

PAPER • OPEN ACCESS

How do probabilistic graphical models and graph neural networks look at network data?

To cite this article: Michela Lapenna and Caterina De Bacco 2026 *J. Phys. Complex.* **7** 015008

View the [article online](#) for updates and enhancements.

You may also like

- [Highly efficient wavefront control based on extremely anisotropic materials](#)
Weicong He, Shan Zhu, Chuanjie Hu et al.
- [An electromagnetic wave attenuation superposition structure for thin-layer plasma](#)
Wenyuan ZHANG, , Haojun XU et al.
- [Phase-Gradient Metasurfaces Based on Local Fabry-Pérot Resonances](#)
Yanyan Cao, , Bocheng Yu et al.



PAPER

OPEN ACCESS

RECEIVED

10 July 2025

REVISED

24 December 2025

ACCEPTED FOR PUBLICATION

28 January 2026

PUBLISHED

12 February 2026

Original content from this work may be used under the terms of the [Creative Commons Attribution 4.0 licence](#).

Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.



How do probabilistic graphical models and graph neural networks look at network data?

Michela Lapenna^{1,*} and Caterina De Bacco^{2,3,*} ¹ Department of Physics and Astronomy, University of Bologna, Via Irnerio 46, Bologna 40126, Italy² Max Planck Institute for Intelligent Systems, Cyber Valley, Max-Planck-Ring 4, Tübingen 72076, Germany³ Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, Delft 2628, The Netherlands

* Authors to whom any correspondence should be addressed.

E-mail: michela.lapenna4@unibo.it and c.debacco@tudelft.nl**Keywords:** probabilistic graphical models, graph neural networks, link prediction, interpretability

Abstract

Graphs are a powerful data structure for representing relational data and are widely used to describe complex real-world systems. Probabilistic graphical models (PGMs) and graph neural networks (GNNs) can both leverage graph-structured data, but their inherent functioning is different. The question is how do they compare in capturing the information contained in networked datasets? We address this objective by solving a link prediction task and we conduct three main experiments, on both synthetic and real networks. The first focuses on how PGMs and GNNs handle input features, considering both their type and dimensionality. The second investigates their robustness to noise injected either into node features or graph edges, while the third investigates the impact of graph heterophily. PGMs do not necessarily require features on nodes, while GNNs cannot exploit the network edges alone, and the choice of input features matters. We compare standard GNNs architectures with three PGMs models, variants of stochastic block models. We find that GNNs are outperformed by PGMs when input features are low-dimensional or noisy, mimicking many real scenarios where node attributes might be scalar, missing or corrupted. When high-quality, high-dimensional input features are available and the graph is assortative, both methods can achieve comparable performance on average, but with GNNs outperforming PGMs on a fold-by-fold basis on real networks. Then, we find that PGMs are more robust than GNNs when the heterophily of the graph is increased. Introducing noise into the edges amplifies these trends: the GNN advantage on real assortative graphs diminishes, while the regimes in which PGMs outperform GNNs become even more pronounced. Finally, to assess performance beyond prediction tasks, we also compare the two frameworks in terms of their computational complexity and interpretability.

1. Introduction

Probabilistic graphical models (PGMs) combine principles from graph theory and probability theory to model complex dependencies among random variables [1]. They offer a structured way to encode prior knowledge and incorporate uncertainty, enabling efficient learning and inference in probabilistic terms. In their standard framework, they are implemented with linear models and take a limited amount of parameters as input. These models are usually designed from interpretable mechanistic principles guided by domain-knowledge, which makes them inherently interpretable. For instance, the stochastic block model (SBM) [2–5] is a popular example designed to represent networks where the mechanism guiding edge formation is assumed to be driven by community structure. The learned latent community memberships are parameters that can be directly interpreted to cluster similar nodes, and can be used to solve link prediction tasks.

On the other side, graph neural networks (GNNs) are designed to extend traditional neural networks to operate on graph-structured data [6–8]. Their main focus is on solving prediction tasks, taking advantage of the input relational data. They do so by learning representations on nodes by iteratively aggregating information among local neighborhoods, using message-passing algorithmic updates [9]. They make use of non-linearities and a large number of parameters to learn complex and expressive node representations. These can then be used for tasks such as node classification, graph classification, and link prediction.

While their main focus is fundamentally different, they are often used to perform similar tasks. For instance, link prediction is a predictive task that is often adopted in PGMs as a model validation routine. Similarly, the node representations learned by GNNs are often used to cluster nodes and efforts are made to interpret them *a posteriori*. However, in the literature we lack a systematic empirical comparison of these two paradigms under a unified experimental framework. Hence, in this work we compare them on common grounds aiming at exploring their strengths and weaknesses, by providing insights into the role of input features and graph structure.

A fundamental difference between PGMs and GNNs lies in their handling of input features. PGMs do not necessarily require node attributes, as they can rely solely on the graph's edge list. This allows PGMs to function effectively even when node attributes are sparse or unavailable. However, if node features are available and informative, there is no unique recipe for PGMs to incorporate them and exploit their information. Instead, one has to make ad-hoc modeling extensions to be able to accommodate them as input. In contrast, GNNs' message-passing algorithm explicitly requires as input node features, also when they may not be available or informative. In this work we consider three types of PGMs, all variants of the SBM, and standard GNN architectures, and evaluate how their performance is affected by performing experiments where we tune the informativeness, the dimensionality and the availability of input features.

Then, we investigate the robustness of the models when increasing the heterophily of the graph, a structural measure that has been investigated in applications of both modeling approaches and has been shown to increase the difficulty of inference tasks. For this, we consider networks with community structures of different types, from highly cohesive communities to more dispersed and interconnected ones. Since GNNs leverage the structural locality in graphs, we expect that high mixing between communities makes it harder for the model to generalize, as it has been already discussed in [10–12]. In addition to varying the level of heterophily, we also corrupt the graph connectivity by injecting noise into the edges, to evaluate robustness against structural perturbations.

To provide a comprehensive evaluation, we conduct experiments on both synthetic and real-world datasets. Synthetic datasets offer controlled environments where we can manipulate network properties and systematically study the models' behaviors under various conditions. Real datasets, on the other hand, typically come equipped with informative feature-engineered vectors as input to the nodes. By testing on both types of datasets, we aim to gain a holistic understanding of each model's strengths and limitations across different contexts. Table 3 provides a high-level summary of our main findings comparing PGMs and GNNs.

2. Related work

Our aim is to directly compare PGMs and GNNs as distinct frameworks for handling network data. This comparison is lacking in the literature, as these two types of approaches usually concern different scientific communities, who adopt distinct languages and distinct model validation criteria, making comparisons hard [13]. While PGMs are occasionally used as baselines in machine learning works [14, 15], these comparisons are far from systematic. Similarly, existing studies [15, 16] highlight specific performance gaps when comparing the two approaches, but do not provide an extensive comparison. Our primary contribution is to conduct a systematic comparison of PGMs and GNNs by contrasting their inner frameworks and empirically examining how they use input data, graph structure and node features.

While several studies have explored hybrid approaches combining the strengths of PGMs and GNNs [17–20], such as leveraging PGMs to enhance the explainability of GNNs [18, 20] or employing GNNs to improve the scalability and efficiency of PGMs [17, 19], we think that a direct comparison between these two frameworks would provide deeper insights into their respective advantages and limitations. This will ultimately help identify more principled ways to combine them or develop models that better leverage the strengths of both.

Our investigation is timely, as it addresses the growing recognition of specific GNN limitations and a renewed interest in the principled insights PGMs can offer to enhance GNNs [13]. Prior work has

shown that GNNs often struggle when provided with insufficiently informative input features and when the graph structure is heterophilic, as they cannot fully leverage graph topology alone to achieve robust performance and strongly rely on input features [21, 22]. Moreover, node clustering is often tackled in an unsupervised regime, where node labels are not available as input. In this setting, GNNs have exhibited weaker performance compared to supervised or semi-supervised approaches [16]. Notably, even in a semi-supervised setting, GNNs have been outperformed by the contextual SBM (CSBM) [15] on a node classification task, underscoring the need for a broader investigation.

Finally, understanding when GNNs offer a clear advantage over simpler or more interpretable alternatives remains an underexplored question [23]. By addressing this gap, our study seeks to provide both conceptual clarity and practical guidance, helping practitioners make more informed decisions about model selection and deployment when working with network datasets.

3. Motivation for link prediction

We choose to compare the two frameworks on a link prediction task. Link prediction is a fundamental task in network analysis, where the goal is to infer missing links or predict future connections between nodes in a graph. Link prediction serves our objective of comparing PGMs and GNNs better than node and graph classification because it directly aligns with both models' core capabilities, thus allowing for a fairer comparison.

On one hand, differently from a node classification task, link prediction does not require explicit labels on the nodes, making it less reliant on external annotations. PGMs are in fact often deployed on input data that only contain network edges, with no extra information on nodes, e.g. attributes. Also, node classification often assumes that network structure is correlated with node labels, to be able to learn from both types of input data. This makes it less general than link prediction, as it relies on observing meaningful node labels, which should not be taken for granted in real datasets, as proved in several works where extra information on nodes was shown not informative in learning from graphs [24–27]. On the other hand, graph classification involves predicting a single label for an entire graph. While this is useful for understanding a graph as a whole, it aggregates information across nodes and edges, which can obscure model-specific performance nuances. Also, graph classification is often more domain-specific, focusing on tasks like molecule classification in chemistry or document categorization, while link prediction is a more universal graph problem valid across domains.

Link prediction is often deployed as a model validation task in PGMs, in particular in clustering nodes (e.g. community detection problem) in the absence of ground truth (GT) information. Furthermore, as link prediction often involves a large number of node pairs, it is a valuable benchmark to assess scalability, efficiency, and the ability to handle sparsity or large graphs. Despite its foundational role in network analysis, link prediction often occupies a peripheral position in GNN research, making it a particularly revealing task for assessing to what extent GNNs exploit graph connectivity beyond mere message-passing.

Finally, link prediction offers the opportunity to assess a fundamental operational difference between PGMs and GNNs: whereas PGMs directly model edge probabilities, GNNs must first learn node-level representations and then map them to edge-level predictions. This distinction provides a clear test of each model's ability to capture pairwise relationships.

4. Background

PGMs and GNNs are different approaches to model network structured datasets. Both take as input data A , the network adjacency matrix of dimension $N \times N$, where N is the number of nodes. Its entry A_{ij} is the weight of an edge between nodes i and j . Sometimes, network datasets contain node features X_i , F -dimensional vectors, where F is the number of features. PGMs may take them as input, but they are not equipped by default to use them. This is in contrast with GNNs, that instead require a $N \times F$ input matrix X to train the model.

We imagine a scenario when a practitioner is willing to analyze a network problem and its corresponding dataset, and is agnostic on the model family GNN vs. PGM. Hence, we primarily consider off-the-shelf implementations of various main models in each of these families, and refrain from using models specifically designed for one main task. This also mimics the realistic scenario where the practitioner does not know in advance if their dataset exhibits a specific structure. We make one exception by including a heterophilic-adapted GNN to investigate in more detail results on heterophilic or disassortative datasets.

4.1. PGM models

PGMs describe the dependencies among nodes via the edges of the network in a probabilistic way. We focus here on a specific class of PGMs, namely generative models with latent variables, based on the SBM [2–5]. These models use latent variables θ that control explicitly the probability of observing a given edge weight A_{ij} , by defining a joint distribution $P(A, \theta)$.

Approach. The probability of observing a given edge weight A_{ij} is specified by a likelihood function $P(A_{ij} | \theta)$, that defines the joint distribution $P(A_{ij}, \theta) = P(A_{ij} | \theta) P(\theta)$. Here, $P(A_{ij} | \theta)$ models the formation of edges given the latent variables, while $P(\theta)$ is a prior that expresses assumptions about the latent variables before observing any data.

Input. The model takes as input the observed network structure, represented by the adjacency matrix A . Optionally, node attributes (or covariates) X can be incorporated with ad-hoc modeling extensions.

Output. Using statistical inference techniques, the model produces estimates of the latent variables θ , either as point estimates $\hat{\theta}$ or full posterior distributions $P(\theta | A)$. These are then used to estimate the likelihood of a given edge weight A_{ij} as:

$$P(A_{ij} | \hat{\theta}) \quad \text{or} \quad \mathbb{E}_{P(\theta|A)} [P(A_{ij} | \theta)] \quad .$$

Learnable parameters. These are interpretable latent variables θ that govern the network's generative mechanism. For models assuming an underlying community structure, such as the SBM and the ones considered here, these typically include:

- **Node memberships (u_i):** a K -dimensional vector for each node i , where K is the number of latent communities, defining each node's membership to these communities (often with positive entries for interpretability). For directed networks, distinct *outgoing* and *incoming* membership vectors u_i and v_i are used, while for undirected graphs we set $u_i = v_i$. The memberships can be collectively encoded in matrices $U \in \mathbb{R}^{N \times K}$ and $V \in \mathbb{R}^{N \times K}$.
- **Affinity matrix (w):** a $K \times K$ matrix controlling the density of edges between and within communities. Its structure determines whether connections are mainly within (*assortative*) or across (*disassortative*) communities.

The probability of observing an edge weight A_{ij} is then expressed as:

$$P(A_{ij} | \theta) = P(A_{ij} | u_i, v_j, w), \quad (1)$$

In this work, we consider three variants of the SBM, which differ in the parameterization of θ and in their statistical inference methods. The first two, MULTITENSOR [28] and MTCOV [26], rely on maximum likelihood estimation via the Expectation-Maximization (EM) algorithm [29, 30], yielding point estimates of the latent variables. The number of communities K is a fixed model parameter that must be specified in advance and selected through cross-validation. The third model, BNP [31], adopts a Bayesian non-parametric formulation and uses Markov chain Monte Carlo (MCMC) sampling [32, 33] for inference, estimating the full posterior distribution of the latent variables. In this context, the term non-parametric means that the number of communities K is not fixed *a priori* but is inferred directly from the data during the inference process.

Among these, MTCOV is the only model that can also incorporate node attributes as input, allowing us to assess the impact of using this additional information alongside the network adjacency matrix. In appendix A.1 we provide more details on these models.

4.2. GNN architectures

GNNs [6–8] leverage neural network architectures to learn node embeddings from network-structured data. Their core underlying mechanism is based on message-passing [34], in which nodes iteratively aggregate information from their neighbors across layers guided by the relational structure encoded in the adjacency matrix.

Approach. In this framework, each node updates its embedding $h_i^{(\ell)}$, a vector of dimension F_ℓ , by aggregating the embeddings $h_{\partial_i}^{(\ell)}$ of its neighbors at each layer ℓ in a message-passing fashion [9]:

$$h_i^{(\ell)} = \phi^{(\ell)} \left(h_i^{(\ell-1)}, \bigoplus_{j \in \partial_i} \psi^{(\ell)} \left(h_i^{(\ell-1)}, h_{\partial_i}^{(\ell-1)} \right) \right), \quad (2)$$

where $h_{\partial i}^{(\ell-1)} = \{h_j^{(\ell-1)} | j \in \mathcal{N}_i\}$ contains the embeddings of the neighbors $\mathcal{N}_i = \{j | A_{ij} > 0\}$; $\psi^{(\ell)}$ is a learnable message function (typically a linear map or a small Multi-Layer Perceptron (MLP)) that computes messages from neighbors to node i ; and $\oplus$ is a permutation-invariant aggregation operator (e.g. sum, mean or max). The update function $\phi^{(\ell)}$ combines the previous embedding of node i with the aggregated neighborhood message, e.g. through concatenation, and then applies a non-linear transformation, such as an MLP, to produce the new embedding $h_i^{(\ell)}$. The embeddings in the first layer are initialized using the input node features $h_i^{(0)} = X_i$. The choices of the functions $\psi^{(\ell)}$, $\oplus$, and $\phi^{(\ell)}$ differ based on the architecture. Contrarily to PGMs, they usually entail non-linearities.

