A PROOFS

A.1 PROOF OF THEOREM 1

Proof. Let α and $(1-\alpha)$ be the prior probabilities of positive and negative examples under $p_\theta(\mathbf{x}_t; t)$. Note that α remains independent of t because

$$\begin{aligned}\alpha &= \int p_\theta(\mathbf{x}_t; t) p_\theta(\mathbf{x}_0|\mathbf{x}_t) p_\theta(y=1|\mathbf{x}_0)\, d\mathbf{x}_0\, d\mathbf{x}_t = \int p_\theta(\mathbf{x}_t; t) p_\theta(\mathbf{x}_0|\mathbf{x}_t)\mathcal{O}(\mathbf{x}_0)\, d\mathbf{x}_0\, d\mathbf{x}_t \\ &= \int p_\theta(\mathbf{x}_0, \mathbf{x}_t; t)\mathcal{O}(\mathbf{x}_0)\, d\mathbf{x}_0 d\mathbf{x}_t = \int p_\theta(\mathbf{x}_0; 0)\mathcal{O}(\mathbf{x}_0)\, d\mathbf{x}_0.\end{aligned}$$

The objective in Eq. (9) is equal to

$$\begin{aligned}&-\mathbb{E}_t\left[\mathbb{E}_{p_\theta(\mathbf{x}_t)}\left[\mathbb{E}_{p_\theta(\mathbf{x}_0|\mathbf{x}_t)}\left[\mathcal{O}(\mathbf{x}_0)\right]\log C_\phi(\mathbf{x}_t; t) + \mathbb{E}_{p_\theta(\mathbf{x}_0|\mathbf{x}_t)}\left[(1-\mathcal{O}(\mathbf{x}_0))\right]\log(1-C_\phi(\mathbf{x}_t; t))\right]\right] \\ &\quad = -\mathbb{E}_t\left[\mathbb{E}_{p_\theta(\mathbf{x}_t)}\left[p(y=1|\mathbf{x}_t)\log C_\phi(\mathbf{x}_t; t) + p(y=0|\mathbf{x}_t)\log(1-C_\phi(\mathbf{x}_t; t))\right]\right] \quad (14)\end{aligned}$$

This is equivalent to Eq. (6) after replacing q with p_θ, i.e. sampling from the reverse process instead of the forward. Therefore, its optimal solution follows Eq. (7). Hence,

$$\begin{aligned}&\nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t; t) + \nabla_{\mathbf{x}_t}\log C_{\phi^*}(\mathbf{x}_t; t) \\ &= \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t; t) + \nabla_{\mathbf{x}_t}\log \alpha p_\theta(\mathbf{x}|y=1; t) \\ &\qquad - \nabla_{\mathbf{x}_t}\log\Big[\overbrace{\alpha p_\theta(\mathbf{x}_t|y=1; t) + (1-\alpha)p_\theta(\mathbf{x}_t|y=0; t)}^{p_\theta(\mathbf{x}_t; t)}\Big] \\ &= \nabla_{\mathbf{x}_t}\log \alpha p_\theta(\mathbf{x}_t|y=1; t) = \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t|y=1; t)\end{aligned}$$

□

A.2 PROOF OF COROLLARY 1.1

Proof. Since p_{θ,ϕ^*} is defined as the distribution generated by simulating the SDE in equation 2, its score function $\nabla_{\mathbf{x}_t}\log p_{\theta,\phi^*}(\mathbf{x}_t; t)$ is by definition equal to $s_{\theta,\phi^*}(\mathbf{x}_t; t)$ (Risken & Risken, 1996; Song & Ermon, 2019). Similarly for the baseline DM we have $s_\theta(\mathbf{x}_t; t) = \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t; t)$. Therefore,

$$\begin{aligned}\nabla_{\mathbf{x}_t}\log p_{\theta,\phi^*}(\mathbf{x}_t; t) = s_{\theta,\phi^*}(\mathbf{x}_t; t) &\overset{\text{Eq. (8)}}{=\!=} s_\theta(\mathbf{x}_t; t) + \nabla_{\mathbf{x}_t}\log C_{\phi^*}(\mathbf{x}_t; t) \quad (15)\\ &= \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t; t) + \nabla_{\mathbf{x}_t}\log C_{\phi^*}(\mathbf{x}_t; t) \overset{\text{Thm. 1}}{=\!=} \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t|y=1) \quad (16)\end{aligned}$$

Here we derived $s_{\theta,\phi^*}(\mathbf{x}_t; t) = \nabla_{\mathbf{x}_t}\log p_\theta(\mathbf{x}_t|y=1)$. By (Anderson, 1982), we proved the first statement.

The second statement follows by decomposing $p_\theta(\mathbf{x}_t|y=1)$:

$$p_{\theta,\phi^*}(\mathbf{x}) = p_\theta(\mathbf{x}|y=1) \propto p_\theta(\mathbf{x})\mathcal{O}(\mathbf{x}) \quad \Rightarrow \quad p_{\theta,\phi^*}(\mathbf{x}) = 0 \quad \forall \mathbf{x} \in \Omega^\complement.$$

For the last statement, we have

$$\left.\begin{aligned}p_{\theta,\phi^*}(\mathbf{x}) &= \tfrac{p_\theta(\mathbf{x})\mathcal{O}(\mathbf{x})}{\int p_\theta(\mathbf{x})\mathcal{O}(\mathbf{x})\, d\mathbf{x}} \\ \int p_\theta(\mathbf{x})\mathcal{O}(\mathbf{x})\, d\mathbf{x} &\leq 1\end{aligned}\right\} \Rightarrow p_{\theta,\phi^*}(\mathbf{x}) \geq p_\theta(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega$$
$$\overset{\mathcal{D}\subseteq\Omega}{\Longrightarrow} \log p_{\theta,\phi^*}(\mathcal{D}) = \sum_{\mathbf{x}\in\mathcal{D}}\log p_{\theta,\phi^*}(\mathbf{x}) \geq \sum_{\mathbf{x}\in\mathcal{D}}\log p_\theta(\mathbf{x}) = \log p_\theta(\mathcal{D}).$$

□

We demonstrate this corollary with a one-dimension density in Fig. 4. We show a "base distribution" $p(x)$ and the positive and negative regions Ω and $\Omega^\complement$, respectively. We can see that the distribution $p(x|y=1)$ assigns no mass to $\Omega^\complement$ and has a larger mass assigned to any point in Ω.

