Diffusion Models

10th Mar, 2025

Speaker: Li Longlong

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

1. What are Generative Models
2. Diffusion Models on Graphs
3. Applications of Graph Diffusion Models
4. Appendix

1 What are Generative Models

2 Diffusion Models on Graphs

3 Applications of Graph Diffusion Models

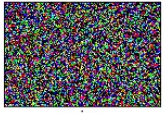
4 Appendix

Why generative modeling [17]

A well-trained deep neural network can classify animal images $\mathbf{x} \in \mathbb{Z}^D$, $y \in \mathcal{Y} = \{\text{cat, dog, horse}\}$ with high confidence $p(y|\mathbf{x})$. However, as shown by Szegedy et al [17], adding small perturbations to an image can drastically alter the predicted label while keeping the image visually unchanged.

$p(y = cat|\mathbf{x}) = 0.90$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.05$

+



noise

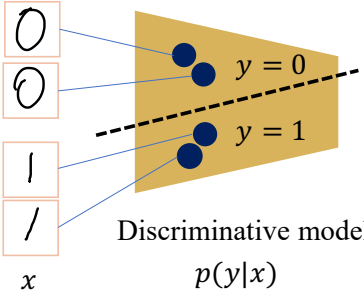
=

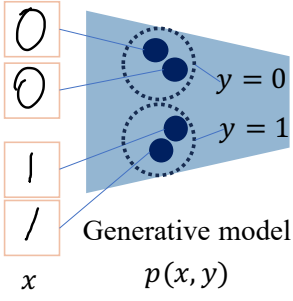
$p(y = cat|\mathbf{x}) = 0.05$
$p(y = dog|\mathbf{x}) = 0.05$
$p(y = horse|\mathbf{x}) = 0.90$

This vulnerability highlights the need for generative models, which can learn the underlying data distribution and improve robustness, adversarial defense, and uncertainty estimation in deep learning models.

Discriminative vs. Generative Models

	Discriminative Models	**Generative Models**
Goal	Learn decision boundary	Model data distribution
Probability	$p(Y\|X)$	$p(X, Y)$ or $p(X)$
Usage	Classification	Generation, Denoising
Examples	SVM, MLP, k-NN	GAN, VAE, Diffusion





Sampling from Bernoulli Space

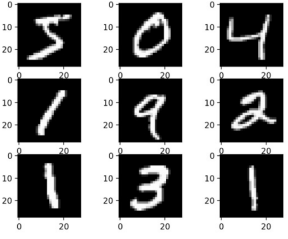
Bernoulli Distribution in MNIST

The MNIST dataset is often represented as binary (black/white) and can be modeled using a Bernoulli Distribution:

$$p(X = x) = p^x(1 - p)^{1-x}, \quad x \in \{0, 1\}$$

where X is a random variable representing the pixel state, x is its specific value (0 for white, 1 for black), and p is the probability of a pixel being 1 (obtained from data statistics).

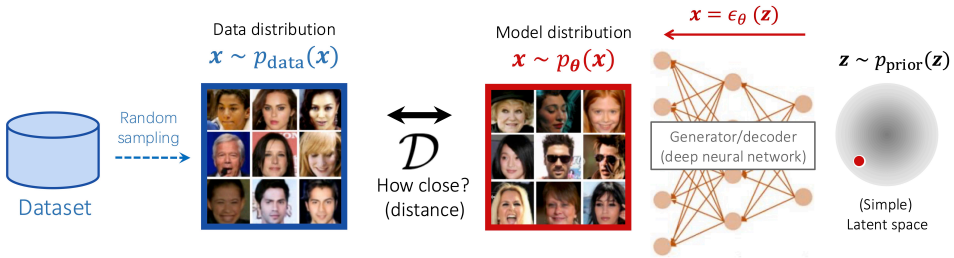
A. MNIST image



B. NN learning data

Deep Learning based Generative Models [8]

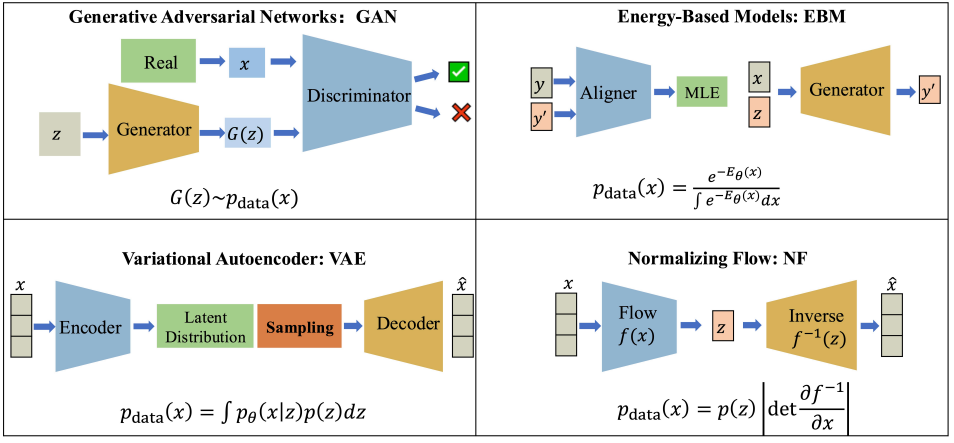
Generative models use neural networks to learn the target distribution.



