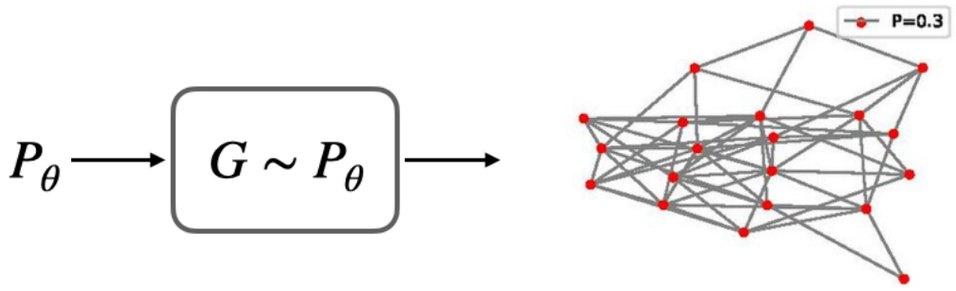
Graph generative models
Part II

Dr Dorina Thanou

May 20, 2025

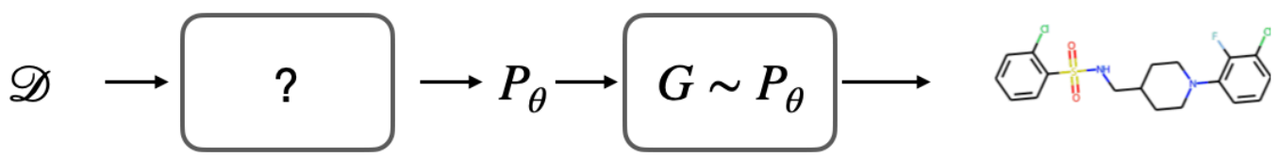
Reminder: Graph generative models

- Random graph models



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

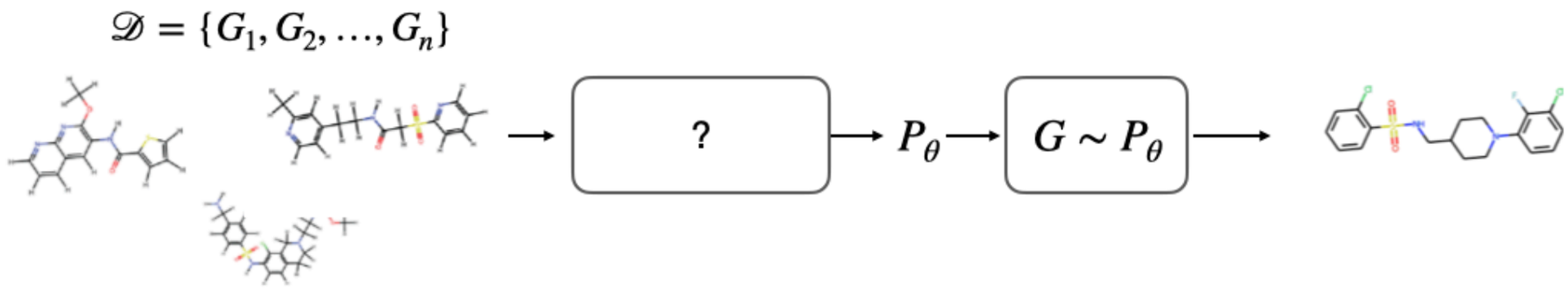
- Deep generative models



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

What will you learn?

- 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



A probabilistic perspective of generative models

Different families of parametrised distributions and how to learn them

Foundations of generative models (last week)

Main challenges for generating graphs

Main families of graph generative models

Specific to graphs (today)

Today's lecture

- Introduction into deep probabilistic graph generative models
- Main architectures
- Applications
- Open discussion/ Feedback on the class

Today's lecture

- **Introduction into deep probabilistic graph generative models**
- Main architectures
- Applications
- Open discussion/ Feedback on the class

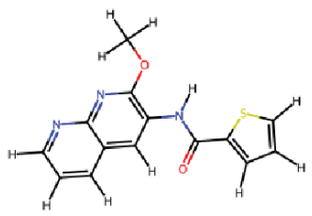
What's New with Graphs?

- **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



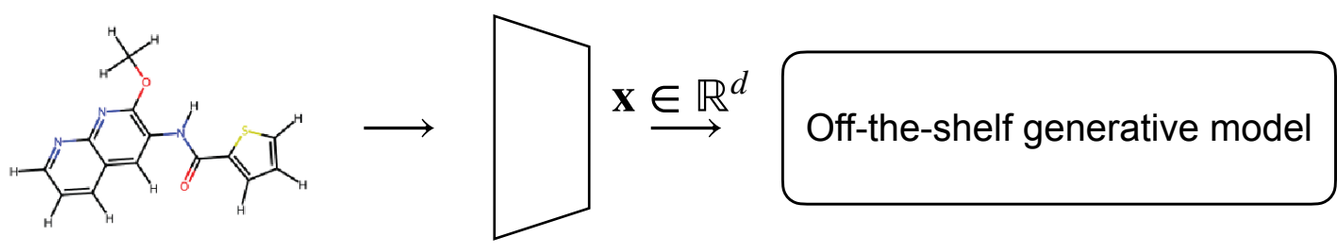
$$G \rightarrow A \rightarrow vec(A) \longrightarrow$$

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

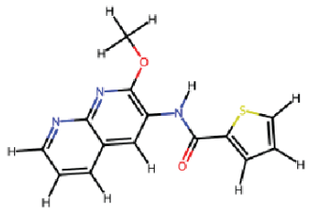
- Rather than vectoring the adjacency matrix:
  - Learn graph embeddings as compact representations
  - Use these embeddings as input of classical generative models



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

Topological Information

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



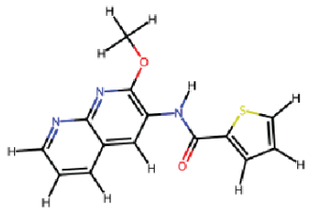
→ **generative model**

- $\mathbf{X} \in \mathbb{R}^{N \times D}$ 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**

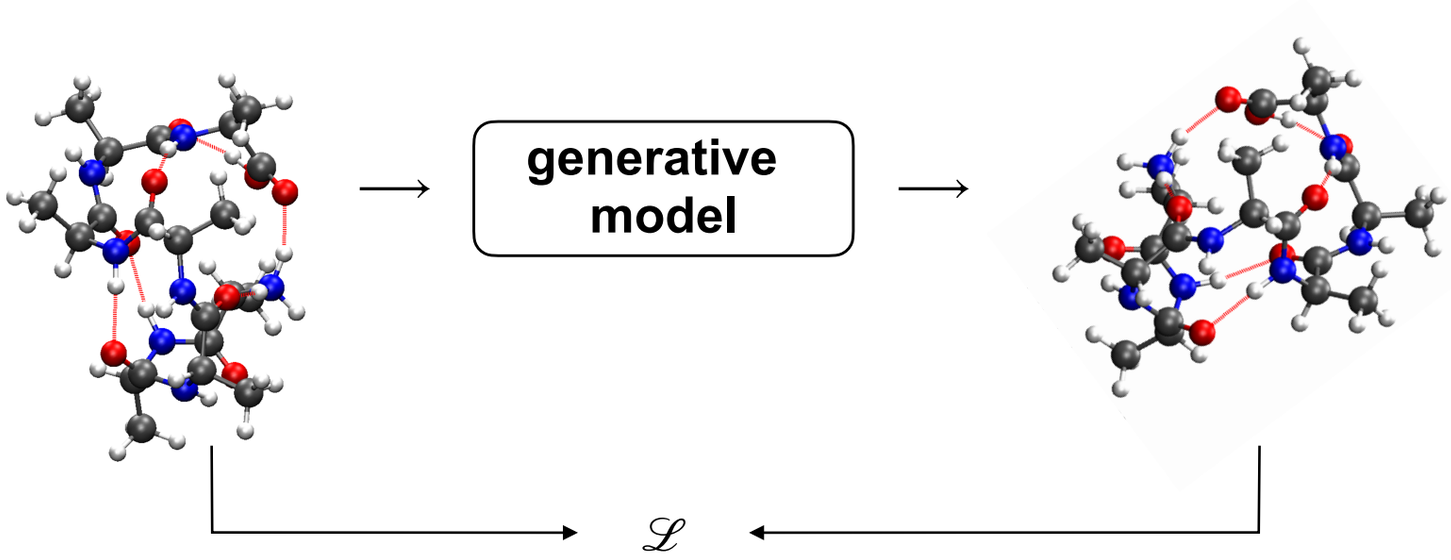
- Should we learn joint distribution of G?
- Should we rather learn the joint distribution of X and E independently?
- Should we treat them as categorical or continuous data?

