM-NF only uses the historical trajectory of the agent and other surrounding agents in the scene, consistent with various prior methods [15, 1] in pedestrian forecasting; we leave exploring the utility of added perception modules as in [26, 25] to future work. Hence, $\mathbf{o} = \{\mathbf{S}^a_{-T_{\text{hist}}:0}\}_{a=1}^{A}$, where A is the number of pedestrians in the scene and T_{hist} the length of history. CAM-NF first encodes the history of all pedestrians in the scene using a LSTM encoder. Then, it computes cross-pedestrian attention feature vectors using self-attention [39] to model the influences of nearby pedestrians, and uses these attention vectors as features for a normalizing flow decoder, which outputs future trajectories of the predicted pedestrian. The decoder is an autoregressive affine flow similar to the one used for nuScenes. Here, we briefly describe the architectures and refer interested readers to the original work.

The encoder first uses an LSTM of hidden dimension 512 [16] to extract a history embedding for every agent up to the most recent timestep $t = 0$:

$$h^a_t = \text{LSTM}(\mathbf{s}^a_{t-1}, h^a_{t-1}) \quad t = T_{\text{hist}} - 1, ..., 0$$

Then, the history embedding of all agents $h^0_t, ..., h^A_t$ are aggregated to compute a corresponding cross-agent attention embedding using self-attention. This embedding is then combined with the history embedding to form the inputs to the normalizing flow decoder:

$$\tilde{h}^a = h^a_0 + \text{SELF-ATTENTION}(Q^a, \mathbf{K}, \mathbf{V})$$

where (Q^a, K^a, V^a) is the query-key-value triple for each agent, and the bold versions are their all-agent aggregated counterparts. Note that $\tilde{h}^a$ is passed through a linear layer of hidden dimension 256 before passed to the decoder.

The decoder is similar to the autoregressive affine flow used for nuScenes with a few minor changes. First, $\sigma_\theta \in \mathbb{R}^{2\times 2}$ to model correlation between the two dimensions (i.e. x, y) of the states. Second, a velocity smoothing term $\alpha(\mathbf{s}_{t-1} - \mathbf{s}_{t-2})$ is added to the step-wise update. That is,

$$\begin{aligned}\mathbf{s}_t - \mathbf{s}_{t-1} &= \tau_\theta(\mathbf{z}_t; \mathbf{z}_{<t}) \\ &= \alpha(\mathbf{s}_{t-1} - \mathbf{s}_{t-2}) + \underbrace{\mu_\theta(\mathbf{s}_{1:t-1}, \phi)}_{\text{MLP}_1(\mathbf{h}_t)} + \underbrace{\sigma_\theta(\mathbf{s}_{1:t-1}, \phi)}_{\text{EXP}\left(\text{MLP}_2(\mathbf{h}_t)\right)} \mathbf{z}_t\end{aligned} \tag{11}$$

We set $\alpha = 0.5$ as in the original work. Our implementation is adapted from the original implementation[4].

G.3. LDS Architecture Details

Since trajectories in FP have different lengths, we set $T = 25$ in the LDS architecture to ensure that a bijective mapping between $\mathbf{Z} \in \mathbb{R}^{T\times D}$ exists for all samples in FP. This also means that during training, we optimize LDS for up to 25 steps. We also slightly increase the size of LDS neural network and set the maximum diversity loss to be 80, which we find to work well empirically. As before, DLow and DSF use the same architecture as LDS. The whole architecture is as follows:

Table 8: LDS Architecture and Hyperparameters Overview

Attributes	Values
LDS Architecture	Linear(Input Size, 128) Linear(128,64) Linear(64, 2 × 25 × K)
Learning Rate	0.001
λ_d	10
Diversity function clip value	80

G.4. Training Details

We train CAM-NF for 200 epochs with learning rate 10^{-3} using Adam optimizer and batch size 64. We train LDS, LDS-TD, and DLow models using mode hyperparameter $K = 20$. LDS and DLow iterate through the full training set once, while LDS-AF-TD directly optimizes on the test set with a minibatch size of 64 and 200 adaptation iterations for every minibatch. For all LDS models, through

[4] https://github.com/kami93/CMU-DATF

a hyperparameter search , we set $\lambda_d = 10$. For all models, we train 5 separate models using random seeds and report the average and standard deviations.