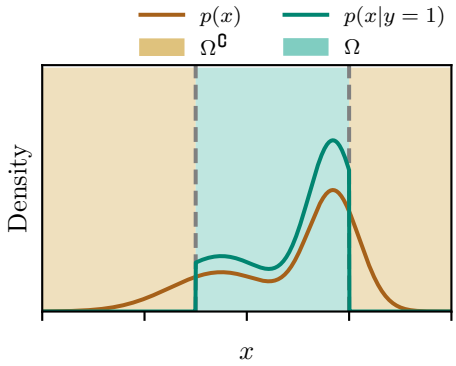


Figure 4: We use this one-dimensional density plot to show how we guide the generation process towards the positive support region indicated by the oracle. The original density curve $p(x)$ is a mixture of two Gaussian distributions, and we depicts the region in cyan as the eligible support restricted by oracle function, otherwise in buff. Ideally, the well-behaved Bayes optimal classifier helps to target exactly the score function of oracle-approved examples such that the resulting density function $p(x|y=1)$ only lands in the cyan area and assigns zero probability outside this region.

A.3 Equivalence of cross-entropy loss minimization and KL divergence minimization

This is a well-known result in the literature. Nonetheless, we include the result and its proof here for completeness and ease of reference.

Claim: minimizing the cross-entropy loss between the classifier output and the true label is equivalent to minimizing the KL divergence between the classifier output and the Bayes optimal classifier.

Proof. The Bayes optimal classifier $C^*(\mathbf{x}_t; t)$ approximates $q(y = 1|\mathbf{x}_t)$. Let $p_\phi(y|\mathbf{x}_t)$ be the distribution represented by the learned classifier $C_\phi(\mathbf{x}_t; t)$ i.e., $p_\phi(y = 1|\mathbf{x}_t) = C_\phi(\mathbf{x}_t; t)$. For an arbitrary diffusion time step t, the expected KL divergence between the Bayes optimal and the learned classifier therefore is

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{x}_t)}\left[\mathrm{KL}\left(q(y|\mathbf{x}_t)||p_\phi(y|\mathbf{x}_t)\right)\right] \\
&= \mathbb{E}_{q(\mathbf{x}_t)}\left[\mathbb{E}_{q(y|\mathbf{x}_t)}\left[q(y|\mathbf{x}_t)\log\frac{q(y|\mathbf{x}_t)}{p_\phi(y|\mathbf{x}_t)}\right]\right] && (17)\\
&= \mathbb{E}_{q(\mathbf{x}_t)}\left[q(y=1|\mathbf{x}_t)\log\frac{q(y=1|\mathbf{x}_t)}{C_\phi(\mathbf{x}_t;t)} + q(y=0|\mathbf{x}_t)\log\frac{q(y=0|\mathbf{x}_t)}{1-C_\phi(\mathbf{x}_t;t)}\right] && (18)\\
&= \mathbb{E}_{q(\mathbf{x}_t)}\left[H(q(y|\mathbf{x}_t))\right] - \mathbb{E}_{q(\mathbf{x}_t)}\left[q(y=1|\mathbf{x}_t)\log C_\phi(\mathbf{x}_t;t) + q(y=0|\mathbf{x}_t)\log(1-C_\phi(\mathbf{x}_t;t))\right]. && (19)
\end{aligned}
$$

The first term is the expected entropy of the optimal classifier and is independent of ϕ. Therefore,

$$
\begin{aligned}
&\arg\min_\phi \mathbb{E}_{q(\mathbf{x}_t)}\left[\mathrm{KL}\left(q(y|\mathbf{x}_t)||p_\phi(y|\mathbf{x}_t)\right)\right] \\
&= \arg\min_\phi -\mathbb{E}_{q(\mathbf{x}_t)}\left[q(y=1|\mathbf{x}_t)\log C_\phi(\mathbf{x}_t;t) + q(y=0|\mathbf{x}_t)\log(1-C_\phi(\mathbf{x}_t;t))\right] && (20)\\
&= \arg\min_\phi \mathrm{CE}(q(y|\mathbf{x}_t), p_\phi(y|\mathbf{x}_t)). && (21)
\end{aligned}
$$

□

Note that Eq. (6) is the expected cross entropy for different time steps t.

