Adv ML for Gen-AI

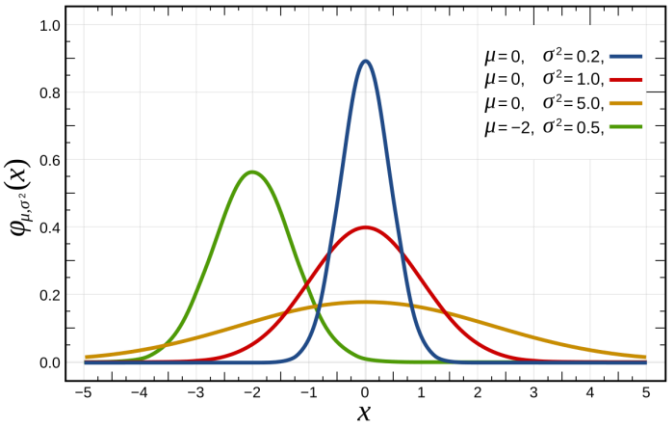
Discrete Generative Model

https://ml-graph.github.io/spring-2026/

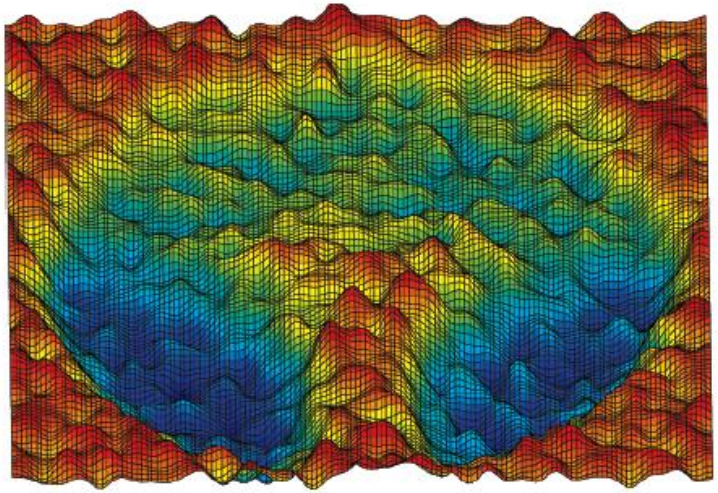
Yu Wang, Ph.D.
Assistant Professor
Department of Computer Science
University of Oregon
Personal: https://yuwang0103.github.io/
Lab: https://kindlab-fly.github.io/

Summary – Distribution

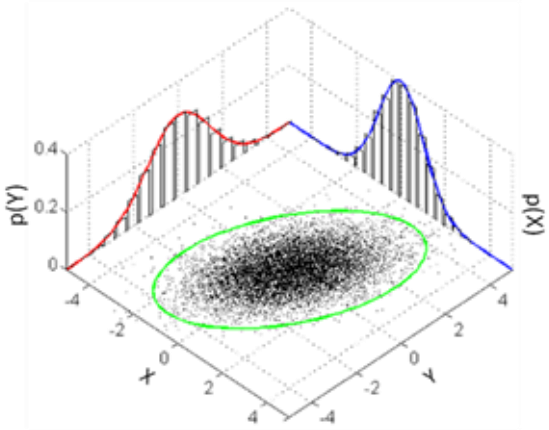
1D Gaussian Distribution



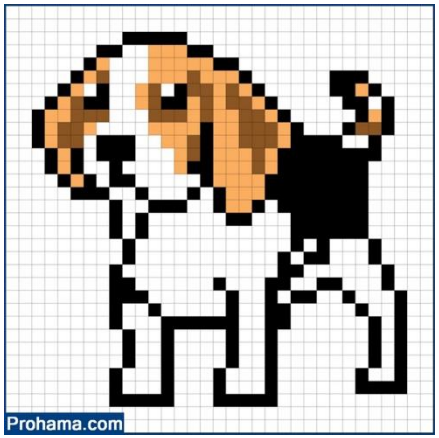
$\mathbb{R}$

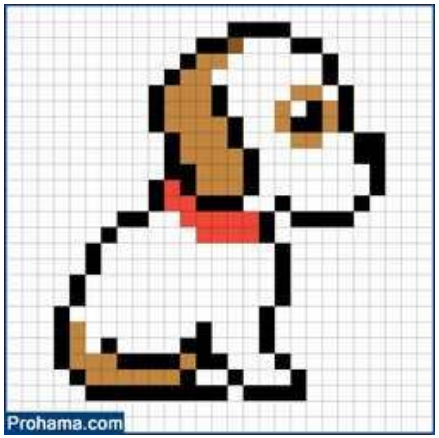


2D Gaussian Distribution



$\mathbb{R}^2$





$\mathbb{R}^{256\times256}$

Summary – Existing Generative Methods

Probability distribution of the **objective** based on the **observed data**

$$\{x_i\}_{i=1}^{N} \xrightarrow[\text{Good Model}]{} P(x) \xrightarrow[\text{Good Data}]{} x$$

- **Machine Learning Methods**
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

Using existing function to estimate what you do not know that can best fit your observation

- **Deep Learning Methods**
  - Auto-Encoder (AE)
  - Variational AE (VAE)
  - Generative Adversarial Network (GAN)
  - Diffusion Model
  - ……

Using learnable function to estimate what you do not know that can best fit your observation

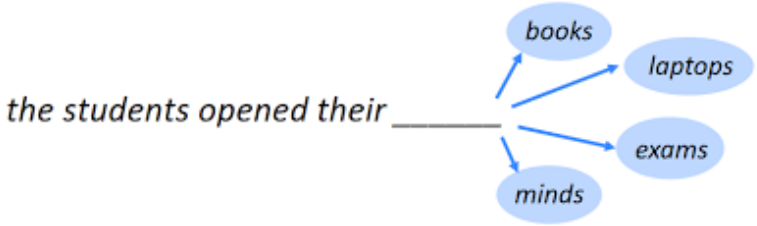
Summary – Continuous Generation

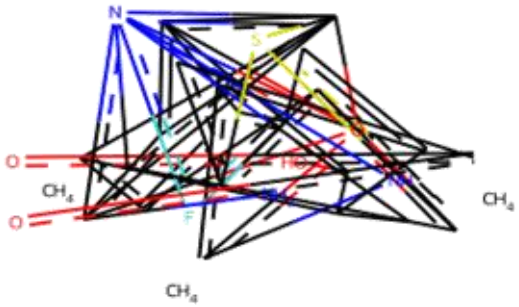
(a) MNIST ($t = 1.0$) (b) CIFAR-10 ($t = 0.7$) (c) CelebA 64 ($t = 0.6$)

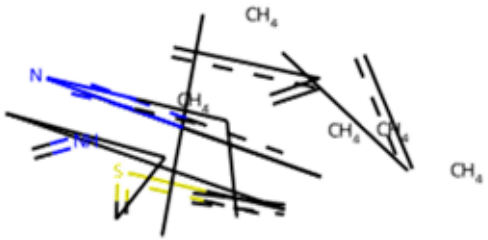
They are all continuous

Problem – Real-world Generation is sometimes discrete

Language Generation







Graph Generation

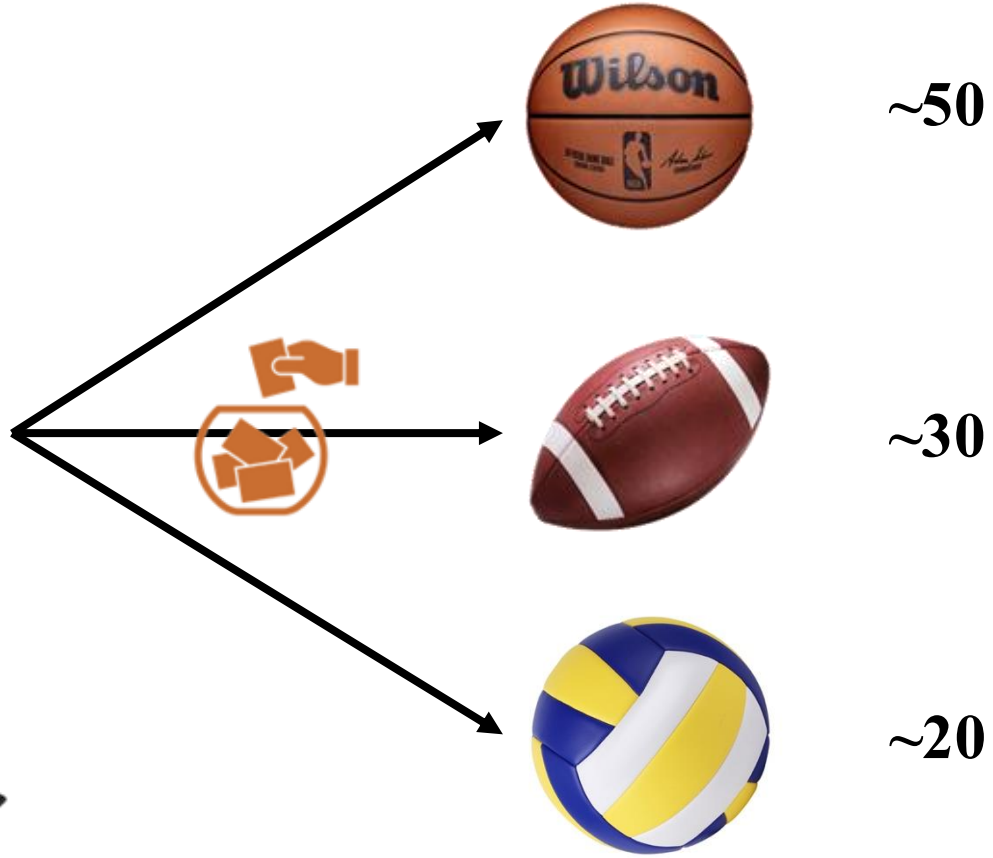
Discrete Generative Model – Ball Example

5 Basketball

2 Volleyball

3 Football

If you Randomly draw 100 times ……



Discrete Generative Model – Ball Example

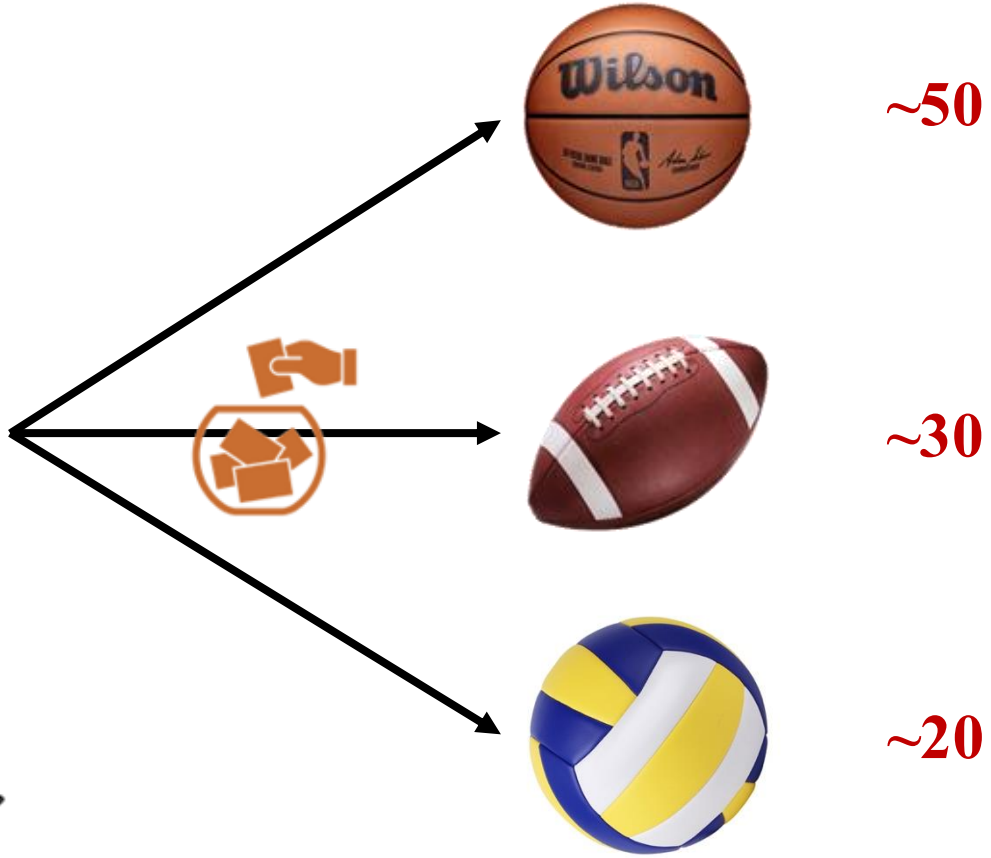
But how can you simulate this process on computer?

5 Basketball

2 Volleyball

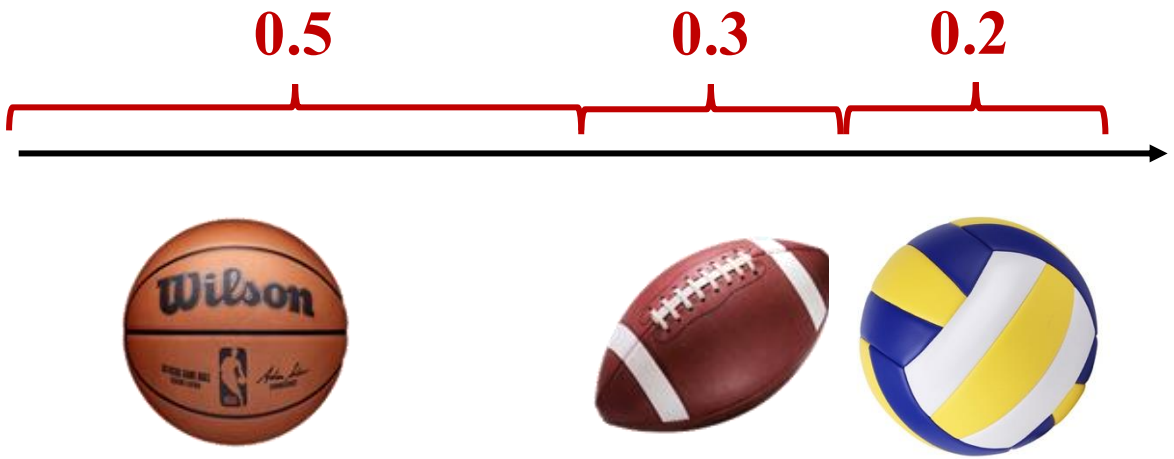
3 Football

If you Randomly draw 100 times ……

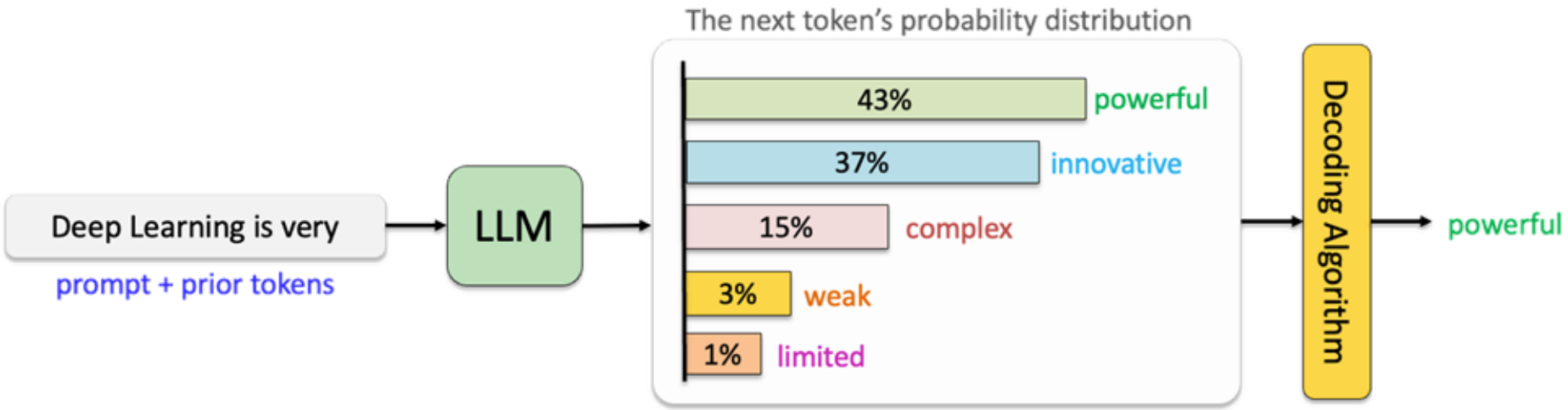


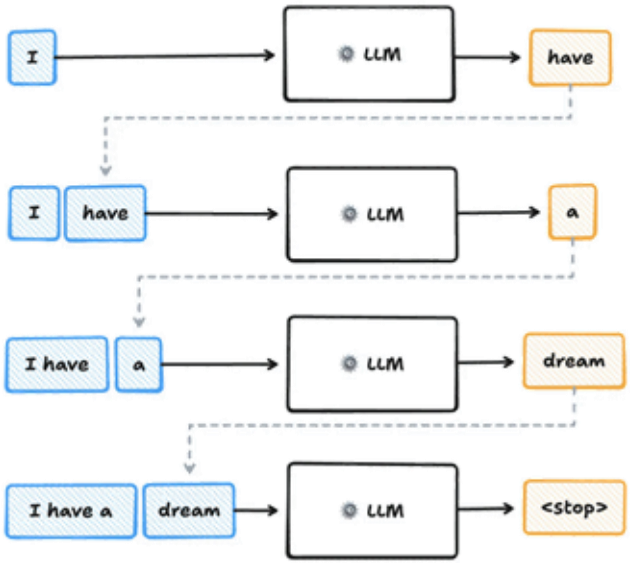
Discrete Generative Model – Ball Example

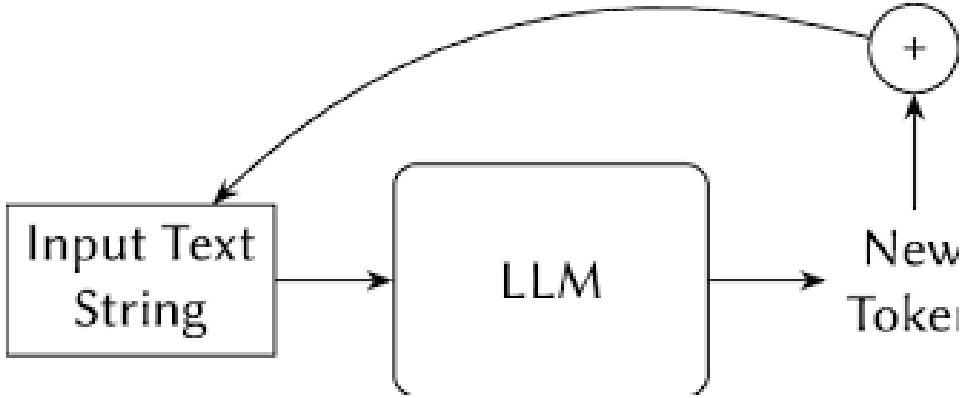
$\alpha \sim \boldsymbol{U}(0,1)$



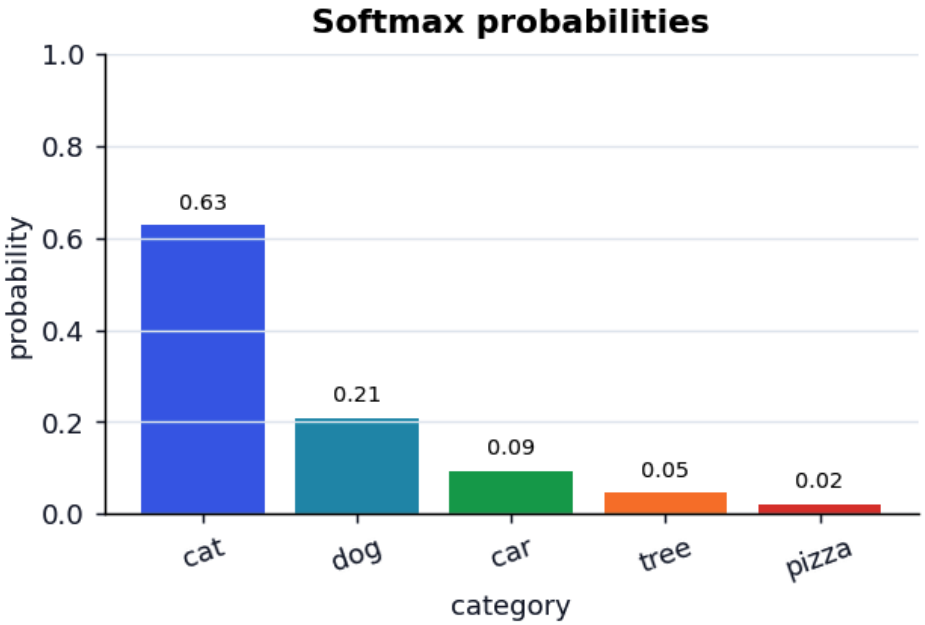
Discrete Generative Model – Language Example