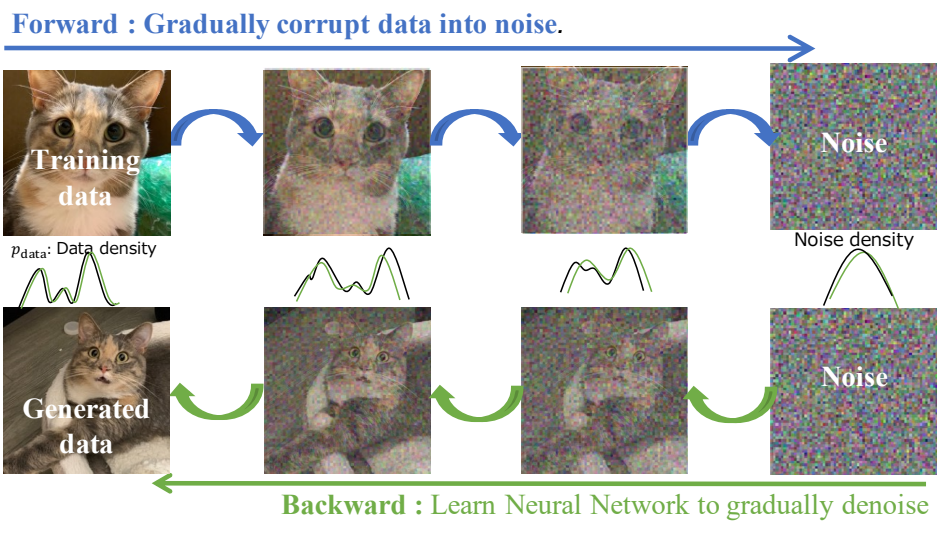
$$\min_\theta \mathcal{D}\left(p_\theta(x), p_{\text{data}}(x)\right)$$

where $\mathcal{D}$ is a divergence metric (e.g., KL divergence, Wasserstein distance).

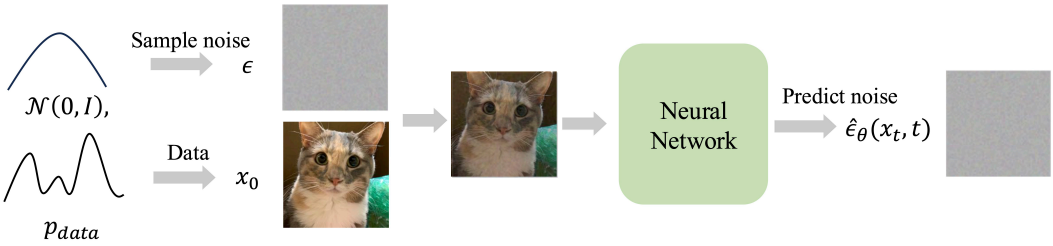
Comparison of deep generative models [1]



Diffusion Model Architecture [2]



DDPM Forward Process [4]



Markov Chain-based:

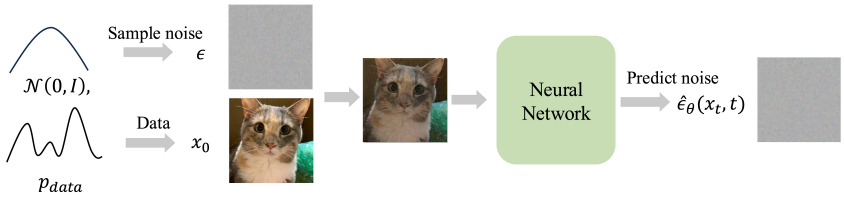
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I),$$

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \text{ where, } \alpha_t = 1-\beta_t, \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

Loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_0, \epsilon, t}\left[\|\epsilon_\theta(x_t, t) - \epsilon\|^2\right]$$

DDPM Reverse Process



Sampling Distribution:

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

where $\mu_\theta(x_t, t)$ is the mean, and $\Sigma_\theta(x_t, t) = \beta_t I$ is the variance.

Reverse Diffusion Mean Formula:

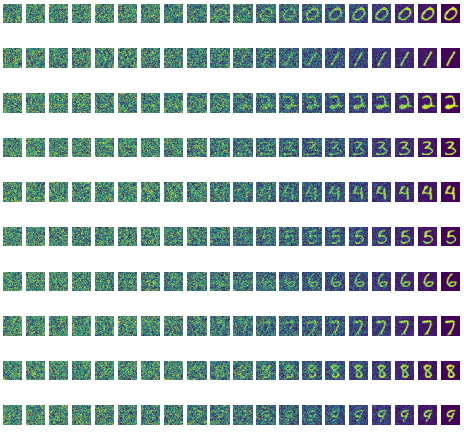
$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) + \sigma_t z$$

DDPM Sampling Process on MNIST [13]

- **Step 1:** Start from Gaussian noise $x_T \sim \mathcal{N}(0, I)$.
- **Step 2:** Iteratively refine the sample by denoising:

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

- **Step 3:** Repeat for $t = T, T-1, \ldots, 1$ to reconstruct a clean sample.