- $\mathbf{X} \in \mathbb{R}^{N \times D}$ 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

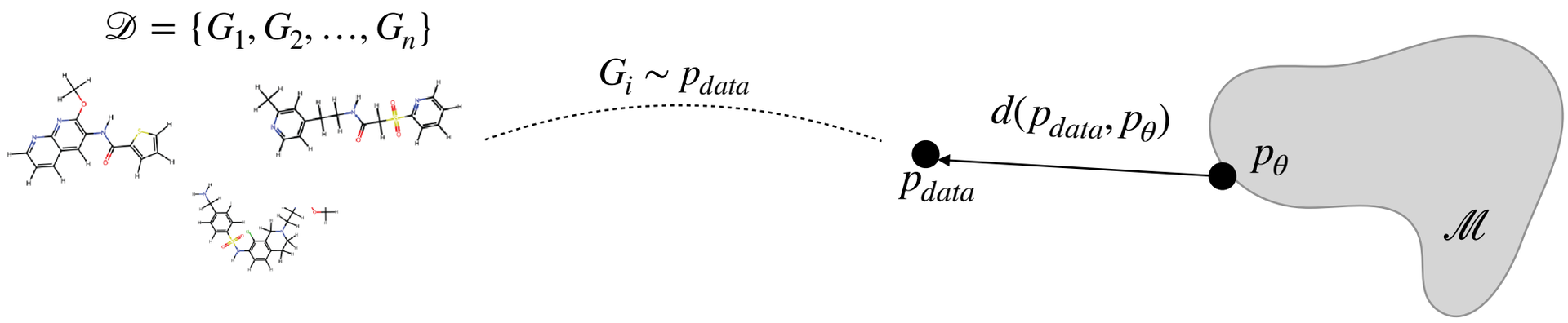
Evaluating Similarity

- 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)



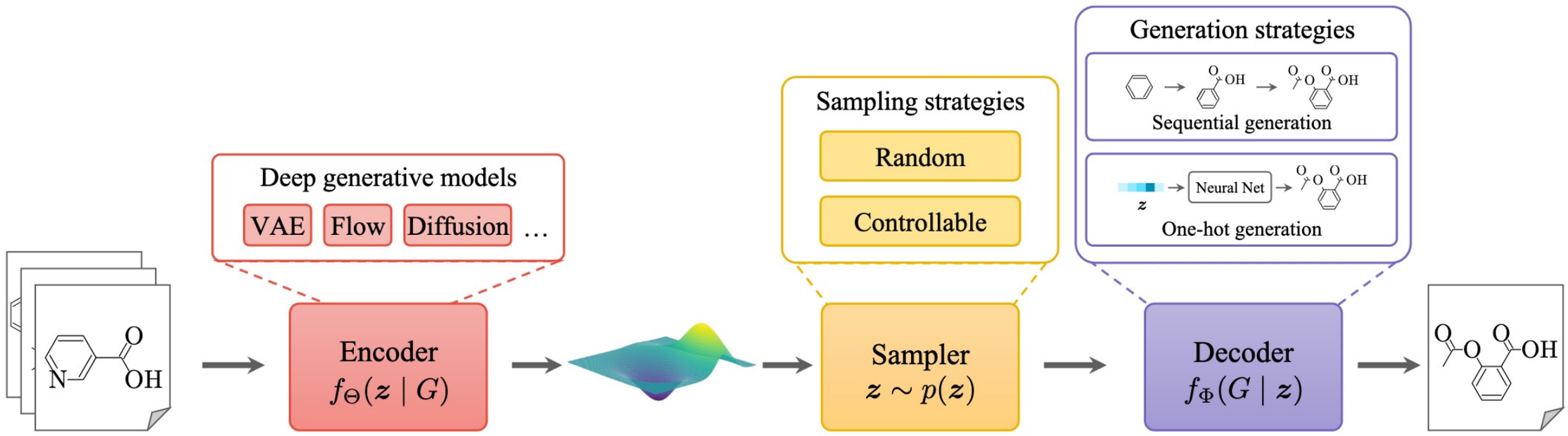
Loss function needs to be permutation equivariant!

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

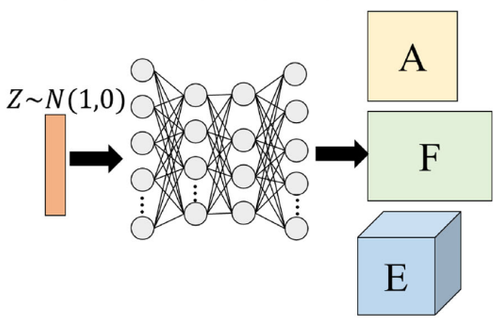
Graph Generator

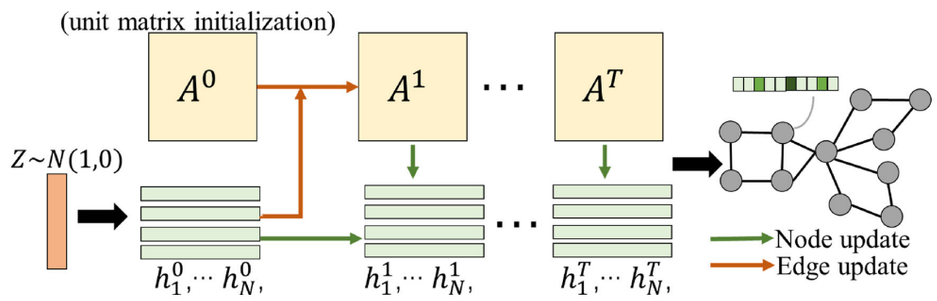
One Shot Generation

Edge based

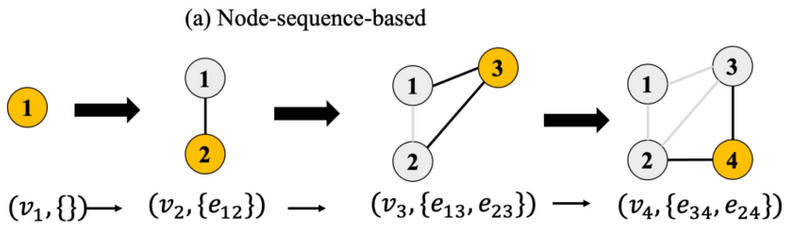
It requires a generative model that predicts edges independently