Input. The model takes as input both the observed network structure, represented by the adjacency matrix A , and node attributes X . Differently from PGMs, in GNNs the features X are a necessary input and they are used to initialize the node embeddings.

Output. The output of a GNN is a matrix $H^{(L)} \in \mathbb{R}^{N \times F_L}$, where each row contains the final layer embeddings $h_i^{(L)}$ for each node. These are learned via gradient descent to optimize a given cost function, which is set depending on the desired task. Differently from PGMs, which return interpretable latent parameters (e.g. community memberships and affinity matrices), GNNs produce quantities that are not directly interpretable. These embeddings serve as the basis for downstream tasks such as link prediction, node clustering, or node classification, depending on the chosen loss function.

Learnable parameters. In addition to the node embedding vectors, GNNs learn weight matrices that parameterize the message-passing functions, determining how node embeddings are transformed at each layer. These weights, appearing inside $\phi^{(\ell)}$ and $\psi^{(\ell)}$ (and, when present, inside the aggregation function $\oplus$), map node representations from their current feature dimension, either the raw input dimension F or the previous hidden size $F_{\ell-1}$, into the new embedding dimension F_ℓ of layer ℓ . The choice of F_ℓ controls the representational capacity of the model, allowing each layer to compress, preserve, or expand the dimensionality of node embeddings as information propagates through the network.

The probability of existence of an edge (i, j) is calculated using the embeddings inferred in the final layer L by applying a scoring function to pairs $(h_i^{(L)}, h_j^{(L)})$. This has to be selected arbitrarily. In this work we use the dot product:

$$P(A_{ij} = 1 | h_i^{(L)}, h_j^{(L)}) = \hat{A}_{ij} = \sigma(h_i^{(L)} \cdot h_j^{(L)}) \quad , \quad (3)$$

where $\sigma(\cdot)$ is the logistic function, a frequent choice that matches the binary cross entropy loss used in training, but other choices have been explored in the literature, from concatenation to more complex non-linear methods [35, 36]. The final embeddings can also be used for node clustering, e.g. by applying K-means [37, 38] on them to assign cluster labels to nodes.

In our experiments, we employ four distinct architectures, which differ in how the message-passing functions are defined. Graph attention networks (GAT) [39] use an attention mechanism inside the aggregation function; graph autoencoders (GAE) [40] and graph variational autoencoders (VGAE) [40] use an autoencoder approach to learn embeddings, with GAE being a non-probabilistic variant of VGAE; H₂GCN [41] is a GNN architecture specifically designed to handle heterophily. In appendix A.2 we provide more details on these models.

A key difference from the PGMs discussed above is that standard GNNs implicitly treat the graph as undirected during message-passing: information is aggregated symmetrically using the node's neighbors, without distinguishing between incoming and outgoing edges. This common practice is proven convenient for node classification in homophilic networks, while it may be detrimental in heterophilic settings [42]. To properly model directed graphs, the message-passing scheme must incorporate directional structure with ad-hoc choices, for example by using separate aggregations for incoming and outgoing neighborhoods [42–44], by processing source and target representations through separate graph convolutional layers [45], or by designing message functions that explicitly encode edge direction as part of the edge features [46]. We do not explore them here. Note also that the dot product used in equation (3) is inherently symmetric. As a consequence, the reconstructed adjacency matrix satisfies $\hat{A}_{ij} = \hat{A}_{ji}$, which implicitly matches the undirected assumption of standard message-passing GNNs. When using GNN architectures with directional message passing, the decoder must also reflect edge asymmetry, for instance through parametric asymmetric scores. Exploring the impact of these choices in link prediction when modeling directed graphs is an interesting direction for future work.

Table 1. Dataset statistics for the ten real datasets analyzed. N is the number of nodes; $\langle k \rangle$ is the average degree of a node; h is the edge homophily ratio (i.e. the fraction of edges connecting nodes with the same class label [41], computed only for datasets with node labels); F is the dimension of the node feature (when available).

Data	Cora	Citeseer	PPI	Peptides	Reddit	PolBlogs	Email	Chameleon	Wisconsin	Texas
Type	Citation	Citation	Biology	Biology	Information	Information	Information	Web	Web	Web
Directed	No	No	No	No	No	Yes	Yes	Yes	Yes	Yes
N	2708	3327	1750	338	228	1490	1005	2277	251	183
$\langle k \rangle$	3.90	2.74	18.47	2.02	2.20	12.77	25.44	15.85	2.05	1.77
h	0.81	0.74	—	—	—	0.91	0.36	0.23	0.21	0.11
F	1433	3703	50	9	—	—	—	2325	1703	1703

5. Experimental details

5.1. Datasets

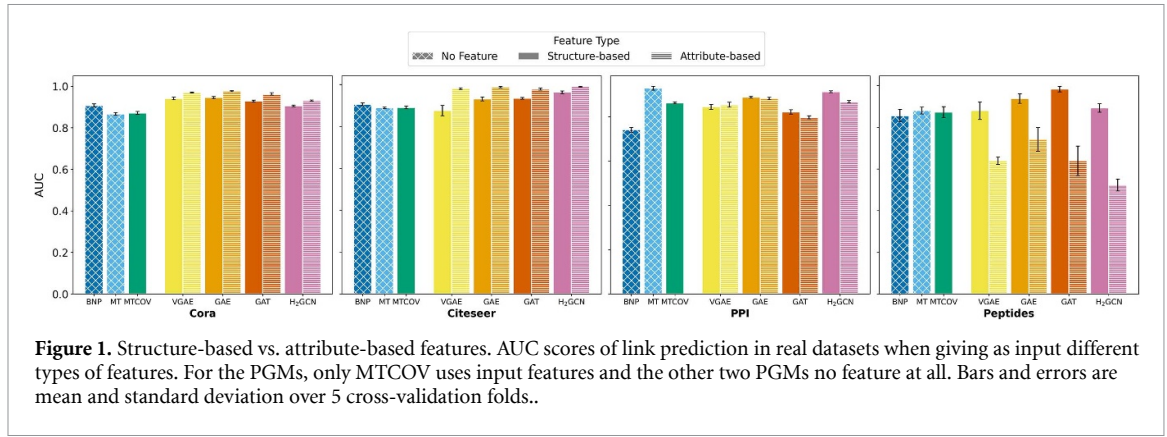
Synthetic data We generate synthetic networks with community structure using MULTITENSOR (details are provided in appendix B). We assign K -dimensional membership vectors to nodes, assuming an underlying community structure consisting of K overlapping groups. We then draw a $K \times K$ affinity matrix controlling the density of edges between pairs of nodes. This follows the generative models behind the PGMs considered here. Specifically, we generate directed networks of $N \in \{100, 1000\}$ number of nodes; $K = 5$ communities and average degree $\langle k \rangle \in \{5, 20\}$. To control the heterophily of the network, we manipulate the density of edges between nodes in different communities. Heterophilic have larger off-diagonal elements of the affinity matrix. For each combination of size and average degree, we generate one homophilic and one heterophilic network. We sample 10 networks for each parameter configuration.

Real data We use ten real datasets from different domains, of varying size, sparsity, and homophily level, as reported in table 1. Cora and Citeseer [47] are citation networks, nodes represent papers connected by citation edges and features are bag-of-word abstracts. Although the original graphs are directed, we employ the undirected versions [47] provided in PyTorch Geometric [48]. Protein–protein interaction (PPI) [49] and Peptides [50] are biological networks, edges represent interactions between proteins and similarity relationships between peptides, respectively, and features contain various protein and peptide properties. Reddit [51], PolBlogs [52] and Email [53] are information networks, describing comments between users in Reddit, links between political blogs and email communications between members of an European institution, respectively. Chameleon, Wisconsin and Texas [54] are web page–page networks, where edges represent hyperlinks between web pages and node features are bag-of-words representations. Seven of these datasets contain additionally labels on nodes, these are used to estimate the level of homophily.

5.2. Cross-validation details

For each input network, we split the edges in training and test sets, and we employ a 5-fold cross validation with a 80% – 20% split in training and test, respectively. For GNNs, we extract from the training set a validation set, corresponding to 10% of the entire dataset. We use the area under the receiver operating curve (AUC) as link prediction metric. This tells the probability that a randomly selected existing edge in the test set is assigned a higher probability of existing than a randomly selected non-existing edge. Its value ranges in $[0, 1]$, with 1 corresponding to perfect reconstruction and 0.5 to random chance.

Hyperparameters tuning and algorithmic implementation For each model, we have used algorithmic implementations available open source, see appendix A for details. For PGMs, we cross-validate the number of communities K in MULTITENSOR and MTCOV, as well as $\gamma \in [0, 1]$ in MTCOV, a parameter tuning the contribution of A and X in the cost function; BNP learns K automatically via MCMC, and we use 100 posterior samples, each obtained after 10 Metropolis-Hastings sweeps. For the GNN models, we tune the following hyperparameters: learning rate $\in \{0.001, 0.01, 0.1\}$; weight-decay $\in \{0.0001, 0.001, 0.01\}$; dropout $\in \{0.0, 0.3, 0.5\}$; hidden dimension $\in \{32, 64, 128\}$; and number of layers $\in \{1, 2\}$. Each GNN model was trained for 200 epochs with Adam optimizer [55] and binary cross-entropy loss. We employ an early stopping criterion and we save the weights of the model minimizing the loss on the validation set with a patience of 50 epochs.



6. The role of input features

One of the main differences between PGMs and GNNs is that the latter require node features X as input, in addition to a graph adjacency matrix A . The choice of what features to give as input impacts performance on downstream tasks [56] and it is not clear how to make this choice in general. This is particularly relevant in datasets where there is no extra information besides the adjacency matrix, and one has to come up with dummy features to feed into the GNN. For instance, in social network analysis, user demographic data may be obscured due to privacy concerns. In such cases, synthetic features derived solely from the input graph, such as node degree and total node count, can be used as node attributes [57–59]. Hence, our first step is to assess the role played by the nature of nodes features in learning.

For this, we run a first experiment where we change the type of information and the dimensionality of features given as input to GNNs and evaluate their impact on the performance of GNNs compared to that of PGMs.

We experiment with two kinds of features: (1) a *structure-based* feature, that uses information about the edges the node is linked to; and (2) an *attribute-based* feature, that uses existing node features (when these are available).

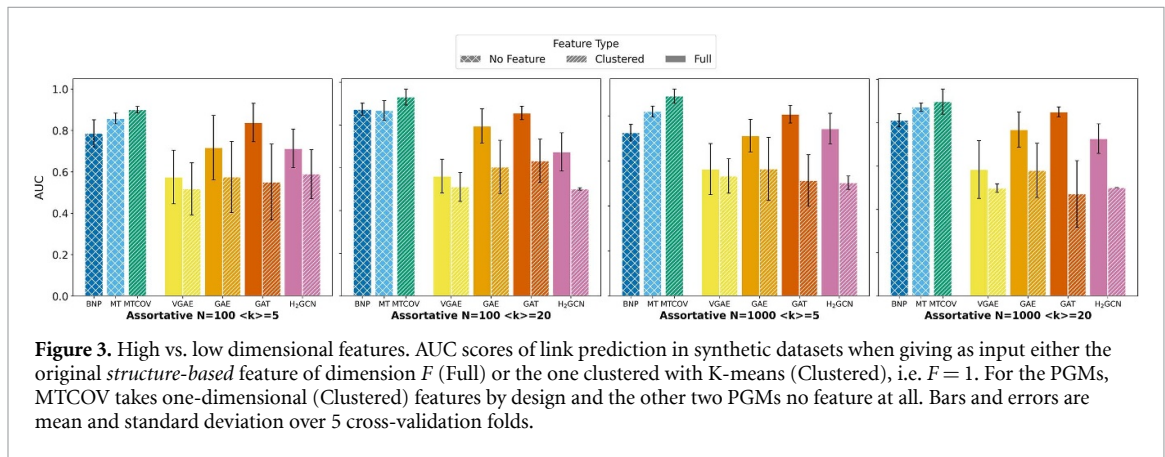
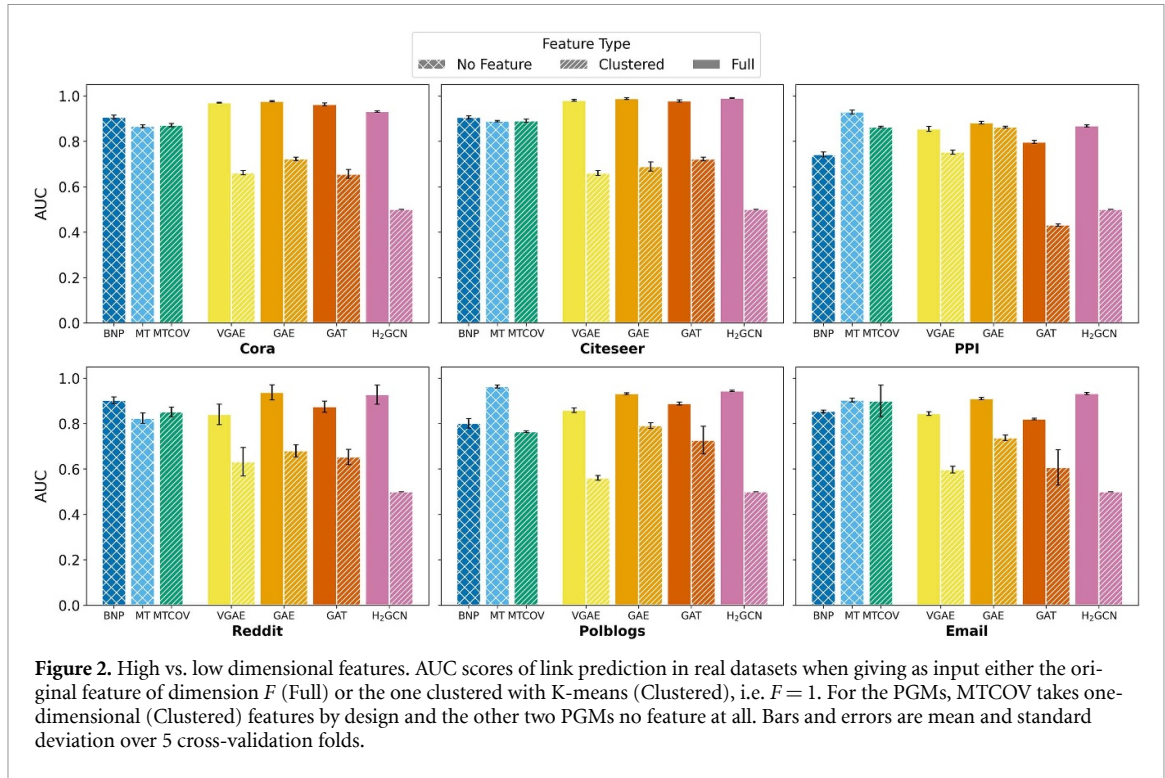
In case 1, we assign to each node i the corresponding row in the adjacency matrix $A_i = [A_{i1}, \dots, A_{iN}]$. This information is readily available as it is a quantity simply derived from the input matrix A . We stress that, when splitting the graph in training, validation and test set in our cross-validation link prediction routine, we take into account the masking of the edges, and we feed the node with the masked adjacency matrix row. Indeed, the training set feature only contains the links between nodes in the training set graph, while for validation and test we progressively include the validation and test set links.