A.4 CONNECTION BETWEEN EQ. (9) AND EQ. (11)

In this section we make the connection between Eq. (6) and Gen-neG's objective (Eq. (11)) more clear. The objective in Eq. (9), ignoring the outer expectation with respect to t, is equal to

$$
\begin{aligned}
& - \left(\mathbb{E}_{p_\theta(\mathbf{x}_0, \mathbf{x}_t)} \left[\mathcal{O}(\mathbf{x}_0) \log C_\phi(\mathbf{x}_t; t) + (1 - \mathcal{O}(\mathbf{x}_0)) \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right) && (22) \\
= & - \int p_\theta(\mathbf{x}_0, \mathbf{x}_t) \Big(p(y = 1 | \mathbf{x}_0) \log C_\phi(\mathbf{x}_t; t) + p(y = 0 | \mathbf{x}_0) \log(1 - C_\phi(\mathbf{x}_t; t)) \Big) \, d\mathbf{x}_0 d\mathbf{x}_t && (23) \\
= & - \int p_\theta(y = 1) p_\theta(x_0 | y = 1) p_\theta(x_t | x_0) \log C_\phi(\mathbf{x}; t) \, d\mathbf{x}_0 d\mathbf{x}_t \\
& + \int p_\theta(y = 0) p_\theta(x_0 | y = 0) p_\theta(x_t | x_0) \log(1 - C_\phi(\mathbf{x}_t; t)) \, d\mathbf{x}_0 d\mathbf{x}_t && (24) \\
\approx & - \int p_\theta(y = 1) p_\theta(x_0 | y = 1) q(x_t | x_0) \log C_\phi(\mathbf{x}; t) \, d\mathbf{x}_0 d\mathbf{x}_t \\
& + \int p_\theta(y = 0) p_\theta(x_0 | y = 0) q(x_t | x_0) \log(1 - C_\phi(\mathbf{x}_t; t)) \, d\mathbf{x}_0 d\mathbf{x}_t && (25) \\
= & \, p(y = 1) \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=1)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log C_\phi(\mathbf{x}_t; t) \right] \right] \\
& + p(y = 0) \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=0)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] && (26)
\end{aligned}
$$

Noting that $\alpha := p(y = 1)$ recovers Eq. (11).

A.5 GEN-NEG'S OBJECTIVE FUNCTION

Here we show why Eq. (12) is an importance sampling estimator of the original objective function in Eq. (11).

$$
\begin{aligned}
\mathcal{L}^{\text{cls}}_\phi(\alpha) := & \, \alpha \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=1)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log C_\phi(\mathbf{x}_t; t) \right] \right] \\
& + (1 - \alpha) \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=0)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] && (27) \\
= & - \mathbb{E}_{p_\theta(y)} \left[\mathbb{E}_{p_\theta(\mathbf{x}_0 | y)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[y \log C_\phi(\mathbf{x}_t; t) + (1 - y) \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] \right]. && (28)
\end{aligned}
$$

Now we apply importance sampling to $p_\theta(y)$ by sampling from $\pi(y)$ as the proposal distribution. Therefore,

$$
\begin{aligned}
\mathcal{L}^{\text{cls}}_\phi(\alpha) = & - \mathbb{E}_{p_\theta(y)} \left[\mathbb{E}_{p_\theta(\mathbf{x}_0 | y)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[y \log C_\phi(\mathbf{x}_t; t) + (1 - y) \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] \right] && (29) \\
= & - \mathbb{E}_{\pi(y)} \left[\frac{p_\theta(y)}{\pi(y)} \mathbb{E}_{p_\theta(\mathbf{x}_0 | y)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[y \log C_\phi(\mathbf{x}_t; t) + (1 - y) \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] \right] && (30) \\
= & \, \frac{p_\theta(y = 1)}{\pi(y = 1)} \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=1)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log C_\phi(\mathbf{x}_t; t) \right] \right] \\
& + \frac{p_\theta(y = 0)}{\pi(y = 0)} \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=0)} \left[\mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] && (31) \\
= & \, \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=1)} \left[\frac{\alpha}{\pi(y = 1)} \mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log C_\phi(\mathbf{x}_t; t) \right] \right] \\
& + \mathbb{E}_{p_\theta(\mathbf{x}_0 | y=0)} \left[\frac{1 - \alpha}{\pi(y = 0)} \mathbb{E}_{q(\mathbf{x}_t | \mathbf{x}_0)} \left[- \log(1 - C_\phi(\mathbf{x}_t; t)) \right] \right] && (32)
\end{aligned}
$$

In our case, $\pi(y)$ is a uniform Bernoulli distribution i.e., $\pi(y = 1) = \pi(y = 0) = 0.5$. Therefore, minimizing Eq. (12) is equivalent to minimizing a Mone Carlo estimate of Eq. (32).

A.6 Error bound on the prior probability of positive examples α

In practice, the prior probability of generating positive samples $\alpha = p(y = 1)$ is not accessible, and we use an empirical $\hat{\alpha}$ obtained from the generated synthetic dataset to estimate it. Here is the statistical analysis for the difference of joint log-probability based on true and estimated α.

Lemma 1. *(Delta method) Suppose $\hat{\theta}_n$ follows an asymptotic normal distribution, $\sqrt{n}(\hat{\theta}_n - \theta)$ converges to $\mathcal{N}(0, \sigma^2)$ in distribution as $n \to \infty$, then if g is a continuous function with a well-defined first derivative at θ and $g'(\alpha) \neq 0$, then*

$$\sqrt{n}(g(\hat{\theta}_n) - g(\theta)) \xrightarrow{D} \mathcal{N}(0, (g'(\theta))^2\sigma^2). \tag{33}$$

Proof. By Taylor approximation expansion,

$$g(\hat{\theta}_n) \approx g(\theta) + g'(\theta)(\hat{\theta}_n - \theta) \Rightarrow \sqrt{n}(g(\hat{\theta}_n) - g(\theta)) \approx \sqrt{n}g'(\theta)(\hat{\theta}_n - \theta) \tag{34}$$

by subtracting $g(\theta)$ and multiplying $\sqrt{n}$ on the both sides. Therefore, $\sqrt{n}(g(\hat{\theta}_n) - g(\theta)) \xrightarrow{D} \mathcal{N}(0, (g'(\theta))^2\sigma^2)$. □