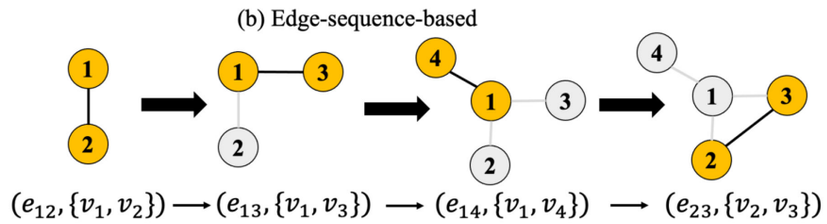
Matrix based

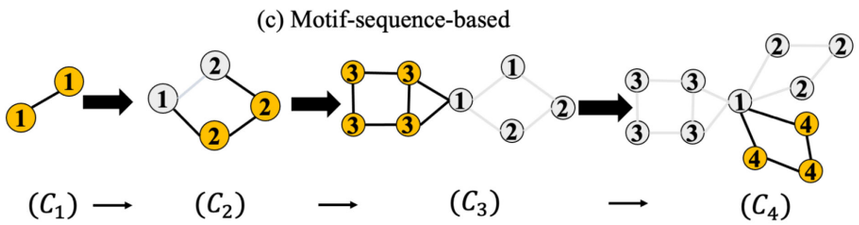


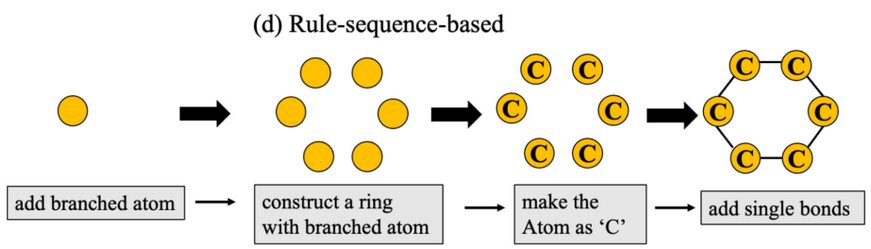


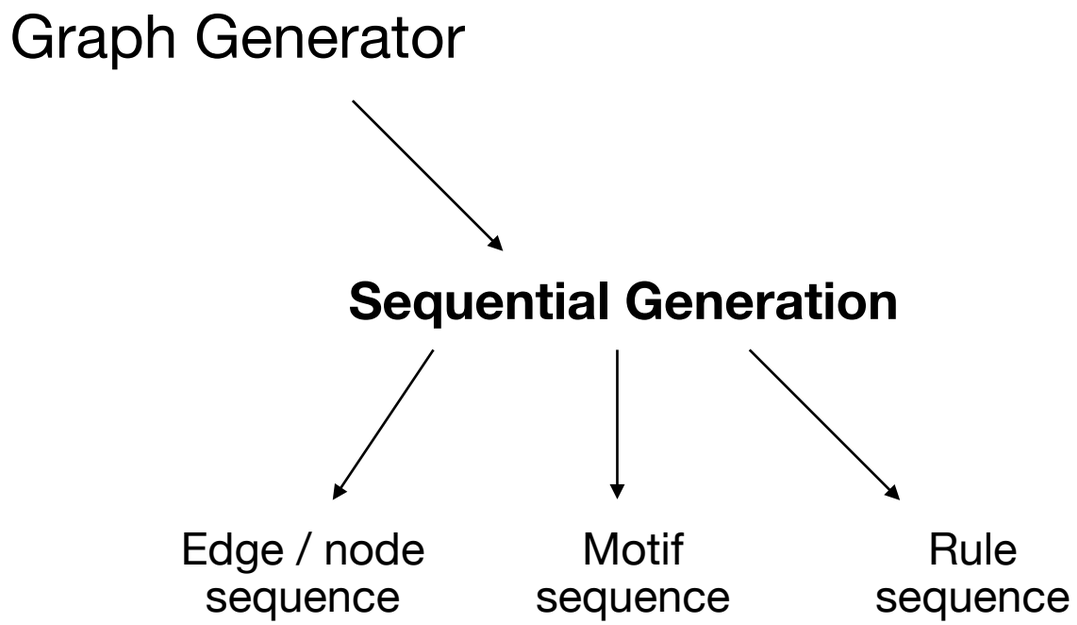
Sequential generation











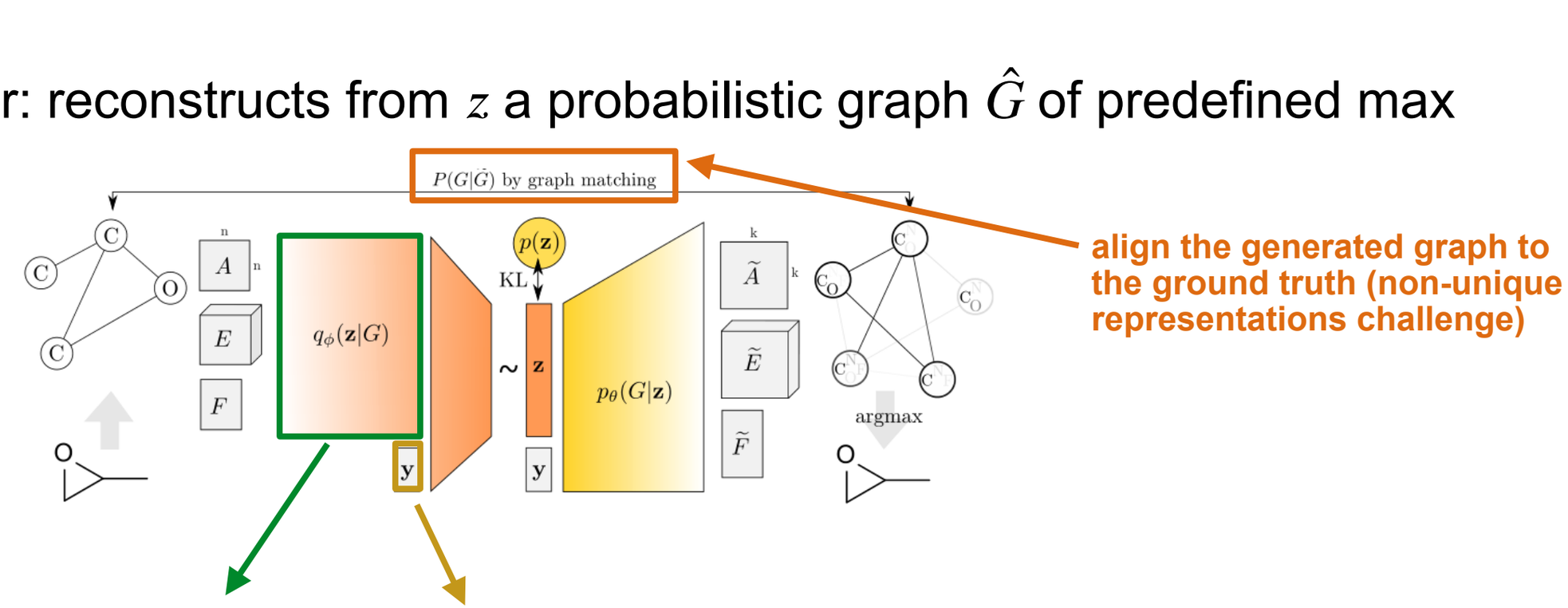
https://arxiv.org/abs/2007.06686

Today's lecture

- Introduction into deep probabilistic graph generative models
- **Main architectures**
- Applications
- Open discussion/ Feedback on the class

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)

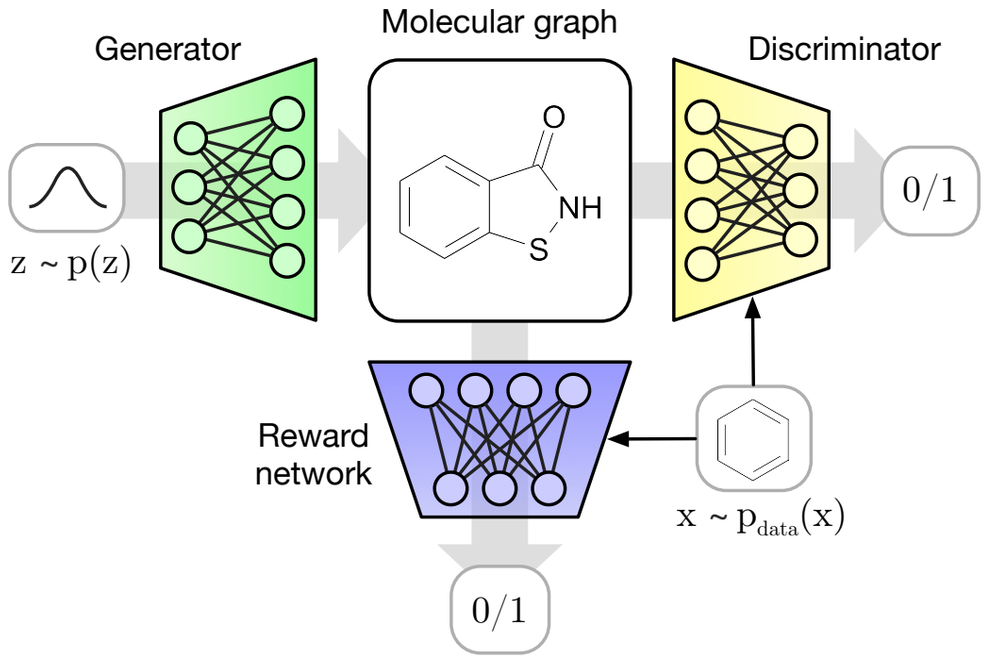
  - VAEs are only feasible in constrained domains (e.g. molecules)
  - Typically small graphs with ~40 nodes

- The max size of the graphs must be predefined

- Graph matching is required

GANs: MolGAN

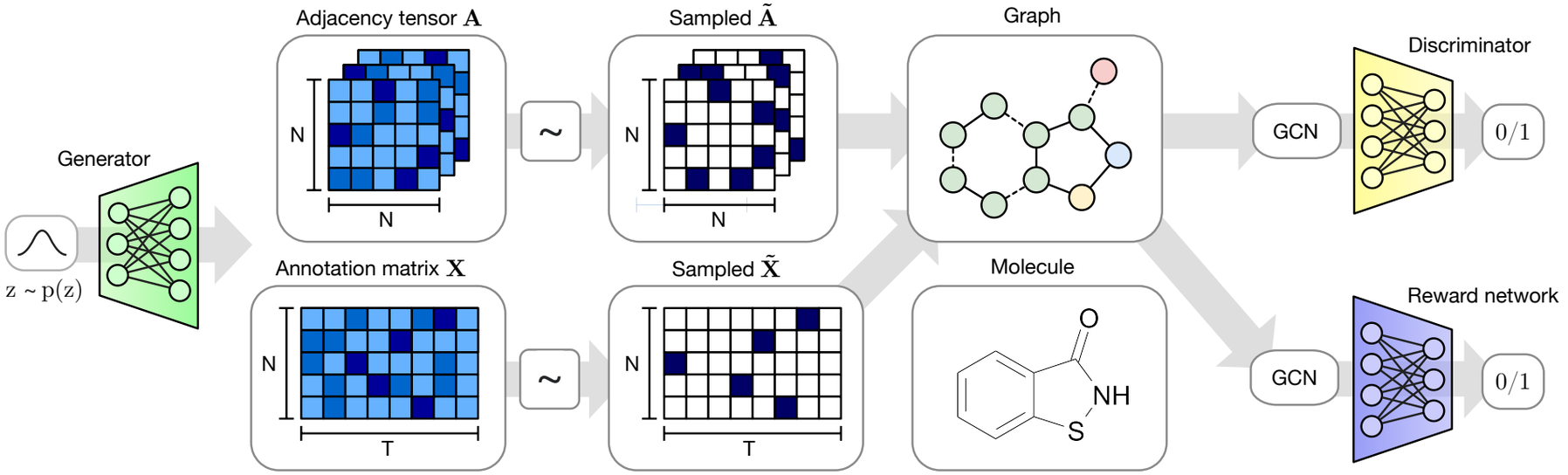
- An implicit, likelihood-free generative model
- Directly generates graphs via learned node and edge likelihoods