In case 2, the features are directly available only for 7 out of 10 real datasets. For the real datasets that do not come equipped with engineered features, we assess their performance by simply feeding the *structure-based* feature.

In the case of PGMs, as input features are not necessary, we employ node features only in the model designed to take them as input, i.e. MTCOV. In this case we employ only one type of node feature, one-dimensional and *structure-based*. In synthetic data, the feature corresponds to the argmax of the GT membership vector of the node u_i . In real data, where we do not have GT memberships, we assign to each node the cluster obtained by applying K-means to the adjacency matrix row $A_i = [A_{i1}, \dots, A_{iN}]$.

We find that GNNs' performance is generally comparable in the two cases, see figure 1, except for Peptides, where AUC is much higher in the *structure-based* case. This is also the only dataset with relatively low-dimensional features ($F = 9$). Feeding the high-dimensional adjacency matrix row improves performance, which is generally comparable to that of PGMs. We find similar results for the synthetic data, see figure 3. A closer inspection of the per-fold performance (see appendix C.3) reveals that on real assortative datasets, GNNs outperform PGMs in roughly 80% of the folds, whereas on synthetic assortative graphs PGMs outperform GNNs in about 65% of the folds. This last result is expected since synthetic graphs are generated from one of the SBM variants.

This experiment shows that there are no clear advantages in utilizing an extra input feature X for link prediction. Instead, one has to be careful in using all the information available, because adding a feature can even hurt performance. Similar results have been found in previous works using PGMs [24–27]. We stress that prior studies [60–62] have proposed strategies to handle missing features in GNNs, specifically focusing on structural and positional encodings.



Motivated by the above results, we assess the impact of the feature dimensionality F by reducing it to one, clustering the feature vectors with K-means and giving the cluster label as feature. This mimics situations where the availability of node attributes is limited and one has access only to one type of information, e.g. age or income in a social network. We find that GNNs' performance is significantly reduced when the input feature is clustered and this happens for every dataset studied, see figures 2, 3 and S9. In particular, in figure 3 we observe that on synthetic assortative graphs GNN performance increases with graph size, which we attribute to the adjacency-row feature becoming higher dimensional. Instead, MTCOV, which takes as input one-dimensional features by design, shows comparable performance to that of the other PGMs and GNNs with full features. This performance is expected in the synthetic data, but we find similar results also in the real datasets as well.

This shows a potential limitation of GNNs, the inability to either properly utilize a low-dimensional feature or to discard it in favor of, for instance, a dummy high-dimensional feature like the *structure-based* one considered above. This was also observed in a node classification task [63], where it was shown how GNNs struggle in handling low-dimensional input features, and emphasized the benefits of graph-related features over manually engineered ones. In addition, in [64] it was shown that using the adjacency matrix row as an input feature substantially improves GNN performance in graph classification tasks, compared to simpler structural features such as the scalar node degree.

In practice, this issue may not occur frequently, as it is standard practice when using GNNs to employ high dimensional features, and when these are not available, one can always design them

artificially, e.g. using the row of the adjacency matrix. However, the question is what to do when a low-dimensional feature is indeed available and a practitioner expects this to be informative. This situation is not uncommon when handling network datasets. While the underlying reasons for GNNs' limited ability to leverage low-dimensional features remain unclear, assessing suitable ways to combine high and low-dimensional features to enhance prediction performance is an interesting direction for future work.

7. Robustness to noise

In the second experiment, we assess the robustness of the PGM and GNN models to noise injected into node features or into graph structure.

Noisy features. We investigate the robustness to noisy features by systematically randomizing features of randomly selected nodes and observing the impact on each model's performance. This simulates scenarios where some node attributes might be missing or noisy. We perform this experiment on the real datasets equipped with *attribute-based* features.

In the case of GNNs, we randomly shuffle the entries in the input vector of features $X_i = (X_{i1}, X_{i2}, \dots, X_{iF})$, by applying a random permutation function π to its indices: $X_i^{\text{shuffled}} = (X_{i\pi(1)}, X_{i\pi(2)}, \dots, X_{i\pi(F)})$. We apply different permutations π to each node feature.

In the case of MTCOV, instead, we replace the single scalar feature with a number extracted uniformly at random from the range of possible clusters, i.e. the number of K-means clusters of the adjacency matrix row.

We measure the impact of feature randomization on a varying percentage of nodes: 0% (original input features are intact); 50% (features are randomized for half of the nodes); and 100% (all nodes have randomized features).

We find that PGMs are more robust when increasing the percentage of nodes whose features are randomized, highlighting the stronger reliance of GNNs on input feature information, see figure 4. In particular, GAT and H₂GCN appear to be overall more sensitive to noise compared to autoencoder-based architectures (VGAE, GAE). This increased sensitivity can stem from the fact that noisy features are further propagated through attention coefficients in GAT and through higher-order neighborhood aggregations in H₂GCN, thereby amplifying the impact of the noise. Overall, the effect of noise is especially pronounced when the input features have lower dimensionality, as in Peptides ($F=9$), or when the dataset is heterophilic, as in Chameleon and Wisconsin.

The better performance of MTCOV may be explained by its ability to tune the hyperparameter γ , which controls the weight assigned to the two types of input data, adjacency matrix and node features. This means that this model is able to discard or attenuate the impact of noisy features, by suitably selecting this hyperparameter, e.g. with cross-validation. This result suggests to investigate ways to incorporate a similar mechanism into GNNs, with the goal of learning directly from data what weight to assign to structure versus node features when performing a given predictive task. One possible approach for addressing this issue is to consider feature-propagation methods to handle missing or unreliable features in GNNs [65], but this applies only under homophily and breaks down on heterophilic graphs. Hence, a main challenge is to devise methods that can be robustly applied on different underlying structures.

Noisy edges. We evaluate the robustness of the models to noise in the graph structure by corrupting the adjacency matrix portion used for training, while keeping the test set unchanged. This experiment is performed on both real and synthetic datasets and allows us to assess how well each method generalizes when trained on a perturbed structure but evaluated on the original unperturbed one.

To inject noise, we randomly select 20% of the positive entries ($A_{ij} = 1$) and an equal number of negative entries ($A_{ij} = 0$) in the training adjacency matrix. For each selected entry, we flip the value of A_{ij} (i.e. removing an existing edge or creating a spurious new one) with probability 0.5, and leave it unchanged otherwise. This results on an average of 10% corrupted entries.

For real datasets where *attribute-based* features are available, GNNs are fed these instead of *structure-based* features. Conversely, for datasets that rely on *structure-based* features, we feed the GNN the original uncorrupted adjacency rows as node features. This ensures that performance changes reflect robustness to noise in the graph structure rather than in the input feature vectors. Results are shown in figure 5 for real datasets and in figure S10 for synthetic ones.

Resilience to randomly perturbed edges is overall comparable across models, as the difference between AUC values in the unperturbed and perturbed datasets is distributed in the same range of values see figures 5 and S10. Overall, performance on assortative data generally suffers more than on disassortative one, as its decrease is larger. This is more pronounced in GNNs, with the AUC difference being

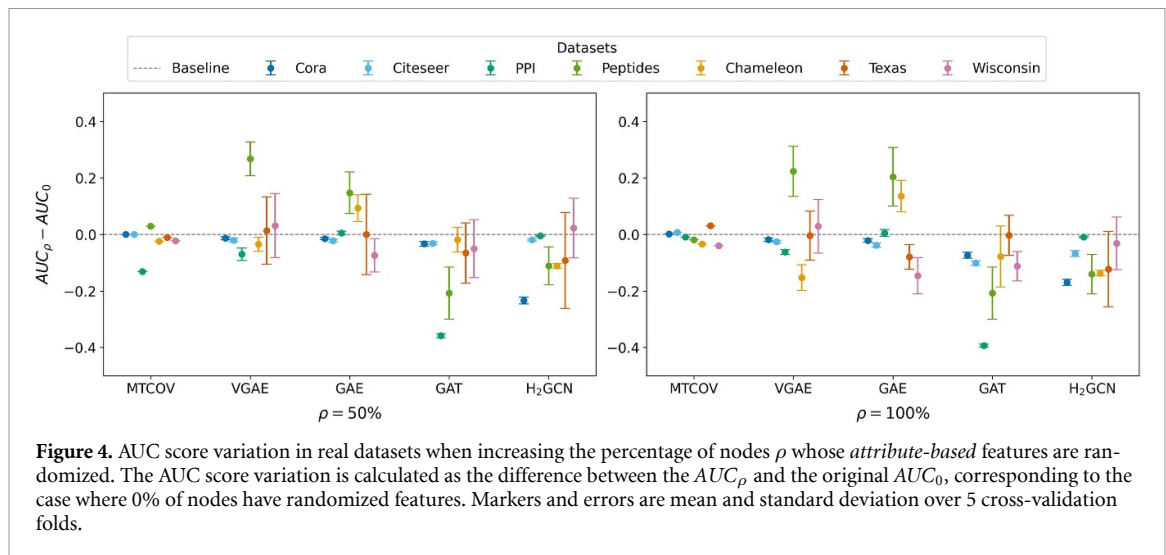


Figure 4. AUC score variation in real datasets when increasing the percentage of nodes ρ whose attribute-based features are randomized. The AUC score variation is calculated as the difference between the AUC_ρ and the original AUC_0 , corresponding to the case where 0% of nodes have randomized features. Markers and errors are mean and standard deviation over 5 cross-validation folds.

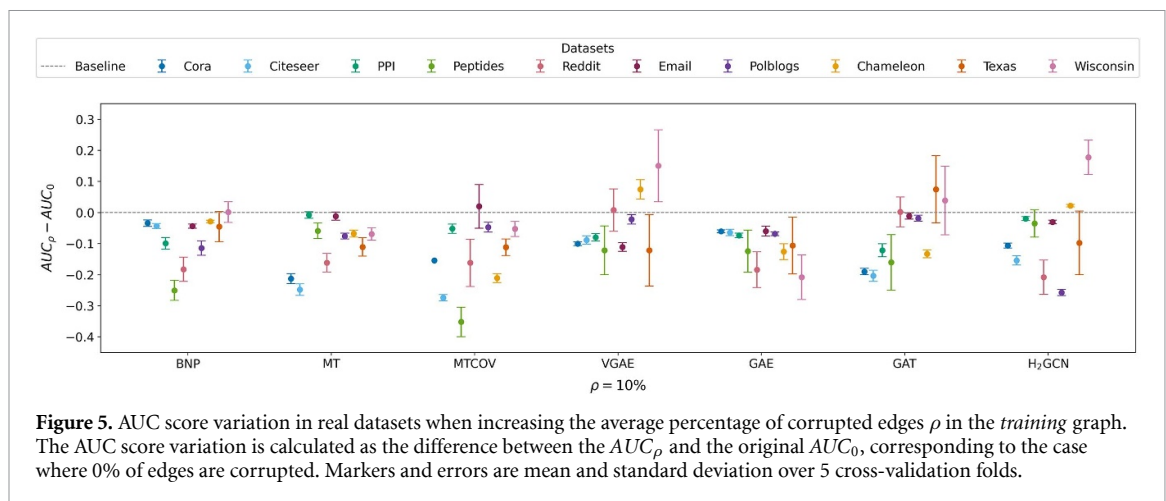


Figure 5. AUC score variation in real datasets when increasing the average percentage of corrupted edges ρ in the training graph. The AUC score variation is calculated as the difference between the AUC_ρ and the original AUC_0 , corresponding to the case where 0% of edges are corrupted. Markers and errors are mean and standard deviation over 5 cross-validation folds.

generally positive in the synthetic disassortative case, see figure S10. However, the per-fold scatterplots in appendix C.3 reveal how edge noise affects the relative performance between PGMs and GNNs. On real assortative datasets, where GNNs outperform PGMs in the unperturbed dataset in about 80% of the folds, their advantage decreases under noisy edges to roughly 60%. On synthetic assortative datasets, the PGM advantage increases, reflecting the same trend. For real disassortative datasets, where PGMs hold a 60% advantage (see section 8), this increases to 70%, whereas on synthetic disassortative graphs PGMs become slightly less dominant (see section 8). Overall, edge noise weakens the regime in which GNNs perform better, while amplifying the regimes where PGMs hold an advantage (except in the synthetic disassortative case).

8. Resilience to increasing heterophily

Having studied in detail the role of input features in learning missing links, we now focus on comparing the two frameworks for different network clustering structure. Specifically, the third experiment explores the impact of increasing the heterophily (or disassortativeness) of the graph. This is the property of observing nodes in different clusters more likely to be connected, which contrasts with homophilic (or assortative) networks, where we expect nodes in the same cluster to have higher chances of being connected. In PGMs, learning in disassortative networks has often been shown to be a more challenging task than in assortative networks [28, 66–68]. Similar declines in performance in heterophilic networks are also now increasingly investigated in GNNs [10–12].

In the synthetic case, we can directly manipulate the level of heterophily by varying the density of edges between nodes in different clusters in the affinity matrix [28]. In real data, when extra node labels are available, and assumed to be informative, we measure the homophily edge ratio h as the fraction of edges in a graph which connect nodes that have the same class label, as done e.g. in [41]. A decreasing

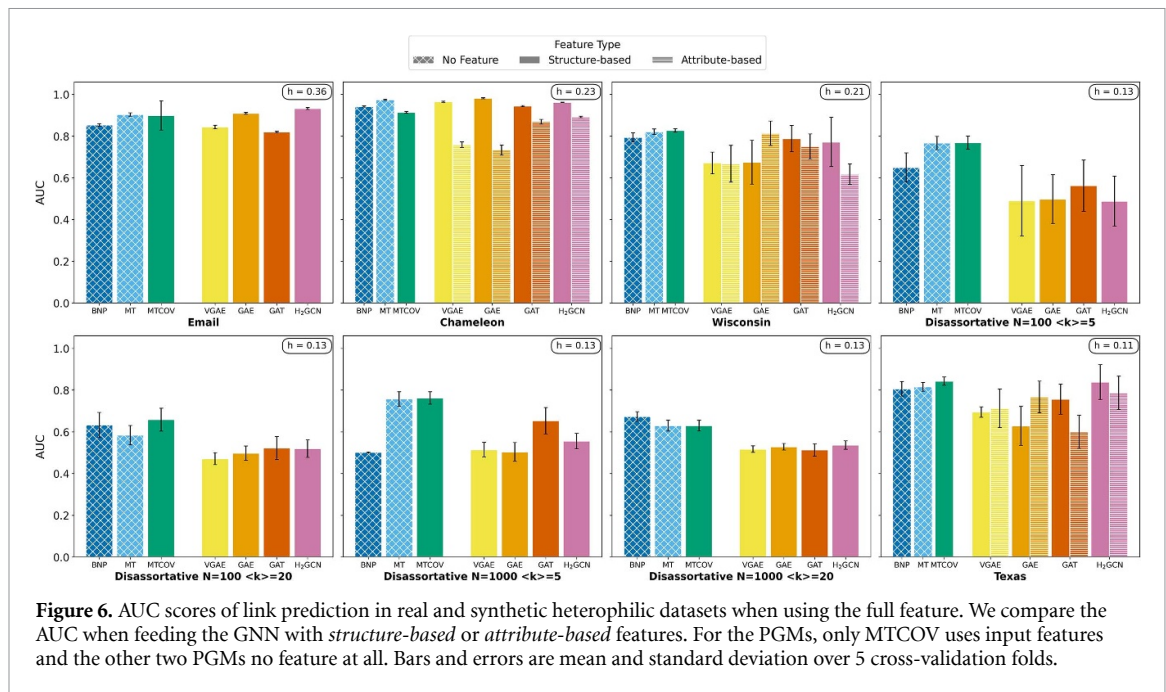


Figure 6. AUC scores of link prediction in real and synthetic heterophilic datasets when using the full feature. We compare the AUC when feeding the GNN with *structure-based* or *attribute-based* features. For the PGMs, only MTCOV uses input features and the other two PGMs no feature at all. Bars and errors are mean and standard deviation over 5 cross-validation folds.

value of h indicates increased heterophily. Four of the real datasets have $h < 0.4$. In the synthetic case we compute h by assigning each node the label corresponding to the argmax of its membership vector.

