

# PRIOR-INFORMED FLOW MATCHING FOR GRAPH RECONSTRUCTION

000  
001  
002  
003  
004  
005  
006  
007  
008  
009  
010  
011  
012  
013  
014  
015  
016  
017  
018  
019  
020  
021  
022  
023  
024  
025  
026  
027  
028  
029  
030  
031  
032  
033  
034  
035  
036  
037  
038  
039  
040  
041  
042  
043  
044  
045  
046  
047  
048  
049  
050  
051  
052  
053  
Anonymous authors  
Paper under double-blind review

## ABSTRACT

We introduce *Prior-Informed Flow Matching (PIFM)*, a conditional flow model for graph reconstruction. Reconstructing graphs from partial observations remains a key challenge; classical embedding methods often lack global consistency, while modern generative models struggle to incorporate structural priors. PIFM bridges this gap by integrating embedding-based priors with continuous-time flow matching. Grounded in a permutation equivariant version of the distortion-perception theory, our method first uses a prior, such as graphons or GraphSAGE/node2vec, to form an informed initial estimate of the adjacency matrix based on local information. It then applies rectified flow matching to refine this estimate, transporting it toward the true distribution of clean graphs and learning a global coupling. Experiments on different datasets demonstrate that PIFM consistently enhances classical embeddings, outperforming them and state-of-the-art generative baselines in reconstruction accuracy.

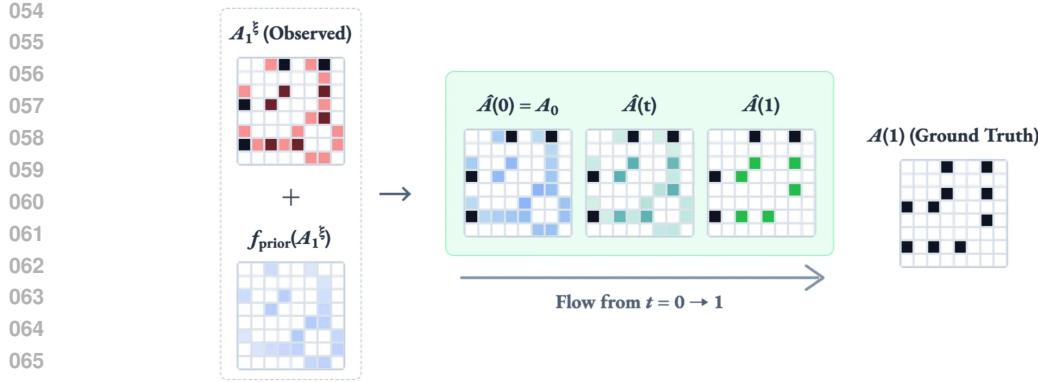
## 1 INTRODUCTION

Graph generative models have seen remarkable progress in recent years, enabling the synthesis of realistic graph structures in domains such as drug design (Yang et al., 2024) and social networks (Grover et al., 2019). In particular, diffusion-based (Niu et al., 2020; Jo et al., 2022; Vignac et al., 2023) and flow-based (Qin et al., 2025; Eijkelboom et al., 2024) approaches have emerged as state-of-the-art. While these models excel at *unconditional* generation and property-controlled generation, their application to inverse problems, and in particular, the reconstruction of a graph from partial observations, remains a fundamental open problem.

Graph reconstruction is a long-standing problem, traditionally framed as a link prediction task. Early transductive methods, such as Node2Vec (Grover & Leskovec, 2016; Perozzi et al., 2014), model edges independently and fail to capture global structural information. While inductive methods (Zhang & Chen, 2018) like GraphSAGE (Hamilton et al., 2017) can capture expressive local patterns, they still lack a global perspective on the graph’s structure. Conversely, recent generative models adapted from image inpainting (Vignac et al., 2023; Trivedi et al., 2024) or guided by posterior sampling (Sharma et al., 2024; Tenorio et al., 2025) can produce plausible completions but are not optimized for the faithful recovery of the ground truth. This leaves open a critical gap: classical and heuristic-based methods are local, while modern solvers are not designed for exact reconstruction.

In this work, we bridge this gap by introducing **Prior-Informed Flow Matching (PIFM)**, a flow-based model designed for high-fidelity graph reconstruction. We reformulate the problem through the lens of the perception-distortion trade-off (Blau & Michaeli, 2018), which postulates that an optimal estimator can be constructed in two stages (Freirich et al., 2021; Ohayon et al., 2025): (*i*) predicting the Minimum Mean Squared Error (MMSE) estimator from local information, and (*ii*) learning an optimal transport map from this initial estimate to the ground-truth graph distribution.

Our method approximates this two-step solution. For (*i*), we represent the posterior mean as the expected value of a Bernoulli latent variable model with unknown probabilities, where the latent structure is estimated using inductive (dataset-informed, such as graphons (Lovász, 2012) and GraphSAGE (Hamilton et al., 2017)) or transductive estimators (instance-specific, such as node2vec (Grover & Leskovec, 2016)). Then, for (*ii*), we approximate the optimal transport step using a rectified flow model (Liu et al., 2023; Albergo et al., 2023), which maps the posterior mean to



067  
068  
069  
070  
071  
072  
073  
074  
075  
076  
077  
078  
079  
080  
081  
082  
083  
084  
085  
086  
087  
088  
089  
090  
091  
092  
093  
094  
095  
096  
097

Figure 1: Overview of the **Prior-Informed Flow Matching (PIFM)** graph reconstruction framework. Starting from a partially observed adjacency matrix  $A_1^\xi = \xi \odot A$ , where  $\xi$  denotes a mask, we form an initialization  $A_0$  by combining the observed entries with prior predictions  $f_{\text{prior}}(A_1^\xi)$  obtained with an element-wise predictor. In dark red we denote the true edges that are masked, while in light red those masked position that do not have an edge between nodes. A rectified flow then interpolates linearly from  $A_0$  to the ground-truth graph  $A_1 = A$ , learning global structural information from a coupling of all the edges. The intermediate states  $A_t$  improve on the prior-informed initialization, enabling recovery of the missing edges.

the distribution of clean graphs (see Fig. 1). Importantly, our architectures are permutation-equivariant, ensuring a permutation-invariant parameterization of the posterior distribution.

We validate the advantages of PIFM through experiments on datasets with diverse characteristics, including dense and sparse graphs. Our results show that PIFM effectively integrates structural priors with flow-based modeling and can be interpreted as a form of graph inpainting, where missing edges are inferred through a learned interpolation process.

Our contributions are as follows:

- We introduce a novel formulation for graph reconstruction based on a permutation-equivariant distortion-perception trade-off.
- We propose PIFM, a new estimator based on flow matching that defines a prior-informed source distribution using embeddings from latent graph models. PIFM enhances these initial embeddings by learning a global structural coupling.
- We empirically validate our approach on link prediction, and two *blind* versions termed expansion (recover the missing edges) and denoising (removing the spurious edges), showing that PIFM significantly improves the reconstruction performance of predictors that rely solely on local information.

## 2 RELATED WORKS

**Flow/Diffusion models on graphs.** Diffusion and flow-based graph generative models have shown impressive performance in recent years. Early models, namely EDP-GNN (Niu et al., 2020) and GDSS (Jo et al., 2022), employ score-based *continuous* diffusion over a relaxation of the graph structure. However, given that graphs are inherently discrete, subsequent work has explored discrete diffusion processes (Austin et al., 2021). Models like DiGress (Vignac et al., 2023) demonstrated the effectiveness of this approach, which has been further advanced by discrete flow-based models like DeFoG (Qin et al., 2025) and variational approaches like CatFlow (Eijkelboom et al., 2024). A common point of these models is their reliance on a simple source distribution, such as Gaussian (continuous) or uniform (discrete) noise. While effective for unconstrained generation, recent work on image-based inverse problems demonstrates the advantages of learning a data-dependent flow, using a prior-informed source distribution (Albergo et al., 2023; Delbracio & Milanfar, 2024; Ohayon et al., 2025).

108 **Graph topology inference via flow/diffusion-based solvers.** Graph topology inference – the  
 109 task of recovering hidden edges from a partially observed graph – is a long-standing inverse prob-  
 110 lem (Segarra et al., 2017; Dong et al., 2016). Several methods adapt diffusion for constrained graph  
 111 generation, which is related to but distinct from topology inference. DiGress (Vignac et al., 2023)  
 112 introduced an inpainting mechanism, inspired by Repaint (Lugmayr et al., 2022), to generate graph  
 113 structures consistent with a partial observation. Similarly, in Trivedi et al. (2024) a similar mech-  
 114 anism is used for completing partially observed graphs. PRODIGY (Sharma et al., 2024) enforces  
 115 hard constraints by projecting the graph estimate onto a feasible set at each sampling step. More  
 116 recently, GGDiff (Tenorio et al., 2025) incorporates a guidance mechanism as a flexible alternative  
 117 to inpainting. However, all these methods are designed for *constrained generation* (e.g., molecule  
 118 generation with a given scaffold) rather than *recovering masked edges from a partially observed  
 119 graph*. Hence, to the best of our knowledge, designing a diffusion-based model explicitly for graph  
 120 topology inference remains an open problem.

### 121 3 BACKGROUND

122 We represent an undirected graph  $\mathcal{G}_0 = \{\mathcal{V}, \mathcal{E}\}$ , where  $\mathcal{V}$  denotes the nodes and  $\mathcal{E}$  the edges, by its  
 123 binary symmetric adjacency matrix  $\mathbf{A}_0 \in \mathbb{R}^{N \times N}$ .

124 **Continuous flow matching for graph generation.** Flow matching (Albergo et al., 2023; Lipman  
 125 et al., 2023) is a family of generative models that defines a continuous-time transport map from  
 126 samples  $\mathbf{A}_0$  drawn from a source distribution  $p_0$  to samples  $\mathbf{A}_1$  from a target distribution  $p_1$ . It is  
 127 governed by the ODE  
 128

$$129 \quad d\mathbf{A}_t = v(\mathbf{A}_t, t) dt, \quad (1)$$

130 where  $v(\cdot, t)$  is a velocity field and  $\mathbf{A}_t$  denotes a forward process, also known as stochastic interpolant,  
 131 for  $t \in [0, 1]$ . Typically,  $p_0$  is a tractable distribution (e.g., a Gaussian distribution), while  $p_1$   
 132 corresponds to the data distribution. To generate new samples, one must specify both  $\mathbf{A}_t$  and  $v$ . A  
 133 common choice for the forward process is  $\mathbf{A}_t = \alpha_t \mathbf{A}_0 + \beta_t \mathbf{A}_1$ , where  $\alpha_t$  and  $\beta_t$  are differentiable  
 134 functions such that  $\alpha_0 = 1$ ,  $\beta_0 = 0$  and  $\alpha_1 = 0$ ,  $\beta_1 = 1$ . Differentiating this path gives a velocity  
 135  $v(\mathbf{A}_t, t) = \dot{\alpha}_t \mathbf{A}_0 + \dot{\beta}_t \mathbf{A}_1$ . Despite its closed-form, this expression depends explicitly on  $\mathbf{A}_1$ , making  
 136 it impractical since the target is unknown at inference/sampling. To circumvent this, we instead  
 137 consider  $v(\mathbf{A}_t, t) = \mathbb{E}_{\mathbf{A}_0, \mathbf{A}_1} [\dot{\alpha}_t \mathbf{A}_1 + \dot{\beta}_t \mathbf{A}_0 \mid \mathbf{A}_t]$ , the conditional expectation of the velocity given  
 138  $\mathbf{A}_t$  (Albergo et al., 2023), which is then approximated with a neural network  $v_\theta$ . The network is  
 139 trained using a mean squared error loss:  
 140

$$141 \quad \mathbb{E}_{t, \mathbf{A}_0, \mathbf{A}_1} \left[ \left\| v_\theta(\mathbf{A}_t, t) - (\dot{\alpha}_t \mathbf{A}_1 + \dot{\beta}_t \mathbf{A}_0) \right\|_2^2 \right]. \quad (2)$$

142 In particular, this formulation does not require  $\mathbf{A}_0$  and  $\mathbf{A}_1$  to be independent; in fact, they might be  
 143 sampled from a joint distribution, allowing for richer transport plans in cases where paired data is  
 144 available. This has been exploited to solve inverse problems on images (Ohayon et al., 2025; Albergo  
 145 et al., 2024; Delbracio & Milanfar, 2024), and is directly related to our proposed method, as described  
 146 later.

147 Throughout this work, we consider the *rectified flow* case (Liu et al., 2023), where  $\alpha_t = 1 - t$  and  
 148  $\beta_t = t$ . As shown in Tong et al. (2024), the velocity field associated with this linear path approximates  
 149 the optimal transport vector field when the joint distribution  $p(\mathbf{A}_0, \mathbf{A}_1)$  closely resembles the optimal  
 150 coupling between the marginals  $p(\mathbf{A}_0)$  and  $p(\mathbf{A}_1)$ . We deferred to Appendix C.2 a more detailed  
 151 background on generative models on graphs beyond continuous flow matching, including diffusion-  
 152 based models, as well as related works.

