EE626: Graph generative models

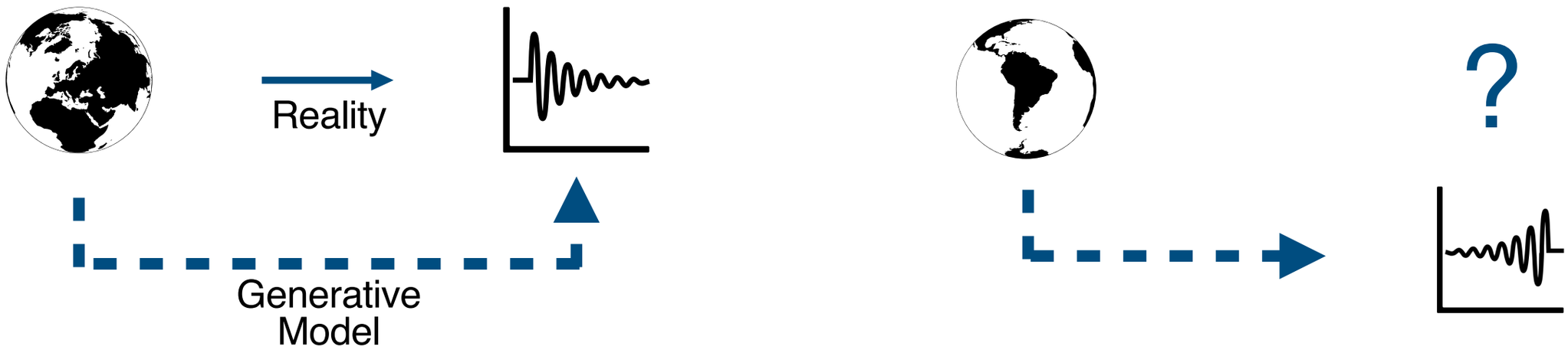
Dr Dorina Thanou

01.10.2025

Generative models as simulators of real world

A generative model simulates how the data is generated in the real world. "Modelling" is understood in almost every science as unveiling this generating process by hypothesizing theories and testing these theories through observations.

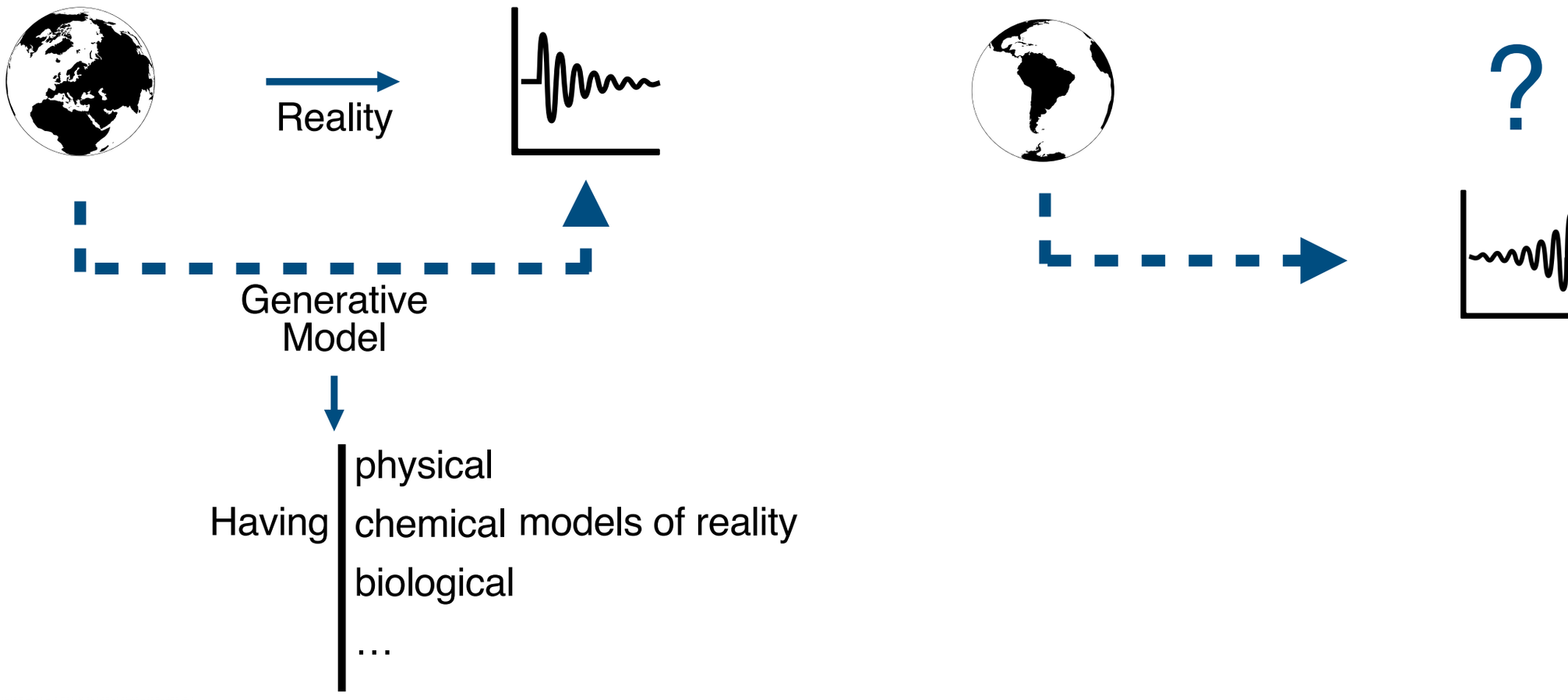
in *An Introduction to Variational Autoencoders*, D. Kingma and M. Welling, 2019



Generative models as simulators of real world

A generative model simulates how the data is generated in the real world. "Modelling" is understood in almost every science as unveiling this generating process by hypothesizing theories and testing these theories through observations.

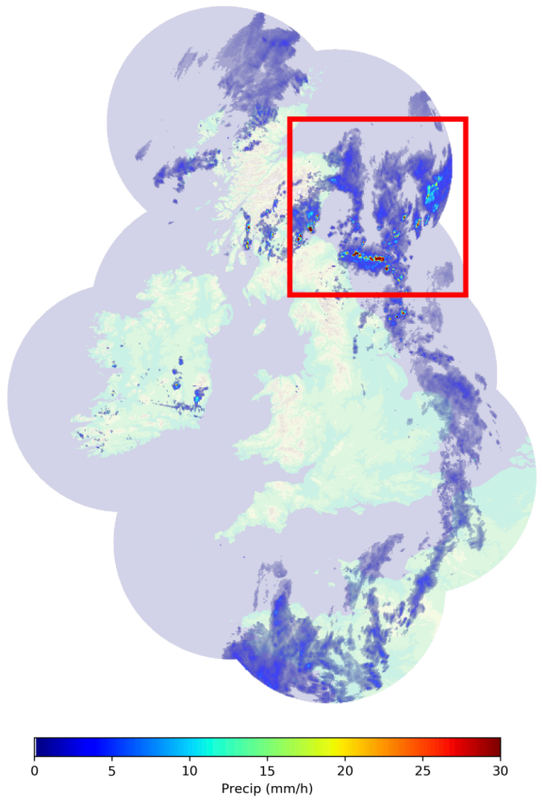
in *An Introduction to Variational Autoencoders*, D. Kingma and M. Welling, 2019



Generative models for science

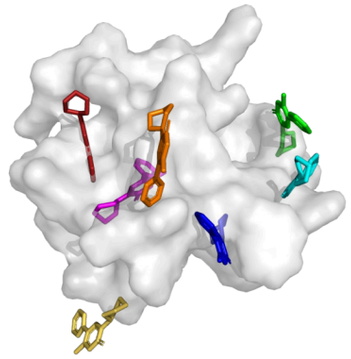
Climate Science

- Precipitation nowcasting



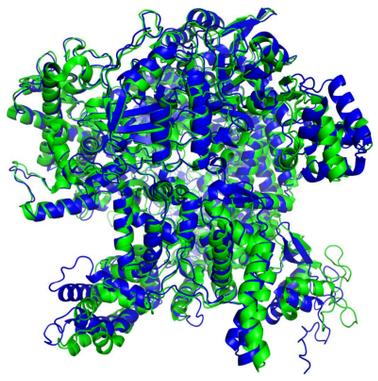
Chemistry/Drug discovery

- Molecular docking



Biology

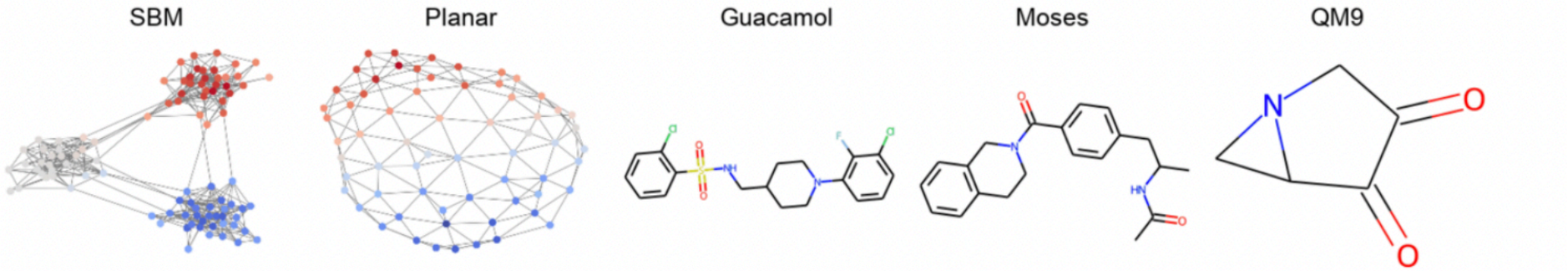
- Protein folding

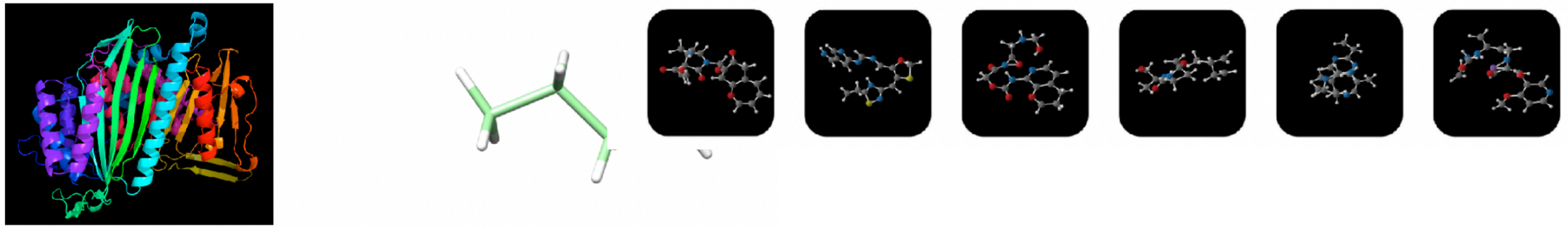


Data Completion and Compressed Representations

[*Skilful precipitation nowcasting using deep generative models of radar*, Ravuri Suman et al., Nature, 2021]
[*DiffDock: Diffusion Steps, Twists, and Turns for Molecular Docking*, Corso et al., ICLR, 2023]
[*Highly accurate protein structure prediction with AlphaFold*, Jumper et al., Nature, 2021]
[*Learning interactive real-world simulators*, Yang et al., ICLR, 2024]

Generative models in non-Euclidean domain

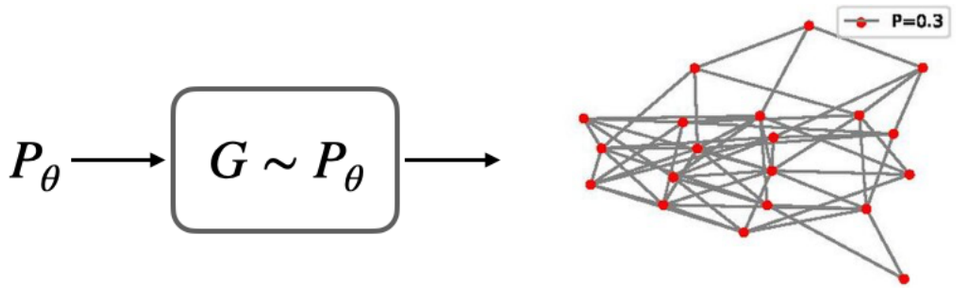




Graph generation includes the process of modelling and generating real graphs

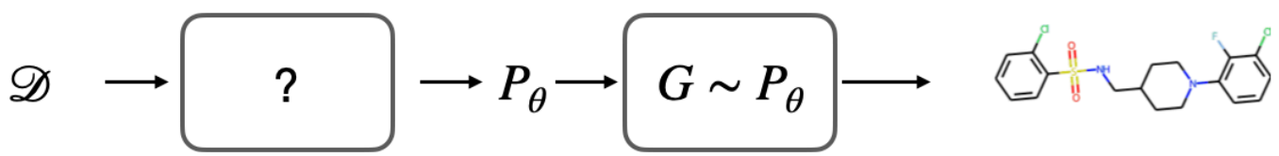
Graph generative models

- Random graph models



- Capture simple graph distribution
- Limited capacity to model complex dependencies
- Only capable of modelling a few statistical properties of graphs

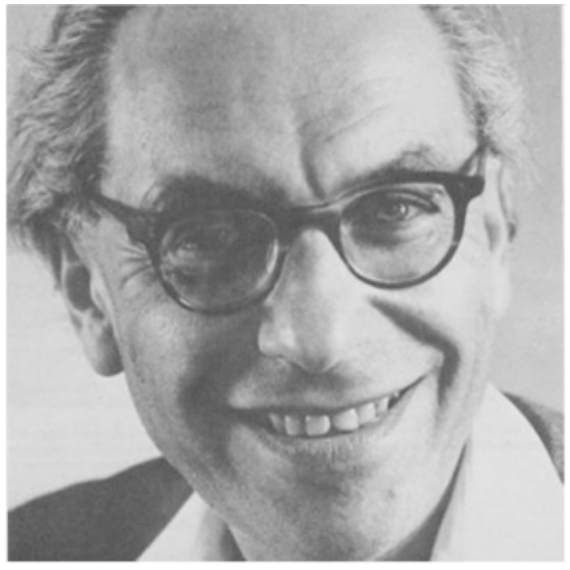
- Deep generative models



- Learn generative models directly from an observed set of graphs
- Can model highly complex structures such as proteins

Erdös-Rényi model (1960)

Pál Erdös
(1913-1996)



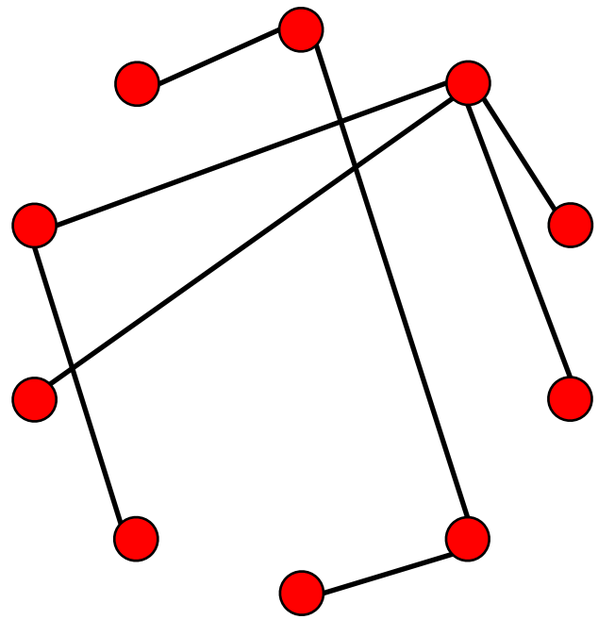
Alfréd Rényi
(1921-1970)

$G(N, L)$ model: N labeled nodes are connected with L randomly placed links.

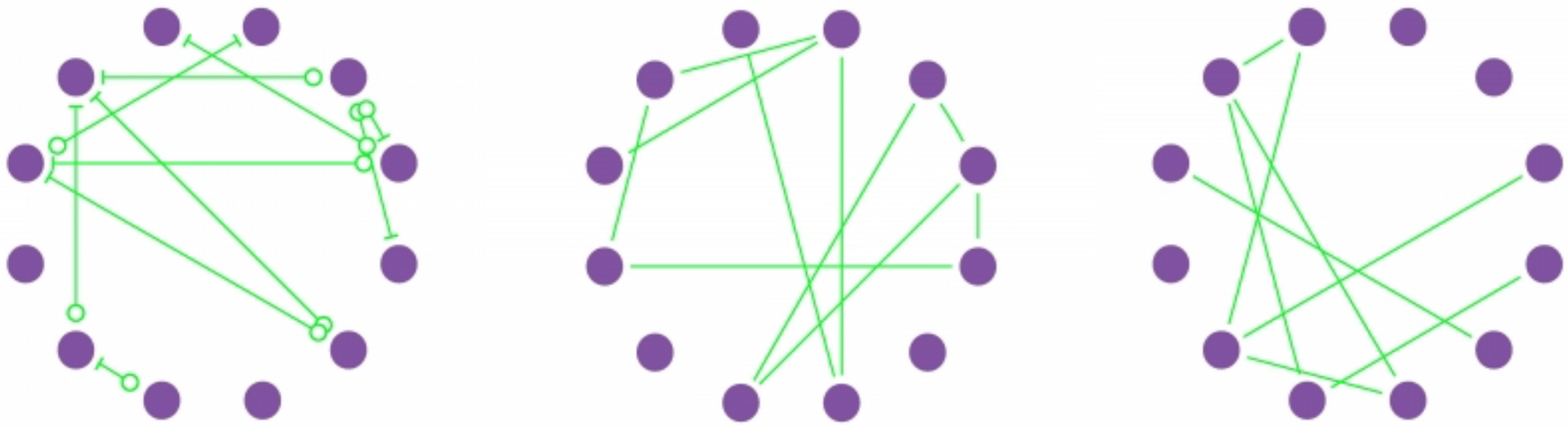
- Gilbert's $G(N, p)$ model: a random network model is a network where each pair of nodes is connected with probability *p*
  - *Example:* $p = 1/6$