Given a fully synthetic dataset of size N, we estimate α with $\hat{\alpha} = \frac{1}{N}\sum_i \mathbb{1}[y_i = 1]$, and its expectation and variance are:

$$\mathbb{E}\left[\hat{\alpha}\right] = \mathbb{E}\left[\frac{1}{N}\sum_i \mathbb{1}[y_i = 1]\right] = \frac{1}{N}\sum_i \mathbb{E}\left[\mathbb{1}[y_i = 1]\right] = \frac{1}{N}\sum_i p_\theta(y_i = 1) = \alpha \tag{35}$$

$$\mathrm{Var}\left[\hat{\alpha}\right] = \mathrm{Var}\left[\frac{1}{N}\sum_i \mathbb{1}[y_i = 1]\right] = \frac{1}{N^2}\sum_i \left(\mathbb{E}\left[\mathbb{1}^2[y_i = 1]\right] - \left(\mathbb{E}\left[\mathbb{1}[y_i = 1]\right]\right)^2\right) = \frac{\alpha - \alpha^2}{N} \tag{36}$$

Let $N \to \infty$, by central limit theorem (CLT), we have $\hat{\alpha} \xrightarrow{D} \mathcal{N}\left(\alpha, \frac{\alpha-\alpha^2}{N}\right)$. If $\frac{\alpha-\alpha^2}{N} \to 0$, we have $\hat{\alpha} \to \alpha$, then by Theorem 1, the joint log-probability is:

$$\begin{aligned}\log p_\theta(\mathbf{x}_t; t, \alpha) + \log C_{\phi^*}(\mathbf{x}_t; t, \alpha) &= \log(\alpha p_\theta(\mathbf{x}_t|y = 1, t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0, t)) \\ &\quad + \log\left(\frac{\alpha p_\theta(\mathbf{x}_t|y = 1, t)}{\alpha p_\theta(\mathbf{x}_t|y = 1, t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 1, t)}\right) \quad (37) \\ &= \log\alpha + \log p_\theta(\mathbf{x}_t|y = 1, t) \quad (38)\end{aligned}$$

otherwise,

$$\begin{aligned}\log p_\theta(\mathbf{x}_t; t, \alpha) + \log C_{\phi^*}(\mathbf{x}_t; t, \hat{\alpha}) &= \log(\alpha p_\theta(\mathbf{x}_t|y = 1, t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0, t)) \\ &\quad + \log\left(\frac{\hat{\alpha} p_\theta(\mathbf{x}_t|y = 1, t)}{\hat{\alpha} p_\theta(\mathbf{x}_t|y = 1, t) + (1 - \hat{\alpha})p_\theta(\mathbf{x}_t|y = 1, t)}\right) \quad (39) \\ &= \log\left(\frac{\alpha p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0; t)}{\hat{\alpha} p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \hat{\alpha})p_\theta(\mathbf{x}_t|y = 0; t)}\right) \\ &\quad + \log\hat{\alpha} + \log p_\theta(\mathbf{x}_t|y = 1, t) \quad (40)\end{aligned}$$

The difference between Eq. (40) and Eq. (38) is:

$$\ell(\mathbf{x}_t; t, \hat{\alpha}) = \left|\log\left(\frac{\hat{\alpha}}{\alpha}\right) - \log\left(\frac{\hat{\alpha} p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \hat{\alpha})p_\theta(\mathbf{x}_t|y = 0; t)}{\alpha p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0; t)}\right)\right| \tag{41}$$

Let $g(\alpha) = \log\alpha - \log\Big(\alpha p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0; t)\Big)$, then

$$g'(\alpha) = \frac{1}{\alpha} - \frac{p_\theta(\mathbf{x}_t|y = 1; t) - p_\theta(\mathbf{x}_t|y = 0; t)}{\alpha p_\theta(\mathbf{x}_t|y = 1; t) + (1 - \alpha)p_\theta(\mathbf{x}_t|y = 0; t)}. \tag{42}$$

Based on Lemma 1, we have

$$\sqrt{N}\left(\ell(\mathbf{x}_t; t, \hat{\alpha})\right) \xrightarrow{D} \mathcal{N}(0, g'(\alpha)(\alpha - \alpha^2)) \tag{43}$$

where $g'(\alpha)$ follows Eq. (42).

Figure 5: Architecture of the network in the toy experiment. Left: the overall of the model. Right: detailed architecture of our "Residual Block". In this architecture, timestep is embedded using sinusoidal embedding and all nonlinearities are SiLU. The output of the network $F_\theta(\mathbf{x}_t, t)$ is then used in a preconditioning function to get an estimate of $\mathbf{x}_0$.

B Experimental details

B.1 Toy Experiment

Architecture We use a fully connected network with 2 residual blocks as shown in Fig. 5. The hidden layer size in our experiment is 256 and timestep embeddings (output of the sinusiodal embedding layer) is 128. Our classifier has a similar architecture, the only difference is that the classifier has a different output dimension of one. Our baseline DM and classifier networks both have around 330k parameters.

Training the baseline DM We train our models baseline models on a single GPU, (we use either of GeForce GTX 1080 Ti or GeForce GTX TITAN X) for 30,000 iterations. We use the Adam optimizer (Kingma & Ba, 2014) with a batch size of 3×10^{-4} and full-batch training i.e., our batch size is 1000 which is the same as the training dataset size.

Training the classifiers Each classifier is trained on a fully-synthetic dataset of 100k samples which consists of 50k positive and 50k negative samples. This dataset is generated with 100 diffusion steps. We train the classifier for 20k iterations with a batch size of 8192. We use Adam optimizer with a learning rate of 3×10^{-3}.

Distillation The distilled models have the same architecture and hyperparameters as the baseline DM model. They are trained for 250k iterations on the true dataset with a batch size of 1000. We use Adam optimizer with a learning rate of 3×10^{-4}.