### 153 4 METHOD

154 In Section 4.1, we introduce the distortion-perception trade-off (Blau & Michaeli, 2018) for graphs.  
 155 Then, in Section 4.2, we introduce methods for approximating the posterior mean. In Section 4.3, we  
 156 describe our implementation of the flow model to transport the predicted mean to the ground-truth  
 157 graphs.

162 4.1 GRAPH TOPOLOGY INFERENCE AS A DISTORTION-PERCEPTION TRADE-OFF  
163164 We aim to reconstruct the ground-truth adjacency matrix  $\mathbf{A}$  of graph  $\mathcal{G}$  from a partially observed  
165 version, denoted by  $\mathbf{A}^{\mathcal{O}}$ . This task can be formalized through the following distortion-perception  
166 function:

167 
$$D(P) = \min_{p(\hat{\mathbf{A}}|\mathbf{A}^{\mathcal{O}})} \left\{ \mathbb{E}_{p(\mathbf{A}, \hat{\mathbf{A}})} [\|\mathbf{A} - \hat{\mathbf{A}}\|_F^2] : d(p_{\mathbf{A}}, p_{\hat{\mathbf{A}}}) \leq P \right\}, \quad (3)$$
  
168

169 where  $\hat{\mathbf{A}}$  is an estimator of  $\mathbf{A}$  given the observation  $\mathbf{A}^{\mathcal{O}}$ , and  $d(p_{\mathbf{A}}, p_{\hat{\mathbf{A}}})$  is a divergence between the  
170 distributions  $p_{\mathbf{A}}$  (the distribution of clean graphs) and  $p_{\hat{\mathbf{A}}}$ . Although we adopt MSE as the distortion  
171 measure in (3), the formulation is general and supports other distortion metrics between the true  
172 and predicted graphs (Blau & Michaeli, 2018). The function in (3) has been extensively studied  
173 in the image domain (Freirich et al., 2021), where different values of  $P$  correspond to estimators  
174 with varying characteristics in terms of average accuracy (distortion), and the degree to which the  
175 reconstructed signal looks like the ground truth (perception). We adapt this framework to graphs by  
176 estimating the matrix  $\mathbf{A}$ , and accounting for symmetry constraints due to the permutation invariance  
177 of graph representations.178 Among all possible values of  $P$ , the most studied cases are  $P = \infty$  and  $P = 0$ . The former  
179 corresponds to the distortion function  $D(P = \infty)$ , whose solution is the posterior mean estimator  
180  $\hat{\mathbf{A}}^* = \mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]$ . This estimator minimizes distortion, but does not impose constraints on the  
181 distribution of outputs, potentially leading to unrealistic graphs. In contrast, when  $P = 0$ , the  
182 estimator achieves a perfect perceptual reconstruction, meaning that the recovered graph has the same  
183 structural properties than the original graph; in this case,  $p_{\mathbf{A}} = p_{\hat{\mathbf{A}}}$ . As shown in Freirich et al. (2021),  
184 the corresponding estimator can be obtained by solving the following optimal transport problem

185 
$$p_{\hat{\mathbf{A}}, \hat{\mathbf{A}}^*}^* = \underset{p \in \Pi(p_{\mathbf{A}}, p_{\hat{\mathbf{A}}^*})}{\operatorname{argmin}} \mathbb{E}[\|\hat{\mathbf{A}} - \hat{\mathbf{A}}^*\|_F^2], \quad (4)$$
  
186

187 where  $\Pi(p_{\mathbf{A}}, p_{\hat{\mathbf{A}}^*})$  is the set of all joint distributions (couplings) with fixed marginals  $p_{\mathbf{A}}$  and  $p_{\hat{\mathbf{A}}^*}$ .  
188 Thus, finding the estimator  $\hat{\mathbf{A}}$  associated with  $D(0)$  boils down to solving the optimal transport  
189 problem between the distribution of clean graphs  $p_{\mathbf{A}}$  and of the MMSE estimator  $p_{\hat{\mathbf{A}}^*}$ . We can  
190 approximate this by (i) computing  $\hat{\mathbf{A}}^* = \mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]$  given an observation  $\mathbf{A}^{\mathcal{O}}$  and (ii) sampling  
191 from the conditional distribution  $p(\mathbf{A} | \hat{\mathbf{A}}^*)$ . Intuitively, this approach builds the final prediction by  
192 refining the initial guess  $\hat{\mathbf{A}}^*$ .193 While the solution for  $P = \infty$  achieves lower MSE, it may produce outputs that deviate from the  
194 structural properties of the original data. This mismatch is problematic in settings like conditional  
195 molecular generation, where the generated molecule must satisfy strict chemical validity constraints.  
196 For such applications, the solution with  $P = 0$  is more appropriate, as it guarantees that the generated  
197 samples are structurally consistent with the data distribution. Therefore, this work focuses on the  
198 case  $P = 0$ .200 **Permutation invariance on graphs: An additional constraint.** Unlike images, graphs lack canonical  
201 node ordering, which imposes symmetry constraints on the data distribution and the reconstruction  
202 function. First, the ground-truth distribution  $p(\mathbf{A})$  is *permutation invariant*, meaning that for any per-  
203 mutation matrix  $\mathbf{P}_{\pi}$  associated with a node relabeling  $\pi$ , it holds that  $p(\mathbf{A}) = p(\mathbf{P}_{\pi}^T \mathbf{A} \mathbf{P}_{\pi})$ . Second,  
204 the estimator  $\hat{\mathbf{A}} = f(\mathbf{A}^{\mathcal{O}})$  must be *permutation equivariant*, i.e.,  $f(\mathbf{P}_{\pi}^T \mathbf{A}^{\mathcal{O}} \mathbf{P}_{\pi}) = \mathbf{P}_{\pi}^T f(\mathbf{A}^{\mathcal{O}}) \mathbf{P}_{\pi}$ ,  
205 ensuring that relabeling the input results in a consistently relabeled output. These conditions are  
206 necessary for solutions to (3) and (4) to be invariant to the node labeling. We now describe how to  
207 implement the solution to (4).208 4.2 APPROXIMATING THE POSTERIOR MEAN  
209210 As discussed in Section 4.1, our goal is to approximate the conditional mean  $\mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]$  with a  
211 *permutation equivariant* estimator. Before moving to particular parameterizations of the conditional  
212 mean, we introduce two assumptions.213 **AS 1.** We assume each edge in  $\mathbf{A} \in \{0, 1\}^{n \times n}$  follows a Bernoulli distribution whose probabilities  
214 depend on latent node variables  $\mathbf{z}_1, \dots, \mathbf{z}_n \in \mathcal{Z}$  such that:

215 
$$A_{ij} \sim \text{Bernoulli}(f(\mathbf{z}_i, \mathbf{z}_j)), \quad 1 \leq i < j \leq n. \quad (5)$$

216 The function  $f$  maps pairs of latent variables to edge probabilities, i.e.,  $f(\mathbf{z}_i, \mathbf{z}_j) = P(A_{ij} | \mathbf{z}_i, \mathbf{z}_j) =$   
 217  $p_{ij}$ .

218 **AS 2.** We assume that the edges are conditionally independent given the latent structure, i.e., given  
 219 the latent structure  $Z = \{\mathbf{z}_1, \dots, \mathbf{z}_n\}$ , we have:

$$221 \quad P(\mathbf{A} | Z) = \prod_{1 \leq i < j \leq n} P(A_{ij} | \mathbf{z}_i, \mathbf{z}_j). \quad (6)$$

224 Under these two assumptions, and assuming access to the mapping  $\mathbf{z}^{-1} : \mathbf{A} \rightarrow \mathcal{Z}$ , the posterior  
 225 mean can be computed element-wise as  $\mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}] = P(\mathbf{A} | \mathbf{z}^{-1}(\mathbf{A}^{\mathcal{O}}))$ . We adopt two different  
 226 type of priors : (i) inductive methods, represented by *graphons* (Lovász, 2012; Avella-Medina  
 227 et al., 2018), which are bounded, symmetric and measurable functions  $\mathcal{W} : [0, 1]^2 \rightarrow [0, 1]$ , and  
 228 *GraphSAGE* (Hamilton et al., 2017), a GNN-based estimator and (ii) transductive ones, obtained  
 229 from *node2vec* (Grover & Leskovec, 2016), which provides an instance-level learned probabilistic  
 230 model.

231 **Posterior mean using inductive methods (dataset-informed).** We approximate the posterior mean  
 232 using two distinct dataset-informed, inductive approaches: graphons and GraphSAGE.

234 A *graphon*, defined as a symmetric function  $\mathcal{W} : [0, 1]^2 \rightarrow [0, 1]$ , serves as a generative model for a  
 235 family of graphs:

$$236 \quad z_i \sim \text{Uniform}[0, 1], \quad i = 1, \dots, n, \quad (7)$$

$$237 \quad A_{ij} \sim \text{Bernoulli}(\mathcal{W}(z_i, z_j)), \quad 1 \leq i < j \leq n.$$

239 Graphons provide a functional representation of exchangeable random graphs where the conditional  
 240 edge probability is  $[\mathbb{E}[\mathbf{A} | \mathbf{z}]]_{ij} = \mathcal{W}(z_i, z_j)$ . This offers a natural, permutation-equivariant  
 241 framework for estimating the posterior mean, though it requires access to the inverse mapping  
 242  $z_i = [\mathbf{z}^{-1}(\mathbf{A}^{\mathcal{O}})]_i$ . Since  $\mathcal{W}$  is unknown, we estimate it using Scalable Implicit Graphon Learning  
 243 (SIGL) (Azizpour et al., 2025), which combines a graph neural network (GNN) encoder with an  
 244 implicit neural representation (INR). SIGL operates in three steps: (1) a GNN-based sorting step to  
 245 estimate latent node positions  $\mathbf{z}$ ; (2) a histogram approximation of the sorted adjacency matrices; and  
 246 (3) learning a graphon parameterization  $f_{\phi}$  by minimizing its error against the histograms. A key  
 247 feature of SIGL is its ability to recover the inverse mapping  $\mathbf{z}^{-1}$ , making it uniquely suitable for our  
 248 model (Xia et al., 2023).

248 As an alternative inductive method, we use *GraphSAGE* (Hamilton et al., 2017). We train the model  
 249 on the partially observed graphs in the dataset to produce node embeddings  $\{\mathbf{z}_i\}_{i=1}^N$ . From these  
 250 embeddings, we train a single logistic predictor on Hadamard edge features  $(\mathbf{z}_i \odot \mathbf{z}_j)$  to estimate edge  
 251 probabilities. The resulting conditional mean is parameterized as  $[\mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]]_{ij} = f_{\phi}(\mathbf{z}_i \odot \mathbf{z}_j)$ .

253 **Posterior mean using transductive methods (instance-specific).** For a transductive approach,  
 254 we use *node2vec* to learn an instance-specific embedding and predictor for each graph. Similar  
 255 to the GraphSAGE method, we first train node2vec on a partially observed graph to obtain node  
 256 embeddings  $\{\mathbf{z}_i\}_{i=1}^N$ . However, in contrast to the single predictor used for GraphSAGE, we fit a  
 257 distinct, *per-graph* logistic link predictor on Hadamard edge features with balanced negative sampling.  
 258 This yields the same conditional mean parameterization,  $[\mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]]_{ij} = f_{\phi}(\mathbf{z}_i \odot \mathbf{z}_j)$ , but with a  
 259 predictor  $f_{\phi}(\cdot)$  that is unique to each graph instance. At inference time, this instance-specific model  
 260 is used to evaluate all masked pairs to compute the posterior mean.

### 262 4.3 LEARNING THE FLOW MODEL

264 We now approximate the posterior density  $p(\mathbf{A}, \hat{\mathbf{A}}^*)$  by learning a flow model. As explained  
 265 in Section 3, we need to specify the forward path  $\mathbf{A}_t$  and the velocity field  $v$ . For the former,  
 266 inspired by Ohayon et al. (2025), we incorporate prior information as the initialization of the  
 267 forward path; with slight abuse of notation, we denote  $f_{\text{prior}}$  as the prediction of the full graph (i.e.,  
 268  $f_{\text{prior}}(\mathbf{A}^{\mathcal{O}}) \triangleq \mathbb{E}[\mathbf{A} | \mathbf{A}^{\mathcal{O}}]$ ). Specifically, we compute the sample  $\mathbf{A}_0$  from the source distribution as  
 269 follows:

$$270 \quad \mathbf{A}_0 = \xi \odot \mathbf{A} + (1 - \xi) \odot (f_{\text{prior}}(\xi \odot \mathbf{A}) + \epsilon_s), \quad (8)$$

270 where  $\mathbf{A}$  is the ground-truth graph,  $\xi$  is the corresponding mask (taking value 1 for the observed pairs  
 271 of nodes and 0 otherwise), and  $f_{\text{prior}}(\xi \odot \mathbf{A})$  is our approximate MMSE estimator for the masked  
 272 edges. We also add a small amount of noise  $\epsilon_s \sim \mathcal{N}(0, \sigma_s^2)$  following Albergo et al. (2024). We  
 273 define  $\mathbf{A}_1 = \mathbf{A}$  for the target distribution.