We find that PGMs generally outperform GNNs when increasing the heterophily of the graph (both synthetic and real), when given the full input feature, see figure 6. However, we also find that by giving as input to the GNN the row of the adjacency matrix (*structure-based* feature) instead of the *attribute-based* feature, the situation changes favorably. In this case, GNNs show similar performance to PGMs in two real datasets (Email and Chameleon). In the synthetic case, PGMs outperform GNNs on each dataset, even if GNNs are given adjacency rows as features. Differently from the synthetic assortative case in figure 3, increasing the graph size, and thus the dimensionality of the adjacency-row feature, does not generally improve the performance here. Consistent with these results, the per-fold scatterplots in appendix C.3 show that on real disassortative datasets, PGMs hold a 60% advantage, while on synthetic disassortative dataset the advantage grows to 90%, again in line with the fact that PGMs match the underlying generative model.

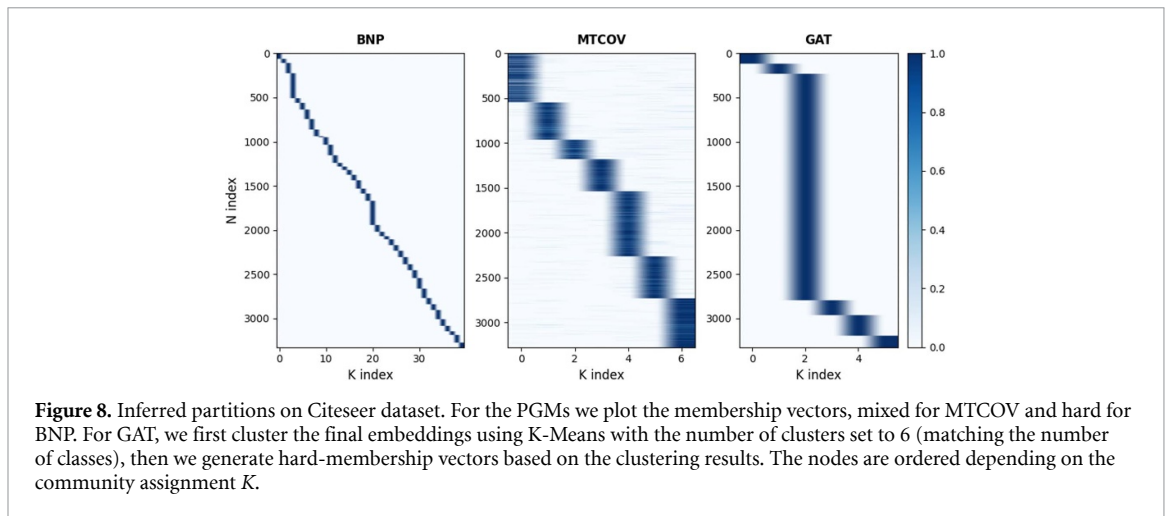
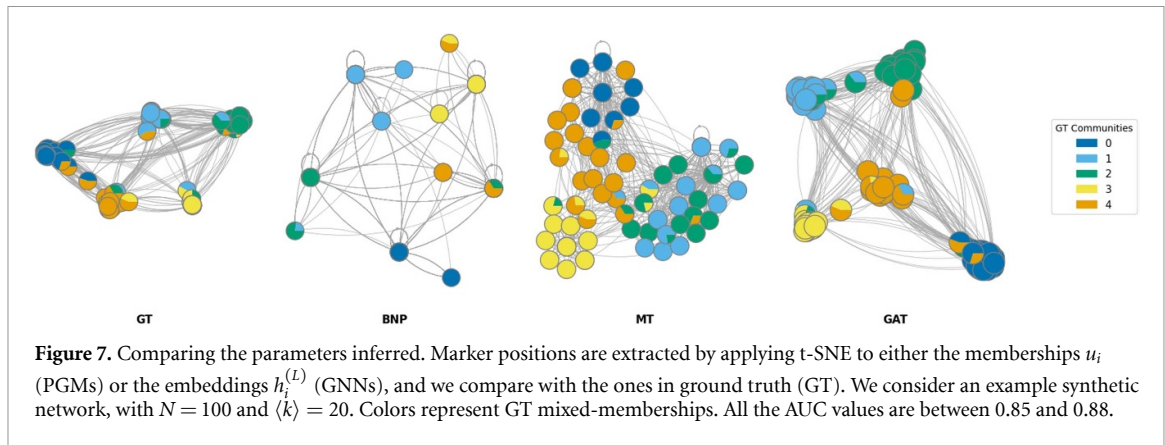
This analysis applies to all GNN models, including H₂GCN, which is explicitly designed to handle heterophily. Nonetheless, even under these favorable conditions, H₂GCN's performance remains comparable to or lower than that of PGMs, suggesting that architectural adaptations alone may not fully address the challenges posed by heterophilic graphs. This result highlights that while GNNs are expected to be generally less robust in heterophilic graphs, performance could be improved by a better choice of input features, rather than acting on a change in architecture design as proposed in [41, 69–71]. A similar conclusion was also observed in node classification tasks [72], where GNN architectures were shown to significantly improve their performance under good conditions of heterophily, specifically when nodes sharing the same label also exhibit similar embeddings and neighbor distributions. Moreover, the phenomenon of oversmoothing [73–76], which is commonly associated with poor performance on heterophilic graphs [70, 72, 77, 78], has been shown to be avoidable without altering the GNN architecture, simply by increasing the variance of weight initialization [79].

9. Interpretability and complexity

9.1. Interpretability

PGMs and GNNs are inherently different in terms of their interpretability. The PGMs analyzed here learn community membership vectors on each node. These could be either hard or soft, depending on whether nodes are allowed to belong to more than one group. Memberships are quantities directly interpretable as they provide clear insights into the community structure of the network. In addition, the linearity of these models allows a direct mapping between these parameters and the probability of observing an edge between two nodes. This is usually a design choice driven by mechanistic principles.

Unlike PGMs, GNNs learn embedding vectors, quantities that are not directly interpretable. Instead, one has to make ad-hoc processing *a posteriori* to allow for interpretability [80] or design GNN models



specifically to optimize for it [81]. A common choice is to cluster them with K-means [82–85], thus assigning them one-dimensional labels. This is a simple and direct way to analyze GNN outputs and matches our approach of prioritizing general-purpose GNN designs. Alternative and more advanced interpretability techniques exist, but they require modifying the neural network architecture itself or the loss function employed [84, 85]. Hence, we do not explore them here. Similarly, it is not directly clear how to combine node embeddings to estimate the probability of an edge. One has to choose a function that combines pairs of embedding, whose output is then associated to edges. Possible choices are the dot product or a concatenation [35, 36].

To create a parallelism with the interpretation of communities in PGMs and the output GNNs' embeddings, we visualize the memberships and embeddings in a two-dimensional plan by applying a dimensionality reduction to these vectors using t-SNE [86]. We use a synthetic network as example, as we have access to GT communities. We first notice how nodes that have a more pronounced mixed-membership are placed on the border of their main community, closer to the other communities they also participate in, see figure 7. This is somehow expected when plotting the GT u_i , given their shape. However, it is not trivial that the models are able to maintain this positional property in the inferred parameters. We find that both MULTITENSOR and GAT are able to place more mixed nodes closer to the border, as observed for the GT. For instance, nodes that belong to a large extent to two communities, are placed in between the two main clusters. However, while mixed-membership PGMs allow for a direct estimation of the mixed-membership, as this is their natural output, it is not clear how to extract this information automatically and in a principled way (beside visual inspection) in GNNs from their output embeddings, as they lack the innate ability to represent these memberships explicitly. To enhance the interpretability of GNNs, one promising direction is to develop methods that enable the extraction of interpretable mixed-membership information from their embeddings. Achieving this, however, would require architectural modifications to GNNs, as done in [82].

Furthermore, we see how models favor different clusters in real data, for a similar model expressiveness, measured by similar AUC values. We show an example of this for the Citeseer dataset in figure 8,

Table 2. Computational complexities for the models under analysis. N the number of nodes, E the total number of edges, $E^{(2)}$ the number of edges between nodes that are connected via a 2-hop path, $d_{\max}$ the maximum degree across all nodes, K the number of communities, Z the number of attribute categories, F the input feature dimension, F_ℓ the hidden layer dimension, and L the number of GNN layers.

Class	Model	Computational Complexity
PGM	MT	$O(E \cdot K^2)$
	MTCOV	$O(E \cdot K^2 + N \cdot K \cdot Z)$
	BNP	$O(N \ln(N^2))$
GNN	GAE	$O(L \cdot (E \cdot F_\ell + N \cdot F \cdot F_\ell))$
	VGAE	$O(L \cdot (E \cdot F_\ell + N \cdot F \cdot F_\ell))$
	GAT	$O(L \cdot (E \cdot F_\ell + N \cdot F \cdot F_\ell))$
	H ₂ GCN	$O(E \cdot d_{\max} + L \cdot ((E + E^{(2)}) \cdot F_\ell) + N \cdot F \cdot F_\ell)$

where we plot the inferred parameters. In this example, BNP tends to place nodes in more and smaller clusters, MULTITENSOR and MTCOV have similar result (we report only one), with 7 clusters with low amount of mixed-membership, and GAT has one dominating larger cluster. All the AUC values are between 0.89 and 0.93, indicating that the models are reconstructing A well, but they do so by learning different partitions.

9.2. Computational complexity

In addition to interpretability, PGMs and GNNs also differ in their computational complexity, as shown in table 2. We notice in particular that all the models are linear (or almost, in BNP) in the system size (the number of nodes N or of edges E).

The main difference is in the multiplicative factors, which reflect each model's internal design. PGMs have the number of communities K (at most squared) and the number of node covariates Z (in MTCOV). Since, in practice, K and Z have a similar order of magnitude and are typically much smaller than N or E , the overall scaling remains manageable. GNNs have the number of layers L , the embedding hidden dimension F_ℓ , and the input feature dimension F . Usually, L is kept small (e.g. 2 or 3), to prevent issues such as oversmoothing [73–76], while the hidden dimension F_ℓ is larger (e.g. 128–512), but still typically smaller than the number of nodes N or edges E . In contrast, the input feature dimension F can be very high, even comparable to N in large graphs with rich node features.

Overall, while both PGMs and GNNs exhibit comparable asymptotic scaling with graph size, PGMs are mainly affected by the quadratic dependence on K , which remains small in most applications, whereas GNNs are sensitive to the potentially large product $F \cdot F_\ell$. As a result, in graphs with high-dimensional features, we conclude that GNNs can be substantially more computationally expensive than PGMs.

An additional practical consideration is the burden of hyperparameter selection. For PGMs, model tuning is limited to a small number of interpretable hyperparameters: the number of communities K (MULTITENSOR, MTCOV); the parameter γ (MTCOV); and parameters related to Monte Carlo sampling, like burn-in period and number of samples BNP. In contrast, GNNs require tuning a substantially larger combination of hyperparameters, including learning rate, weight decay, dropout, hidden dimension, and number of layers, all of which must be carefully tuned to ensure stable training and good convergence. Exploring this hyperparameter space introduces a significant computational overhead that may far exceed the cost of a single model evaluation. As a result, hyperparameter optimization becomes a non-negligible component of the overall computational cost for GNNs, further amplifying their computational cost relative to PGMs.

10. Conclusions

In this work, we compare PGMs and GNNs in how they handle network data, and we conduct our experiments on a link prediction task.

Our findings reveal that PGMs outperform GNNs when input features are either low-dimensional or noisy, a common scenario in real-world networks where node attributes may be minimal or unreliable. Additionally, we observe that PGMs exhibit greater robustness to increasing heterophily of the graph compared to GNNs. Interestingly we find that GNNs performance on heterophilic graphs could be improved by a better choice of input features, as similarly shown for a node classification task in [16, 72]. When reliable, high-dimensional node features are available and the graph is assortative, GNNs can achieve performance comparable to, or exceeding, that of PGMs.

Table 3. Summary of empirical findings comparing PGMs and GNNs across different conditions. Symbol meaning: $\uparrow$ AUC > 0.85 or noisy Δ AUC > 0.05 ; $\uparrow$ AUC $\in [0.75, 0.85]$ or noisy Δ AUC $\in [0, 0.05]$; $\downarrow$ AUC $\in [0.65, 0.75]$ or noisy Δ AUC $\in [-0.05, 0]$; $\downarrow$ AUC < 0.65 or noisy Δ AUC < -0.05 ; all values denote the mean AUC across folds and datasets (real and synthetic). Here, Z denotes the number of attribute categories in MTCOV, while F denotes the input feature dimension for GNNs.

Criterion	Condition	Community structure	PGM	GNN
Feature type	Structure-based	Assortative	$\uparrow$	$\uparrow$
		Disassortative	$\uparrow$	$\downarrow$
	Attribute-based	Assortative	—	$\uparrow$
		Disassortative	—	$\downarrow$
Feature dimensionality	High-dimensional	Any	—	$\uparrow$
	Low-dimensional	Any	$\uparrow$	$\downarrow$
Noisy input	Features	Any	$\downarrow$	$\downarrow$
	Edges	Assortative	$\downarrow$	$\downarrow$
		Disassortative	$\downarrow$	$\uparrow$
Computational complexity	Network size		linear	linear
	Feature dimensionality		small Z	large F
	Hyperparameters to tune		fewer than 4	more than 4

However, an important advantage of GNNs lies in their flexibility in processing input features. Unlike PGMs, GNNs do not rely on ad-hoc design choices to handle heterogeneous features, thus simplifying their implementation and widening their usage. Nevertheless, recent work [87] suggests that it may be possible to benefit from similar black-box tools as those used in GNNs to incorporate arbitrary types of attributes as input to PGMs.

Moreover, previous studies have shown that linear models can outperform GNNs in link prediction tasks [88, 89]. Given these limitations, GNNs should ideally be designed to learn automatically whether to discard irrelevant features or to select what features to use, potentially incorporating structural information such as the adjacency matrix of the graph. In the absence of this additional learning step, care should be taken when choosing what features to give as input.

We have focused on link prediction tasks. For this, GNNs require an additional step of mapping node embeddings to edge-level predictions, unlike PGMs that directly model the probability of observing an edge. One can expect that GNNs may demonstrate advantages on other types of tasks that require direct usage of the output representation, e.g. in node classification. Investigating how the insights derived from this work translate to other tasks is an interesting direction for future work. This would require selecting PGMs that are able to handle input features, but also to consider as baseline for comparison standard neural networks that only use input features. We expect that the ability to discard or attenuate uninformative features would play a similar role as the one we observed for link prediction.

Data availability statement

No new data were created or analysed in this study.

Acknowledgments

This research project has been developed during a visiting period at Max Planck Research School for Intelligent Systems (IMPRS-IS), Tübingen, Germany.