Diffusion process We use the EDM framework in this experiment with a preconditioning similar to the one proposed in Karras et al. (2022). In particular, the following precoditioning is applied to the the network in Fig. 5, called $F_\theta(\mathbf{x}_t, t)$, to get $D_\theta(\mathbf{x}_t; t)$ which returns an estimate of $\mathbf{x}_0$.

$$D_\theta(\mathbf{x}_t; t) = \frac{\sigma_{\text{data}}^2}{\sigma(t)^2 + \sigma_{\text{data}}^2}\mathbf{x}_t + \sigma(t) F_\theta\left(\frac{1}{\sqrt{\sigma(t)^2 + \sigma_{\text{data}}^2}}\mathbf{x}_t; \frac{1}{4}\ln(\sigma(t))\right). \tag{44}$$

Following (Karras et al., 2022), we simply let $\sigma_{\text{data}} = 0.5$.

In total, we spent around 250 GPU-hours for this experiment.

B.2 INFRACTIONS IN TRAFFIC SCENE GENERATION

Overview This section provides additional details for the traffic scene generation task. The architecture for training the baseline DM model, classifiers and distillation models is majorly based on transformers introduced by Vaswani et al. (2017) . In particular, the architecture backbone consists of an encoder, a stack of attention residual blocks, and a decoder. Each of them will be discussed in detail later. The original data input shape is $[B, A, F]$ corresponding to A vehicles and F feature dimensions in a batch with B many scenes.

In terms of parameters, the attention layers comprise the majority portion of the entire architectures so that the difference in decoder is relatively small, and the resulting architectures all contain approximately 6.3 million parameters. We use NVIDIA A100 GPUs for training and validating models, synthetic datasets generation with around 400 GPU-hours in total. We train each model with a batch size of 64 and Adam optimizer with a learning rate of 10^{-4}.

Encoder and time embeddings To generate input features, we use sinusoidal positional embeddings to embed the diffusion time and 2-layer MLP with activation function SiLU to embed the original data separately into $H = 196$ hidden feature dimensions. The sum of the two embeddings is the input that is fed into the attention-based architecture.

Self-attention Layer and Cross-attention Layer The major implementation of multi-head ($k = 4$) attention blocks is built on Transformer (Vaswani et al., 2017). Applying self-attention across agents enables model to learn the multi-agent interactions, while applying map-conditional cross-attention between agents and map allows agents to interact with the road representations. To prepare road image for model input, we use a convolutional neural network and a feed-forward network (Carion et al., 2020) to generate a lower-resolution map $m' \in \mathbb{R}^{196\times 32\times 32}$ from the original image $m \in \mathbb{R}^{3\times 256\times 256}$. Since the transformer architecture is permutation-invariant, we add a 2D positional encoding (Parmar et al., 2018; Bello et al., 2019) based on m' on the top of the map representation to preserve the spatial information of the image.

Relative Positional Encodings (RPEs) During experiments, we find the collision rate is much higher than the offroad rate. In order to effectively lower the frequency of or completely avoid vehicle collision occurrence, we attempt to capture the relative positions by performing relative positional encodings (RPEs) in self-attention residual blocks and enforce the vehicles being aware of the other vehicles in close proximity in each scene. Following (Shaw et al., 2018; Wu et al., 2021; Harvey et al., 2022), we compute the distances of each pair of vehicles and summarise into a tensor of shape $[B, A, A]$, where d_{ij}^b is the distance between vehicle i and j in the b^{th} scene. We choose to use sinusoidal embeddings (similar to how we embed diffusion time t) to parameterize d_{ij}^b rather than logarithm function $f_{\text{RPE}}(d_{ij}^b) = \log(1 + d_{ij}^b)$, as we need to adequately amplify the pairwise distances between vehicles when it is comparably small. We perform this operation together with diffusion time embedding at each diffusion time step, and we regard their sum as the complete pairwise distance embeddings. The resulting embedding tensor $\mathbf{p}$ is of the shape $[B, A, A, H]$, where $\mathbf{p}_{ij}^b$ is the encoding vector of length H representing the pairwise distance of vehicle i and j in the b^{th} scene.

In each scene, we have an input sequence, $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_A)$, and each $\mathbf{x}_i$ is linearly transformed to query $\mathbf{q}_i = W^Q\mathbf{x}_i$, key $\mathbf{k}_i = W^K\mathbf{x}_i$ and value $\mathbf{v}_i = W^V\mathbf{x}_i$. We also apply linear transformation onto RPEs to obtain query $\mathbf{p}_{ij}^Q = U^Q\mathbf{p}_{ij}$, key $\mathbf{p}_{ij}^K = U^K\mathbf{p}_{ij}$ and value $\mathbf{p}_{ij}^V = U^V\mathbf{p}_{ij}$. Then the add-on output from the self-attention residual block is the aggregated outputs of the vanilla transformer and the relative-position-aware transformer:

$$\mathbf{x}_i^{\text{output}} = \mathbf{x}_i + \sum_{j=1}^{A} \alpha_{ij}(\mathbf{v}_j + \mathbf{p}_{ij}^V) \tag{45}$$

$$\text{where} \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{A}\exp(e_{ik})} \text{ and } e_{ij} = \frac{\mathbf{q}_i^\top\mathbf{k}_j + \mathbf{p}_{ij}^{Q^\top}\mathbf{k}_j + \mathbf{q}_i^\top\mathbf{p}_{ij}^K}{\sqrt{d_{\mathbf{x}}}} \tag{46}$$

Decoder The settings for baseline, distillation models and classifiers are almost identical except the decoder for producing the final output. For baseline and distillation models, we apply 2-layer MLP