274 Regarding the velocity field  $v$ , we use the architecture from Jo et al. (2022), a GNN-based network  
 275 that yields a permutation-equivariant parameterization (see Appendix E for details). Combined with  
 276 the assumptions for the posterior mean parameterization in Section 4.2, this gives a permutation-  
 277 invariant parameterization of the target density  $p(\mathbf{A}_1)$ . Since graphs are exchangeable, the target  
 278 density should not depend on node ordering, making permutation invariance a desirable property.  
 279 This is formalized in Theorem 1.

280 **Theorem 1.** *Let the prior estimator  $f_{\text{prior}} : \mathbb{R}^{N \times N} \rightarrow \mathbb{R}^{N \times N}$  and the velocity field  $v_\theta : \mathbb{R}^{N \times N} \times$   
 281  $[0, 1] \rightarrow \mathbb{R}^{N \times N}$  both be **permutation-equivariant**. For any ground-truth graph  $\mathbf{A}_1$  and mask  $\xi$ ,  
 282 define the flow path from  $t = 0$  to  $t = 1$  as:*

$$283 \quad \mathbf{A}_t = (1 - t)\mathbf{A}_0 + t\mathbf{A}_1, \quad \text{where} \quad \mathbf{A}_0 = \xi \odot \mathbf{A}_1 + (1 - \xi) \odot f_{\text{prior}}(\xi \odot \mathbf{A}_1).$$

284 Then the estimated density for  $\mathbf{A}_1$ , computed as

$$285 \quad \log p(\mathbf{A}_1) = \log p(\mathbf{A}_0) - \int_0^1 \text{tr} \left( \frac{\partial v_\theta(\mathbf{A}_t, t)}{\partial \mathbf{A}_t} \right) dt, \quad (9)$$

286 is guaranteed to be **permutation-invariant**.

287 The proof can be found in Appendix D.1. Since we know that the target probability density should  
 288 be permutation invariant, Theorem 1 guarantees that we are introducing the right inductive bias by  
 289 learning a distribution within the family of permutation invariant distributions.

290 **Final algorithm.** In Alg. 1, we describe our training and sampling algorithms. In essence, PIFM is  
 291 a general framework that learns a global graph structure to enhance simple, conditionally independent  
 292 edge-wise priors.

293 To illustrate what we mean by *learning a global and dependent predictor*, we now describe a toy  
 294 experiment. Consider a four-node graph  $\mathcal{G}$  (see Fig. 2 (a)) where the goal is to predict the diagonal  
 295 edges under a specific constraint: the only valid outcomes are that both edges are present or both are  
 296 absent, i.e.,  $\mathcal{E} = \{[e_{02} = 1, e_{13} = 1], [e_{02} = 0, e_{13} = 0]\}$ . Moreover, we assume that the probability  
 297 of observing the first case is 0.6, while the second one is 0.4.

298 We first train an edge-wise prior using node2vec, which yields a probability of 0.6 for each diagonal  
 299 edge. Crucially, because node2vec models each edge prediction independently, this prior is misspecified.  
 300 A standard predictor based on this prior would always predict  $[1, 1]$  if used as conditional mean  
 301 or, if sampling were to be performed, could generate invalid predictions such as  $[1, 0]$ .

302 We then train a flow model using this node2vec prior to construct the initial state  $\mathbf{A}_0$  as in (8).  
 303 After training (see Appendix E for details), we generate 200 samples, illustrated in Fig. 2(b); the  
 304 proportion of each mode is shown in Fig. 2(c). The results clearly demonstrate that the flow model  
 305 (i) successfully leverages global information, learning a *probabilistic coupling* between the edges, to  
 306 generate samples only from the two valid states, and (ii) learns the probability of each mode.

## 311 5 EXPERIMENTS

### 312 5.1 SETUP

313 We evaluate our method on three graph datasets: IMDB-B, PROTEINS, and ENZYMES. Thus, we  
 314 focus on families of graphs that are diverse to show that our model learns a general predictor. Future  
 315 work will focus on scaling to larger graphs, such as Cora. Each dataset is split into 85% train, 10%  
 316 validation, and 5% test graphs, and we evaluate reconstruction quality under two masking levels (10%  
 317 and 50% of edges, masks generated uniformly at random). The implementation details are provided  
 318 in Appendix E.

319 **Evaluation metrics.** Performance is measured exclusively on masked edges. We report both  
 320 threshold-dependent classification metrics (FPR, FNR) and threshold-independent metrics (ROC-  
 321 AUC, AP). Threshold-dependent metrics are computed by binarizing predictions at a fixed cutoff of

---

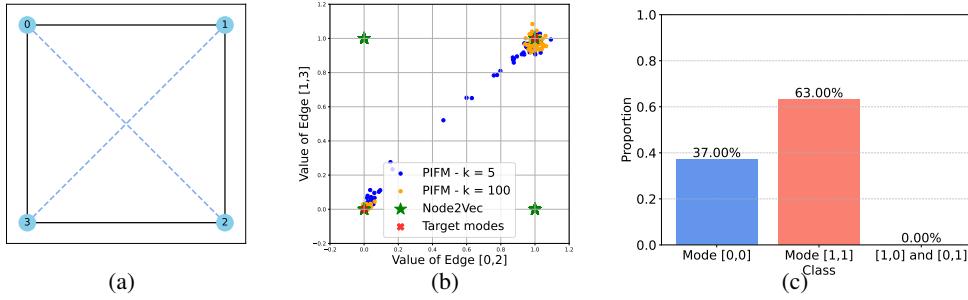
**324 Algorithm 1** Training and Sampling
 

---

**325 Training**

- 326 1: Sample  $\mathbf{A}_1 \sim p(\mathbf{A})$ , a mask  $\xi$ , and time  $t \sim U[0, 1]$ .
- 327 2: Train MMSE estimator:  $f_{\text{prior}}(\mathbf{A}^{\mathcal{O}})$
- 328 3: Compute  $\mathbf{A}_0 \triangleq \xi \odot \mathbf{A}_1 + (1 - \xi) \odot (f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}}) + \epsilon_s)$ ,  $\epsilon_s \sim \mathcal{N}(0, \sigma_s^2)$
- 329 4: Compute  $\mathbf{A}_t \triangleq (1 - t)\mathbf{A}_0 + t\mathbf{A}_1$ .
- 330 5: Train flow model:  $\theta^* = \operatorname{argmin}_{\theta} \mathbb{E}_{\mathbf{A}_1, \mathbf{A}_0, \xi, t} \|v_{\theta}(\mathbf{A}_t, t) - (\mathbf{A}_1 - \mathbf{A}_0)\|_F^2$
- 331 **Sampling (Reconstruction)**
- 332 6: Initialize  $\hat{\mathbf{A}} \leftarrow \xi \odot \mathbf{A}_1^{\mathcal{O}} + (1 - \xi) \odot f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}}) + (1 - \xi) \odot \epsilon_s$ ,  $\epsilon_s \sim \mathcal{N}(0, \sigma_{\text{samp}}^2)$ .
- 333 7: **for**  $i \leftarrow 0, \dots, K - 1$  **do**
- 334 8:  $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{A}} + \frac{1}{K} v_{\theta^*} \left( \hat{\mathbf{A}}, \frac{i}{K} \right)$
- 335 9: **end for**
- 336 10: Return  $\hat{\mathbf{A}}$

---



349 Figure 2: Toy experiment showcasing the advantage of PIFM (in this case, for link prediction). a) Graph  $\mathcal{G}$  with  
 350 four nodes, where the hidden edges are  $e_{02}$  and  $e_{13}$ . b) Generated samples by using node2vec and PIFM (our  
 351 proposed method): clearly, our method learns a probabilistic coupling, rendering a model that generates only the  
 352 two valid modes. c) Proportions of samples generated with PIFM from each mode; remarkably, the method also  
 353 learns a good approximation of the probability of each mode.

354 0.5, so entries with predicted probability  $\geq 0.5$  are treated as edges and those  $< 0.5$  as non-edges. In  
 355 addition, we use maximum mean discrepancy (MMD) (O’Bray et al., 2022) to compute the distance  
 356 between the generated graphs and the ground truth, serving as a proxy for computing the perception  
 357 quality. More details on these metrics are deferred to Appendix E.4.

358 **Baselines.** We compare PIFM against several baselines, including diffusion-based. Recall that  
 359 PIFM is composed of a one-shot prediction used as prior followed by a flow model. Naturally, we  
 360 compare PIFM to the accuracy of the one-shot prediction (without the flow) and with a flow with a  
 361 random starting point:

- 363 • **SIGL Prior (Azizpour et al., 2025)/Node2Vec Prior (Grover & Leskovec, 2016)/GraphSAGE**  
 364 **Prior (Hamilton et al., 2017):** one-shot predictions using the structural prior directly.
- 365 • **Flow with Gaussian prior:** flow model initialized from uniform Gaussian  $\mathcal{N}(0.5, 1)$  noise on  
 366 masked entries.
- 367 • **DiGress + RePaint (Vignac et al., 2023):** unconditional DiGress combined with RePaint-style  
 368 resampling (Lugmayr et al., 2022).
- 369 • **GDSS + RePaint (Jo et al., 2022):** unconditional GDSS combined with RePaint-style resam-  
 370 pling (Lugmayr et al., 2022).

371 Algorithmic details of the baselines are provided in Appendix A. **Lastly, we consider an additional**  
 372 **experiment on a transductive case (CORA Yang et al. (2016)), where we compared with traditional**  
 373 **baselines Li et al. (2023).**

374 **5.2 LINK PREDICTION**

375 Tables 1 and 2 report results for 10% and 50% masking, respectively. Overall, PIFM improves the  
 376 AUC-ROC of all base priors (SIGL, node2vec, and GraphSAGE). E.g., compare node2vec with PIFM

378 Table 1: Graph reconstruction performance with **10% of edges masked (0.1 Drop)**. We report AUC, Average  
 379 Precision (AP↑), False Positive Rate (FPR↓), and False Negative Rate (FNR↓), all in percent (%). The best  
 380 result for each metric is in **blue** and the second best is **green**.

Method	Mask Rate: 10% (0.1 Drop)											
	ENZYMEs				PROTEINS				IMDB-B			
	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓
<i>Baselines</i>												
Node2Vec	24.62	59.60	<b>51.39</b>	37.96	33.24	64.40	48.56	35.37	65.00	56.36	50.27	41.68
SIGL	18.17	48.04	69.33	25.43	26.77	48.91	100.00	<b>0.00</b>	58.91	50.61	88.44	16.33
GraphSAGE	41.28	73.70	<b>13.49</b>	60.59	46.36	74.58	<b>11.00</b>	63.50	83.55	83.26	16.42	36.89
DiGress + RePaint	33.39	67.86	58.92	5.19	40.34	72.39	<b>47.82</b>	6.00	59.25	58.63	76.44	7.68
GDSS + RePaint	18.35	47.04	74.31	32.19	26.96	51.39	63.07	32.09	57.89	46.11	69.75	36.17
Flow w/ Gaussian prior	40.09	72.44	71.03	5.87	<b>57.86</b>	80.83	65.09	<b>3.07</b>	<b>98.89</b>	<b>98.37</b>	2.26	<b>2.54</b>
<i>Ours</i>												
PIFM (Node2Vec)	<b>41.67</b>	<b>76.86</b>	72.09	<b>5.11</b>	<b>58.25</b>	<b>81.74</b>	59.37	6.34	97.60	97.28	<b>1.37</b>	3.77
PIFM (GraphSAGE)	<b>47.21</b>	<b>80.25</b>	72.85	<b>2.40</b>	54.79	<b>81.02</b>	55.73	5.40	<b>99.37</b>	<b>98.79</b>	<b>1.81</b>	<b>3.37</b>
PIFM (SIGL)	26.93	59.48	71.33	11.33	42.21	60.76	60.75	7.48	85.60	83.21	16.37	18.41

392 Table 2: Graph reconstruction performance with **50% of edges masked (0.5 Drop)** (see Table 1 for definitions).

Method	Mask Rate: 50% (0.5 Drop)											
	ENZYMEs				PROTEINS				IMDB-B			
	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓
<i>Baselines</i>												
Node2Vec	19.14	55.22	<b>46.37</b>	44.29	23.51	53.83	<b>51.37</b>	44.36	54.20	52.22	48.41	47.51
SIGL	16.88	49.30	72.01	27.85	22.90	52.55	100.00	<b>0.00</b>	50.05	45.41	87.68	18.80
GraphSAGE	22.79	57.77	<b>40.02</b>	52.16	27.71	53.99	<b>32.16</b>	66.86	75.74	75.54	<b>18.18</b>	44.86
DiGress + RePaint	17.34	55.22	77.95	11.62	23.65	55.45	71.46	17.65	56.47	58.89	73.00	<b>10.27</b>
GDSS + RePaint	16.43	49.65	69.45	30.46	22.33	51.42	66.44	32.23	53.39	51.20	69.35	29.22
Flow w/ Gaussian prior	17.43	51.84	98.49	<b>1.07</b>	26.40	55.55	93.21	<b>5.25</b>	78.72	79.76	41.56	14.62
<i>Ours</i>												
PIFM (Node2Vec)	<b>22.95</b>	<b>59.14</b>	90.71	3.53	27.57	<b>59.68</b>	87.05	8.98	<b>84.46</b>	<b>85.71</b>	32.95	15.03
PIFM (GraphSAGE)	<b>25.44</b>	<b>61.36</b>	95.62	<b>1.86</b>	<b>35.50</b>	<b>60.61</b>	85.05	10.23	<b>93.13</b>	<b>93.84</b>	<b>17.52</b>	<b>7.61</b>
PIFM (SIGL)	17.08	49.15	86.06	12.28	<b>28.38</b>	59.58	61.20	20.38	59.83	58.11	38.90	36.76

406 initialized with node2vec. The marked consistent gain can be attributed to the value added by the  
 407 flow model in capturing the distribution of the true graphs of interest. Moreover, the fact that PIFM  
 408 with some of the informative priors tends to outperform the flow with a Gaussian prior highlights the  
 409 value of the two-step procedure advocated here. Among the different priors used, PIFM(GraphSAGE)  
 410 tends to perform better, especially at a 50% drop rate and in the dense IMDB-B graphs.

