Deep Generative Models – Part II

Martin Renqiang Min
NEC Laboratories America
Mar 31, 2025

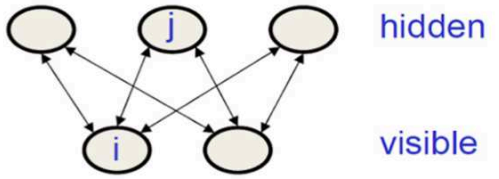
Outline

- Recap: RBM, VAE, GAN, Wasserstein Distance, WGAN
- Wasserstein Autoencoder and Applications
- Diffusion Probabilistic Models and Applications

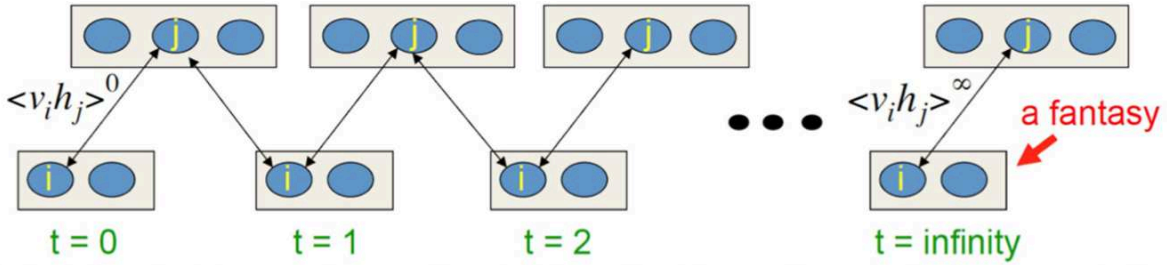
Recap: RBM

$$p(\mathbf{v},\mathbf{h}) \propto e^{-E(\mathbf{v},\mathbf{h})} \qquad E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_i \sum_j v_i w_{i,j} h_j$$

- We restrict the connectivity to make inference and learning easier.
  - Only one layer of hidden units.
  - No connections between hidden units.
- In an RBM it only takes one step to reach thermal equilibrium when the visible units are clamped.
  - So we can quickly get the exact value of : $< v_i h_j >_{\mathbf{v}}$



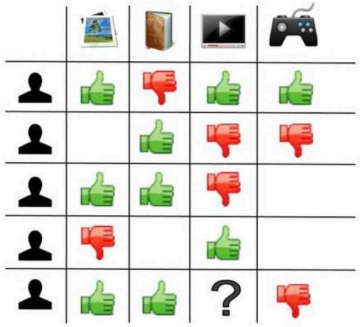
$$p(h_j = 1) = \frac{1}{1 + e^{-\left(b_j + \sum_{i \in vis} v_i w_{ij}\right)}}$$



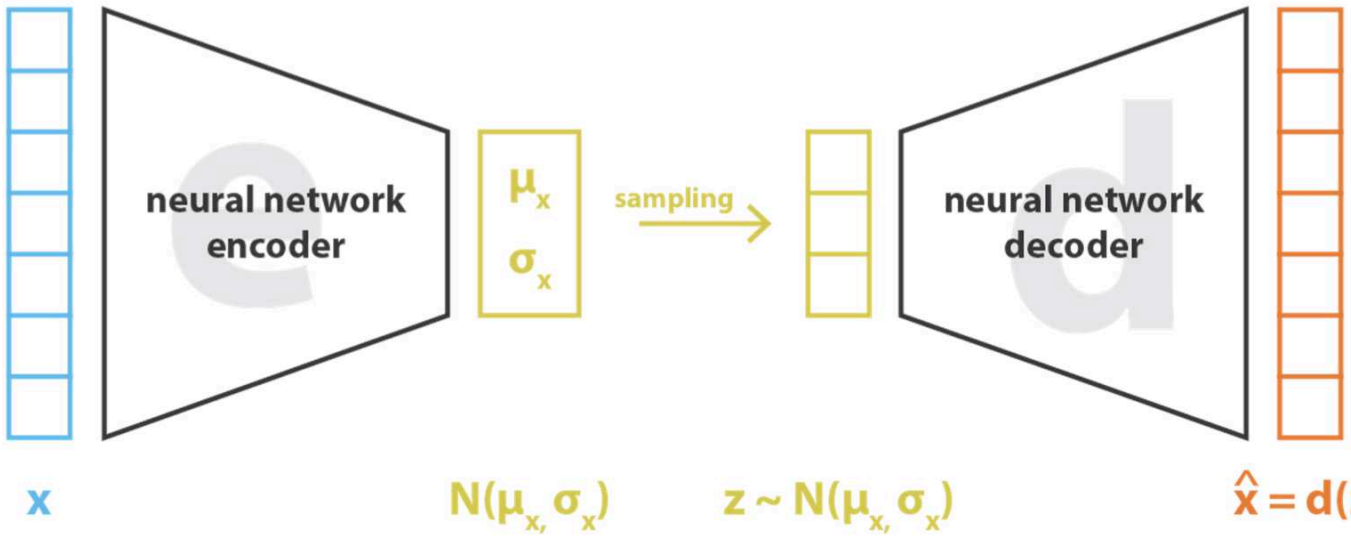
Start with a training vector on the visible units. Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^\infty \right)$$

Parameter-sharing RBM for Collaborative Filtering: Treat each user rating vector as a data point



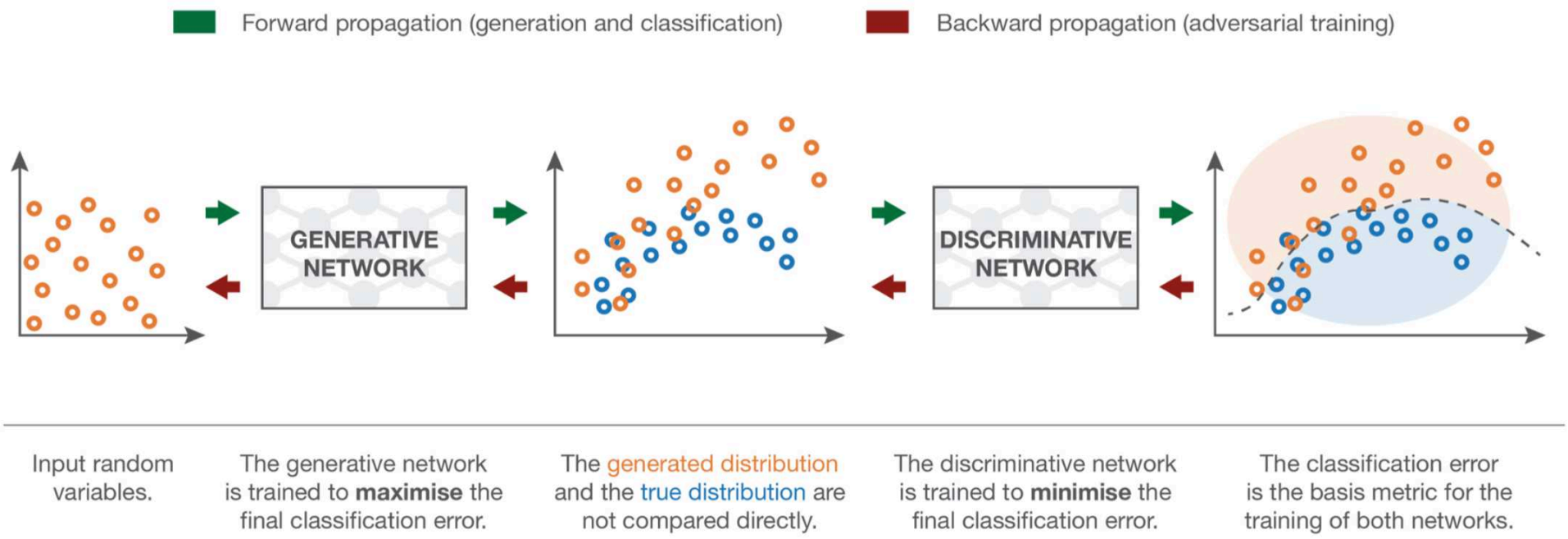
Recap: Variational Autoencoder



$$\text{loss} = \|x - \hat{x}\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Recap: Generative Adversarial Network (GAN)

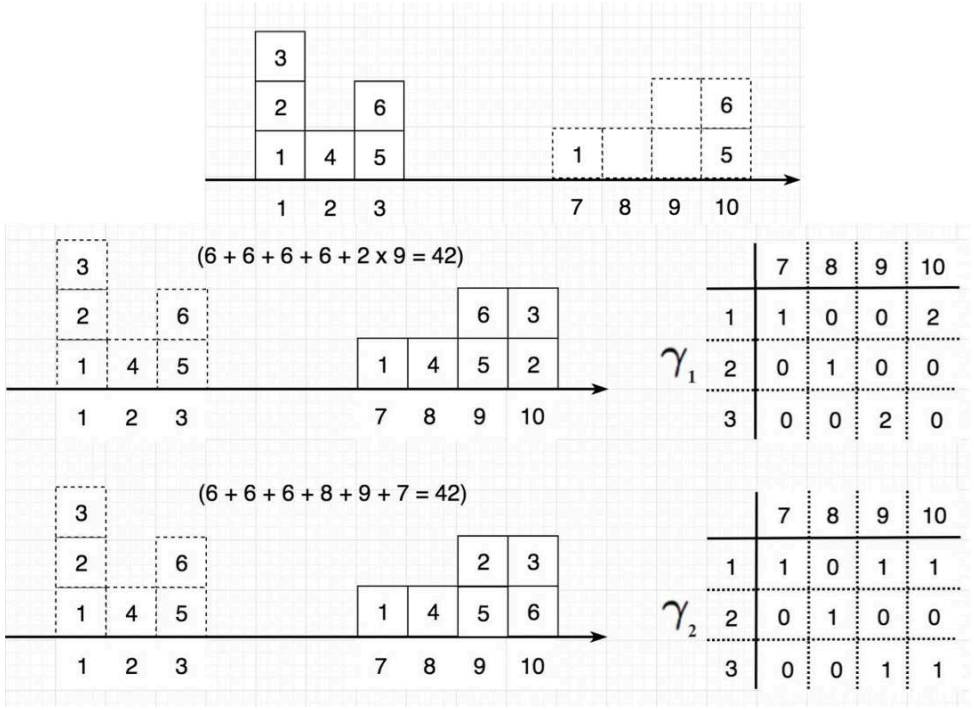


Picture Credit:https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Goodfellow *et al.*, Generative Adversarial Nets. NIPS 2014.

Wasserstein Distance

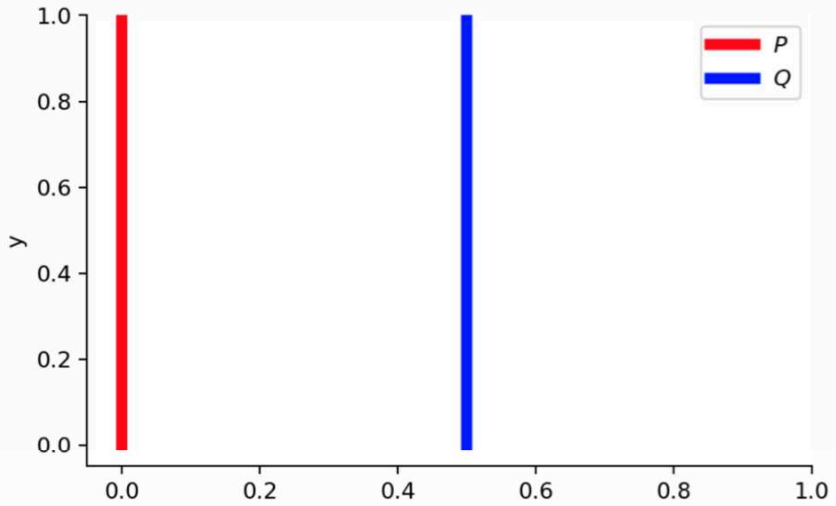
The Wasserstein distance of $\boldsymbol{p}$ and $\boldsymbol{q}$ is the minimum cost of transporting mass in converting the shape of a data distribution $\boldsymbol{q}$ to the shape of a data distribution $\boldsymbol{p}$. It is also called Optimal Transport Cost or Earth Mover Distance.



Picture Credit: https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

Comparing Wasserstein Distance with KLD and JSD

$$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0, 1)$$
$$\forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)$$



When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P, Q) = \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

$$W(P, Q) = |\theta|$$

But when $\theta = 0$, two distributions are fully overlapped:

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P, Q) = 0$$
$$W(P, Q) = 0 = |\theta|$$

Picture Credit:https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

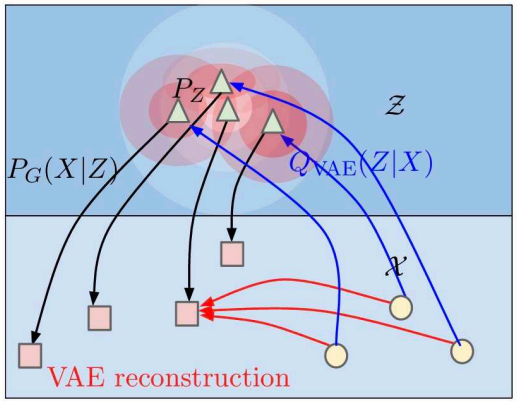
WGAN vs. GAN

	Discriminator/Critic	Generator
GAN	$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$	$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$
WGAN	$\nabla_w \frac{1}{m} \sum_{i=1}^{m} \left[f(x^{(i)}) - f(G(z^{(i)})) \right]$	$\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} f(G(z^{(i)}))$

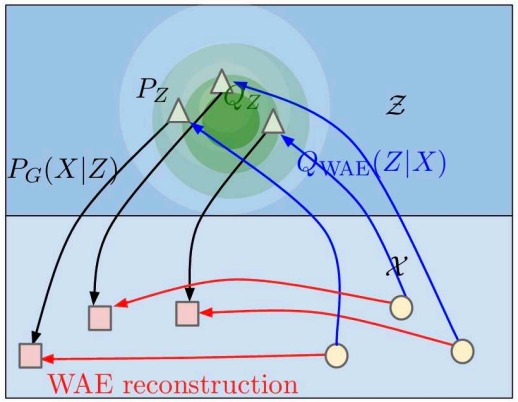
In WGAN, we have a critic with a scalar output without log