Appendix A. Details of the models

A.1. PGMs variants

The MULTITENSOR model [28]. In MULTITENSOR⁴, the learnable parameters $\theta = (u_i, v_j, w)$ are combined through a bilinear tensor factorization:

$$M_{ij} = \sum_{k,l=1}^K u_{ik} v_{jl} w_{kl}.$$

⁴ <https://github.com/MPI-IS/multitensor>.

The resulting expected edge weight M_{ij} captures the interactions between latent communities, accommodating both directed and undirected networks. The values of M_{ij} are used as link-prediction scores in the AUC calculation on the test set, both for existent and non-existent edges. MULTITENSOR adopts a *mixed-membership* representation, meaning that each node u_i can have multiple non-zero entries corresponding to partial community affiliations. Each edge is modeled as a Poisson random variable with mean M_{ij} , yielding the likelihood:

$$P(A_{ij} | \theta) = \frac{e^{-M_{ij}} (M_{ij})^{A_{ij}}}{A_{ij}!}.$$

This formulation naturally captures degree heterogeneity. The membership vectors u_i and v_j are non-negative but not normalized, so their overall magnitude (e.g. their L_1 norm) is free to vary across nodes. Nodes with larger membership magnitudes therefore yield larger expected values of $\sum_j M_{ij}$ (or $\sum_i M_{ij}$ for directed networks) and consequently higher expected degrees, without altering their relative affinities toward different communities. This provides an implicit form of degree correction, avoiding the need for an explicit degree-correction term. In contrast, a Bernoulli likelihood would require normalized memberships to ensure probabilities remain in $[0, 1]$, which constrains degree variability and makes heterogeneity more difficult to represent.

Assuming conditional independence of edges given θ , this yields the overall network log-likelihood:

$$\mathcal{L}_G(U, V, w) = \sum_{i,j} \left[A_{ij} \log \left(\sum_{k,l} u_{ik} v_{jl} w_{kl} \right) - \sum_{k,l} u_{ik} v_{jl} w_{kl} \right], \quad (\text{S4})$$

where constant terms independent of the parameters have been omitted and the matrices U, V have as rows the vectors u_i, v_j , respectively.

The MTCOV model [26]. MTCOV⁵ extends MULTITENSOR by jointly modeling the network structure and node covariates within a unified probabilistic framework. The structural component retains the same tensor factorization of the parameters in $\theta = (u_i, v_j, w)$ and mixed-membership representation:

$$M_{ij} = \sum_{k,l=1}^K u_{ik} v_{jl} w_{kl}.$$

As in MULTITENSOR, the values of M_{ij} serve as link-prediction scores. Node attributes are assumed to be probabilistically generated from the same latent membership vectors that govern the network structure. A correlation matrix $\beta \in \mathbb{R}^{K \times Z}$ captures the relationships between the K latent communities and the Z attribute categories, explicitly modeling how node covariates influence both community formation and connectivity patterns.

The observed node attributes are organized in a binary feature matrix $X^{N \times Z}$, where each row is a one-hot encoded vector $x_i = (x_{i1}, \dots, x_{iZ})$ with $x_{iz} = 1$ if node i belongs to category z and 0 otherwise, ensuring $\sum_{z=1}^Z x_{iz} = 1$.

The probability of observing category z for node i is given by:

$$\pi_{iz} = \frac{1}{2} \sum_{k=1}^K \beta_{kz} (u_{ik} + v_{ik}), \quad (\text{S5})$$

where β_{kz} represents the probability of observing category z given membership in community k . This formulation ensures $\pi_i = (\pi_{i1}, \dots, \pi_{iZ})$ forms a valid probability vector where $\pi_{iz} \in [0, 1]$ and $\sum_{z=1}^Z \pi_{iz} = 1$ for all nodes i .

The attributes are modeled using a multinomial distribution with parameter π_i , yielding the likelihood:

$$P_X(X_i = x_i | U, V, \beta) = \prod_{z=1}^Z \pi_{iz}^{x_{iz}}. \quad (\text{S6})$$

This formulation assumes attributes can be binned into a finite number Z of unordered categories. Multiple attributes can be combined by encoding each combination as a distinct category.

⁵ <https://github.com/MPI-IS/probinet>.

To satisfy the normalization constraints $\sum_{z=1}^Z \pi_{iz} = 1$, the parameters are constrained such that $\sum_{z=1}^Z \beta_{kz} = 1$ for all communities k , and $\sum_{k=1}^K u_{ik} = \sum_{k=1}^K v_{ik} = 1$ for all nodes i . Assuming conditional independence of attributes across nodes, given the parameters, leads to the attribute log-likelihood:

$$\mathcal{L}_X(U, V, \beta) = \sum_{i=1}^N \sum_{z=1}^Z x_{iz} \log(\pi_{iz}) = \sum_{i,z} x_{iz} \log \left(\frac{1}{2} \sum_{k=1}^K \beta_{kz} (u_{ik} + v_{ik}) \right). \quad (S7)$$

The complete model combines both the network structure likelihood $\mathcal{L}_G(U, V, W)$ and the node attribute likelihood $\mathcal{L}_X(U, V, \beta)$ using a scaling parameter $\gamma \in [0, 1]$ that balances their contributions:

$$\mathcal{L}(U, V, W, \beta) = (1 - \gamma) \mathcal{L}_G(U, V, W) + \gamma \mathcal{L}_X(U, V, \beta). \quad (S8)$$

This formulation allows flexible interpolation between purely using the network data ($\gamma = 0$) and purely using the attribute data ($\gamma = 1$). The optimal value of γ can be determined via cross-validation, it can vary based on the specific application at hand.

The BNP model [31]. The Bayesian non-parametric (BNP⁶) model [31] adopts a microcanonical formulation of the SBM, in which each node i is assigned to a single latent community $c_i \in \{1, \dots, K\}$, corresponding to a one-hot encoded membership vector u_i , and includes a $K \times K$ block edge-count matrix e with entries e_{kl} being the number of edges between groups k and l :

$$e_{kl} = \sum_{i,j} A_{ij} \delta_{c_i,k} \delta_{c_j,l},$$

where δ is the Kronecker delta. This formulation is valid for both directed and undirected networks. We use the degree-corrected formulation, where the inter-block edge counts $\{e_{kl}\}$ and the degree sequence $\{d_i\}$ are fixed exactly. Thus, the parameter set is $\theta = \{c_i, e, d_i\}$, but only the community assignments $\{c_i\}$ are learnable latent variables. All graphs sharing the same constraints are considered equally likely. The likelihood of observing a network A given a partition $\{c_i\}$ is:

$$P(A | \{c_i\}) = \frac{1}{\Omega(\{e_{kl}\}, \{n_k\}, \{d_i\})},$$

where n_k is the number of nodes belonging to group k and $\Omega(\{e_{kl}\}, \{n_k\}, \{d_i\})$ is the number of distinct graphs consistent with the specified inter-block edge counts, block sizes and degree sequences. In this Bayesian non-parametric formulation, the number of blocks K is inferred automatically from the data by maximizing the posterior $P(\{c_i\} | A)$, using non-parametric priors. The posterior is maximized via MCMC [32, 33], following the agglomerative heuristic proposed in [90]. In our experiments, we employ the `minimize_blockmodel_dl()` function from the `graph-tool` library.

Link prediction is performed as follows. Denoting by δA_{ij} an edge candidate in the test set, the plausibility of its existence used in the AUC calculation is evaluated via the posterior estimation using the training set encoded in A_{train} :

$$P(\delta A | A_{\text{train}}) \propto \sum_{\{c_i\}} \frac{P(A_{\text{train}} \cup \delta A | \{c_i\})}{P(A_{\text{train}} | \{c_i\})} P(\{c_i\} | A_{\text{train}}),$$

where the sum runs over Monte Carlo samples of the posterior. Here we consider 100 samples. We perform posterior sampling via repeated calls to the `graph-tool` function `mcmc_equilibrate()`, where each of the 100 iterations executes 10 Metropolis-Hastings sweeps. For a fixed partition $\{c_i\}$, the predictive probability $P(\delta A_{ij} | A_{\text{train}}, \{c_i\})$ is computed using the `get_edges_prob()` function from `graph-tool`. AUC is computed over both existent and non-existent test edges.

A.2. GNNs variants

Graph attention networks (GAT) [39]. In GAT⁷, aggregation is done with implicit interaction weights $\alpha(x_i, x_j)$ as:

$$h_i^{(\ell+1)} = \phi \left(h_i^{(\ell)}, \bigoplus_{j \in \partial_i} \alpha(h_i^{(\ell)}, h_j^{(\ell)}) \psi(h_j^{(\ell)}) \right). \quad (S9)$$

⁶ <https://graph-tool.skewed.de/>.

⁷ GATConv layer from https://github.com/pyg-team/pytorch_geometric [48].

The weights $\alpha(h_i^{(\ell)}, h_j^{(\ell)})$ are learnable ‘attention’ coefficients that encode the relative importance of neighbor j when updating node i . Note that these are feature-dependent, as they depend on the embeddings $h_i^{(\ell)}$ and $h_j^{(\ell)}$. The operator $\oplus$ is instantiated as a simple summation.

Following the formulation in [39], the attention coefficients are computed as:

$$\varepsilon_{ij} = \text{LeakyReLU}\left(a^\top \left[Wh_i^{(\ell)} \parallel Wh_j^{(\ell)}\right]\right) \in \mathbb{R}, \quad (\text{S10})$$

$$\alpha(h_i^{(\ell)}, h_j^{(\ell)}) = \text{softmax}_j(\varepsilon_{ij}) = \frac{\exp(\varepsilon_{ij})}{\sum_{k \in \partial_i} \exp(\varepsilon_{ik})}, \quad (\text{S11})$$

where $a^\top \in \mathbb{R}^{2F'}$ is a learnable vector of parameters, $W \in \mathbb{R}^{F' \times F}$ is a weight matrix that transforms an F -dimensional representation $h_i^{(\ell)}$ into a (usually) lower-dimensional representation of dimension F' , and $\parallel$ denotes concatenation operation. To enhance stability and expressiveness, we employ multi-head attention. In all our experiments, we fix the number of heads to two and concatenate their outputs.

GAE and VGAE [40]. GAE and VGAE⁸ extend GNNs with an autoencoding framework in which the goal is to learn latent node variables $Z \in \mathbb{R}^{N \times D}$, where D is the dimensionality of the latent space. The two models differ in a fundamental way: GAE provides a deterministic encoding, while VGAE is a probabilistic variant that learns an approximate distribution over latent representations.

In both cases, the model uses a network likelihood function as:

$$p(A_{ij} = 1 \mid z_i, z_j) = \sigma(z_i^\top z_j), \quad p(A \mid Z) = \prod_{i,j} p(A_{ij} \mid z_i, z_j),$$

where the z_i are the rows of the matrix Z . Training is done using a two-layer Graph Convolutional Network (GCN) [91], meaning using two successive message-passing layers of the form:

$$h_i^{(\ell+1)} = \phi\left(h_i^{(\ell)}, \bigoplus_{j \in \partial_i} c_{ij} \psi(h_j^{(\ell)})\right), \quad (\text{S12})$$

where the operator $\bigoplus$ denotes summation over neighbors and c_{ij} are coefficients determined by the nodes’ degrees.

GAE (non-probabilistic). This minimizes a binary cross-entropy loss as in equation (S16) to compute deterministic embeddings:

$$Z = H^{(L)}, \quad z_i = h_i^{(L)}.$$

VGAE (probabilistic variant). This maximizes a variational lower bound (ELBO):

$$\mathcal{L} = \mathbb{E}_{q(Z|A,X)}[\log p(A \mid Z)] - \text{KL}[q(Z \mid A, X) \parallel p(Z)],$$

where $\text{KL}[\cdot]$ denotes the Kullback–Leibler divergence [92]; the prior factorizes as $p(Z) = \prod_i \mathcal{N}(z_i \mid 0, I)$; and one learns a probabilistic representation by estimating a variational posterior over Z , following the Variational Autoencoder (VAE) framework [93, 94]:

$$q(Z \mid A, X) = \prod_{i=1}^N q(z_i \mid A, X), \quad q(z_i \mid A, X) = \mathcal{N}(z_i \mid \mu_i, \text{diag}(\sigma_i^2)), \quad (\text{S13})$$

with mean μ and log-variance $\log \sigma$ produced by the two-layer GCN:

$$\mu = \text{GCN}_\mu(A, X), \quad \log \sigma = \text{GCN}_\sigma(A, X). \quad (\text{S14})$$

H₂GCN [41]. H₂GCN⁹ accounts for the fact that in heterophilic networks neighboring nodes may have dissimilar features and labels, unlike standard GNNs that instead assume similarity. It does so by specific design choices such as separate $\phi_i^{(\ell)}$ and $\phi_{\partial i}^{(\ell)}$ for self and neighbor embeddings, skip connections, and embeddings from second-order neighbors.

⁸ https://github.com/DaehaKim/vgae_pytorch/.

⁹ <https://github.com/GitEventhandler/H2GCN-PyTorch>.

A representative layer update is:

$$h_i^{(\ell+1)} = \phi \left(\psi_{\text{self}}(h_i^{(\ell)}), \bigoplus_{j \in \partial_i} \psi_{1\text{-hop}}(h_j^{(\ell)}), \bigoplus_{k \in \partial_i^{(2)}} \psi_{2\text{-hop}}(h_k^{(\ell)}) \right). \quad (\text{S15})$$

Here, ψ_{self} transforms the node's own embedding, $\psi_{1\text{-hop}}$ aggregates information from its first-order neighbors ∂_i , and $\psi_{2\text{-hop}}$ aggregates information from its second-order neighbors $\partial_i^{(2)}$. In all cases, the operator $\bigoplus$ denotes summation.

These design elements have been shown to mitigate the oversmoothing effect [73–76], which is known to worsen GNN performance in heterophilic settings [70, 72, 77, 78].

A.2.1. GNNs training

Learning is done by optimizing a given cost function, which depends on the task. As we focus on link prediction and node clustering, we use binary cross-entropy on the adjacency matrix (in our experiments we consider binary A). In the link prediction setting, the model is trained to assign high scores to *existent* edges, i.e. the pairs of nodes (i, j) for which an edge is observed in the network, and low scores to *non-existent* edges, i.e. node pairs that do not share an edge. Let E denote the set of observed edges and E^- a set of sampled node pairs with no edge (negative samples). The corresponding binary cross-entropy loss is:

$$\mathcal{L} = - \sum_{(i,j) \in E} \log \sigma(h_i^{(L)} \cdot h_j^{(L)}) - \sum_{(i,j) \in E^-} \log(1 - \sigma(h_i^{(L)} \cdot h_j^{(L)})), \quad (\text{S16})$$

where $\sigma(\cdot)$ denotes the logistic function. The VGAE loss combines this binary cross-entropy reconstruction with the KL divergence term.

Appendix B. Synthetic dataset details

We generate the synthetic networks following the generative modeling provided by MULTITENSOR [28]. The network consists of N nodes and its connectivity is described by an adjacency matrix A , where A_{ij} represents the number of edges from node i to node j . The networks are generated probabilistically, assuming an underlying structure consisting of K overlapping groups. Each node belongs to each group to an extent described by a K -dimensional membership vector. In order to generate directed networks, each node i has two membership vectors, u_i and v_i , which determine how i forms outgoing and incoming links, respectively. When modeling undirected networks, we set $u_i = v_i$. Then, the density of edges within and across communities is regulated by a $K \times K$ -dimensional affinity matrix w .