411 For the experiments in this section, the reported PIFM results use  $K = 1$ , which yields the lowest  
 412 MSE and, accordingly, the highest AUC-ROC (consistent with the distortion–perception trade-off  
 413 discussed in Section 4.1). Notably, PIFM with  $K = 1$  outperforms the priors (see Appendix F.2 for  
 414 an ablation of parameters), even though the latter approximate the MMSE estimator, which should  
 415 be optimal in terms of MSE. While this configuration is optimal for distortion, perceptual quality  
 416 improves with more steps, as explained below in Section 5.3. Finally, the assumptions in Section 4.2  
 417 are quite strong and lead to an approximate MMSE that is not truly optimal, allowing PIFM with  
 418  $K = 1$  to outperform by capturing global information that the different priors miss.

### 419 5.3 BLIND GRAPH RECONSTRUCTION

420 We focus on two blind versions of link prediction, namely expansion and denoising. In the expansion  
 421 case, we only get to observe a subset of the edges (but no non-edges), and we need to determine which  
 422 other entries correspond to existing edges. Conversely, for denoising, we get to observe a subset  
 423 of the non-edges (but no actual edge), and we need to determine which other entries correspond  
 424 to non-edges. These cases are more challenging than link prediction since transductive priors like  
 425 node2vec cannot be trained on the masked graphs (since we do not have positive and negative edges).  
 426 We present here the results for expansion. The results for denoising can be found in Appendix F.1.

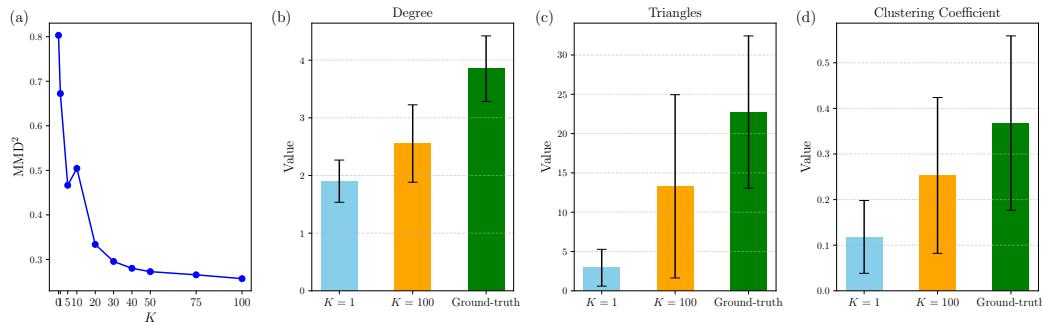
427 **Expansion.** The goal in expansion is to predict a set of hidden edges  $\mathcal{E}_M$  given  $\mathbf{A}^{\mathcal{O}}$ , such that the  
 428 edge set of the ground truth is  $\mathcal{E} = \mathcal{E}_M \cup \mathcal{E}_O$ . Therefore, defining  $\mathbf{A}_1 = \mathbf{A}$ , the initialization becomes  
 429  $\mathbf{A}_0 = \mathbf{A}_1^{\mathcal{O}} + (1 - \mathbf{A}_1^{\mathcal{O}}) \odot (f_{prior}(\mathbf{A}_1^{\mathcal{O}}) + \mathbf{e}_s)$ . The results for a drop rate of 50% are shown in Table 3.  
 430 Among all baselines, PIFM (GraphSAGE) attains the top AUC/AP on most of the metrics, surpassing  
 431 both the GraphSAGE prior and other diffusion baselines. Compared to a Gaussian start, the informed

432 Table 3: Performance for the **expansion** task with **50% of edges masked (0.5 Drop)** (see Table 1 for definitions).  
433

434 435 436 437 438 439 440 441	Mask Rate: 50% (0.5 Drop)											
	442 443 444 445 446 447 448 449 450 451 452 453 454				442 443 444 445 446 447 448 449 450 451 452 453 454				442 443 444 445 446 447 448 449 450 451 452 453 454			
	Method	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓
<i>Baselines</i>												
GraphSAGE	<b>13.95</b>	<b>57.54</b>	<b>40.29</b>	52.74	<b>18.91</b>	53.91	<b>31.62</b>	67.18	<b>67.18</b>	<b>74.92</b>	<b>19.04</b>	45.20
DiGress + RePaint	2.41	54.25	84.03	<b>7.54</b>	4.52	<b>56.87</b>	81.10	<b>13.00</b>	22.93	56.37	81.05	<b>6.69</b>
GDSS + RePaint	9.21	49.63	<b>69.45</b>	30.80	14.66	51.03	<b>66.44</b>	32.06	39.43	50.68	69.35	30.00
Flow w/ Gaussian prior	9.45	50.40	90.64	9.35	14.71	50.31	82.32	17.28	49.46	62.28	71.41	13.64
<i>Ours</i>												
PIFM (GraphSAGE)	<b>13.17</b>	<b>60.09</b>	100.00	<b>0.00</b>	<b>21.70</b>	<b>62.34</b>	94.75	<b>4.54</b>	<b>83.49</b>	<b>87.28</b>	<b>29.74</b>	<b>11.27</b>

443 prior is crucial to improve AUC and AP, indicating effective global coupling beyond local scores.  
444 Overall, PIFM serves as a better reconstructor in this challenging case, with  $K$  providing a tunable  
445 perception–distortion trade-off (cf. Appendix F.3).

447 **Distortion-perception trade-off.** While a single-step reconstruction ( $K = 1$ ) yields the lowest  
448 distortion (AUC-ROC), we assess if more steps improve perceptual quality. We measure the  $MMD^2$   
449 score between the generated and ground-truth graph distributions on the ENZYMES dataset as a  
450 function of the number of steps,  $K$ . As shown in Fig. 3(a), the  $MMD^2$  score decreases as  $K$  increases,  
451 signifying a closer match to the true data distribution and thus higher realism. We further validate  
452 this by comparing graph statistics (degree, triangles, clustering coefficients), which also show that a  
453 larger  $K$  more closely matches the ground-truth. Additional results and details are in Appendices F.3  
454 and F.4.



466 Figure 3: Increasing the number of reconstruction steps ( $K$ ) improves perceptual quality. (a) The  $MMD^2$ ,  
467 measuring the distance to the true data distribution, decreases as  $K$  increases.  
468 (b-d) This result is corroborated by key graph statistics, where the average degree, number of triangles,  
469 and clustering coefficient for graphs generated with  $K = 100$  more closely match the ground-truth distribution  
470 compared to those generated with  $K = 1$ . Error bars indicate the standard deviation over 300 samples (10 samples for each of the 30 test graphs).

## 471 6 CONCLUSIONS

473 In this paper, we introduced Prior-Informed Flow Matching (PIFM), a method for graph reconstruction  
474 that learns global structural information by integrating local edge predictors within a flow-based  
475 generative model. PIFM formulates graph topology inference as a distortion-perception problem,  
476 learning an optimal transport map from a local estimator to the ground-truth graph distribution.  
477 We evaluate PIFM using two types of local estimators, inductive (graphons and graphSAGE) and  
478 transductive (node2vec), which induce different reconstruction behaviors. Experiments on multiple  
479 benchmark datasets show that PIFM consistently outperforms both classical embedding methods and  
480 recent flow-based baselines, demonstrating the significant value of learning global edge correlations.

481 Our method has limitations, primarily inheriting the scalability challenges of diffusion models in  
482 graphs. Future work could explore sub-graph-based alternatives to improve efficiency (Trivedi et al.,  
483 2024). Additionally, our current formulation is limited to homogeneous graphs; extending PIFM to  
484 heterogeneous graphs by defining the process in the probability simplex (Eijkelenboom et al., 2024) or  
485 using discrete flow models Qin et al. (2025) is another promising direction for future research.

486 REPRODUCIBILITY STATEMENT  
487

488 The experimental setups and results are detailed in Section 5 of the main paper. Further specifics,  
489 including comprehensive dataset descriptions, additional experimental details, and ablation studies,  
490 are provided in Appendix E. Furthermore, and to facilitate full reproducibility, we include a complete  
491 codebase as supplementary material. This supplementary package contains clearly organized configu-  
492 ration files (e.g., YAML files) that detail all hyperparameters used across our experiments, enabling  
493 straightforward replication of our reported findings.

494  
495 REFERENCES  
496

497 Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Soc. Networks*, 25(3):211–230,  
498 2003. doi: 10.1016/S0378-8733(03)00009-1.

499 Edo M Airoldi, Thiago B Costa, and Stanley H Chan. Stochastic blockmodel approximation of a  
500 graphon: Theory and consistent estimation. *Advances in Neural Inf. Process. Syst. (NIPS)*, 26,  
501 2013.

502 Michael S Albergo, Nicholas M Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying  
503 framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

504 Michael Samuel Albergo, Mark Goldstein, Nicholas Matthew Boffi, Rajesh Ranganath, and Eric  
505 Vanden-Eijnden. Stochastic interpolants with data-dependent couplings. In *Intl. Conf. on Machine  
506 Learning (ICML)*, pp. 921–937. PMLR, 2024.

507 Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured  
508 denoising diffusion models in discrete state-spaces. *Advances in Neural Inf. Process. Syst. (NIPS)*,  
509 34:17981–17993, 2021.

510 Marco Avella-Medina, Francesca Parise, Michael T Schaub, and Santiago Segarra. Centrality  
511 measures for graphons: Accounting for uncertainty in networks. *IEEE Transactions on Network  
512 Science and Engineering*, 7(1):520–537, 2018.

513 Ali Azizpour, Nicolas Zilberstein, and Santiago Segarra. Scalable implicit graphon learning. In *Int.  
514 Conf. on Artif. Intell. and Stat.*, 2025.

515 Yochai Blau and Tomer Michaeli. The perception-distortion tradeoff. In *Proceedings of the IEEE/CVF  
516 Int. Conf. Comput. Vis. Pattern Recogn. (CVPR)*, pp. 6228–6237, 2018.

517 Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative  
518 flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design.  
519 In *Intl. Conf. on Machine Learning (ICML)*, pp. 5453–5512. PMLR, 2024.

520 Stanley Chan and Edoardo Airoldi. A consistent histogram estimator for exchangeable graph models.  
521 In *Intl. Conf. on Machine Learning (ICML)*, pp. 208–216. PMLR, 2014.

522 Katherine Crowson, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z  
523 Kaplan, and Enrico Shippole. Scalable high-resolution pixel-space image synthesis with hourglass  
524 diffusion transformers. In *Intl. Conf. on Machine Learning (ICML)*, 2024.

525 Mauricio Delbracio and Peyman Milanfar. Inversion by direct iteration: An alternative to denoising  
526 diffusion for image restoration. *Trans. Mach. Learn. Res.*, 2024.

527 Xiaowen Dong, Dorina Thanou, Pascal Frossard, and Pierre Vandergheynst. Learning laplacian  
528 matrix in smooth graph signal representations. *IEEE Trans. Signal Process.*, 64(23):6160–6173,  
529 2016.

530 Floor Eijkelboom, Grigory Bartosh, Christian Andersson Naesseth, Max Welling, and Jan-Willem  
531 van de Meent. Variational flow matching for graph generation. *Advances in Neural Inf. Process.  
532 Syst. (NIPS)*, 37:11735–11764, 2024.

533 Dror Freirich, Tomer Michaeli, and Ron Meir. A theory of the distortion-perception tradeoff in  
534 wasserstein space. In *Advances in Neural Inf. Process. Syst. (NIPS)*, 2021.

540 Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM*  
 541 *SIGKDD Int. Conf. on Knowledge Discovery and Data Mining.*, pp. 855–864, 2016.  
 542

543 Aditya Grover, Aaron Zweig, and Stefano Ermon. Graphite: Iterative generative modeling of graphs.  
 544 In *Intl. Conf. on Machine Learning (ICML)*, pp. 2434–2444. PMLR, 2019.