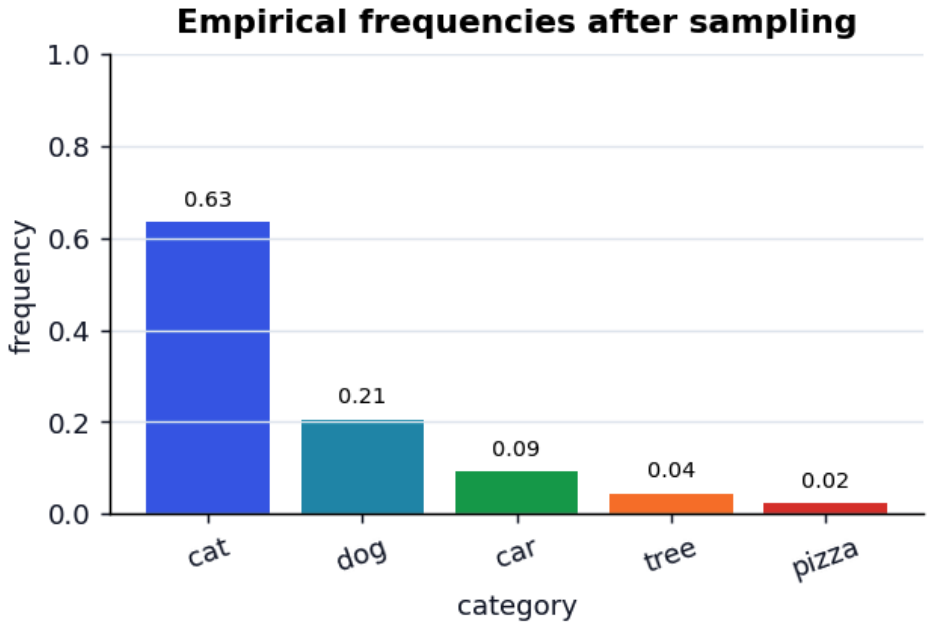






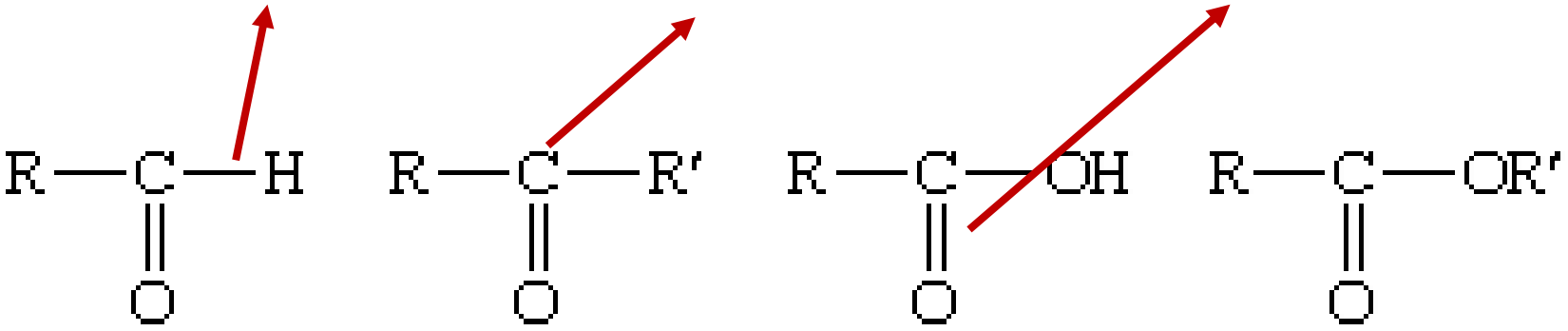
Discrete Generative Model – Language Example



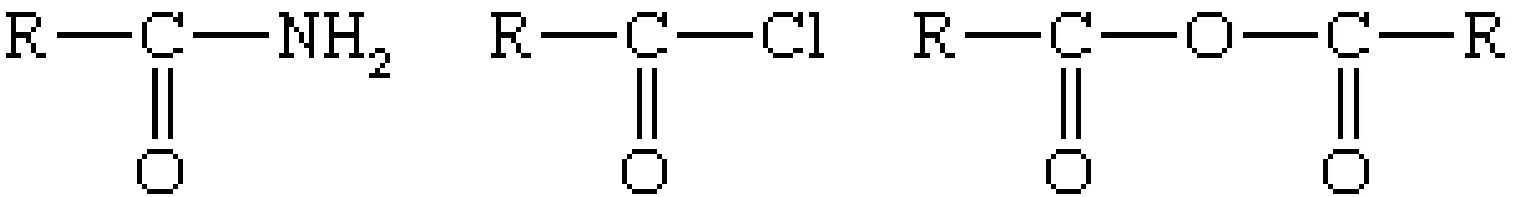


Discrete Generative Model – Graph Example

$$p_\theta(A_{ij} = 1 \mid z), \qquad p_\theta(\text{node}_i = \text{carbon} \mid z), \qquad p_\theta(\text{bond}_{ij} = \text{double} \mid z).$$

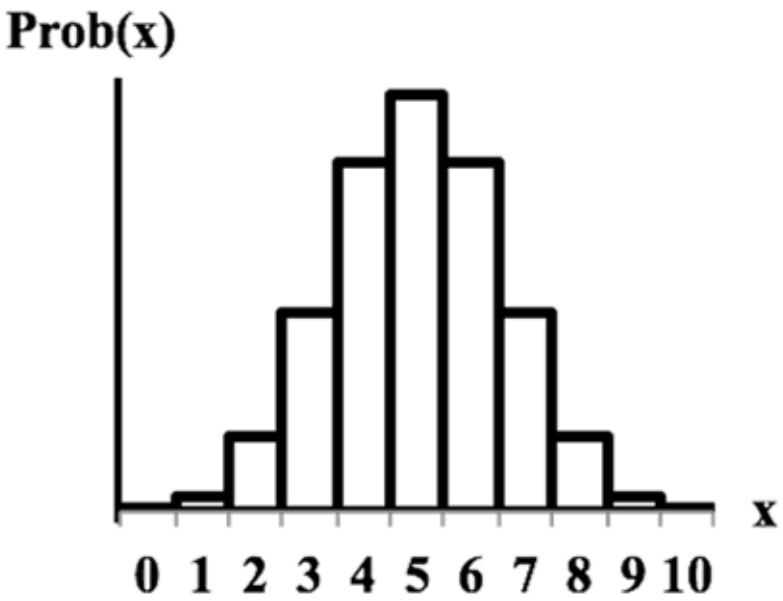


aldehyde ketone carboxylic acid carboxylic ester

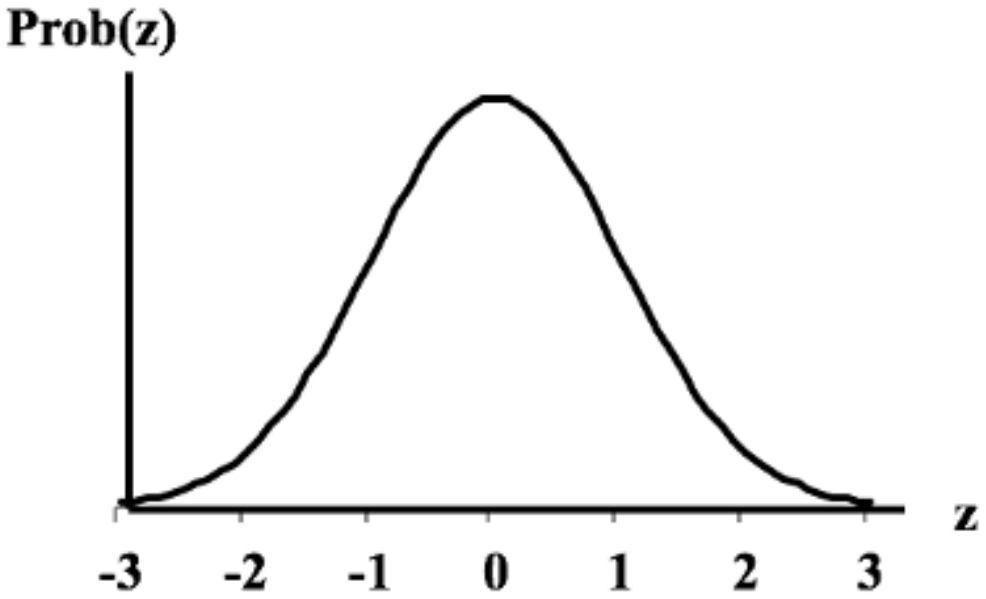


amide acyl chloride anhydride

Discrete vs Continuous Generative Model



Binomial Distribution
Discrete Data & Discrete
Probability Curve



Standard Normal Distribution
Continuous Data and Continuous
Probability Curve

Overview

- **Statistical Methods**
  - Gaussian Estimation (1D or 2D or higher)
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

- **Machine/Deep Learning Methods**
  - Variational AE (VAE)
  - Generative Adversarial Network (GAN)
  - Diffusion

Continuous version:
Point Cloud/Image/Video

Discrete version:
Graph/Language

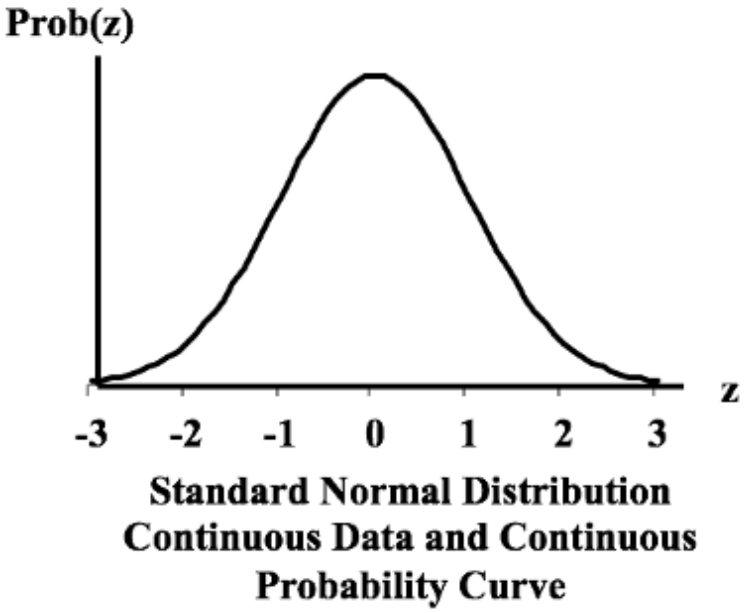
Three key questions

(1) How to model the distribution?

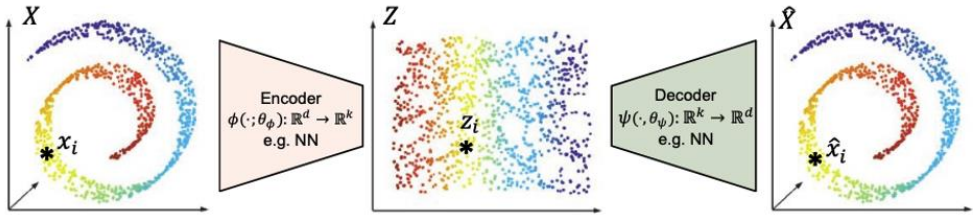
(2) How to train the model to learn the distribution?

(3) How to use the learned distribution to generate data?

Three key questions



VAE



Neural Decoder Mapping

Mean-squared error loss

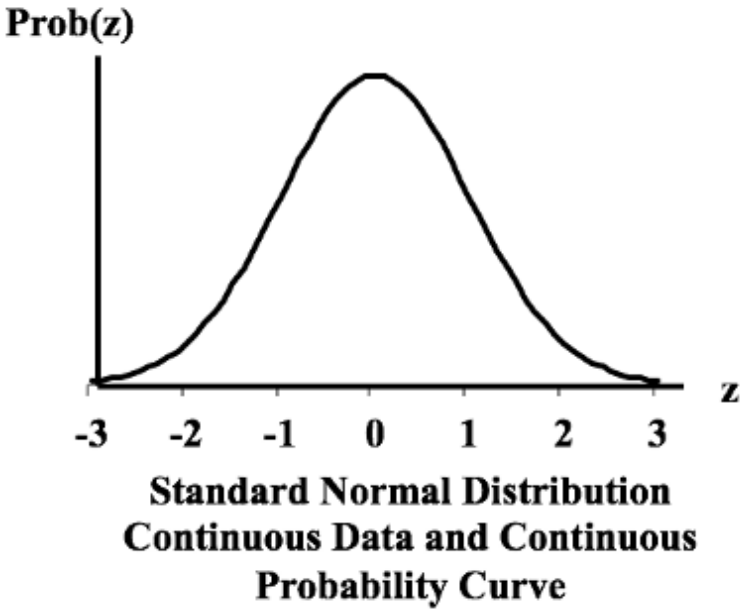
Sampling Gaussian and transform

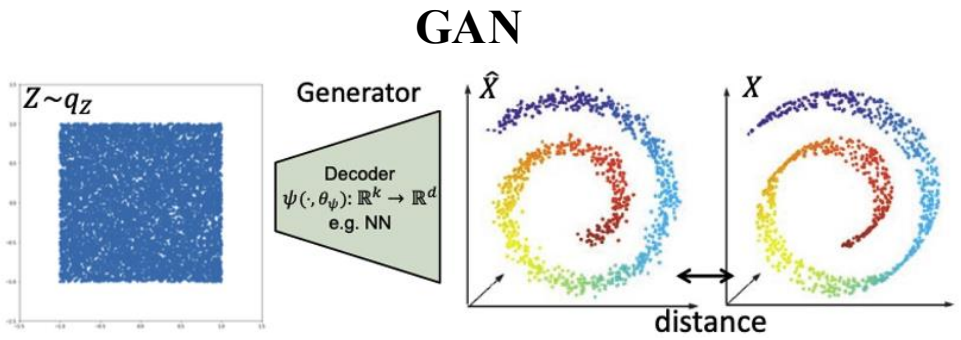
(1) How to model the distribution?

(2) How to train the model to learn the distribution?

(3) How to use the learned distribution to generate data?

Three key questions





Neural Decoder Mapping

Discriminator – Distribution Difference

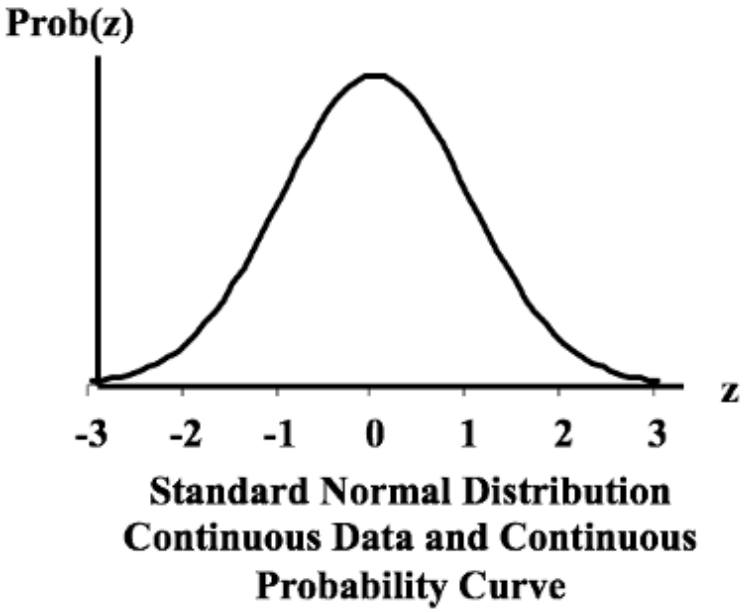
Sampling from Gaussian and transform

(1) How to model the distribution?

(2) How to train the model to learn the distribution?

(3) How to use the learned distribution to generate data?

Three key questions



Diffusion

Noise Decoding + Denoise

Mean-squared error loss

Autoregressively sampling and decoding

(1) How to model the distribution?

(2) How to train the model to learn the distribution?

(3) How to use the learned distribution to generate data?

Overview

- **Statistical Methods**
  - Gaussian Estimation (1D or 2D or higher)
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

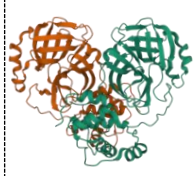
- **Machine/Deep Learning Methods**
  - **Graph** Auto-Encoder (VAE)
  - **Graph** Generative Adversarial Network (GAN)
  - **Graph** Diffusion (Diffusion)

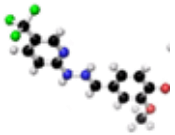
Continuous version:
Point Cloud/Image/Video