- 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

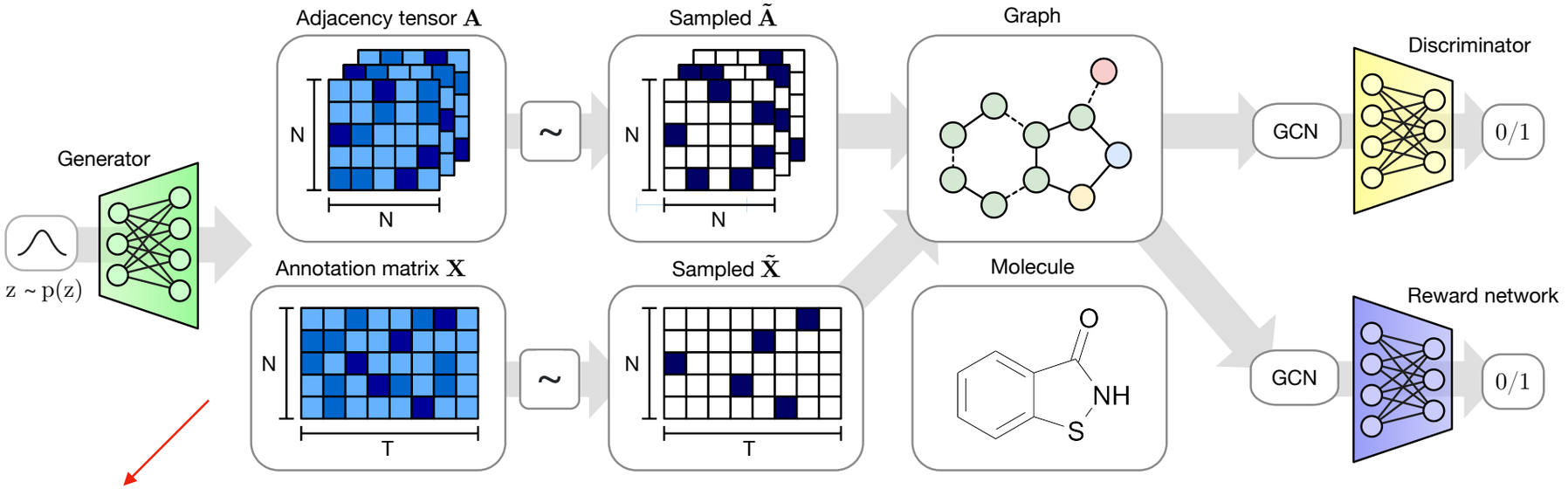
[Cao'18]

GANs: MolGAN in details



[MolGAN, De Cao et al. 2018]

GANs: MolGAN in details

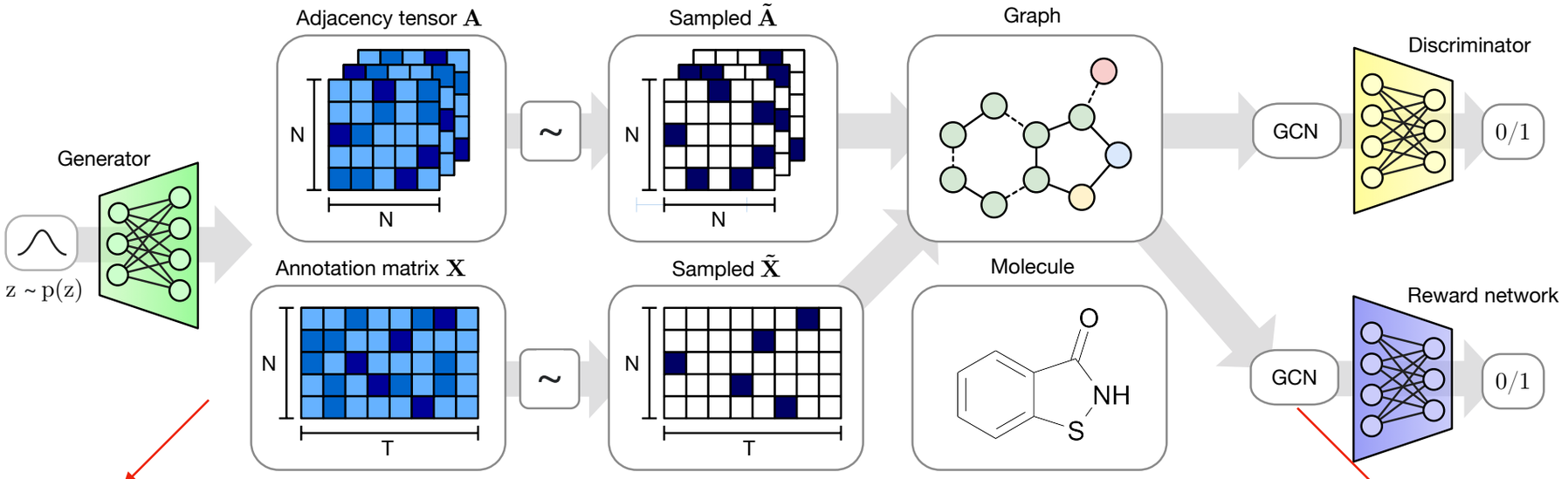


One shot generation

- 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]

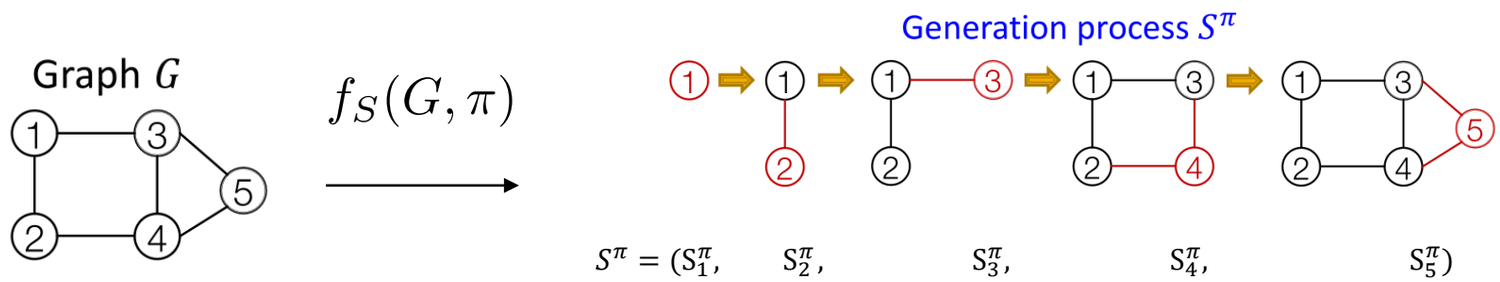
Practical considerations in graph GANs

- ✓ 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



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

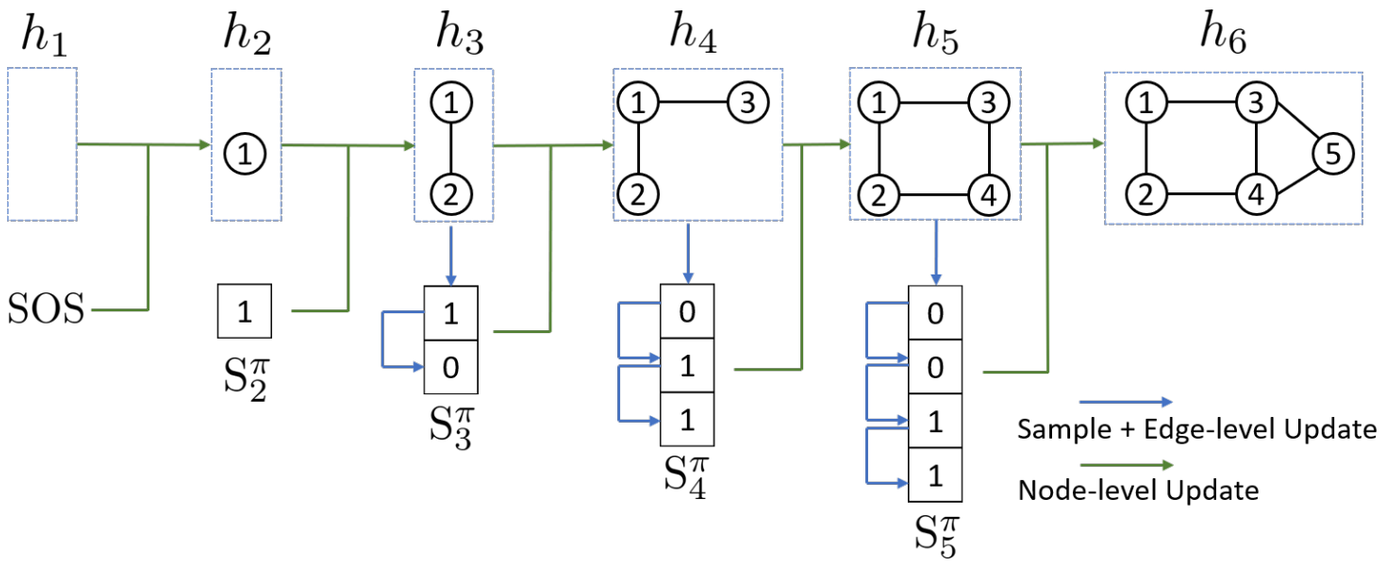
$$p(S^{\pi}) = \prod_{i=1}^{n+1} p(S_i^{\pi} | S_1^{\pi}, ..., S_{i-1}^{\pi})$$