Given the parameters u_i, v_j, w , we sample entries A_{ij} of the adjacency matrix from a Poisson distribution as:

$$A_{ij} \sim \text{Pois}(M_{ij}) \quad (\text{S17})$$

$$M_{ij} = \sum_{k, \ell=1}^K u_{ik} v_{j\ell} w_{k\ell}. \quad (\text{S18})$$

In our current applications, we concentrate on the sparse case, where M_{ij} is small, and we then clip A_{ij} to consider only binary entries $A_{ij} \in \{0, 1\}$. Although we ultimately clip A_{ij} to binary entries, we use the Poisson distribution because it provides a natural and interpretable framework for handling degree heterogeneity. While the membership vectors u_i and v_i have nonnegative entries, the model does not impose normalization constraints on them. This design allows for degree heterogeneity by learning higher overall entries (and thus a larger L_1 norm of u_i and v_i) for high-degree nodes. A node with a large membership vector will have higher M_{ij} values, leading to a higher expected number of connections, without altering its relative preference for different communities. Therefore, this framework naturally handles realistic variations in node degrees without requiring explicit degree-correction mechanisms, a property that is not as straightforwardly captured by a Bernoulli distribution. Notice also, that in sparse datasets of large enough size as it is the case in our experiments, a Poisson distribution approximates a Bernoulli one.

To generate networks with either assortative or disassortative community structure, we control the form of the affinity matrix w , which governs the interaction strengths between latent communities. In the assortative setting, w is generated as a one-dimensional array of size K , corresponding to a diagonal

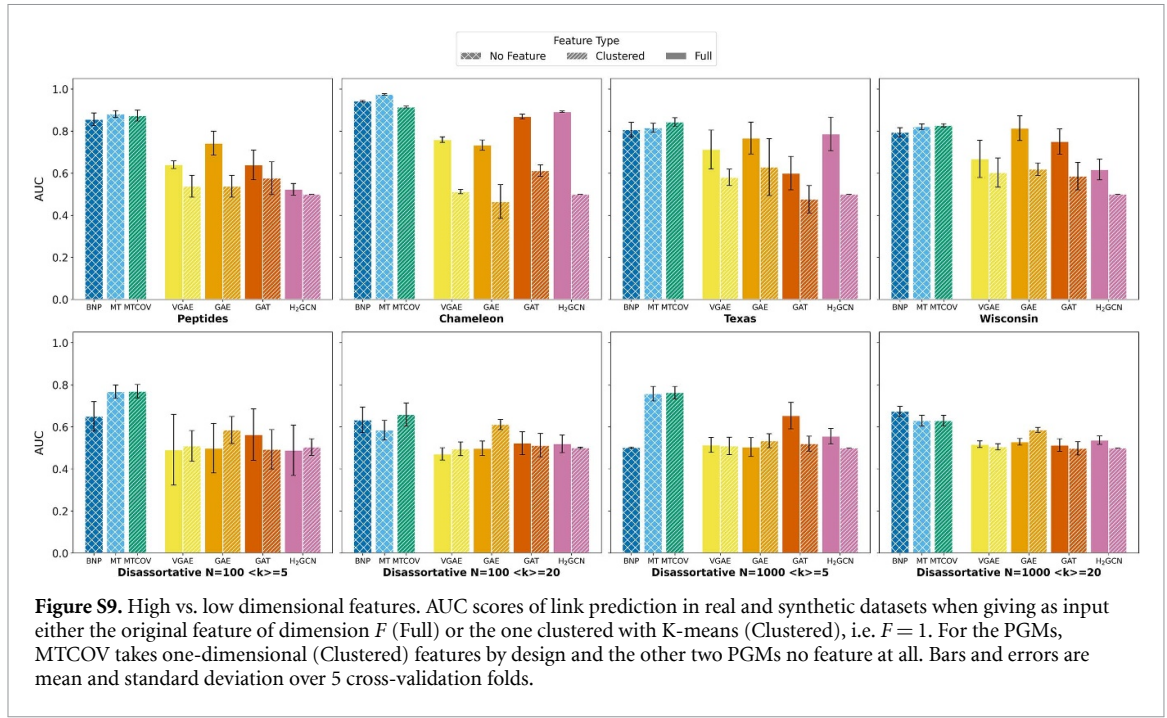


Figure S9. High vs. low dimensional features. AUC scores of link prediction in real and synthetic datasets when giving as input either the original feature of dimension F (Full) or the one clustered with K-means (Clustered), i.e. $F = 1$. For the PGMs, MTCOV takes one-dimensional (Clustered) features by design and the other two PGMs no feature at all. Bars and errors are mean and standard deviation over 5 cross-validation folds.

affinity matrix. This implies that nodes are more likely to connect within their own community, resulting in higher intra-community edge density. In contrast, for disassortative networks, w is sampled as a full $K \times K$ matrix, where the off-diagonal entries enable significant interactions between different communities. The entries of w are independently drawn from a Gamma distribution with shape and rate parameters both equal to 1.0, i.e. $w_{k\ell} \sim \text{Gamma}(1.0, 1.0)$. Node community memberships u_i and v_j are drawn from a symmetric Dirichlet distribution $u_i, v_j \sim \text{Dirichlet}(\alpha \mathbf{1})$, with $\alpha = 0.05$ ensuring that each node predominantly belongs to a small subset of communities, and $\mathbf{1}$ being a K -dimensional vector with entries equal to 1.

To summarize, the parameters of the SBM are generated as:

$$u_i, v_i \sim \text{Dirichlet}(\alpha \mathbf{1}) \text{ with } \alpha = 0.05, \quad \text{for each node } i = 1, \dots, N \quad (\text{S19})$$

$$w_{k\ell} \sim \begin{cases} \text{Gamma}(a, b) & \text{for all } k, \ell \in \{1, \dots, K\} \quad (\text{disassortative}) \\ \text{Gamma}(a, b) \cdot \delta_{k\ell} & (\text{assortative}) \end{cases}, \quad (\text{S20})$$

where $\delta_{k\ell}$ is the Kronecker delta ensuring a diagonal matrix in the assortative case and $a = b = 1.0$ in both cases.

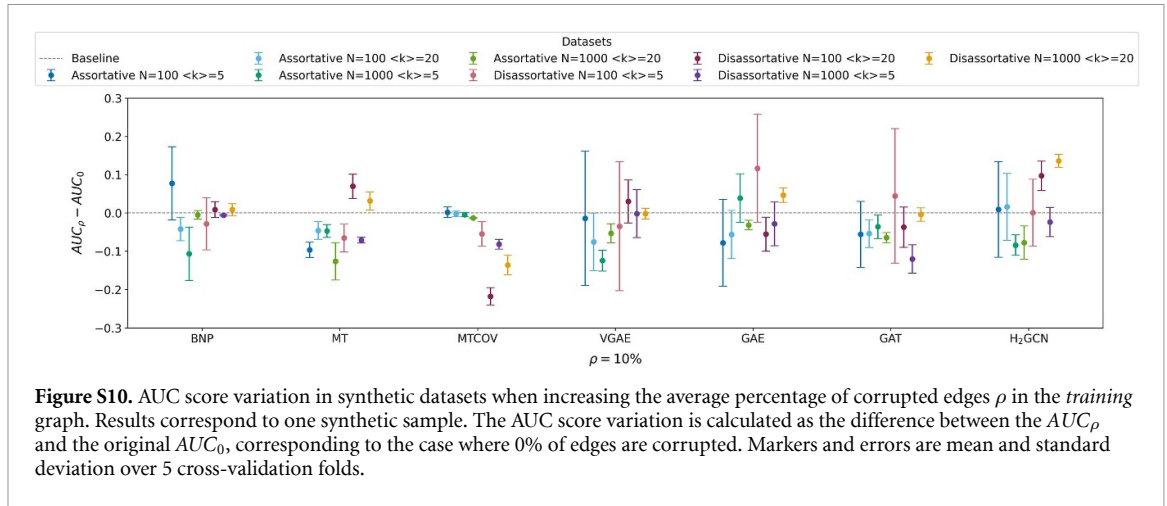
We generate directed networks of $N \in \{100, 1000\}$ number of nodes; $K = 5$ communities and average degree $\langle k \rangle \in \{5, 20\}$. For each combination of size and average degree, we generate one homophilic and one heterophilic network. We sample 10 networks for each parameter configuration.

Appendix C. Additional plots

In this section, we present additional plots complementing the ones presented in the main manuscript.

C.1. Clustered feature additional plot

In figure S9 we present an additional plot complementing the clustered features experiment presented in section 6. We further illustrate the performance of GNNs when fed either the full input feature or the clustered one on both real and synthetic datasets. As shown previously in figures 2 and 3, we observe a significant drop in GNN performance when the input features are clustered, and this effect is consistent across most datasets. The difference in performance between full and clustered features tends to be smaller when the input is already low-dimensional, as in the case of Peptides ($F = 9$), or when the full feature performance is already low, such as in the disassortative synthetic setting.



C.2. Noisy edges additional plot

In figure S10 we present an additional plot complementing the noisy edges experiment presented in section 7. We show results on synthetic data. We observe that performance is generally better, i.e. models are more resilient to noisy edges, than in real data. In several cases, the difference in AUC calculated on perturbed and on unperturbed data is around 0 (accounting for error bars). As shown in figure 5 for real data, assortative graphs are more affected than disassortative ones, except for MTCOV. Notably, several models exhibit a positive AUC difference on the synthetic disassortative data.

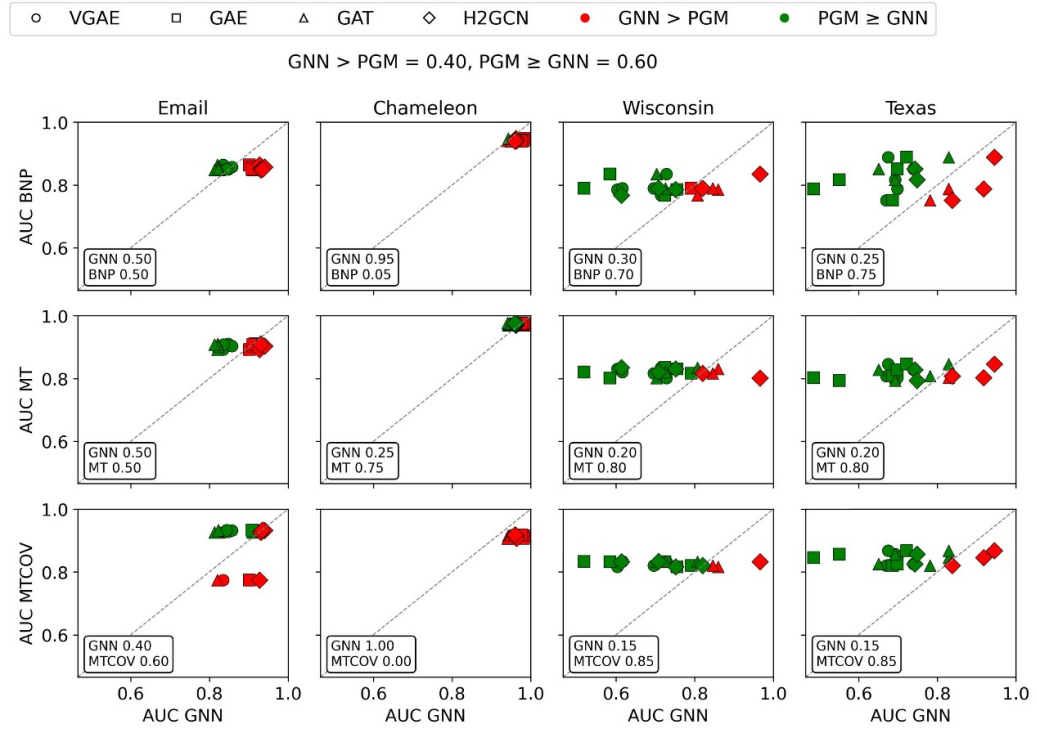
C.3. PGM vs. GNN scatterplots

To complement the results presented in the main manuscript, we analyze the performance of PGMs and GNNs in greater detail by comparing their per-fold AUC scores. This allows us to assess the relative performance of the models fold-by-fold, going beyond averages and standard deviations. Each point in the scatterplots corresponds to one of the 5 cross-validation folds, plotted as (AUC_{GNN}, AUC_{PGM}) . Points above the diagonal indicate that the PGM performs better on that fold, while points below indicate that the GNN does. For each subplot, we also report the percentage of folds won by each model family.

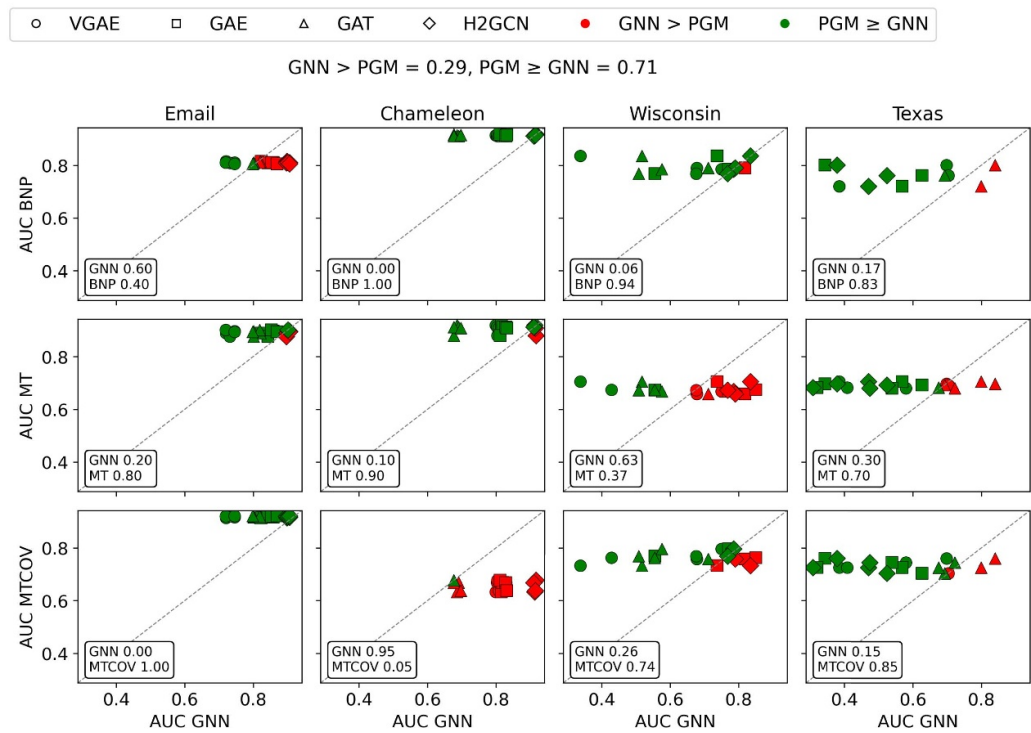
We compare the per-fold performance for both real and synthetic datasets. For real datasets where features are available, we report either the *attribute-based* or *structure-based* GNN performance, selecting whichever achieves the highest score. All plots correspond to the case where the feature dimensionality is kept full and the feature vector entries are not randomized, i.e. no injected feature noise. We focus on both the original graphs and their corrupted versions, in which a fraction of the edges has been randomly perturbed (section 7). This side-by-side comparison highlights whether corrupting the graph changes which model family dominates on each dataset.

In figure S11 and S12, we see that edge noise generally shifts performance in favor of PGMs on real data, reducing the advantage of GNNs on assortative graphs and amplifying the advantage of PGMs on disassortative ones. In figure S13, we observe the same shift for synthetic assortative graphs (with an amplified PGM advantage), whereas in figure S14 GNN performance improves under noise for the synthetic disassortative ones, yet remains lower than that of PGMs.