SGM (Score-Based Generative Model) [16, 14, 15]

Fokker-Planck Equation:

$$\frac{\partial p(x,t)}{\partial t} = -\nabla_x \cdot (p(x,t)v(x,t)) + D\nabla_x^2 p(x,t)$$

where, the first term is the **drift term**, describing how the data moves along a certain direction and the second term is the **diffusion term**, describing how noise propagates.

Forward Process (Stochastic Differential Equation):

$$dx = f(x,t)dt + g(t)dW_t$$

where $f(x,t)$ is the drift coefficient, $g(t)$ controls noise intensity, and dW_t represents a standard Wiener process.

Reverse Process (Inverse Stochastic Differential Equation):

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]dt + g(t)d\bar{W}_t$$

where $\nabla_x \log p_t(x)$ is the **score function**, which we need to estimate.

Score Matching Loss Function:

$$\mathcal{L}(\theta) = \mathbb{E}_{p_t(x)}\left[\lambda(t)\|\nabla_x \log p_t(x) - \epsilon_\theta(x,t)\|^2\right]$$

where $\epsilon_\theta(x,t)$ is the neural network estimating the score function.

Process of SGM

Density

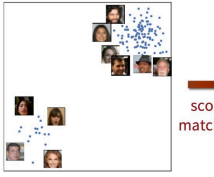
Densities follow **Fokker-Planck Equation (FPE)**

Forward SDE: $d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$

Data

Data follow SDE

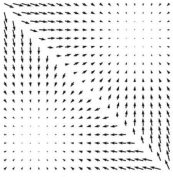
Stochastic reverse SDE : $d\boldsymbol{x}_t = [\boldsymbol{f}(\boldsymbol{x}_t, t) - g^2(t)\nabla_x \log p_t(\boldsymbol{x}_t)]dt + g(t)d\bar{\boldsymbol{w}}_t$



Data samples

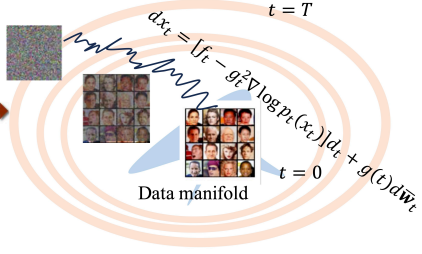
$\{x_1, x_2, \ldots, x_N\} \overset{\text{iid}}{\sim} p(x)$