    $N = 10$

    $\langle D \rangle \sim 1.5$

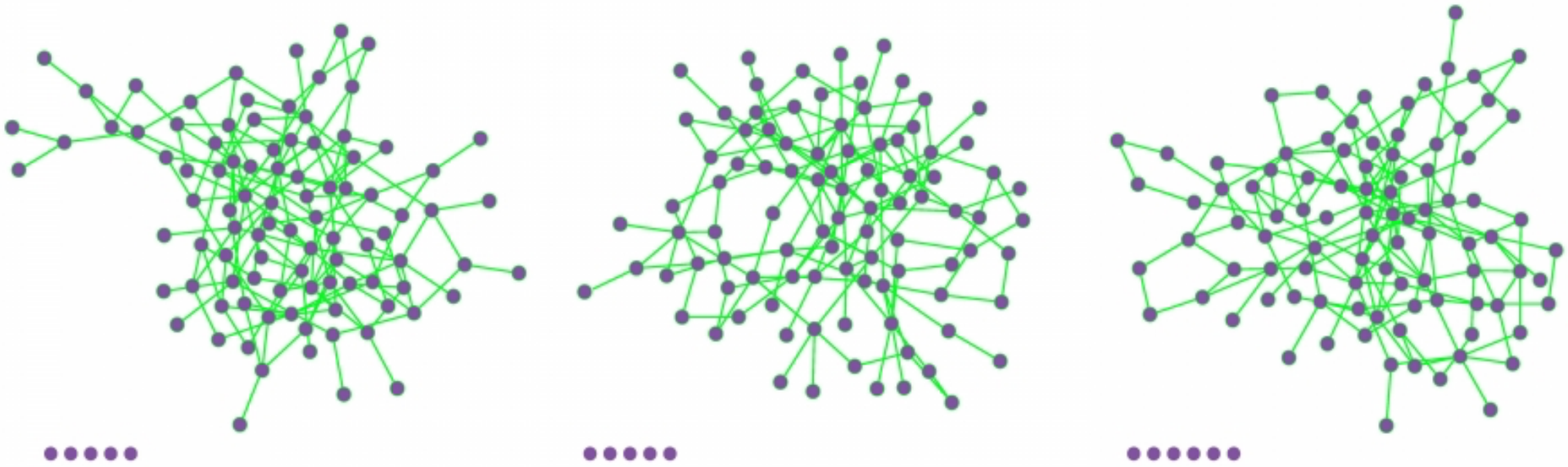


Random network examples



$p = 1/6$

$N = 12$

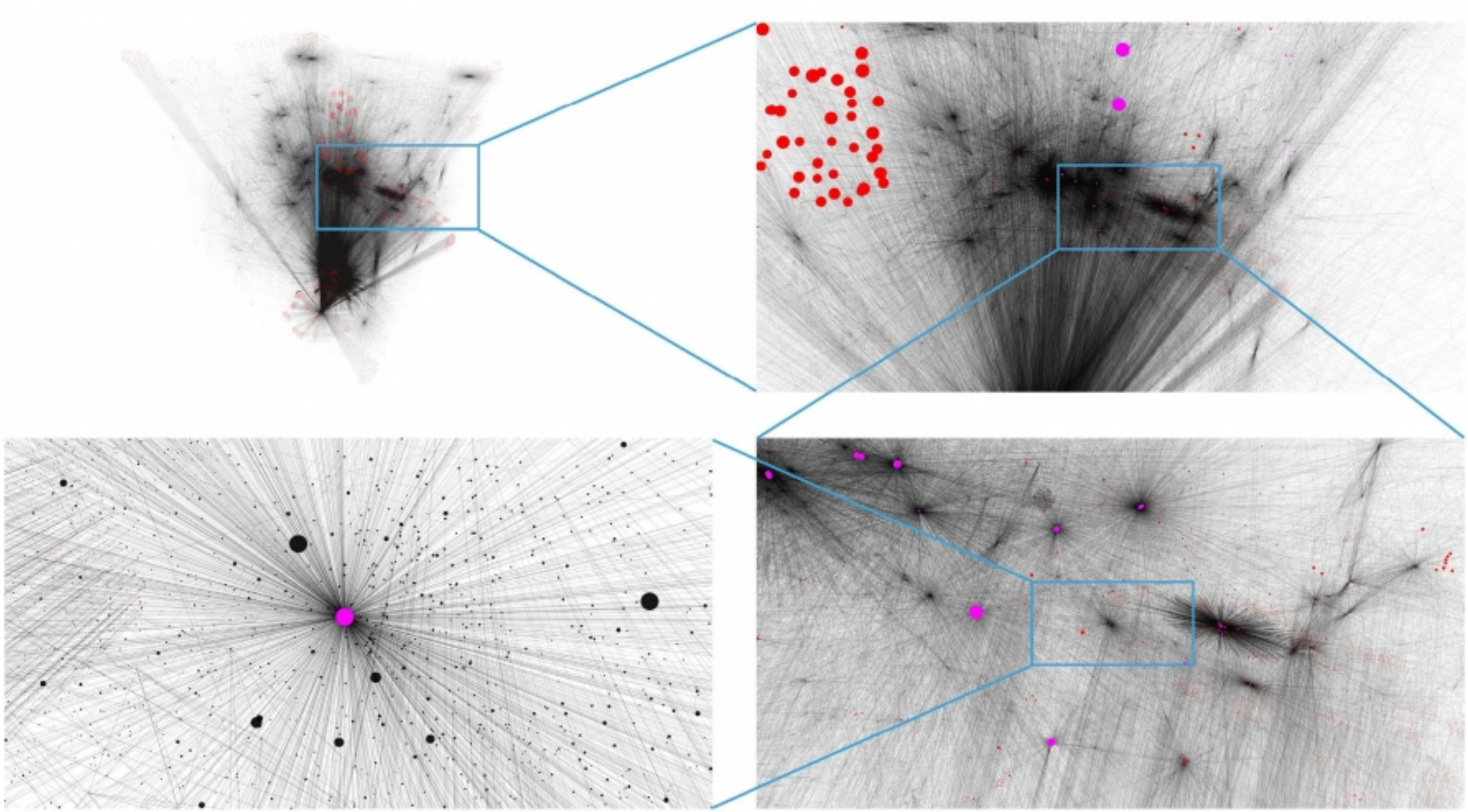


$p = 0.03$

$N = 100$

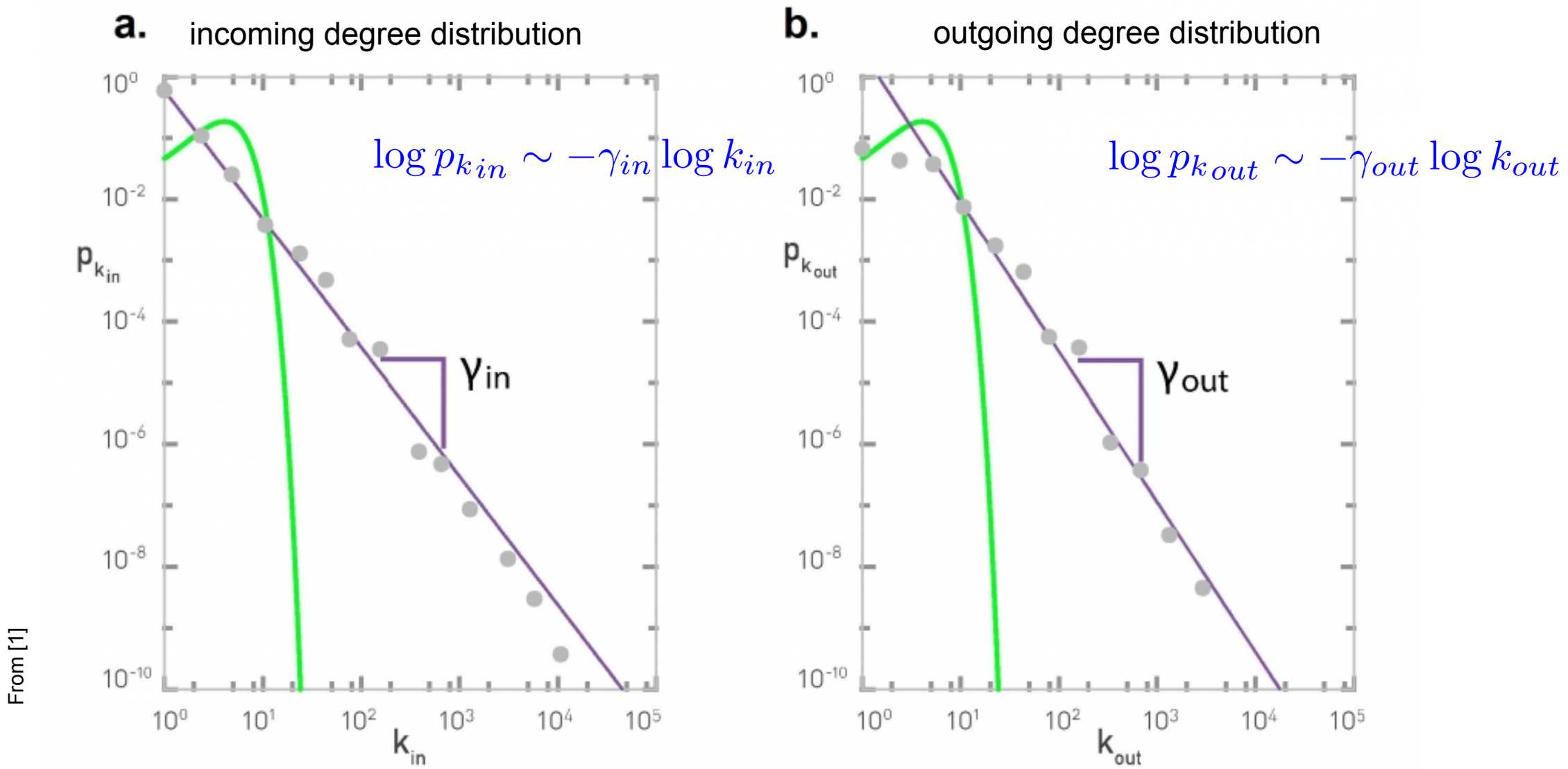
From [1]

The Web is not random



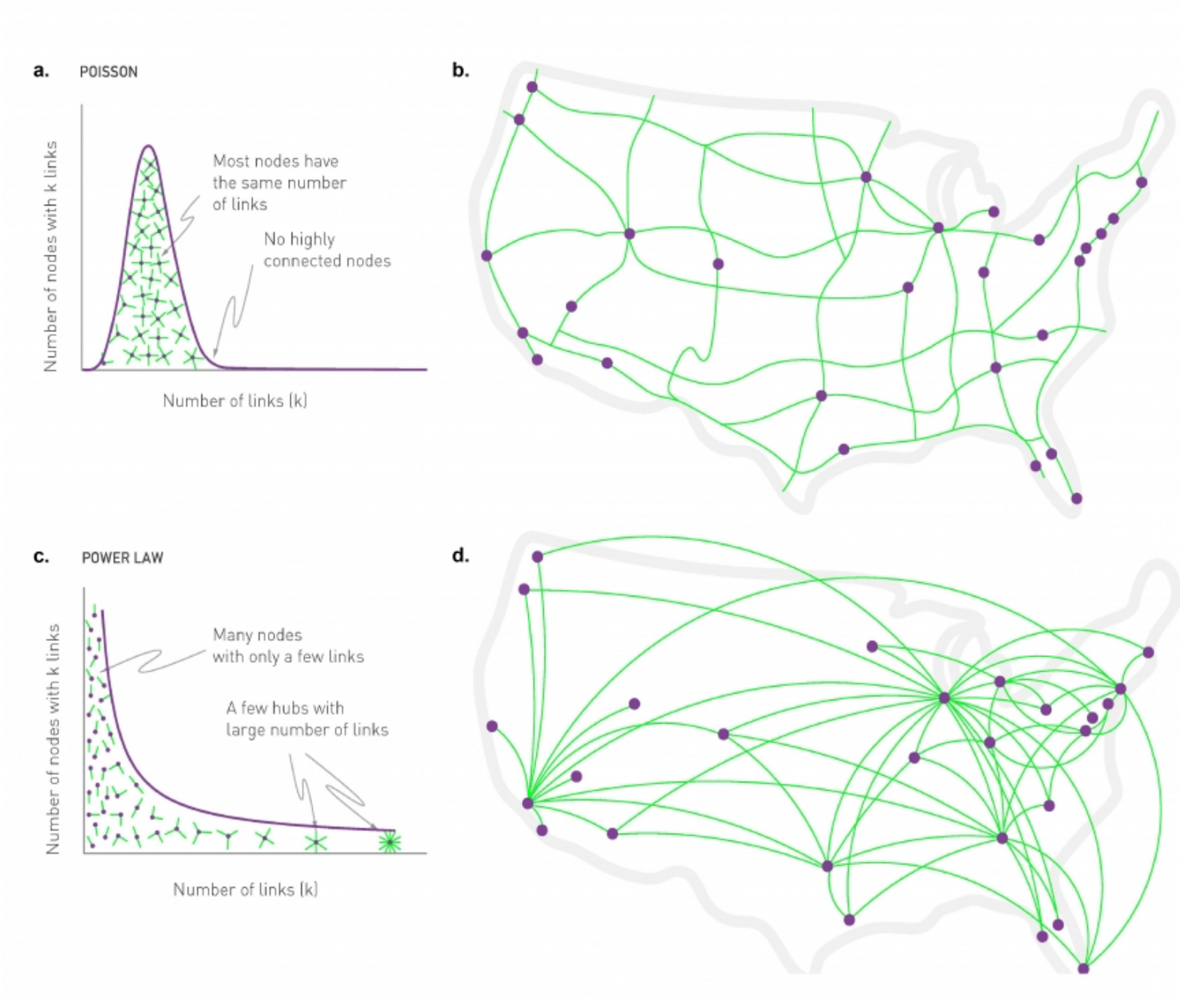
From [1]

WWW: degree distribution



- Degrees do not follow a Poisson distribution (like random networks) but rather a *power law distribution*, of the form $p_k \sim k^{-\gamma}$

Random vs Scale-free networks



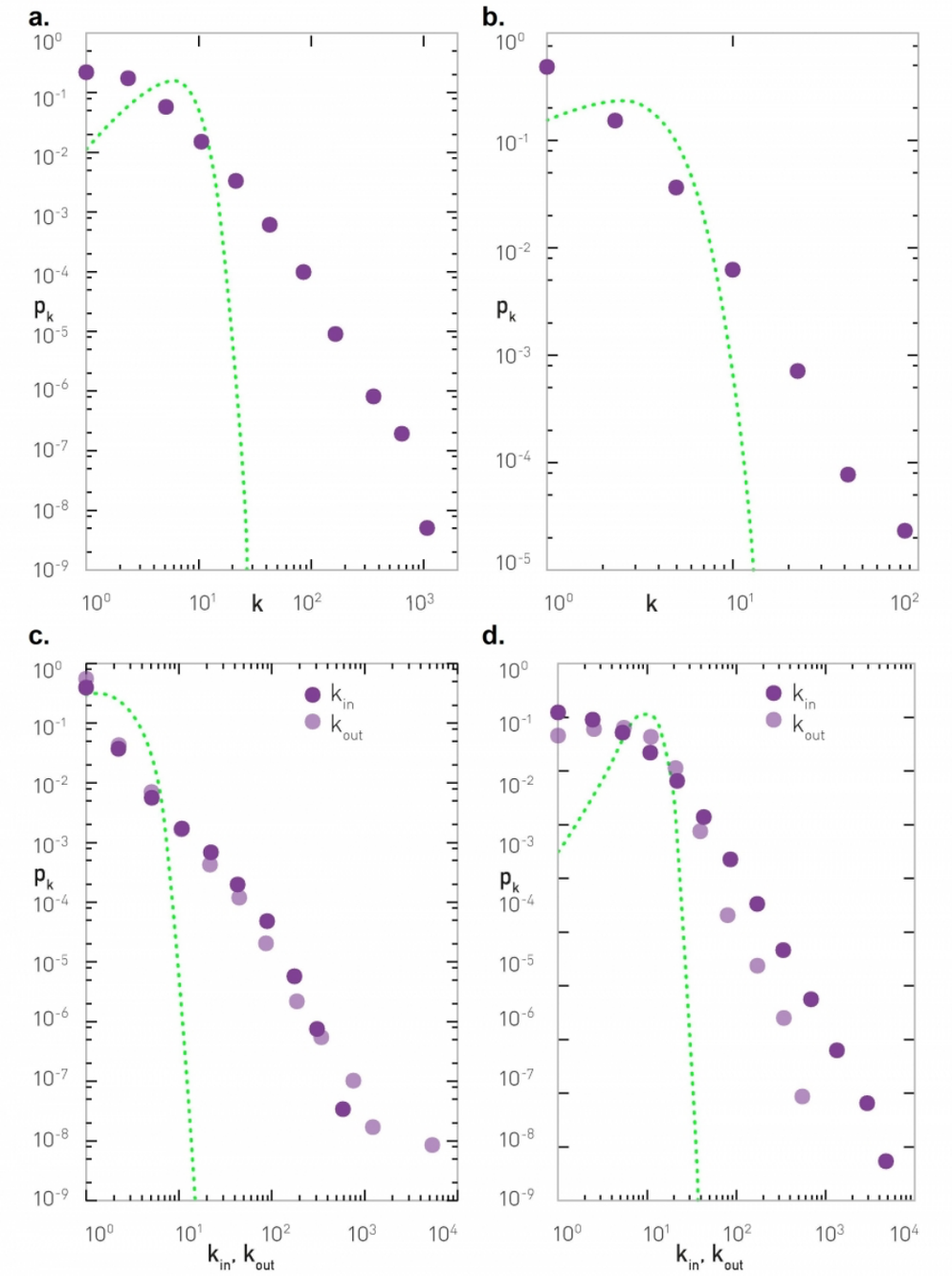
Highway network

Airline network

From [1]

Universality of scale-free properties

Internet at the router level



Protein–protein interaction network

Email network

Citation network

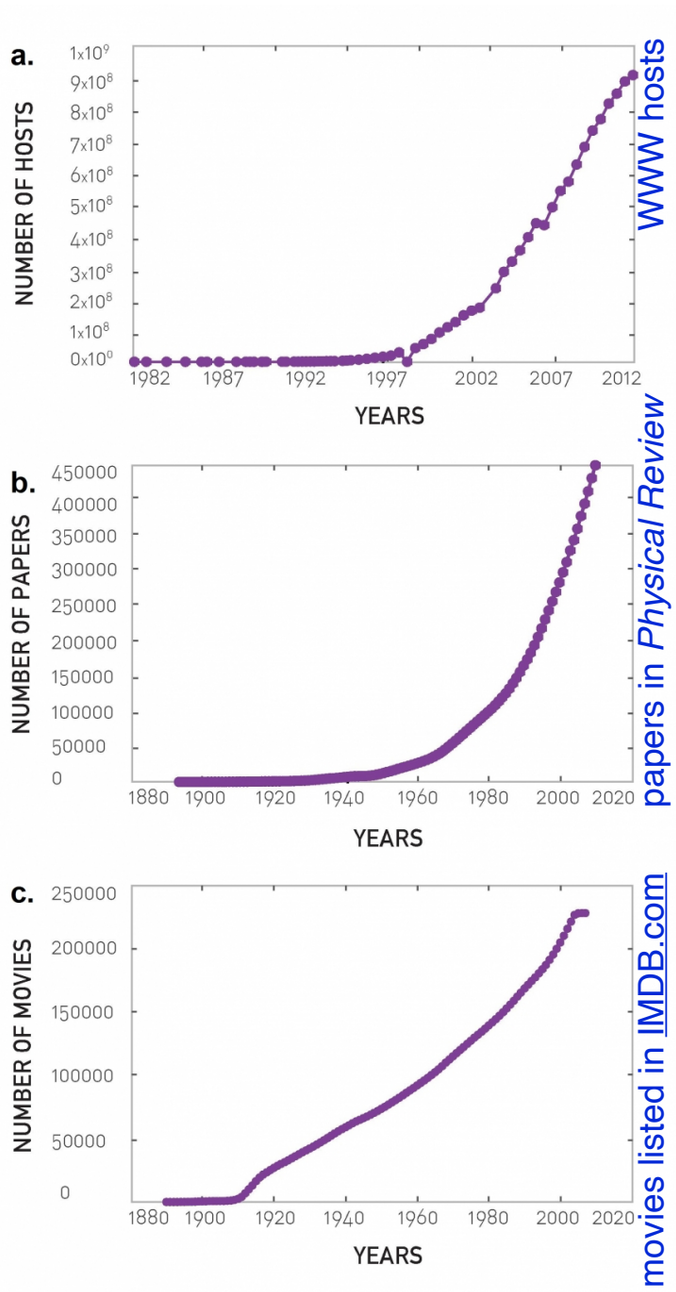
From [1]

Formation of Scale-Free Networks

- Many networks seem to have the same properties, but they capture very different data
  - WWW and biological cellular networks are both scale-free, while they are very different

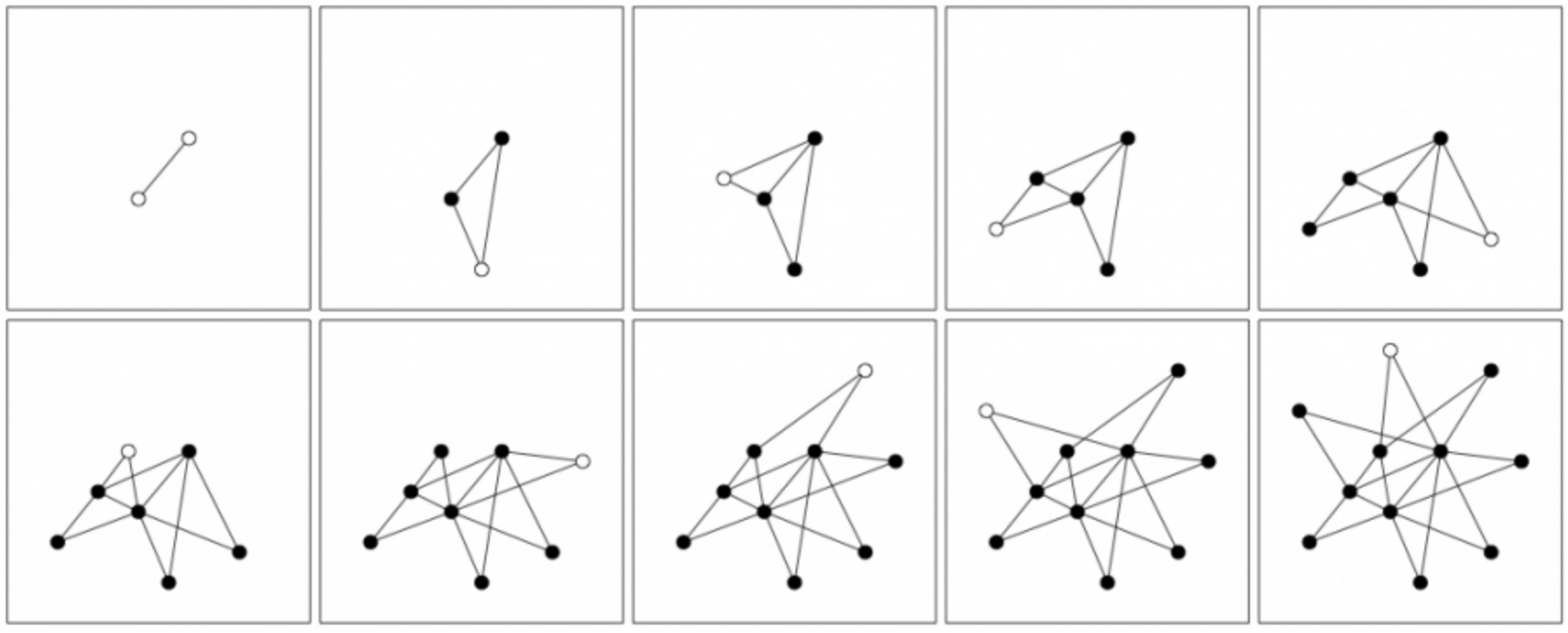
- It is important to understand how networks get formed
  - models explaining the properties of networks

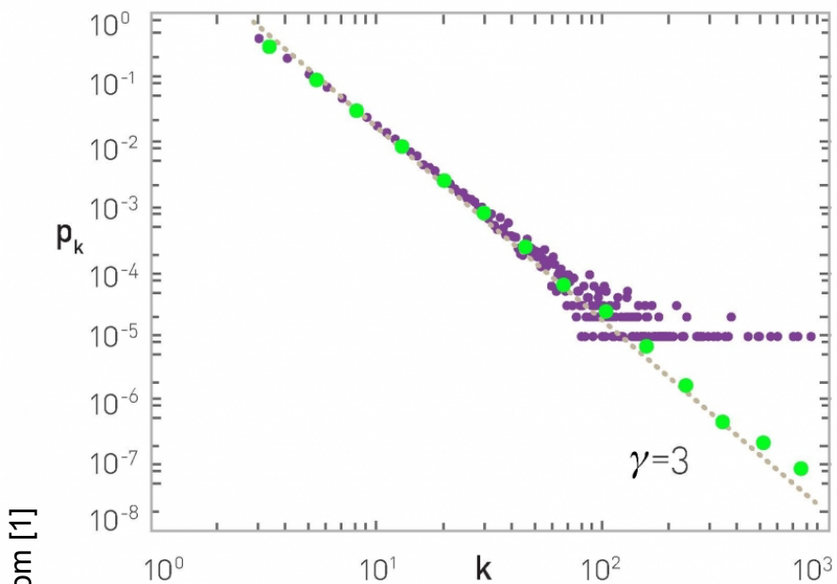
- Growth and preferential attachment lead to properties similar to the ones of real networks
  - the degree distribution of real networks is quite different from the random network one
  - Barabási-Albert model, and other models



The Barabási-Albert Model

- The BA model generates scale-free networks
  - start with m_0 nodes, and choose links arbitrarily (with at least one link per node)
  - then develop the networks with growth and preferential attachment
    - Growth: add a new node with $m \leq m_0$ links that connects to *m* nodes already in the network
    - Preferential attachment: probability that the new node connects to node *i* depends on $\Pi(k_i) = \frac{k_i}{\sum_j k_j}$
  - after *t* steps, the network has $N = t + m_0$ nodes and $mt + m_0$ links, and a power-law distribution





From [1]

$N = 100000 \quad m = 3$

Summary of network models

- Network models are useful to understand or generate data
  - Simple and tractable models (often do not describe exactly real data)
  - They are based on *assumptions / heuristics* oversimplifying the underlying distributions of graphs

- Network models can help calculate many quantities and properties
  - Those can be compared to the real data
  - They can help develop insights about real data

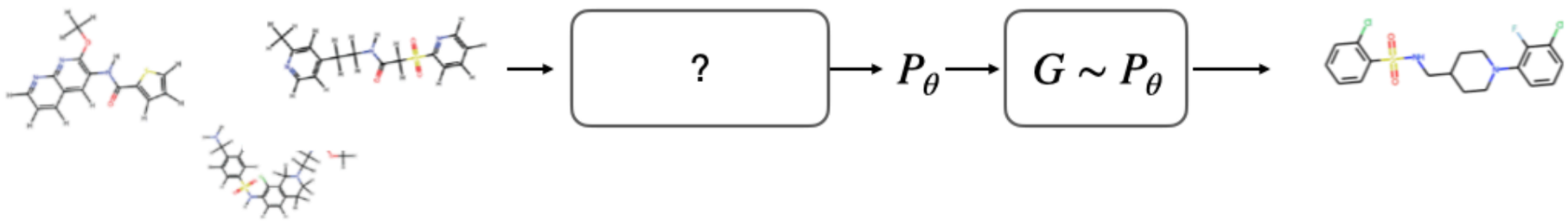
- In order to identify these properties, we need to understand how a network would look like if it is driven entirely by a model

- In particular, the random network model is a sort of benchmark model

Limited capacity to model complex dependencies

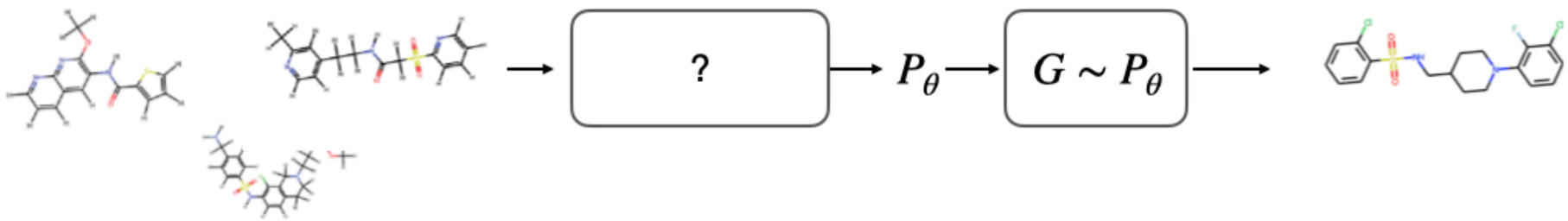
Graph Generative Models

- Given the observation $\mathscr{D} = \{G_i\}_i$, with $G_i \sim P_{data}$, we aim at learning the distribution of the observed set of graphs $P_\theta(G)$ such that sampled graphs looks like the ones in the dataset [unconditional generation]



Graph Generative Models

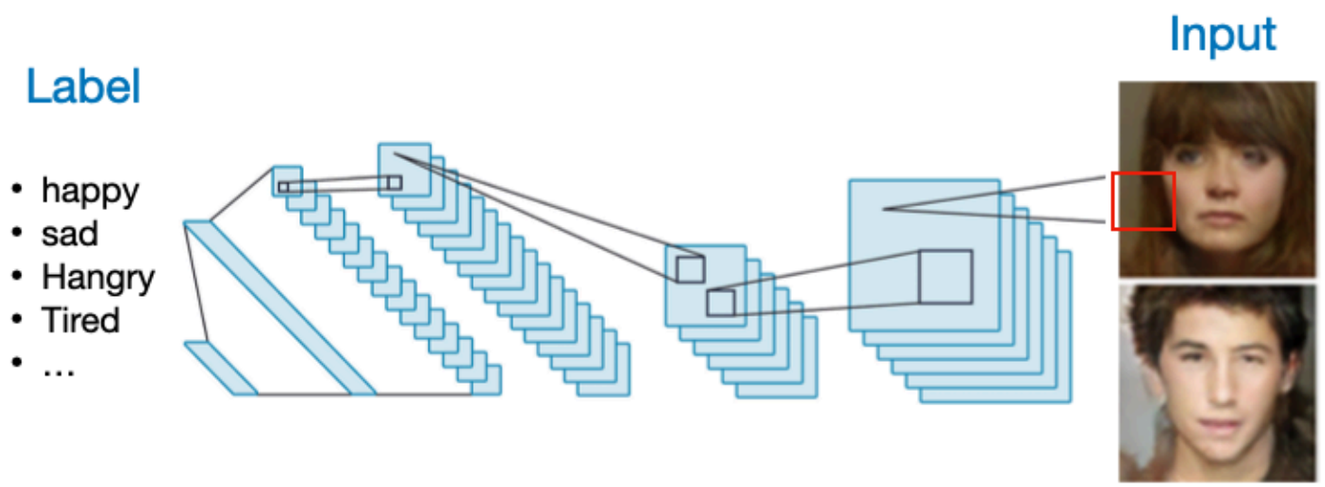
- Given the observation $\mathscr{D} = \{G_i\}_i$, with $G_i \sim P_{data}$, we aim at learning the distribution of the observed set of graphs $P_\theta(G)$ such that sampled graphs looks like the ones in the dataset [unconditional generation]



How do we learn the distribution?