G.5. ActEV/VIRAT Results

Method	$\text{minADE}_1(\downarrow)$	$\text{minFDE}_1(\downarrow)$
Linear*	32.19	60.92
LSTM*	23.98	44.97
Social-LSTM*	23.10	44.27
Social-GAN*	23.10	44.27
Next*	19.78	42.43
Multiverse*	**18.51**	**35.84**
CAM-NF	19.69 ± 0.15	39.12 ± 0.31

Table 9: Training Results on ActEV/VIRAT. Our backbone flow model CAM-NF achieves comparable performance to the current state-of-art Multiverse.

G.6. Forking Paths Full Sub-Category Split Results

Method	$\text{minADE}_{20}(\downarrow)$		$\text{minFDE}_{20}(\downarrow)$	
	45-Degree	Top Down	45-Degree	Top Down
Linear*	213.2	197.6	403.2	372.9
LSTM*	201.0 ± 2.2	183.7 ± 2.1	381.5 ± 3.2	355.0 ± 3.6
Social-LSTM* [1]	197.5 ± 2.5	180.4 ± 1.0	377.0 ± 3.6	350.3 ± 2.3
Social-GAN* [15]	187.1 ± 4.7	172.7 ± 3.9	342.1 ± 10.2	326.7 ± 7.7
Next* [26]	186.6 ± 2.7	166.9 ± 2.2	360.0 ± 7.2	326.6 ± 5.0
Multiverse* [25]	168.9 ± 2.1	157.7 ± 2.5	333.8 ± 3.7	316.5 ± 3.4
CAM-NF [30]	155.2 ± 2.4	140.8 ± 2.2	305.0 ± 4.6	282.2 ± 4.9
DSF [40]	169.7 ± 1.8	155.77 ± 2.1	331.7 ± 3.7	309.5 ± 3.5
DLow [41]	144.5 ± 3.8	131.0 ± 8.1	284.6 ± 8.4	262.1 ± 20.5
LDS (Ours)	**103.8** ± 6.9	**93.4** ± 4.8	**190.6** ± 16.3	**173.4** ± 12.8
LDS-TD-NN (Ours)	**105.1** ± 4.3	**94.9** ± 2.1	**188.7** ± 10.4	**167.4** ± 4.8

Table 10: Evaluation results on Forking Paths. LDS-augmented CAM-NF significantly outperforms all other methods, including Multiverse and DLow-augmented CAM-NF.

G.7. Additional Analysis on Forking Paths

As shown in Table 2, compared to nuScenes experiments, LDS exhibits larger improvement over DLow on the FP dataset. We believe that this is because the ground truth futures in the FP test set tend to be longer than in the ActEV/VIRAT training set. Since DLow optimizes for the L_2 reconstruction loss between its forecasts and the ground-truth future trajectories in the *training* set, it is limited to improving diversity over the horizon of these training trajectories. Thus, it is unable to produce diverse predictions for longer horizon trajectories, such as those in the test set. In contrast, since LDS directly optimizes the likelihood of future trajectory according to the flow model, it does not rely on ground truth futures. Thus, it can improve diversity over much longer time horizons than in the training data. This contrast further highlights the flexibility of LDS. Finally, we note that we were not able to achieve positive results for DSF; this is likely due to the much larger latent sample dimension $(2 \times 20 = 40)$ for this dataset, which as stated in Section 2, would be an issue for DSF.

G.8. Additional Visualizations

In this section, we provide some additional visualizations of LDS, CAM-NF, and Multiverse outputs (Figure 11).