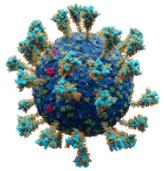
Discrete version:
Graph/Language

Graph Definition

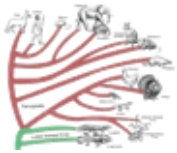
Scientific Graph


Protein


Small Molecule


Virus

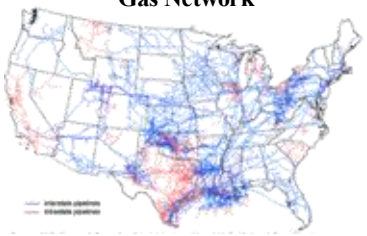

Brain Neural


Phylogenetic Tree

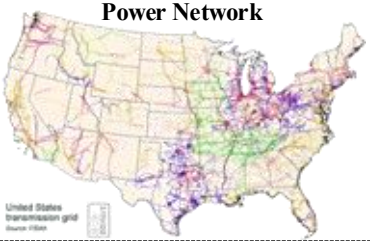

3D Grid

Infrastructure Graph

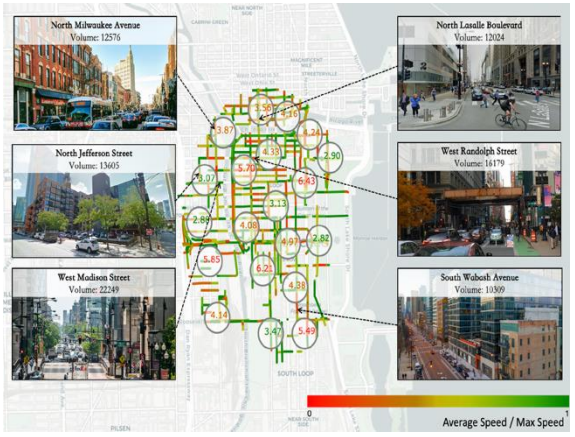
Gas Network



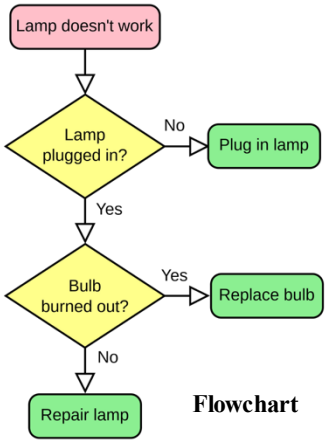
Power Network



Transportation Network

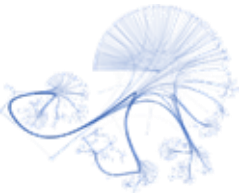


Decision Graph



Flowchart

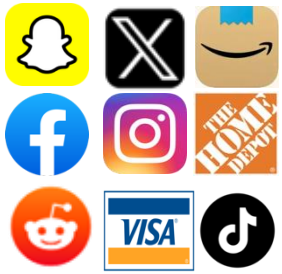
Social Interaction Graph


Citation Network

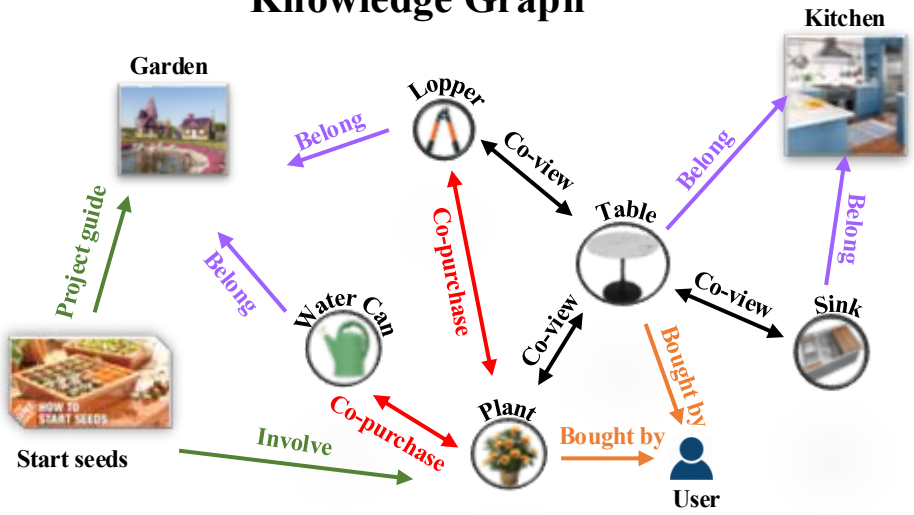

Transaction Network

User-Entity Interaction Graph


Virtual Village with AI Agents



Knowledge Graph

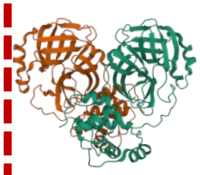


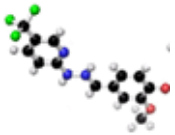
The Home Depot Product Knowledge Graph

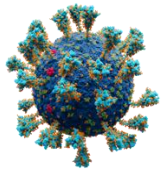
Graph Definition

Microscopic Graph

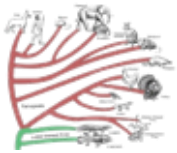
Scientific Graph


Protein


Small Molecule


Virus

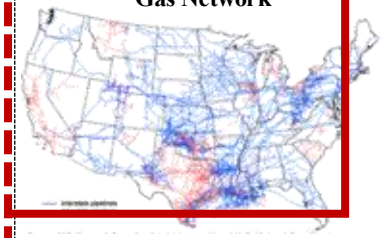

Brain Neural


Phylogenetic Tree

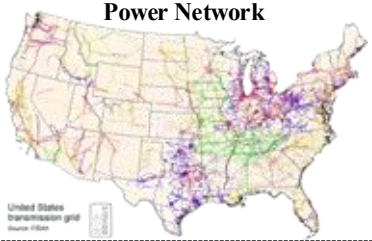

3D Grid

Infrastructure Graph

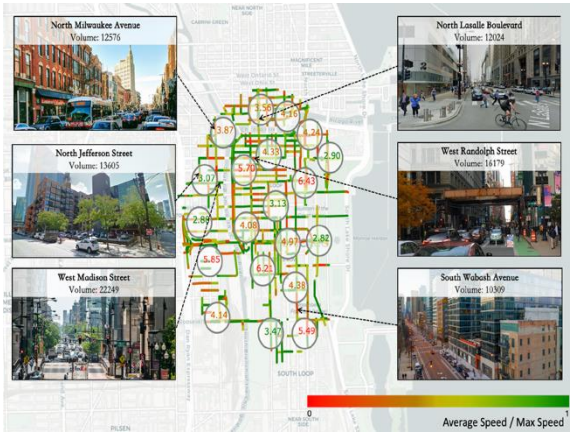
Gas Network



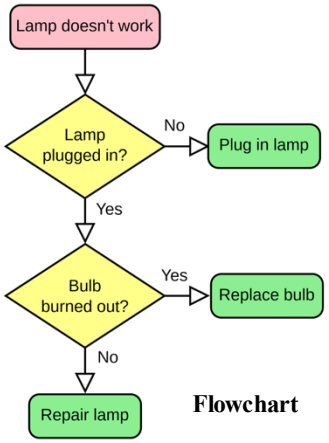
Power Network



Transportation Network

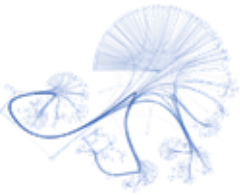


Decision Graph



Flowchart

Social Interaction Graph

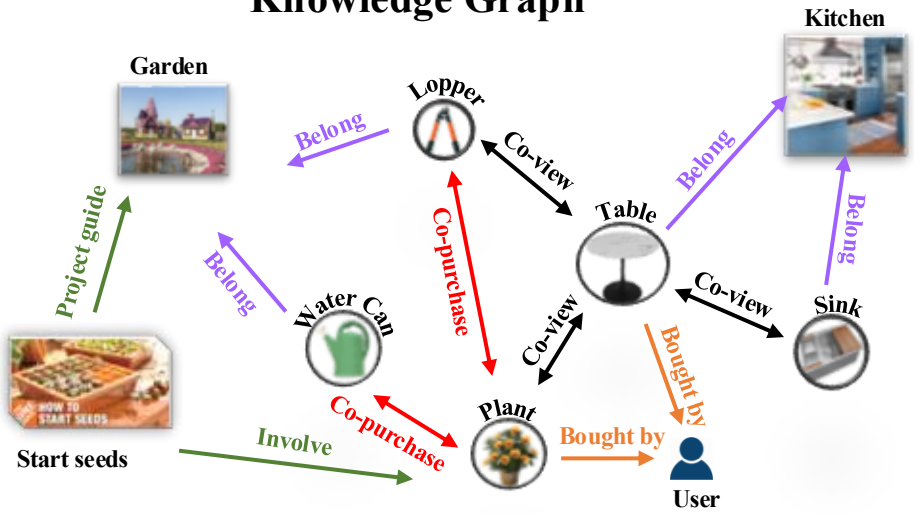

Citation Network


Transaction Network

User-Entity Interaction Graph


Virtual Village with AI Agents

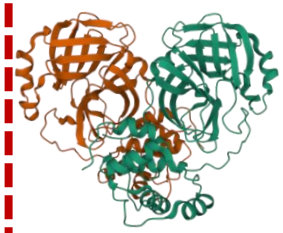
Knowledge Graph



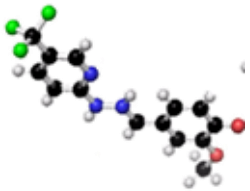
The Home Depot Product Knowledge Graph

Microscopic Graph Generation

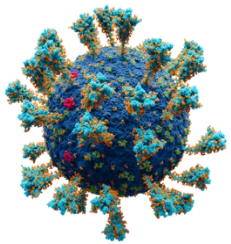
Scientific Graph



Protein

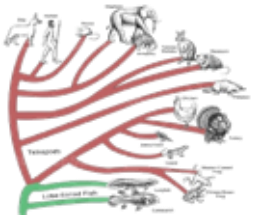


Small Molecule



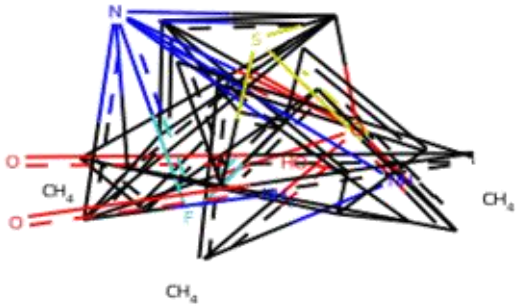
Virus

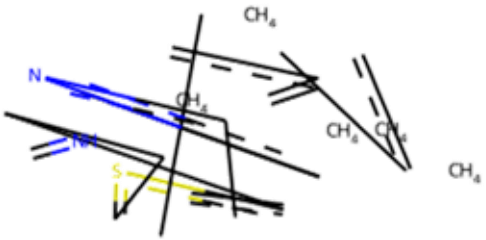
Brain Neural



Phylogenetic Tree

3D Grid



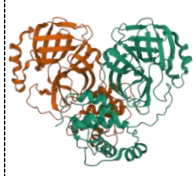


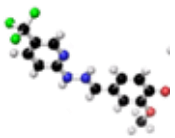
Diffusion

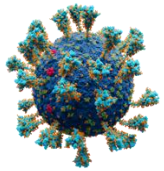
Graph Definition

Macroscopic Graph

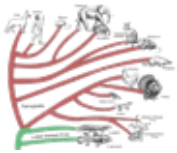
Scientific Graph


Protein


Small Molecule


Virus

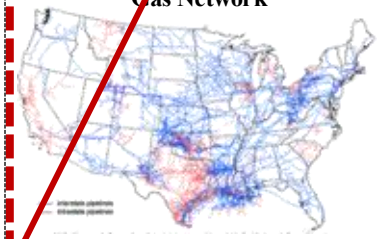

Brain Neural


Phylogenetic Tree

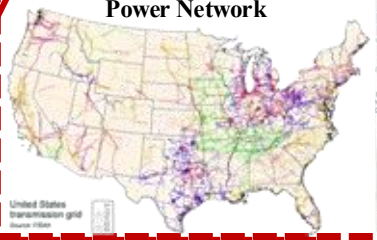

3D Grid

Infrastructure Graph

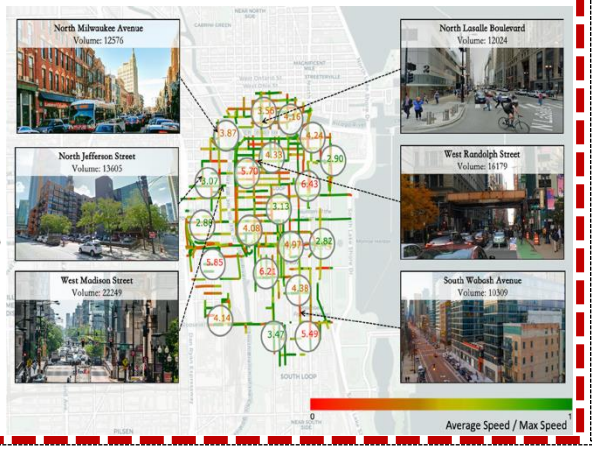
Gas Network



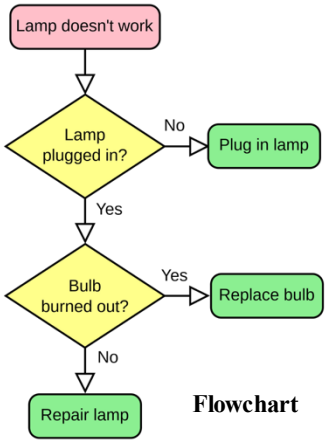
Power Network



Transportation Network

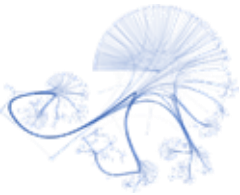


Decision Graph



Flowchart

Social Interaction Graph

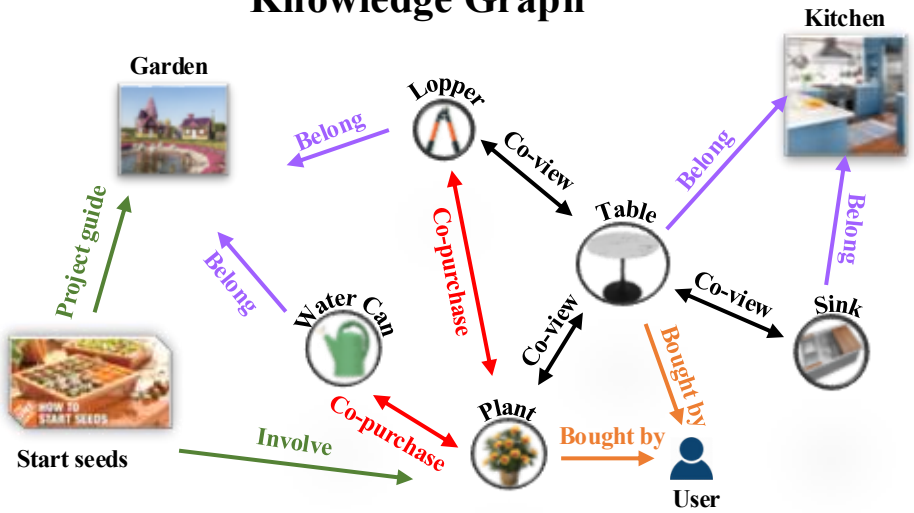

Citation Network


Transaction Network

User-Entity Interaction Graph


Virtual Village with AI Agents

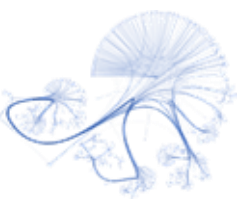
Knowledge Graph



The Home Depot Product Knowledge Graph

Macroscopic Graph Generation

Social Interaction Graph



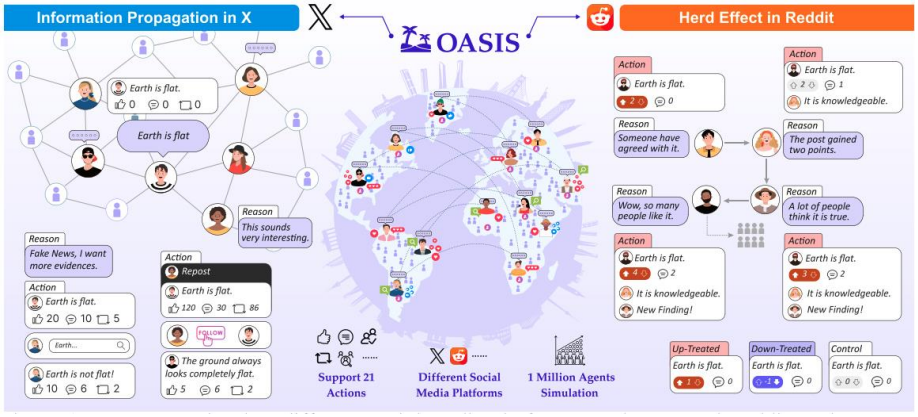
Citation Network

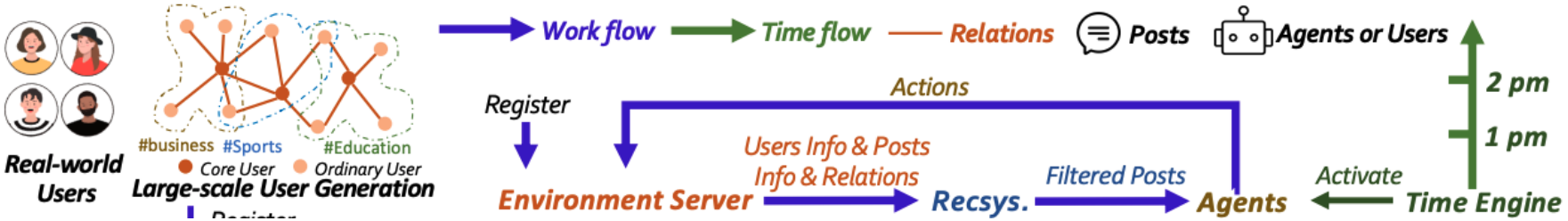
Transaction Network

User-Entity Interaction Graph

Virtual Village with AI Agents

Macroscopic Graph

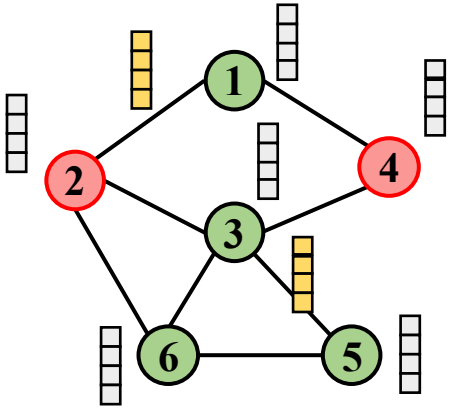




Overview

- Graph Definition
- Graph Auto-Encoder
- Graph Generative Adversarial Network
- Graph Diffusion
- Feature and Topology
- Permutation Invariant

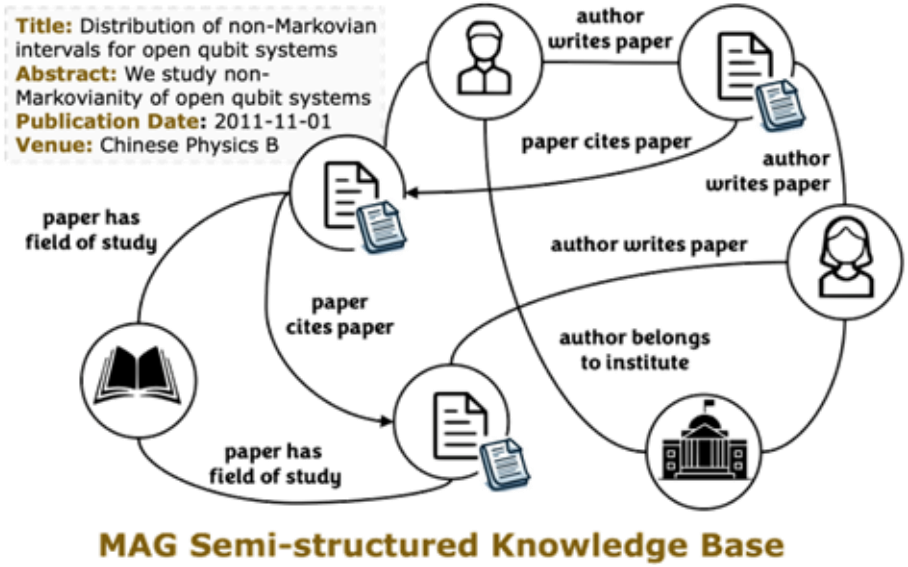
Graph Definition



Node Attributes

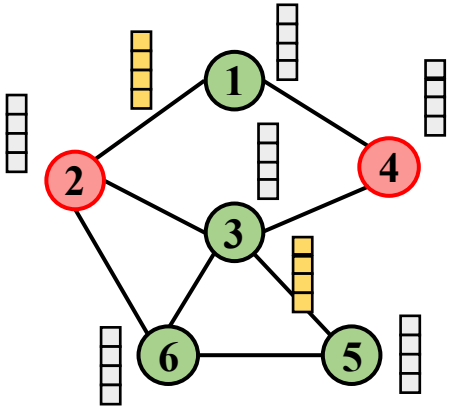
Edge Attributes (Existence)

Graph Attributes



Academic Graph

Graph Definition

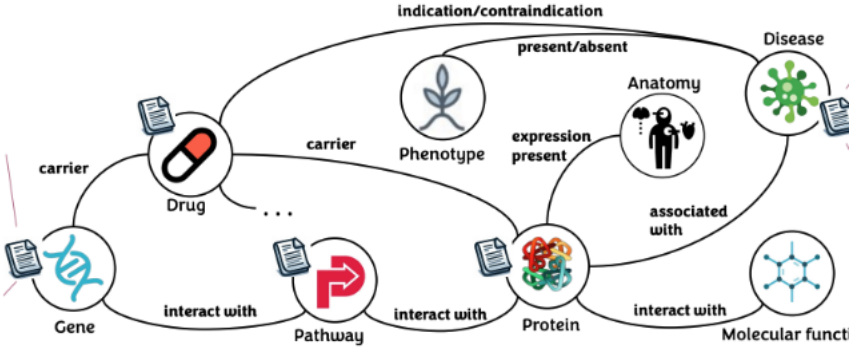


Node Attributes

Edge Attributes (Existence)

Graph Attributes

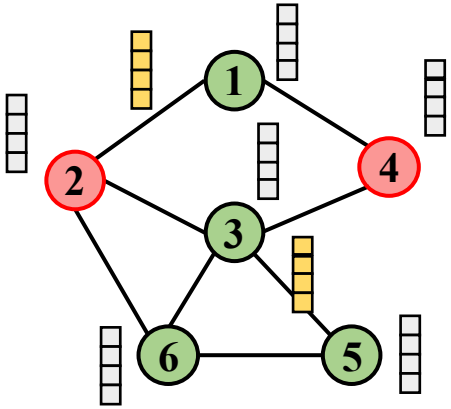
Name: GPANK1
Alias: DYRK1AP3, PAHX-AP, PAHXAP1
Description: This gene encodes a protein which is thought to play a role in immunity. Multiple alternatively spliced variants, encoding the same protein, have been identified.