What are the main challenges when graphs is the data modality?

(Probabilistic Deep) Generative Task

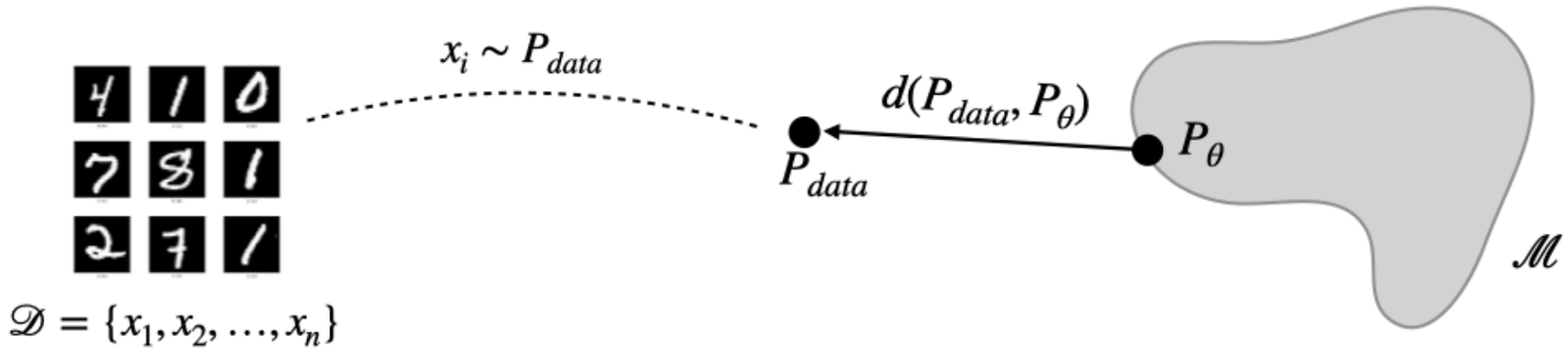


- **Generation:** Focuses on modelling the joint distribution $p(x, y)$
  - Representing $p(x, y)$ is usually intractable
  - We can impose structure on the data (e.g., conditional independence)

- A good generative model P_θ can improve downstream inference

Which is the right hypothesis/dependency to impose?
We learn it from the data!

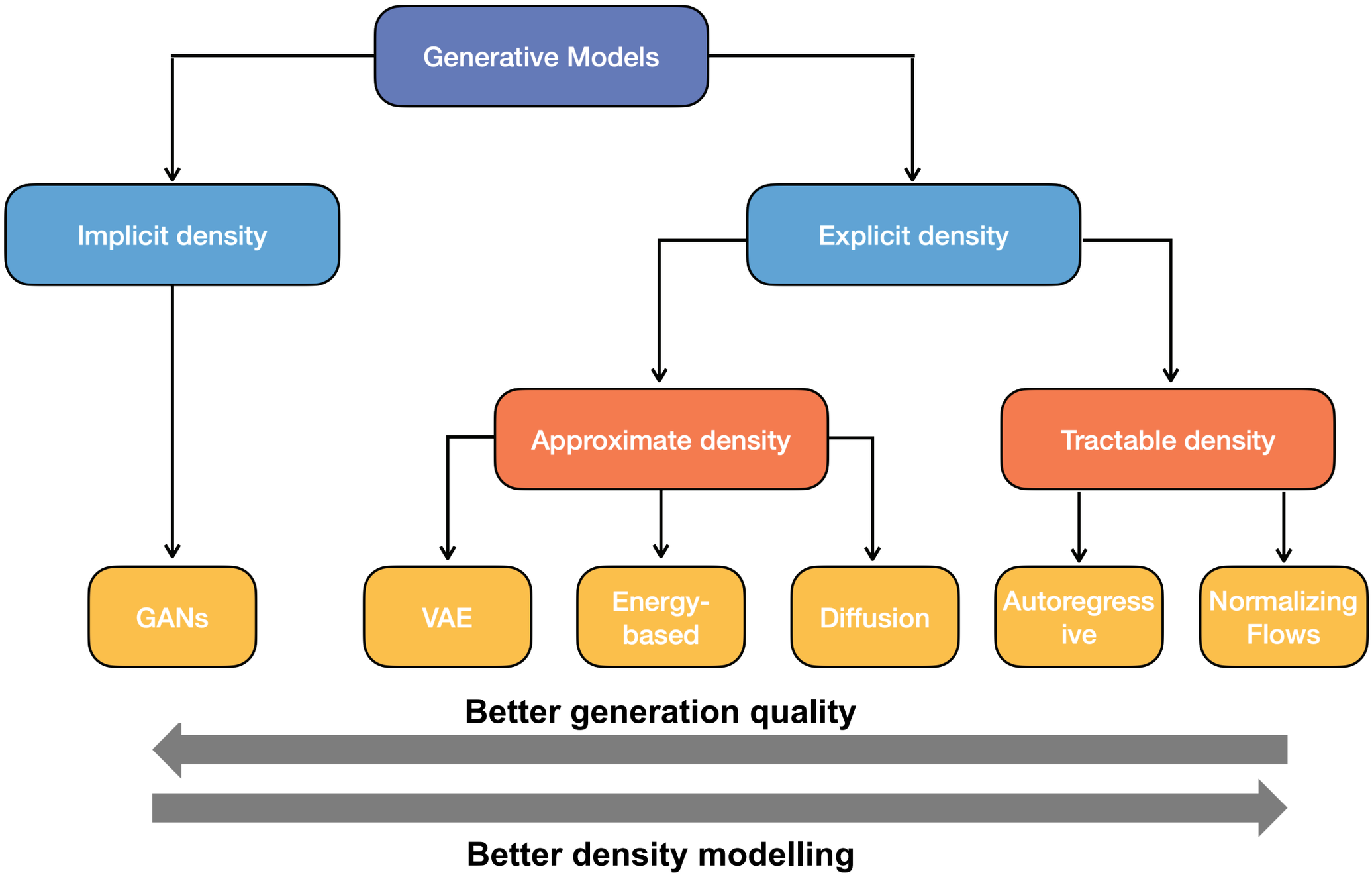
The task of generative modelling



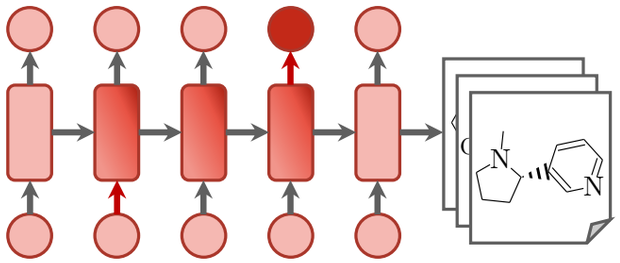
- **Hypothesis:** $\mathscr{D}$ is an observed finite set of samples from an underlying distribution P_{data}
- **Goal:** Approximate this data distribution from $\mathscr{D}$
- **How:** We learn P_θ by maximum likelihood estimation $\theta^\star = \arg\max_{P_\theta} \mathbb{E}_{x \sim P_{data}} \left[\log P_\theta(x)\right]$

$$\begin{aligned} \mathbf{D}(P_{\text{data}} || P_\theta) &= \mathbf{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log \left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}\right)\right] \\ &= \mathbf{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log P_{\text{data}}(\mathbf{x})\right] - \mathbf{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log P_\theta(\mathbf{x})\right] \end{aligned}$$

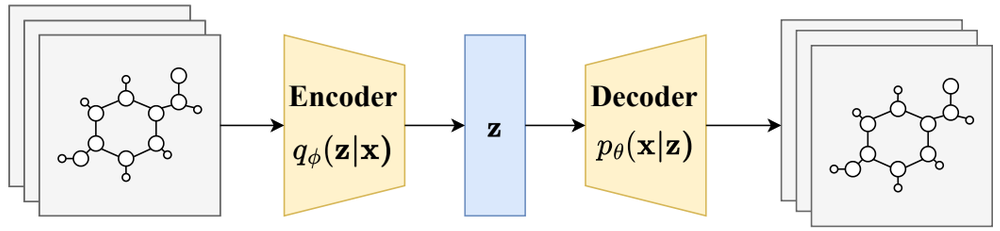
Types of generative models



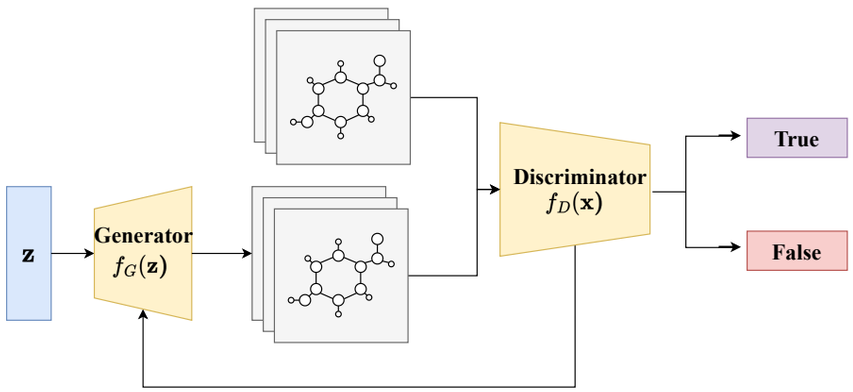
Overview (I)



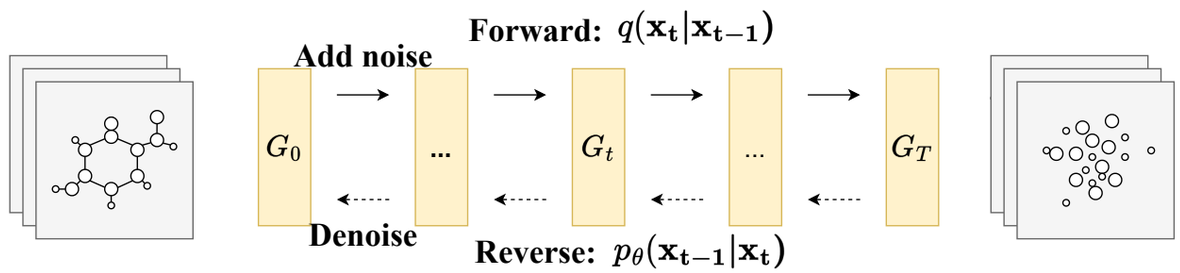
Auto-regressive models



Variational AutoEncoders



Generative Adversarial Networks (GAN)



Diffusion models

Overview (II)

- **Implicit models** (GANs) generate high-quality, realistic samples but are hard to train and prone to mode collapse

- **Tractable models** (Autoregressive) offer exact likelihoods and interpretability but suffer from slow sampling due to their sequential nature; Errors are accumulated over iterations

- **Approximate models** (VAEs) enable stable training via a likelihood lower bound but produce blurrier samples

- **Diffusion models** combine benefits of others by transforming noise into data through a reverse process, achieving high fidelity but at the cost of slower sampling than VAEs and GANs; faster than autoregressive models

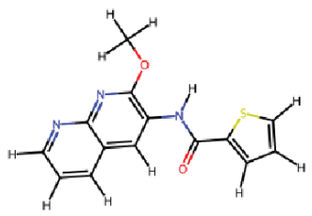
What's New with Graphs?

- **Non-Unique Representations:** A graph with n nodes can be represented by up to $n!$ equivalent adjacency matrices

Can we use generative models for Euclidean data?

Non-Unique Representation

- **Non-Unique Representations:** A graph with n nodes can be represented by up to $n!$ equivalent adjacency matrices



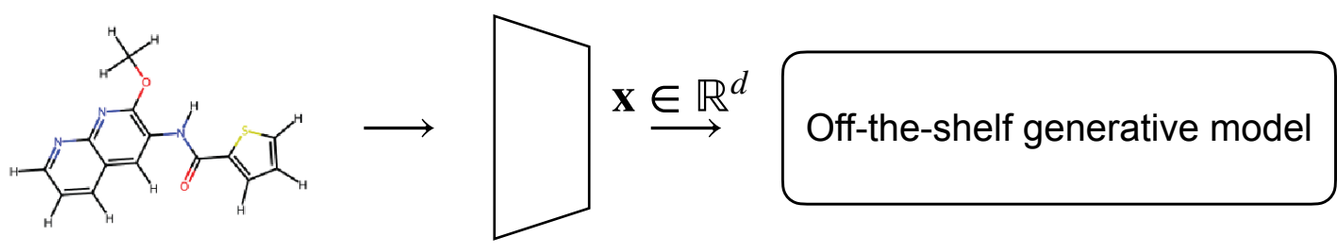
$$G \rightarrow A \rightarrow vec(A) \longrightarrow$$

Off-the-shelf generative model

- Existing methods cannot naturally generalize to graphs of varying size
  - training on all possible node permutations or specifying a canonical permutation is required, both of which require $\mathcal{O}(n!)$ time

Varying Size Input

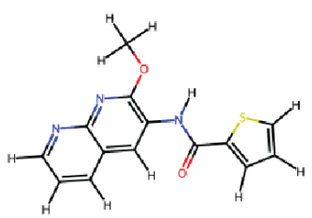
- Rather than vectoring the adjacency matrix:
  - Learn graph embeddings as compact representations
  - Use these embeddings as input of classical generative models



- Key limitations:
  - Still constrained to a single input graph and a fixed number of nodes

Topological Information

$G = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$



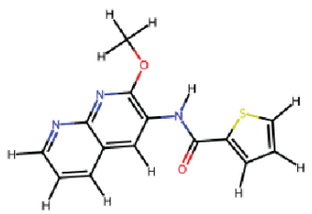
→ **generative model**

- $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $\mathbf{E} \in \mathbb{R}^{N \times N \times F}$ are the node features matrix and the edge attributes tensor, respectively

- Information relies on both data features and topology
- Complex Dependencies: Edges and nodes cannot be treated independently
- Large Output Spaces: To generate a graph with n nodes the generative model may have to output $\mathcal{O}(n^2)$ values to specify its structure
- Discrete Objects by Nature: Not differentiable

Topological Information

$G = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E})$



→ **generative model**

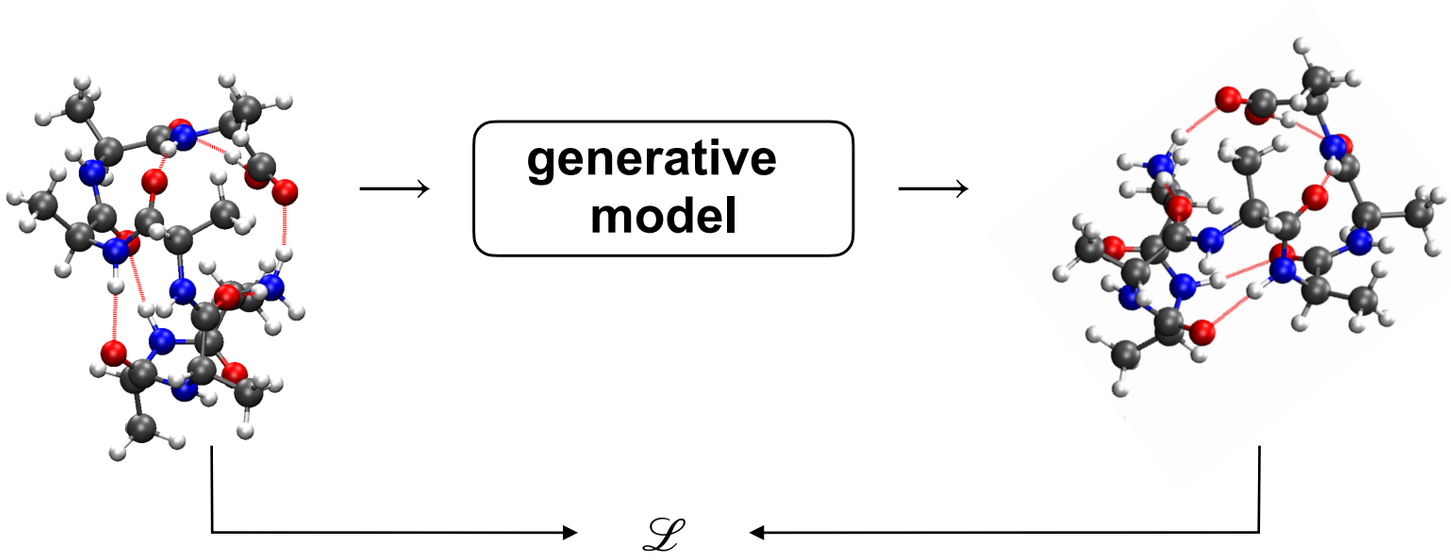
- Should we learn joint distribution of G?
- Should we rather learn the joint distribution of X and E independently?
- Should we treat them as categorical or continuous data?

- $\mathbf{X} \in \mathbb{R}^{N \times D}$ and $\mathbf{E} \in \mathbb{R}^{N \times N \times F}$ are the node features matrix and the edge attributes tensor, respectively

- Information relies on both data features and topology
- Complex Dependencies: Edges and nodes cannot be treated independently
- Large Output Spaces: To generate a graph with n nodes the generative model may have to output $\mathcal{O}(n^2)$ values to specify its structure
- Discrete Objects by Nature: Not differentiable

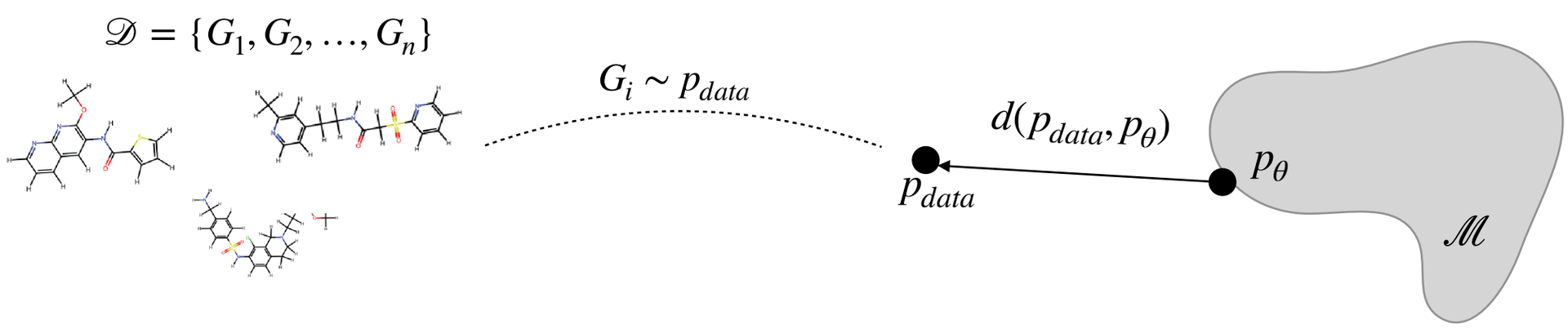
Evaluating Similarity

- Exposing the model to $n!$ permutations is infeasible
- Pre-defining an order is computationally expensive
  - Only practical in constrained domains (e.g., molecules via SMILES)



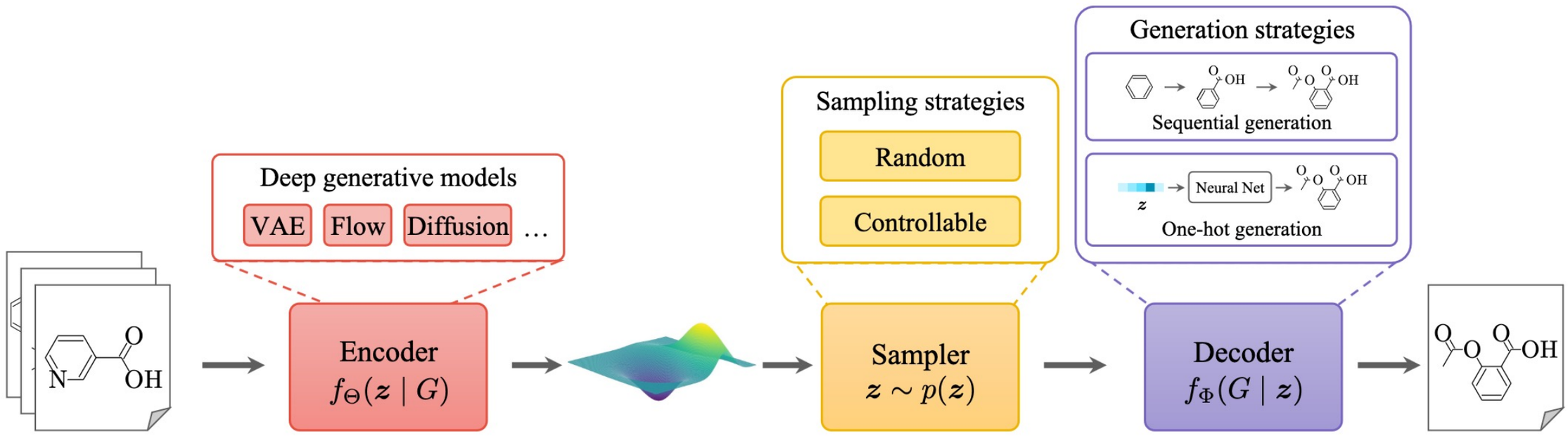
Loss function needs to be permutation equivariant!