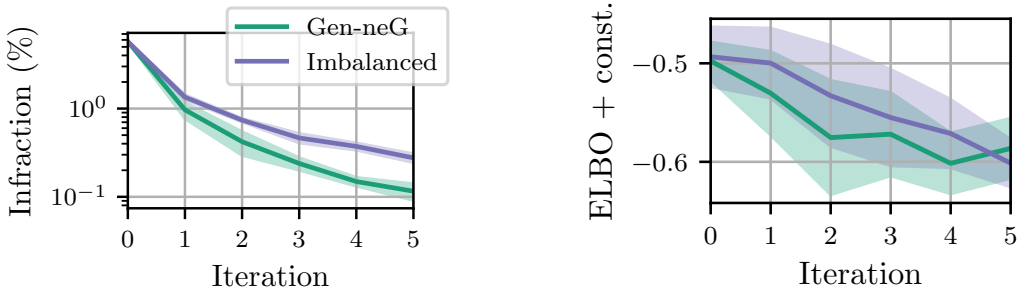


Figure 6: Infraction and ELBO estimations from different iterations of Gen-neG and an ablated version of it without label imbalance correction. Gen-neG achieves a lower infraction rate and a comparable ELBO.

and reconstruct the output of the shape $[B, A, H]$ from the final attention layer into $[B, A, D]$ through the decoder. To ensure we output individual label for each vehicle with by-agent classifiers, and a collective label for a scene with by-scene classifiers, and we conduct the operations as follows. The decoder takes the hidden representation of the shape $[B, A, H]$ and produces a tensor with feature dimension $F' = 1$ with a 2-layer MLP, which is the predicted labels from the by-agent classifiers. For by-scene classifier, we add additional MLP layer to extract the first column from the second dimension of the by-agent classifier resulting predictions and obtain a tensor of the shape $[B, 1, 1]$.

B.3 Motion Diffusion

We used the official implementation of MDM[1] for our Motion Diffusion experiment. For the baseline DM, we used their officially released best pretrained checkpoint of text to motion task on HumanML3D dataset. We generate a synthetic dataset of around 250k positive and 250k negative examples from the baseline DM which is a DDPM-based model with 1000 diffusion steps. We then define our classifier architecture using their code base. Following our other experiments, our classifier architecture is the same as the baseline DM model. We train the classifier with a batch size of 256 and a learning rate of 10^{-4} for 100k iterations. Otherwise, we use the same hyperparameters as in (Tevet et al., 2023). All the training and data generation is done on A100 GPUs.

To compute the FID scores, in accordance with (Tevet et al., 2023), we generate one motion for each caption in the HumanML3D test set, resulting in a total of 4,626 generated motions. In contrast to (Tevet et al., 2023), we refrain from applying classifier-free guidance during the sample generation process. Subsequently, we calculate the FID score between the generated motions and the ground-truth motions in the test set. The FID metric quantifies the distance in the latent space of a motion encoder, which has been pre-trained using contrastive learning (Guo et al., 2022).

The total compute used for this experiment (generating the datasets and training the classifiers) was around 600 GPU-hours.

C Additional Results

C.1 Alternative approaches to conditional generation with diffusion models

In our journey towards solving this problem, we explored multiple different approaches. Here we briefly mention a few of the more promising ones.

Data redaction Kong & Chaudhuri (2023) proposes a GAN-based framework for data redaction relying on the pre-trained generative models, and this is performed by restricting the learned data distribution within the complement of a specified redaction set. One of the proposed algorithm is to keep updating the fake dataset with invalid samples from GAN and keep the true dataset fixed. We

[1] https://github.com/GuyTevet/motion-diffusion-model

implement this data redaction baseline for our toy experiment and performed network architecture and hyperparameter tuning to achieve the best baseline results. In line with the results in our paper we report infraction rate and a metric of faithfulness to the true data distribution. Before we report ELBO or r-ELBO, but since GANs are implicit generative models i.e., do not provide densities, we compute the Maximum Mean Discrepancy (MMD) between the generated samples and a held-out test set instead. To compare it against our method, we computed MMD for our results in the paper as well.

The best infraction rate we got from this GAN-based baseline was 0.595%. This is better than the baseline diffusion model in iteration 0 of Gen-neG (5.75%) but Gen-neG outperforms it after two iterations of guidance, and improves even further in later iterations.In terms of MMD (lower is better), all iterations of Gen-neG are between 7×10^{-4} and 8.1×10^{-4} while the best MMD from the GAN baseline is larger than 9.4×10^{-4} (more than 16% worse than the weakest MMD in Gen-neG). Moreover, visual inspection of quality of samples shows a clear sign of overfitting in the GAN baseline. For example, samples very close to the boundaries are underrepresented. This is expected since data redaction only changes the distribution of "fake" samples fed to the GAN discriminator and the set of "real" samples is always fixed to the training set. Consequently, training the model longer for data redaction results in a side effect of overfitting to the training set. This is less of a problem when the training set is large enough which is not the case in our setup. It is in contrast to Gen-neG where overfitting in the refinement steps does not pose a threat because the training data for refinement iterations is fully synthetic and abundant.

Diffusion bridges Liu et al. (2023) proposes a framework for diffusion modeling of constrained domains. Their proposed method requires adding a guidance term to the diffusion process which comes in form of an expectation. This expectation is only tractable for simple constraints such as the one in our toy experiment. We applied this method on our toy experiment which lead to practically zero infraction, without overfitting. As the update term is intractable for the larger scale experiment, we tried to learn an estimation of it, but we failed to reliably estimate the update term.