Prime Semi-structured Knowledge Base

Biomedical Graph

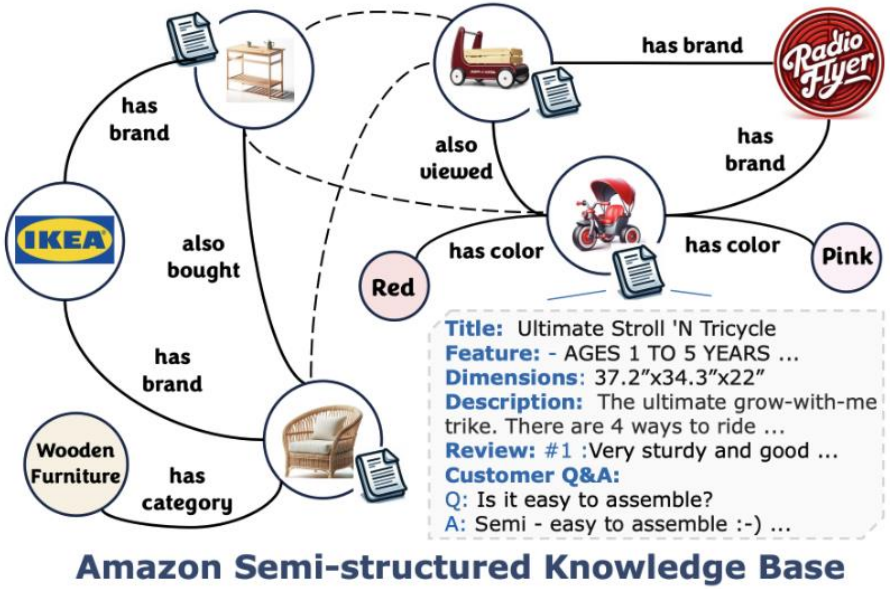
Graph Definition



Node Attributes

Edge Attributes (Existence)

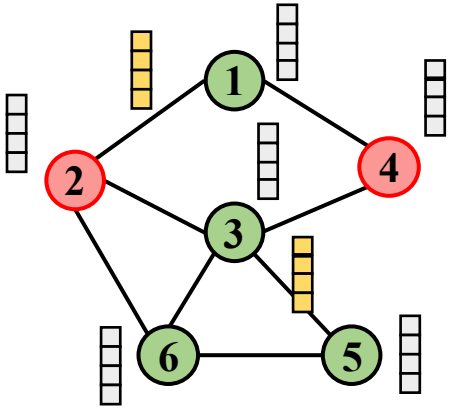
Graph Attributes



Amazon Semi-structured Knowledge Base

Amazon

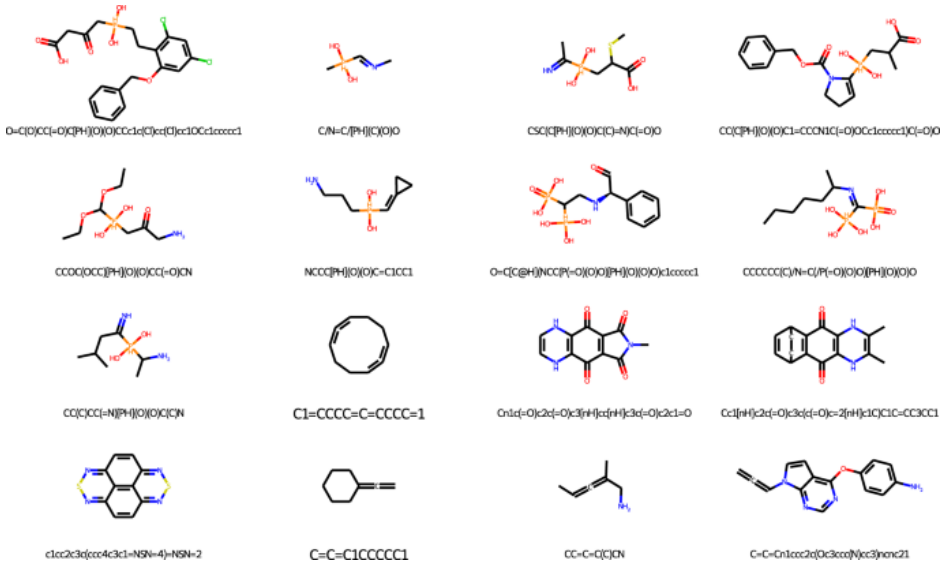
Graph Definition



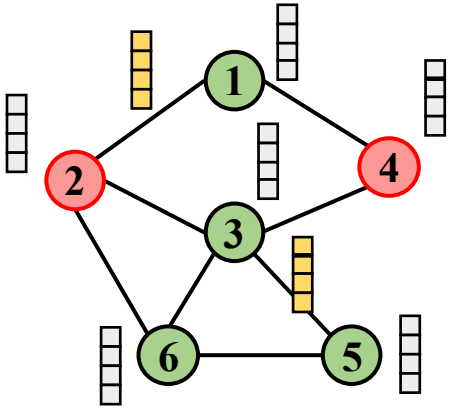
Node Attributes

Edge Attributes (Existence)

Graph Attributes



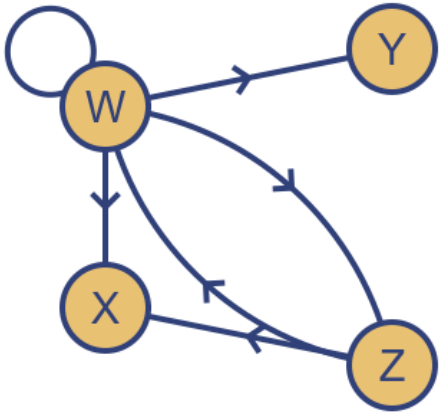
Graph Definition



Node Attributes $\boldsymbol{X} \in \mathbb{R}^{6 \times K^{\text{node}}}$

Edge Attributes (Existence) $\boldsymbol{E} \in \mathbb{R}^{8 \times K^{\text{Edge}}}$

Graph Attributes $\boldsymbol{Y} \in \mathbb{R}^{1 \times K^{\text{Graph}}}$



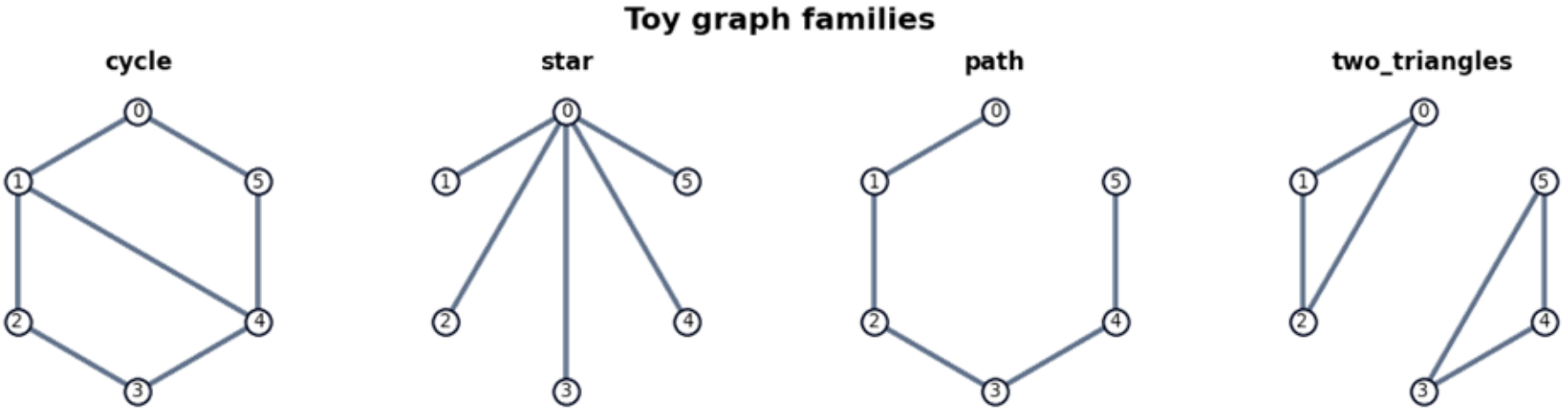
Directed graph with loop

	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

Overview

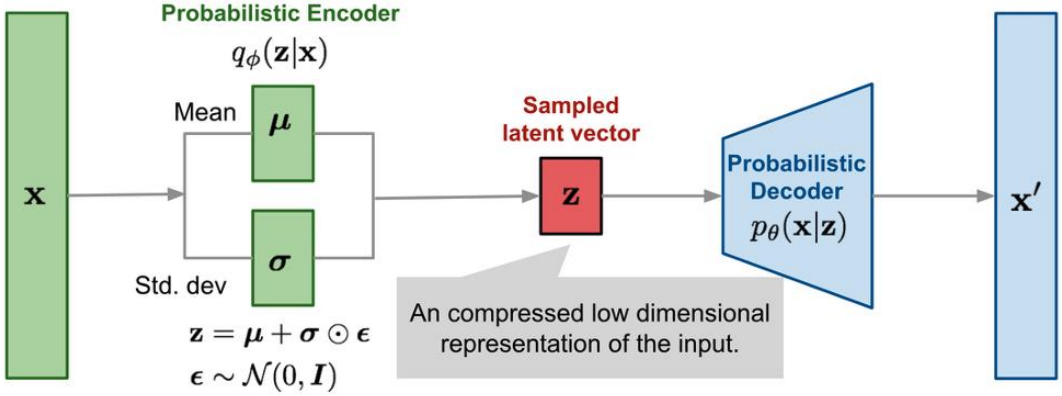
- Graph Definition
- Graph Auto-Encoder
- Graph Generative Adversarial Network
- Graph Diffusion
- Feature and Topology
- Permutation Invariant

Microscopic Graph Generation - GraphVAE

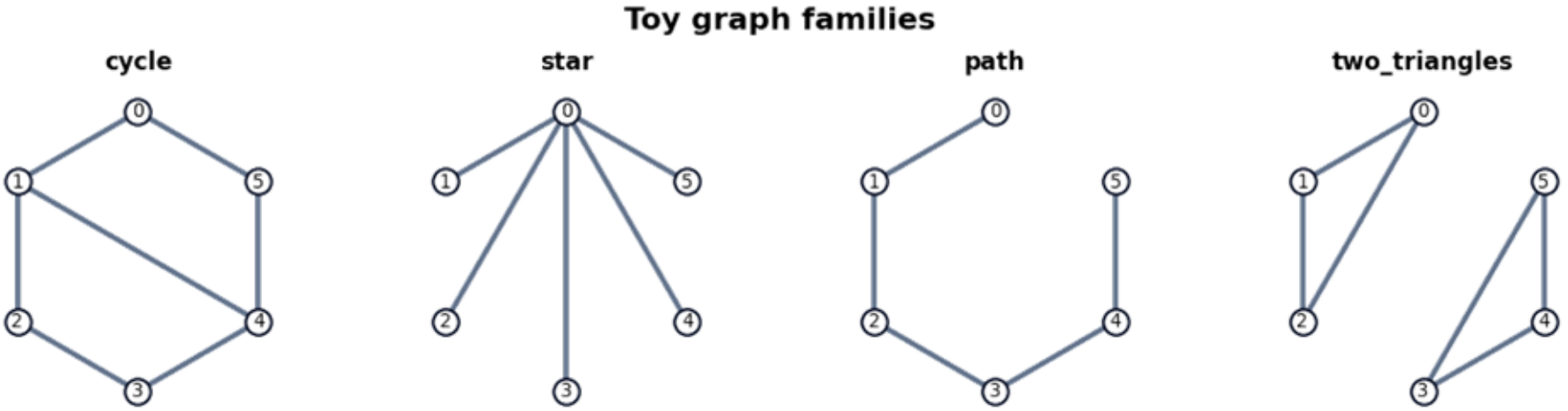


$$x \xrightarrow{\text{encoder}} q_\phi(z \mid x) \xrightarrow{\text{sample}} z \xrightarrow{\text{decoder}} p_\theta(x \mid z).$$

$$G \xrightarrow{\textbf{Encoder}} q_\phi(z|G) \xrightarrow{\textbf{Sample}} z \xrightarrow{\textbf{Decoder}} p_\theta(G|z)$$



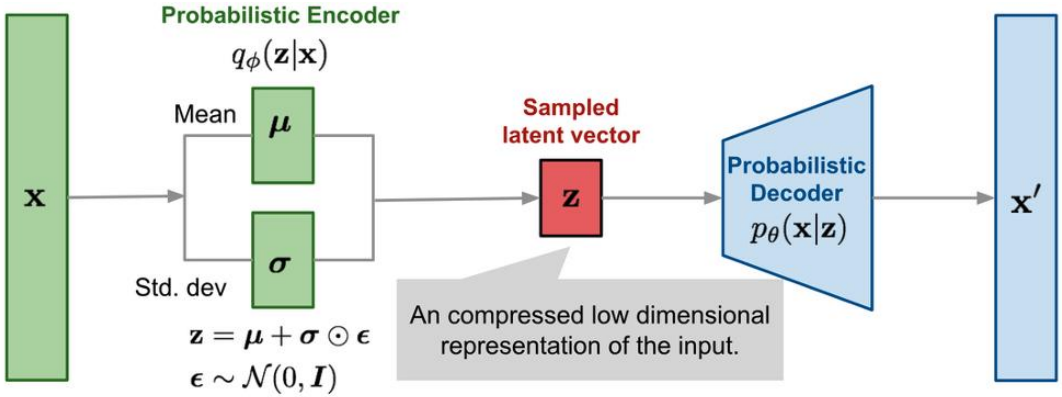
Microscopic Graph Generation - GraphVAE



$$\mathcal{L}_{\text{VAE}} = \underbrace{\|x - \hat{x}\|^2}_{\text{Gaussian decoder} \Rightarrow \text{MSE}} + \underbrace{D_{\text{KL}}(q_\phi(z|x) \,\|\, \mathcal{N}(0, I))}_{\text{KL regulariser}}$$

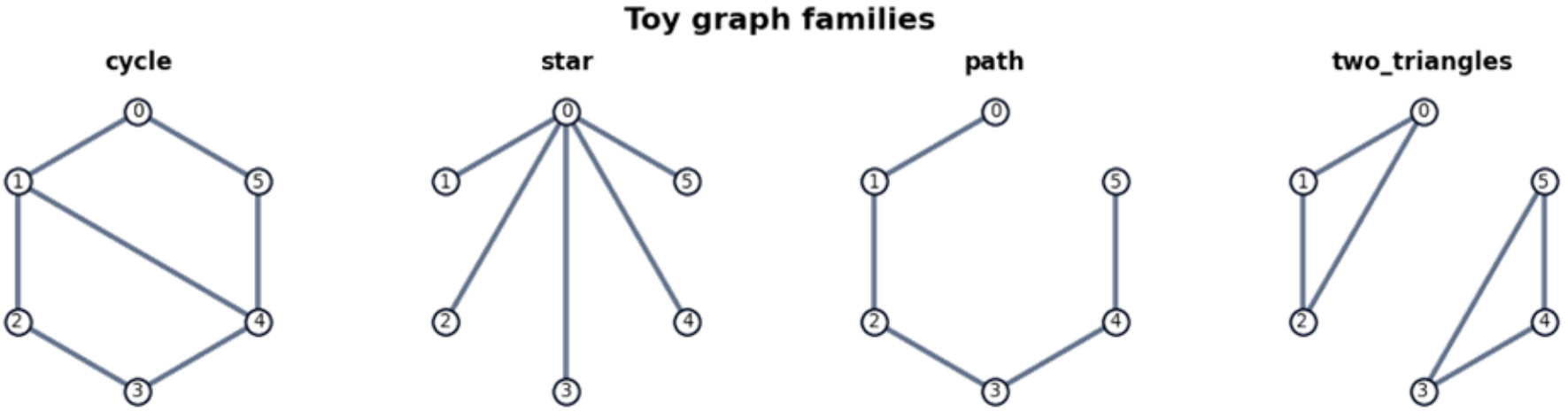
$$\mathcal{L}_{\text{G-VAE}} = -p_\theta(G^*|z) + D_{\text{KL}}(q_\phi(z|G^*)||\mathcal{N}(0,I))$$

$$\mathcal{L}_{\text{A-VAE}} = -p_\theta(A^*|z) + D_{\text{KL}}(q_\phi(z|A^*)||\mathcal{N}(0,I))$$



Maximum Likelihood

Microscopic Graph Generation - GraphVAE

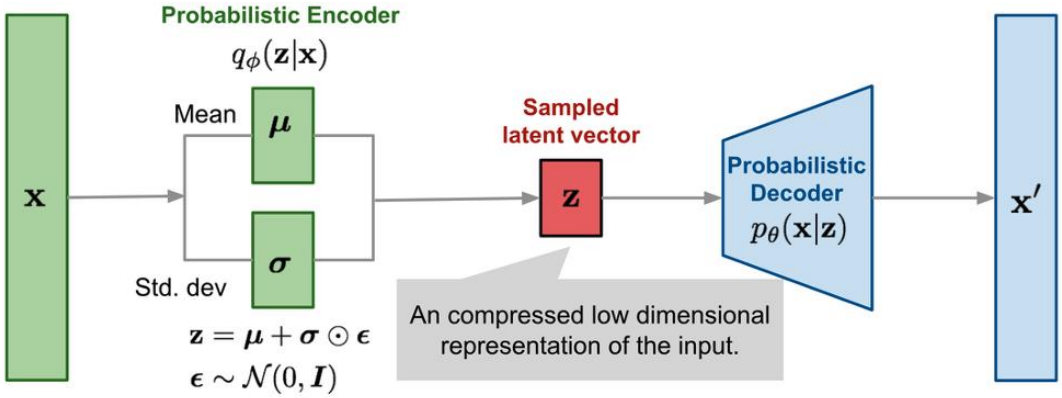


$$\mathcal{L}_{\text{G-VAE}} = -p_\theta(A^*|z) + D_{\text{KL}}(q_\phi(z|A^*)||\mathcal{N}(0,I))$$

$$p_\theta(A \mid z) = \prod_{i<j} \text{Bernoulli}(A_{ij}; \pi_{ij}(z)).$$

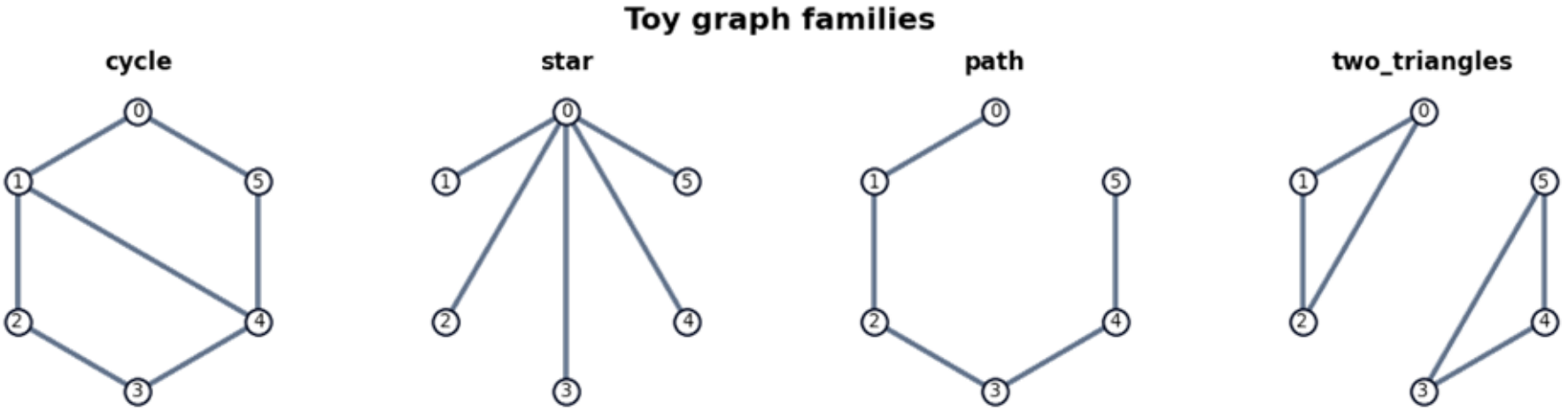
Maximum Likelihood

Bernoulli Distribution



$$p_\theta(A|z) = \prod_{i<j} \pi_{ij}^{A_{ij}} (1-\pi_{ij})^{1-A_{ij}}$$

Microscopic Graph Generation - GraphVAE



$$\mathcal{L}_{\text{G-VAE}} = -p_\theta(A^*|z) + D_{\text{KL}}(q_\phi(z|A^*)||\mathcal{N}(0,I))$$

Maximum Likelihood

$$p_\theta(A|z) = \prod_{i<j} \pi_{ij}^{A_{ij}} (1-\pi_{ij})^{1-A_{ij}}$$

$$\mathcal{L}_{\text{KL}} = -\frac{1}{2}\sum_{j=1}^{d}\left(1+\log\sigma_j^2-\mu_j^2-\sigma_j^2\right)$$

$$\mathcal{L}_{\text{edge}} = -\sum_{i<j}\left[A_{ij}\log\pi_{ij} + (1-A_{ij})\log(1-\pi_{ij})\right].$$