[https://arxiv.org/abs/1802.08773]

Modeled with RNNs

GraphRNN

- Need to model two processes:
  - GraphRNN has two RNNs: node-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

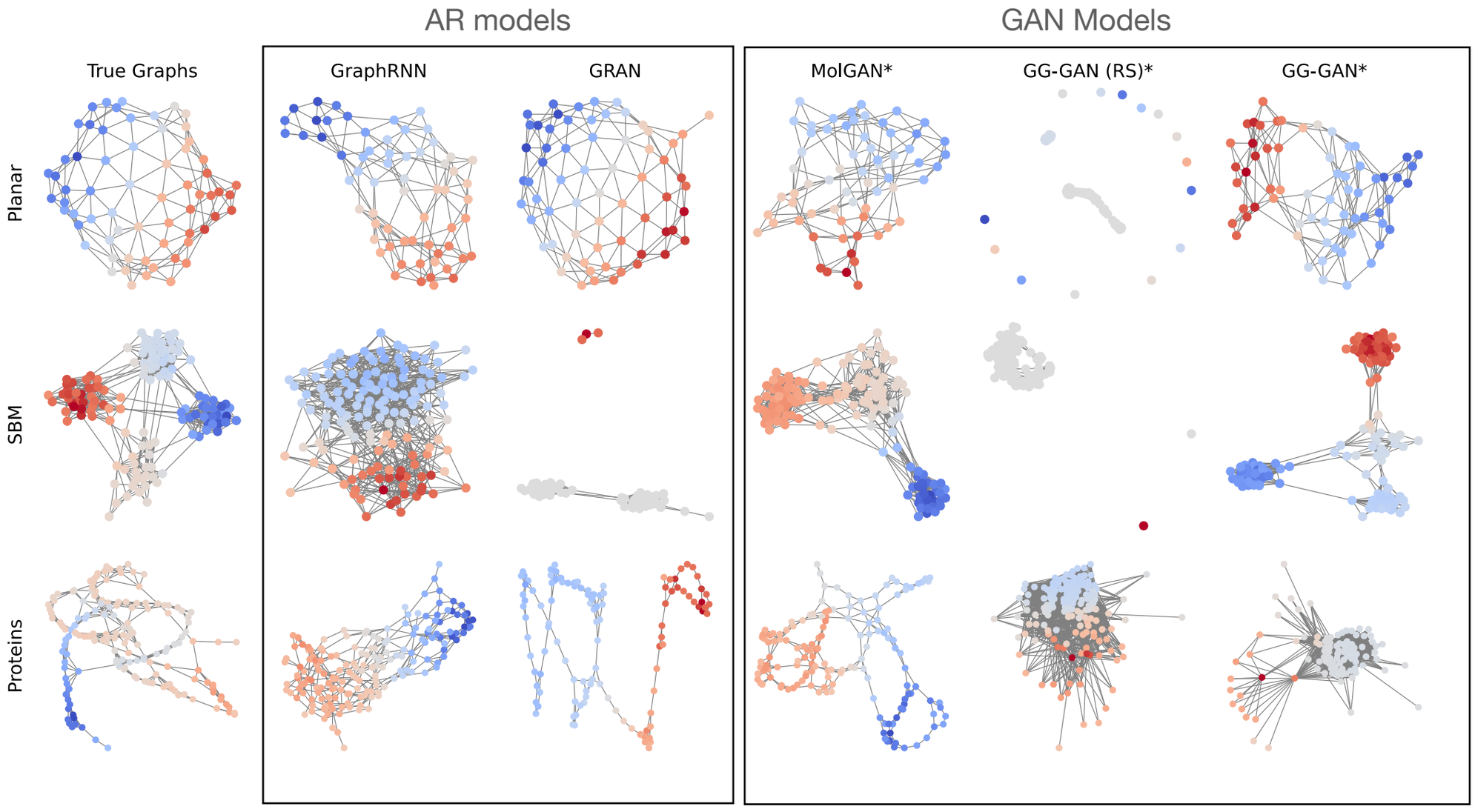


[GraphRNN, You et al. 2018]

GraphRNN limitations

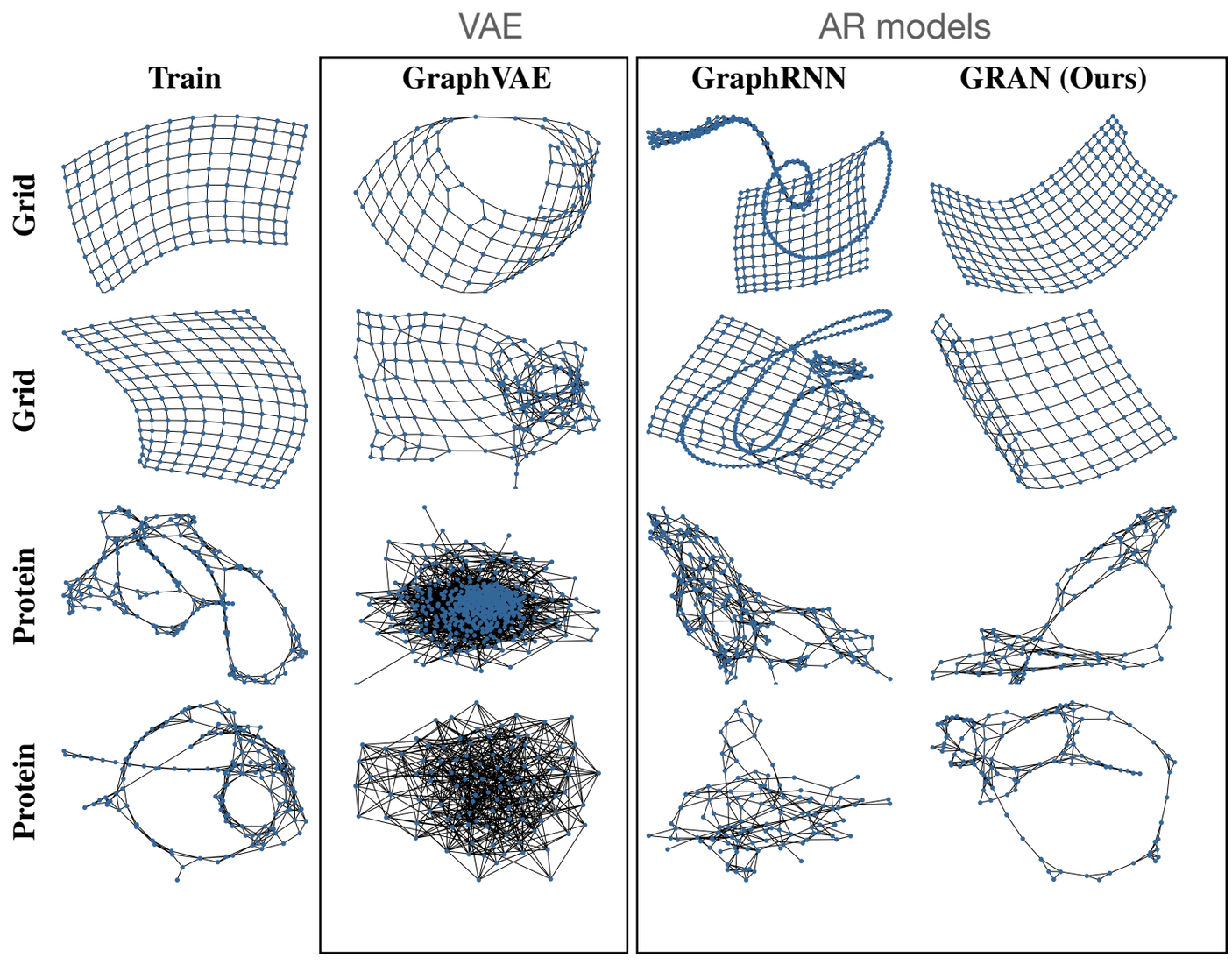
- It generates unrealistic artefacts when trained on samples of grids
- It can be difficult to train and scale due to the need to back propagate through many steps of RNN recurrence
- It requires node ordering: struggling with permutation invariance

Some comparisons



[Martinkus et al., 2022]

Some comparisons



[Liao et al., 2019]