Wasserstein Autoencoder

(a) VAE



(b) WAE

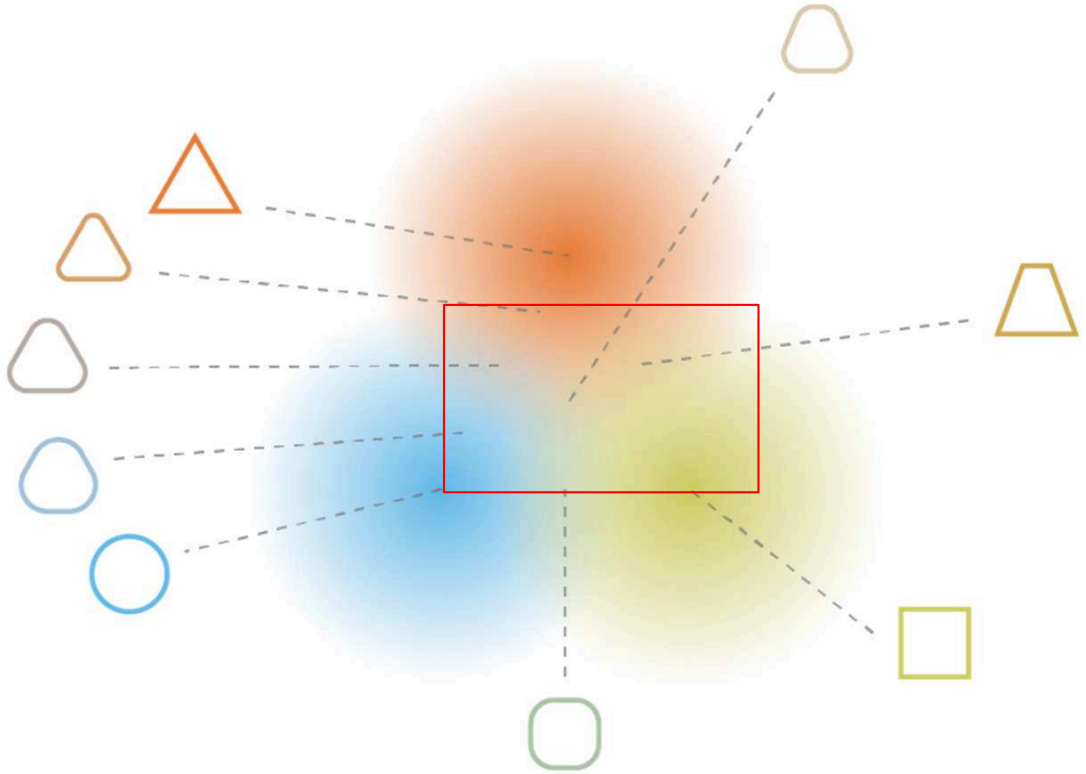


Both VAE and WAE minimize two terms: the reconstruction cost and the regularizer penalizing discrepancy between P_Z and distribution induced by the encoder Q. VAE forces $Q(Z|X = x)$ to match P_Z for all the different input examples x drawn from P_X. This is illustrated on picture (a), where every single red ball is forced to match P_Z depicted as the white shape. Red balls start intersecting, which leads to problems with reconstruction. In contrast, WAE forces the continuous mixture $Q_Z := \int Q(Z|X)dP_X$ to match P_Z, as depicted with the green ball in picture (b). As a result latent codes of different examples get a chance to stay far away from each other, promoting a better reconstruction.

$$D_{\text{WAE}}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\big[c\big(X, G(Z)\big)\big] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z)$$

Tolstikhin *et al.*, ICLR 2018: https://arxiv.org/pdf/1711.01558

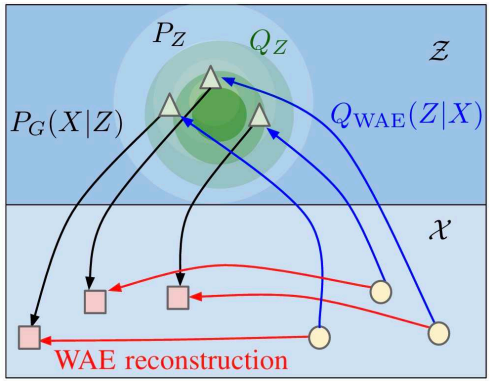
WAE Solves the Problems of VAE with Overlapping Latent Space



Picture Credit: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

Properties of Wasserstein Autoencoder

$$D_{\mathrm{WAE}}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\big[c\big(X, G(Z)\big)\big] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z)$$



If P_z is usually chosen as a high-dimensional Gaussian with unit variance $\mathrm{N}(\mathbf{0}, \mathbf{I})$, we essentially encourage each dimension of z from Q_z to be independent to encode different semantic attribute of input data x.

The loss function of (recurrent) WAE maximizes the mutual information between input data and different disentangled latent factors

Han, Min, Han, Li, and Zhang, ICLR 2021: https://arxiv.org/pdf/2101.07496

Training of Wasserstein Autoencoder

$$D_{\mathrm{WAE}}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X}\mathbb{E}_{Q(Z|X)}\big[c\big(X, G(Z)\big)\big] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z)$$

Algorithm 1 Wasserstein Auto-Encoder with GAN-based penalty (WAE-GAN).

Require: Regularization coefficient $\lambda > 0$.

Initialize the parameters of the encoder Q_ϕ, decoder G_θ, and latent discriminator D_γ.

while (ϕ, θ) not converged **do**

Sample $\{x_1, \dots, x_n\}$ from the training set

Sample $\{z_1, \dots, z_n\}$ from the prior P_Z

Sample $\tilde{z}_i$ from $Q_\phi(Z|x_i)$ for $i = 1, \dots, n$

Update D_γ by ascending:

$$\frac{\lambda}{n}\sum_{i=1}^{n} \log D_\gamma(z_i) + \log\big(1 - D_\gamma(\tilde{z}_i)\big)$$

Update Q_ϕ and G_θ by descending:

$$\frac{1}{n}\sum_{i=1}^{n} c\big(x_i, G_\theta(\tilde{z}_i)\big) - \lambda \cdot \log D_\gamma(\tilde{z}_i)$$

end while

Algorithm 2 Wasserstein Auto-Encoder with MMD-based penalty (WAE-MMD).

Require: Regularization coefficient $\lambda > 0$, characteristic positive-definite kernel k.

Initialize the parameters of the encoder Q_ϕ, decoder G_θ, and latent discriminator D_γ.

while (ϕ, θ) not converged **do**

Sample $\{x_1, \dots, x_n\}$ from the training set

Sample $\{z_1, \dots, z_n\}$ from the prior P_Z

Sample $\tilde{z}_i$ from $Q_\phi(Z|x_i)$ for $i = 1, \dots, n$

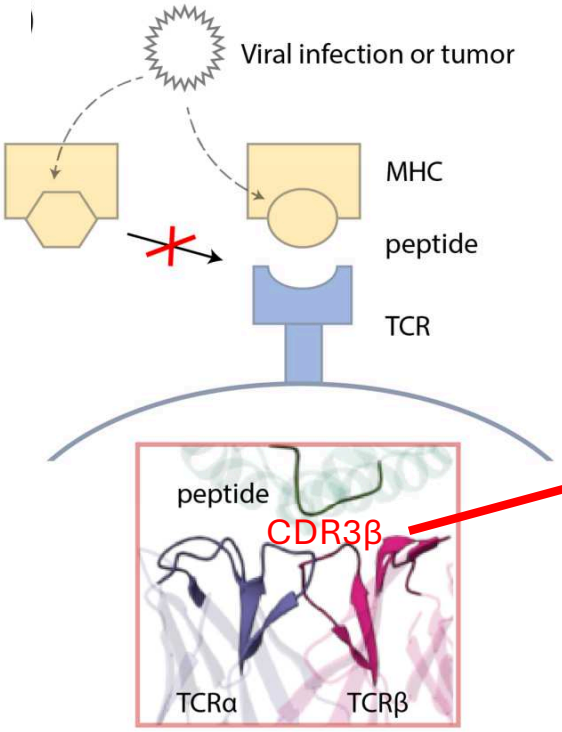
Update Q_ϕ and G_θ by descending:

$$\frac{1}{n}\sum_{i=1}^{n} c\big(x_i, G_\theta(\tilde{z}_i)\big) + \frac{\lambda}{n(n-1)}\sum_{\ell\neq j} k(z_\ell, z_j) + \frac{\lambda}{n(n-1)}\sum_{\ell\neq j} k(\tilde{z}_\ell, \tilde{z}_j) - \frac{2\lambda}{n^2}\sum_{\ell,j} k(z_\ell, \tilde{z}_j)$$

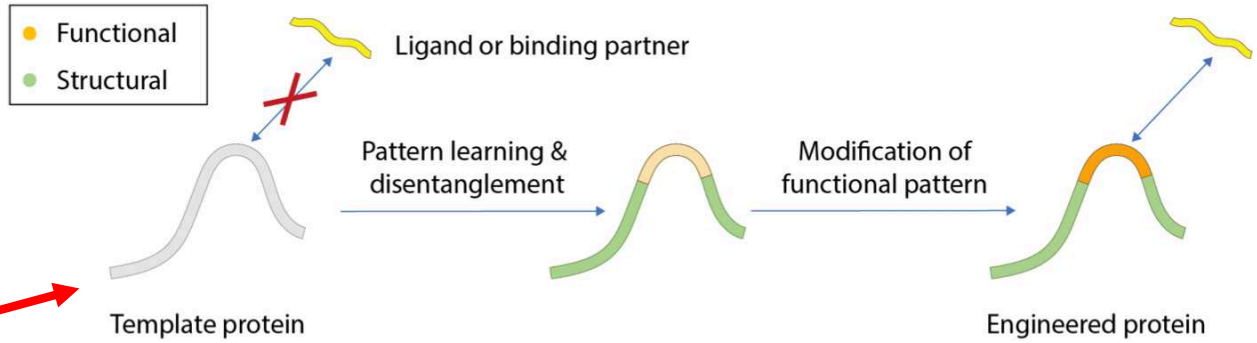
end while

Tolstikhin *et al.*, ICLR 2018: https://arxiv.org/pdf/1711.01558

Disentangled Wasserstein Autoencoder for T-Cell Receptor Engineering (NeurIPS 2023)



- Functional pattern: defines the function (e.g. binding specificity)
- Structural pattern: defines the structural backbone, required to maintain the validity of the protein

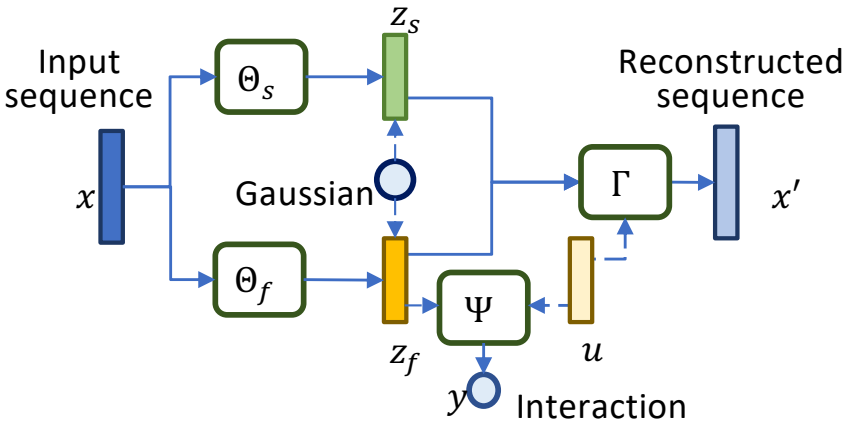


Dataset

- Positive: known TCR-peptide interaction pairs from databases (VDJdb and McPAS-TCR)
- Negative: randomly shuffled TCR-peptide pairs

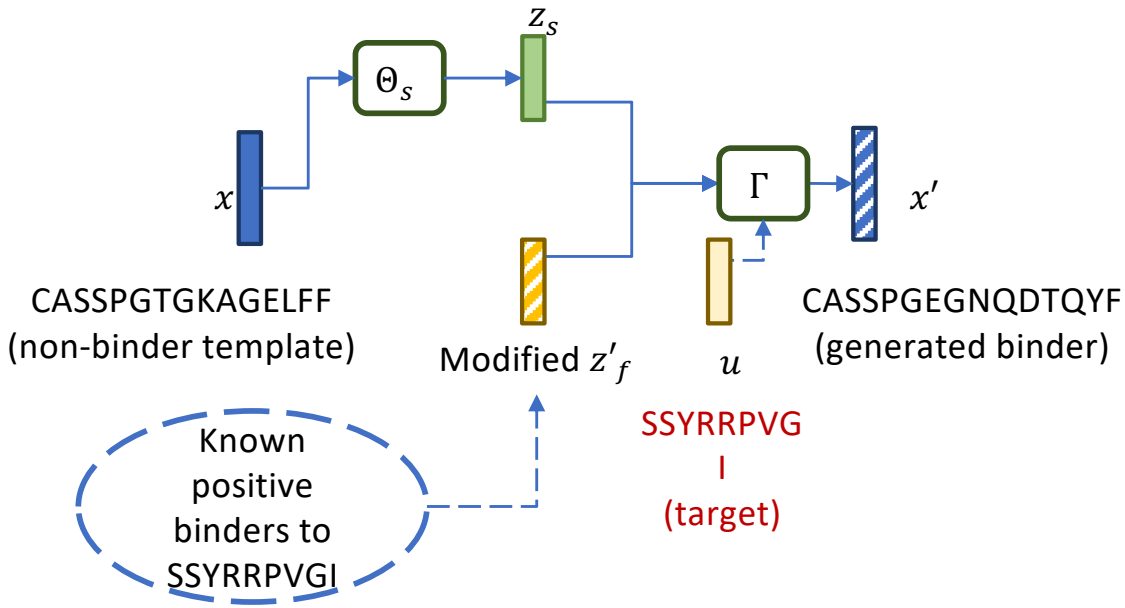
Objective: modify the binding specificity given a TCR CDR3β and a target peptide, by only altering its functional pattern.