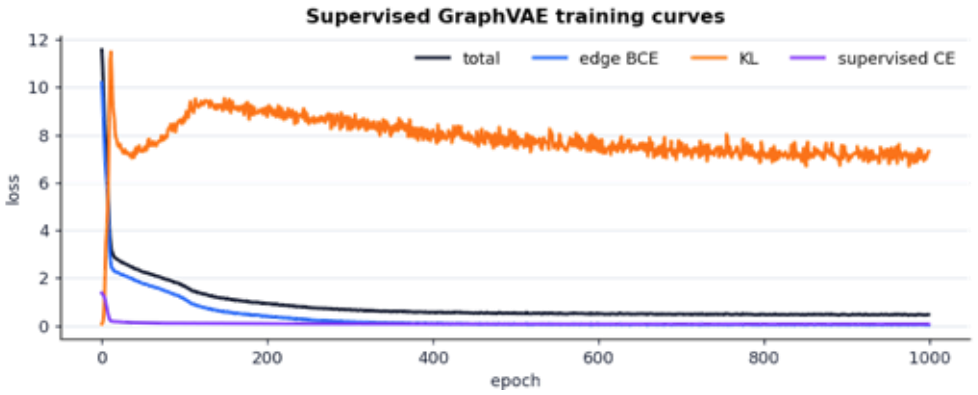
Binary Cross Entropy Loss

Microscopic Graph Generation - GraphVAE

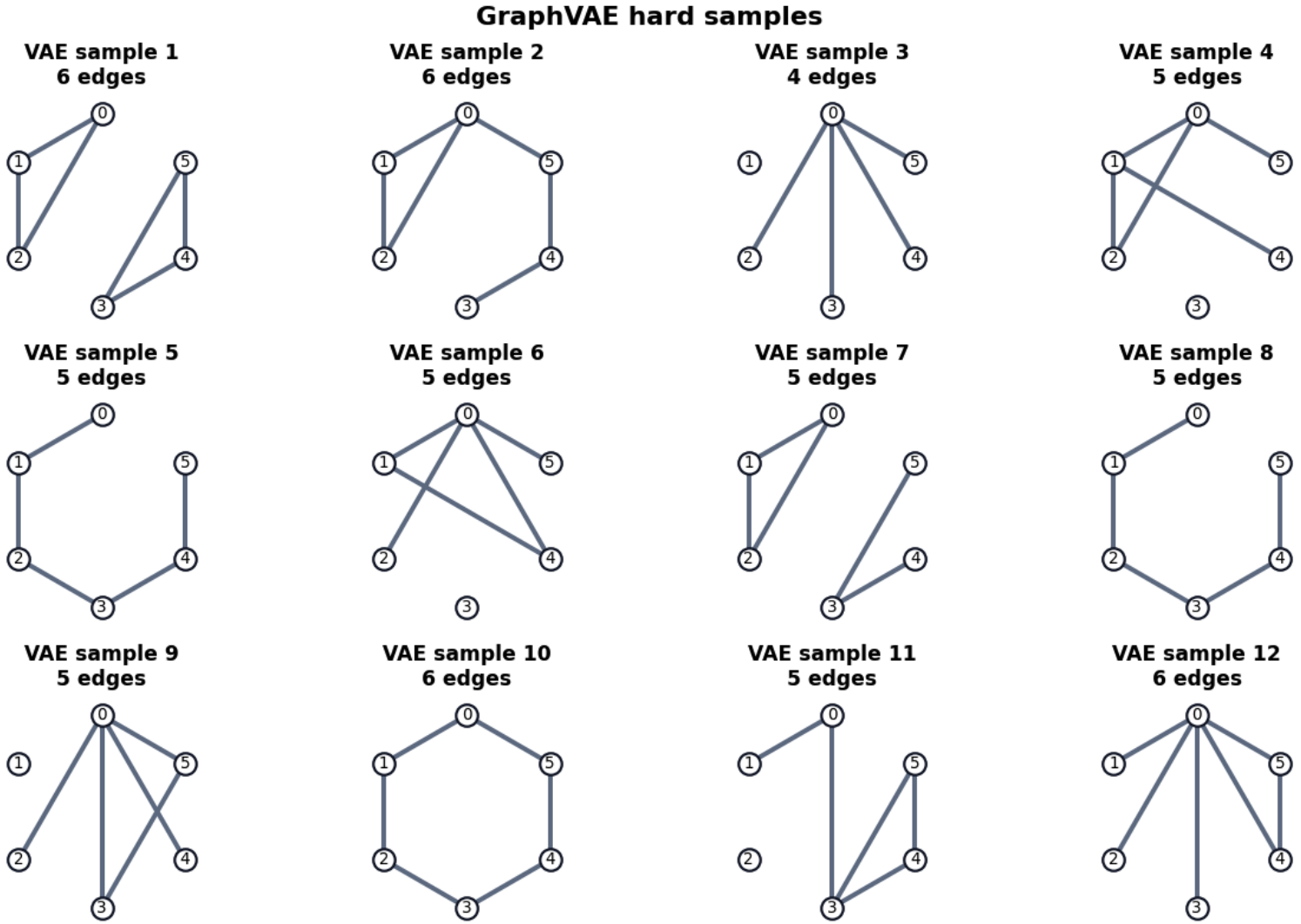
```python
class GraphVAE(nn.Module):
    def __init__(self, n_edges=N_EDGES, latent_dim=8, hidden=64, n_classes=4):
        super().__init__()
        self.encoder = nn.Sequential(
            nn.Linear(n_edges, hidden), nn.ReLU(),
            nn.Linear(hidden, hidden), nn.ReLU(),
        )
        self.mu = nn.Linear(hidden, latent_dim)
        self.logvar = nn.Linear(hidden, latent_dim)
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, hidden), nn.ReLU(),
            nn.Linear(hidden, hidden), nn.ReLU(),
            nn.Linear(hidden, n_edges),
        )
        self.class_centers = nn.Parameter(0.1 * torch.randn(n_classes, latent_dim))
```

```python
def vae_loss(logits, x, mu, logvar, class_logits=None, labels=None, beta=0.05, gamma=1.0):
    recon = F.binary_cross_entropy_with_logits(logits, x, reduction="sum") / x.size(0)
    kl = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp()) / x.size(0)
    supervised = torch.zeros((), device=x.device)
    if class_logits is not None and labels is not None:
        supervised = F.cross_entropy(class_logits, labels)
    total = recon + beta * kl + gamma * supervised
    return total, recon.detach(), kl.detach(), supervised.detach()
```

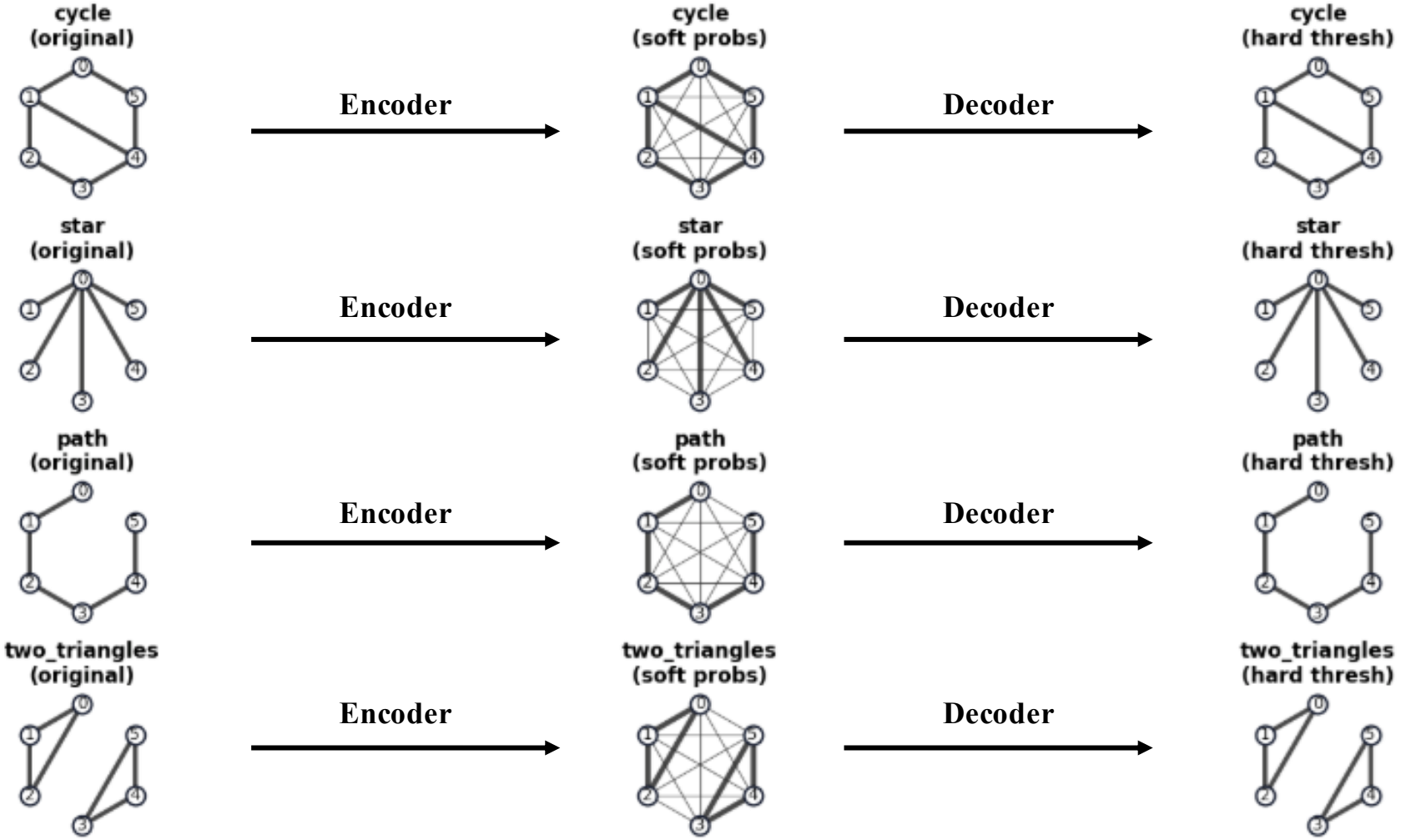
```python
    def encode(self, x):
        h = self.encoder(x)
        return self.mu(h), self.logvar(h)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode_logits(self, z):
        return self.decoder(z)

    def latent_class_logits(self, mu):
        # Higher score means closer to a learned class prototype in latent space.
        return -torch.cdist(mu, self.class_centers).pow(2)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        logits = self.decode_logits(z)
        return logits, mu, logvar
```