Graph Generative Models



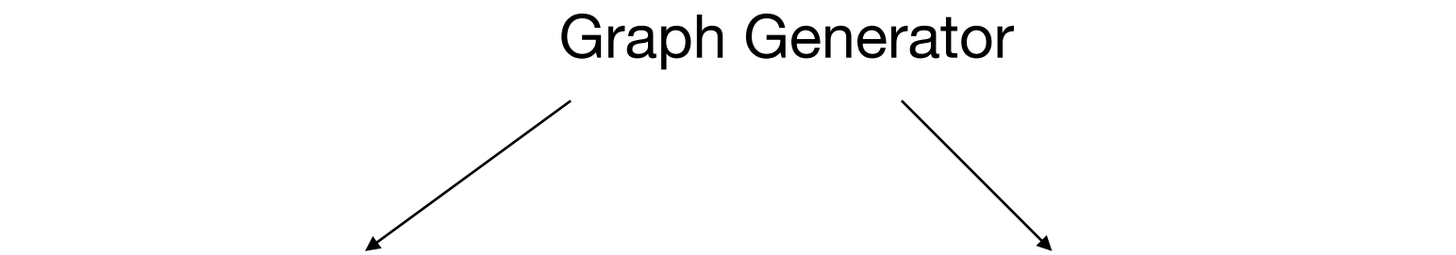
- Given the observation $\mathscr{D} = \{G_i\}_i$, with $G_i \sim p_{data}$, we aim at
  - learning the distribution of the observed set of graphs $p_\theta(G)$ such that sampled graphs looks like the ones in the dataset [unconditional generation]
  - learning the distribution of the observed set of graphs $P_\theta(G \mid y)$ such that sampled graphs looks like the ones in the dataset and conditioned to some prior information y [conditional generation]

A unifying view of graph generation



[Zhou22]

How to Decode/Generate Graphs?



One Shot Generation:

builds a probabilistic graph model based on the matrix representation that generates all nodes and edges in one shot

Sequential Generation

builds the nodes and edges in a sequential way, one after another

One shot generation

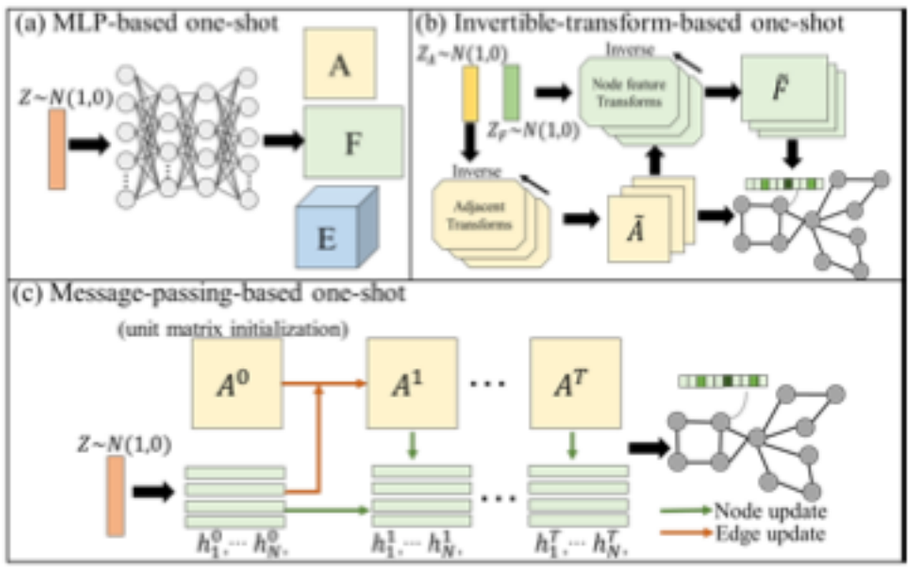
Graph Generator

One Shot Generation

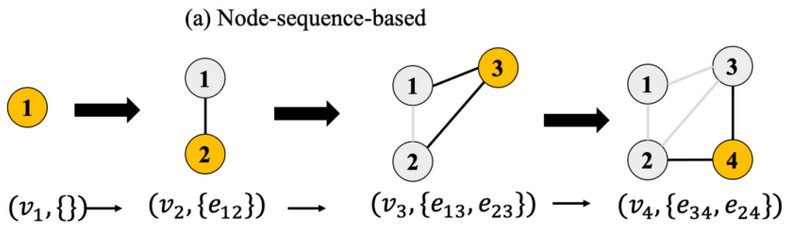
Edge based

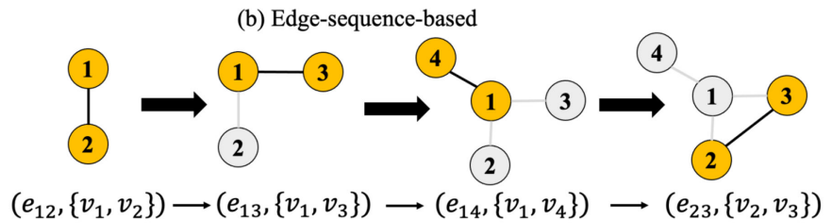
Matrix based

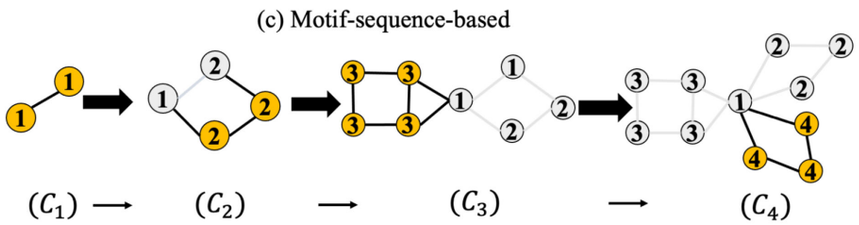
It requires a generative model that predicts edges independently; edge probabilities are generated with techniques such as random-walk or node similarity based

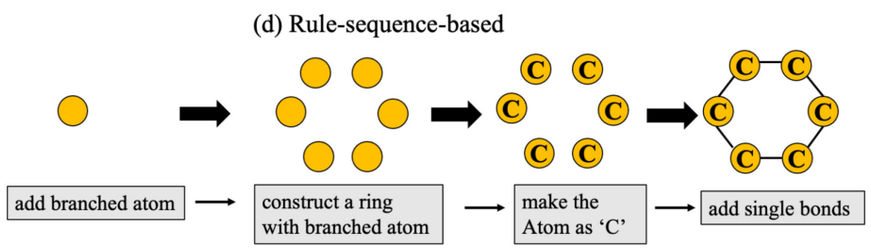


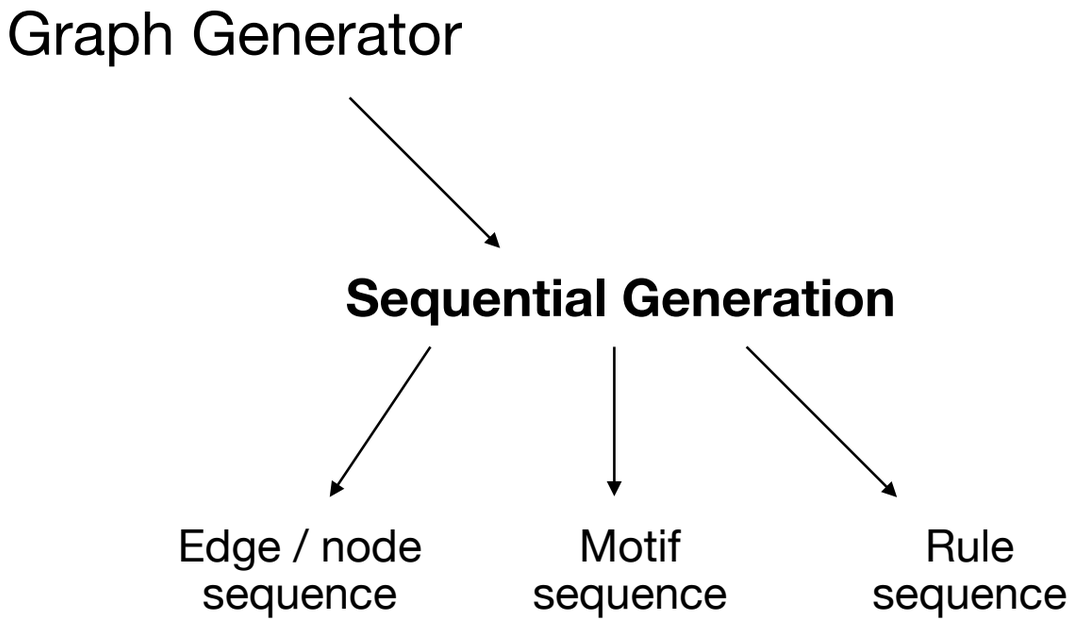
Sequential generation







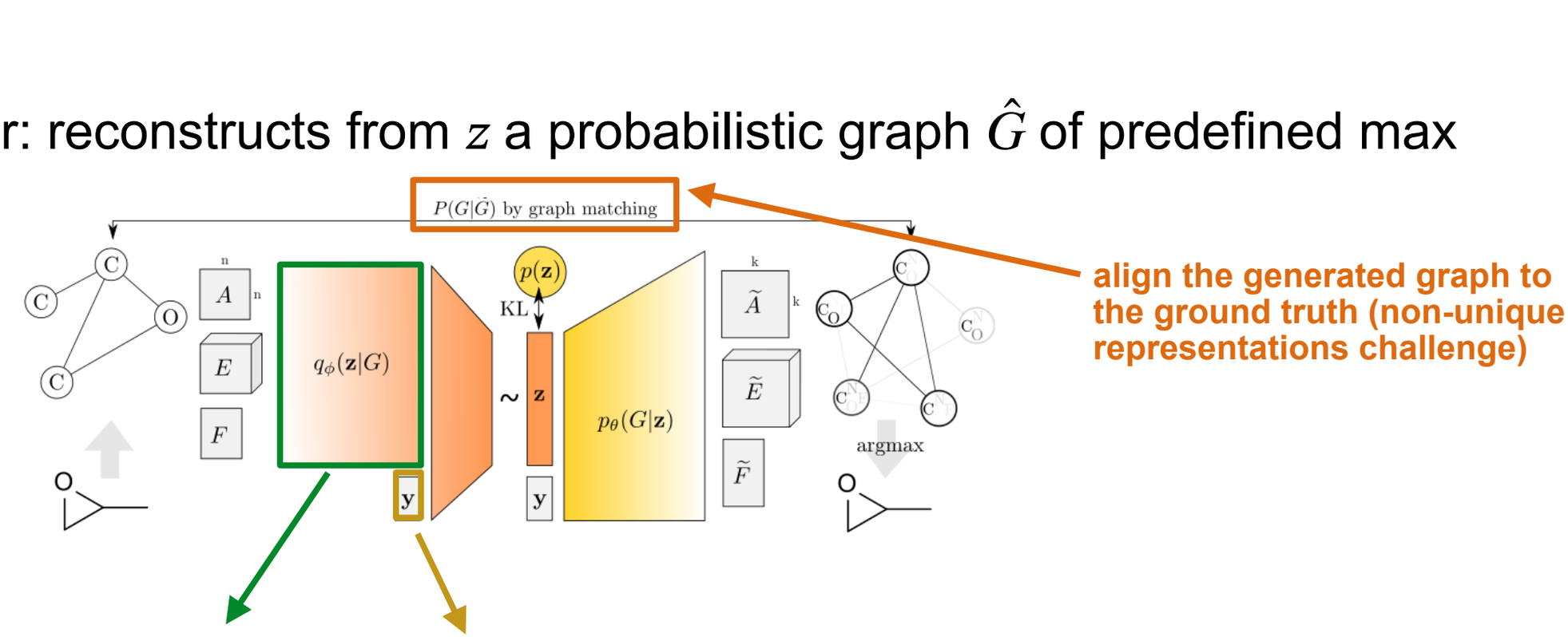




https://arxiv.org/abs/2007.06686

Graph VAE

- Circumvent non-differentiability problem through loss on constructed probabilistic graph
- Variational Autoencoder:
  - Encoder $q_\theta(z \mid G)$: embeds graph $G = (A, E, F)$ into continuous latent space z
  - Decoder: reconstructs from z a probabilistic graph G of predefined max size



[Simonovsky'18]

Limitations of graph VAEs

- Desirable: Different node orderings of the same graph should be mapped to the same latent space

- This implies solving graph isomorphism (NP-hard)

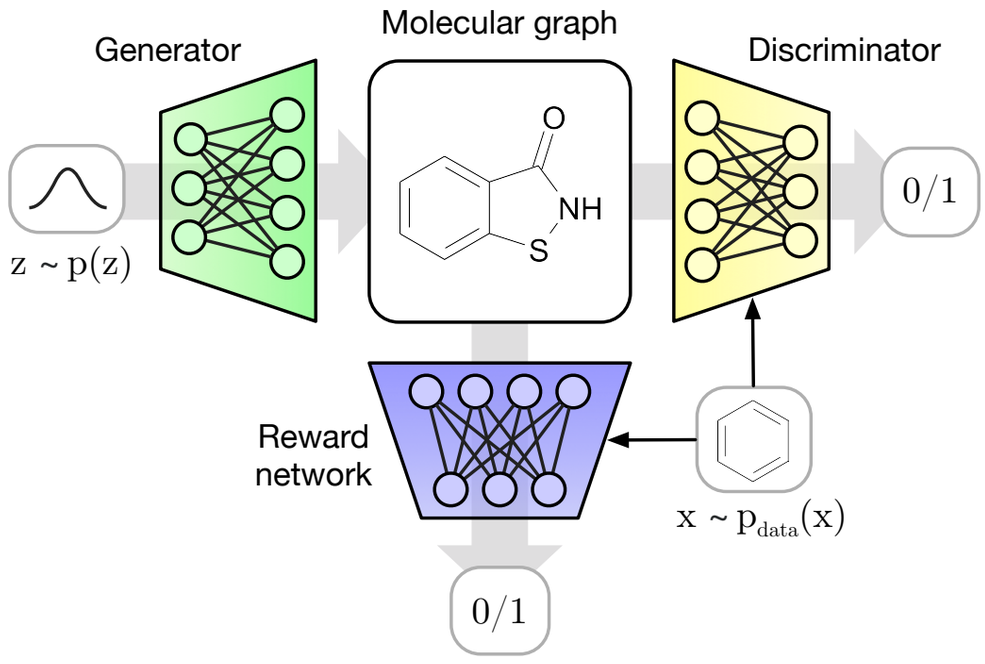
  - VAEs are only feasible in constrained domains (e.g. molecules)
  - Typically small graphs with ~40 nodes

- The max size of the graphs must be predefined

- Graph matching is required

GANs: MolGAN

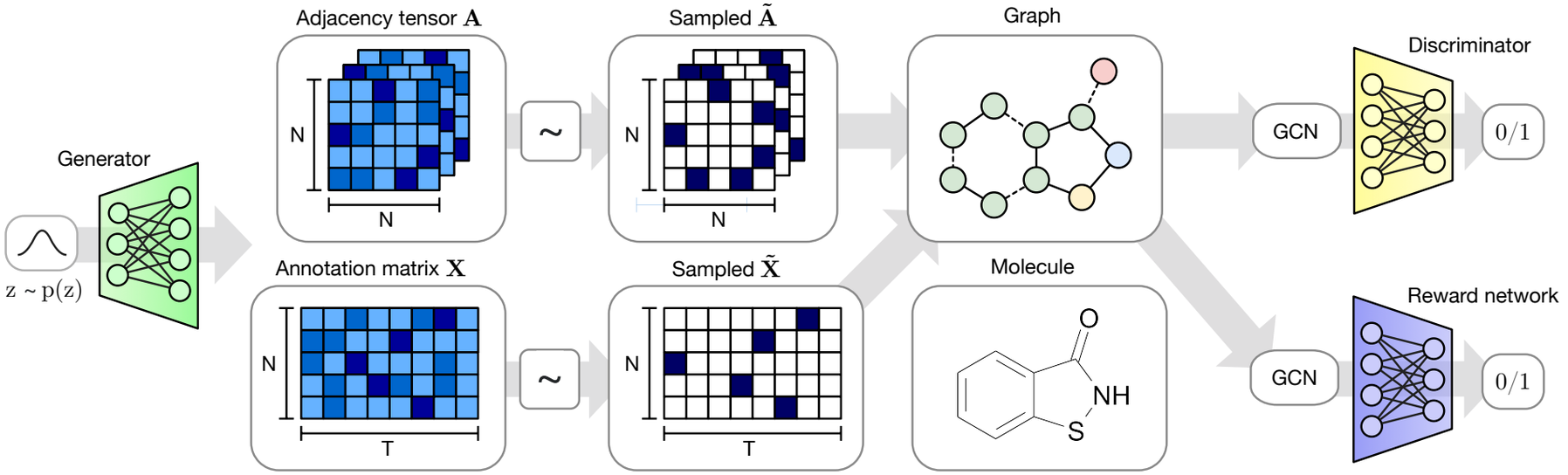
- An implicit, likelihood-free generative model
- Directly generates graphs via learned node and edge likelihoods



- Reward discriminator evaluates graph properties
- Combined with reinforcement learning (loss function)
  - Encourages generation of molecules with desirable properties
- Permutation equivariance is achieved using graph convolution in the discriminator

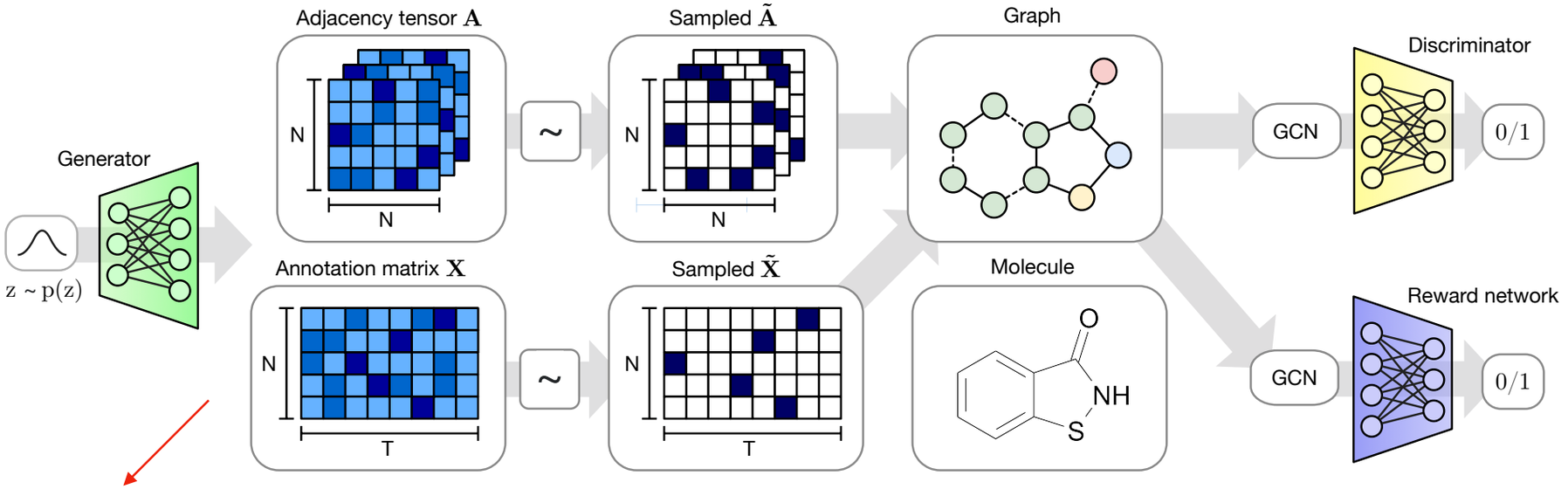
[Cao'18]

GANs: MolGAN in details



[MolGAN, De Cao et al. 2018]

GANs: MolGAN in details

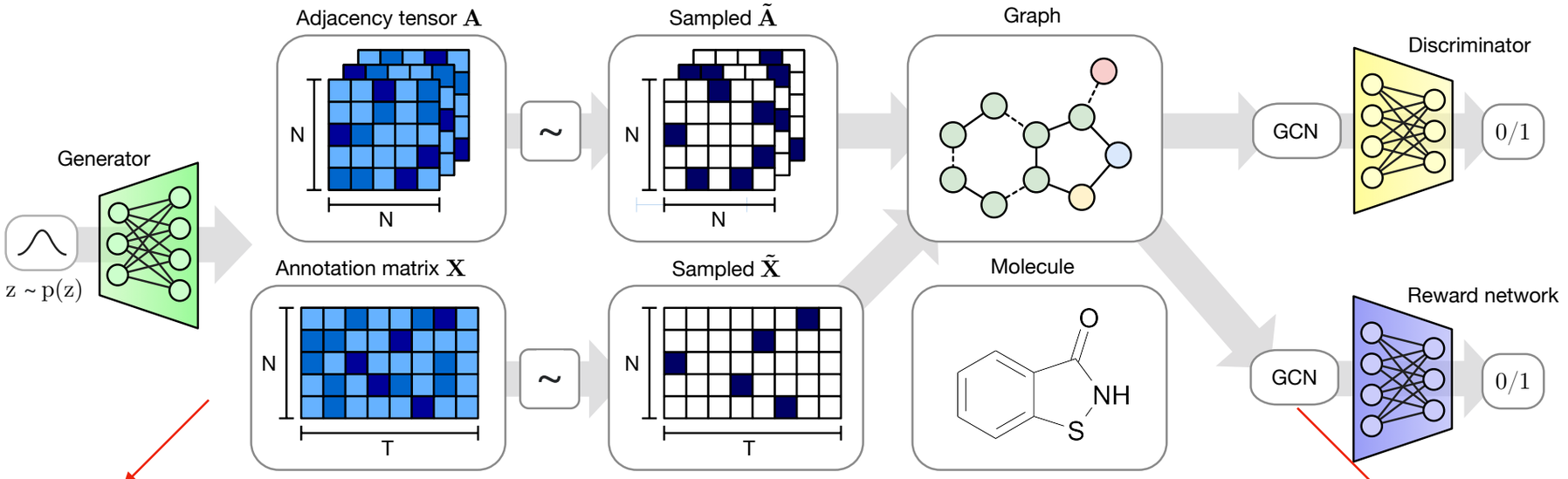


One shot generation

- faster and easier to optimize than sequential generation
- Limited to graphs of a pre-chosen maximum size

[MolGAN, De Cao et al. 2018]

GANs: MolGAN in details



One shot generation

- faster and easier to optimize than sequential generation
- Limited to graphs of a pre-chosen maximum size

Graph convolution layers convolve the node signals using the graph adjacency tensor

[MolGAN, De Cao et al. 2018]

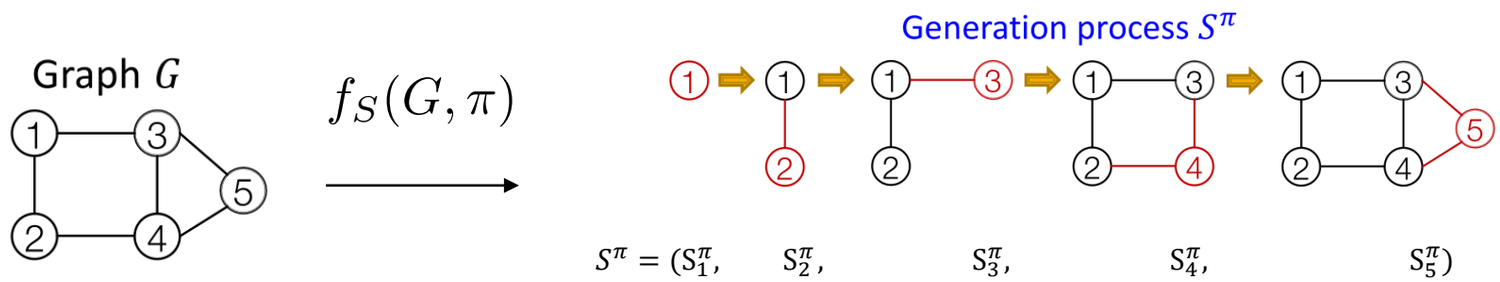
Practical considerations in graph GANs

- ✓ No graph matching is required
- ✓ Higher validity and novelty than VAEs

- Predefined max graph size is needed
- Few graph GAN models exist
  - Mainly due to the complexity of designing effective generators
- Expressivity challenges:
  - One-shot generators struggle to capture global graph properties
  - Issues more pronounced in large graphs
  - Results in training instability and non-convergence