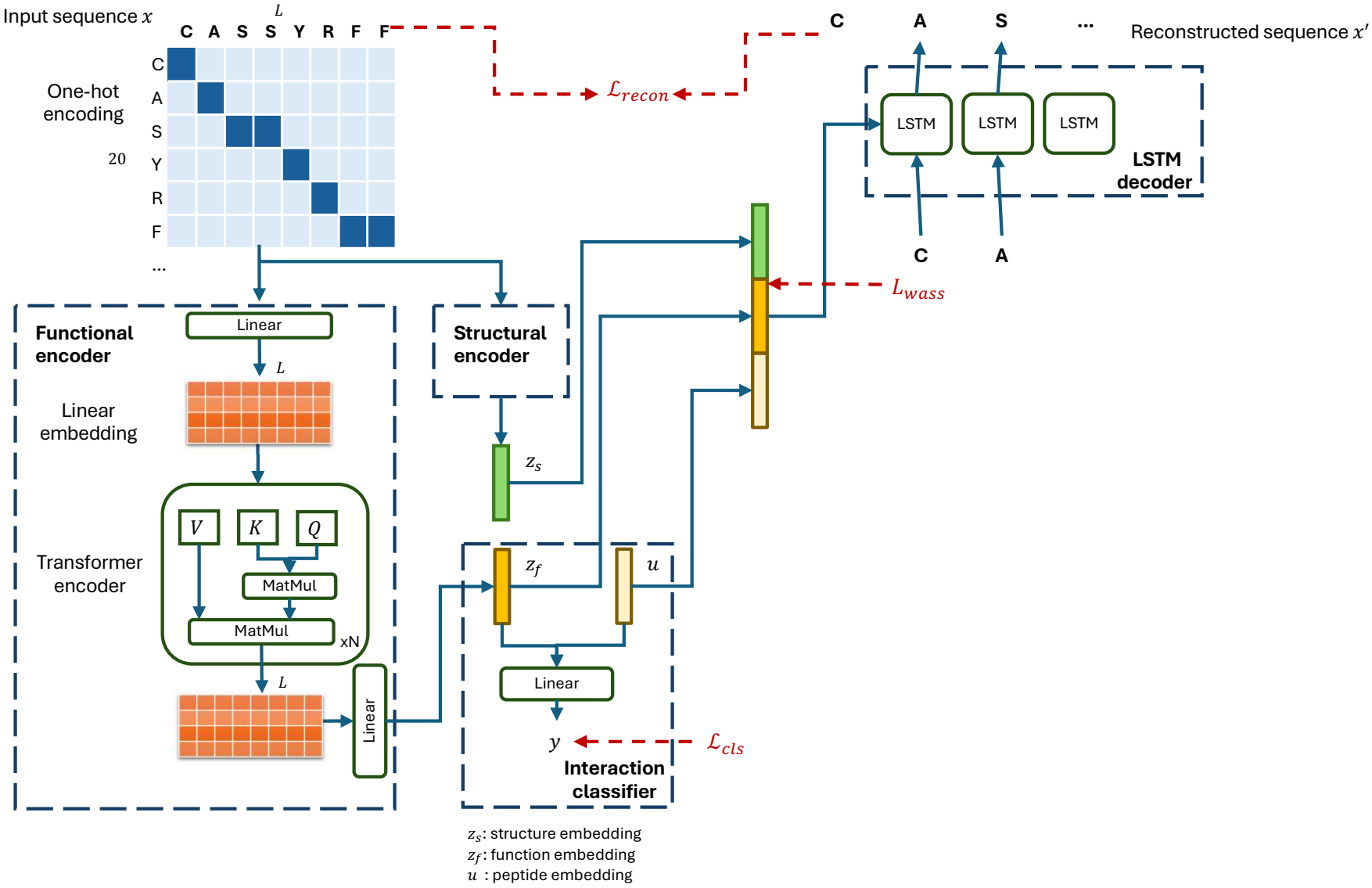
Training



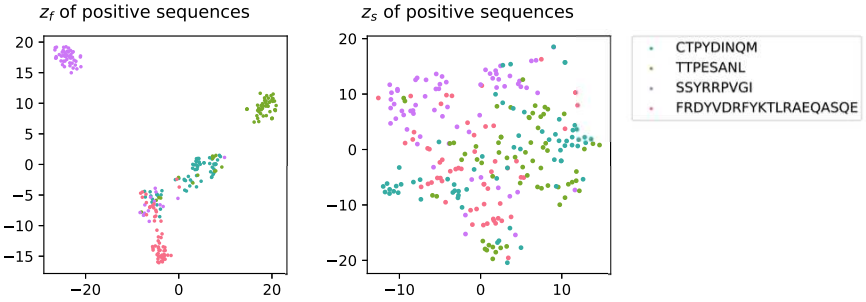
- Reconstruction loss $\mathcal{L}_{recon}\ (x, x')$
- Classification loss $\mathcal{L}_{cls}\ (\Psi(u, z_f), y)$
- Wasserstein loss $L_{wass}\ ((z_f, z_s), N(0, I))$

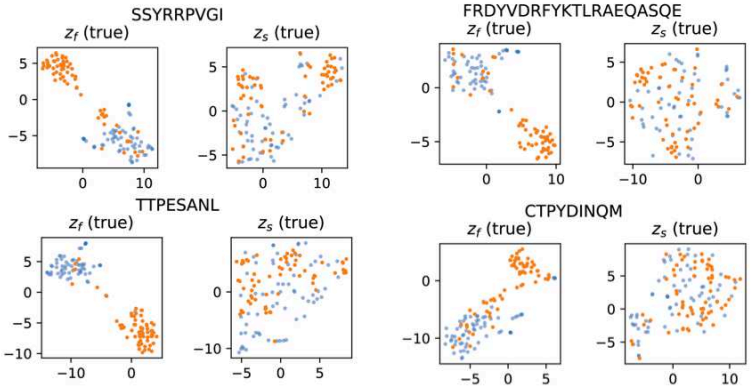
Optimization




Input sequence x
L
C A S S Y R F F
One-hot encoding
20
C
A
S
Y
R
F
...
$\mathcal{L}_{recon}$
C A S ...
Reconstructed sequence x'
LSTM
LSTM
LSTM
LSTM decoder
C A
L_{wass}
Functional encoder
Linear
L
Linear embedding
Transformer encoder
V K Q
MatMul
MatMul
xN
L
Linear
Structural encoder
z_s
z_f
u
Linear
y
$\mathcal{L}_{cls}$
Interaction classifier
z_s: structure embedding
z_f: function embedding
u : peptide embedding

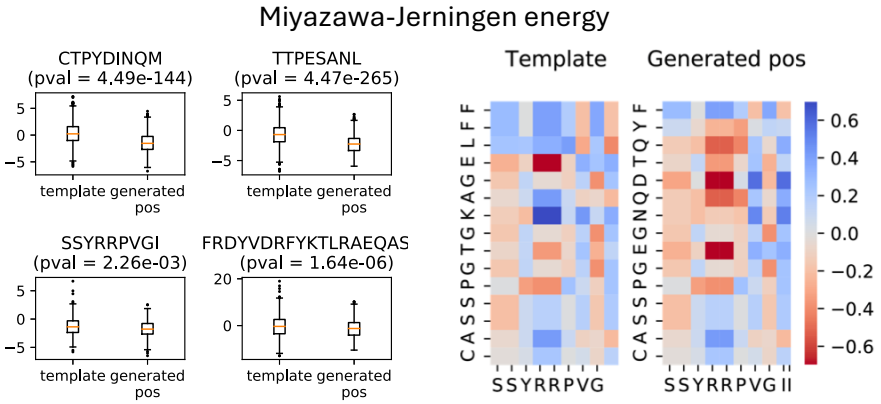
Disentanglement analysis



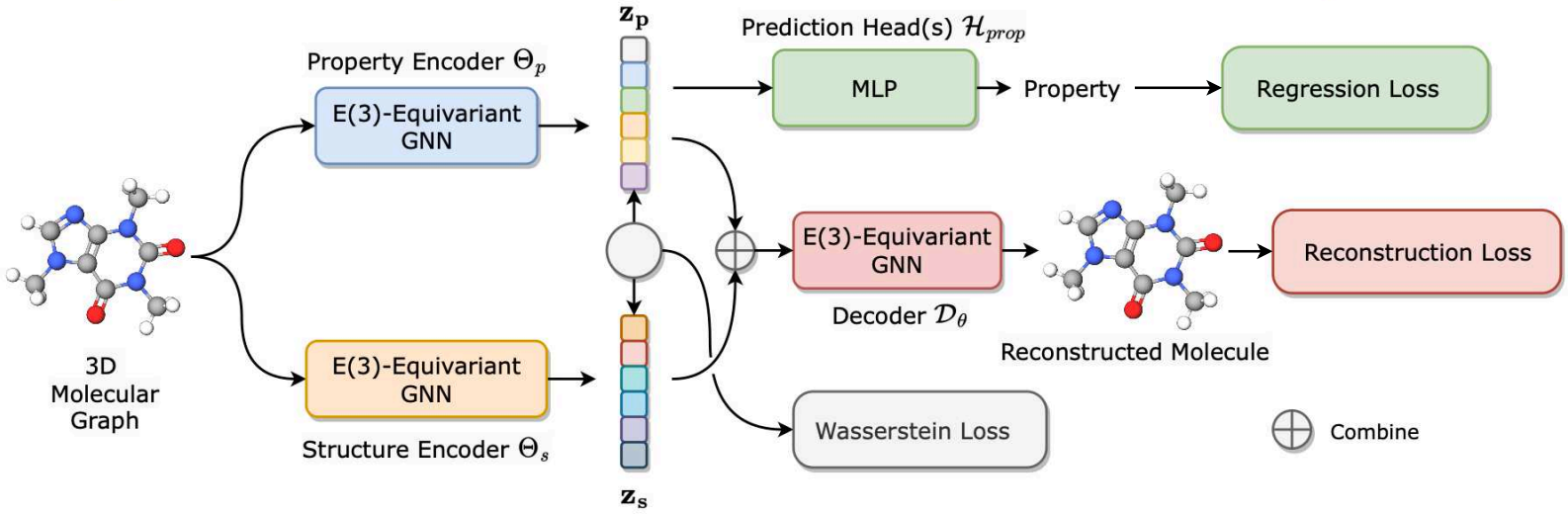


TCR engineering results

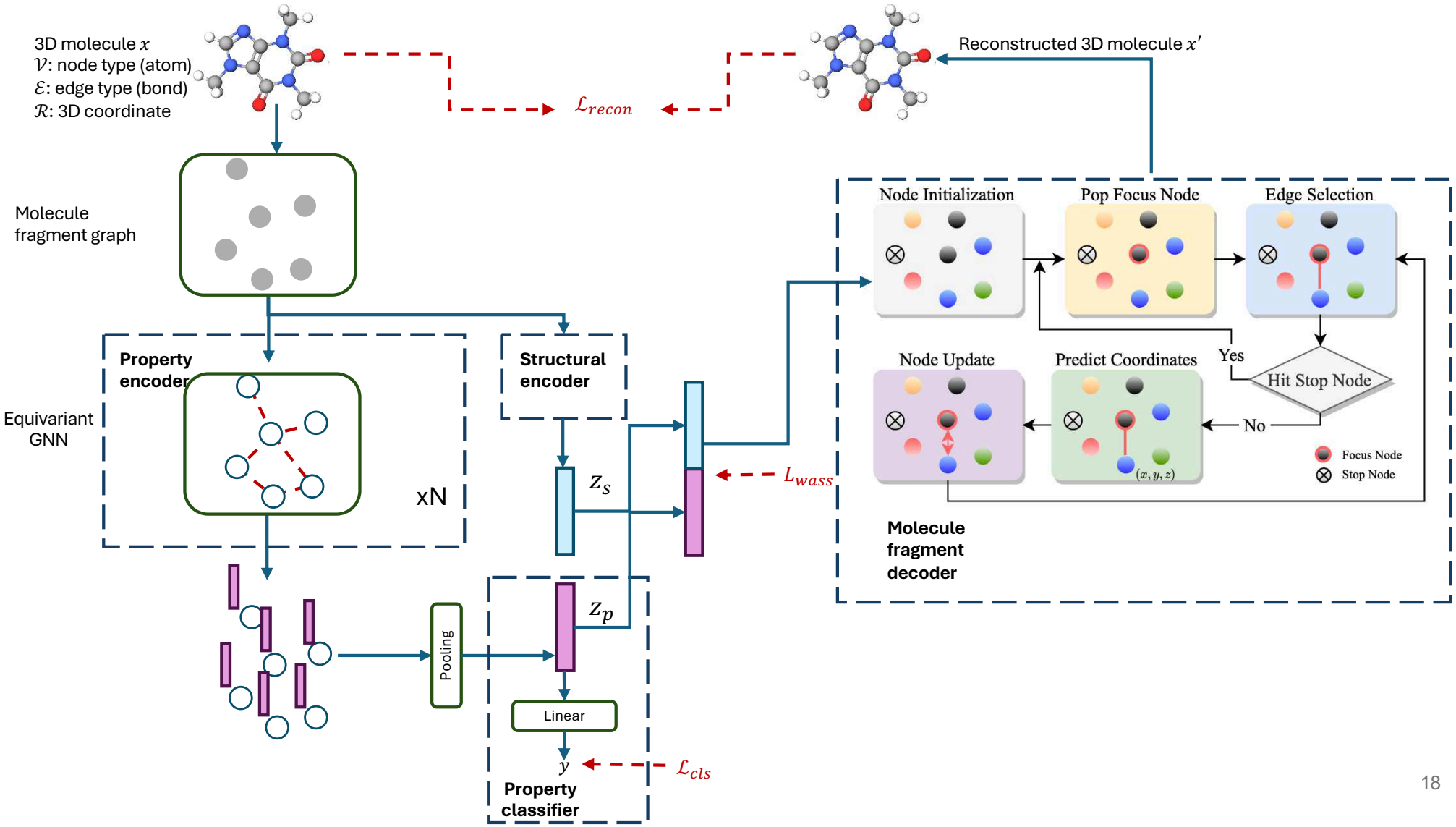
	%valid	#mut/len	%positive valid ↑
TCR-dWAE	0.61±0.06	0.49±0.05	**0.23±0.02**
TCR-dVAE	0.64±0.05	0.4±0.02	0.16±0.01
greedy	0.02±0.0	0.34±0.0	0.02±0.0
genetic	0.02±0.0	NA	0.02±0.0
naive rm	0.03±0.0	0.35±0.01	0.0±0.0
MCTS	0.0±0.0	NA	0.0±0.0
TCR-dWAE (null)	0.85±0.01	0.45±0.07	0.04±0.03
original	0.92	NA	0.01



Learning Disentangled Equivariant Representation for Explicitly Controllable 3D Molecule Generation (AAAI 2025)

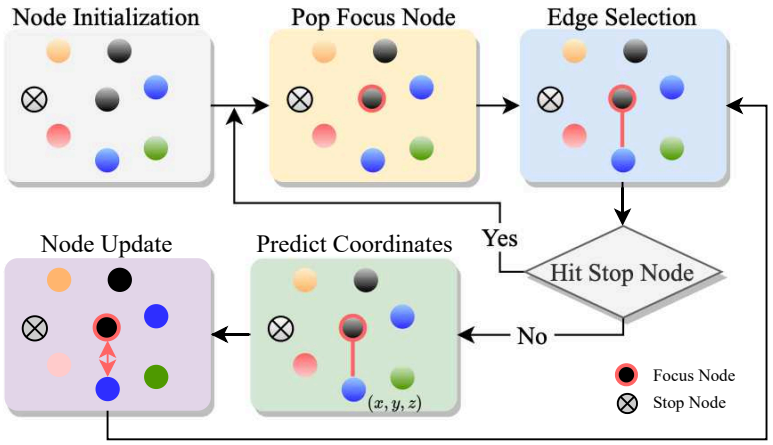


- Reconstruction loss $\mathcal{L}_{recon}\ (x, x') = \mathcal{L}_{nodeType} + \mathcal{L}_{edge} + \mathcal{L}_{coords}$
- Classification loss $\mathcal{L}_{cls}\ (\Psi(z_p), y)$
- Wasserstein loss $L_{wass}\ ((z_p, z_s), N(0, I))$