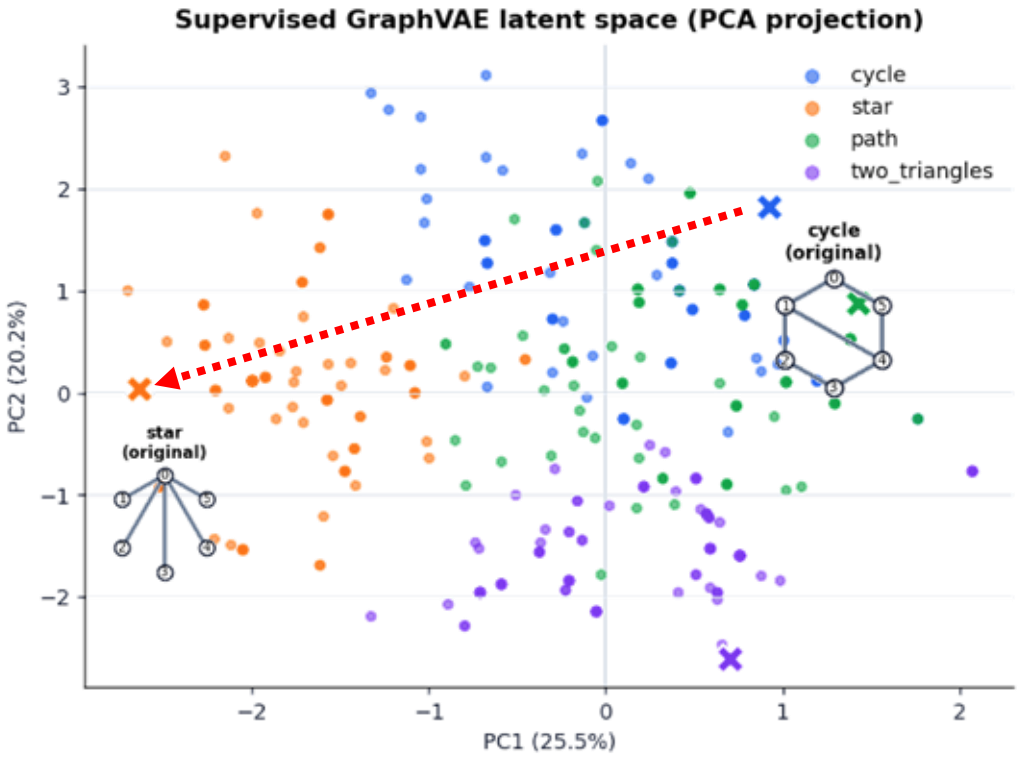
Microscopic Graph Generation - GraphVAE

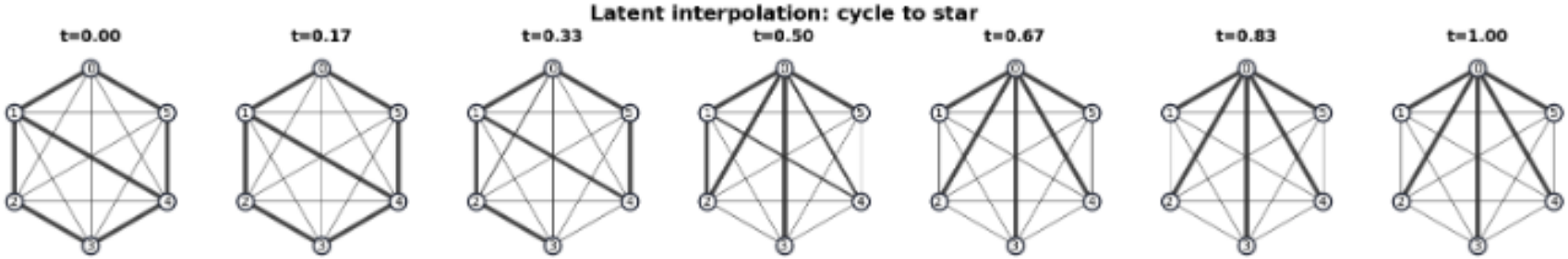


Microscopic Graph Generation - GraphVAE

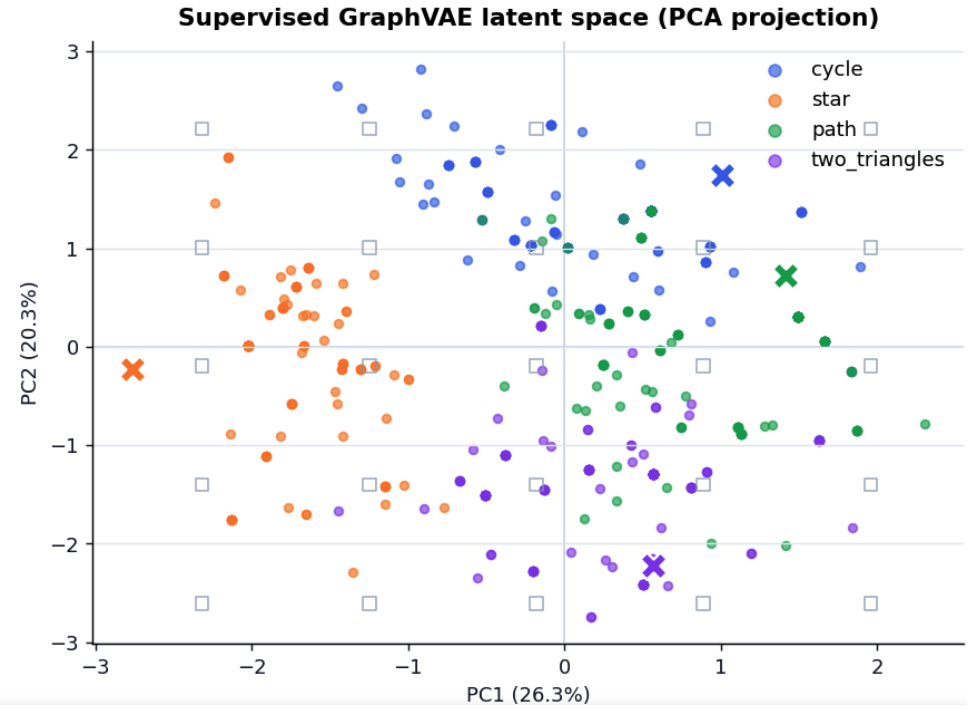


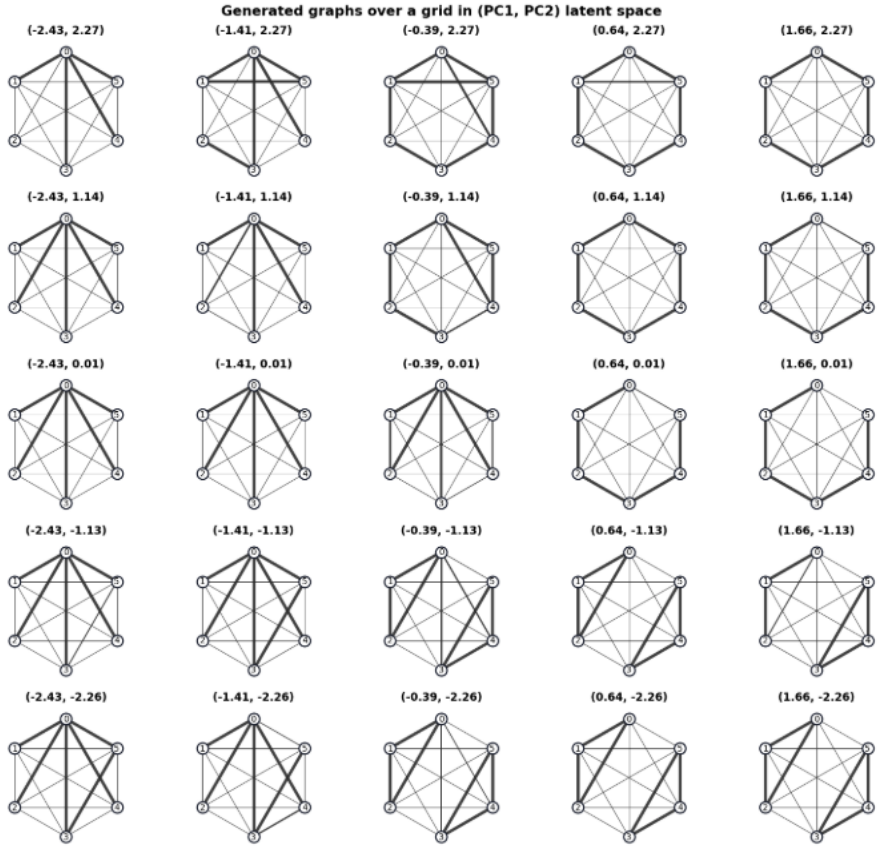
Microscopic Graph Generation - GraphVAE





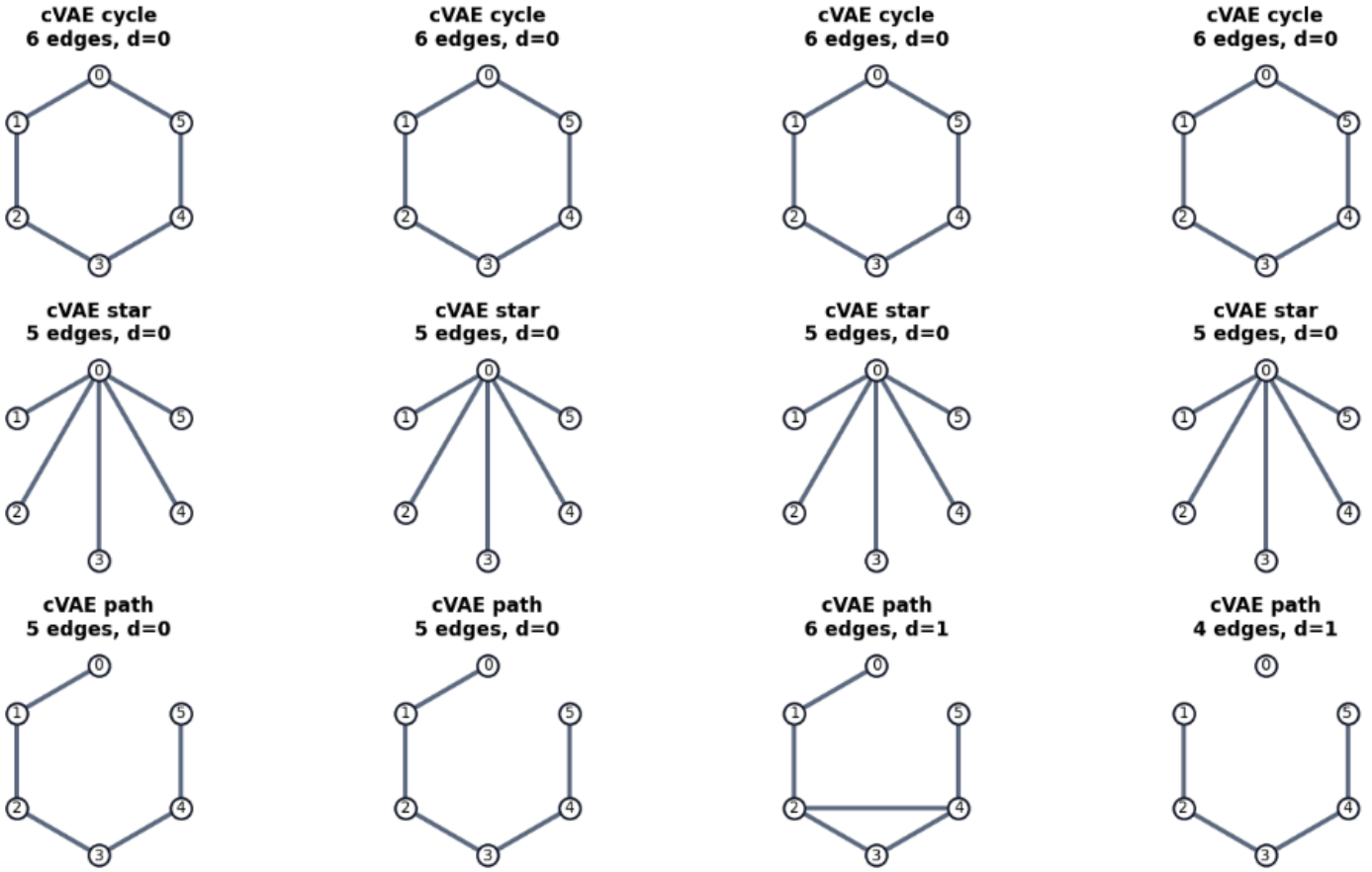
Microscopic Graph Generation - GraphVAE





Microscopic Graph Generation – Conditional GraphVAE

$$q_\phi(z \mid A, y), \qquad p_\theta(A \mid z, y).$$



Microscopic Graph Generation – Conditional GraphVAE

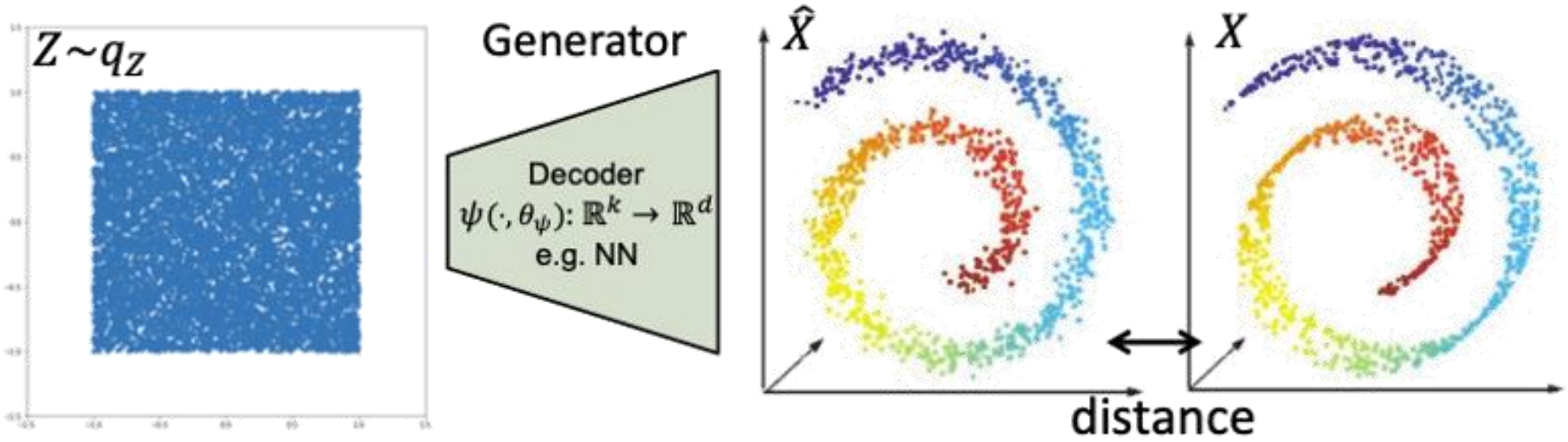
$$q_\phi(z \mid A, y), \qquad p_\theta(A \mid z, y).$$

```python
class ConditionalGraphVAE(nn.Module):
    def __init__(self, n_edges=N_EDGES, latent_dim=8, hidden=64, n_classes=4):
        super().__init__()
        self.n_classes = n_classes
        self.encoder = nn.Sequential(
            nn.Linear(n_edges + n_classes, hidden), nn.ReLU(),
            nn.Linear(hidden, hidden), nn.ReLU(),
        )
        self.mu = nn.Linear(hidden, latent_dim)
        self.logvar = nn.Linear(hidden, latent_dim)
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim + n_classes, hidden), nn.ReLU(),
            nn.Linear(hidden, hidden), nn.ReLU(),
            nn.Linear(hidden, n_edges),
        )

    def encode(self, x, labels):
        y = one_hot(labels, self.n_classes).to(x.device)
        h = self.encoder(torch.cat([x, y], dim=1))
        return self.mu(h), self.logvar(h)
```

Overview

- Graph Definition
- Graph Auto-Encoder
- Graph Generative Adversarial Network
- Graph Diffusion
- Feature and Topology
- Permutation Invariant

Graph GAN



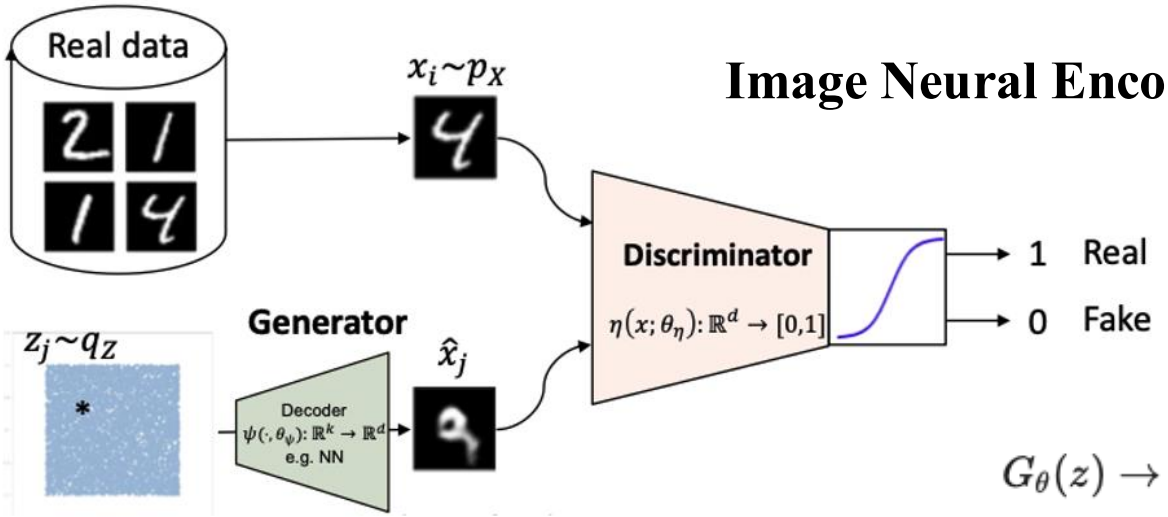
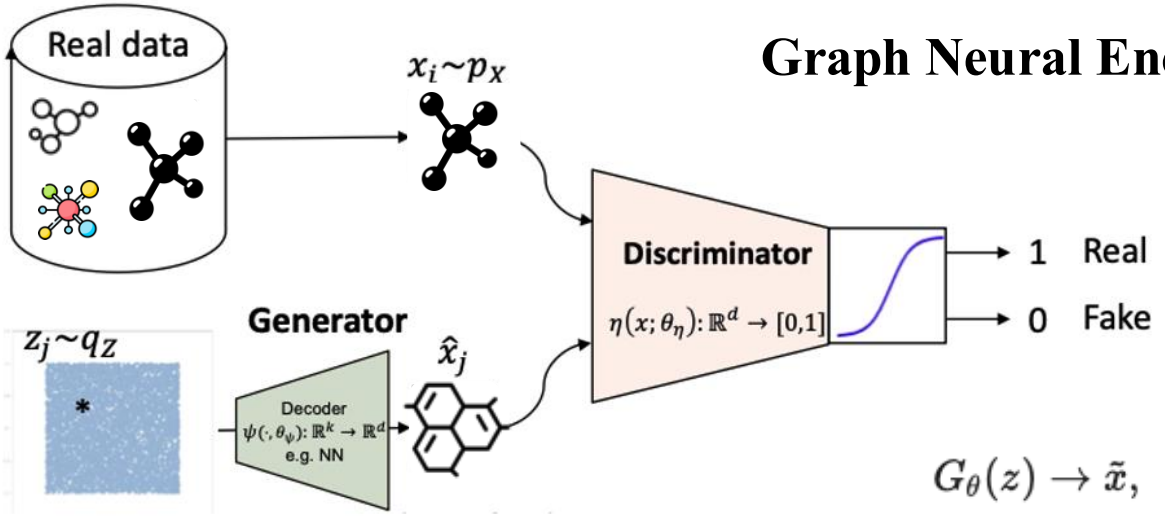


Image Neural Encoder

$G_\theta(z) \to \tilde{x}, \qquad D_\phi(x) \to \text{real/fake score.}$

Graph GAN

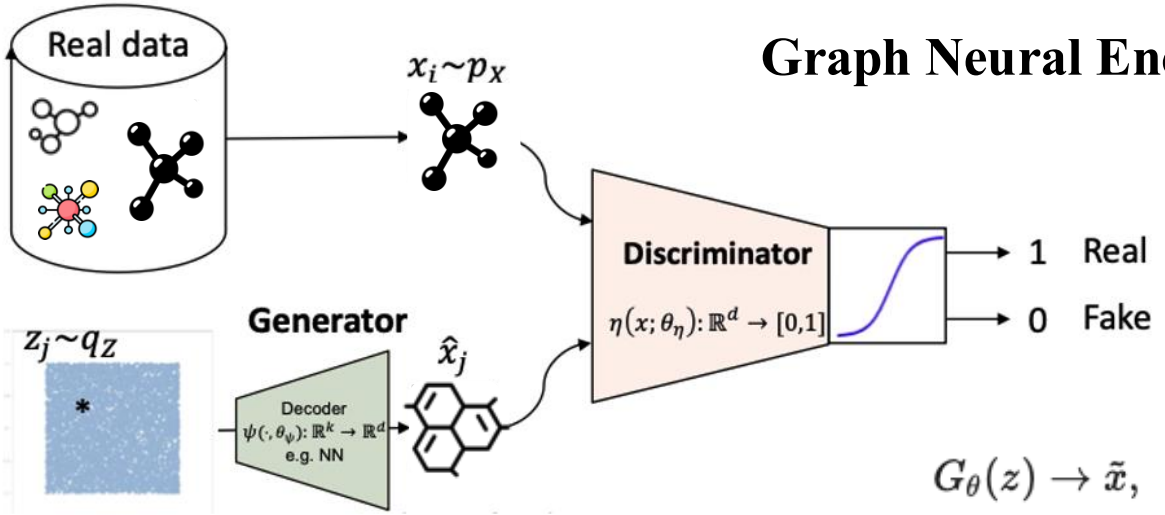


$$G_\theta(z) \to \tilde{x}, \qquad D_\phi(x) \to \text{real/fake score.}$$

$$x \in \{0,1\}^{N_{\text{edges}}}, \qquad x_k = A_{ij} \text{ for one upper-triangular pair } (i,j).$$

$$\tilde{x}_k \sim \text{Bernoulli}(\pi_k(z)) \quad \text{or} \quad \tilde{x}_k = \mathbf{1}[\pi_k(z) \geq 0.5].$$

Graph GAN



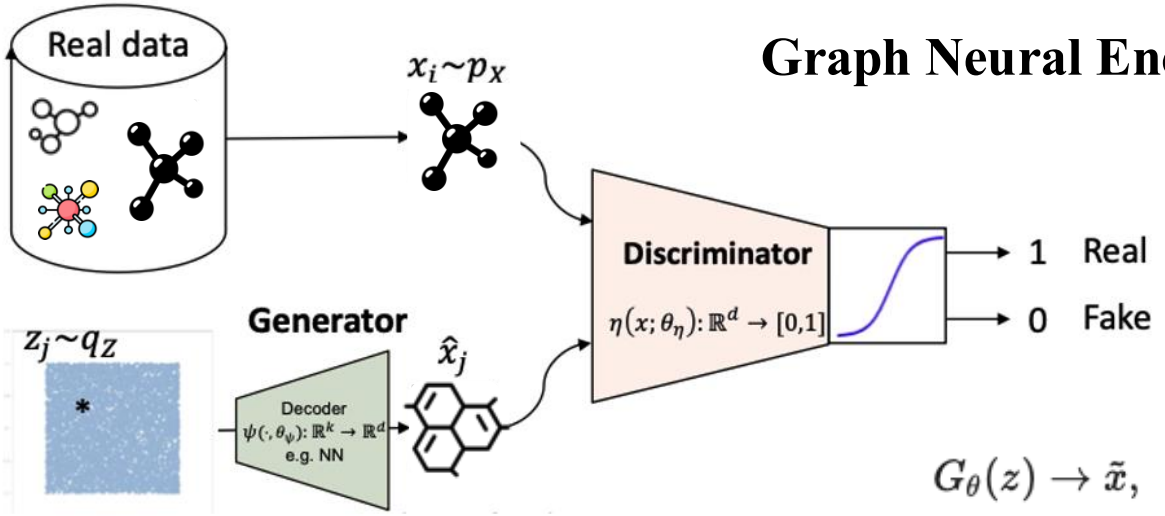
Graph Neural Encoder

$G_\theta(z) \to \tilde{x}, \qquad D_\phi(x) \to \text{real/fake score.}$

```
class GraphGenerator(nn.Module):
    def __init__(self, latent_dim=16, n_edges=N_EDGES, hidden=128):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(latent_dim, hidden), nn.LeakyReLU(0.2),
            nn.Linear(hidden, hidden), nn.LeakyReLU(0.2),
            nn.Linear(hidden, hidden // 2), nn.LeakyReLU(0.2),
            nn.Linear(hidden // 2, n_edges),
        )

    def forward(self, z):
        return self.net(z)
```

```
class GraphDiscriminator(nn.Module):
    def __init__(self, n_edges=N_EDGES, hidden=64):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(n_edges, hidden), nn.LeakyReLU(0.2),
            nn.Dropout(0.25),
            nn.Linear(hidden, hidden), nn.LeakyReLU(0.2),
            nn.Dropout(0.25),
            nn.Linear(hidden, 1),
        )
```

Graph GAN



$$G_\theta(z) \to \tilde{x}, \qquad D_\phi(x) \to \text{real/fake score.}$$

$$\mathcal{L}_D = \frac{1}{2}\left[\mathrm{BCE}(D(x_{\text{real}}), 1) + \mathrm{BCE}(D(x_{\text{fake}}^{\text{detach}}), 0)\right]$$

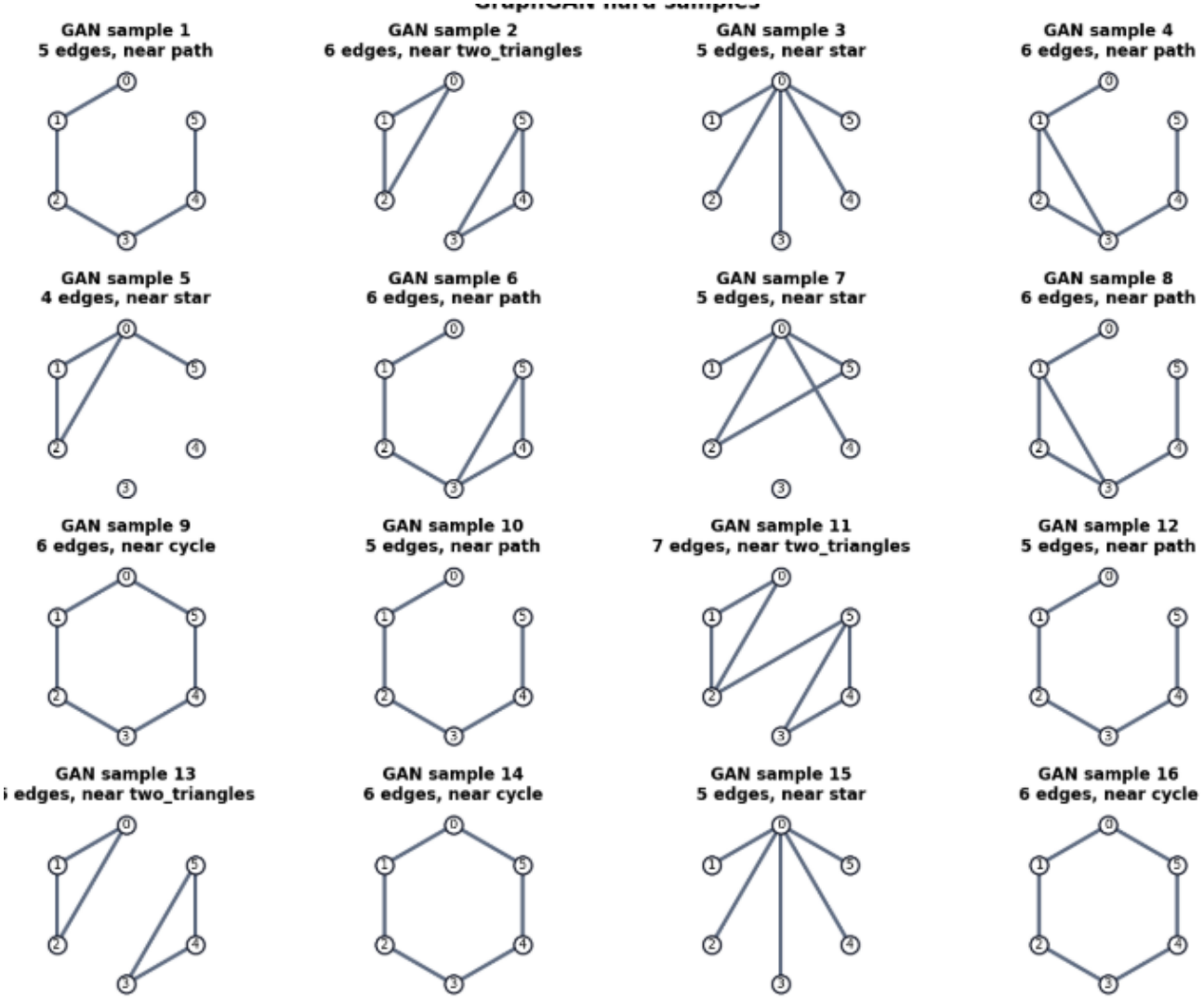
$$\mathcal{L}_{G,\text{adv}} = \mathrm{BCE}(D(x_{\text{fake}}), 1).$$

$$\mathcal{L}_{\text{FM}} = \|\mathbb{E}[x_{\text{fake}}] - \mathbb{E}[x_{\text{real}}]\|_2^2.$$

Edge Frequency Matching

$$\mathcal{L}_G = \mathcal{L}_{G,\text{adv}} + \lambda_{\text{FM}}\,\mathcal{L}_{\text{FM}}.$$

Graph GAN


GAN sample 1
5 edges, near path
GAN sample 2
6 edges, near two_triangles
GAN sample 3
5 edges, near star
GAN sample 4
6 edges, near path
GAN sample 5
4 edges, near star
GAN sample 6
6 edges, near path
GAN sample 7
5 edges, near star
GAN sample 8
6 edges, near path
GAN sample 9
6 edges, near cycle
GAN sample 10
5 edges, near path
GAN sample 11
7 edges, near two_triangles
GAN sample 12
5 edges, near path
GAN sample 13
edges, near two_triangles
GAN sample 14
6 edges, near cycle
GAN sample 15
5 edges, near star
GAN sample 16
6 edges, near cycle