score matching

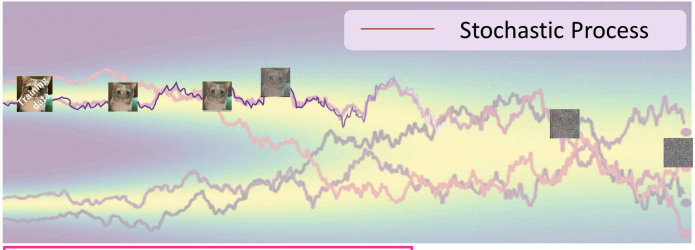


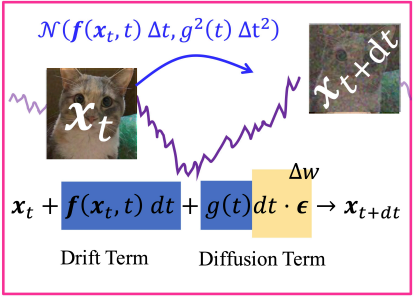
Scores

$\epsilon_\theta(x) \approx \nabla_x \log p(x)$



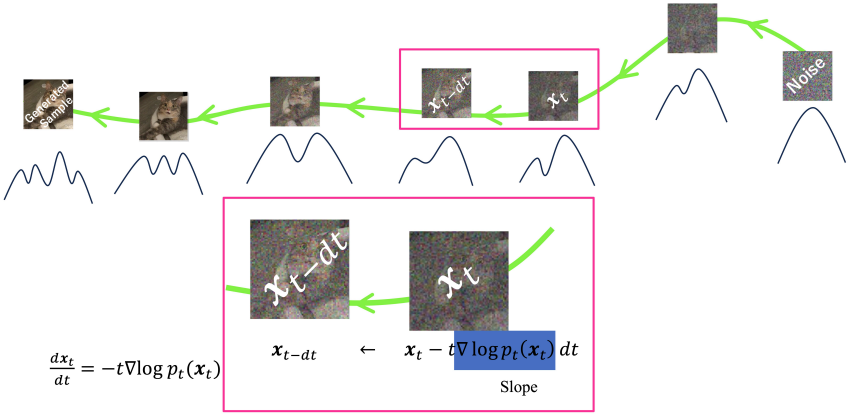
From DDPM to SGM: Forward Process



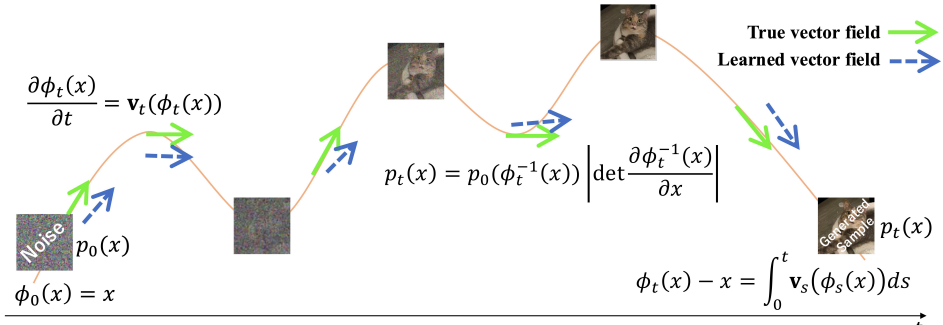


$$dx = f(x,t)dt + g(t)dW_t$$

From DDPM to SGM: Reverse Process



Flow Matching (FM) [9]



$$\mathcal{L} = \mathbb{E}_{p_t(x)}\left[\left\|\mathbf{v}_{t,\theta}(x) - \mathbf{v}_t(x)\right\|^2\right],$$

$$p_t(x) = \int p_t(x|x_1)q(x_1)dx_1 = \mathbb{E}_{q(x_1)}[p_t(x|x_1)],$$

$$\mathbf{v}_t(x) = \int \mathbf{v}_t(x|x_1)\frac{p_t(x|x_1)q(x_1)}{p_t(x)}dx_1 = \mathbb{E}_{p(x_1|x)}[\mathbf{v}_t(x|x_1)],$$

where x_1 is the true data, and $q(x_1)$ isits distribution

From SGM (Score-Based Generative Models) to Flow Matching

SGM (Score-Based Generative Models) relies on stochastic differential equations (SDEs) for diffusion:

$$dx = f(x, t)dt + g(t)dW_t$$

Flow Matching, however, adopts a deterministic ODE:

$$\frac{dx}{dt} = v_\theta(x, t)$$

How does Flow Matching transition from SGM?

- In SGM, we learn the score function $s_\theta(x, t) \approx \nabla_x \log p_t(x)$ to estimate data movement direction.
- In Flow Matching, we directly learn how data moves via the vector field $v_\theta(x, t)$, thereby eliminating the stochastic term dW_t.

This means:

- Flow Matching can be seen as an ODE version of SGM, removing stochasticity and making training and sampling more stable.
- Sampling only requires solving an ODE, avoiding SDE inversion, and thus improving speed.

From SGM to Flow Matching

Fokker-Planck Equation (FPE): $\frac{\partial p_t(x)}{\partial t} + \nabla_{\boldsymbol{x}} \cdot \left(p_t(x)\mathbf{v_t}(x)\right) + D\nabla_x^2 p_t(x) = 0$

Continuity Equation $\frac{\partial p_t(x)}{\partial t} + \mathbf{div}\left(p_t(x)\mathbf{v_t}(x)\right) = 0$

$$\partial_t p_t(\boldsymbol{x}) + \mathrm{div}_{\boldsymbol{x}}\left(\left[\boldsymbol{f}(\boldsymbol{x},t) - \frac{1}{2}g^2(t)\nabla_{\boldsymbol{x}} \log p_t(\boldsymbol{x})\right] p_t(\boldsymbol{x})\right) = 0$$

Density

Forward SDE: $d\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_t, t)dt + g(t)d\boldsymbol{w}_t$

Data

Probability flow ODE: $d\boldsymbol{x}_t = \left[\boldsymbol{f}(\boldsymbol{x}_t, t) - \frac{1}{2}g^2(t)\nabla_x \log p_t(\boldsymbol{x}_t)\right] dt$

Summary

Dimension	DDPM	SGM	FM
Core Concept	Markov chain noise addition & denoising	Score function of data distribution $\nabla_x \log p(x)$	Deterministic vector field $v_t(x)$
Mathematical Formulation	Discrete Markov chain: $q(x_t, x_{t-1}) = \mathcal{N}(\sqrt{1-\beta_t}\, x_{t-1}, \beta_t\, I)$	Stochastic Differential Equation (SDE): $dx = f(x,t)dt + g(t)dW_t$	Ordinary Differential Equation (ODE): $\frac{dx}{dt} = v_\theta(x,t)$
Forward Process (Noise Addition)	Discrete noise addition	Continuous diffusion (SDE)	Deterministic path transformation
Randomness	Yes, $\sigma_t z$	Yes, $g(t)d\bar{W}_t$	No
Reverse Process (Sampling)	Discrete denoising	Inverse SDE sampling:	ODE sampling:
Training Objective	Predict added noise: $\mathcal{L}(\theta) = \mathbb{E}\|\epsilon_\theta(x_t,t) - \epsilon\|^2$	Predict score function of data: $\mathcal{L}(\theta) = \mathbb{E}\|\nabla_x \log p_t(x) - s_\theta(x_t,t)\|$	Train vector field $v_\theta(x,t)$ to match true transport direction: $\mathcal{L}(\theta) = \mathbb{E}\|v_\theta(x,t) - u_t(x)\|^2$

1 What are Generative Models

2 Diffusion Models on Graphs

3 Applications of Graph Diffusion Models

4 Appendix

Graph Diffusion Models: Applications

Molecular Generation

- **GraphDF [11]**: Diffusion-based molecular graph generation for drug discovery.
- **EDM [5]**: E(3)-equivariant diffusion model for 3D molecular structures.