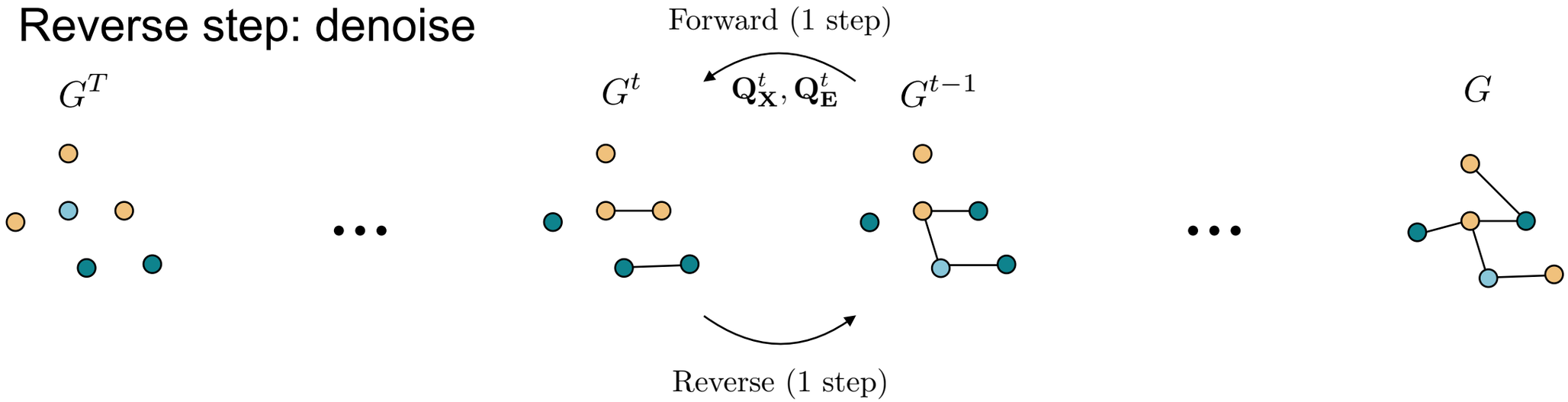
Diffusion models on graphs

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

[DiGress, Vignac et al., 2023]

Diffusion models on graphs

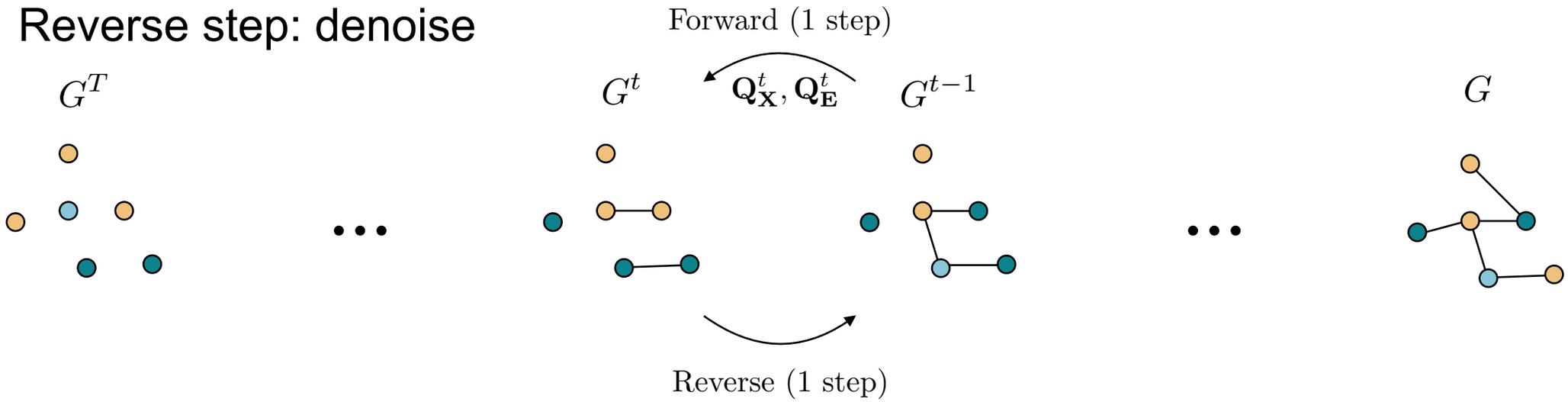
- Two main processes:
  - Forward step: add noise
  - Reverse step: denoise



[DiGress, Vignac et al., 2023]

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)

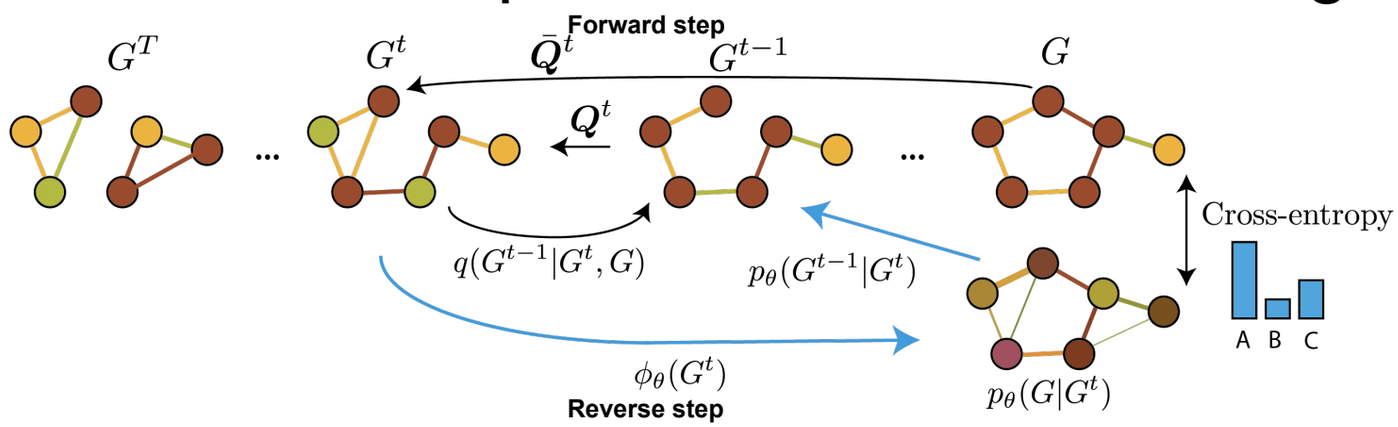
Forward: noise applied independently to each node and edge using $\mathbf{Q}^t_X$ and $\mathbf{Q}^t_E$

Reverse: use a denoising graph neural network - GNN_θ

[DiGress, Vignac et al., 2023]

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

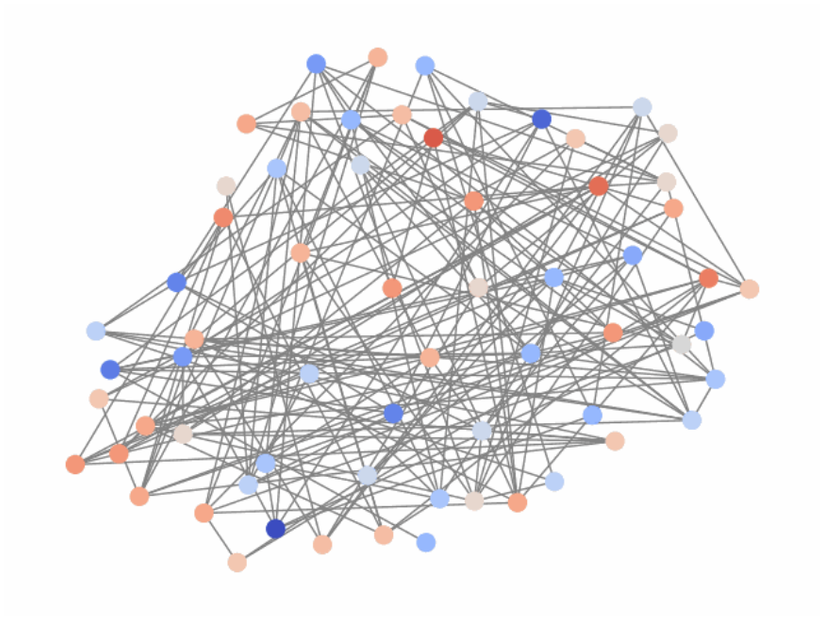
$$\boldsymbol{Q}_X^t = \alpha^t \boldsymbol{I} + \beta^t \, \mathbf{1}_a \boldsymbol{m}_X'$$

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

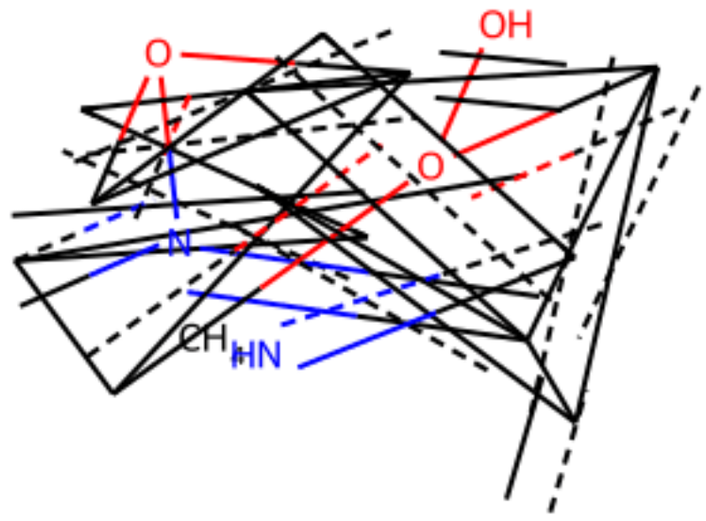
[DiGress, Vignac et al., 2023]

Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)