3D molecule x
$\mathcal{V}$: node type (atom)
$\mathcal{E}$: edge type (bond)
$\mathcal{R}$: 3D coordinate
$\mathcal{L}_{recon}$
Reconstructed 3D molecule x'
Molecule fragment graph
Property encoder
Equivariant GNN
xN
Structural encoder
z_s
z_p
Pooling
Linear
y
$\mathcal{L}_{cls}$
Property classifier
L_{wass}
Node Initialization
Pop Focus Node
Edge Selection
Yes
Hit Stop Node
No
Node Update
Predict Coordinates
(x, y, z)
Focus Node
Stop Node
Molecule fragment decoder

Algorithm 1: 3D Molecule Graph Reconstruction in E3WAE

Input: Latent variables $\mathbf{z}_h$, $\mathbf{z}_v$
Initialize:
  $\{x_i\}_{i=1,\cdots,n} \leftarrow \text{NodeTypes}(\mathbf{z}_h, \mathbf{z}_v)$
  Queue $Q \leftarrow \emptyset$
  Q.push(RandomSelect($\{1, \cdots, n\}$)
  3D Graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}, \mathcal{R}\}$, where $\mathcal{V} \leftarrow \emptyset, \mathcal{E} \leftarrow \emptyset, \mathcal{R} \leftarrow \emptyset$
while $Q \neq \emptyset$ **do**
  $f \leftarrow Q$.pop()
  Add node f to graph $\mathcal{G}$
  isStopNode $\leftarrow$ False
  while not isStopNode **do**
    $(i, \text{isStopNode}) \leftarrow \text{PredictEdge}(f, \mathcal{G})$
    if not isStopNode **and** FirstLink(i) **then**
      $\mathbf{r}_u = (x_u, y_u, z_u) \leftarrow \text{PredictCoords}(i)$
      $\mathcal{V} \leftarrow \mathcal{V} \cup \{i\}, \mathcal{E} \leftarrow \mathcal{E} \cup \{(f, i)\},$
      $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbf{r}_i\}$
      Q.push(i)
    end if
    MarkVisited(f)
  end while
end while
Return: Reconstructed $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R})$



Property-guided generation

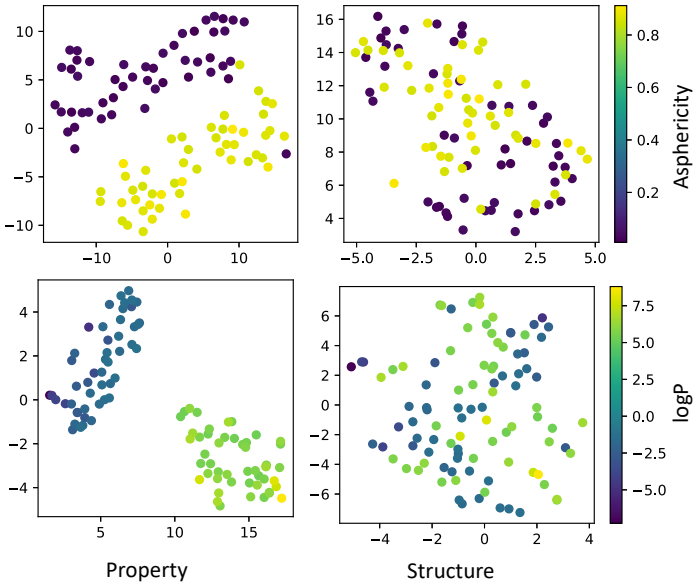
Isolated molecule

		Asphericity	QED	SAS	logP
		MSE/MAE	MSE/MAE	MSE/MAE	MSE/MAE
GEOM	EDM	0.626/0.455	0.113/0.285	12.395/3.385	5.704/1.903
	HierDiff	0.176/0.406	0.120/0.289	2.618/1.347	**4.124/1.572**
	Ours	**0.095/0.246**	**0.072/0.221**	**1.563/1.002**	4.490/1.630
Cross-Docked 2020	EDM	0.109/0.274	0.147/0.309	11.244/3.205	5.715/1.957
	HierDiff	0.107/0.268	0.089/0.278	2.364/1.468	5.752/1.897
	Ours	**0.100/0.259**	**0.062/0.205**	**2.356/1.243**	**4.244/1.644**

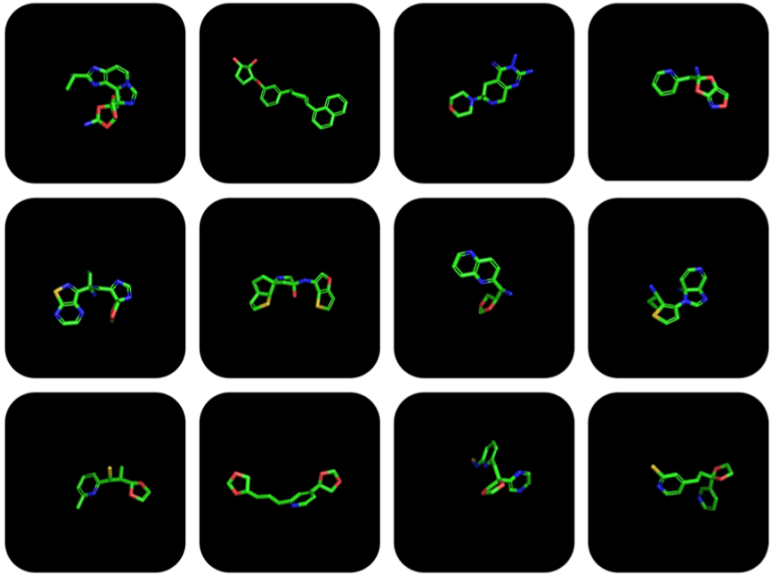
Ligand

	Asphericity			QED			SAS			logP		
	Vina	MSE	MAE	Vina	MSE	MAE	Vina	MSE	MAE	Vina	MSE	MAE
HierDiff	-4.253	0.125	0.296	-4.429	0.113	0.275	-5.053	2.364	1.620	-4.293	5.938	1.855
TargetDiff	-5.742	0.117	0.288	-5.706	0.112	0.307	-5.479	3.369	1.604	-5.501	4.509	1.946
Ours	**-5.891**	**0.102**	**0.271**	**-5.866**	**0.086**	**0.247**	**-5.940**	**2.358**	**1.529**	**-5.827**	**4.376**	**1.794**

Disentanglement analysis

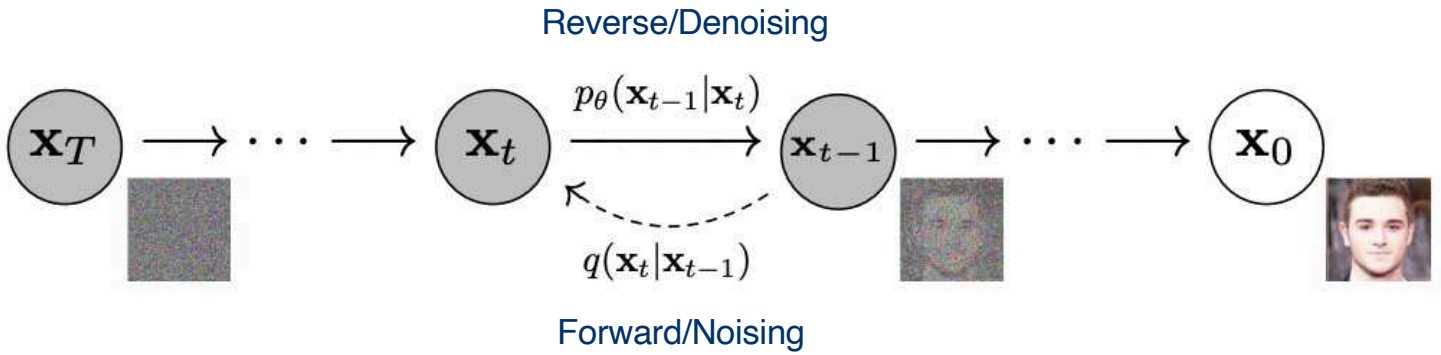


Generated molecules



Diffusion Probabilistic Model

- Forward process (“noising”): corrupt the data signal into random noise
- Reverse process (“denoising”): generate data from random noise
- Train a “noise estimator” to predict the noise added at each step

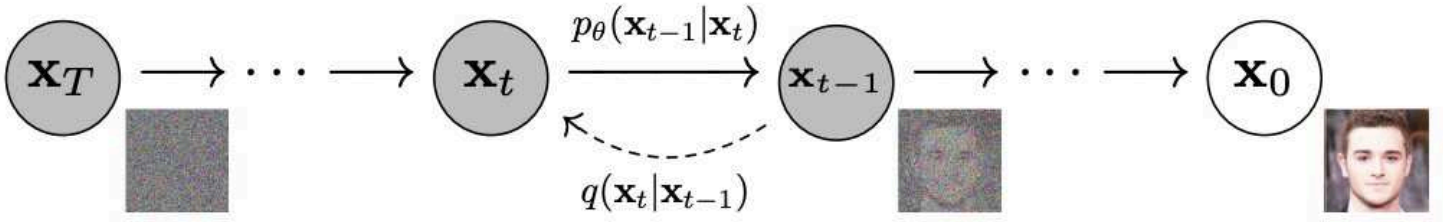


Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics. ICML 2015.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in neural information processing systems* 33 (2020): 6840-6851.

Diffusion Probabilistic Model

Reverse/Denoising



Forward/Noising

$\beta_1, \dots, \beta_T \in (0, 1)$ are fixed constants.
$\alpha_t := 1 - \beta_t$
$\bar{\alpha}_t := \alpha_1 \cdots \alpha_t$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

A **forward diffusion process** starts at some starting point $x_0 \sim q$, where q is the probability distribution to be learned, then repeatedly adds noise to it by

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}z_t$$

where $z_1, \dots, z_T$ are IID samples from $\mathcal{N}(0, I)$. This is designed so that for any starting distribution of x_0, we have $\lim_t x_t|x_0$ converging to $\mathcal{N}(0, I)$.

The key idea of DDPM is to use a neural network parametrized by θ. The network takes in two arguments x_t, t, and outputs a vector $\mu_\theta(x_t, t)$ and a matrix $\Sigma_\theta(x_t, t)$, such that each step in the forward diffusion process can be approximately undone by $x_{t-1} \sim \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$. This then gives us a backward diffusion process p_θ defined by

$$p_\theta(x_T) = \mathcal{N}(x_T|0, I)$$
$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}|\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics. ICML 2015.
Ho, Jonathan, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models." *Advances in neural information processing systems* 33 (2020): 6840-6851.

Diffusion model

For training, we can form variational upper bound that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020 show that:

$$L = \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1))}_{L_0}\right]$$

where $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}),$$

$$\text{where } \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \text{ and } \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

Diffusion Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C$$

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$. Ho et al. NeurIPS 2020 observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(\mathbf{x}_t, t)\right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{\mathbf{x}_t}, t)||^2\right] + C$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Diffusion Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q\left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C$$

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\,\epsilon$. Ho et al. NeurIPS 2020 observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\epsilon_\theta(\mathbf{x}_t, t)\right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0\sim q(\mathbf{x}_0),\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon}_{\mathbf{x}_t}, t)||^2\right] + C$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Diffusion Model with Classifier Guidance

$$\nabla_x \ln p(x|y) = \underbrace{\nabla_x \ln p(x)}_{\text{score}} + \underbrace{\nabla_x \ln p(y|x)}_{\text{classifier guidance}}$$

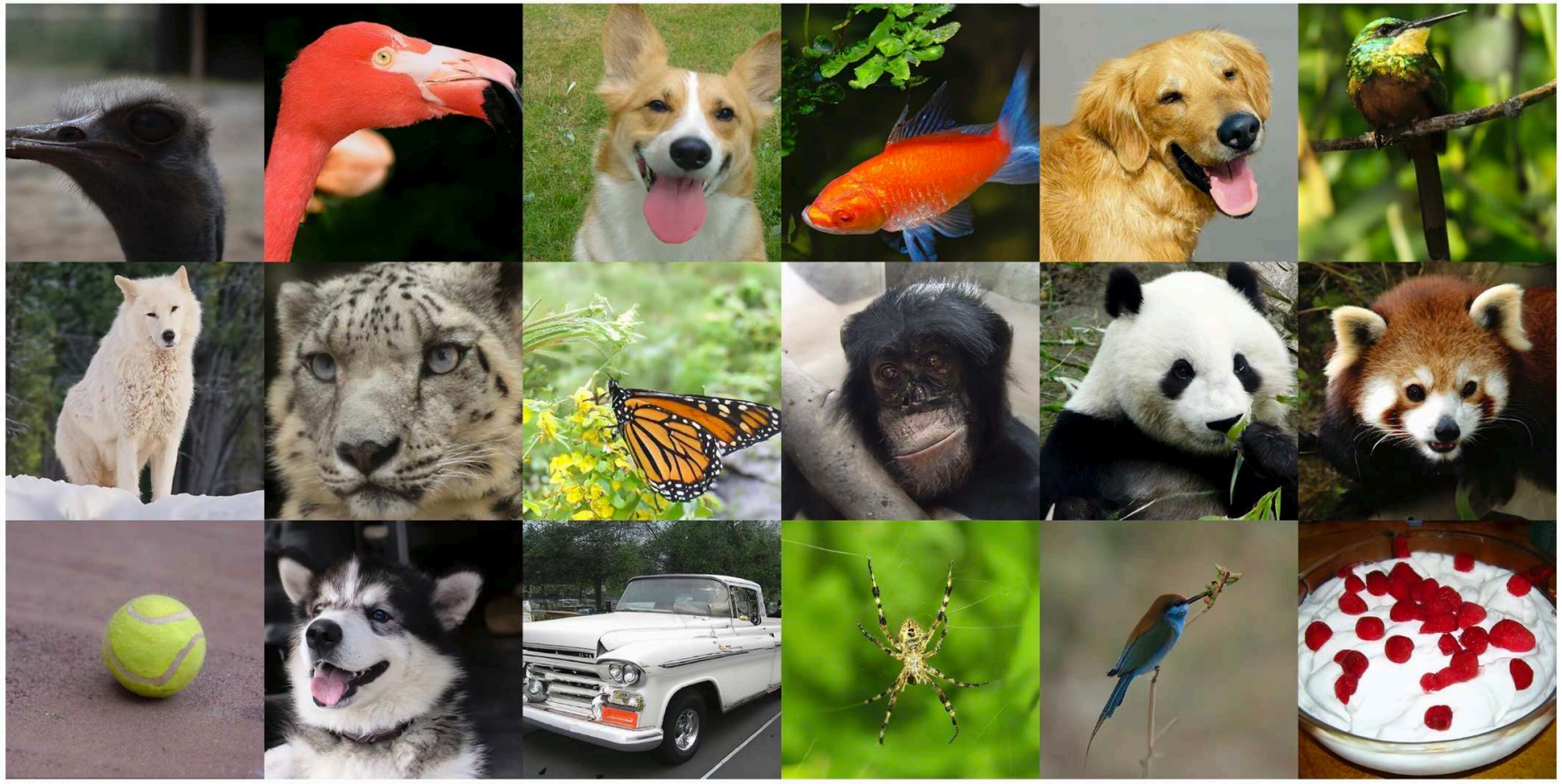
Algorithm 1 Classifier guided diffusion sampling, given a diffusion model $(\mu_\theta(x_t), \Sigma_\theta(x_t))$, classifier $p_\phi(y|x_t)$, and gradient scale s.