Social Network Modeling

- **DiGress [18]**: Discrete denoising diffusion model for graph generation.
- **GraphGDP [6]**: Diffusion-based modeling of dynamic graph relationships for recommendations.

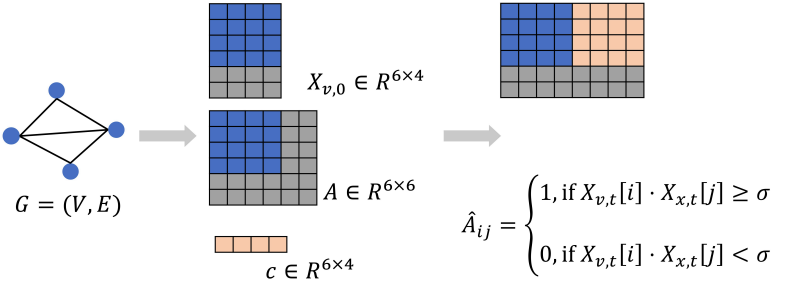
Knowledge Graph Completion

- **DiffKG [7]**: Enhancing knowledge graph representation with generative diffusion.
- **KGGen [12]**: Extracting structured knowledge graphs from unstructured text.

Other Applications

- **Diff3DHPE [19]**: Human 3D posture estimation using diffusion models.
- **Diff3F [3]**: Diffusion-based descriptors for untextured 3D shapes (meshes or point clouds).

Architecture of Graph Diffusion Model



Graph Diffusion Model: Core Workflow

- Graph Data Representation: A graph $G = (V, E)$ consists of nodes V and edges E.
- Forward Process:
  - Noise is gradually added to node features X and graph structure A.
  - At $t = T$, data transforms into Gaussian noise:

$$q(X_t|X_{t-1}) = \mathcal{N}(X_t; \sqrt{\alpha_t}X_{t-1}, (1 - \alpha_t)I) \tag{1}$$

- Backward Process via GNN:
  - A Graph Neural Network (GNN) is trained to learn the denoising function:

$$p(X_{t-1}|X_t) = \mathcal{N}(X_{t-1}; \mu_\theta(X_t, t), \Sigma_\theta(X_t, t)) \tag{2}$$

  - Here, μ_θ and Σ_θ are predicted by the GNN to remove noise.
- Sampling Process:
  - Start from Gaussian noise and iteratively reverse the diffusion process to restore the original graph $G = (V, E)$.

1 What are Generative Models

2 Diffusion Models on Graphs

3 Applications of Graph Diffusion Models

4 Appendix

Node Feature Diffusion

Graph DiT [10]

Graph Diffusion Transformers for Multi-Conditional Molecular Generation

NeurIPS 2024

Gang Liu, Jiaxin Xu, Tengfei Luo, Meng Jiang
University of Notre Dame
{gliu7, jxu24, tluo, mjiang2}@nd.edu

- Graph Diffusion Transformer (Graph DiT) addresses multi-conditional molecular generation.
- Challenges in molecular generation:
  - Existing diffusion models lack effective multi-condition control.
  - Need for an adaptive noise model for both nodes (atoms) and edges (bonds).
- Solution: A Transformer-based graph diffusion model that integrates conditional constraints into the diffusion process.

Condition Constraints in Graph DiT

- Types of Conditions:
  - Numerical Conditions:
    - Gas permeability (e.g., $O_2/N_2/CO_2$ separation) HIV inhibition rate LogP (lipophilicity), QED (drug-likeness)
  - Categorical Conditions:
    - Synthetic accessibility (SA) Blood-brain barrier penetration (BBBP) Beta-secretase 1 inhibitors (BACE)
- Condition Encoding:
  - Numerical conditions: Cluster encoding, get the One-hot encoding
  - Categorical conditions: One-hot encoding

Forward Process in Graph DiT

- The goal of diffusion is to gradually add noise to graph data until it becomes a Gaussian distribution.
- Standard Forward Process process:

$$q(G_t|G_{t-1}) = \mathcal{N}(G_t; \sqrt{\alpha_t}G_{t-1}, (1-\alpha_t)I) \tag{3}$$