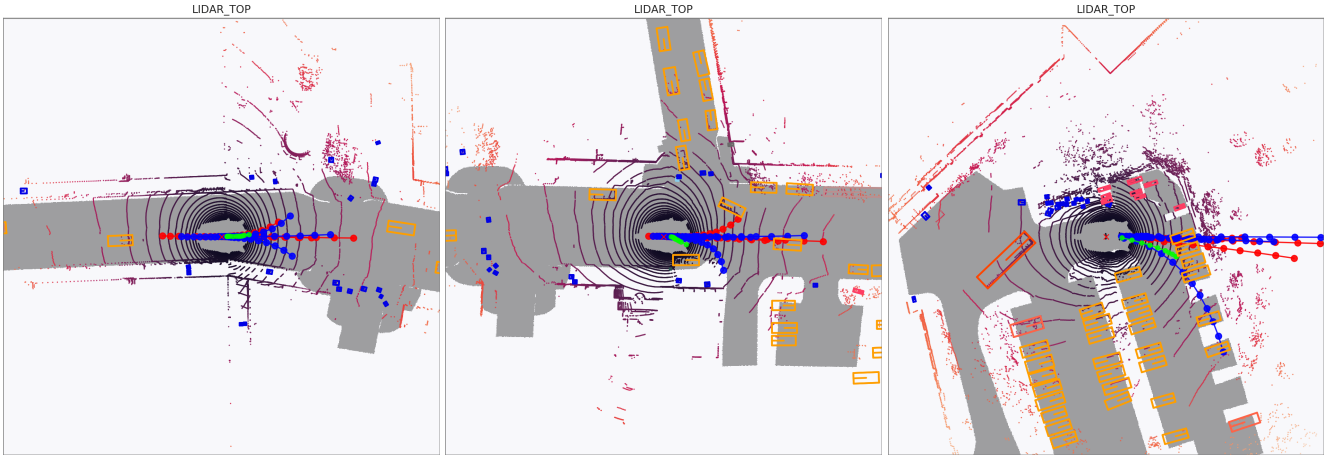

Figure 8: **LDS-AF** (red) vs. **LDS-AF-TD-NN** (blue) in NuScenes.

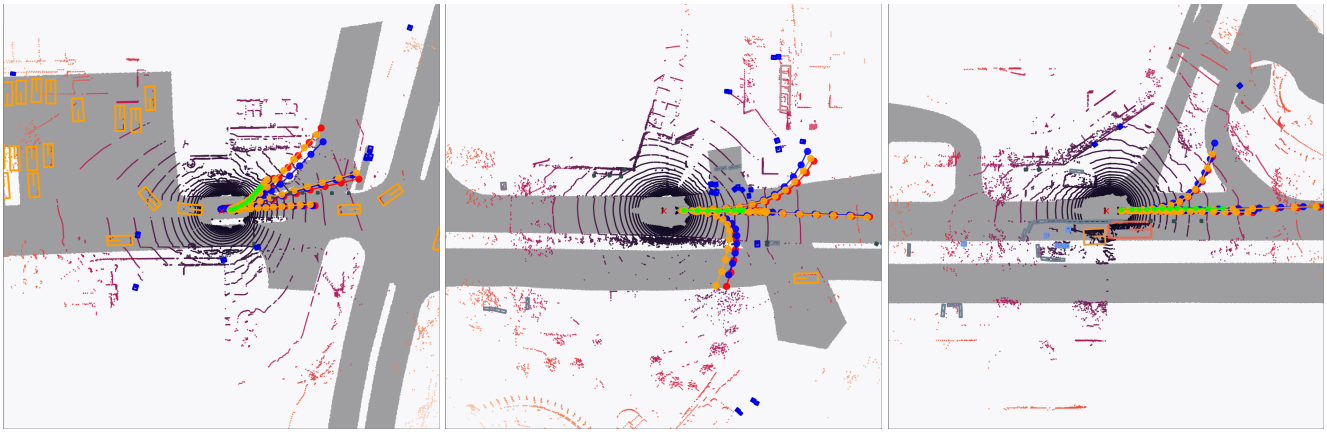

Figure 9: Effect of varying ϵ on **LDS-AF** in NuScenes. The three colors (red, blue, orange) denote three different sets of trajectories by randomly sampling ϵ.