545 William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs.  
 546 In *Advances in Neural Inf. Process. Syst. (NIPS)*, NIPS’17, pp. 1025–1035, 2017.  
 547

548 Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via  
 549 the system of stochastic differential equations. In *Intl. Conf. on Machine Learning (ICML)*, pp.  
 550 10362–10383. PMLR, 2022.

551 Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-  
 552 based generative models. *Advances in Neural Inf. Process. Syst. (NIPS)*, 35:26565–26577, 2022.  
 553

554 Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei  
 555 Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking,  
 556 2023. URL <https://arxiv.org/abs/2306.10453>.

557 Stratis Limnios, Praveen Selvaraj, Mihai Cucuringu, Carsten Maple, Gesine Reinert, and Andrew  
 558 Elliott. Sagess: Sampling graph denoising diffusion model for scalable graph generation. *arXiv*  
 559 *preprint*, arXiv:2306.16827, 2023. doi: 10.48550/arXiv.2306.16827. URL <https://arxiv.org/abs/2306.16827>.  
 560

561 Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching  
 562 for generative modeling. In *Intl. Conf. Learn. Repr. (ICLR)*, 2023.

563 Xingchao Liu, Chengyue Gong, et al. Flow straight and fast: Learning to generate and transfer data  
 564 with rectified flow. In *Intl. Conf. Learn. Repr. (ICLR)*, 2023.

565 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.  
 566

567 Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van  
 568 Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the*  
 569 *IEEE/CVF Int. Conf. Comput. Vis. Pattern Recogn. (CVPR)*, 2022.

570 Mark E. J. Newman. Clustering and preferential attachment in growing networks. *Phys. Rev. E*, 64  
 571 (2):025102, 2001. doi: 10.1103/PhysRevE.64.025102.

572 Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permu-  
 573 tation invariant graph generation via score-based generative modeling. In *Int. Conf. on Artif. Intell.*  
 574 *and Stat.*, pp. 4474–4484. PMLR, 2020.

575 Leslie O’Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph  
 576 generative models: Problems, pitfalls, and practical solutions. *Intl. Conf. Learn. Repr. (ICLR)*,  
 577 2022.

578 Guy Ohayon, Tomer Michaeli, and Michael Elad. Posterior-mean rectified flow: Towards minimum  
 579 MSE photo-realistic image restoration. In *Intl. Conf. Learn. Repr. (ICLR)*, 2025.

580 Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations.  
 581 In *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining.*, pp. 701–710, 2014.

582 Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. Defog: Discrete flow matching  
 583 for graph generation. *Intl. Conf. on Machine Learning (ICML)*, 2025.

584 Santiago Segarra, Antonio G. Marques, Gonzalo Mateos, and Alejandro Ribeiro. Network topology  
 585 inference from spectral templates. *IEEE Transactions on Signal and Information Processing over*  
 586 *Networks*, 3(3):467–483, 2017.

587 Kartik Sharma, Srijan Kumar, and Rakshit Trivedi. Diffuse, sample, project: plug-and-play control-  
 588 able graph generation. In *Intl. Conf. on Machine Learning (ICML)*, 2024.

589

594 Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit  
 595 neural representations with periodic activation functions. *Advances in Neural Inf. Process. Syst.*  
 596 (*NIPS*), 33:7462–7473, 2020.

597

598 Victor M Tenorio, Nicolas Zilberstein, Santiago Segarra, and Antonio G Marques. Graph guided  
 599 diffusion: Unified guidance for conditional graph generation. *arXiv preprint arXiv:2505.19685*,  
 600 2025.

601 Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-  
 602 Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models  
 603 with minibatch optimal transport. *Trans. Mach. Learn. Res.*, 2024.

604

605 Puja Trivedi, Ryan A Rossi, David Arbour, Tong Yu, Franck Dernoncourt, Sungchul Kim, Nedim  
 606 Lipka, Namyong Park, Nesreen K Ahmed, and Danai Koutra. Editing partially observable networks  
 607 via graph diffusion models. In *Intl. Conf. on Machine Learning (ICML)*, 2024.

608 Clément Vignac, Igor Krawczuk, Antoine Sraordin, Bohan Wang, Volkan Cevher, and Pascal Frossard.  
 609 Digress: Discrete denoising diffusion for graph generation. In *Intl. Conf. Learn. Repr. (ICLR)*,  
 610 2023.

611 Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link  
 612 prediction. In *Intl. Conf. Learn. Repr. (ICLR)*, 2024.

613

614 Xinyue Xia, Gal Mishne, and Yusu Wang. Implicit graphon neural representation. *Int. Conf. on Artif.*  
 615 *Intell. and Stat.*, pp. 10619–10634, 2023.

616 Hongteng Xu, Dixin Luo, Lawrence Carin, and Hongyuan Zha. Learning graphons via structured  
 617 gromov-wasserstein barycenters. In *Proc. AAAI Conf. Artif. Intell.*, pp. 10505–10513, 2021.

618

619 Nianzu Yang, Huaijin Wu, Kaipeng Zeng, Yang Li, Siyuan Bao, and Junchi Yan. Molecule generation  
 620 for drug design: a graph learning perspective. *Fundam. Res.*, 2024.

621

622 Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with  
 623 graph embeddings. In *Intl. Conf. on Machine Learning (ICML)*, pp. 40–48. PMLR, 2016.

624

625 Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural*  
*626 Inf. Process. Syst. (NIPS)*, 31, 2018.

627

628 Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information, 2009.  
 URL <https://arxiv.org/abs/0901.0553>. Journal version: *Eur. Phys. J. B* 71:623–630  
 (2009), doi:10.1140/epjb/e2009-00335-8.

629

630 Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford  
 631 networks: a general graph neural network framework for link prediction. In *Advances in Neural Inf.*  
*632 Process. Syst. (NIPS)*, Red Hook, NY, USA, 2021. Curran Associates Inc. ISBN 9781713845393.

633

634

635

636

637

638

639

640

641

642

643

644

645

646

647

648  
649

## A ALGORITHM

650

651 In this section, we describe the algorithms that we use as baselines. Each method serves distinct  
 652 purposes: SIGL/node2vec/GraphSAGE  $A_0$  tests whether the flow model provides meaningful im-  
 653 provement beyond the one-shot estimates given by the priors, uniform + flow evaluates whether the  
 654 SIGL/node2vec/GraphSAGE predicted graphons are good structural priors for effective denoising,  
 655 and DiGress + RePaint compares our model to standard modified unconditionally generation models.

656

657  
658

## A.1 UNIFORM + FLOW BASELINE

659  
660  
661

This baseline ablates the structural prior by initializing the flow from a state where unknown entries  
 are filled with uniform noise. The model then learns to denoise from this less-informed starting point.

662

663 **Algorithm 2** Uniform + flow Training and Sampling

664

## Training

665  
666

- 1: Sample  $\mathbf{A}_1$ , a mask  $\xi$ , and time  $t \sim U[0, 1]$ .
- 2: Define initial state with Gaussian noise added to the masked region:

667  
668

$$\mathbf{A}_0 \triangleq \xi \odot \mathbf{A}_1 + (1 - \xi) \odot \mathcal{U}(0, 1)^{N \times N} + (1 - \xi) \odot \boldsymbol{\epsilon}_{\text{train}}, \quad \boldsymbol{\epsilon}_{\text{train}} \sim \mathcal{N}(0, \sigma_{\text{train}}^2).$$

669  
670

- 3: Define interpolant  $\mathbf{A}_t \triangleq (1 - t)\mathbf{A}_0 + t\mathbf{A}_1$ .
- 4: Solve  $\theta^* = \operatorname{argmin}_{\theta} \mathbb{E}_{\mathbf{A}_1, \xi, t} \|v_{\theta}(\mathbf{A}_t, t) - (\mathbf{A}_1 - \mathbf{A}_0)\|_F^2$ .

671  
672

**Sampling (Reconstruction)**  
 5: Given observed graph  $\mathbf{A}_1^{\mathcal{O}}$ , define the initial state with masked noise:

673  
674

$$\hat{\mathbf{A}} \leftarrow \xi \odot \mathbf{A}_1^{\mathcal{O}} + (1 - \xi) \odot \mathcal{U}(0, 1)^{N \times N} + (1 - \xi) \odot \boldsymbol{\epsilon}_{\text{samp}}, \quad \boldsymbol{\epsilon}_{\text{samp}} \sim \mathcal{N}(0, \sigma_{\text{samp}}^2).$$

675  
676  
677  
678

- 6: for  $i \leftarrow 0, \dots, K - 1$  do
- 7:    $\hat{\mathbf{A}} \leftarrow \hat{\mathbf{A}} + \frac{1}{K} v_{\theta^*} \left( \hat{\mathbf{A}}, \frac{i}{K} \right)$
- 8: Return  $\hat{\mathbf{A}}$

679  
680

## A.2 DiGRESS + REPAINT BASELINE

681  
682

**Training (Unconditional)** The model  $p_{\theta}$  is trained unconditionally on complete graphs  $\mathbf{A}_1 \sim p_{\text{data}}$  to reverse a discrete forward noising process  $q$ . The forward process is a fixed Markov chain  $q(\mathbf{A}_t | \mathbf{A}_{t-1})$  that corrupts the graph over  $T$  steps. The training objective is to learn the denoising distribution  $p_{\theta}(\mathbf{A}_1 | \mathbf{A}_t)$ , modeled as a categorical prediction task for each node and edge.

683

684  
685686 **Algorithm 3** DiGress Unconditional Training687  
688  
689

**Forward Process:** Sample a noised graph at any timestep  $t$  directly via  $\mathbf{A}_t \sim q(\mathbf{A}_t | \mathbf{A}_1)$ .

**Denoising Objective:**

690  
691  
692  
693

- 1: Train a denoising network  $p_{\theta}(\cdot, t)$  to predict the original graph  $\mathbf{A}_1$  from  $\mathbf{A}_t$ .
- 2: Minimize the expected cross-entropy loss w.r.t. the ground truth:

694  
695  
696  
697

$$\theta^* = \operatorname{argmin}_{\theta} \mathbb{E}_{\mathbf{A}_1 \sim p_{\text{data}}, t \sim \mathcal{U}\{1..T\}} [\mathcal{L}_{\text{CE}}(\mathbf{A}_1, p_{\theta}(\mathbf{A}_t, t))]$$

698  
699  
700

**Sampling (Conditional Reconstruction via RePaint)** At inference, given an observed graph  $\mathbf{A}_1^{\mathcal{O}} \triangleq \xi \odot \mathbf{A}_1$ , the unconditionally trained model  $p_{\theta^*}$  generates the missing entries. This is achieved by iteratively re-imposing the known (unmasked) information during the reverse diffusion process (Lugmayr et al., 2022).

---

702 **Algorithm 4** DiGress + RePaint Sampling

---

703  
 704   **Input:** Observed graph  $\mathbf{A}_1^{\mathcal{O}}$ , mask  $\xi$ , trained model  $p_{\theta^*}$ , steps  $T$ .  
 705   **Output:** Reconstructed graph  $\hat{\mathbf{A}}_1$ .  
 706   1: Initialize  $\hat{\mathbf{A}}_T \sim p_{\text{prior}}(\cdot)$ , where  $p_{\text{prior}}$  is a random graph distribution.  
 707   2: **for**  $t = T, T-1, \dots, 1$  **do**  
 708     3:    *// Predict clean graph from current state*  
 709     4:     $\hat{\mathbf{A}}_1 = p_{\theta^*}(\hat{\mathbf{A}}_t, t)$ .  
 710     5:  
 711     6:    *// Impose known data by noising it to the current step*  
 712     7:     $\mathbf{A}_t^{\text{known}} \sim q(\mathbf{A}_t | \mathbf{A}_1^{\mathcal{O}})$ .  
 713     8:  
 714     9:    *// Sample the unknown region by noising the prediction to the next step*  
 715     10:    $\mathbf{A}_{t-1}^{\text{unknown}} \sim q(\mathbf{A}_{t-1} | \hat{\mathbf{A}}_1)$ .  
 716     11:  
 717     12:   *// Combine known and unknown parts for the next state*  
 718     13:     $\hat{\mathbf{A}}_{t-1} = \xi \odot \mathbf{A}_t^{\text{known}} + (1 - \xi) \odot \mathbf{A}_{t-1}^{\text{unknown}}$ .  
 719   14: **end for**  
 720   15: **return**  $p_{\theta^*}(\hat{\mathbf{A}}_1, 1)$ 

---

721  
 722 **A.3 NODE2VEC PRIOR (PER-GRAPH CLASSIFIER)**  
 723

724 This baseline learns a *per-graph* edge-probability model from the observed subgraph. We (i) fit  
 725 node2vec embeddings on the observed topology and (ii) train a logistic classifier on Hadamard edge  
 726 features to produce probabilities on the masked pairs.  
 727