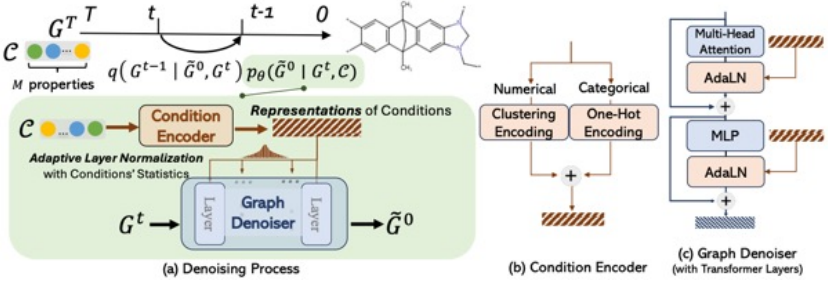
- To incorporate conditional information, Graph DiT introduces Condition Embedding:

$$q(G_t|G_{t-1}, c) = \mathcal{N}(G_t; \sqrt{\alpha_t}G_{t-1} + f(c), (1-\alpha_t)I) \tag{4}$$

- Here:
  - $f(c)$ is a condition transformation function that maps numerical/categorical conditions to the molecular feature space.
  - This ensures that the diffusion process is influenced by attribute constraints.

Backward Process in Graph DiT

- The goal of the Backward Process is to gradually recover condition-specific molecules from Gaussian noise.
- Reverse diffusion is formulated as: $p(G_{t-1}|G_t, c) = \mathcal{N}(G_{t-1}; \mu_\theta(G_t, c, t), \Sigma_\theta(G_t, c, t))$
  Here:
  - μ_θ and Σ_θ are predicted by Graph DiT to guide the denoising process.
  - Conditional information c directly influences each layer of the denoising network via the Transformer architecture.



Evaluation Metrics for Molecular Generation

1. **Validity** (V)
   - Measures the fraction of chemically valid molecules:

$$V = \frac{N_{\text{valid}}}{N_{\text{total}}}$$

2. **Distribution Learning** (D)
   - Evaluates similarity to training data using metrics like KL divergence or Fréchet ChemNet Distance (FCD):

$$D_{\text{KL}} = \sum_i P_i \log \frac{P_i}{Q_i}$$

3. **Condition Control** (C)
   - Assesses whether generated molecules satisfy specified conditions:

$$C = \frac{N_{\text{condition}}}{N_{\text{total}}}$$

Datasets and Evaluation Metrics

- Datasets:
  - Polymers: Focuses on molecular generation with gas permeability properties (O_2/ N_2 / CO_2 separation).
- Evaluation Metrics:
  - Validity: Ensures that generated molecules follow chemical rules.
  - Distribution Learning: Measures similarity between generated molecules and training data.
  - Condition Control: Evaluates whether generated molecules satisfy specified property constraints.

Table 1: Multi-Conditional Generation of 10K Polymers: Results on the synthetic score (Synth.) and three numerical properties (gas permeability for O_2, N_2, CO_2). MAE is calculated between the input conditions and the properties of the generated polymers using Oracles. Best results are **highlighted**.

Model	Validity ↑ (w/o rule checking)	Distribution Learning				Condition Control				
		Coverage ↑	Diversity ↑	Similarity ↑	Distance ↓	Synth. ↓	O_2Perm ↓	N_2Perm ↓	CO_2Perm ↓	Avg. MAE ↓
Graph GA	1.0000 (N.A.)	11/11	0.8828	0.9269	9.1882	1.3307	1.9840	2.2900	1.9489	1.8884
MARS	1.0000 (N.A.)	11/11	0.8375	0.9283	7.5620	1.1658	1.5761	1.8327	1.6074	1.5455
LSTM-HC	0.9910 (N.A.)	10/11	0.8918	0.7937	18.1562	1.4251	1.1003	1.2365	1.0772	1.2098
JTVAE-BO	1.0000 (N.A.)	10/11	0.7366	0.7294	23.5990	**1.0714**	1.0781	1.2352	1.0978	1.1206
DiGress	0.9913 (0.2362)	11/11	0.9099	0.2724	22.7237	2.9842	1.7163	2.0630	1.6738	2.1093
DiGress v2	0.9812 (0.3057)	11/11	**0.9105**	0.2771	21.7311	2.7507	1.7130	2.0632	1.6648	2.0479
GDSS	0.9205 (0.9076)	9/11	0.7510	0.0000	34.2627	1.3701	1.0271	1.0820	1.0683	1.1369
MOOD	0.9866 (0.9205)	11/11	0.8349	0.0227	39.3981	1.4019	1.4961	1.7603	1.4748	1.5333
Graph DiT-LCC (Ours)	0.9753 (0.8437)	11/11	0.8875	0.9560	7.0949	1.3099	0.8001	0.9562	0.8125	0.9697
Graph DiT (Ours)	0.8245 (0.8437)	11/11	0.8712	**0.9600**	**6.6443**	1.2973	**0.7440**	**0.8857**	**0.7550**	**0.9205**

1 What are Generative Models

2 Diffusion Models on Graphs

3 Applications of Graph Diffusion Models

4 Appendix

DDPM: Denoising Diffusion Probabilistic Model [4]

Goal: Learn a generative model by reversing a Markovian forward diffusion process.