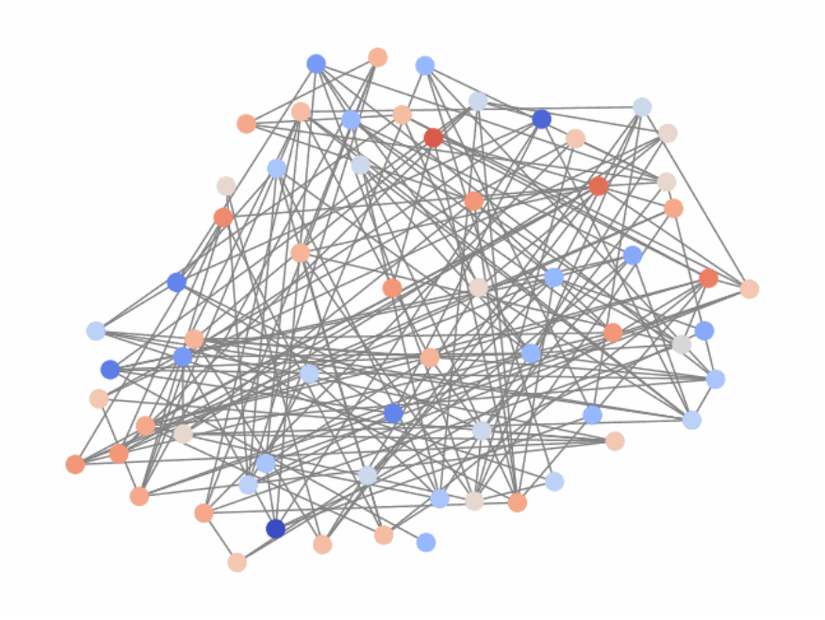


Real World Applications:
Molecular Generation

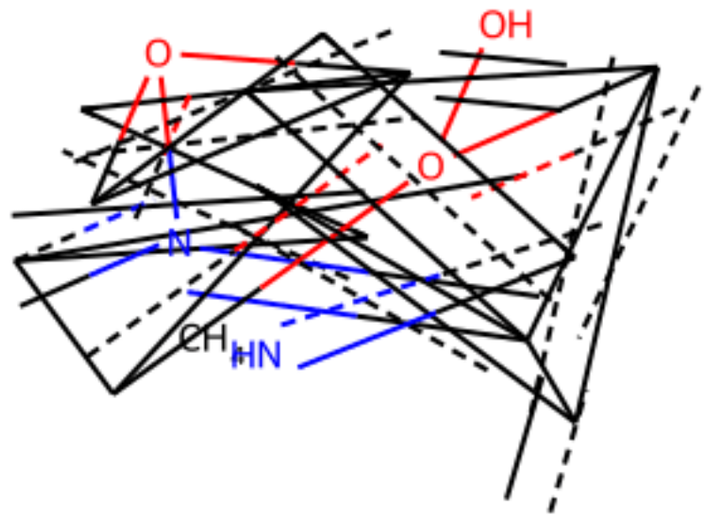


Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)

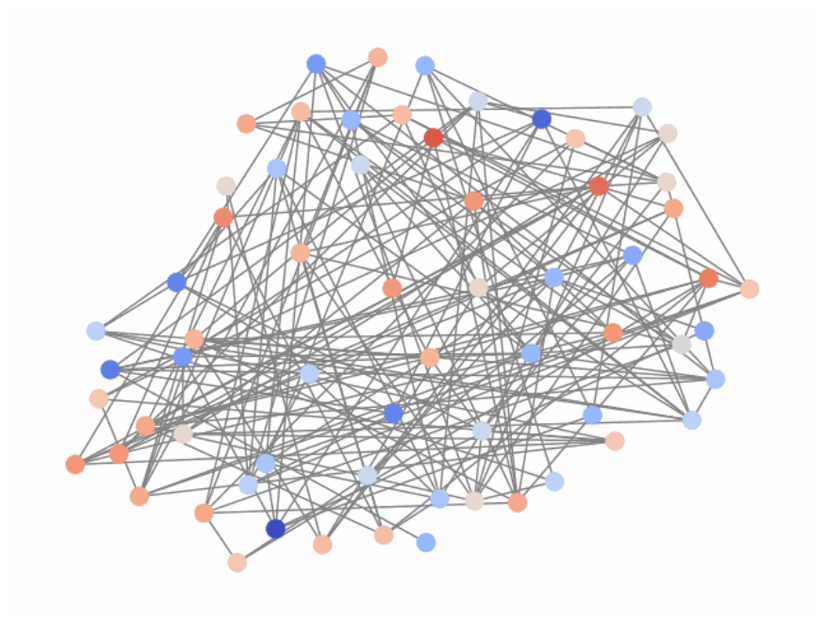


Real World Applications:
Molecular Generation

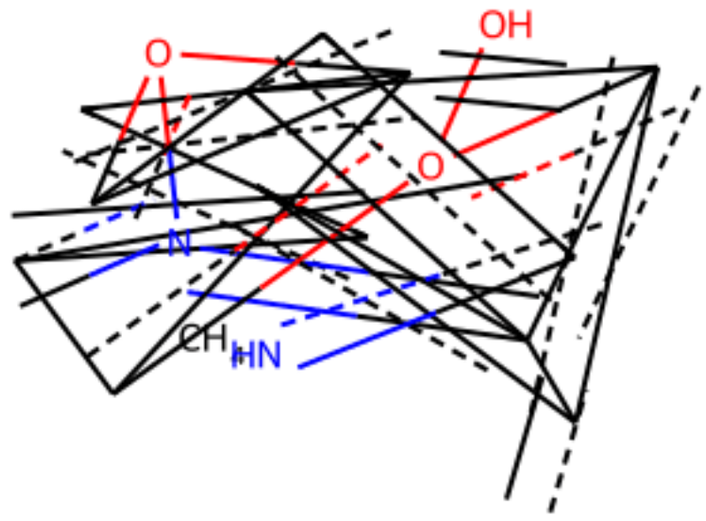


Illustrative example: Generation of new samples

Synthetic Graphs
(Planar)



Real World Applications:
Molecular Generation

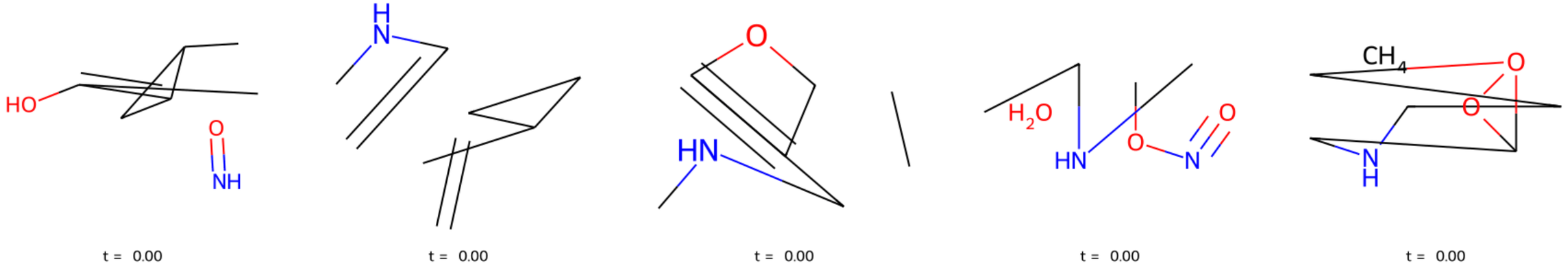


Today's lecture

- Introduction into deep probabilistic graph generative models
- Main architectures
- **Applications**
- Open discussion/ Feedback on the class

Molecular Generation

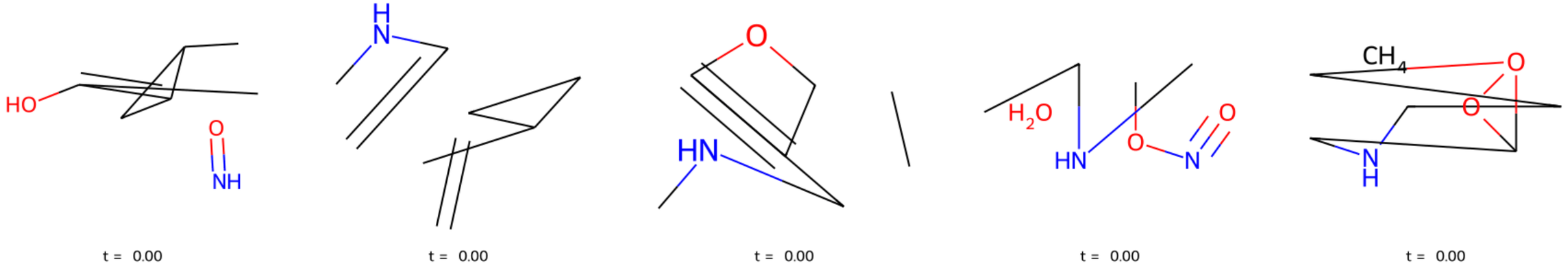
- Enables exploration of large, unstructured chemical spaces
- **Accelerate drug discovery and material design:** generate chemically valid molecular graphs beyond known molecules