Learning in the joint space of $(\mathbf{x}, y)$ Inspired by Weilbach et al. (2022), we trained a model to estimate all the marginals and conditionals on the joint space of $(\mathbf{x}, y)$ where $q(\mathbf{x}|y = 1)$ is the training set, $q(\mathbf{x}|y = 0)$ is a synthetically generated set of negative examples and $q(y = 1)$ is a hyper-parameter. We assign a probability vector for choosing a training task among learning conditional probabilities $q(\mathbf{x}|y), q(y|x)$ and joint probability $q(\mathbf{x}, y)$ collectively with one architecture parameterized by θ with implicit classifier learner $p_\theta(y|\mathbf{x})$. Intuitively, feeding the model with both positive and negative samples helps the model learn the boundary between the positive region Ω and negative region $\Omega^{\complement}$. At test time, we only sample from $p_\theta(\mathbf{x}|y = 1)$. Our preliminary results suggested that it helps reducing the infraction rate, but is outperformed by Gen-neG.

C.2 Ablation on label imbalance

As mentioned in Section 3.2, naively generating synthetic datasets for training the classifier causes a major label imbalance issue hinders training of the classifier. In this section we perform an ablation study to empirically demonstrate its effect in our experiments.

As a reminder, Gen-neG guarantees an equal number of positive and negative examples in the synthetic datasets, and it employs importance sampling to address any distribution shift introduced. In the ablated experiment (referred to as "imbalanced" in the results), the ratio of positive to negative examples is model-dependent, being equal to the model's infraction rate. It results in a gradual increase in the dominance of positive examples as the model's infraction rate decreases.

Toy experiment We repeat our toy experiment with and without Gen-neG's imbalance correction. In each iteration, we create a synthetic dataset of 20,000 samples and train a binary classifier for 10,000 iterations. The other details are the same as the experiment in the main text. We show the performance of these models in Fig. 6. We observe that "imbalaced", the ablated version of Gen-neG, is consistently outperformed by Gen-neG in infraction rate. Furthermore, the performance gap between the two methods widens as the models improve. However, it is worth noting that Gen-neG achieves a comparable ELBO to that of "imbalanced" despite these infraction rate differences.

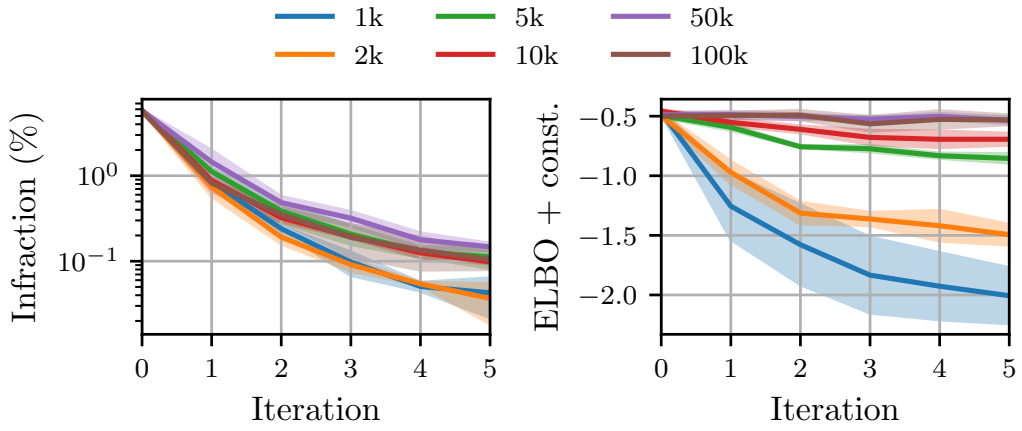


Figure 7: Ablation study for toy experiment on synthetic dataset size

Table 3: Ablation study for the traffic scene generation experiment on synthetic dataset size

Classifier dataset size	Collision (%) ↓	Offroad (%) ↓	Infraction (%) ↓	r-ELBO ($\times 10^2$) ↑
baseline DM	28.3 ± 0.7	1.3 ± 0.1	29.3 ± 0.6	-27.5 ± 0.01
8000	23.8 ± 0.4	0.9 ± 0.2	24.5 ± 0.5	-28.0 ± 0.01
40000	19.7 ± 0.8	0.8 ± 0.2	20.3 ± 0.9	-27.8 ± 0.01
80000	17.6 ± 0.7	0.7 ± 0.2	18.2 ± 0.6	-27.7 ± 0.01
400000	16.6 ± 0.7	0.8 ± 0.2	17.5 ± 0.7	-27.7 ± 0.01
800000	16.4 ± 0.5	0.9 ± 0.1	17.2 ± 0.4	-27.7 ± 0.01

Table 4: Results for traffic scene generation, in terms of collision, offroad, and overall infractions as well as ELBO. Two varieties ("by-scene" and "by-agent") for the classifier are presented, as well as results with (Gen-neG) and without importance sampling. The final two rows provide the results of distilling the models labelled with † and *.