- Forward process:

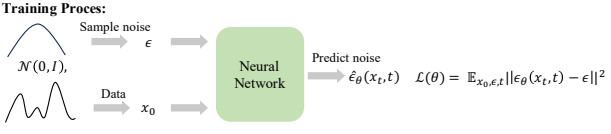
$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)I)$$

- Reverse process:

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

- Loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{t,x_0,\epsilon}\left[\|\epsilon_\theta(x_t, t) - \epsilon\|^2\right]$$



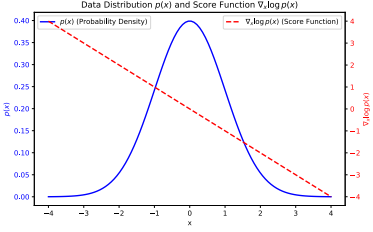
Score Function: Guiding the Sampling Process

Definition

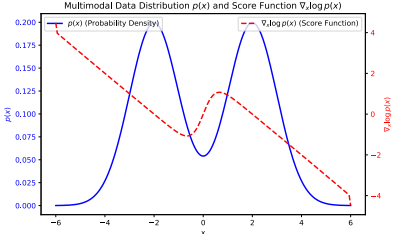
The score function is the gradient of the log probability density:

$$s(x) = \nabla_x \log p(x)$$

A. Unimodal



B. Multimodal



- Indicates the direction where probability density increases the most.
- Helps guide samples towards high-probability regions.

Introduction to SGM

Score-Based Generative Models (SGM) rely on Score Matching and Stochastic Differential Equations (SDEs) for data generation. The core ideas are:

- Learning the gradient of data distribution, known as the score function $\nabla_x \log p(x)$.
- Reversing data distribution using SDEs or ODEs for efficient sampling.

Why Score-Based Generative Models?

In diffusion models (e.g., DDPM), we use a Markov chain to gradually add and remove noise. SGM instead models data evolution using continuous noise diffusion (SDE):

- Forward Process (Noise Diffusion): Data gradually transforms into Gaussian noise.
- Reverse Process (Denoising): Recover data using the score function $\nabla_x \log p_t(x)$.

SGM learns $\nabla_x \log p_t(x)$ via Score Matching, directly guiding denoising in data space.

Forward Diffusion Process

SGM models data evolution using Stochastic Differential Equations (SDEs):

$$dx = f(x, t)dt + g(t)dW_t$$

- $f(x, t)$ controls drift (systematic movement).
- $g(t)dW_t$ controls diffusion (random noise), where dW_t is a Wiener process.

A common example is Variational Perturbation SDE (VP-SDE):

$$dx = -\frac{1}{2}\beta(t)xdt + \sqrt{\beta(t)}dW_t$$

- $\beta(t)$ is a noise schedule function, similar to DDPM's β_t.
- This process progressively moves data toward a standard Gaussian distribution.

Reverse Denoising Process

If we know the probability distribution $p_t(x)$ at time t, we can restore data using the Reverse SDE:

$$dx = [f(x,t) - g(t)^2 \nabla_x \log p_t(x)]\,dt + g(t)\,d\bar{W}_t$$

- $\nabla_x \log p_t(x)$ is the score function, describing the gradient of data density.
- This process transforms noise $p_T(x)$ back into data distribution $p_0(x)$.

Sampling in SGM

After training, we can generate data using two methods:

(1) Reverse SDE Sampling

$$dx = [f(x,t) - g(t)^2 s_\theta(x,t)]\,dt + g(t)\,d\bar{W}_t$$

- Uses stochastic SDE sampling, introducing some randomness.
- Typically solved using Euler-Maruyama method.

(2) Score-Based ODE Sampling

$$dx = [f(x,t) - \frac{1}{2} g(t)^2 s_\theta(x,t)]\,dt$$

- No stochastic noise dW_t, making it more stable.
- Solved efficiently using Runge-Kutta or Adjoint methods.

Conclusion and Importance of Score Matching

- Score Matching estimates $\nabla_x \log p(x)$ without normalizing $p(x)$.
- Fisher Score properties allow us to simplify expectation terms.
- Integration by parts transforms the loss function into a computable form.
- Hutchinson's Trick approximates the trace term efficiently.
- Enables scalable training of energy-based models and diffusion models.

Flow Matching (FM)

Core Idea: Flow Matching (FM) learns a eterministic ODE-based vector field, enabling data to evolve from an initial distribution $p_0(x)$ (typically noise) to a target distribution $p_1(x)$ (e.g., images or molecular structures).

Mathematical Formulation of Flow Matching: Flow Matching is built upon a deterministic transformation $\phi_t(x)$, which describes how data evolves from $p_0(x)$ to $p_1(x)$:

$$\frac{\partial \phi_t(x)}{\partial t} = \mathbf{v}_t(\phi_t(x)), \quad \phi_0(x) = x$$

- $\phi_t(x)$ is a time-evolving mapping that describes data movement.
- $\mathbf{v}_t(x)$ is the vector field that governs the flow direction at time t.

By integrating over time:

$$\phi_t(x) - x = \int_0^t \mathbf{v}_s(\phi_s(x))\, ds$$

This means that given an initial data point x, we can solve the ODE to obtain its position $\phi_t(x)$ at time t.