Input: class label y, gradient scale s
$x_T \leftarrow$ sample from $\mathcal{N}(0, \mathbf{I})$
for all t from T to 1 **do**
  $\mu, \Sigma \leftarrow \mu_\theta(x_t), \Sigma_\theta(x_t)$
  $x_{t-1} \leftarrow$ sample from $\mathcal{N}(\mu + s\Sigma \nabla_{x_t} \log p_\phi(y|x_t), \Sigma)$
end for
return x_0

For classifier-free guidance, we simply train the diffusion model by presenting both (x, y) and (x, None)

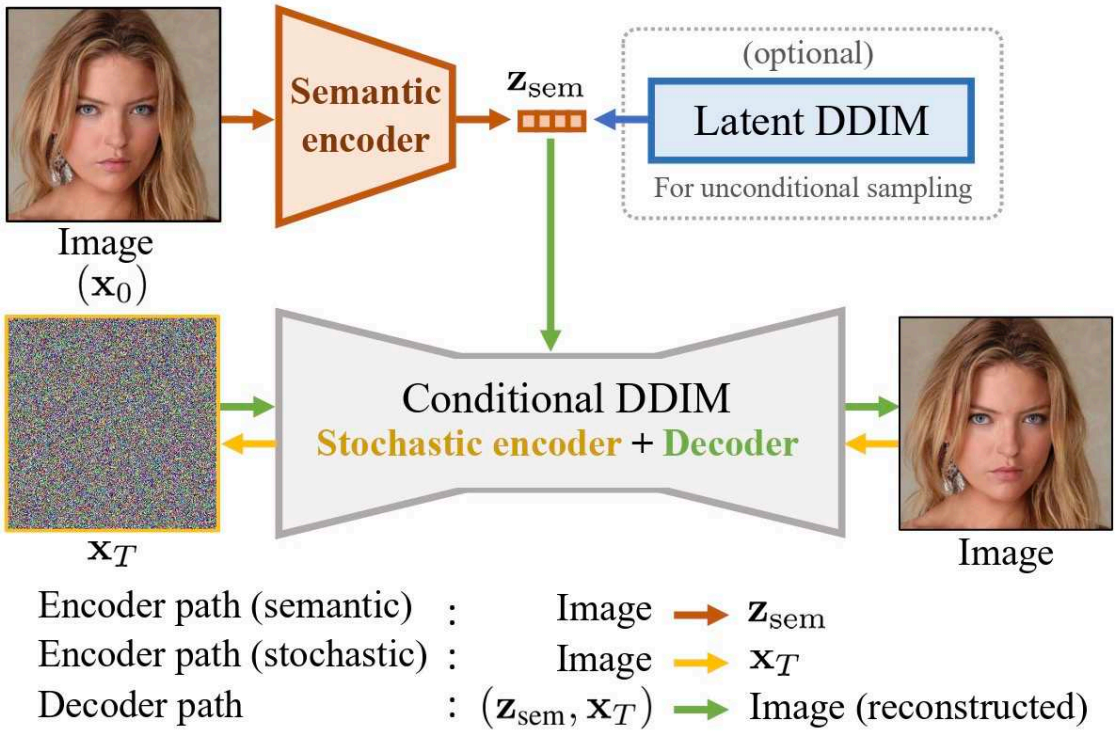
Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. NeurIPS 2021. https://arxiv.org/abs/2105.05233

Diffusion Model with Classifier Guidance



Prafulla Dhariwal and Alex Nichol. Diffusion Models Beat GANs on Image Synthesis. NeurIPS 2021. https://arxiv.org/abs/2105.05233

Diffusion Autoencoder



Preechakul et al., Diffusion Autoencoders: Toward a Meaningful and Decodable Representation. CVPR 2022.
https://openaccess.thecvf.com/content/CVPR2022/papers/Preechakul_Diffusion_Autoencoders_Toward_a_Meaningful_and_Decodable_Representation_CVPR_2022_paper.pdf

Compositional Generation with Diffusion Autoencoder

Compositionally manipulate multiple attributes in generation

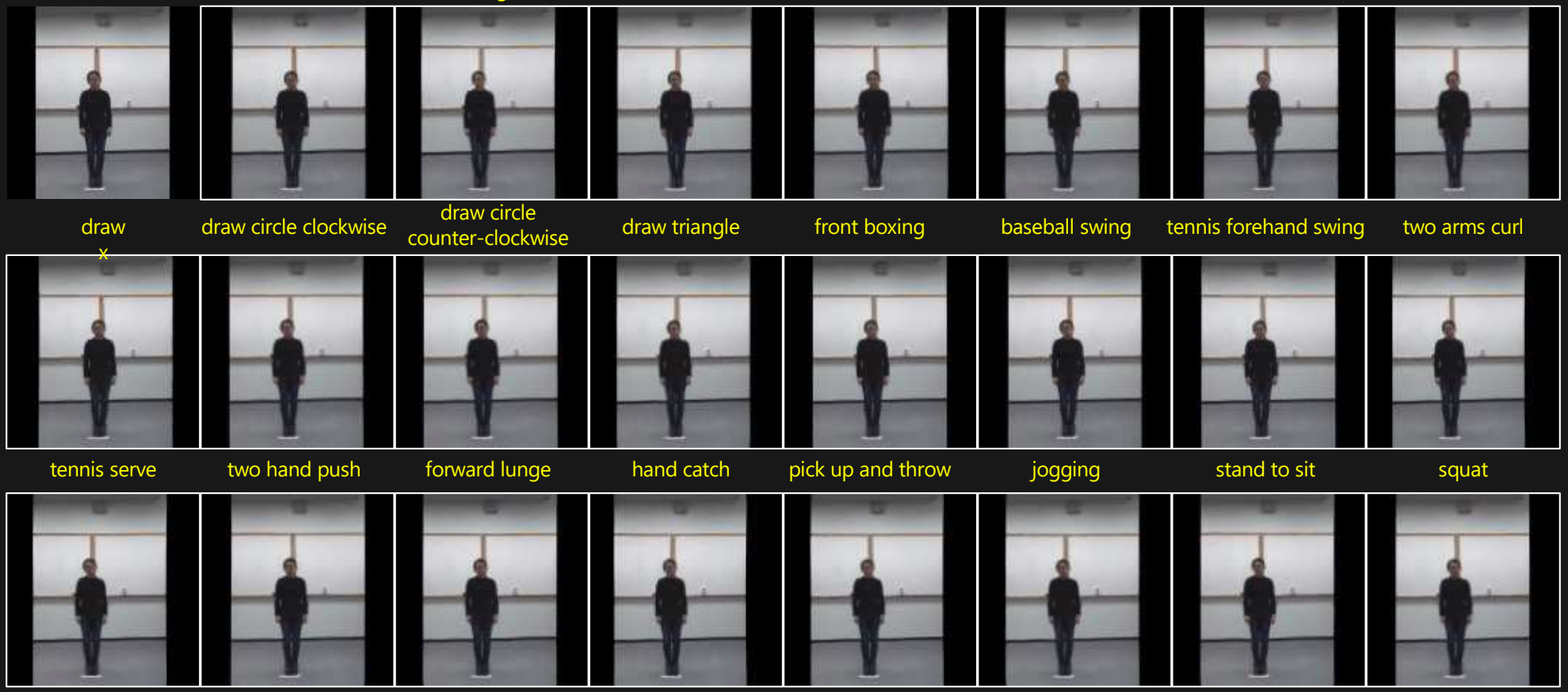
Three attributes: smiling, young, wavy hair.
The middle figure is from unconditional generation.
—> is the + direction.

Compositional Generation with Diffusion Autoencoder

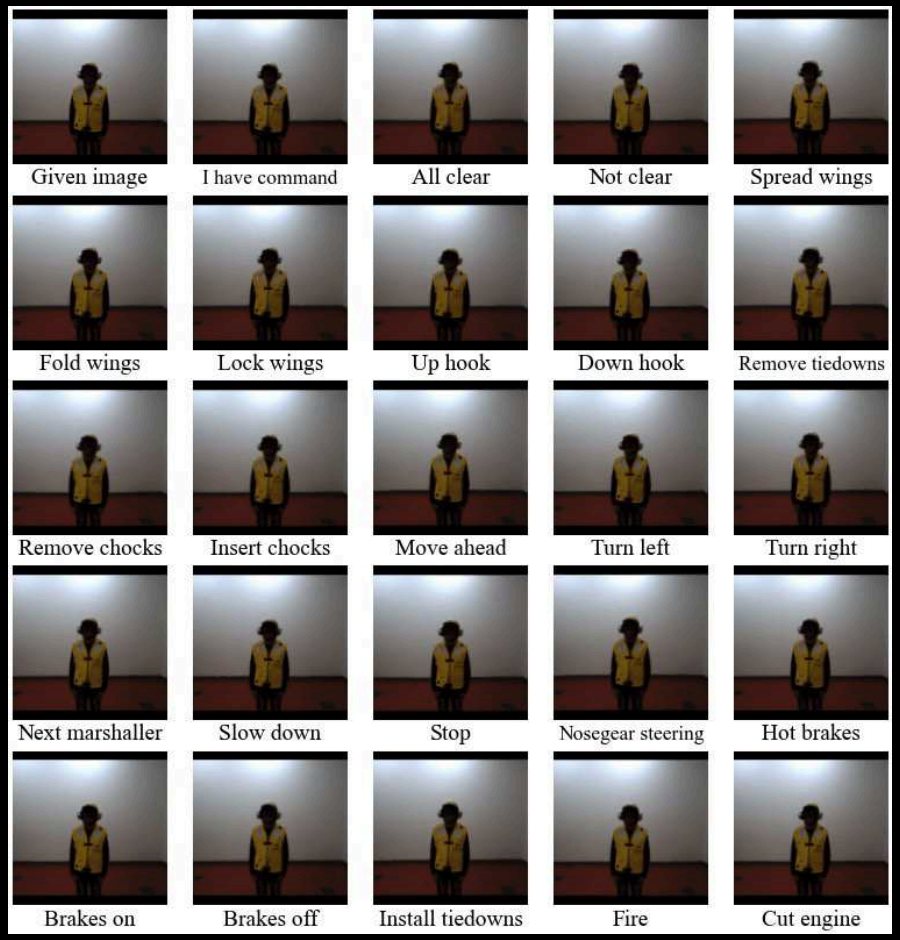
Multi-level manipulation

Multi-attribute manipulation

Text-Conditioned Image-to-Video Generation with Latent Feature Diffusion



Text-Conditioned Image-to-Video Generation with Latent Feature Diffusion



Text-Conditioned Image-to-Video Generation with Latent Feature Diffusion

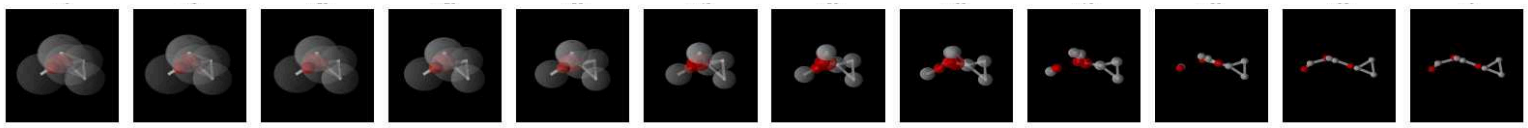
Expression generation

Equivariant diffusion model

- Consider the Euclidean group E(3) (translation, rotation, reflection)
  - Let f be some function
  - g is some E3 transformation, T_g and S_g are its linear representations
  - Equivariant: $T_g(f(x)) = f\left(T_g(x)\right)$
  - Invariant: $f(h) = f\left(S_g(h)\right)$

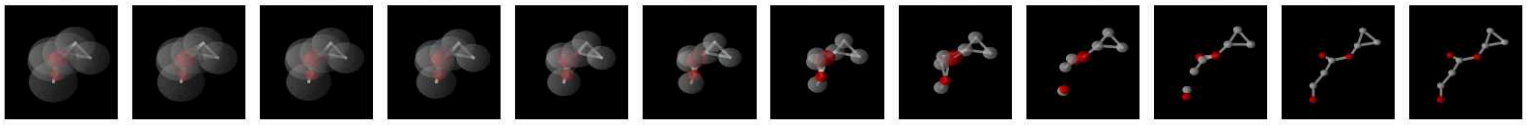
Equivariant diffusion model

- Denoising process with equivariant constraints
  - The equivariant denoising process produces distributions **that are invariant to transformations**
  - Increases data efficiency
  - Useful in 3D objects like molecules