Autoregressive models

- **Sequential generation**: Graph generation process decomposed into a sequence of node and edge formations, conditioned on the graph structure previously generated (AR)
- Example: GraphRNN accommodates variable-sized graphs
- Instead of learning (and sampling from) $p_{data}(G)$, it learns $p(S^{\pi})$, modelled auto-regressively



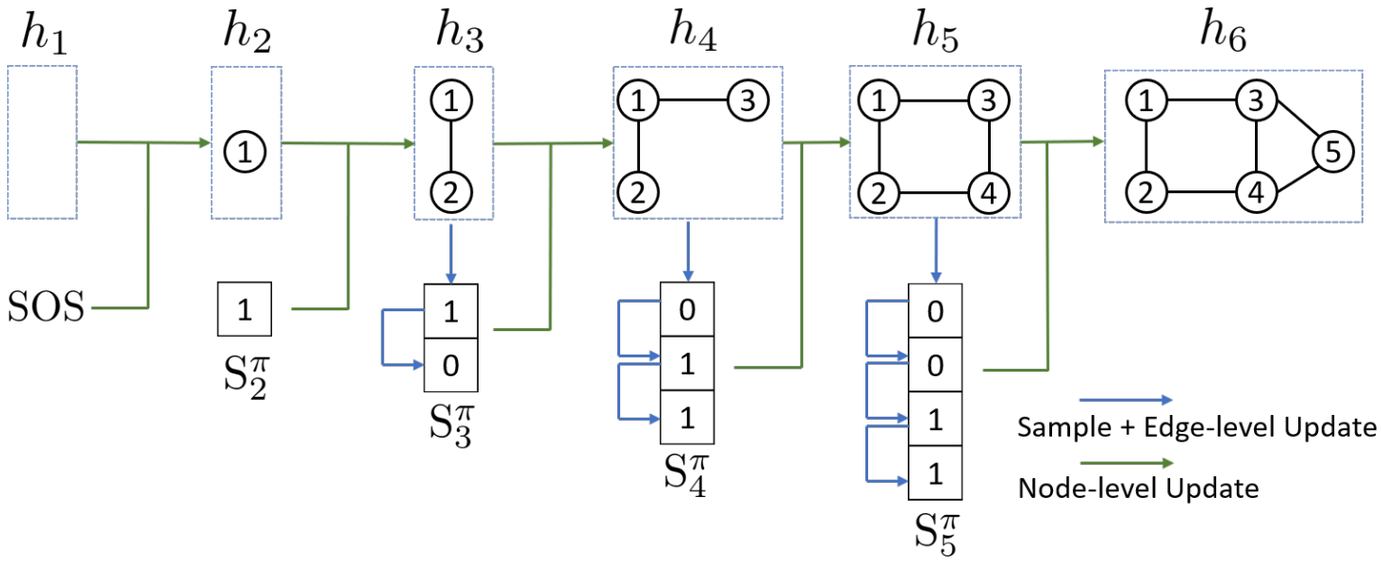
$$p(G) = \sum p(S^{\pi})\ \mathbb{1}[f_G(S^{\pi}) = G]$$

$$p(S^{\pi}) = \prod_{i=1}^{n+1} p(S_i^{\pi} | S_1^{\pi}, ..., S_{i-1}^{\pi})$$

Modeled with RNNs

GraphRNN

- Need to model two processes (hierarchical model):
  - GraphRNN has two RNNs: graph-level RNN and edge-level RNN
- Relationship between two RNNs:
  - Graph-level RNN maintains the state of the graph and generates new nodes
  - Edge-level RNN generates the edges for each newly generated node

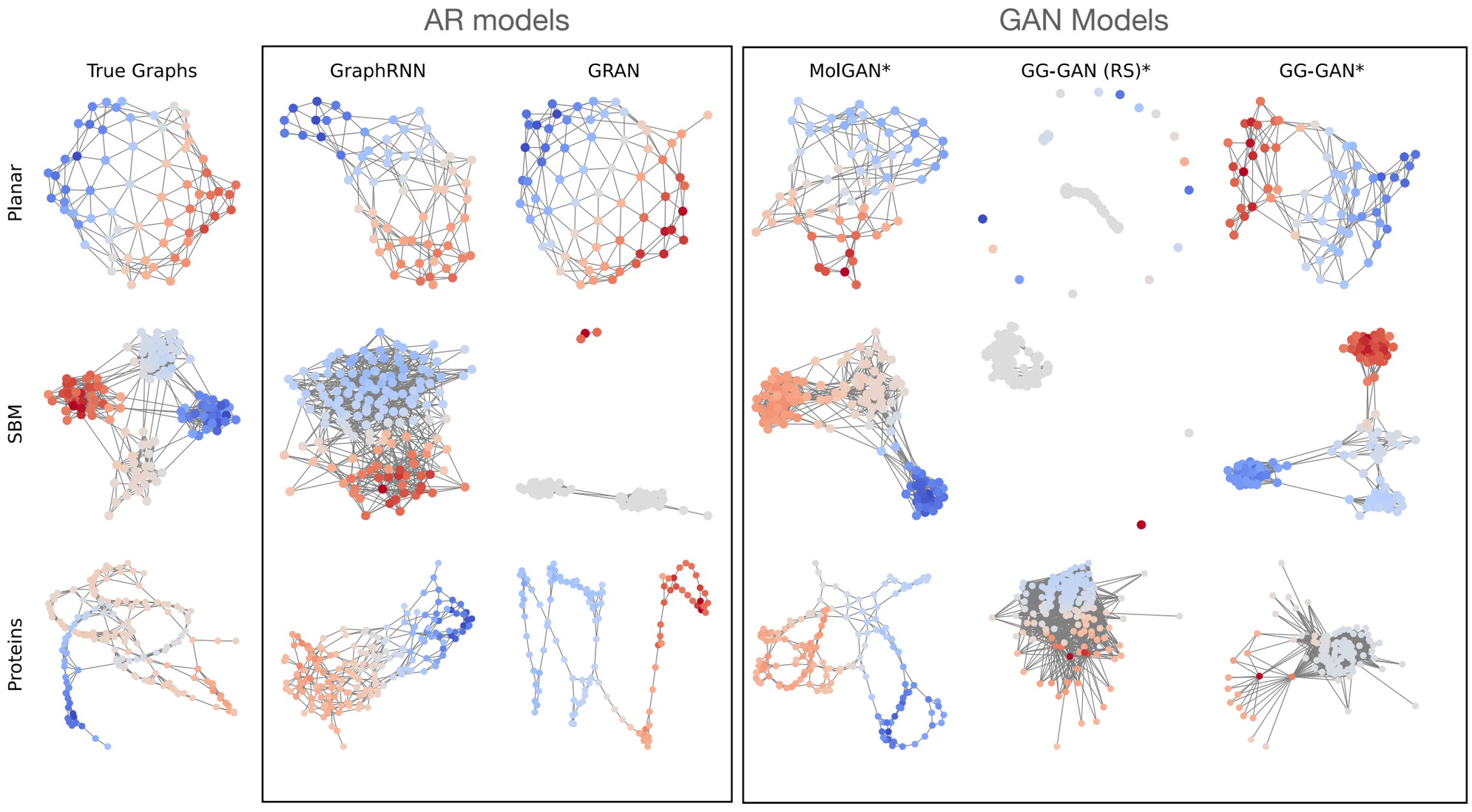


[GraphRNN, You et al. 2018]

GraphRNN limitations

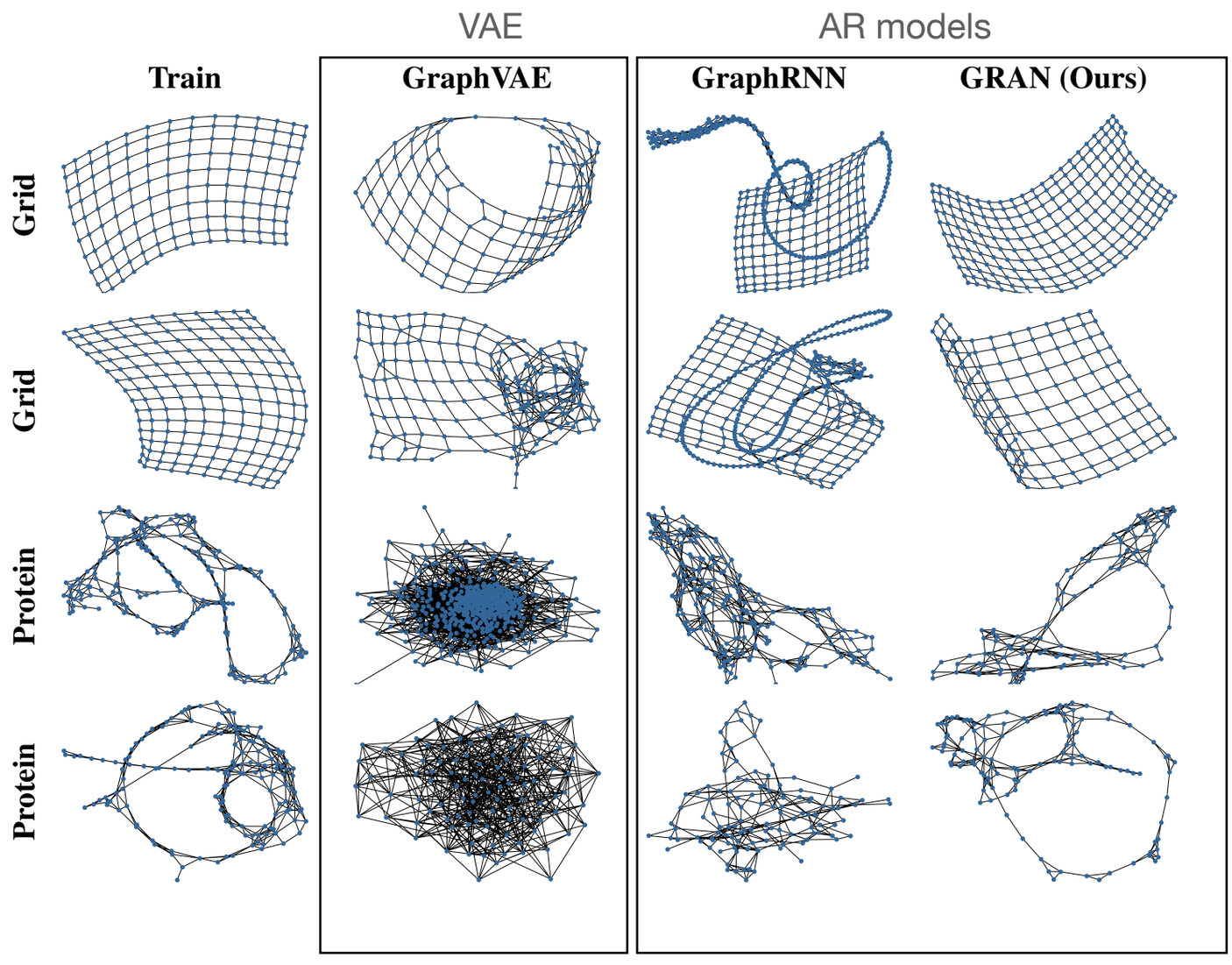
- It generates unrealistic artefacts when trained on samples of grids
- It can be difficult to train and scale due to the need to back propagate through many steps of RNN recurrence
- It requires node ordering: struggling with permutation invariance

Some comparisons



[Martinkus et al., 2022]

Some comparisons



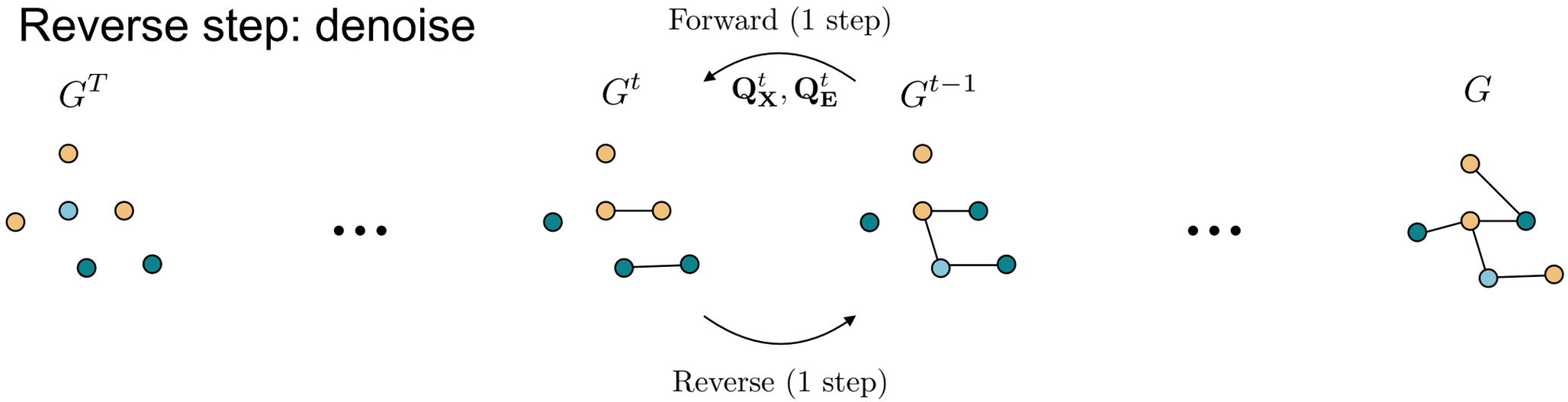
[Liao et al., 2019]

Diffusion models on graphs

- Two main processes:
  - Forward step: add noise
  - Reverse step: denoise

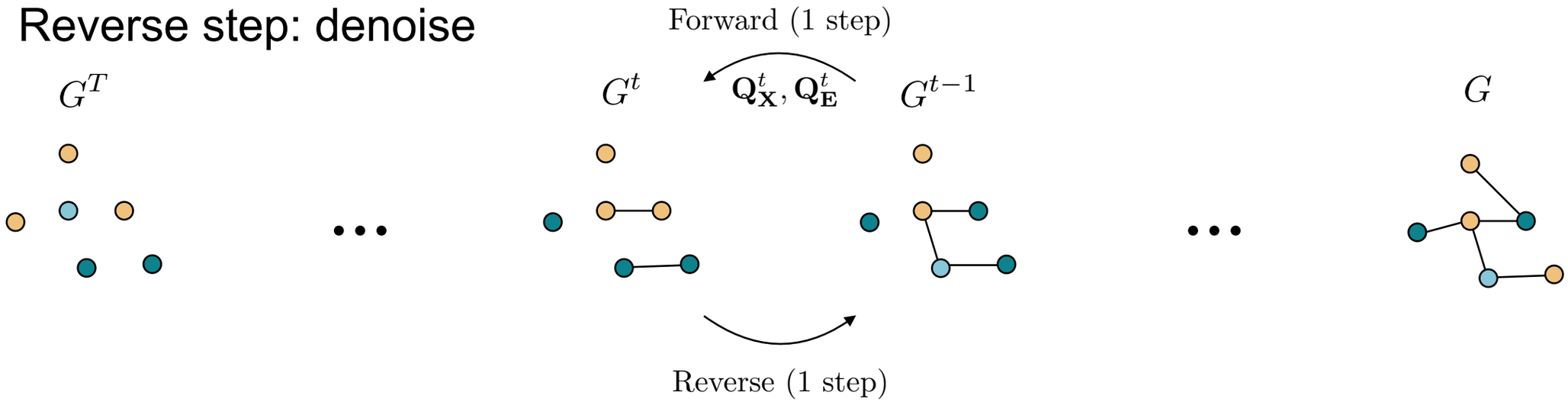
Diffusion models on graphs

- Two main processes:
  - Forward step: add noise
  - Reverse step: denoise



Diffusion models on graphs

- Two main processes:
  - Forward step: add noise
  - Reverse step: denoise



Each node and each edge lie in a discrete state-space:

- Nodes: $\mathbf{x}_i \in \{0,1\}^b \longrightarrow \mathbf{X} \in \{0,1\}^{n \times b}$
- Edges: $\mathbf{e}_{ij} \in \{0,1\}^{c+1} \longrightarrow \mathbf{E} \in \{0,1\}^{n \times n \times (c+1)}$

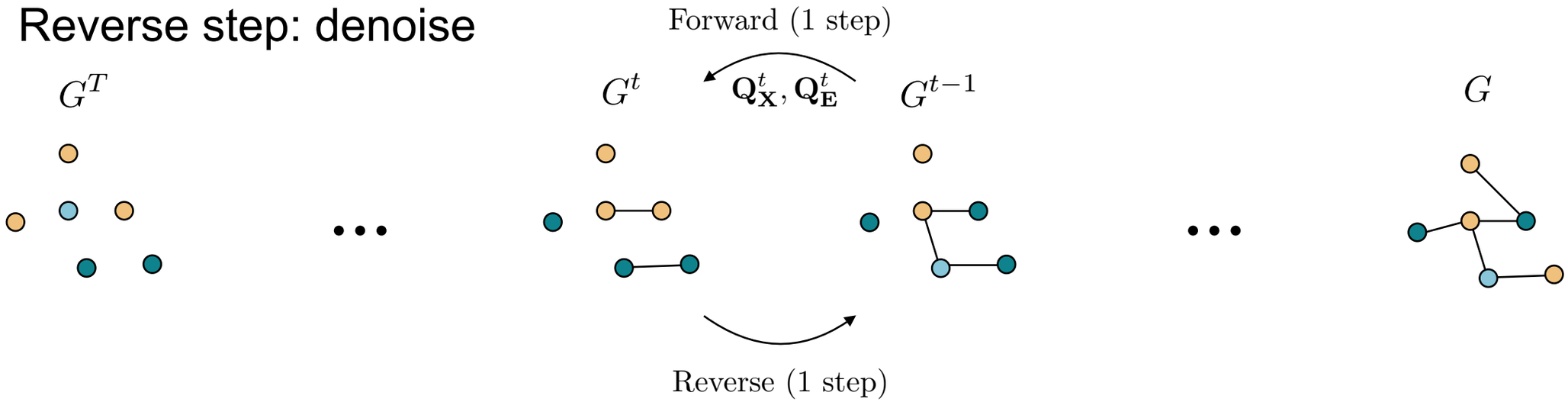
("No edge" is an edge state)

Forward: noise applied independently to each node and edge using $\mathbf{Q}^t_X$ and $\mathbf{Q}^t_E$

Reverse: use a denoising graph neural network - GNN_θ

Diffusion models on graphs

- Two main processes:
  - Forward step: add noise
  - Reverse step: denoise



Each node and each edge lie in a discrete state-space:

- Nodes: $\mathbf{x}_i \in \{0,1\}^b \longrightarrow \mathbf{X} \in \{0,1\}^{n \times b}$
- Edges: $\mathbf{e}_{ij} \in \{0,1\}^{c+1} \longrightarrow \mathbf{E} \in \{0,1\}^{n \times n \times (c+1)}$

(“No edge” is an edge state)

✓ Permutation Equi/Invariant

✓ Good generative performance

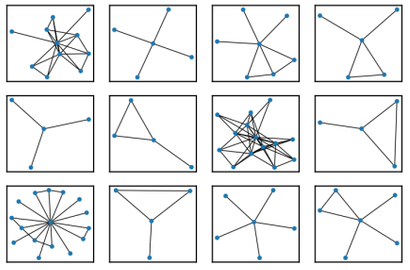
Forward: noise applied independently to each node and edge using $\mathbf{Q}^t_X$ and $\mathbf{Q}^t_E$