---

728 **Algorithm 5** Node2Vec Prior: Training and Inference

---

729   **Inputs:** Full adjacency  $\mathbf{A}_1$ , mask  $\xi$  ( $\xi_{ij} = 1$  if observed), Node2Vec hyperparams (dim  $d$ , walk length  $L$ ,  
 730   walks/node  $R$ , window  $w$ ,  $p$ ,  $q$ ), negatives/positive ratio  $k$ .  
 731   **Outputs:** Probabilities  $\hat{P}$  on masked entries, i.e.,  $f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}})$ .  
 732   **Training (per graph)**  
 733   1: Construct observed graph  $\mathbf{A}_1^{\mathcal{O}} \leftarrow \xi \odot \mathbf{A}_1$ .  
 734   2: Train Node2Vec on  $\mathbf{A}_1^{\mathcal{O}}$  to obtain node embeddings  $\{\mathbf{z}_i\}_{i=1}^N \in \mathbb{R}^d$ .  
 735   3: Build labeled edge set on *observed* pairs (upper triangle  $i < j$ ):  
 736      $\mathcal{P}^+ = \{(i, j) : \xi_{ij} = 1, A_{ij} = 1\}, \quad \mathcal{P}^- \sim k\text{-to-1 balanced samples from } \{(i, j) : \xi_{ij} = 1, A_{ij} = 0\}$ .  
 737   4: Features:  $\mathbf{x}_{ij} \leftarrow \mathbf{z}_i \odot \mathbf{z}_j$  (Hadamard product); Labels:  $y_{ij} \in \{0, 1\}$ .  
 738   5: Fit a logistic classifier  $g_{\phi}(\mathbf{x}) = \sigma(\mathbf{w}^{\top} \mathbf{x} + b)$  (L2-regularized; class-balanced).  
 739   **Inference (per graph)**  
 740   6: For each *masked* pair  $(i, j)$  with  $\xi_{ij} = 0$ , compute  $\mathbf{x}_{ij} \leftarrow \mathbf{z}_i \odot \mathbf{z}_j$ .  
 741   7: Predict  $\hat{P}_{ij} \leftarrow g_{\phi}(\mathbf{x}_{ij})$  and set  $\hat{P}_{ji} \leftarrow \hat{P}_{ij}$ .  
 742   8: Return  $\hat{P}$  as  $f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}})$  (used in Eq. (8)).

---

743  
 744 *Notes.* (i) We train embeddings *only* on  $\mathbf{A}_1^{\mathcal{O}}$  to avoid leakage. (ii) The Hadamard feature works well  
 745 and is symmetric; concatenation can be used but breaks symmetry unless sorted. (iii) Thresholding at  
 746 0.5 yields hard reconstructions; we use scores  $\hat{P}$  directly in PIFM.  
 747

748  
 749 **B BACKGROUND**  
 750751 **B.1 GRAPHONS AND GRAPHON ESTIMATION**  
 752

753 As described in Section 4.2, a graphon is defined as a bounded, symmetric, and measurable function  
 754  $\mathcal{W} : [0, 1]^2 \rightarrow [0, 1]$  (Lovász, 2012). By construction, a graphon acts as a *generative model for*  
 755 *random graphs*, allowing the sampling of graphs that exhibit similar structural properties. To generate  
 an undirected graph  $\mathcal{G}$  with  $N$  nodes from a given graphon  $\mathcal{W}$ , the process consists of two main steps:

(1) assigning each node a latent variable drawn uniformly at random from the interval  $[0, 1]$ , and (2) connecting each pair of nodes with a probability given by evaluating  $\mathcal{W}$  at their respective latent variable values.

The generative process in (7) can also be viewed in reverse: given a collection of graphs (represented by their adjacency matrix)  $\mathcal{D} = \{\mathbf{A}_t\}_{t=1}^M$  that are sampled from an *unknown* graphon  $\mathcal{W}$ , estimate  $\mathcal{W}$ . Several methods have been proposed for this task (Chan & Airoldi, 2014; Airoldi et al., 2013; Xu et al., 2021; Xia et al., 2023; Azizpour et al., 2025). We focus on SIGL (Azizpour et al., 2025), a resolution-free method that, in addition to estimating the graphon, also *infers the latent variables*  $\eta$ , making it particularly useful for model-driven augmentation in GCL. This method parameterizes the graphon using an implicit neural representation (INR) (Sitzmann et al., 2020), a neural architecture defined as  $f_\phi(x, y) : [0, 1]^2 \rightarrow [0, 1]$  where the inputs are coordinates from  $[0, 1]^2$  and the output approximates the graphon value  $\mathcal{W}$  at a particular position. In a nutshell, SIGL works in three steps: (1) a sorting step using a GNN  $g_{\phi'}(\mathbf{A})$  that estimates the latent node positions or representations  $\eta$ ; (2) a histogram approximation of the sorted adjacency matrices; and (3) learning the parameters  $\phi$  by minimizing the mean squared error between  $f_\phi(x, y)$  and the histograms (obtained in step 2). More details of SIGL are provided in Appendix B.1.

## B.2 NODE2VEC

*node2vec* (Grover & Leskovec, 2016) is a scalable model for learning continuous node representations in graphs. This methods is *transductive*, meaning that it generates an embedding per graph. It extends the Skip-gram model from natural language processing to networks by sampling sequences of nodes through biased random walks. Node2vec introduces two hyperparameters  $(p, q)$  that interpolate between breadth-first and depth-first exploration. This flexibility allows embeddings to capture both *homophily* (nodes in the same community) and *structural equivalence* (nodes with similar roles, e.g., hubs), which frequently coexist in real-world graphs.

The embeddings are learned via stochastic gradient descent with negative sampling to maximize the likelihood of preserving sampled neighborhoods. Once learned, node embeddings can be combined through simple binary operators (e.g., Hadamard product) to form edge features, enabling applications such as link prediction. Empirically, node2vec has been shown to outperform prior unsupervised embedding methods across tasks like classification and link recovery, while remaining computationally efficient and scalable to large graphs (Grover & Leskovec, 2016).

## B.3 GRAPHSAGE

GraphSAGE (Hamilton et al., 2017) is an *inductive* technique for link prediction based on graph neural networks (GNN) framework designed to generate embeddings for nodes in large, evolving graphs. It consists of two-steps: for a target node, it first samples a fixed-size neighborhood of adjacent nodes, and then it aggregates feature information from these sampled neighbors. By learning aggregation functions (such as a mean, pool, or LSTM aggregator) rather than embeddings for every single node, GraphSAGE can efficiently generate predictions for nodes that were not part of the training set, making it highly scalable and effective for real-world applications like social networks and recommendation systems.

## B.4 GRAPH DIFFUSION MODELS

Diffusion models are generative frameworks composed by two processes: a **forward process** that systematically adds noise to data until it becomes pure noise, and a **reverse process** that learns to reverse this, generating new data by starting from noise and progressively denoising it. While these models exist for both discrete (Vignac et al., 2023) and continuous domains (Jo et al., 2022), we describe the continuous case which is the most related to our method. Here, a graph  $\mathbf{G}_0$  is defined by its node features  $\mathbf{X}_0 \in \mathbb{R}^{N \times F}$  and its weighted adjacency matrix  $\mathbf{A}_0 \in \mathbb{R}^{N \times N}$ . Following the GDSS framework, the forward process is described by a stochastic differential equation (SDE) that gradually perturbs the graph data over a time interval  $t \in [0, T]$ :

$$d\mathbf{G}_t = -\frac{1}{2}\beta(t)\mathbf{G}_t dt + \sqrt{\beta(t)} d\mathbf{W}_t$$

In this equation,  $\mathbf{W}_t$  represents standard Brownian motion (i.e., noise), and  $\beta(t)$  is a noise schedule that typically increases over time. This process is designed so that by the final time  $T$ , the original data distribution  $\mathbf{G}_T$  is indistinguishable from a standard Gaussian.

The generative reverse process is defined by another SDE that traces the path from noise back to data. This process relies on the **score function**,  $\nabla_{\mathbf{G}_t} \log p(\mathbf{G}_t)$ , which is the gradient of the log-density of the noisy data at time  $t$ . Since the true score function is unknown, it must be approximated. This is done using a neural network, or **score network**, which is trained to predict the score. For graphs, separate networks are often used for the adjacency matrix and node features:  $\epsilon_{\theta_A}(\mathbf{A}_t, t)$  and  $\epsilon_{\theta_X}(\mathbf{X}_t, t)$ . These networks are trained by minimizing the denoising score-matching loss.

Once trained, these score networks can be plugged into the reverse SDE. New graphs are then generated by solving this SDE numerically using standard samplers like DDPM or DDIM.

## C RELATED WORKS

### C.1 LINK PREDICTION

Link prediction aims to determine if an unobserved edge should exist between two nodes within a partially observed graph (Newman, 2001; Adamic & Adar, 2003; Zhou et al., 2009). Classical approaches rely on topology-only heuristics.

More recently, unsupervised node embedding methods have become an effective strategy for link prediction. These methods learn a low-dimensional vector for each node that represents neighborhood similarity and community structure, often using random walks and an objective similar to Skip-gram. Consequently, nodes that are close in the embedding space are more likely to be linked. DeepWalk was a pioneering method that modeled short random walks to learn generalizable representations for tasks like predicting missing links (Perozzi et al., 2014). Node2vec builds on DeepWalk by employing biased, second-order random walks to balance breadth-first and depth-first searches and by converting node embeddings into edge features. In node2vec, embeddings for nodes  $f(u)$  and  $f(v)$  are combined with binary operators to create an edge representation  $g(u, v)$ , which a classifier then uses to determine if the edge  $(u, v)$  exists (Grover & Leskovec, 2016).

Graph neural networks (GNNs) are also widely used for edge reconstruction. A typical encoder-decoder framework uses message passing to learn node embeddings and a simple decoder to generate link scores. Inductive frameworks like GraphSAGE learn functions to sample and aggregate features from a node’s neighborhood, allowing the model to generalize to new nodes or graphs (Hamilton et al., 2017). A different approach focuses on modeling the pair representation directly. For example, Neural Bellman-Ford Networks (NBFNet) frame link prediction as a path-aggregation problem. The score for a pair of nodes is calculated as the sum of all path representations between them, with each path being a product of its edge representations. This formulation is solved using a generalized Bellman-Ford iteration, where NBFNet parameterizes the operators with neural functions, creating an interpretable and inductive framework (Zhu et al., 2021).

### C.2 DIFFUSION-BASED INVERSE PROBLEMS SOLVER FOR GRAPHS

We now expand on diffusion-based solvers for graph inverse problems. Given a condition  $\mathcal{C}$  and a reward function  $r(\mathbf{G}_0)$  that quantifies how close the sample  $\mathbf{G}_0$  is to meeting  $\mathcal{C}$ , the objective is to generate graphs  $\mathbf{G}_0$  that maximize the reward function. From a Bayesian perspective, this problem boils down to sampling from the posterior  $p(\mathbf{G}_0|\mathcal{C}) \propto p(\mathcal{C}|\mathbf{G}_0)p(\mathbf{G}_0)$  where  $p(\mathcal{C}|\mathbf{G}_0) \propto \exp(r(\mathbf{G}_0))$  is a likelihood term and  $p(\mathbf{G}_0)$  is a prior given by the pre-trained diffusion model. We now describe previous works for both differentiable and non-differentiable reward functions.

**Guidance with Differentiable Reward Functions.** Several approaches have been developed to guide generative models when the objective can be expressed as a **differentiable reward function**, particularly for inverse problems in imaging. These methods typically leverage the differentiability of the reward – often a likelihood tied to a noisy measurement – to calculate a *conditional score* using Bayes’ rule:

$$\nabla_{\mathbf{G}_t} \log p(\mathbf{G}_t|\mathcal{C}) = \nabla_{\mathbf{G}_t} \log p(\mathcal{C}|\mathbf{G}_t) + \nabla_{\mathbf{G}_t} \log p(\mathbf{G}_t)$$

In this formulation, the diffusion model naturally serves as the prior ( $p(\mathbf{G}_t)$ ), while the likelihood term ( $p(\mathcal{C}|\mathbf{G}_t)$ ) provides the guidance. However, directly computing the score of the likelihood term is intractable because it requires integrating over all possible clean data:  $p(\mathcal{C}|\mathbf{G}_t) = \int p(\mathcal{C}|\mathbf{G}_0)p(\mathbf{G}_0|\mathbf{G}_t)d\mathbf{G}_0$ .

To overcome this, a common technique is to approximate the posterior distribution  $p(\mathbf{G}_0|\mathbf{G}_t)$  with a Gaussian centered at the MMSE denoiser. This denoised estimate can be calculated efficiently using **Tweedie's formula**:

$$\mathbb{E}[\mathbf{G}_0|\mathbf{G}_t] = \frac{1}{\alpha_t} (\mathbf{G}_t + \sigma_t^2 \nabla_{\mathbf{G}_t} \log p(\mathbf{G}_t, t))$$

While this framework is established for images, its application to graph-based inverse problems is less explored. This is primarily because most interesting properties and constraints in graphs are **not differentiable**. Some graph-specific methods, like DiGress (Vignac et al., 2023), implement guidance by training an auxiliary model, similar to classifier-free guidance, which introduces additional complexity.