Flow Matching (FM)

Evolution of the Target Distribution: In Flow Matching, we aim to learn $\mathbf{v}_t(x)$ such that the data distribution evolves over time:

$$p_t(x) = p_0(\phi_t^{-1}(x)) \left| \det \frac{\partial \phi_t^{-1}(x)}{\partial x} \right|$$

- This equation describes how to compute the evolving data distribution $p_t(x)$ through the transformation $\phi_t(x)$.
- The term $\det \frac{\partial \phi_t^{-1}(x)}{\partial x}$ is the Jacobian determinant, which accounts for the change in scale during transformation.

Loss Function: To learn the optimal vector field $\mathbf{v}_t(x)$, we minimize:

$$\mathcal{L} = \mathbb{E}_{p_t(x)} \left[\|\mathbf{v}_{t,\theta}(x) - v_t(x)\|^2 \right]$$

- $\mathbf{v}_{t,\theta}(x)$ is the neural network approximation of the true vector field.
- $v_t(x)$ is the true target flow direction.
- The training objective is to ensure that $\mathbf{v}_{t,\theta}(x)$ closely approximates $v_t(x)$, guiding the correct data flow.

Flow Matching (FM)

Computing the Target Vector Field: To compute $v_t(x)$, we use conditional probability paths:

$$p_t(x) = \int p_t(x|x_1)\, q(x_1)\, dx_1 = \mathbb{E}_{q(x_1)}[p_t(x|x_1)]$$

This means:

- From a given data sample x_1, we define a probability path $p_t(x|x_1)$, describing how data transitions from x_1 to x.
- By marginalizing over x_1, we obtain $p_t(x)$, which describes the overall data distribution at time t.

Further, we compute the vector field:

$$\mathbf{v}_t(x) = \mathbb{E}_{p(x_1|x)}[v_t(x|x_1)]$$

This implies:

- We can estimate $v_t(x)$ by sampling x_1.
- During training, we approximate this expectation using Monte Carlo methods.

[1] Zeeshan Ahmad et al. “Understanding GANs: Fundamentals, variants, training challenges, applications, and open problems”. In: *Multimedia Tools and Applications* (2024), pp. 1–77.

[2] Deep Learning AI. *Graph Diffusion Models Explained*. YouTube video. Accessed: 2024-02-10. 2024. URL: https://www.youtube.com/watch?v=B-d_3xX6ss4.

[3] Niladri Shekhar Dutt, Sanjeev Muralikrishnan, and Niloy J Mitra. “Diffusion 3d features (diff3f): Decorating untextured shapes with distilled semantic features”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4494–4504.

[4] Jonathan Ho, Ajay Jain, and Pieter Abbeel. “Denoising diffusion probabilistic models”. In: *Advances in neural information processing systems* 33 (2020), pp. 6840–6851.

[5] Emiel Hoogeboom et al. “Equivariant diffusion for molecule generation in 3d”. In: *International conference on machine learning*. PMLR. 2022, pp. 8867–8887.

[6] Han Huang et al. “Graphgdp: Generative diffusion processes for permutation invariant graph generation”. In: *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2022, pp. 201–210.

[7] Yangqin Jiang et al. “Diffkg: Knowledge graph diffusion model for recommendation”. In: *Proceedings of the 17th ACM international conference on web search and data mining*. 2024, pp. 313–321.

[8] Chieh-Hsin Lai and Yuki Mitsufuji. *Evolution of Diffusion Models: From Birth to Enhanced Accuracy and Efficiency*. 2024. URL: https://github.com/ChiehHsinJesseLai/SeriesTalksDiffusionModels2024.

[9] Yaron Lipman et al. "Flow matching for generative modeling". In: *arXiv preprint arXiv:2210.02747* (2022).

[10] Gang Liu et al. "Graph Diffusion Transformers for Multi-Conditional Molecular Generation". In: *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. 2024.

[11] Youzhi Luo, Keqiang Yan, and Shuiwang Ji. "Graphdf: A discrete flow model for molecular graph generation". In: *International conference on machine learning*. PMLR. 2021, pp. 7192–7203.

[12] Belinda Mo et al. "KGGen: Extracting Knowledge Graphs from Plain Text with Language Models". In: *arXiv preprint arXiv:2502.09956* (2025).

[13] William Peebles and Saining Xie. "Scalable diffusion models with transformers". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 4195–4205.

[14] Yang Song and Stefano Ermon. "Improved Techniques for Training Score-Based Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 12438–12448.

[15] Yang Song et al. “Maximum Likelihood Training of Score-Based Diffusion Models”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021.

[16] Yang Song et al. “Score-based generative modeling through stochastic differential equations”. In: *arXiv preprint arXiv:2011.13456* (2020).

[17] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[18] Clement Vignac et al. “Digress: Discrete denoising diffusion for graph generation”. In: *arXiv preprint arXiv:2209.14734* (2022).

[19] Jieming Zhou et al. “Diff3dhpe: A diffusion model for 3d human pose estimation”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 2092–2102.

Thanks!