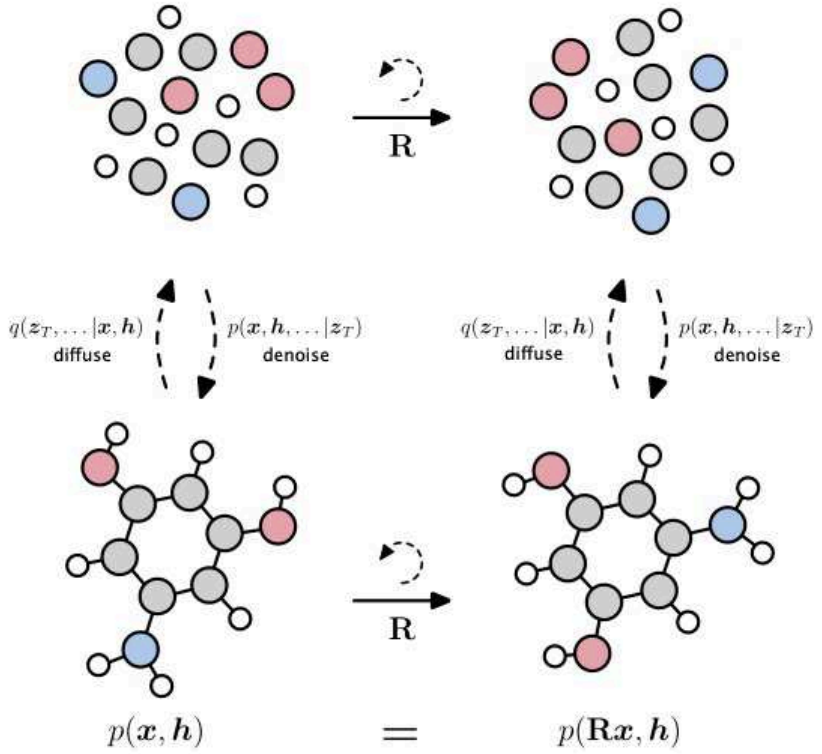
$p(x_0)$

=



$p(Rx_0)$

Equivariant diffusion model



Algorithm 1 Optimizing EDM

Input: Data point $\boldsymbol{x}$, neural network ϕ
Sample $t \sim \mathcal{U}(0, \ldots, T)$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Subtract center of gravity from $\boldsymbol{\epsilon}^{(x)}$ in $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}^{(x)}, \boldsymbol{\epsilon}^{(h)}]$
Compute $\boldsymbol{z}_t = \alpha_t[\boldsymbol{x}, \boldsymbol{h}] + \sigma_t \epsilon$
Minimize $||\boldsymbol{\epsilon} - \phi(\boldsymbol{z}_t, t)||^2$

Algorithm 2 Sampling from EDM

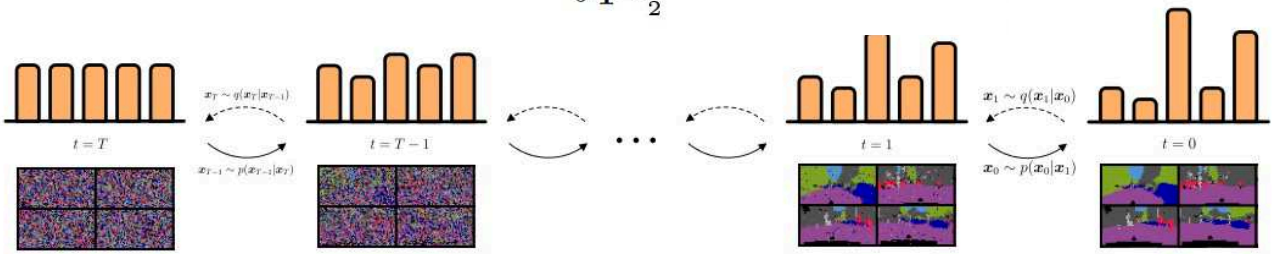
Sample $\boldsymbol{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for t in $T, T-1, \ldots, 1$ where $s = t - 1$ **do**
  Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Subtract center of gravity from $\boldsymbol{\epsilon}^{(x)}$ in $\boldsymbol{\epsilon} = [\boldsymbol{\epsilon}^{(x)}, \boldsymbol{\epsilon}^{(h)}]$
  $\boldsymbol{z}_s = \frac{1}{\alpha_{t|s}} \boldsymbol{z}_t - \frac{\sigma^2_{t|s}}{\alpha_{t|s} \sigma_t} \cdot \phi(\boldsymbol{z}_t, t) + \sigma_{t \to s} \cdot \boldsymbol{\epsilon}$
end for
Sample $\boldsymbol{x}, \boldsymbol{h} \sim p(\boldsymbol{x}, \boldsymbol{h} | \boldsymbol{z}_0)$

Hoogeboom, Emiel, et al. "Equivariant diffusion for molecule generation in 3d." *International conference on machine learning*. PMLR, 2022.

Equivariant diffusion model

- Discrete diffusion for categorical data (e.g. atom type, amino acid)

$$p(\boldsymbol{h}|\boldsymbol{z}_0^{(h)}) = \mathcal{C}(\boldsymbol{h}|\mathbf{p}), \mathbf{p} \propto \int_{\mathbf{1}-\frac{1}{2}}^{\mathbf{1}+\frac{1}{2}} \mathcal{N}(\boldsymbol{u}|\boldsymbol{z}_0^{(h)}, \sigma_0)\mathrm{d}\boldsymbol{u}$$



$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathcal{C}(\boldsymbol{x}_t|(1-\beta_t)\boldsymbol{x}_{t-1} + \beta_t/K),$$

$$q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathcal{C}(\boldsymbol{x}_t|\bar{\alpha}_t\boldsymbol{x}_0 + (1-\bar{\alpha}_t)/K)$$

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \mathcal{C}(\boldsymbol{x}_{t-1}|\boldsymbol{\theta}_{\text{post}}(\boldsymbol{x}_t, \boldsymbol{x}_0)), \quad \text{where} \quad \boldsymbol{\theta}_{\text{post}}(\boldsymbol{x}_t, \boldsymbol{x}_0) = \tilde{\boldsymbol{\theta}} / \sum_{k=1}^{K} \tilde{\theta}_k$$

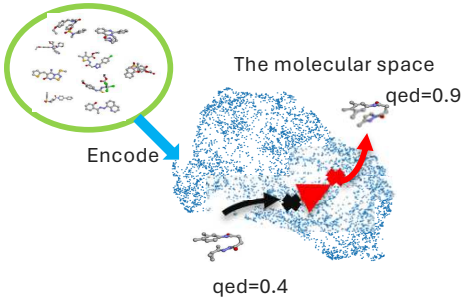
$$\text{and } \tilde{\boldsymbol{\theta}} = [\alpha_t\boldsymbol{x}_t + (1-\alpha_t)/K] \odot [\bar{\alpha}_{t-1}\boldsymbol{x}_0 + (1-\bar{\alpha}_{t-1})/K].$$

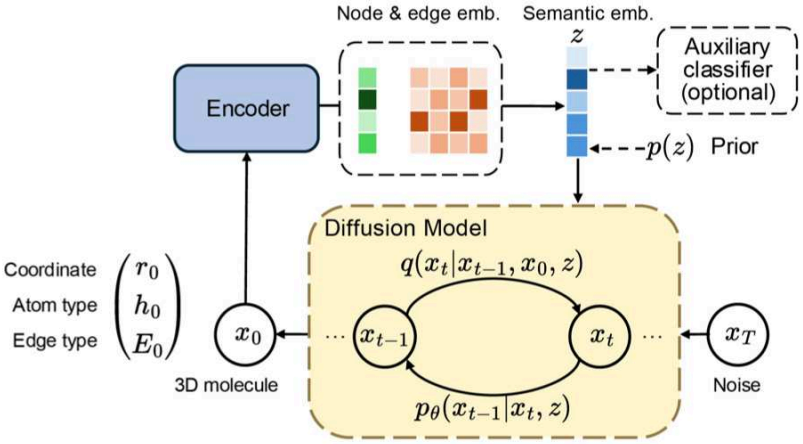
Hoogeboom, Emiel, et al. "Equivariant diffusion for molecule generation in 3d." *International conference on machine learning*. PMLR, 2022.

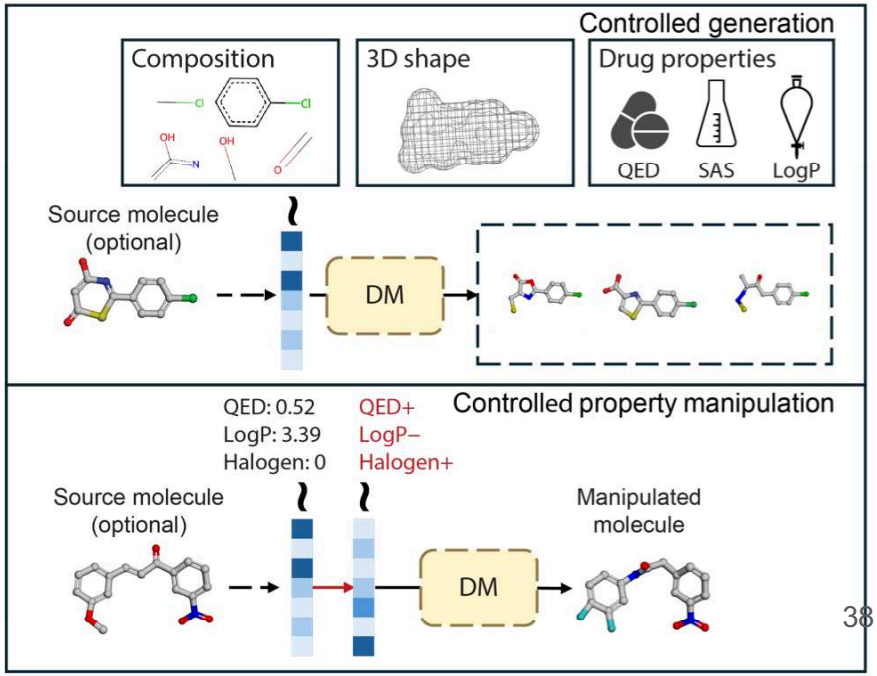
Hoogeboom, Emiel, et al. "Argmax flows and multinomial diffusion: Learning categorical distributions." *Advances in neural information processing systems* 34 (2021): 12454-12465.

Controlled Generation of 3D Molecules

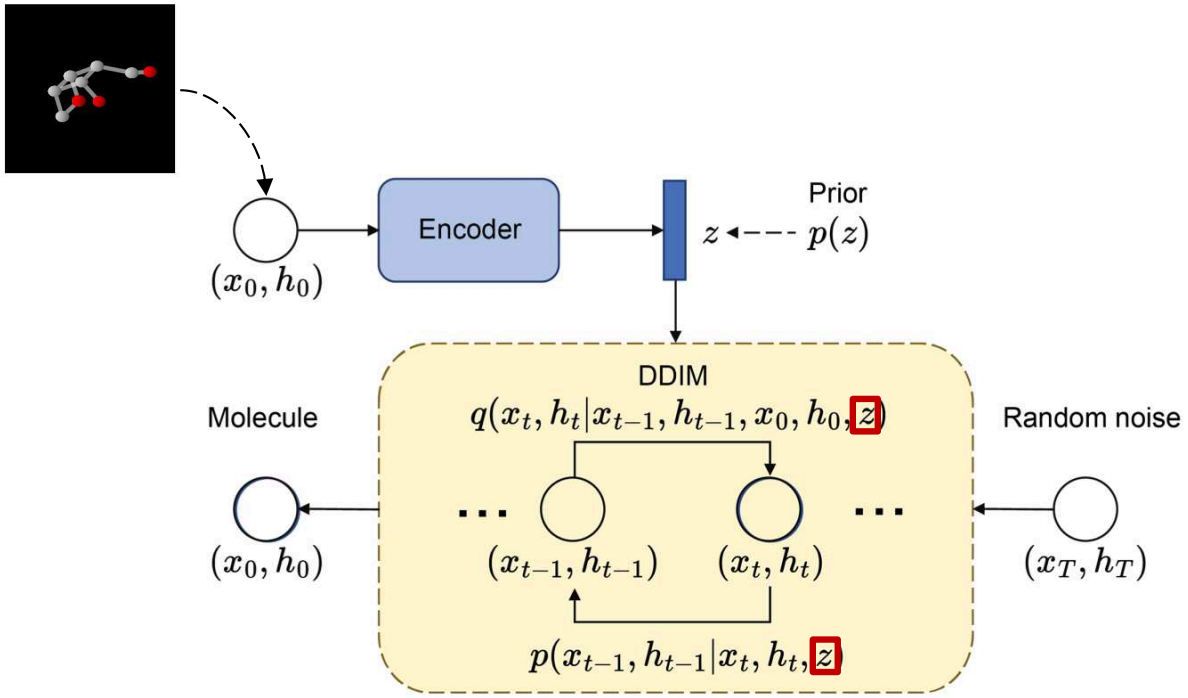
- Controlling the equivariant diffusion model for 3D molecule generation
  - Semantic control on diffusion models
  - Disentanglement of the embedding space