Reverse: use a denoising graph neural network - GNN_θ

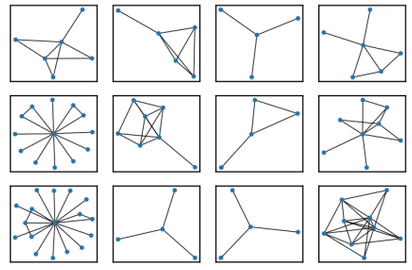
Some prominent graph diffusion models

EDP-GNN

- First Graph Diffusion Model
- Permutation Invariance



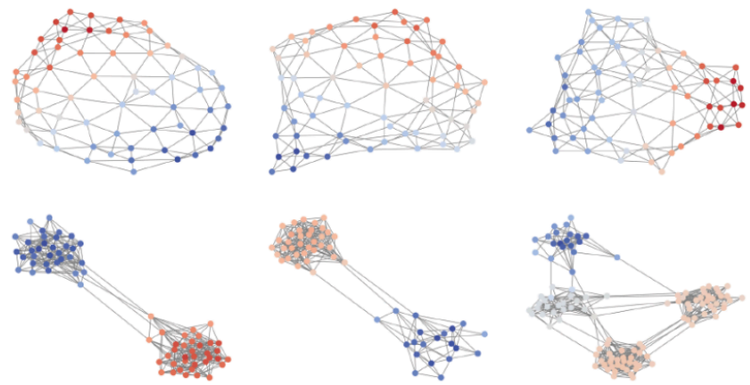
Training data



EDP-GNN samples

DiGress

- SOTA performance for a long time
- Widely adopted for different applications



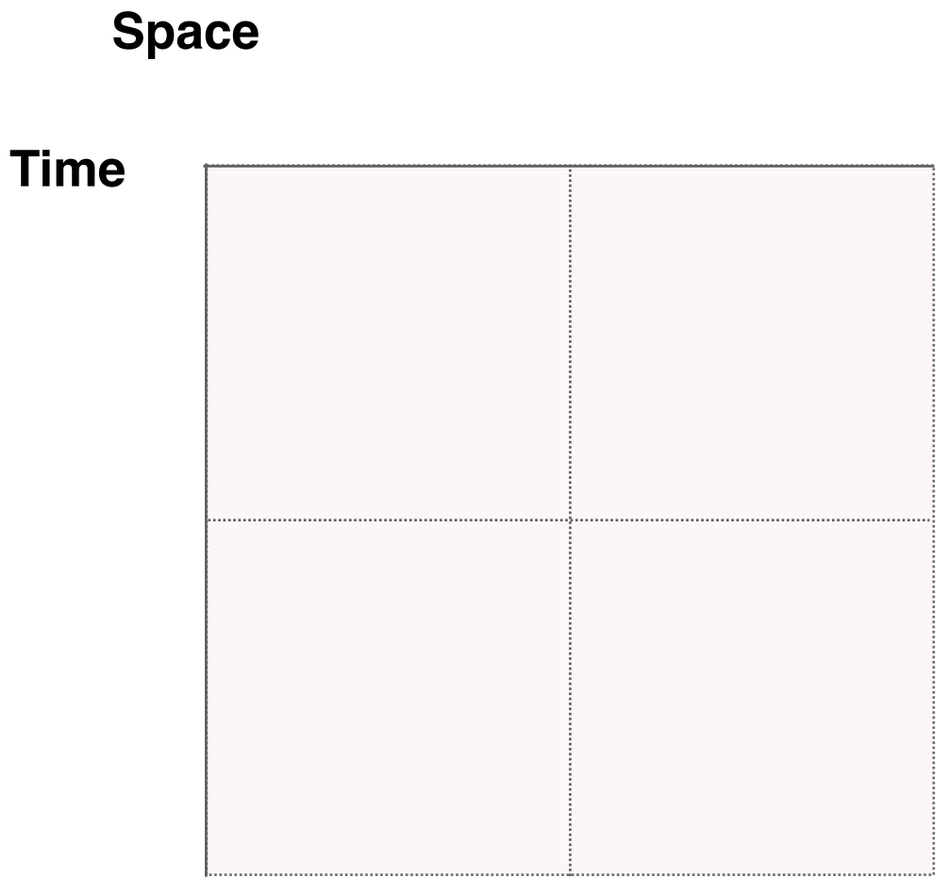
DiGress samples

Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023

Taxonomy: State-space and time

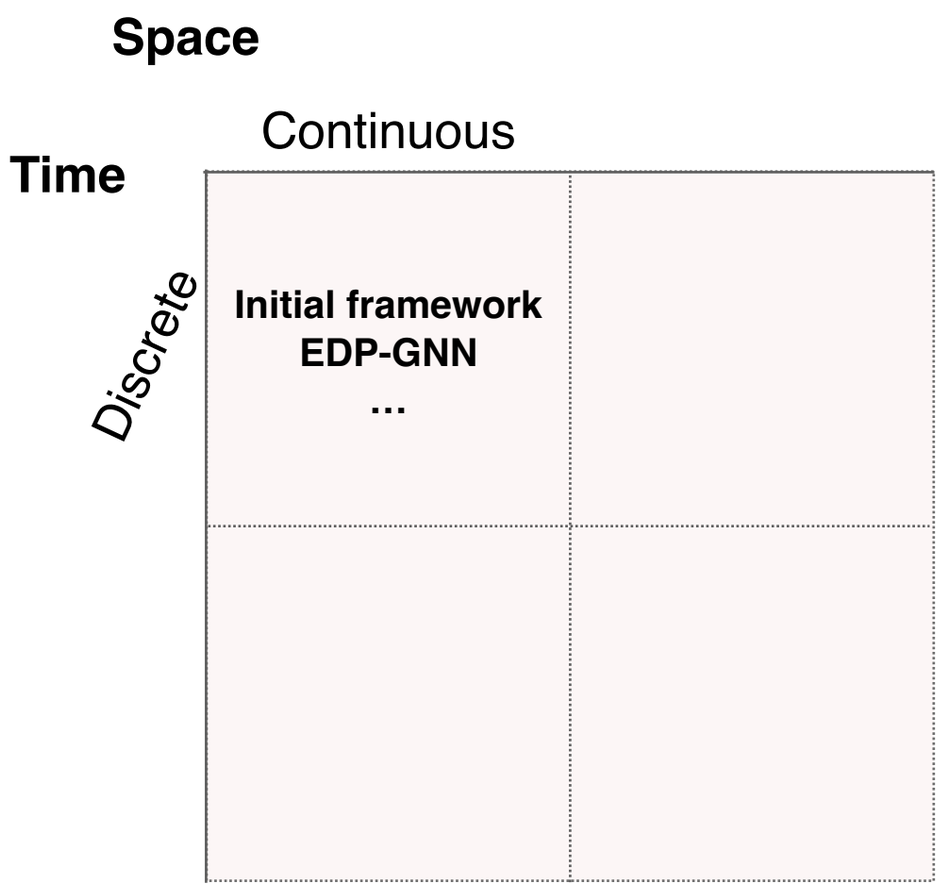
Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

Taxonomy: State-space and time



Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

Taxonomy: State-space and time

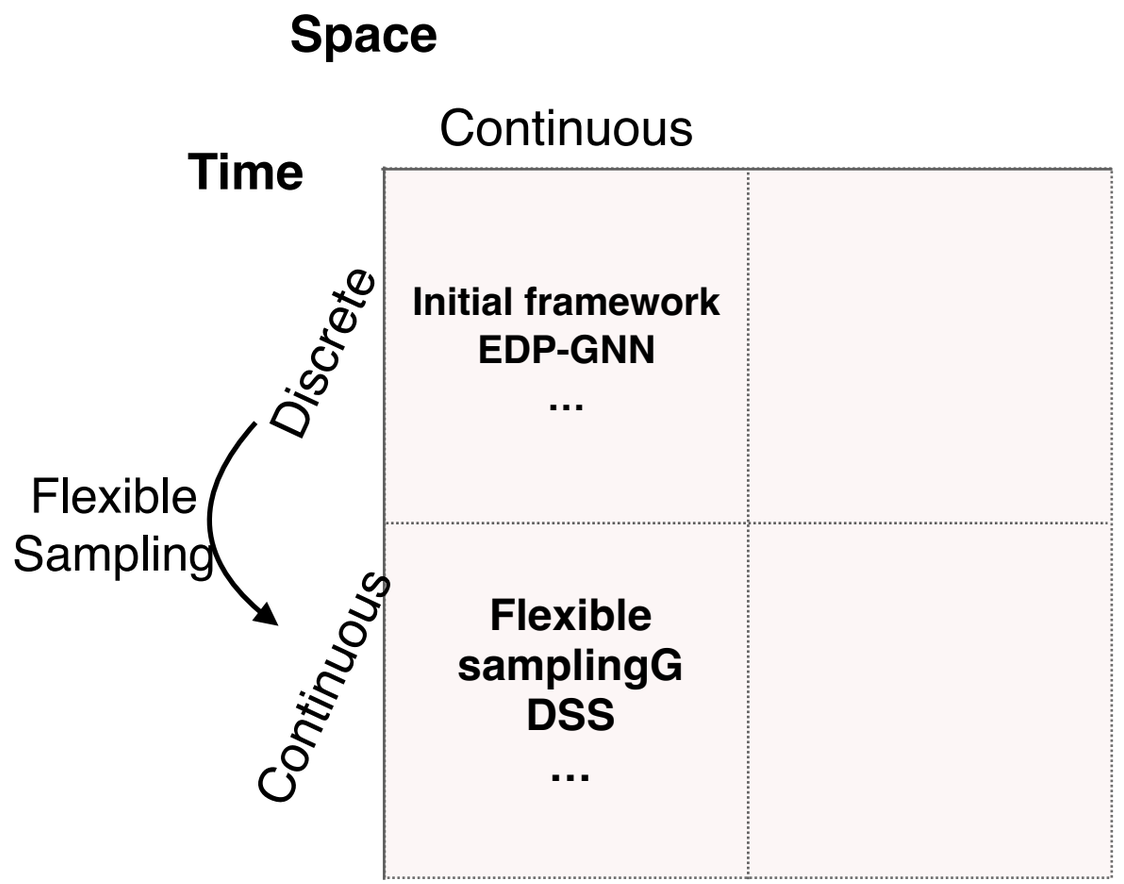


Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

Taxonomy: State-space and time

Continuous time:

- More robust training
- More flexible sampling



Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

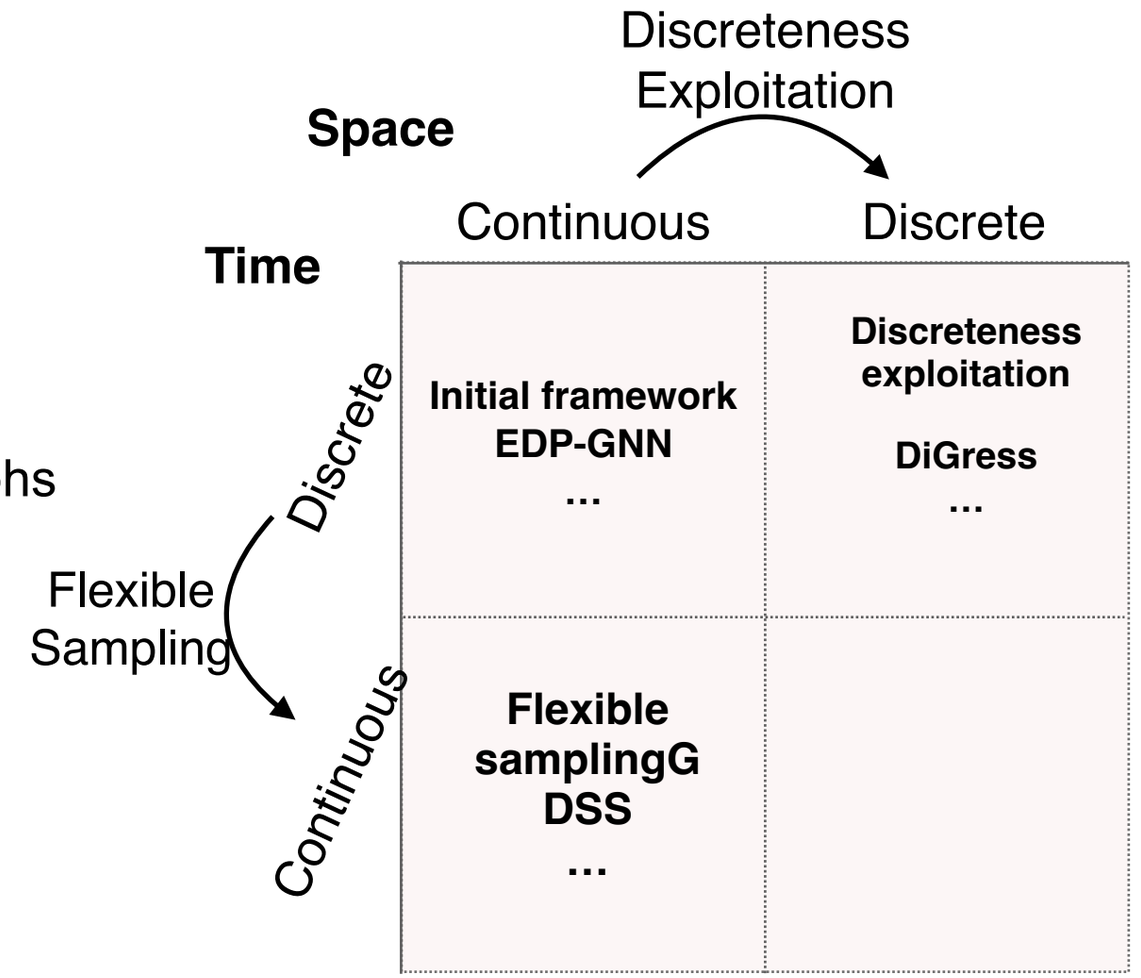
Taxonomy: State-space and time

Continuous time:

- More robust training
- More flexible sampling

Discrete State-spaces:

- Preserve topological characteristics of graphs



Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

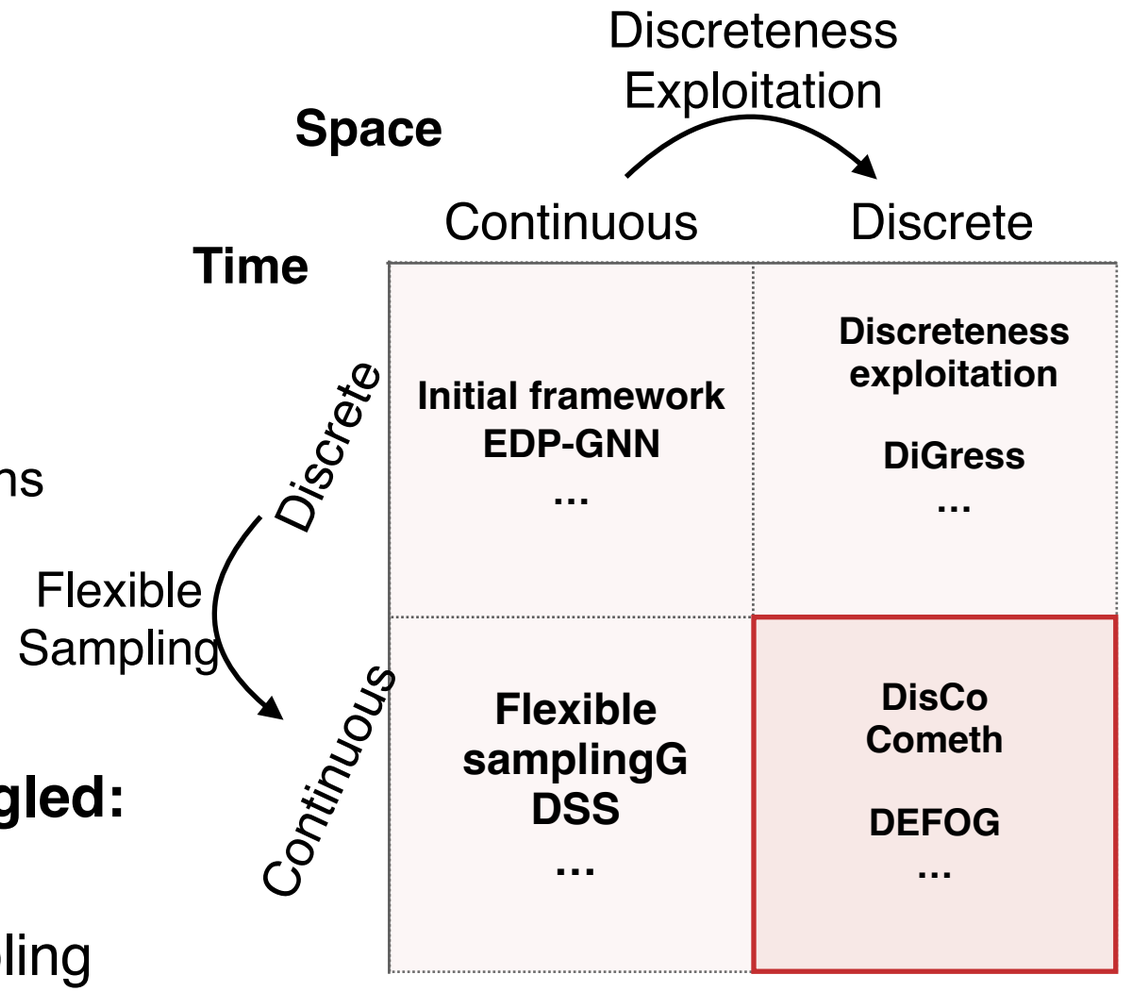
Taxonomy: State-space and time

Continuous time:

- More robust training
- More flexible sampling

Discrete State-spaces:

- Preserve topological characteristics of graphs



Training and sampling are often entangled:

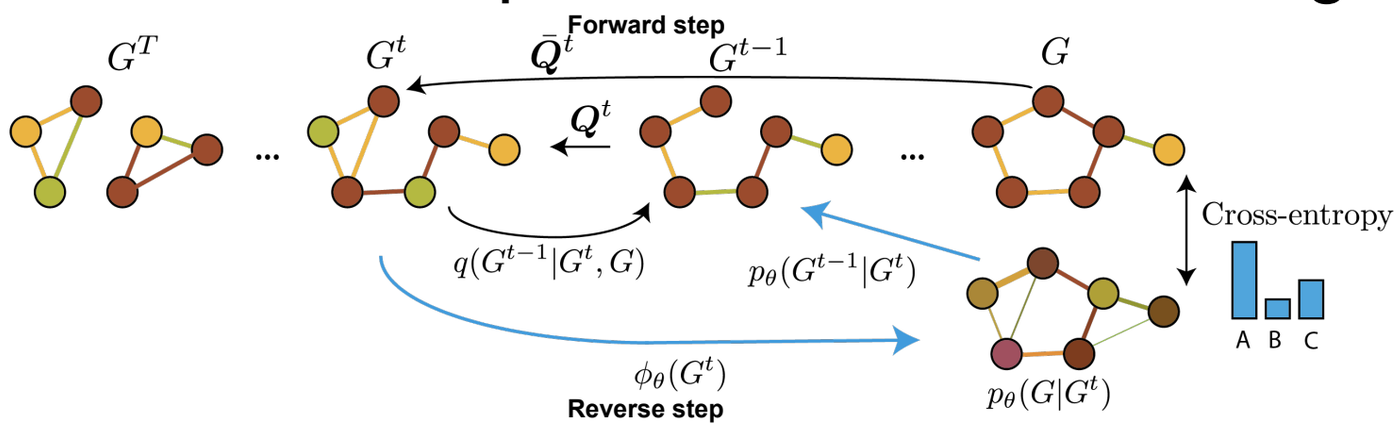
- Choices during training constrain sampling
- Hyperparameter tuning requires retraining, limiting adaptability

SOTA

Permutation Invariant Graph Generation via Score-Based Generative Modeling, Niu et al., AISTATS 2020
Score-based Generative Modeling of Graphs via the System of Stochastic Differential Equations, Jo et al., ICML 2022
DiGress: Discrete denoising diffusion for graph generation, Vignac et al., ICLR 2023
Discrete-state Continuous-time Diffusion for Graph Generation, Xu et al., NeuIPS 2024
Cometh: A continuous-time discrete-state graph diffusion model, Siraudin et al., ArXiv 2024

Discrete diffusion on graphs - DiGress

- Graph generation = sequence of a node and edge classification tasks



Graphs are **inherently discrete** → Replace continuous diffusion framework by their discrete state-spaces counterpart, e.g.:

Discrete noise model:

$$q(G^t|G^{t-1}) = (\boldsymbol{X}^{t-1}\boldsymbol{Q}_X^t, \mathbf{E}^{t-1}\boldsymbol{Q}_E^t)$$

Markov transition matrices

Denoise the process with an equivariant architecture → Graph Transformer with expressive features as input: cyclic and spectral features

Graphs are sparse → Use sparsity-inducing transition matrices: marginal distribution of nodes/edges

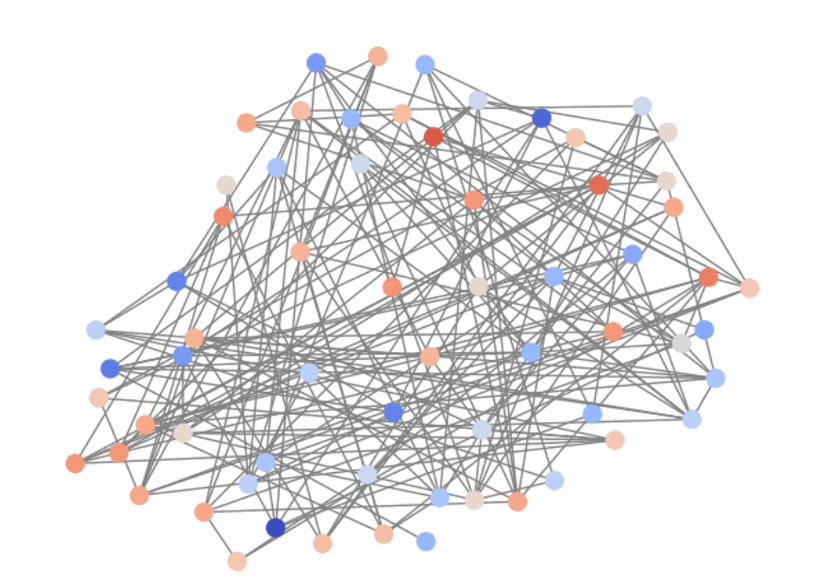
$$\boldsymbol{Q}_X^t = \alpha^t \boldsymbol{I} + \beta^t \, \mathbf{1}_a \boldsymbol{m}_X'$$

$$\boldsymbol{Q}_E^t = \alpha^t \boldsymbol{I} + \beta^t \, \mathbf{1}_b \boldsymbol{m}_E'$$

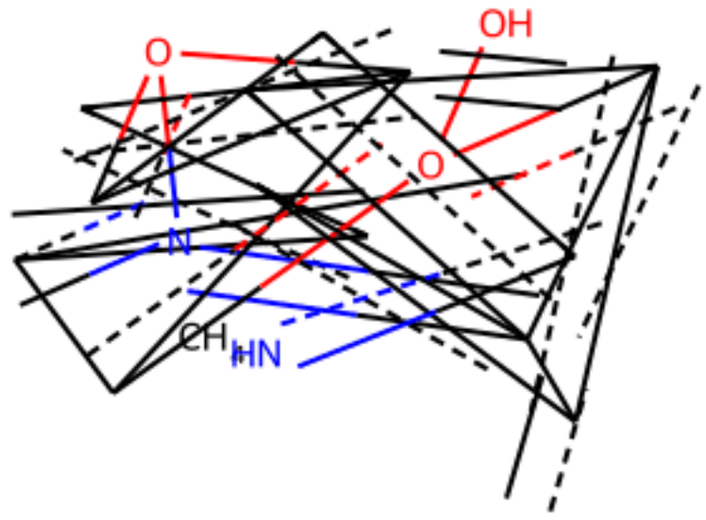
[DiGress, Vignac et al., 2023]

Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)

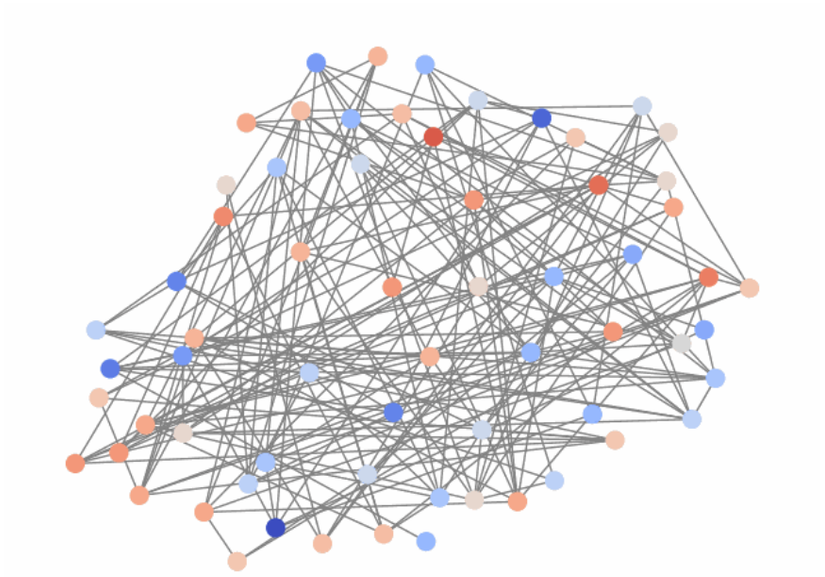


Real World Applications:
Molecular Generation

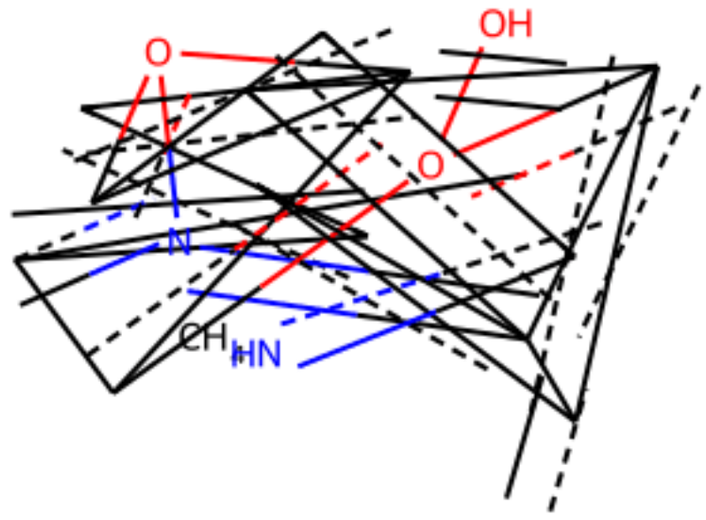


Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)