**Guidance with Non-Differentiable Reward Functions.** For the more common scenario of non-differentiable constraints in graph generation, alternative strategies have emerged. The **PRODIGY** method, for instance, operates by repeatedly applying a two-step process at each denoising step: generation followed by projection.

First, it uses the unconditional diffusion model to produce a candidate sample  $\hat{\mathbf{G}}_{t-1}$ . Second, it projects this candidate onto the set of valid solutions using a projection operator:  $\Pi_{\mathcal{C}}(\hat{\mathbf{G}}_{t-1}) = \arg \min_{\mathbf{Z} \in \mathcal{C}} \|\mathbf{Z} - \hat{\mathbf{G}}_{t-1}\|_2^2$ . Since applying the full projection at every step can destabilize the generation process, PRODIGY uses a partial update to balance constraint satisfaction with the learned diffusion trajectory:

$$\mathbf{G}_{t-1} \leftarrow (1 - \gamma_t) \hat{\mathbf{G}}_{t-1} + \gamma_t \Pi_{\mathcal{C}}(\hat{\mathbf{G}}_{t-1})$$

This approach has two main limitations. First, it is only practical for simple constraints where the projection operator  $\Pi_{\mathcal{C}}(\cdot)$  has an efficient, closed-form solution. Second, it applies the projection directly to the noisy intermediate sample  $\mathbf{G}_t$ , whereas the constraint  $\mathcal{C}$  is defined on the clean data  $\mathbf{G}_0$ , creating a domain mismatch. Recently, in Tenorio et al. (2025), the authors leverage zeroth-order optimizaton to build a guidance term, improving over PRODIGY in challenging tasks.

### C.3 FLOW-BASED INVERSE SOLVERS

More recently, two flow-based generative models for graphs have been proposed. Catflow, introduced in Eijkelboom et al. (2024), formulates flow matching as a variational inference problem, allowing to build a model for categorical data. The key difference between Catflow and traditional flow matching is that in the former, the objective is to approximate the posterior probability path, which is a distribution over possible end points of a trajectory. Compared to discrete diffusion, this formulation defines a path in the probability simplex, building a continuous path. This formulation boils down to a cross-entropy loss. Another recent work is DeFoG, introduced in Qin et al. (2025). This method is inspired by discrete flow matching (Campbell et al., 2024), where a discrete probability path is used. Similarly, the loss is a cross-entropy.

## D PROOFS

### D.1 PROOF FOR THEOREM 1

*Proof.* Our goal is to show that for any permutation matrix  $\mathbf{P}$ , our estimated density satisfies  $\log p(\mathbf{P}^\top \mathbf{A}_1 \mathbf{P}) = \log p(\mathbf{A}_1)$ . First, we notice that  $\text{tr} \left( \frac{\partial v_\theta(\mathbf{A}_t, t)}{\partial \mathbf{A}_t} \right) = \langle v_\theta(\mathbf{A}_t, t), d\mathbf{A}_t \rangle_F$ . Let's define a permuted graph  $\mathbf{A}'_1 = \mathbf{P}^\top \mathbf{A}_1 \mathbf{P}$  and a similarly permuted mask  $\xi' = \mathbf{P}^\top \xi \mathbf{P}$ ; to simplify notation, we denote  $F(\cdot) = \log p(\cdot)$ .

918 First, we establish the equivariance of the initial state  $\mathbf{A}_0$ . Let  $\mathbf{A}'_0$  be the initial state constructed  
 919 from the permuted graph  $\mathbf{A}'_1$  and mask  $\xi'$ .

$$\begin{aligned} 920 \mathbf{A}'_0 &= \xi' \odot \mathbf{A}'_1 + (1 - \xi') \odot f_{\text{prior}}(\xi' \odot \mathbf{A}'_1) \\ 921 &= (\mathbf{P}^\top \xi \mathbf{P}) \odot (\mathbf{P}^\top \mathbf{A}_1 \mathbf{P}) + (1 - \mathbf{P}^\top \xi \mathbf{P}) \odot f_{\text{prior}}((\mathbf{P}^\top \xi \mathbf{P}) \odot (\mathbf{P}^\top \mathbf{A}_1 \mathbf{P})) \\ 922 &= \mathbf{P}^\top (\xi \odot \mathbf{A}_1) \mathbf{P} + \mathbf{P}^\top (1 - \xi) \mathbf{P} \odot f_{\text{prior}}(\mathbf{P}^\top (\xi \odot \mathbf{A}_1) \mathbf{P}) \quad (\text{since } \odot \text{ distributes over } \mathbf{P}) \\ 923 \end{aligned}$$

924 The key requirement for this proof is the permutation equivariance of the prior estimator,  $f_{\text{prior}}$ . This  
 925 condition is satisfied by both prior models used in our work. Our SIGL-based prior is permutation  
 926 equivariant by design, as it uses a GNN encoder to learn the graphon structure. Our node2vec-based  
 927 prior enforces permutation equivariance by first mapping nodes to a canonical ordering based on  
 928 the principal components of their embeddings, ensuring that any permutation of an input graph is  
 929 processed identically.

930 With the permutation equivariance of  $f_{\text{prior}}$  established, such that  $f_{\text{prior}}(\mathbf{P}^\top \mathbf{X} \mathbf{P}) = \mathbf{P}^\top f_{\text{prior}}(\mathbf{X}) \mathbf{P}$ ,  
 931 we can apply this property:

$$\begin{aligned} 932 \mathbf{A}'_0 &= \mathbf{P}^\top (\xi \odot \mathbf{A}_1) \mathbf{P} + \mathbf{P}^\top (1 - \xi) \mathbf{P} \odot (\mathbf{P}^\top f_{\text{prior}}(\xi \odot \mathbf{A}_1) \mathbf{P}) \\ 933 &= \mathbf{P}^\top (\xi \odot \mathbf{A}_1) \mathbf{P} + \mathbf{P}^\top ((1 - \xi) \odot f_{\text{prior}}(\xi \odot \mathbf{A}_1)) \mathbf{P} \\ 934 &= \mathbf{P}^\top (\xi \odot \mathbf{A}_1 + (1 - \xi) \odot f_{\text{prior}}(\xi \odot \mathbf{A}_1)) \mathbf{P} \\ 935 &= \mathbf{P}^\top \mathbf{A}_0 \mathbf{P} \\ 936 \\ 937 \end{aligned}$$

938 Thus, the initial state  $\mathbf{A}_0$  is permutation-equivariant.

939 Next, we examine the flow path  $\mathbf{A}'_t$  corresponding to the permuted graph  $\mathbf{A}'_1$ :

$$\begin{aligned} 942 \mathbf{A}'_t &= (1 - t) \mathbf{A}'_0 + t \mathbf{A}'_1 \\ 943 &= (1 - t) (\mathbf{P}^\top \mathbf{A}_0 \mathbf{P}) + t (\mathbf{P}^\top \mathbf{A}_1 \mathbf{P}) \\ 944 &= \mathbf{P}^\top ((1 - t) \mathbf{A}_0 + t \mathbf{A}_1) \mathbf{P} \\ 945 &= \mathbf{P}^\top \mathbf{A}_t \mathbf{P}. \\ 946 \\ 947 \end{aligned}$$

948 The path itself is equivariant. The differential element also transforms equivariantly:  $d\mathbf{A}'_t =$   
 949  $\mathbf{P}^\top d\mathbf{A}_t \mathbf{P}$ .

950 Now, we evaluate the scalar function  $F(\mathbf{A}'_1)$  by integrating along the permuted path  $\mathbf{A}'_t$ :

$$952 F(\mathbf{A}'_1) = - \int_0^1 \langle v_\theta(\mathbf{A}'_t, t), d\mathbf{A}'_t \rangle_F dt + C \\ 953$$

954 Substituting the equivariant forms for the path and its differential:

$$955 F(\mathbf{A}'_1) = - \int_0^1 \langle v_\theta(\mathbf{P}^\top \mathbf{A}_t \mathbf{P}, t), \mathbf{P}^\top d\mathbf{A}_t \mathbf{P} \rangle_F dt + C \\ 956 \\ 957$$

958 By the assumed permutation equivariance of the velocity field  $v_\theta$ , we have  $v_\theta(\mathbf{P}^\top \mathbf{A}_t \mathbf{P}, t) =$   
 959  $\mathbf{P}^\top v_\theta(\mathbf{A}_t, t) \mathbf{P}$ . Substituting this in:

$$960 F(\mathbf{A}'_1) = - \int_0^1 \langle \mathbf{P}^\top v_\theta(\mathbf{A}_t, t) \mathbf{P}, \mathbf{P}^\top d\mathbf{A}_t \mathbf{P} \rangle_F dt + C \\ 961 \\ 962$$

963 The Frobenius inner product  $\langle \mathbf{A}, \mathbf{B} \rangle_F = \text{tr}(\mathbf{A}^\top \mathbf{B})$  is invariant to unitary transformations. Specifically,  
 964 for any orthogonal matrix  $\mathbf{P}$  (where  $\mathbf{P}^\top \mathbf{P} = \mathbf{I}$ ):

$$\begin{aligned} 966 \langle \mathbf{P}^\top \mathbf{X} \mathbf{P}, \mathbf{P}^\top \mathbf{Y} \mathbf{P} \rangle_F &= \text{tr}((\mathbf{P}^\top \mathbf{X} \mathbf{P})^\top (\mathbf{P}^\top \mathbf{Y} \mathbf{P})) \\ 967 &= \text{tr}(\mathbf{P}^\top \mathbf{X}^\top \mathbf{P} \mathbf{P}^\top \mathbf{Y} \mathbf{P}) \\ 968 &= \text{tr}(\mathbf{P}^\top \mathbf{X}^\top \mathbf{Y} \mathbf{P}) \\ 969 &= \text{tr}(\mathbf{X}^\top \mathbf{Y}) \\ 970 &= \langle \mathbf{X}, \mathbf{Y} \rangle_F. \\ 971 \end{aligned}$$

972 Applying this property, the integrand simplifies:  
 973

$$\langle \mathbf{P}^\top v_\theta(\mathbf{A}_t, t) \mathbf{P}, \mathbf{P}^\top d\mathbf{A}_t \mathbf{P} \rangle_F = \langle v_\theta(\mathbf{A}_t, t), d\mathbf{A}_t \rangle_F$$

974 The integrand is identical for both the original and permuted inputs. Therefore, the integrals are  
 975 equal:  
 976

$$F(\mathbf{P}^\top \mathbf{A}_1 \mathbf{P}) = - \int_0^1 \langle v_\theta(\mathbf{A}_t, t), d\mathbf{A}_t \rangle_F dt + C = F(\mathbf{A}_1)$$

981 This confirms that the scalar function  $F$  is permutation-invariant.  $\square$   
 982

## 983 E EXPERIMENTAL DETAILS

### 984 E.1 DETAILS ABOUT THE ARCHITECTURE

985 Our model adopts a modified version of the adjacency score network architecture introduced in  
 986 GDSS (Jo et al., 2022). The network is a permutation-equivariant graph neural network designed  
 987 to approximate the scores  $\nabla_{\mathbf{A}_t} \log p_t(\mathbf{X}_t, \mathbf{A}_t)$  and  $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t, \mathbf{A}_t)$  at each diffusion step; in this  
 988 paper, we use only score w.r.t.  $\mathbf{A}_t$ . Concretely, the architecture consists of stacked message-passing  
 989 layers followed by a multi-layer perceptron. Each layer propagates node and edge information  
 990 through adjacency-based aggregation, ensuring equivariance under node relabeling. Time information  
 991  $t$  is incorporated by scaling intermediate activations with the variance of the forward diffusion process,  
 992 following the practice in continuous-time score models. Residual connections and normalization  
 993 layers are used to stabilize training. The final output is an  $N \times N$  tensor matching the adjacency  
 994 dimension. This design provides the required permutation-equivariance and expressive power while  
 995 remaining computationally tractable for mid-sized benchmark graphs.  
 996

997 The modification that incorporates is a module to build an embedding for the variable  $t$  and a FiLM  
 998 style modulation to incorporate noise conditioning. In particular, we incorporate the following  
 999 modules:  
 1000

- 1001 • A positional encoding based on a sinusoidal embedding following Karras et al. (2022)
- 1002 • An MLP layer with SiLU activation per attention layer
- 1003 • A modulation at each attention layer, where we scale the hidden features by an adaptive  
 1004 RMS norm operation (Crowson et al., 2024)

### 1005 E.2 DETAILS ABOUT THE DATASETS

1006 In Table 4 we report the statistics of the datasets used in the main text.

1007  
 1008 Table 4: Statistics of the datasets used for evaluation.

1009 Dataset	1010 # Graphs	1011 Avg. Nodes	1012 Avg. Edges	1013 # Classes	1014 Domain
1015 ENZYMES	1016 600	1017 32.63	1018 62.14	1019 6	1020 Bioinformatics
1021 PROTEINS	1022 1,113	1023 39.06	1024 72.82	1025 2	1026 Bioinformatics
1027 IMDB-B	1028 1,000	1029 19.77	1030 96.53	1031 2	1032 Social Network

### 1033 E.3 HYPERPARAMETERS

#### 1034 E.3.1 FLOW-BASED BASELINES

1035 We report the hyperparameters governing the model and training. All three baselines use the same  
 1036 rectified-flow architecture and optimizer family; the only substantive differences are the prior settings.