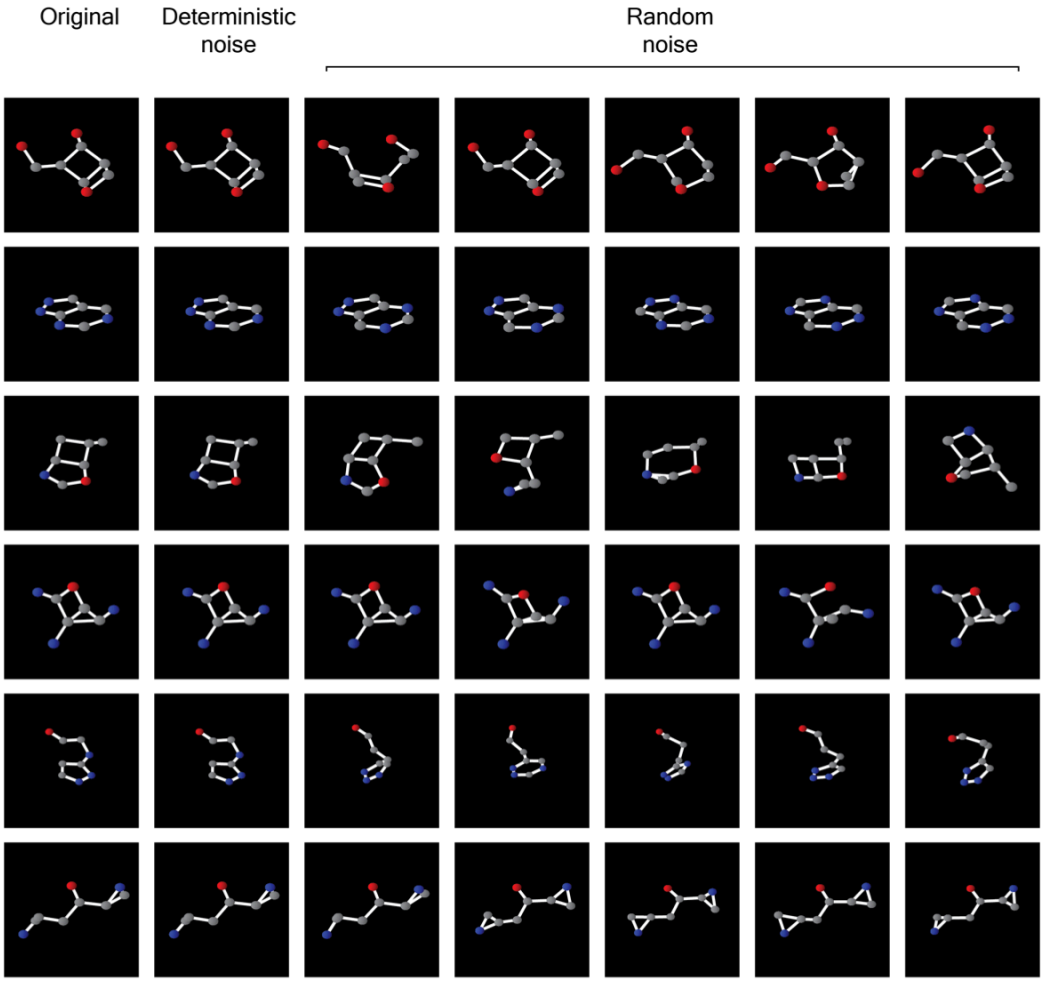


Semantics-guided diffusion model

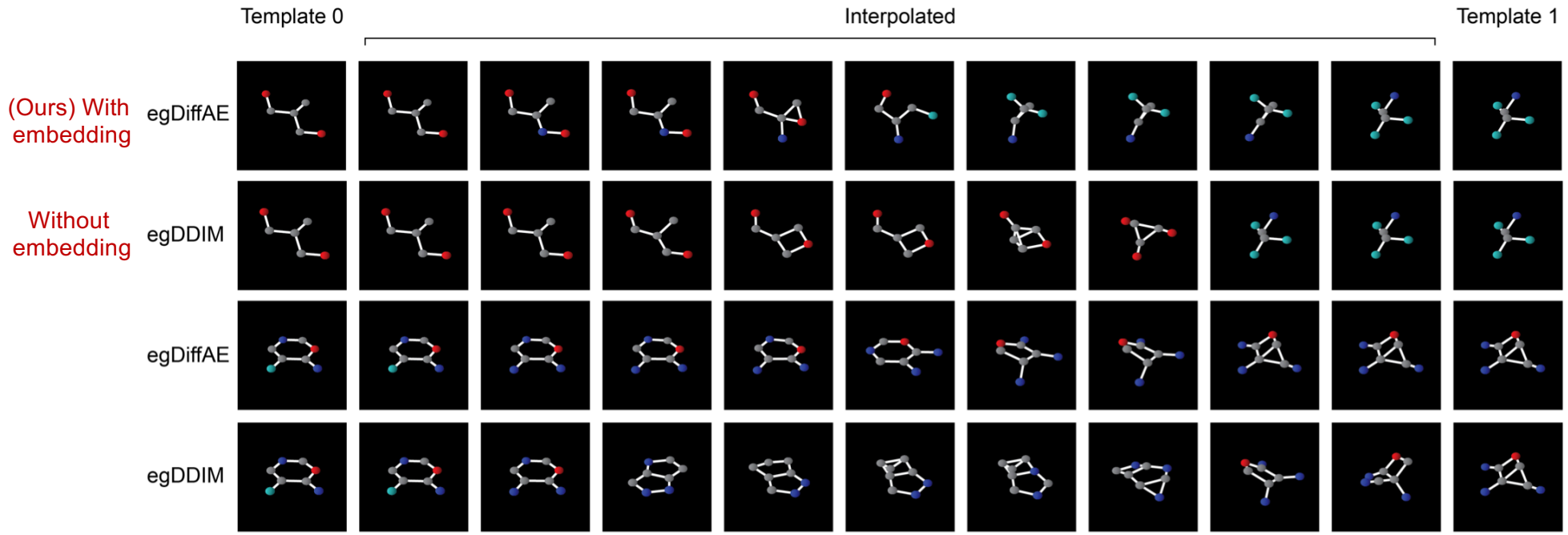


Equivariant diffusion model

Semantic guidance for controllable generation



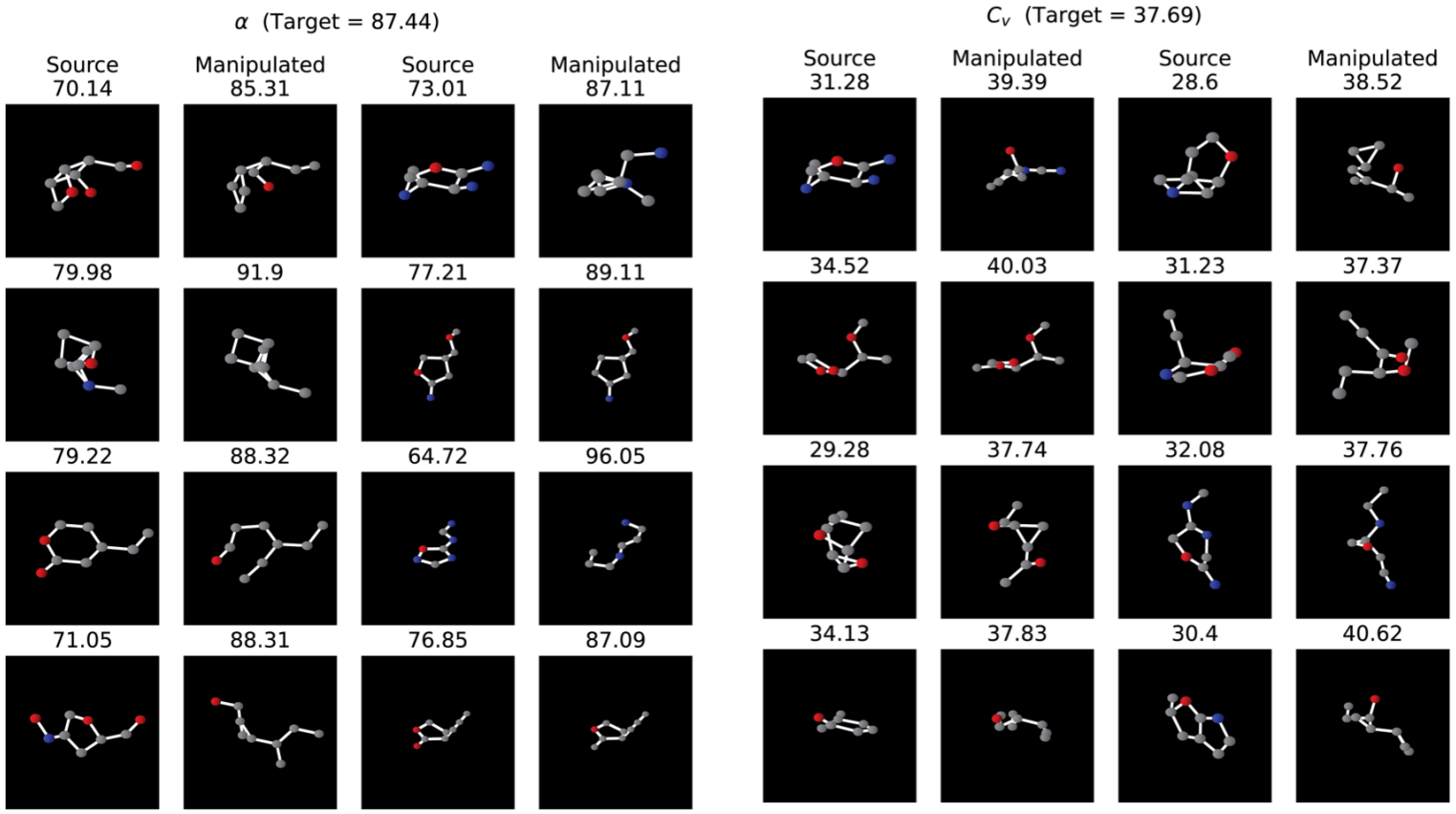
Preserving the molecular semantics during generation



$$h_T^{(\lambda)} = \lambda h_T^{(0)} + (1-\lambda) h_T^{(1)},\ x_T^{(\lambda)} = \lambda x_T^{(0)} + (1-\lambda) x_T^{(1)}$$