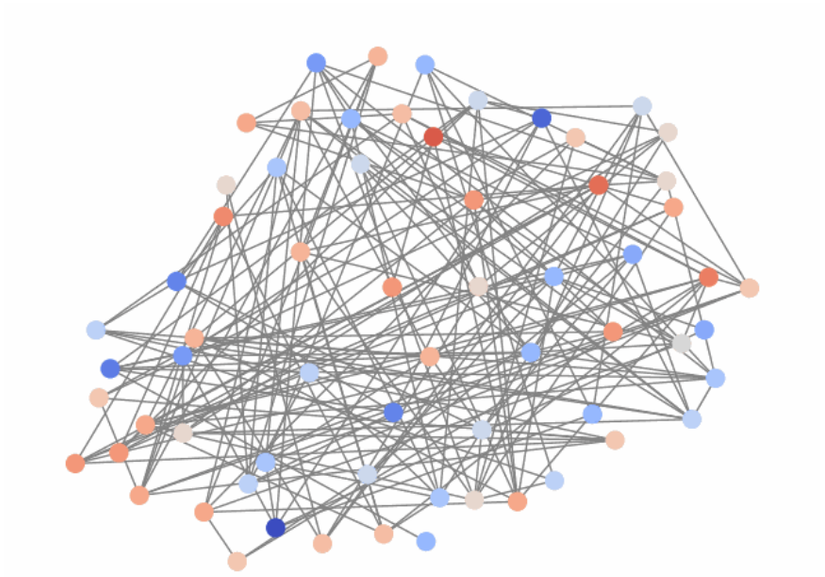


Real World Applications:
Molecular Generation

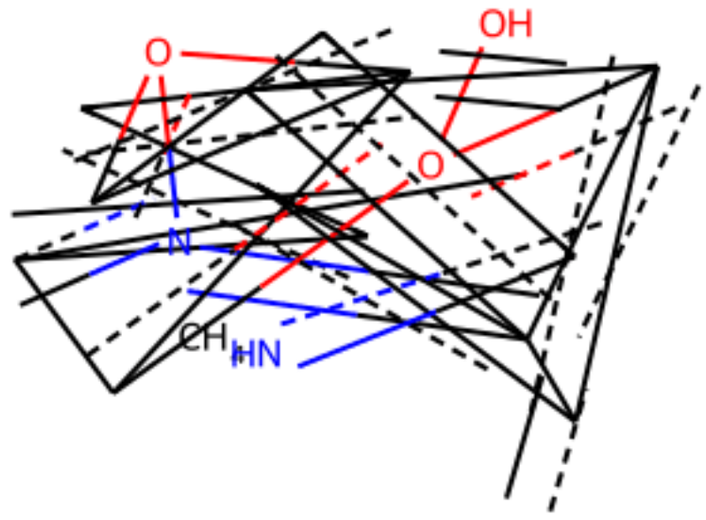


Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)



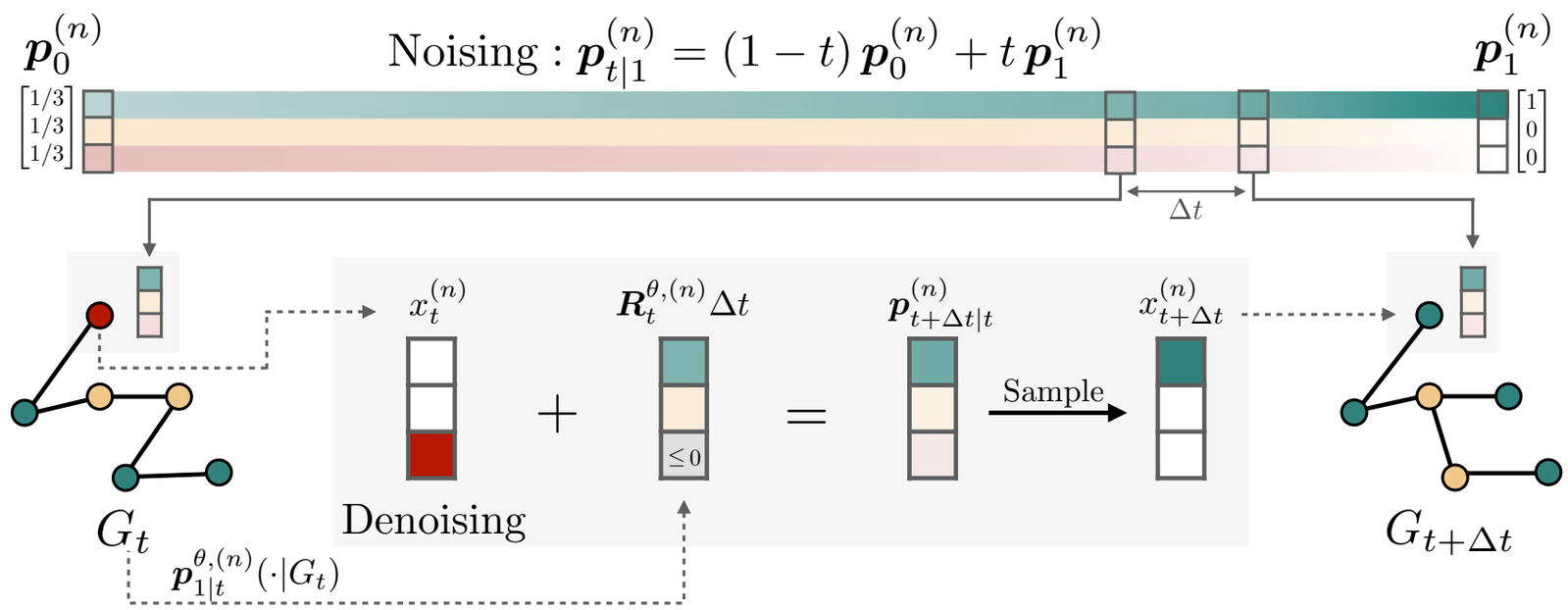
Real World Applications:
Molecular Generation



DeFoG: decoupling training and sampling

- Builds on discrete flow matching

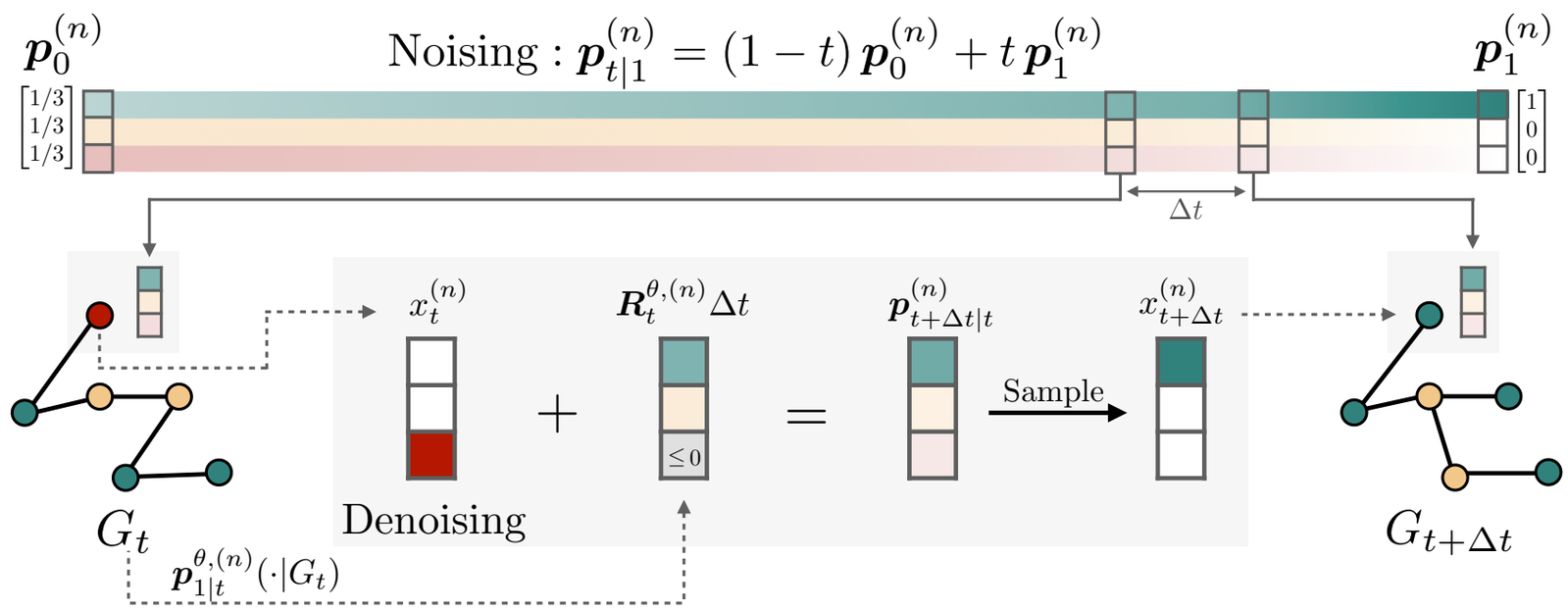
Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0



DeFoG: decoupling training and sampling

- Builds on discrete flow matching

Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0

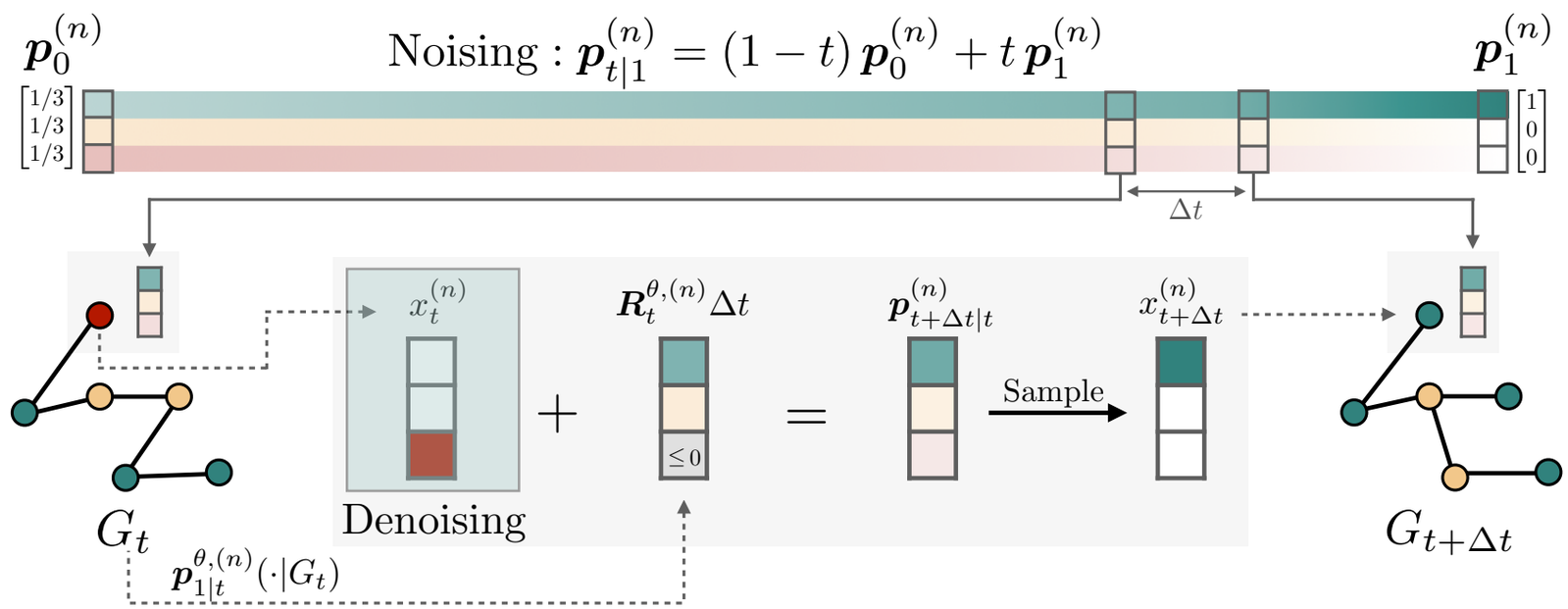


Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

- Builds on discrete flow matching

Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0

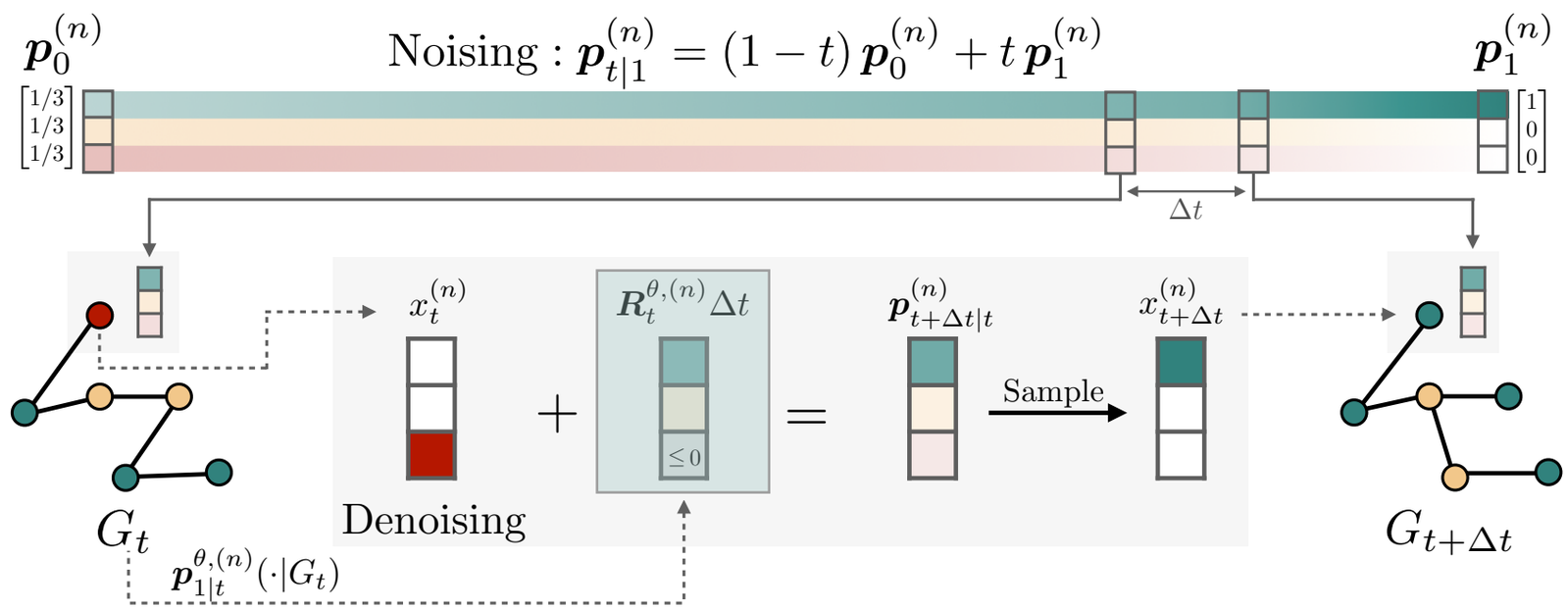


Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

- Builds on discrete flow matching

Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0

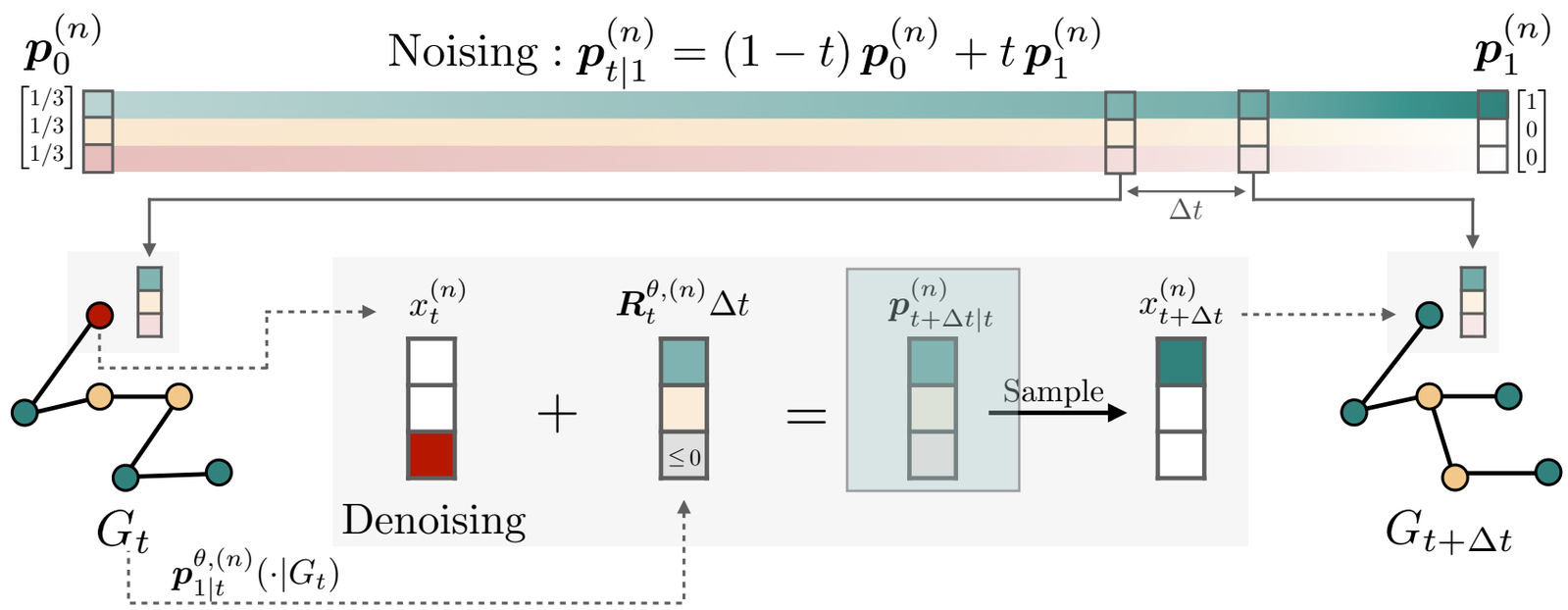


Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

- Builds on discrete flow matching

Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0

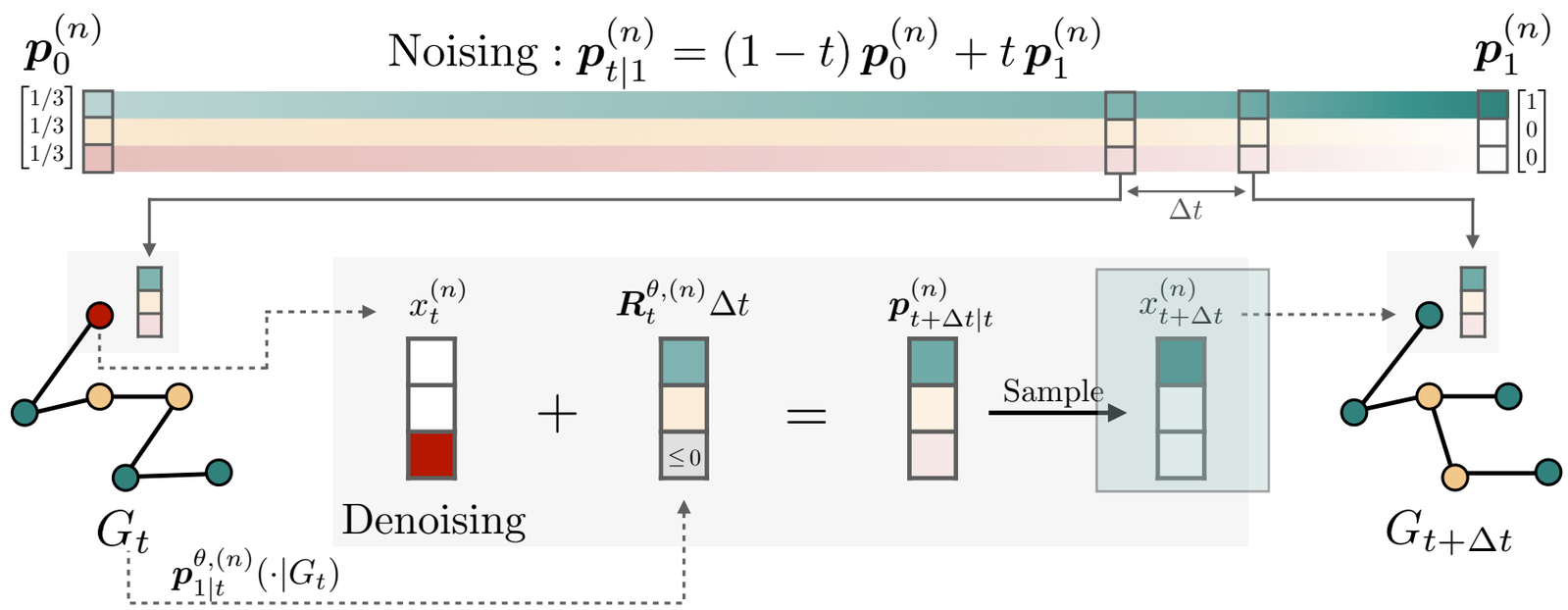


Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

- Builds on discrete flow matching

Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0

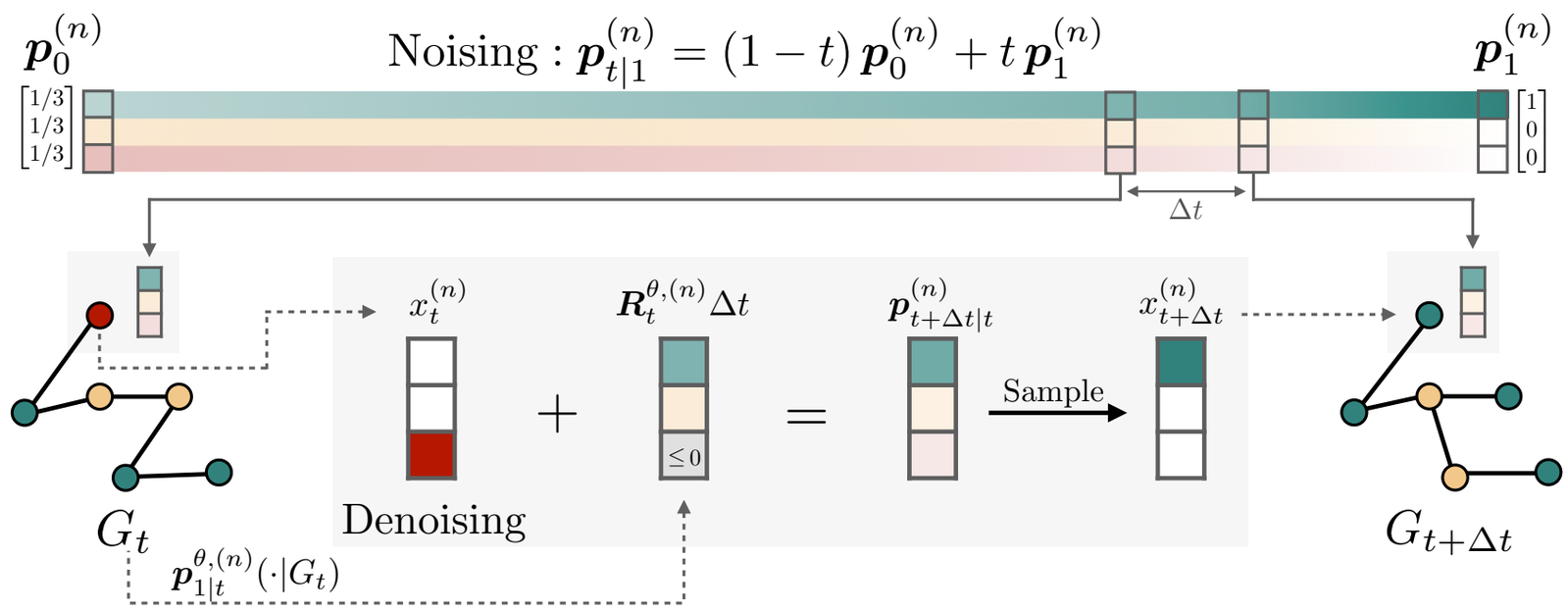


Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

- Builds on discrete flow matching

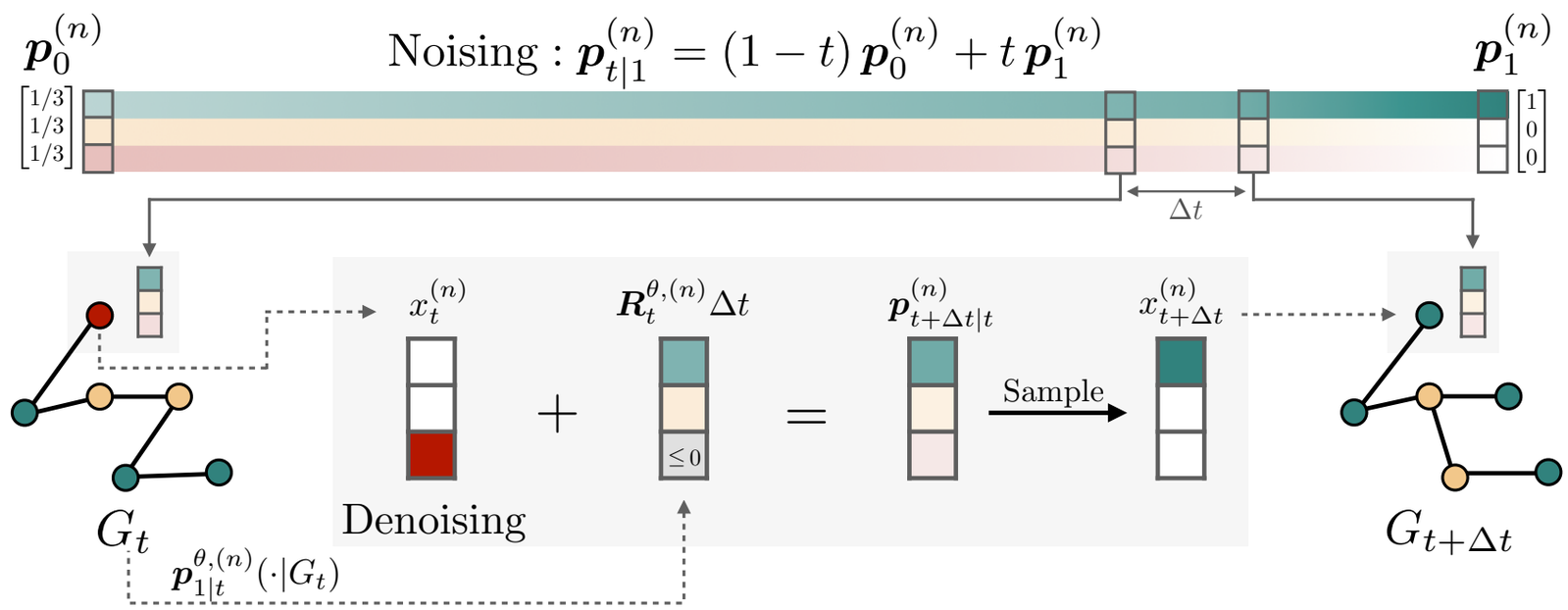
Noising process ($t = 1 \rightarrow t = 0$):
Linear interpolation between data distribution p_1 and initial distribution p_0