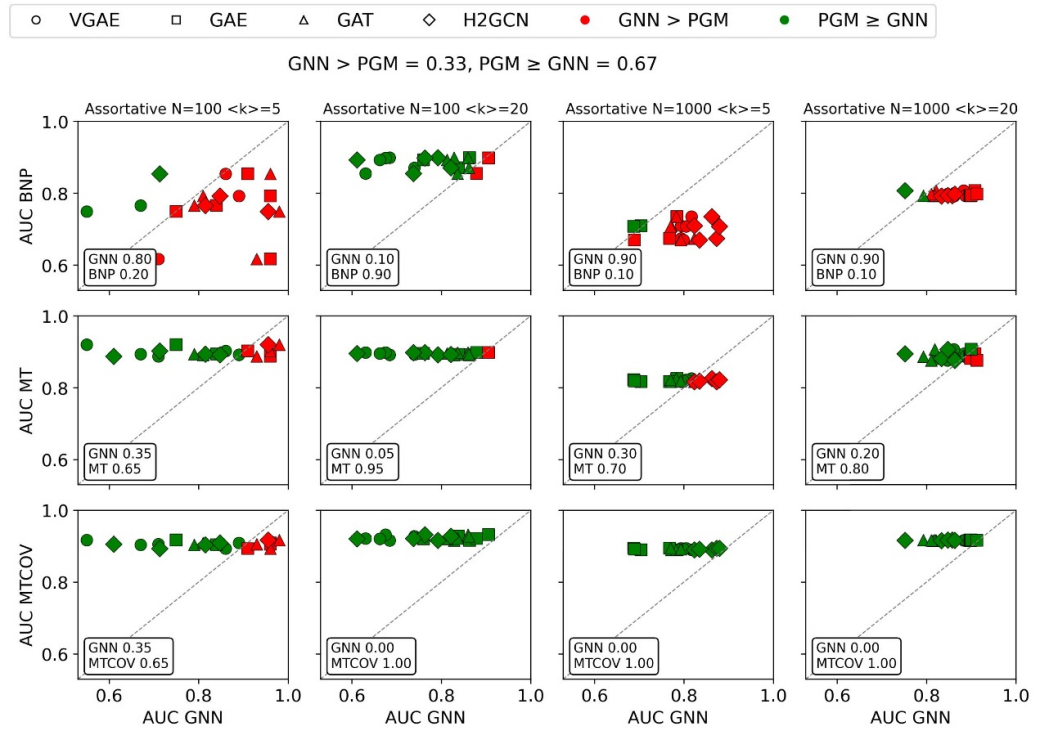


(a) Unperturbed graph (original edges).

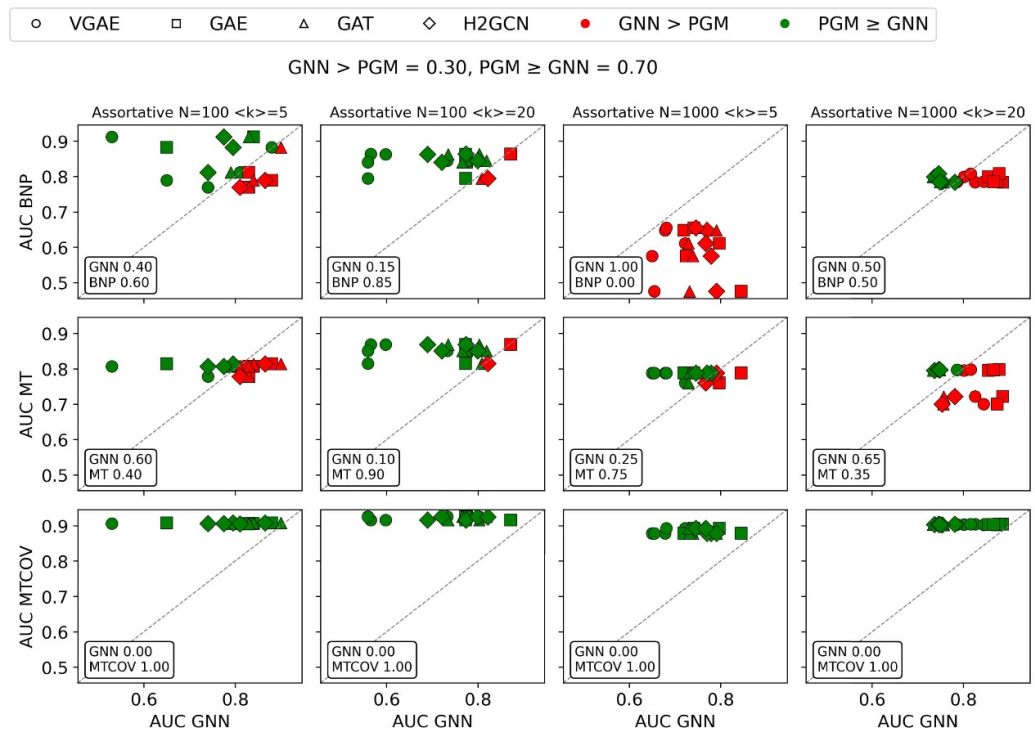


(b) Graph with 10% of edges randomly corrupted.

Figure S12. Real disassortative datasets: per-fold comparison between PGMs and GNNs.. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.

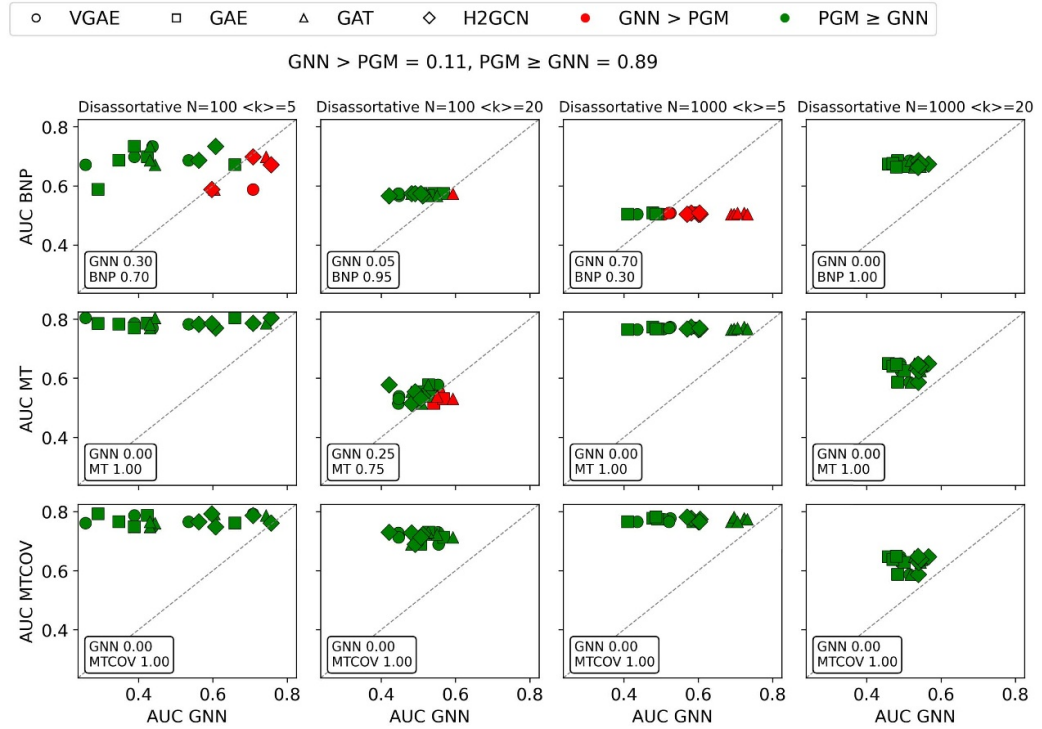


(a) Unperturbed graph (original edges).

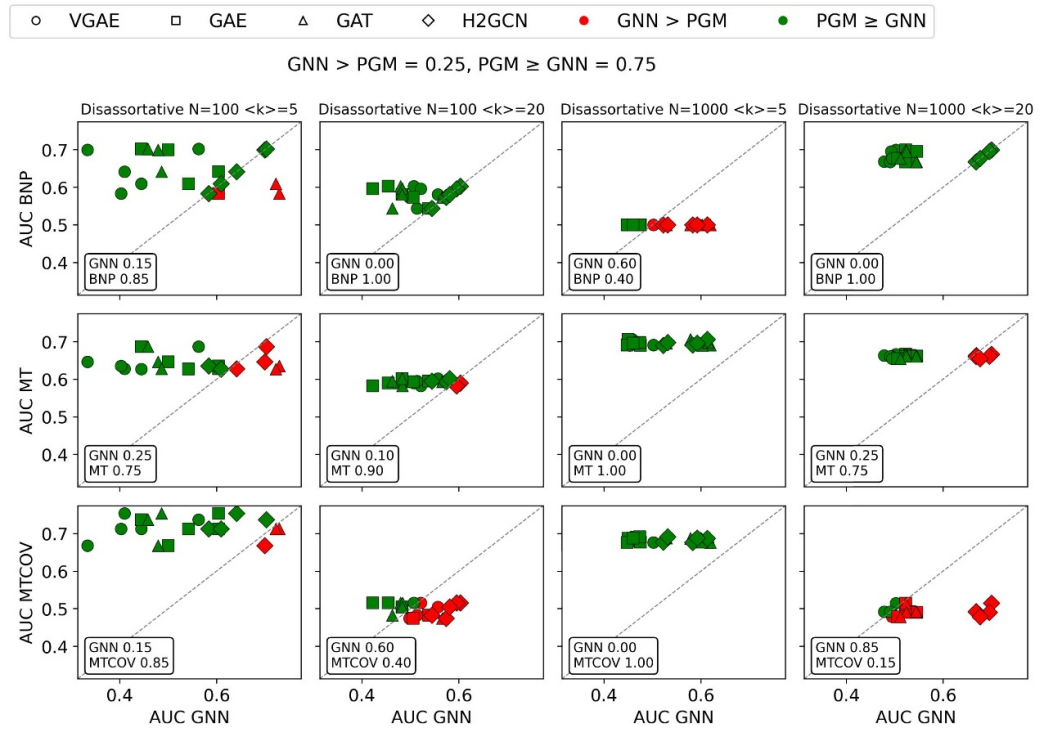


(b) Graph with 10% of edges randomly corrupted.

Figure S13. Synthetic assortative datasets: per-fold comparison between PGMs and GNNs. Results correspond to one of the 10 synthetic samples, allowing a direct comparison between the unperturbed and perturbed case. We note that the relative performance between PGMs and GNNs in the unperturbed setting is consistent across all 10 samples. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the 5 cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.



(a) Unperturbed graph (original edges).



(b) Graph with 10% of edges randomly corrupted.

Figure S14. Synthetic disassortative datasets: per-fold comparison between PGMs and GNNs. Results correspond to one of the ten synthetic samples, allowing a direct comparison between the unperturbed and perturbed case. We note that the relative performance between PGMs and GNNs in the unperturbed setting is consistent across all ten samples. For the PGMs, only MTCOV uses input features. Each point corresponds to one of the five cross-validation folds. Numbers inside the boxes are the fraction of folds where each model family outperforms the others.

ORCID iDs

Michela Lapenna  0000-0001-5293-9499

Caterina De Bacco  0000-0002-8634-0211

References

- [1] Jordan M I 2003 An introduction to probabilistic graphical models (available at: <https://api.semanticscholar.org/CorpusID:16194458>)
- [2] Holland P W, Laskey K B and Leinhardt S 1983 Stochastic blockmodels: first steps *Soc. Netw.* **5** 109–37
- [3] Wang Y J and Wong G Y 1987 Stochastic blockmodels for directed graphs *J. Am. Stat. Assoc.* **82** 8–19
- [4] Snijders T and Nowicki K 1997 Estimation and prediction for stochastic blockmodels for graphs with latent block structure *J. Classif.* **14** 75–100
- [5] Abbe E 2017 Community detection and stochastic block models: recent developments *J. Mach. Learn. Res.* **18** 6446–531 (available at: <https://api.semanticscholar.org/CorpusID:9134861>)
- [6] Scarselli F, Gori M, Tsoi A C, Hagenbuchner M and Monfardini G 2009 The graph neural network model *IEEE Trans. Neural Netw.* **20** 61–80
- [7] Defferrard M, Bresson X and Vandergheynst P 2016 Convolutional neural networks on graphs with fast localized spectral filtering *Proc. 30th Int. Conf. on Neural Information Processing Systems (NIPS'16) (Red Hook, NY, USA) (Curran Associates Inc.)* pp 3844–52 (arXiv:1606.09375v3)
- [8] Monti F, Boscaini D, Masci J, Rodolá E, Svoboda J and Bronstein M M 2017 Geometric deep learning on graphs and manifolds using mixture model CNNs *2017 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 5425–34
- [9] Veličković P 2022 Message passing all the way up (arXiv:2202.11097)
- [10] Wang J, Guo Y, Yang L and Wang Y 2024 Understanding heterophily for graph neural networks *Proc. 41st Int. Conf. on Machine Learning (Proc. Machine Learning Research)* vol 235, ed R Salakhutdinov, Z Kolter, K Heller, A Weller, N Oliver, J Scarlett and F Berkenkamp (PMLR) pp 50489–529 (available at: <https://proceedings.mlr.press/v235/wang24u.html>)
- [11] Luan S et al 2024 The heterophilic graph learning handbook: Benchmarks, models, theoretical analysis, applications and challenges (arXiv:2407.09618)
- [12] Zhu J, Gaotang Li, Yang Y-A, Zhu J, Cui X and Koutra D 2024 On the impact of feature heterophily on link prediction with graph neural networks (arXiv:2409.17475)
- [13] Blöcker C, Rosvall M, Scholtes I, and West J D 2025 Insights from network science can advance deep graph learning (arXiv:2502.01177)
- [14] Zhang M and Chen Y 2018 Link prediction based on graph neural networks *Proc. 32nd Int. Conf. on Neural Information Processing Systems (NIPS'18) (Red Hook, NY, USA) (pubCurran Associates Inc)* pp 5171–81 (arXiv:1802.09691)
- [15] Duranthon O and Zdeborova L 2023 Optimal inference in contextual stochastic block models *Trans. Mach. Learn. Res.* **2024** 9–11
- [16] Tsitsulin A, Palowitch J, Perozzi B and Müller E 2023 Graph clustering with graph neural networks *J. Mach. Learn. Res.* **24** 1–21 (available at: <https://www.jmlr.org/papers/v24/20-998.html>)
- [17] Yoon K, Liao R, Xiong Y, Zhang L, Fetaya E, Urtasun R, Zemel R and Pitkow X 2019 Inference in probabilistic graphical models by graph neural networks *2019 53rd Asilomar Conf. on Signals, Systems and Computers* pp 868–75 (arXiv:1803.07710)
- [18] Wang X and Shen H-W 2023 Gnninterpreter: a probabilistic generative model-level explanation for graph neural networks *Int. Conf. on Learning Representations (ICLR)* (arXiv:2209.07924)
- [19] Boutin R, Latouche P and Bouveyron C 2024 The deep latent position block model for the block clustering and latent representation of networks (arXiv:2412.01302)
- [20] Rubio-Madriral C, Jamadandi A and Burkholz R 2025 GNNs getting comfy: community and feature similarity guided rewiring *The 13th Int. Conf. on Learning Representations* (available at: <https://api.semanticscholar.org/CorpusID:276235753>)
- [21] Hoang N T and Maehara T 2019 Revisiting graph neural networks: all we have is low-pass filters (arXiv:1905.09550)
- [22] Alon U and Yahav E 2021 On the bottleneck of graph neural networks and its practical implications *Int. Conf. on Learning Representations (ICLR)* (available at: <https://api.semanticscholar.org/CorpusID:219558760>)
- [23] Angelini M C and Ricci-Tersenghi F 2023 Modern graph neural networks do worse than classical greedy algorithms in solving combinatorial optimization problems like maximum independent set *Nat. Mach. Intell.* **5** 29–31
- [24] Peel L, Larremore D B and Clauset A 2017 The ground truth about metadata and community detection in networks *Sci. Adv.* **3** e1602548
- [25] Newman M E J and Clauset A 2016 Structure and inference in annotated networks *Nat. Commun.* **7** 11863
- [26] Contisciani M, Power E and De Bacco C 2020 Community detection with node attributes in multilayer networks *Sci. Rep.* **10** 15736
- [27] Badalyan A, Ruggeri N'o and De Bacco C 2024 Structure and inference in hypergraphs with node attributes *Nat. Commun.* **15** 7073
- [28] De Bacco C, Power E A, Larremore D B and Moore C 2017 Community detection, link prediction and layer interdependence in multilayer networks *Phys. Rev. E* **95** 042317
- [29] Moon T K 1996 The expectation-maximization algorithm *IEEE Signal Process. Mag.* **13** 47–60
- [30] Dempster A P, Laird N M and Rubin D B 1978 Maximum likelihood from incomplete data via the em algorithm *J. R. Stat. Soc. B* **39** 1–22
- [31] Peixoto T P 2012 Entropy of stochastic blockmodel ensembles *Phys. Rev. E* **85** 056122
- [32] Hastings W K 1970 Monte Carlo sampling methods using markov chains and their applications *Biometrika* **57** 97–109
- [33] Newman M E J and Barkema G T 1999 *Monte Carlo Methods in Statistical Physics* (Oxford University Press) (available at: <https://books.google.ca/books?id=HgBREAAQBAJ>)
- [34] Gilmer J, Schoenholz S S, Riley P F, Vinyals O and Dahl G E 2017 Neural message passing for quantum chemistry *Proc. 34th Int. Conf. on Machine Learning - Volume 70 (ICML 17) (JMLR.org)* pp 1263–72 (available at: <https://proceedings.mlr.press/v70/gilmer17a.html>)
- [35] Chao L, Jianshan H, Wang T and Chu W 2020 Pairre: knowledge graph embeddings via paired relation vectors *Annual Meeting of the Association for Computational Linguistics* (available at: <https://api.semanticscholar.org/CorpusID:226281660>)