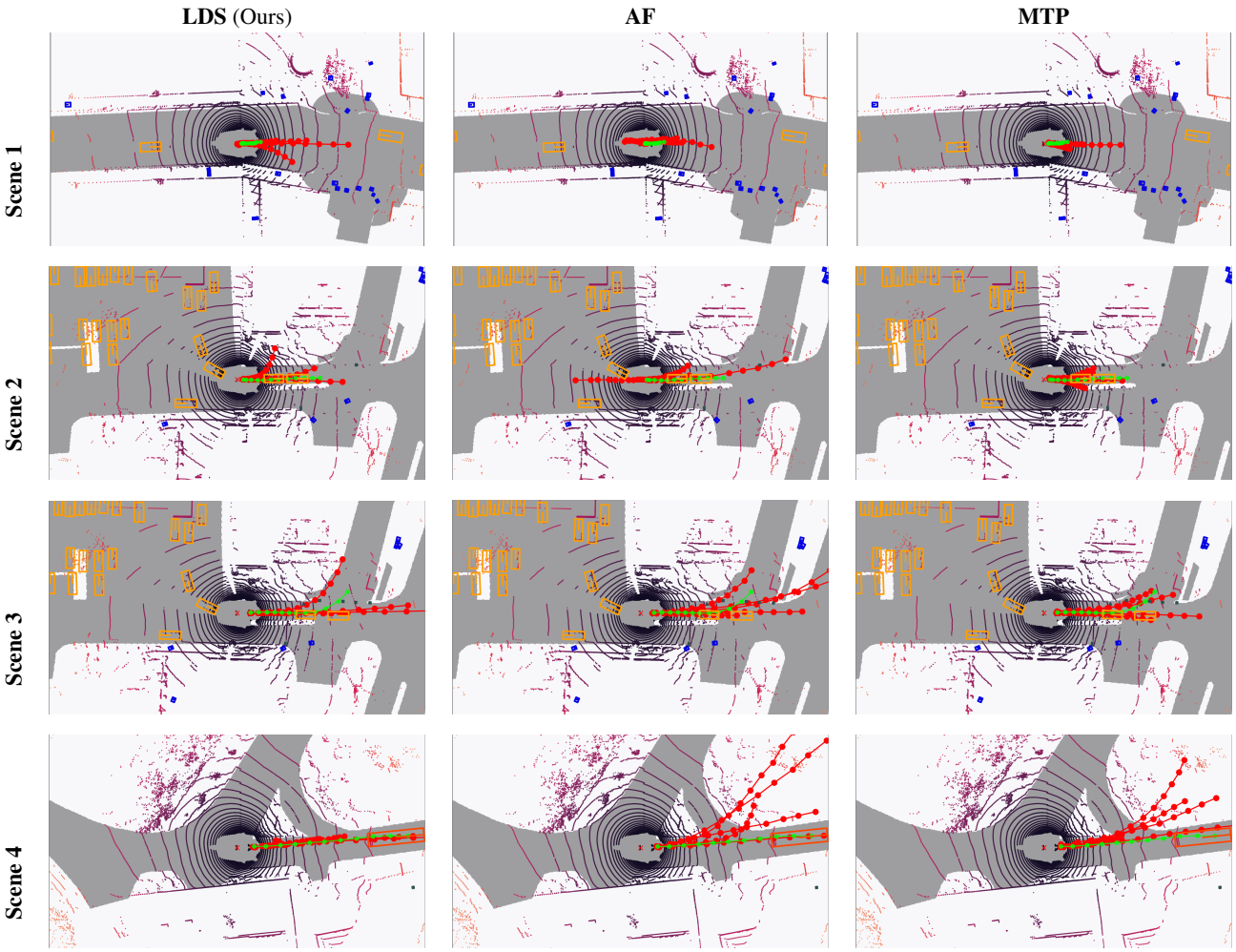

Figure 10: Additional model visualizations. The models from left to right: **LDS**, **AF**, and **MTP-Lidar**.