Denoising process ($t = 0 \rightarrow t = 1$):
Reverts noising process based on a CTMC formulation

DeFoG: decoupling training and sampling

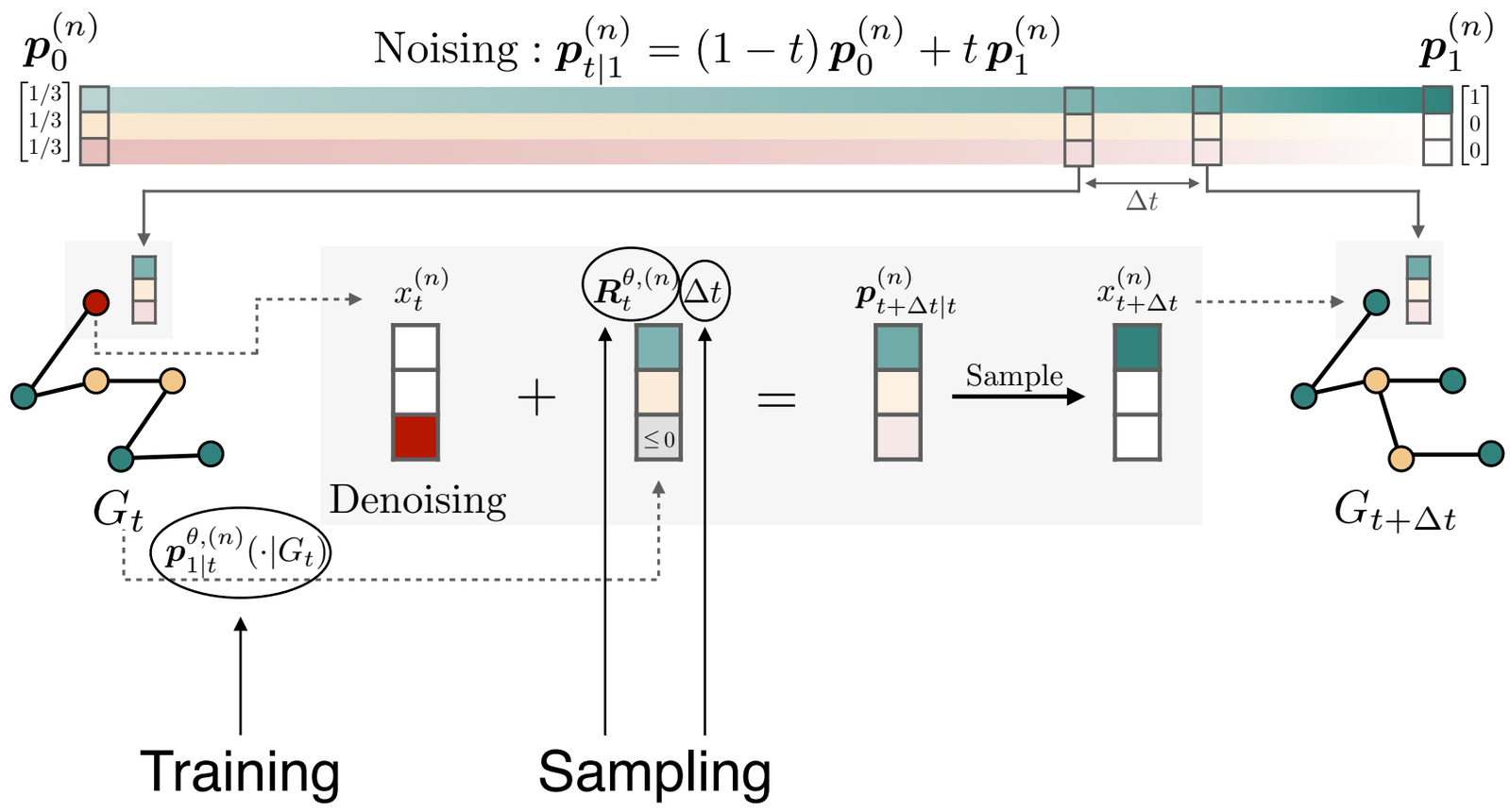
But where does the **decoupling** come from?



[Qin et al., *DeFoG,* ICML 2025]

DeFoG: decoupling training and sampling

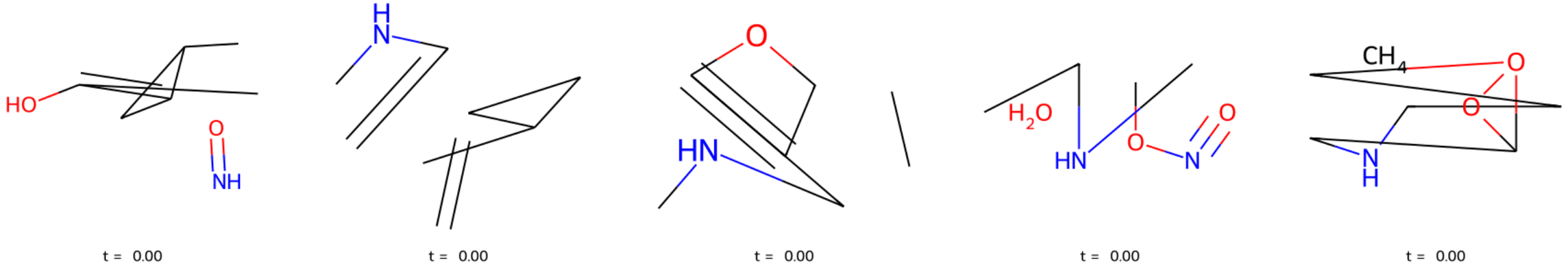
But where does the **decoupling** come from?



[Qin et al., *DeFoG,* ICML 2025]

Molecular Generation

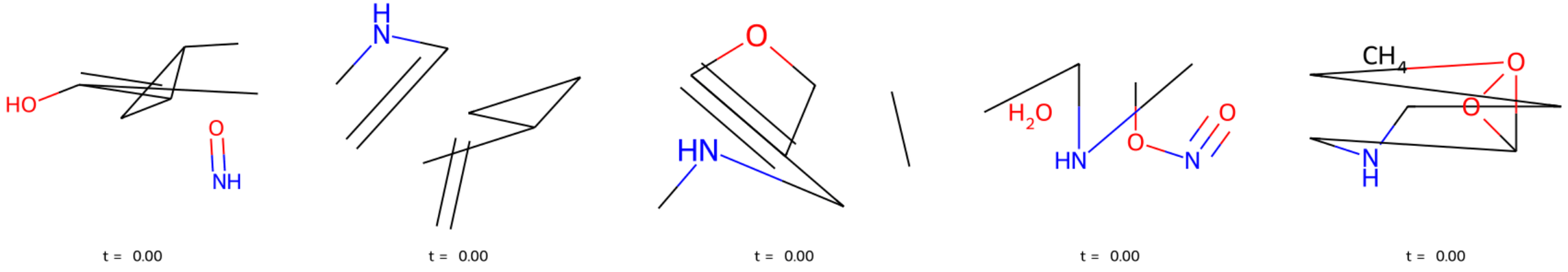
- Enables exploration of large, unstructured chemical spaces
- **Accelerate drug discovery and material design:** generate chemically valid molecular graphs beyond known molecules



[Qin et al., *DeFoG: Discrete Flow Matching for Graph Generation*, ICML 2025]

Molecular Generation

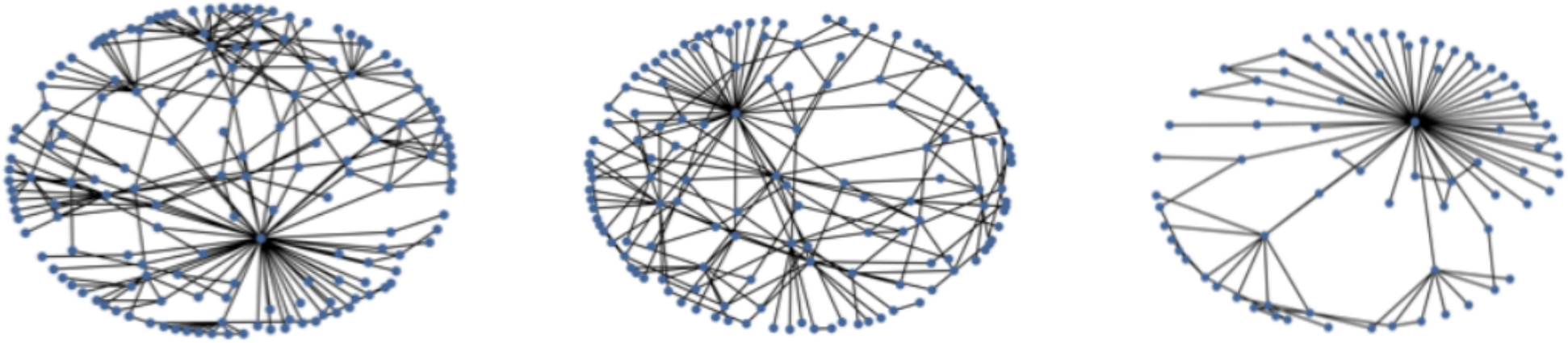
- Enables exploration of large, unstructured chemical spaces
- **Accelerate drug discovery and material design:** generate chemically valid molecular graphs beyond known molecules



[Qin et al., *DeFoG: Discrete Flow Matching for Graph Generation*, ICML 2025]

Network Simulation

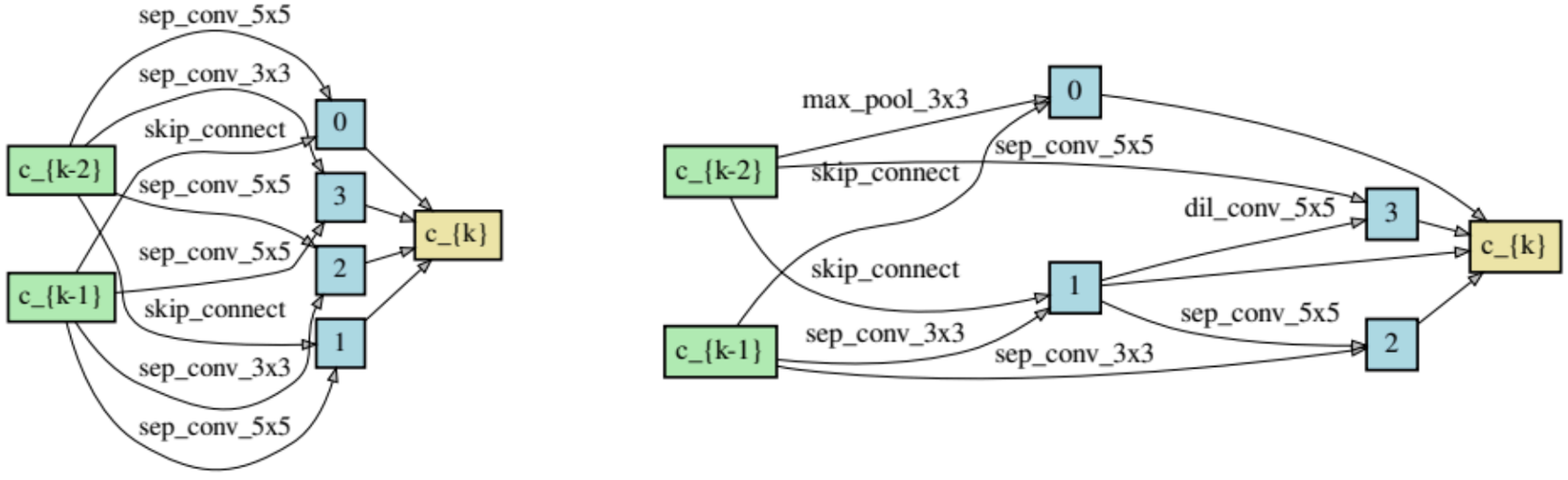
- Create synthetic graphs that **mimic real-world network statistics** (e.g., degree distribution, clustering)
- Useful for **benchmarking algorithms and studying social or information diffusion without privacy concerns**



[You et al., GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models, ICML 2018]

Neural Architecture Search

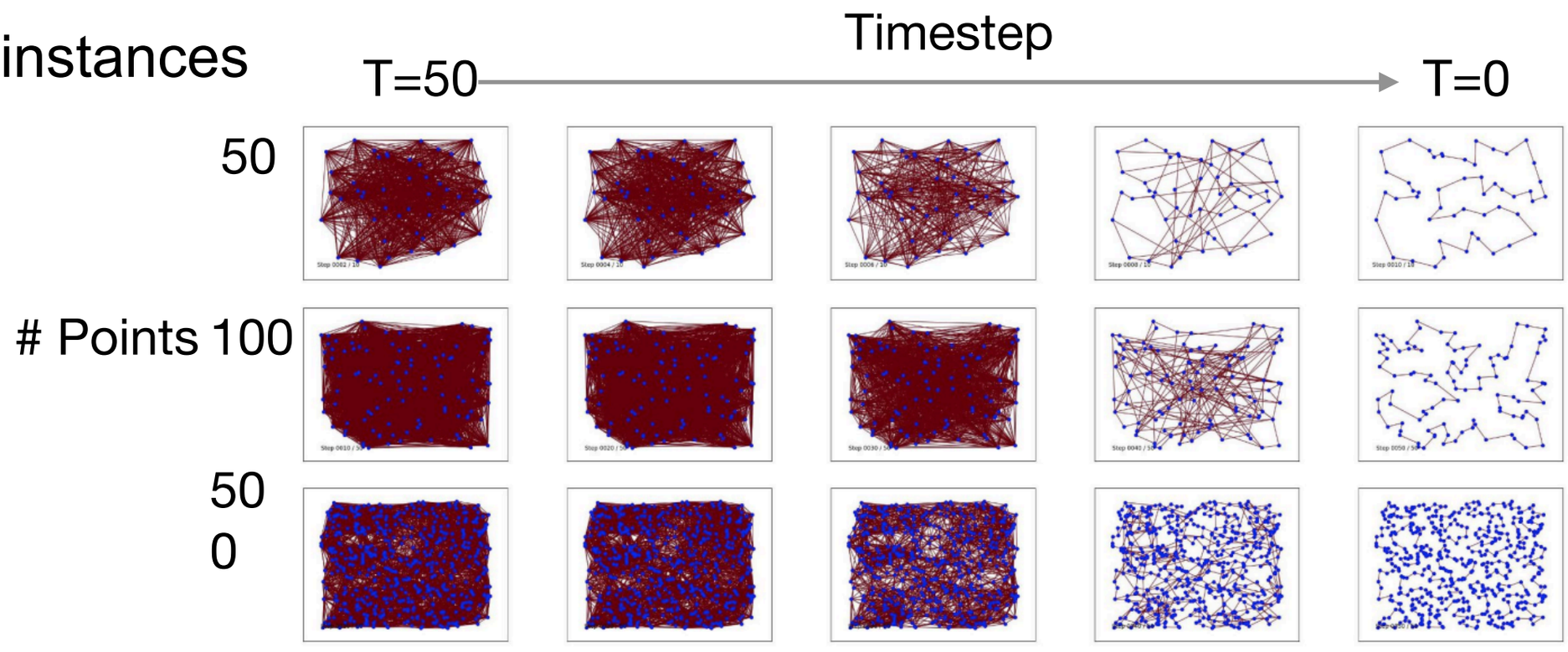
- Represent neural networks as **directed acyclic graphs** to **explore architectural design spaces**
- Graph generation enables **automated search for performant models** under multiple constraints (e.g. latency, accuracy)



[Asthana et al., *Multi-conditioned Graph Diffusion for Neural Architecture Search*, TMLR 2024]

Combinatorial Optimisation

- Generate high-quality **graph-structured solutions for NP-hard problems** like Traveling Salesman Problem (TSP) and Maximal Independent Set (MIS)
- Provides **scalable, learned solvers that generalize** across problem instances

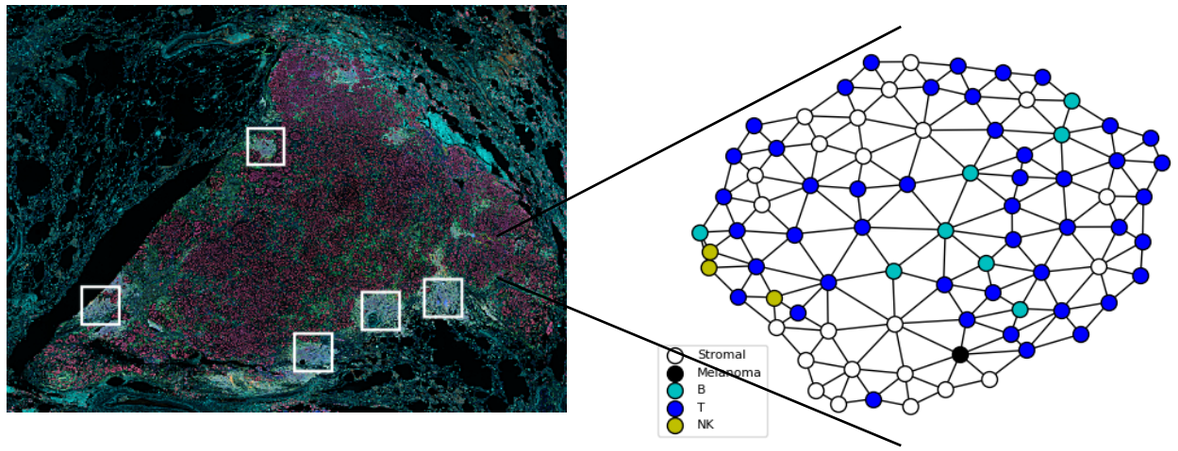


[Sun et al., *DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization*, NeurIPS 2023]

Spatial biology

- Tertiary lymphoid organs (TLSs) are immune infiltrations that play an important role in immunotherapy response
  - They are highly organized structures that consist of distinct B-cell clusters and surrounding T-cell compartments
- Graph generative models can generate structures with low TLS and high TLS content reflecting the cell organization structure

Model	TLS Dataset	
	V.U.N. ↑	TLS Val. ↑
Train set	0.0	100
GraphGen (Goyal et al., 2020)	40.2±3.8	25.1±1.2
BiGG (Dai et al., 2020)	0.6±0.4	16.7±1.6
SPECTRE (Martinkus et al., 2022)	7.9±1.3	25.3±0.8
DiGress+ (Madeira et al., 2024)	13.2±3.4	12.6±3.0
ConStruct (Madeira et al., 2024)	**99.1**±1.1	92.1±1.3
DeFoG (# steps = 50)	44.5±4.2	<u>93.0</u>±5.6
DeFoG (# steps = 1,000)	<u>94.5</u>±1.8	**95.8**±1.5



Take home messages

- Deep generative models can model graphs with complicated topology and constrained structural properties

- Autoregressive models progressively grow the graph by inserting nodes and edges (e.g., GraphRNN)
  - High flexibility in sampling
  - Need for node ordering

- One shot generation predicts the entire graph in a single step (e.g., VAEs, GANs, Diffusion)
  - Permutation invariance properties

- Ample room for further development!

References

1. Graph representation learning (chap 9), William Hamilton
   - https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book-Chapter_9-Deep_Graph_Generation.pdf

2. A Survey on Deep Graph Generation: Methods and Applications, Zhu et al, 2022
   - https://arxiv.org/pdf/2203.06714

3. GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models,. You et al. 2018
   - https://cs.stanford.edu/people/jure/pubs/graphrnn-icml18.pdf

4. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders.
   - https://arxiv.org/pdf/1802.03480.pdf

References

5. MolGAN: An implicit generative model for small molecular graphs, De Cao et al., 2018
   - https://arxiv.org/pdf/1805.11973.pdf

6. DiGress: Discrete Denoising diffusion for graph generation, Vignac et al., ICLR, 2023
   - https://arxiv.org/pdf/2203.06714

References: tutorials & surveys

1. Tutorial on Deep Generative Models. A. Grover and S. Ermon. International Joint Conference on Artificial Intelligence, 2018

2. Tutorial on Generative Adversarial Networks. Computer Vision and Pattern Recognition, 2018

3. Tutorial on Deep Generative Models. S. Mohamed and D. Rezende. Uncertainty in Artificial Intelligence, 2017

4. Tutorial on Generative Adversarial Networks. I. Goodfellow. Neural Information Processing Systems, 2016

5. Learning deep generative models. R. Salakhutdinov. Annual Review of Statistics and Its Application, 2015