Conditional Graph GAN

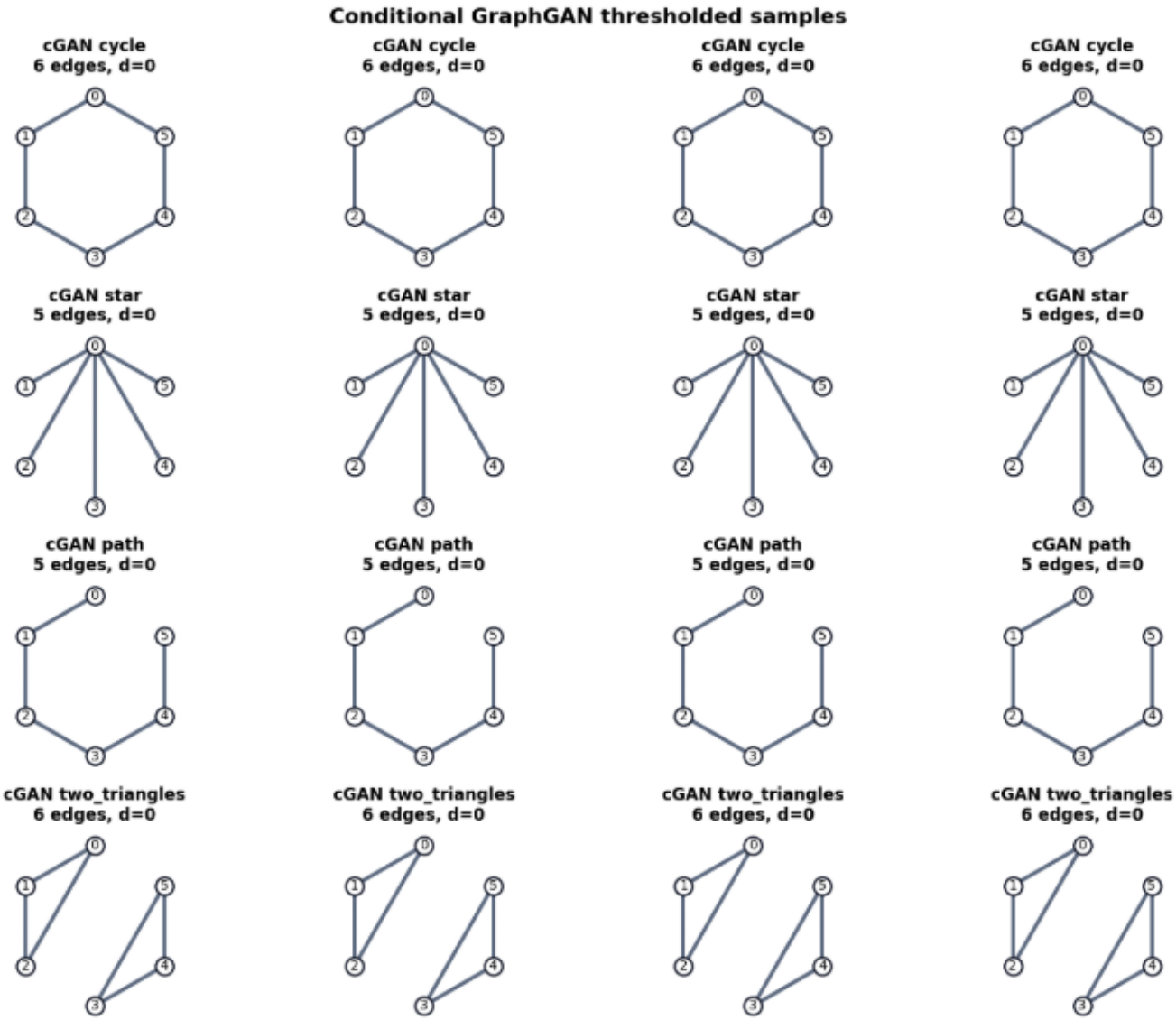
$$\ell = G_\theta([z, \text{onehot}(y)]), \qquad x_{\text{fake}} = \sigma(\ell).$$

```python
class ConditionalGraphGenerator(nn.Module):
    def __init__(self, latent_dim=16, n_edges=N_EDGES, hidden=128, n_classes=4):
        super().__init__()
        self.n_classes = n_classes
        self.net = nn.Sequential(
            nn.Linear(latent_dim + n_classes, hidden), nn.LeakyReLU(0.2),
            nn.Linear(hidden, hidden), nn.LeakyReLU(0.2),
            nn.Linear(hidden, hidden // 2), nn.LeakyReLU(0.2),
            nn.Linear(hidden // 2, n_edges),
        )

    def forward(self, z, labels):
        y = one_hot(labels, self.n_classes).to(z.device)
        return self.net(torch.cat([z, y], dim=1))
```

$$s = D_\phi([x, \text{onehot}(y)]).$$

```python
class ConditionalGraphDiscriminator(nn.Module):
    def __init__(self, n_edges=N_EDGES, hidden=64, n_classes=4):
        super().__init__()
        self.n_classes = n_classes
        self.net = nn.Sequential(
            nn.Linear(n_edges + n_classes, hidden), nn.LeakyReLU(0.2),
            nn.Dropout(0.25),
            nn.Linear(hidden, hidden), nn.LeakyReLU(0.2),
            nn.Dropout(0.25),
            nn.Linear(hidden, 1),
        )

    def forward(self, x, labels):
        y = one_hot(labels, self.n_classes).to(x.device)
        return self.net(torch.cat([x, y], dim=1))
```

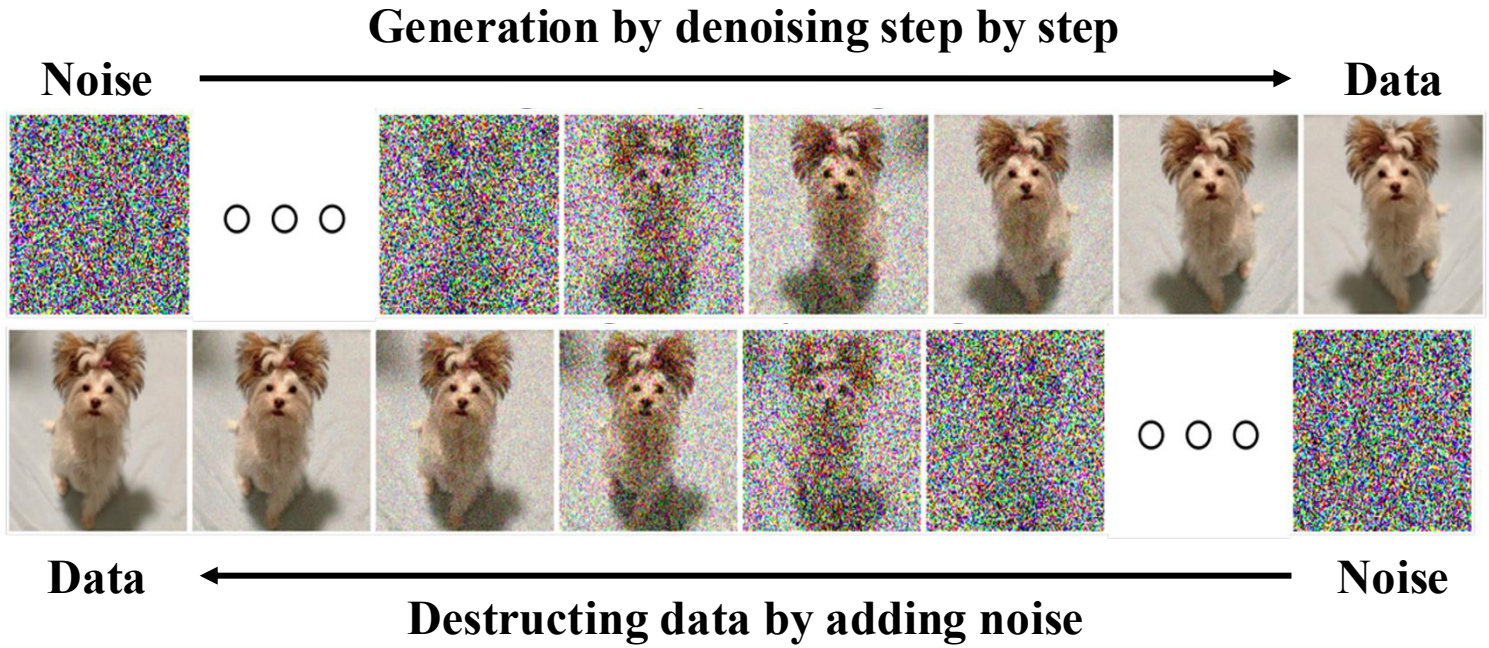
Conditional Graph GAN



Overview

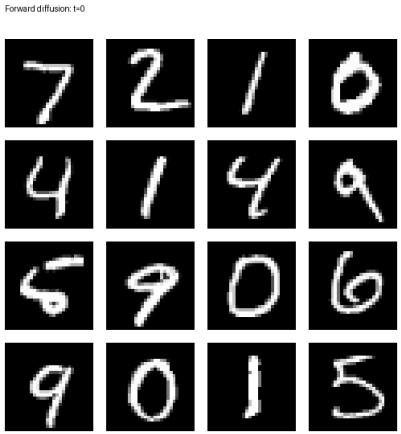
- Graph Definition
- Graph Auto-Encoder
- Graph Generative Adversarial Network
- Graph Diffusion
- Feature and Topology
- Permutation Invariant

Graph Diffusion



$q(x_t|x_{t-1})$ $\quad x_t \sim \mathcal{N}(\sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$

$x_t \sim \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$



Graph Diffusion

$$\theta^* = argmin\mathbb{E}_{x_0 \sim p_{data}(x), t \sim U(1,T), \epsilon \sim \mathcal{N}(0,I)} \|\epsilon_\theta(x_t, t) - \epsilon\|_2^2$$

$$x_T \sim \mathcal{N}(0, I) \qquad \hat{\epsilon} = \epsilon_{\theta^*}(x_t, t)$$

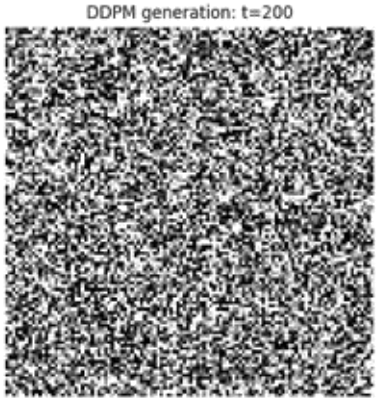
$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t.$$

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \bar{\alpha}_t}\,\hat{\epsilon}}{\sqrt{\bar{\alpha}_t}}. \qquad q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I).$$

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t$$

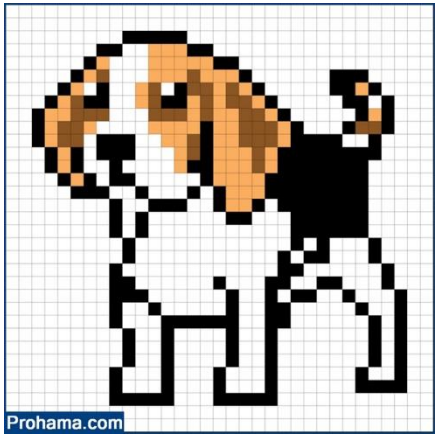
$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \varepsilon_\theta(x_t, t) \right)$$

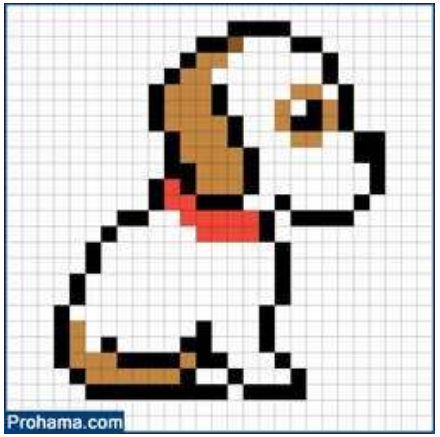
$$x_{t-1} = \mu_\theta(x_t, t) + \sqrt{\tilde{\beta}_t}\, z, \qquad z \sim \mathcal{N}(0, I)$$



Graph Diffusion

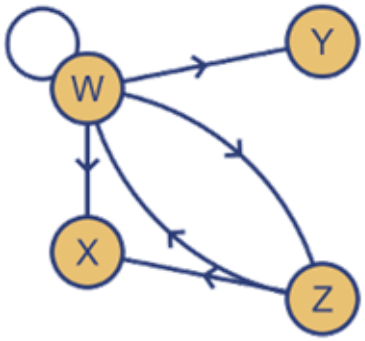
Continuous Pixel





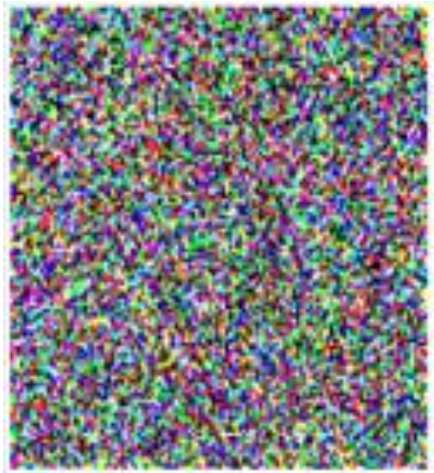
⊕

Discrete Label



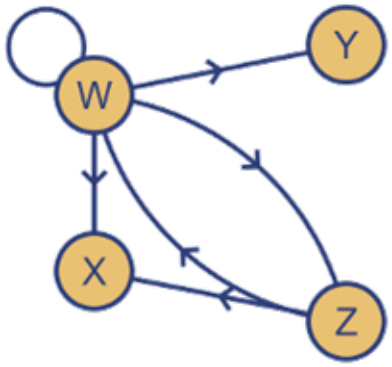
Directed graph with loop

	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0



⊕

Discrete Label



Directed graph with loop

	W	X	Y	Z
W	1	1	1	1
X	0	0	0	0
Y	0	0	0	0
Z	1	1	0	0

Random Flip →

	W	X	Y	Z
W	0	1	0	0
X	1	1	0	1
Y	1	1	0	0
Z	1	0	1	0

$$q(x_t \mid x_{t-1}): \quad x_t = x_{t-1} \oplus \eta_t, \qquad \eta_t \sim \text{Bernoulli}(\beta_t).$$

$$Q_t = \begin{pmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{pmatrix}$$

Graph Diffusion

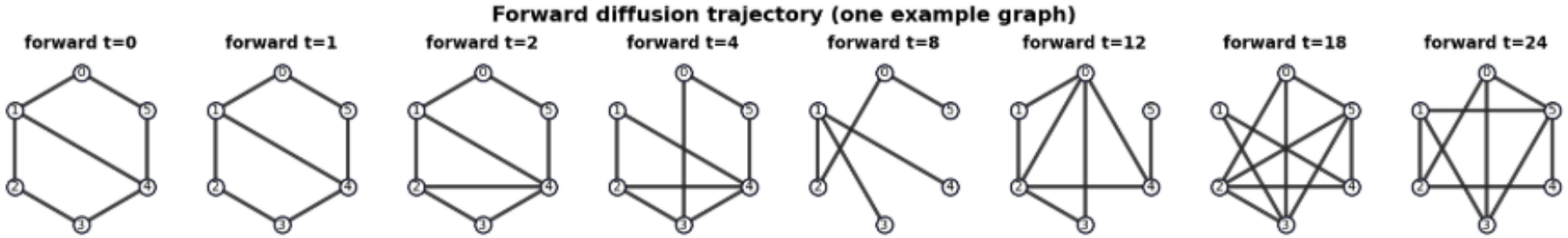
$$q(x_t \mid x_{t-1}): \quad x_t = x_{t-1} \oplus \eta_t, \qquad \eta_t \sim \text{Bernoulli}(\beta_t).$$

$$Q_t = \begin{pmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{pmatrix}$$

State 1 State 2 State Transition

State 1

State 2



Graph Diffusion

$$q(x_t \mid x_{t-1}): \quad x_t = x_{t-1} \oplus \eta_t, \qquad \eta_t \sim \text{Bernoulli}(\beta_t).$$

State 1 State 2 State Transition

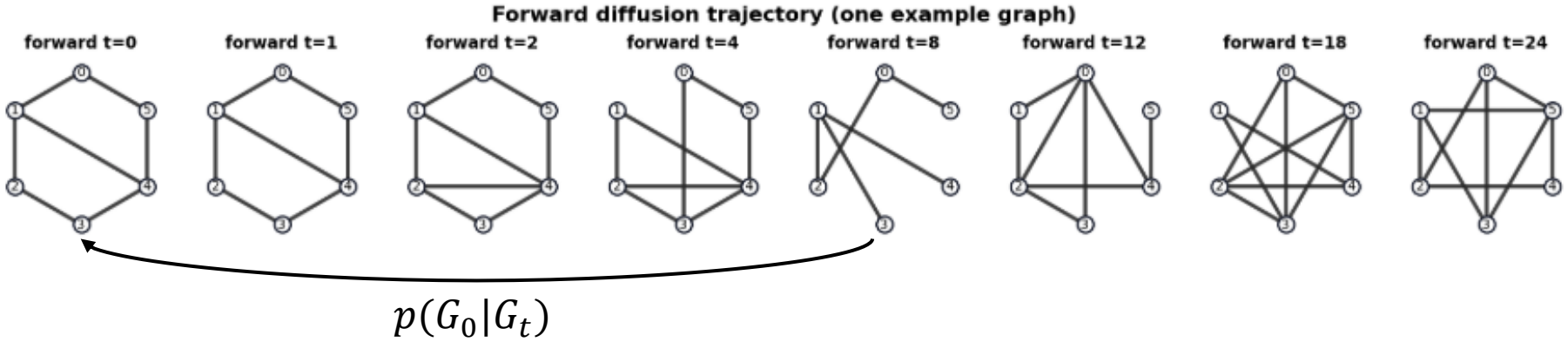
$$Q_t = \begin{pmatrix} 1-\beta_t & \beta_t \\ \beta_t & 1-\beta_t \end{pmatrix}$$

State 1

State 2

```python
def corrupt_edges(x0, t):
    """Sample x_t ~ q(x_t | x0) for the flip Markov chain.

    x0: (B, E) binary. t: (B,) integer timesteps in [1, T_disc].
    Implemented by sampling an effective flip mask with probability flip_marginal[t].
    """
    p = flip_marginal[t].unsqueeze(1)
    flips = torch.bernoulli(torch.ones_like(x0) * p)
    return torch.abs(x0 - flips)
```

Graph Diffusion



$$p_\theta(x_0 \mid x_t, t).$$

$$\ell_\theta(x_t, t) \in \mathbb{R}^{N_{\text{edges}}}, \qquad \hat{x}_0 = \sigma(\ell_\theta), \qquad \mathcal{L} = \text{BCEWithLogits}(\ell_\theta, x_0).$$

1. Sample a clean graph vector x_0 from the dataset.
2. Sample a timestep $t \sim \text{Uniform}\{1, \ldots, T\}$.
3. Corrupt: $x_t \leftarrow \texttt{corrupt\_edges}(x_0, t)$.
4. Predict: $\ell \leftarrow \texttt{denoiser}(x_t, t)$.
5. Optimize: `BCEWithLogitsLoss(ℓ, x0)`.

1. Initialize $x_T \sim \text{Bernoulli}(0.5)$ (random edges).
2. For $t = T, T-1, \ldots, 1$:
   - Predict clean probabilities: $\hat{x}_0 \leftarrow \sigma(\texttt{model}(x_t, t))$.
   - Make a hard estimate $\tilde{x}_0$ by thresholding (or Bernoulli sampling).
   - To get the next state, we **re-noise** that estimate at the lower noise level

     $$x_{t-1} \leftarrow \texttt{corrupt\_edges}(\tilde{x}_0, t-1).$$

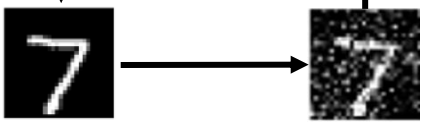
3. Return the final hard sample x_0.