- [36] Wang X, Jiang Y, Bach N, Wang T, Huang Z, Huang F and Kewei T 2020 Automated concatenation of embeddings for structured prediction *Annual Meeting of the Association for Computational Linguistics* (available at: <https://api.semanticscholar.org/CorpusID:22290783>)
- [37] MacQueen J 1967 Some methods for classification and analysis of multivariate observations *Proc. 5th Berkeley Symp. on Mathematical Statistics and Probability* pp 281–97 (available at: <https://api.semanticscholar.org/CorpusID:6278891>)
- [38] Lloyd S 1982 Least squares quantization in pcm *IEEE Trans. Inf. Theory* **28** 129–37
- [39] Veličković P, Cucurull G, Casanova A, Romero A, Liò P and Bengio Y 2018 Graph attention networks *Int. Conf. on Learning Representations* (available at: <https://api.semanticscholar.org/CorpusID:3292002>)
- [40] Kipf T N and Welling M 2016 Variational graph autoencoders *Proc. NeurIPS Workshop on Bayesian Deep Learning* (available at: <https://api.semanticscholar.org/CorpusID:14249137>)
- [41] Zhu J, Yan Y, Zhao L, Heimann M, Akoglu L and Koutra D 2020 Beyond homophily in graph neural networks: current limitations and effective designs *Proc. 34th Int. Conf. on Neural Information Processing Systems (NIPS '20)* (Red Hook, NY, USA) (Curran Associates Inc) (arXiv:2006.11468)
- [42] Rossi E, Charpentier B, Di Giovanni F, Frasca F, Günnemann S and Bronstein M M 2023 Edge directionality improves learning on heterophilic graphs (arXiv:2305.10498)
- [43] Schlichtkrull M, Kipf T N, Bloem P, van den Berg R I and Welling M 2018 Modeling relational data with graph convolutional networks *The Semantic Web: 15th Int. Conf., ESWC 2018 (Heraklion, Crete, Greece, 3–7 June 2018)* (Springer) pp 593–607 (available at: <https://api.semanticscholar.org/CorpusID:5458500>)
- [44] Tong Z, Liang Y, Sun C, Rosenblum D S, and Lim A 2020 Directed graph convolutional network (arXiv:2004.13970)
- [45] Kollias G, Kalantzis V, Id'e T, Lozano A C and Abe N 2022 Directed graph auto-encoders *AAAI Conf. on Artificial Intelligence* (available at: <https://api.semanticscholar.org/CorpusID:247154899>)
- [46] Gong L and Cheng Q 2019 Exploiting edge features for graph neural networks *2019 IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)* pp 9203–11
- [47] Yang Z, Cohen W W and Salakhutdinov R 2016 Revisiting semi-supervised learning with graph embeddings *Proc. 33rd Int. Conf. on Int. Conf. on Machine Learning - Volume 48 (ICML 16)* (JMLR.org) pp 40–48 (available at: <https://api.semanticscholar.org/CorpusID:7008752>)
- [48] Fey M and Lenssen J E 2019 Fast graph representation learning with pytorch geometric (arXiv:1903.02428)
- [49] Zitnik M, Agrawal M and Leskovec J 2017 Predicting multicellular function through multi-layer tissue networks *Bioinformatics* **33** 1916–23
- [50] Dwivedi V P, Rampásek L, Galkin M, Parviz A, Wolf G, Luu A T and Beaini D 2022 Long range graph benchmark *Proc. 36th Int. Conf. on Neural Information Processing Systems (NIPS '22)* (Red Hook, NY, USA) (Curran Associates Inc) (available at: <https://api.semanticscholar.org/CorpusID:249712241>)
- [51] Yanardag P and Vishwanathan S V N 2015 Deep graph kernels *Proc. 21th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD '15)* (New York, NY, USA) (Association for Computing Machinery) pp 1365–74 (available at: <https://api.semanticscholar.org/CorpusID:207227372>)
- [52] Adamic L A and Glance N 2005 The political blogosphere and the 2004 U.S. election: divided they blog *Proc. 3rd Int. Workshop on Link Discovery (LinkKDD '05)* (New York, NY, USA) (Association for Computing Machinery) pp 36–43
- [53] Yin H, Benson A R, Leskovec J and Gleich D F 2017 Local higher-order graph clustering *Proc. 23rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD '17)* (New York, NY, USA) (Association for Computing Machinery) pp 555–64
- [54] Pei B, Wei B, Chang K C-C, Lei Y, and Yang B 2020 Geom-gcn: geometric graph convolutional networks (arXiv:2002.05287)
- [55] Kingma D P and Jimmy B 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)
- [56] Pan Li and Leskovec J 2022 The expressive power of graph neural networks *Graph Neural Networks: Foundations, Frontiers, and Applications* ed L Wu, P Cui, J Pei and L Zhao (Springer) pp 63–98
- [57] Errica F, Podda M, Bacciu D, and Micheli A 2019 A fair comparison of graph neural networks for graph classification (arXiv:1912.09893)
- [58] Hamilton W L, Ying R and Leskovec J 2017 Inductive representation learning on large graphs *Proc. 31st Int. Conf. on Neural Information Processing Systems (NIPS'17)* (Red Hook, NY, USA) (Curran Associates Inc) pp 1025–35 (available at: <https://api.semanticscholar.org/CorpusID:4755450>)
- [59] Keyulu X, Weihua H, Leskovec J, and Jegelka S 2018 How powerful are graph neural networks? (arXiv:1810.00826)
- [60] Cui H, Zijie L, Pan Li and Yang C 2022 On positional and structural node features for graph neural networks on non-attributed graphs *Proc. 31st ACM Int. Conf. on Information & Knowledge Management (CIKM '22)* (New York, NY, USA) (Association for Computing Machinery) pp 3898–902
- [61] Tenorio V M, Navarro M, Segarra S and Marques A G 2024 Recovering missing node features with local structure-based embeddings *ICASSP 2024 - 2024 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)* pp 9931–5
- [62] Taguchi H, Liu X and Murata T 2021 Graph convolutional networks for graphs containing missing features *Future Gener. Comput. Syst.* **117** 155–68
- [63] Sato R 2023 Graph neural networks can recover the hidden features solely from the graph structure *Proc. 40th Int. Conf. on Machine Learning (ICML'23)* (JMLR.org) (available at: <https://api.semanticscholar.org/CorpusID:256274851>)
- [64] Cui H, Dai W, Zhu Y, Kan X, Gu A A C, Lukemire J, Zhan L, Lifang H, Guo Y and Yang C 2022 Braingb: a benchmark for brain network analysis with graph neural networks *IEEE Trans. Med. Imaging* **42** 493–506
- [65] Rossi E, Kenlay H, Gorinova M I, Chamberlain B P, Dong X and Bronstein M M 2022 On the unreasonable effectiveness of feature propagation in learning on graphs with missing node features *Learning on Graphs Conference* (PMLR) pp 11–1 (arXiv:2111.12128)
- [66] Decelle A, Krzakala F, Moore C and Zdeborova L 2011 Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications *Phys. Rev. E* **84** 066106
- [67] Ruggeri N'ò, Contisciani M, Battiston F and De Bacco C 2023 Community detection in large hypergraphs *Sci. Adv.* **9** eadg9159
- [68] Hood J, De Bacco C, and Schein A 2025 Broad spectrum structure discovery in large-scale higher-order networks (arXiv:2505.21748)
- [69] Xiang Li, Zhu R, Cheng Y, Shan C, Luo S, Dongsheng Li and Qian W 2022 Finding global homophily in graph neural networks when meeting heterophily *Int. Conf. on Machine Learning* PMLR pp 13242–56 (available at: <https://api.semanticscholar.org/CorpusID:248810929>)
- [70] Di Giovanni F, Rowbottom J, Chamberlain B P, Markovich T and Bronstein M M 2022 Understanding convolution on graphs via energies *Trans. Mach. Learn. Res.* **2023** 5–10 (available at: <https://api.semanticscholar.org/CorpusID:258987606>)

- [71] Di Francesco A G, Caso F, Bucarelli M S, and Silvestri F 2024 Link prediction under heterophily: a physics-inspired graph neural network approach (arXiv:2402.14802)
- [72] Yao M, Liu X, Shah N, and Tang J 2021 Is homophily a necessity for graph neural networks? (arXiv:2106.06134)
- [73] Rusch T K, Bronstein M M, and Mishra S 2023 A survey on oversmoothing in graph neural networks (arXiv:2303.10993)
- [74] Xinyi W, Ajorlou A, Zihui W and Jadbabaie A 2023 Demystifying oversmoothing in attention-based graph neural networks *Proc. 37th Int. Conf. on Neural Information Processing Systems (NIPS '23)* (Red Hook, NY, USA) (Curran Associates Inc) (available at: <https://api.semanticscholar.org/CorpusID:258888081>)
- [75] Oono K and Suzuki T 2019 Graph neural networks exponentially lose expressive power for node classification (available at: <https://api.semanticscholar.org/CorpusID:209994765>)
- [76] Cai C and Wang Y 2020 A note on over-smoothing for graph neural networks (arXiv:2006.13318)
- [77] Luan S, Hua C, Qincheng L, Zhu J, Zhao M, Zhang S, Chang X-W and Precup D 2022 Revisiting heterophily for graph neural networks *Proc. 36th Int. Conf. on Neural Information Processing Systems (NIPS '22)* (Red Hook, NY, USA) (Curran Associates Inc) (available at: <https://api.semanticscholar.org/CorpusID:252907555>)
- [78] Yunchong S, Zhou C, Wang X, and Lin Z 2023 Ordered gnn: ordering message passing to deal with heterophily and over-smoothing (arXiv:2302.01524)
- [79] Epping B, René A, Helias M and Schaub M T 2025 Graph neural networks do not always oversmooth *Proc. 38th Int. Conf. on Neural Information Processing Systems (NIPS '24)* (Red Hook, NY, USA) (Curran Associates Inc) (arXiv:2406.02269)
- [80] Xuhong Li, Xiong H, Xingjian Li, Xuanyu W, Zhang X, Liu J, Bian J and Dou D 2022 Interpretable deep learning: interpretation, interpretability, trustworthiness and beyond *Knowl. Inf. Syst.* **64** 3197–234
- [81] Huang Q, Yamada M, Tian Y, Singh D and Chang Y 2022 Graphlime: local interpretable model explanations for graph neural networks *IEEE Trans. Knowl. Data Eng.* **35** 6968–72
- [82] Mehta N, Carin L, and Rai P 2019 Stochastic blockmodels meet graph neural networks (arXiv:1905.05738)
- [83] Tian F, Gao B, Cui Q, Chen E and Liu T-Y 2014 Learning deep representations for graph clustering *Proc. 28th AAAI Conf. on Artificial Intelligence (AAAI'14)* (AAAI Press) pp 1293–9 (available at: <https://api.semanticscholar.org/CorpusID:2701448>)
- [84] Wang C, Pan S, Ruiqi H, Long G, Jiang J and Zhang C 2019 Attributed graph clustering: a deep attentional embedding approach *Proc. 28th Int. Joint Conf. on Artificial Intelligence (IJCAI'19)* (AAAI Press) pp 3670–6
- [85] Shin H and Lee K Y 2024 Deep k-means clustering based on graph neural networks: leveraging cohesion and separation in graph nodes 2024 *IEEE Int. Conf. on Big Data and Smart Computing (BigComp)* pp 190–3 (available at: <https://api.semanticscholar.org/CorpusID:269090741>)
- [86] van der Maaten L and Hinton G 2008 Visualizing data using t-sne *J. Mach. Learn. Res.* **9** 2579–605 (available at: <https://api.semanticscholar.org/CorpusID:5855042>)
- [87] Contisciani M, Hobbhahn M, Power E A, Hennig P and De Bacco C 2025 Flexible inference in heterogeneous and attributed multilayer networks *PNAS Nexus* **4** gaf005
- [88] Wang X and Zhang M 2022 How powerful are spectral graph neural networks *Int. Conf. on Machine Learning* (PMLR) pp 23341–62 (available at: <https://api.semanticscholar.org/CorpusID:248987544>)
- [89] Srinivasan B and Ribeiro B 2019 On the equivalence between positional node embeddings and structural graph representations (arXiv:1910.00452)
- [90] Peixoto T P 2014 Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models *Phys. Rev. E* **89** 012804
- [91] Kipf T N and Welling M 2017 Semi-supervised classification with graph convolutional networks (arXiv:1609.02907)
- [92] Cover T M and Thomas J A 2006 *Elements of Information Theory* (Wiley Series in Telecommunications and Signal Processing) (Wiley) (available at: <https://api.semanticscholar.org/CorpusID:53827957>)
- [93] Kingma D P and Welling M 2019 An introduction to variational autoencoders *Found. Trends Mach. Learn.* **12** 307–92
- [94] Rezende D J, Mohamed S and Wierstra D 2014 Stochastic backpropagation and approximate inference in deep generative models *Int. Conf. on Machine Learning* (available at: <https://api.semanticscholar.org/CorpusID:16895865>)