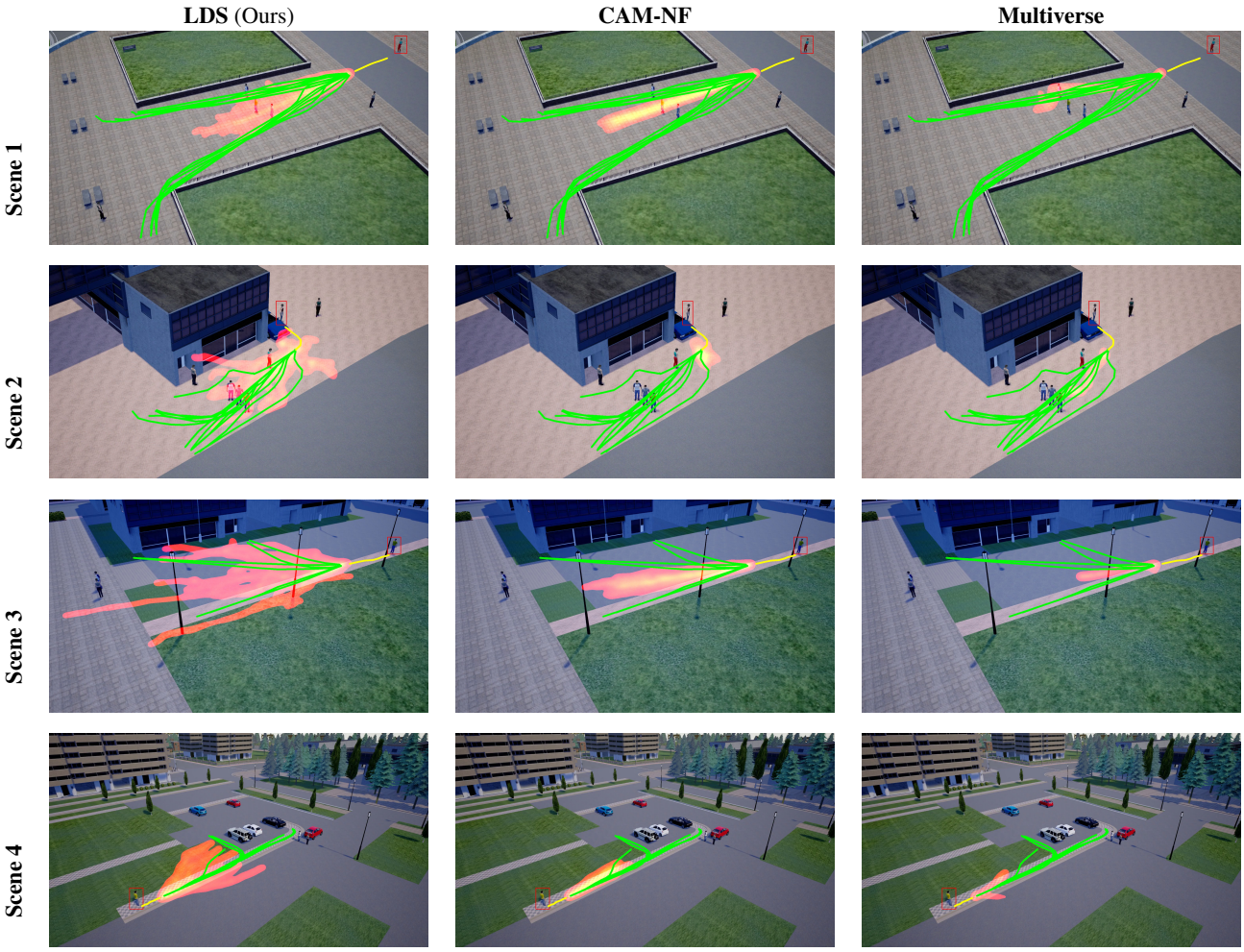

Figure 11: Additional model visualizations. The models from left to right: **LDS**, **CAM-NF**, and **Multiverse**.