Graph Diffusion

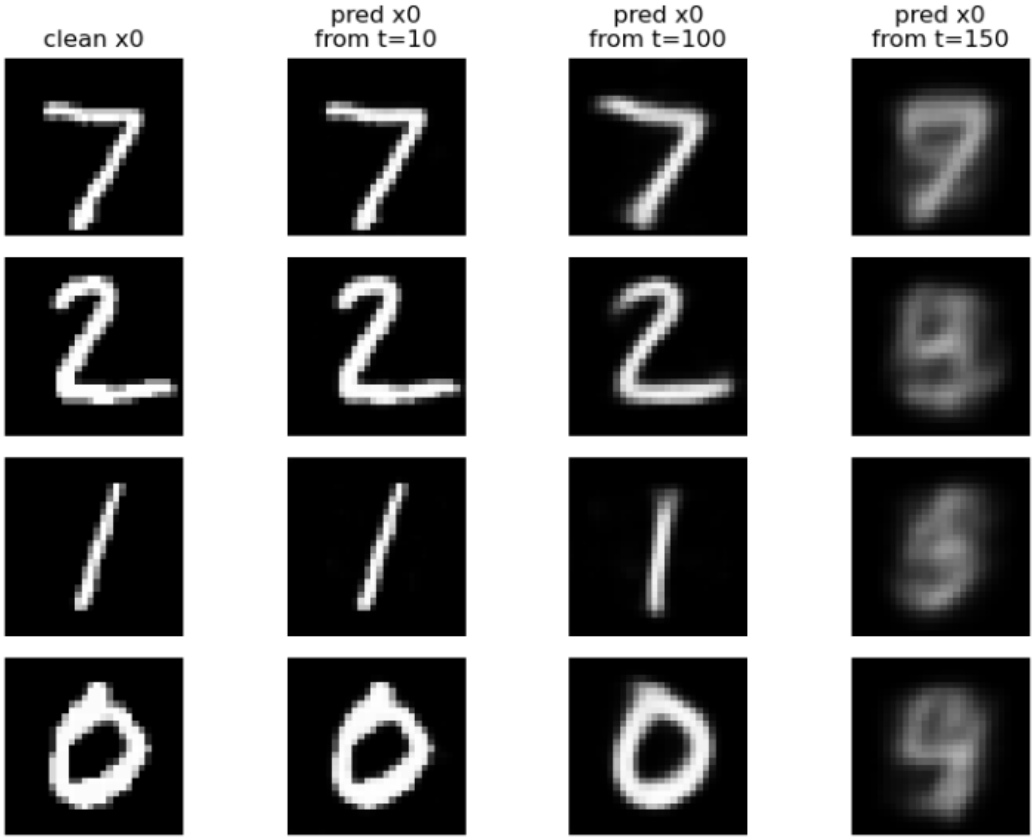
$$\theta^* = argmin\mathbb{E}_{x_0 \sim p_{data}(x), t \sim U(1,T), x_t \sim q(x_t|x_0)} \|\mathcal{F}_\theta(x_t, t) - x_0\|_2^2$$

$$x_T \sim \mathcal{N}(0, I) \qquad \mathcal{F}_{\theta^*}(x_T, t) = \bar{x}_0$$

$x_0 \sim \mathcal{F}_{\theta^*}(x_0|x_t, t)$

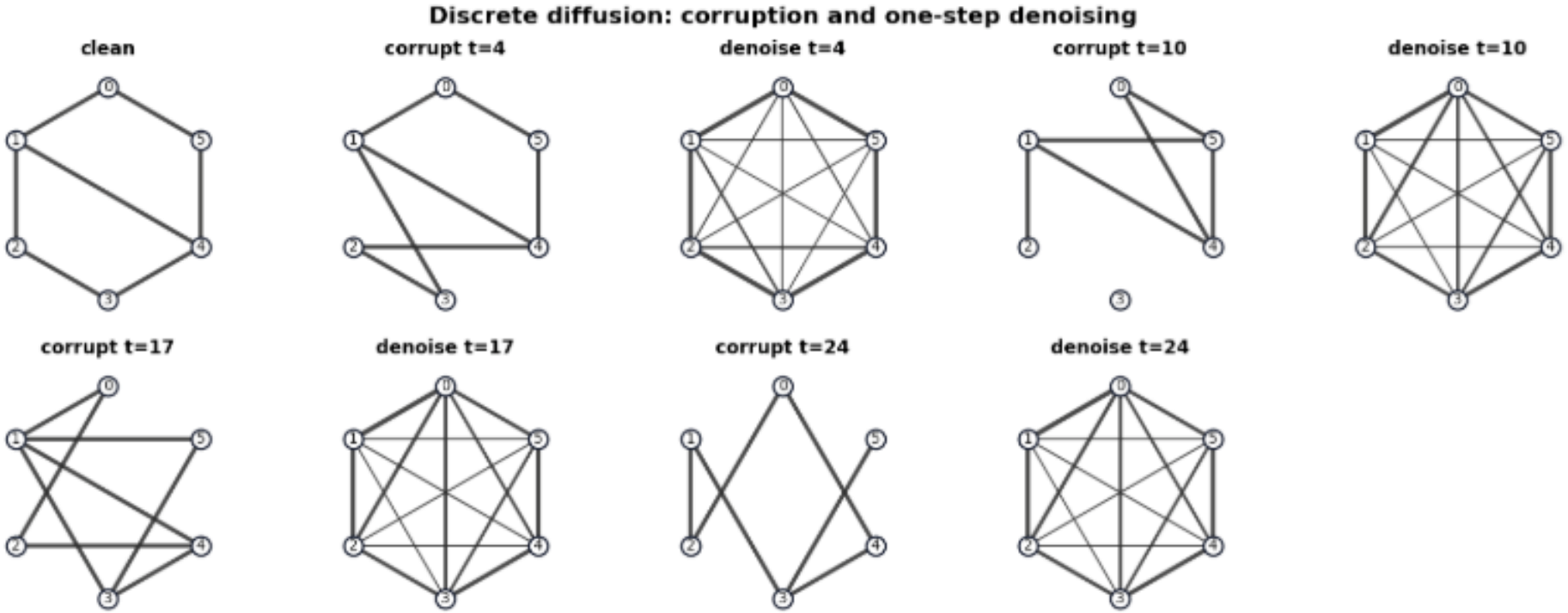


$x_t \sim q(x_t|x_0)$

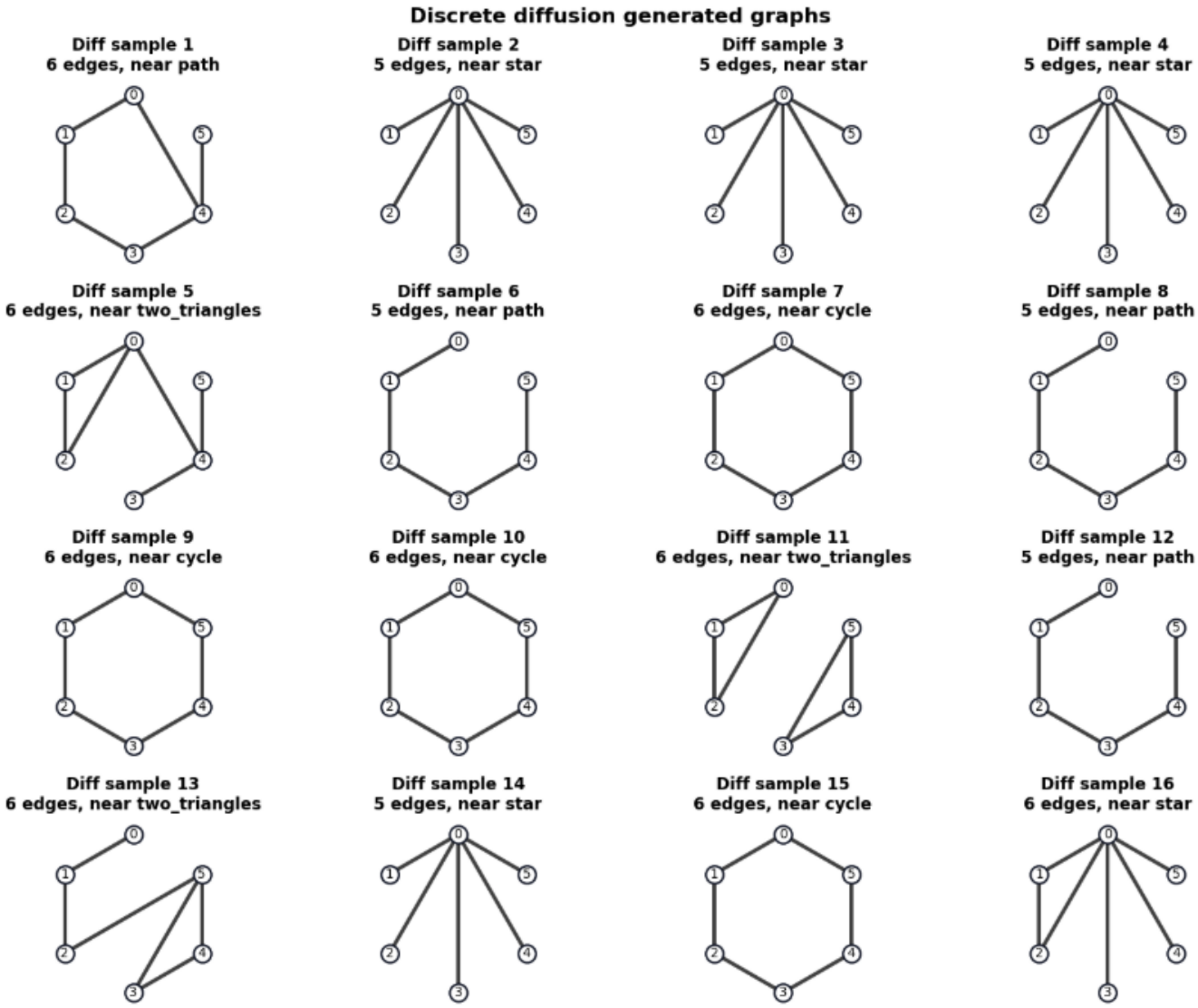


Predict from later time step gives more noisy images

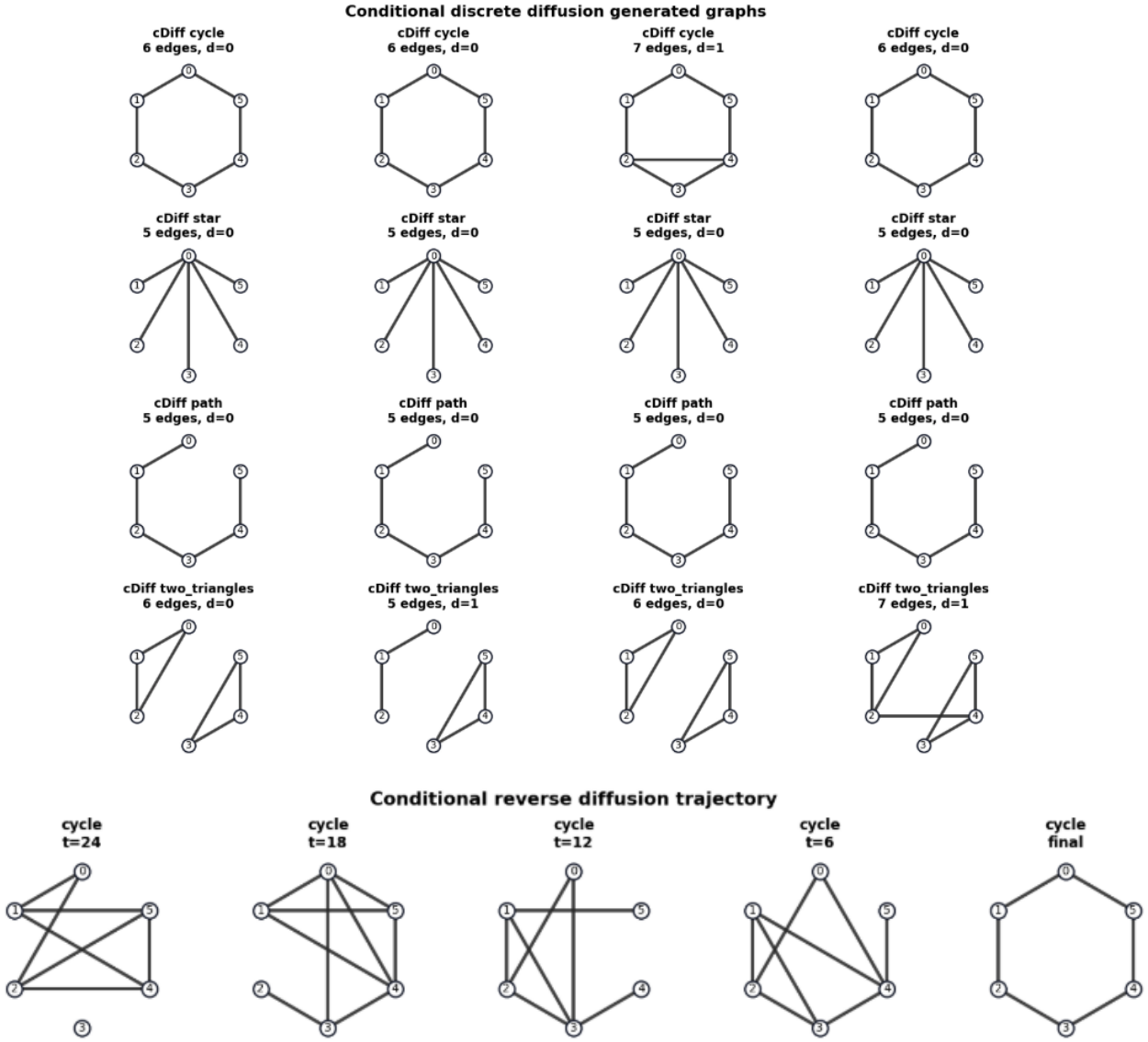
Graph Diffusion



Graph Diffusion



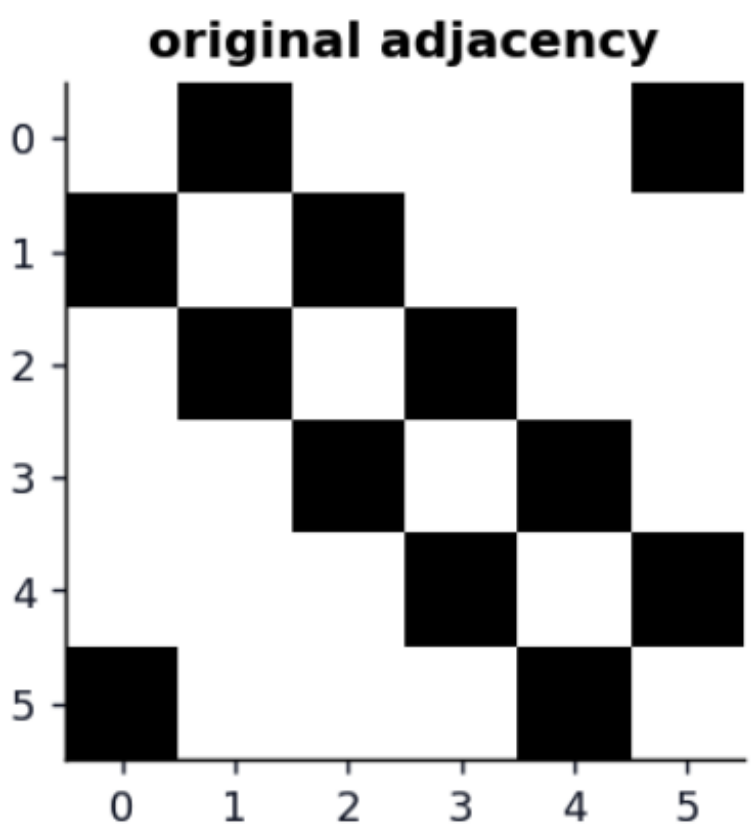
Graph Diffusion – Conditional Denoiser

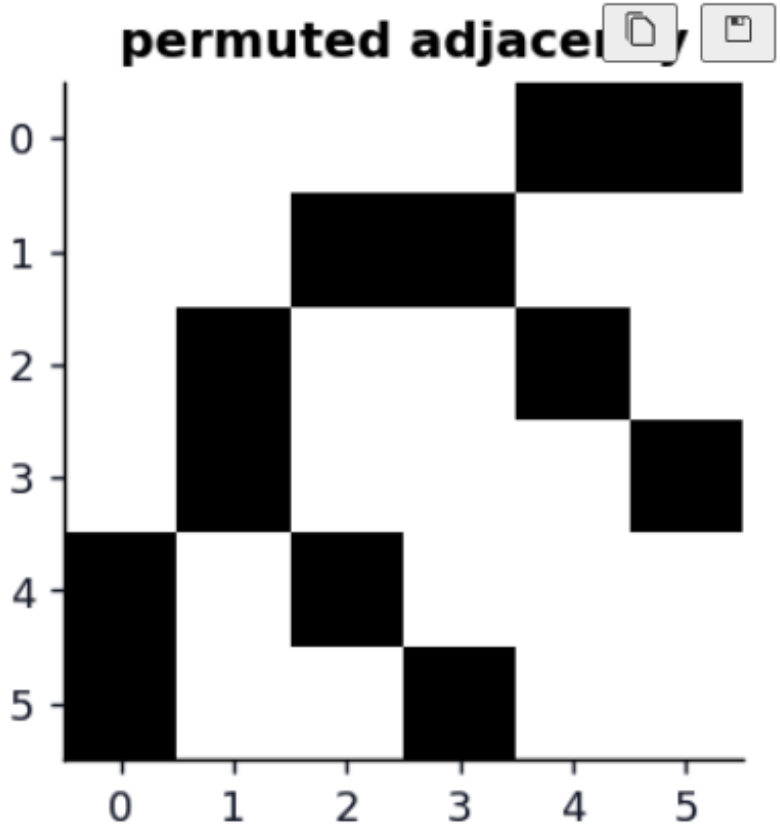


Overview

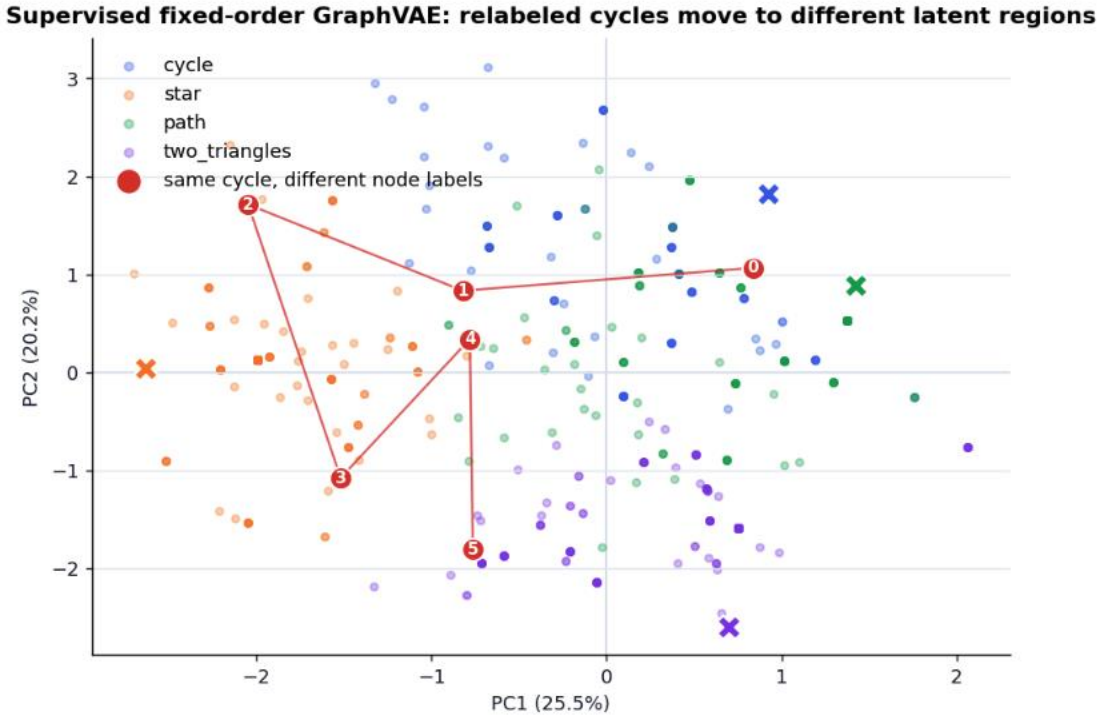
- Graph Definition
- Graph Auto-Encoder
- Graph Generative Adversarial Network
- Graph Diffusion
- Challenges
  - Size of the graph
  - Feature and Topology
  - Permutation Invariant

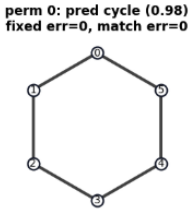
Overview

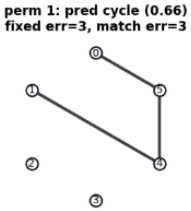

original adjacency
0
1
2
3
4
5
0
1
2
3
4
5

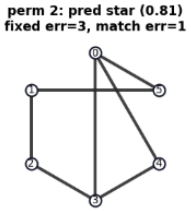

permuted adjace
0
1
2
3
4
5
0
1
2
3
4
5

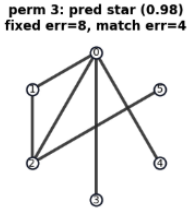
Overview

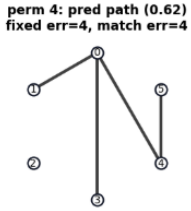

Supervised fixed-order GraphVAE: relabeled cycles move to different latent regions
cycle
star
path
two_triangles
same cycle, different node labels
PC2 (20.2%)
PC1 (25.5%)
3
2
1
0
−1
−2
−2
−1
0
1
2

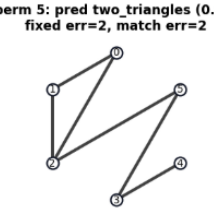

perm 0: pred cycle (0.98)
fixed err=0, match err=0


perm 1: pred cycle (0.66)
fixed err=3, match err=3


perm 2: pred star (0.81)
fixed err=3, match err=1


perm 3: pred star (0.98)
fixed err=8, match err=4


perm 4: pred path (0.62)
fixed err=4, match err=4


perm 5: pred two_triangles (0.97)
fixed err=2, match err=2

QA