Method	Collision (%) ↓	Offroad (%) ↓	Infraction (%) ↓	r-ELBO ($\times 10^2$) ↑
baseline DM	28.3 ± 0.70	1.3 ± 0.14	29.3 ± 0.64	-27.5 ± 0.01
by-scene	23.3 ± 0.7	1.0 ± 0.28	24.1 ± 0.67	-27.6 ± 0.01
by-scene imbalanced	23.8 ± 0.6	1.0 ± 0.3	24.6 ± 0.54	-27.6 ± 0.01
by-agent	16.4 ± 0.5	0.9 ± 0.12	17.2 ± 0.44	-27.7 ± 0.01
by-agent imbalanced	17.8 ± 1.21	0.9 ± 0.16	18.6 ± 1.3	-27.7 ± 0.01

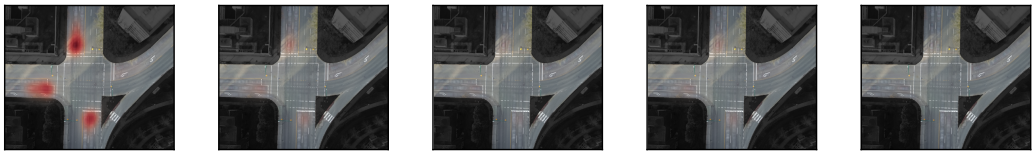
Traffic Scene Generation We conduct a similar ablation as described above, where we train classifiers on a imbalanced datasets of the same size as our method. The results of this ablated experiment are presented in Table 4, and they are compared with the results reported in the main text.

C.3 Ablation on synthetic dataset size

We conducted an ablation on the number of samples required on the toy and the initial conditions tasks. Fig. 7 and Table 3 shows our results. We observe that the infraction rate constantly decreases irrespective of the dataset size. However, in order to avoid distribution shift we need a large enough dataset, as is evident from the ELBO plot. It is important to emphasize that the classifiers are trained on fully synthetic data generated by the model itself. Therefore, in principle we have access to an unbounded number of samples. As our results show, for the best performance, it is important to ensure the sample size is sufficiently large.

Table 5: Computation cost of different experiments

	Time per diffusion step (s)		
Method	Toy	Traffic Scenes	Motion Diffusion
baseline DM / distilled model	$0.002 \pm 1 \times 10^{-4}$	$0.032 \pm 9 \times 10^{-6}$	$0.083 \pm 2 \times 10^{-4}$
baseline DM + 1 classifier	$0.005 \pm 2 \times 10^{-4}$	$0.086 \pm 2 \times 10^{-5}$	$0.171 \pm 8 \times 10^{-4}$
baseline DM + 2 classifiers	$0.008 \pm 2 \times 10^{-4}$	$0.138 \pm 3 \times 10^{-5}$	-



(a) (b) (c) (d) (e)

Figure 8: Complete visualization comparisons for infraction in traffic scene generation experiments. Subplots show infraction per unit area under different models. (a): before Gen-neG is applied; (b): the second is the a classifier added onto the baseline model; (c): third is a stack of classifiers added onto the baseline model. (d) and (e) are the distillation models trained to learn the baseline model with one classifier or a stack of classifiers. A clear reduction in terms of infractions per unit area can be observed from left to right.

C.4 Computational Cost

We compute the wall-clock time of different iterations of Gen-neG for different experiments and present the results in Table 5. It is important to note that our distilled model has the same sampling computational complexity as the baseline diffusion modell, since we use the same architecture for the distilled model.

For each experiment, we run our model at different interactions on the same machine and report the time it takes for one forward pass through the model. Table 5 shows the average wall-clock time for running one denoising step in each of our experiments with a different number of classifiers from Gen-neG. We observe that additional classifiers linearly increase the running time (for conciseness, we have not included further iterations of Gen-neG on toy experiments in the table. Its runtime simply continues growing linearly). Moreover, in the Traffic Scenes and toy experiments, the overhead of each classifier is larger than the runtime of the baseline model alone (almost 1.5 times). This is because the architecture of our classifier is the same as the baseline model and for each forward pass of a classifier-guided model, one forward and one backward pass through the classifier is required. However, on the Motion diffusion experiment, the overhead of the classifier is relatively smaller. This is because the model is text-conditional, but the classifier is not. Therefore, the classifier is cheaper to run.

C.5 More visualization results for traffic scene generation

In Fig. 8 we report more visualization results from our traffic scene generation experiment. This figure follows from and adds more details to Fig. 1.

C.6 Overfitting in the toy experiment

Here we report our results for overfitting the baseline DM in the toy experiment. We run an experiment with 250,000 training iterations, much larger than the 30,000 iterations in the reported results. As we can see in Fig. 9, the infraction rate keeps decreasing. However, the model starts overfitting after around 30,000 iterations, as measure by the ELBO on a held-put set. This suggests that the architecture is expressive enough to model sharp jumps in the learned density. However, simply training it on a small dataset without incorporating any prior on "where to allocate its capacity" fails

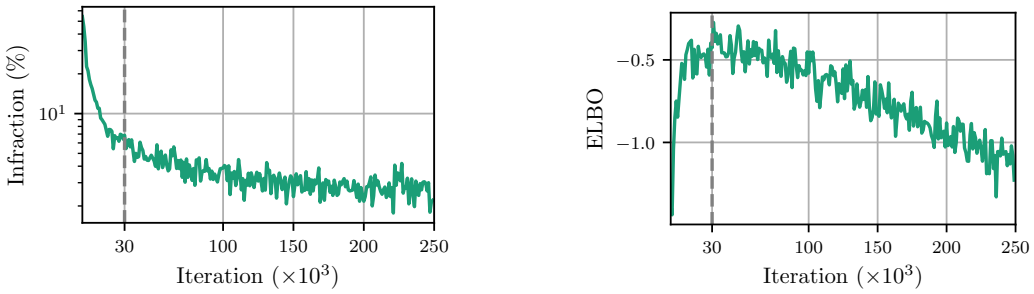


Figure 9: Overfitting results in the toy experiment. The plot on the left shows the infraction rate and the one on the right shows the ELBO on a held-out validation set. We observe that training baseline DM for longer can achieve much lower infraction rates that reported. However, it quickly starts to overfit, leading to poor ELBO estimates on the held-out validation set.

because the model does not receive any signal on where the actual "sharp jump" is. Gen-neG, on the other hand, provides this kind of signal through the oracle-assisted guidance.