$$z^{(\lambda)} = \lambda z^{(0)} + (1-\lambda) z^{(i)}$$

Smooth interpolation between 3D structures



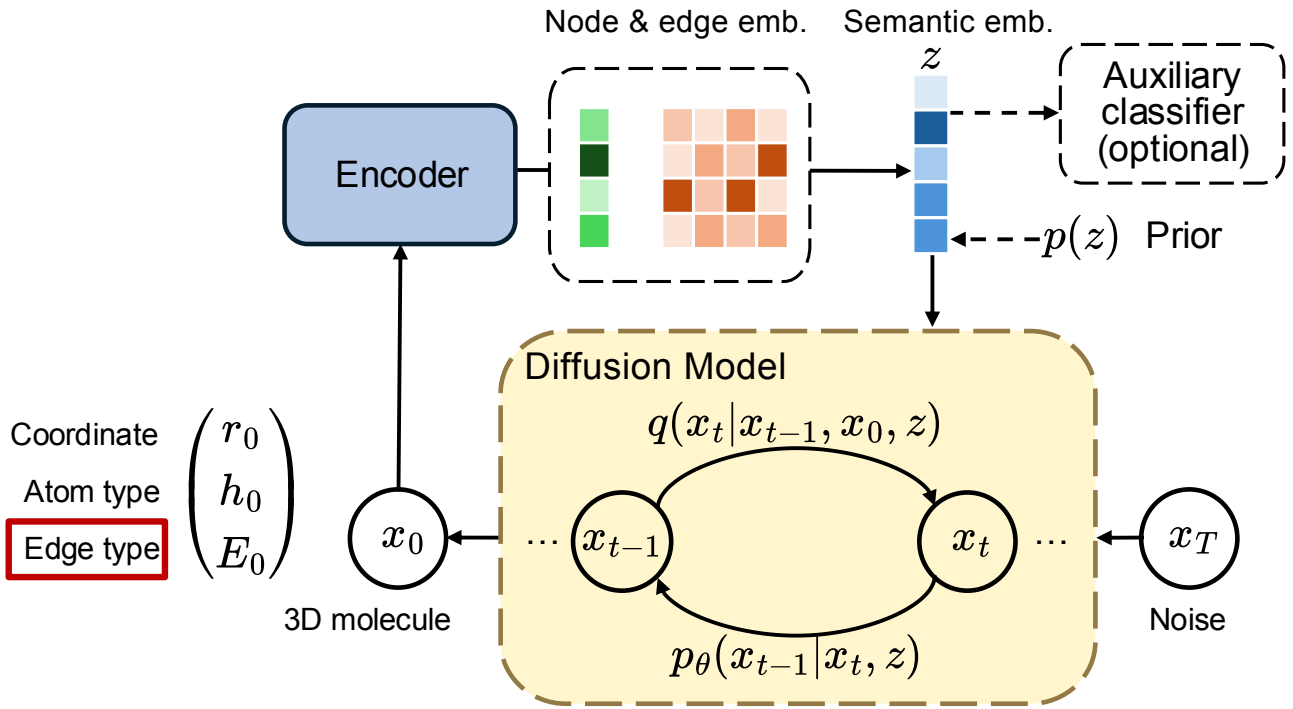
$$y = wz + b$$

$$s = \frac{y^t - b - z^T w}{w^T w}$$

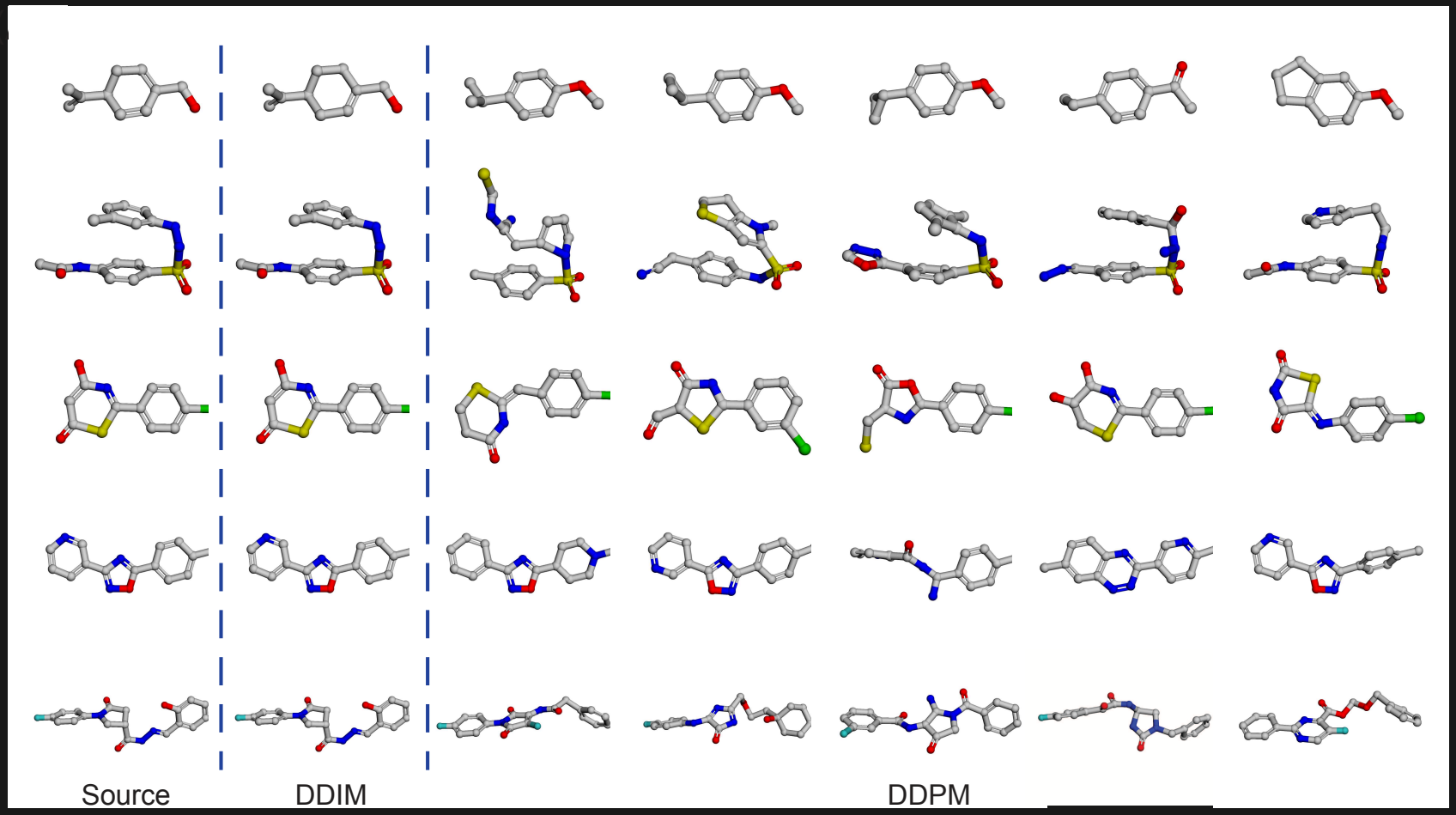
$$z^t = z + sw$$

Manipulate molecular properties while maintaining the semantics

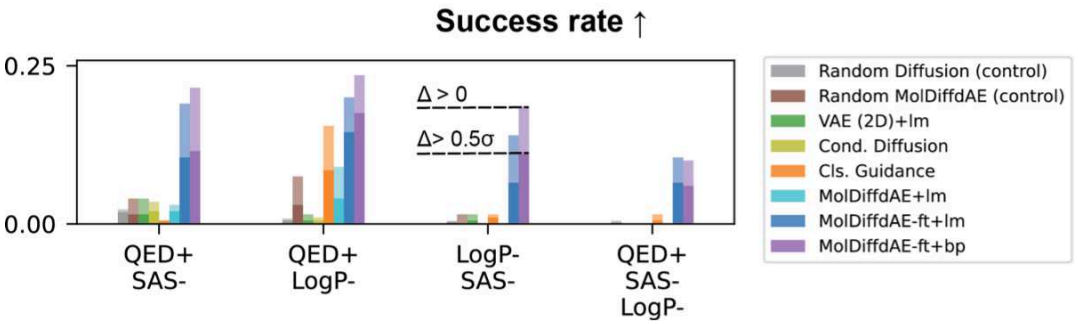
Semantics-guided diffusion model

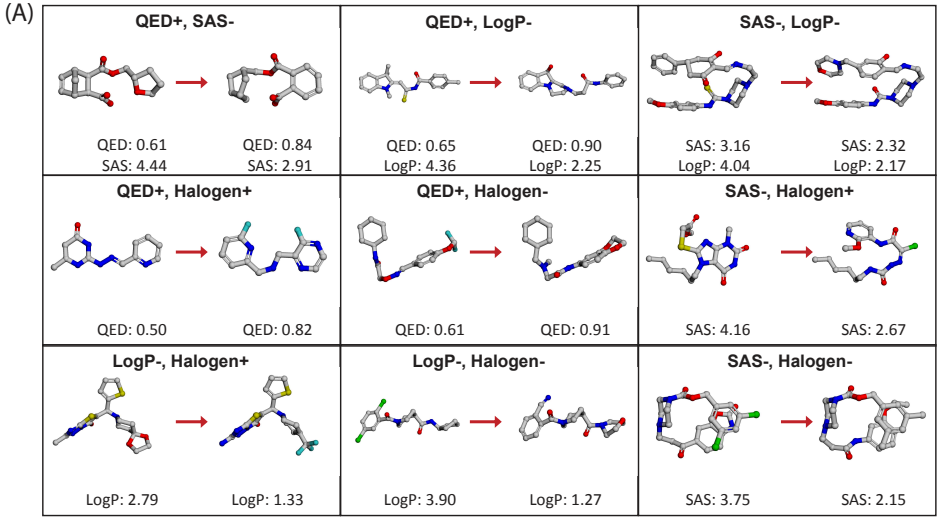


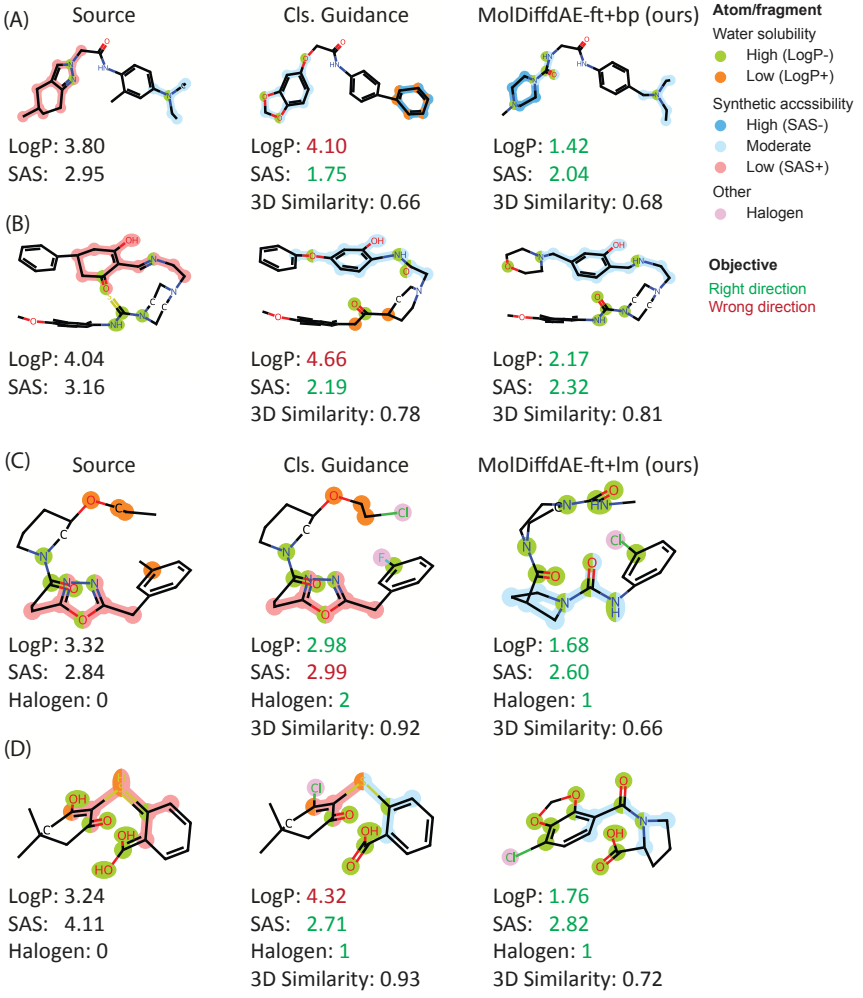
Semantic guidance for controllable generation



Preserving the molecular semantics during generation

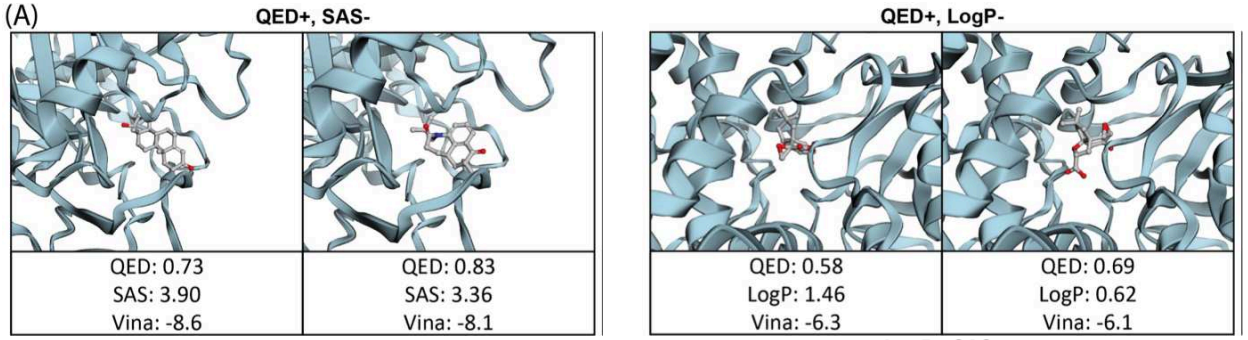




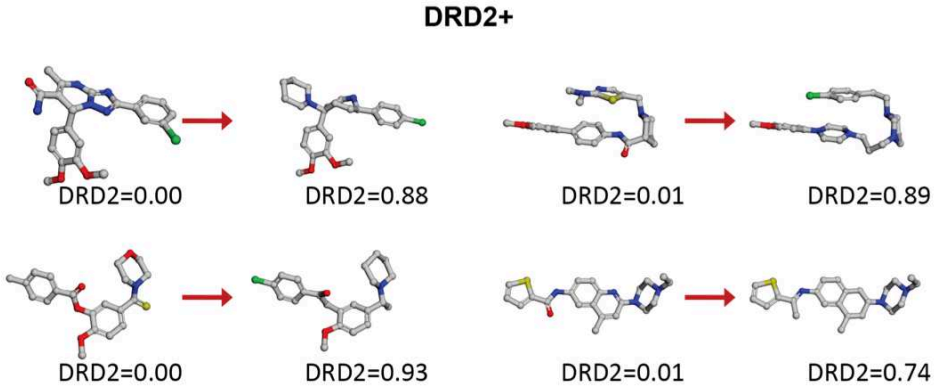


Efficient multi-objective manipulation

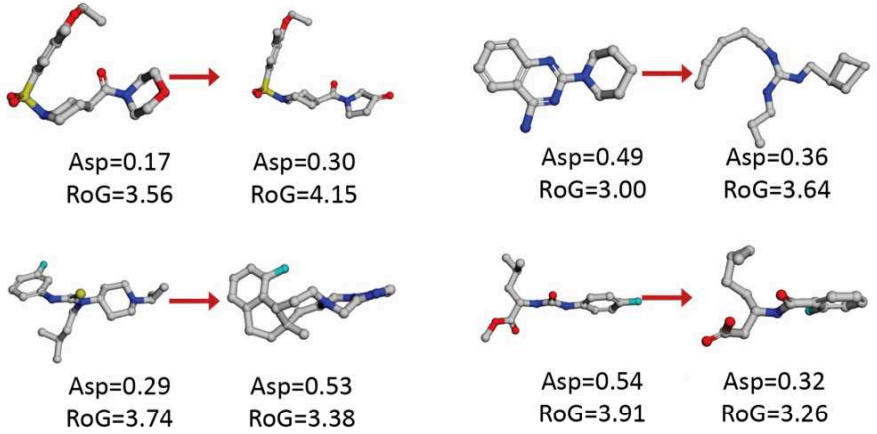
(1) Binding-preserving manipulation



(2) Data efficient binding affinity enhancement
(~1000 positive training data)



(3) Manipulation of 3D shapes and properties



Efficient multi-objective manipulation

Robust Design with Latent-Variable Generative Models

$$p(\mathbf{x}|S,\boldsymbol{\theta}^{(0)}) = \frac{P(S|\mathbf{x})p(\mathbf{x}|\boldsymbol{\theta}^{(0)})}{P(S|\boldsymbol{\theta}^{(0)})}, \quad (1)$$

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}}\, D_{KL}\left(p(\mathbf{x}|S,\boldsymbol{\theta}^{(0)})||q(\mathbf{x}|\boldsymbol{\phi})\right) \quad (2)$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} -\mathbb{E}_{p(\mathbf{x}|S,\boldsymbol{\theta}^{(0)})}[\log q(\mathbf{x}|\boldsymbol{\phi})] - H_0 \quad (3)$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \frac{1}{P(S|\boldsymbol{\theta}^{(0)})}\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta}^{(0)})}[P(S|\mathbf{x})\log q(\mathbf{x}|\boldsymbol{\phi})] \quad (4)$$

$$= \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\, \mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta}^{(0)})}[P(S|\mathbf{x})\log q(\mathbf{x}|\boldsymbol{\phi})], \quad (5)$$

$$\mathbb{E}_{p(\mathbf{x}|\boldsymbol{\theta}^{(0)})}[P(S^{(t)}|\mathbf{x})\log q(\mathbf{x}|\boldsymbol{\phi})]$$
$$= \mathbb{E}_{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta}^{(0)})}[P(S^{(t)}|\mathbf{x})\log q(\mathbf{x}|\boldsymbol{\phi})]$$
$$= \mathbb{E}_{q(\mathbf{x},\mathbf{z}|\boldsymbol{\phi}^{(t)})}\left[\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta}^{(0)})}{q(\mathbf{x},\mathbf{z}|\boldsymbol{\phi}^{(t)})}P(S^{(t)}|\mathbf{x})\log q(\mathbf{x}|\boldsymbol{\phi})\right]$$

Given a prior data distribution p(x) that can be trained with a VAE, WAE, or Diffusion Probabilistic Model, how to sample from p(x | y >= desirable value)

Challenges: we are moving out of the comfortable zone of p(x) because y_x >= desirable value is rare

Algorithm 1. **Maximization of a single, continuous property.** $h_{\text{oracle}}(\mathbf{x}_i)$ is a function returning the expected value of the property oracle. $\text{CDF}_{\text{oracle}}(\mathbf{x}, \gamma)$ is a function to compute the CDF of the oracle predictive model for threshold γ. $\text{GenProb}(\mathbf{x}_i, \mathbf{z}_i, \boldsymbol{\theta})$ is a method that evaluates the probability of an input in a generative model with parameters $\boldsymbol{\theta}$. $\text{GenTrain}(\{(\mathbf{x}_i, w_i)\}$ is a procedure to take weighted training data $\{(\mathbf{x}_i, w_i)\}$, and return the parameters of the trained model. Q is a parameter that determines the next iteration's relaxation threshold; M is the number of samples to generate at each iteration. See main text for convergence criteria. $\{\mathbf{x}_{\text{init}}\}$ is an initial set of samples with which to train the prior. Any line with an i subscript implicitly denotes $\forall i \in [1 \ldots M]$.

procedure $CbAS(h_{\text{oracle}}(\mathbf{x}), \text{CDF}_{\text{oracle}}(\mathbf{x}, \gamma), \text{GenTrain}(\{\mathbf{x}_i\, w_i\}), \text{GenProb}(\mathbf{x}_i), [Q = 0.9], [M = 100])$
- $\boldsymbol{\theta}^{(0)} \leftarrow \text{GenTrain}(\{\mathbf{x}_{\text{init}}\}, w_i = 1)$
- $\boldsymbol{\phi}^{(1)} \leftarrow \boldsymbol{\theta}^{(0)}$
- $t \leftarrow 1$
- **while** not converged **do**
  - $set_i \leftarrow \mathbf{x}_i, \mathbf{z}_i \sim G(\boldsymbol{\phi}^{(t)})$
  - $scores_i \leftarrow h_{\text{oracle}}(\mathbf{x}_i)$
  - $q \leftarrow$ index of Q^{th} percentile of $scores$
  - $\gamma^{(t)} \leftarrow scores_q$
  - $weights_i \leftarrow \frac{\text{GenProb}(set_i, \boldsymbol{\theta}^{(0)})}{\text{GenProb}(set_i, \boldsymbol{\phi}^{(t)})}$
  - $weights_i \leftarrow weights_i * (1 - CDF_{\text{oracle}}(\mathbf{x}_i, \gamma^{(t)}))$
  - $\boldsymbol{\phi}^{(t)} =\leftarrow \text{GenTrain}(set, weights)$
  - $t \leftarrow t + 1$
- **return** $set, weights$

David Brookes, Hahnbeom Park, Jennifer Listgarten. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:773-782, 2019.

The End