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

Molecular Generation

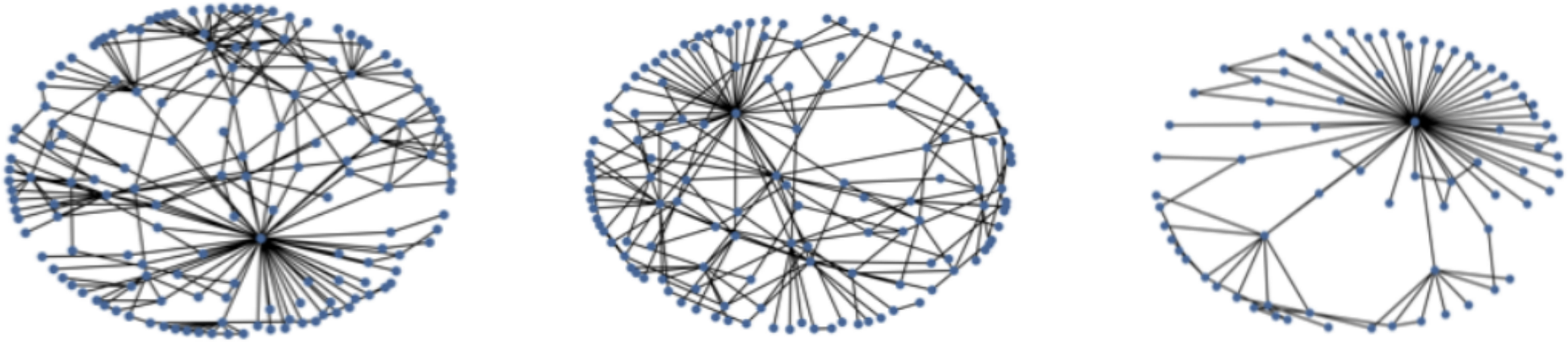
- Enables exploration of large, unstructured chemical spaces
- **Accelerate drug discovery and material design:** generate chemically valid molecular graphs beyond known molecules



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

Network Simulation

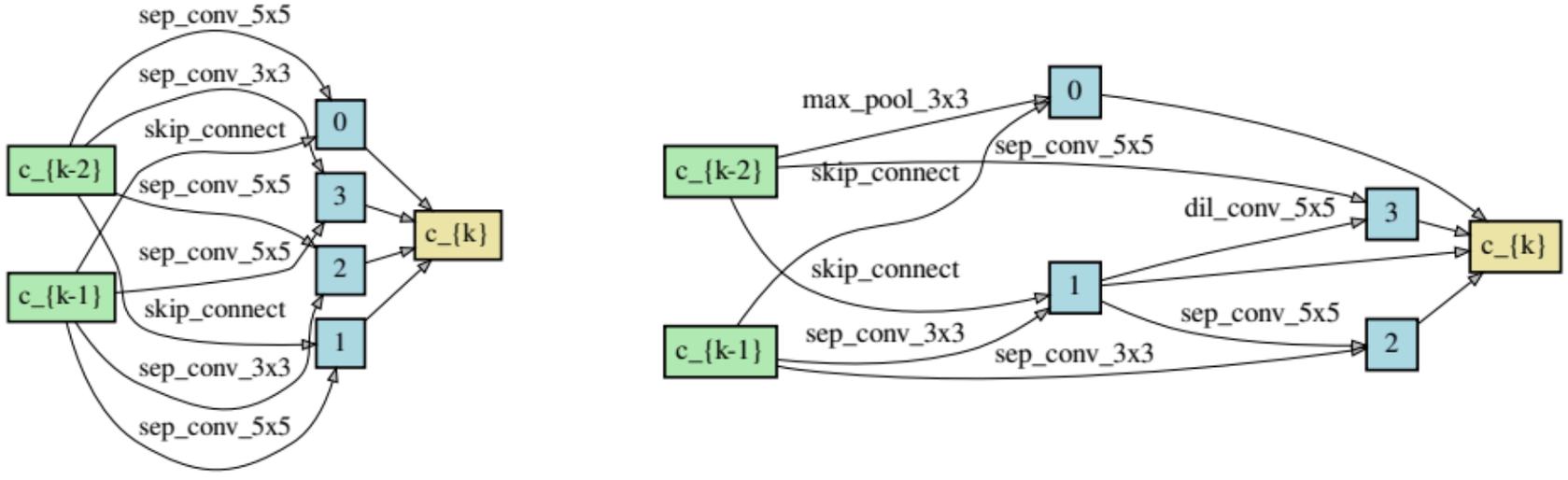
- Create synthetic graphs that **mimic real-world network statistics** (e.g., degree distribution, clustering)
- Useful for **benchmarking algorithms and studying social or information diffusion without privacy concerns**



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

Neural Architecture Search

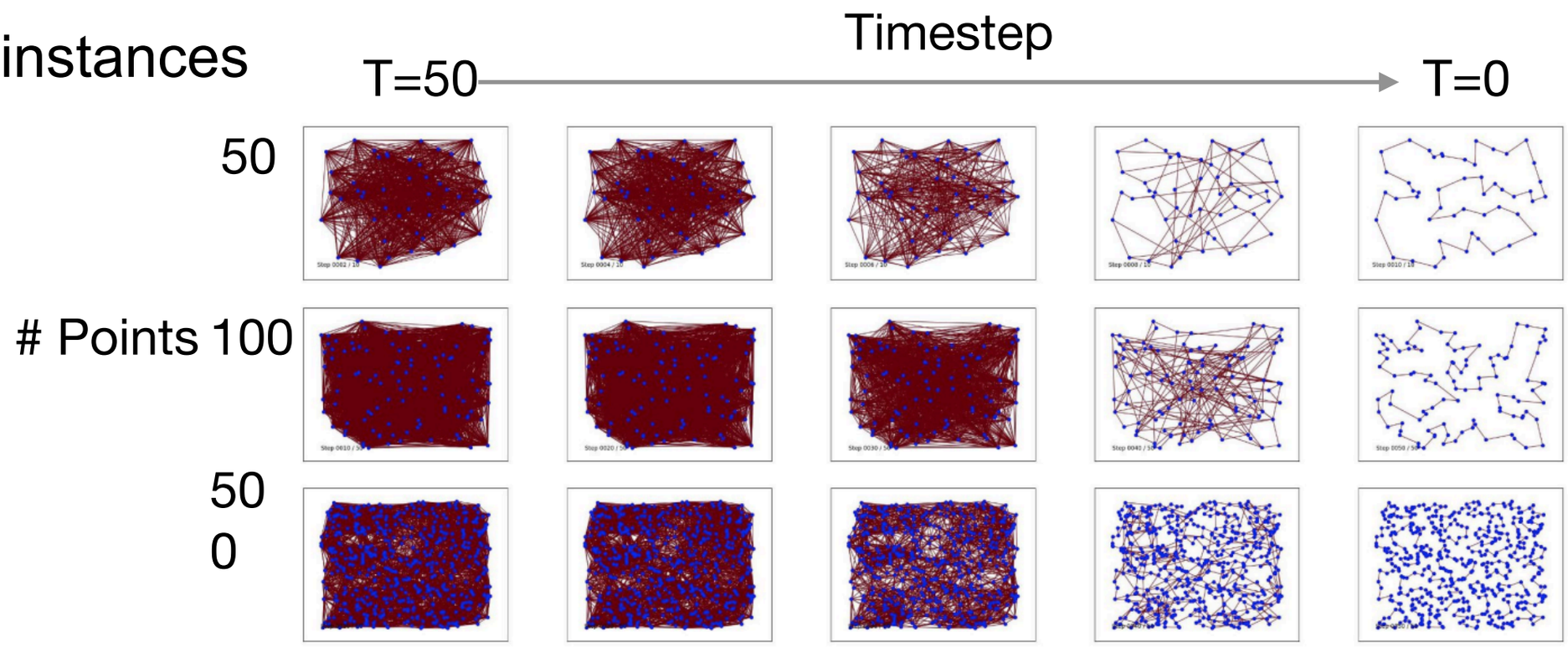
- Represent neural networks as **directed acyclic graphs** to **explore architectural design spaces**
- Graph generation enables **automated search for performant models** under multiple constraints (e.g. latency, accuracy)



[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]

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!

Take home messages

Model Type	Advantages	Limitations
Auto-Regressive	• Sequential generation fits RL frameworks • Supports reward-based control	• Requires node ordering • Costly for large graphs • Weak global structure
VAE-Based	• Flexible latent space • Control via loss or property mapping • Enables latent space optimization	• Blurry outputs • Latent-property alignment can be unreliable • Struggles with complex graphs
GAN-Based	• Enables conditional generation via property discriminators • Supports structure-aware (e.g., motif) modeling	• Training instability • Generator design complexity • Sensitive to node permutations
Diffusion-Based	• High-quality graph and feature generation • Captures topology and node features jointly	• High computational cost • Controllability still evolving • Requires complex training (score-based)

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

Today's lecture

- Introduction into deep probabilistic graph generative models
- Main architectures
- Applications
- **Open discussion/ Feedback on the class**

Thank you!