---

1026	<b>Node2Vec prior (per-graph link predictor)</b>	
1027	n2v_dim	64
1028	n2v_walk_length	30
1029	n2v_walks_per_node	10
1030	n2v_context	10
1031	n2v_p, n2v_q	1.0, 1.0
1032	n2v_epochs	1000
1033	clf_epochs	1000
1034	<b>PIFM (Node2Vec) Link Prediction, 10% masked</b>	
1035	batch_size	64 (IMDB-B & ENZYMES), 32 (PROTEINS)
1036	optimizer	Adam
1037	learning_rate	2e-4
1038	dropout	0.2
1039	hidden_dim	32
1040	num_layers	5
1041	num_linears	2
1042	channels	{c_init: 2, c_hid: 8, c_final: 4}
1043	train/val/test_noise_std	0.01(IMDB-B), 0.1 (PROTEINS & ENZYMES)
1044	ode_steps (Euler, $K$ )	1 to 100
1044	prior	Node2Vec (per-graph classifier)
1045	<b>PIFM (Node2Vec) Link Prediction, 50% masked</b>	
1046	batch_size	64 (IMDB-B & ENZYMES), 32 (PROTEINS)
1047	optimizer	Adam
1048	learning_rate	2e-4
1049	dropout	0.2
1050	hidden_dim	32
1051	num_layers	5
1052	num_linears	2
1052	channels	{c_init: 2, c_hid: 8, c_final: 4}
1053	train/val/test_noise_std	0.01(IMDB-B), 0.1 (PROTEINS & ENZYMES)
1054	ode_steps (Euler, $K$ )	1 to 100
1055	prior	Node2Vec (per-graph classifier)
1056		
1057		
1058		
1059		
1060		
1061		
1062	<b>Hyperparameter: PIFM(SIGL)</b>	
1063	<b>Value</b>	
1064	denoiser_epochs	1000
1065	SIGL_hyperparams	same as original paper
1065	optimizer	Adam
1066	learning_rate	2e-4
1067	dropout	0.2
1068	hidden_dim	32
1069	num_layers	5
1070	num_linears	2
1071	channels	{c_init: 2, c_hid: 8, c_final: 4}
1071	train_noise_std (masked $t=0$ )	0.05
1072	val_noise_std (masked $t=0$ )	0.05
1073	ode_steps (Euler, $K$ )	1000 (default)
1074	prior	SIGL (pretrained graphon; sort_ckpt, inr_ckpt)
1075		
1076		
1077		
1078		
1079		

---

1080	<b>PIFM (GraphSAGE) Link Prediction, 10% masked</b>	
1081	batch_size	64 (IMDB-B & ENZYMEs), 32 (PROTEINS)
1082	optimizer	Adam
1083	learning_rate	2e-4
1084	dropout	0.2
1085	hidden_dim	32
1086	num_layers	5
1087	num_linears	2
1088	channels	{c_init: 2, c_hid: 8, c_final: 4}
1089	train/val/test_noise_std	0.05 (IMDB-B), 0.1 (PROTEINS & ENZYMEs)
1090	ode_steps (Euler, $K$ )	1 to 100
1091	prior	GraphSAGE (default hyperparameters)
1092	<hr/>	
1093	<hr/>	
1094	<b>PIFM (GraphSAGE) Link Prediction, 50% masked</b>	
1095	batch_size	64 (IMDB-B & ENZYMEs), 32 (PROTEINS)
1096	optimizer	Adam
1097	learning_rate	2e-4
1098	dropout	0.2
1099	hidden_dim	32
1100	num_layers	5
1101	num_linears	2
1102	channels	{c_init: 2, c_hid: 8, c_final: 4}
1103	train/val/test_noise_std	0.05 (IMDB-B & PROTEINS), 0.1 (ENZYMEs)
1104	ode_steps (Euler, $K$ )	1 to 100
1105	prior	GraphSAGE (default hyperparameters)
1106	<hr/>	
1107	<hr/>	
1108	<b>Flow w/ Gaussian Prior</b>	<b>Value</b>
1109	epochs	1000
1110	batch_size	64 (default), 32 (PROTEINS)
1111	optimizer	Adam
1112	learning_rate	2e-4
1113	dropout	0.2
1114	hidden_dim	32
1115	num_layers	5
1116	num_linears	2
1117	channels	{c_init: 2, c_hid: 8, c_final: 4}
1118	train_noise_std (masked $t=0$ )	0.00
1119	val_noise_std (masked $t=0$ )	0.00
1120	ode_steps (Euler, $K$ )	1 to 100
1121	prior	None (masked entries initialized from $\mathcal{N}(0.5, 1)$ )
1122	<hr/>	
1123	<hr/>	
1124	<hr/>	
1125	<b>DiGress + RePaint</b>	<b>Value</b>
1126	train.n_epochs	3000
1127	train.batch_size	12
1128	model.diffusion_steps	1000
1129	model.n_layers	8
1130	model.lambda_train	[5, 0]
1131	model.extra_features	all
1132	model.hidden_mlp_dims	{X: 128, E: 64, y: 128}
1133	model.hidden_dims	{dx: 256, de: 64, dy: 64, n_head: 8, dim_ffX: 256, dim_ffE: 64, dim_ffy: 256}
1134	<hr/>	

1134	GDSS + RePaint	Setting
1135	Sampler predictor / corrector	S4 / None
1136	$n_{\text{steps}}$ / SNR / scale_eps	1 / 0.15 / 0.7
1137	Probability flow / noise removal / $\epsilon$	False / True / $10^{-5}$
1138	Batch size (DataLoader)	from config (e.g., 12)
1139	Mask mode (default)	dataset
1140	Observed graph $A_{\text{obs}}$	( $A_{\text{true}} \odot \text{mask}$ ); symmetrized, no self-loops
1141	Binarization threshold (metrics)	0.5

#### 1144 E.4 METRICS CALCULATION

1145 We evaluate performance only on the set of masked (unknown) edges in the upper triangle of the  
 1146 adjacency matrix. For each test graph, all metrics are computed on these entries and then averaged  
 1147 across graphs.

1149 **Metrics Used in Tables** We report the following four metrics in the main results:

- 1152 • **Area Under the ROC Curve (AUC).** Computed on the raw predicted scores (when available).  
 1153 AUC measures the probability that a randomly chosen true edge receives a higher predicted score  
 1154 than a randomly chosen non-edge. Larger AUC indicates stronger ranking performance.
- 1155 • **Average Precision (AP).** Computed from the precision–recall curve induced by ranking the  
 1156 predictions. AP summarizes how well the model recovers true edges across all possible thresholds,  
 1157 with higher values indicating better precision–recall trade-offs.
- 1158 • **False Positive Rate (FPR).** After thresholding predictions at 0.5, the FPR is defined as

$$1161 \quad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}},$$

$$1162$$

- 1163 • **False Negative Rate (FNR).** After thresholding predictions at 0.5, the FNR is defined as

$$1166 \quad \text{FNR} = \frac{\text{FN}}{\text{FN} + \text{TP}},$$

$$1167$$

- 1168 • **MMD.** A kernel-based method that measures the difference between two probability distributions  
 1169 by embedding them in a feature space and finding the maximum difference between their mean  
 1170 embeddings.

1172 AUC and AP are threshold-independent metrics (computed directly on the provided scores), while  
 1173 FPR and FNR are threshold-dependent error rates (obtained after binarizing at 0.5). All values  
 1174 reported in the tables are averaged over test graphs and expressed in percentage.

## 1177 F ADDITIONAL RESULTS

### 1179 F.1 DENOISING.

1181 This second problem is the complement of expansion, meaning that we seek to remove a set of  
 1182 spurious edges  $\mathcal{E}_S$  from  $A^{\mathcal{O}}$ , such that the edge set of the ground truth is  $\mathcal{E} = \mathcal{E}_O \setminus \mathcal{E}_S$ . Hence,  
 1183 the initialization becomes  $\mathbf{A}_0 = \mathbf{A}_1^{\mathcal{O}} \odot (f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}}) + \epsilon_s)$ . We assume that 20% of the edges are  
 1184 flipped; the results are shown in Table 5. Similarly to expansion, PIFM (GraphSAGE) attains the best  
 1185 AUC/AP on all datasets, again surpassing the GraphSAGE prior and remaining baselines. It *strongly*  
 1186 reduces false positives from the given prior initialization, while FNR is low on dense IMDB-B (2.67)  
 1187 and higher on sparser sets. Overall, PIFM removes spurious edges more reliably while improves  
 1188 other metrics as well.

1188 Table 5: Performance for the **denoising task with 20% of upper-triangle 0-entries flipped (0.2 Flip)**. We  
 1189 report AUC, Average Precision (AP↑), False Positive Rate (FPR↓), and False Negative Rate (FNR↓), all in  
 1190 percent (%). The best result for each metric is in **bold blue** and the second best is **green**. The metrics are  
 1191 restricted on the upper-triangle 1-region of  $A^{\mathcal{O}}$ , and compared against  $\mathbf{A}_1$  on that region.

Method	Flip Rate: 20% (0.2 Flip)											
	ENZYMEs				PROTEINS				IMDB-B			
	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓	AP ↑	AUC ↑	FNR ↓	FPR ↓
<i>Baselines</i>												
GraphSAGE	<b>68.19</b>	<b>73.89</b>	<b>16.14</b>	61.72	<b>73.79</b>	<b>76.70</b>	<b>12.68</b>	60.47	92.54	77.29	16.77	53.37
DiGress + RePaint	41.98	49.38	87.91	<b>13.33</b>	49.36	51.20	78.44	<b>18.19</b>	80.54	51.59	73.41	23.49
GDSS + RePaint	44.59	50.86	69.18	29.70	49.72	49.43	68.39	32.64	82.36	53.23	69.05	26.00
Flow w/ Gaussian prior	49.90	54.56	52.30	38.93	57.18	58.70	62.84	24.32	<b>96.75</b>	<b>94.63</b>	3.80	<b>12.66</b>
<i>Ours</i>												
PIFM (GraphSAGE)	<b>69.40</b>	<b>77.17</b>	<b>45.66</b>	18.14	<b>77.43</b>	<b>81.78</b>	32.41	20.91	<b>98.46</b>	<b>96.52</b>	<b>2.67</b>	<b>12.10</b>

## 1202 F.2 ADDITIONAL EXPERIMENTS

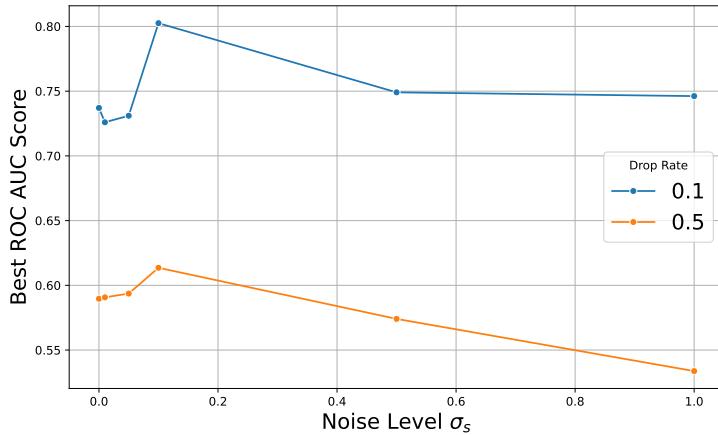
1204 Our method has two main hyperparameters:

1206 1.  $\sigma_s$ , which is used for computing  $\epsilon_s \sim \mathcal{N}(0, \text{sigma}_s^2)$  in  $\mathbf{A}_0 = \xi \odot \mathbf{A}_1 + (1 - \xi) \odot$   
 1207  $(f_{\text{prior}}(\mathbf{A}_1^{\mathcal{O}}) + \epsilon_s)$   
 1208 2.  $K$ , which are the total number of steps in the Euler approximation

### 1211 F.2.1 PERFORMANCE AS A FUNCTION OF $\sigma_s$

1213 We run an ablation of the performance of PIFM with GraphSAGE as a function of  $\sigma_s$ . We focus on  
 1214 ENZYMEs and IMDB, and we evaluate the ROC for the best value of  $K$  for each noise level.

1215 The ablation is illustrated in Fig. 4. First, we observe that the gains of using PIFM are higher for  
 1216 a smaller drop rate, as expected; in particular, we observe that PIFM with  $\sigma_s$  jumps from  $\approx 0.73$   
 1217 for  $\sigma_s = 0$  to  $\approx 0.81$  for  $\sigma_s = 0.1$ . Second, for both configurations, performance peaks not at zero  
 1218 noise, but at a small noise level of  $\sigma_s = 0.1$ . This suggests that a slight injection of noise benefits  
 1219 model generalization. Beyond this optimal point, increasing the noise level leads to a steady decline  
 1220 in performance, meaning that the effect of the prior decreases, as expected.

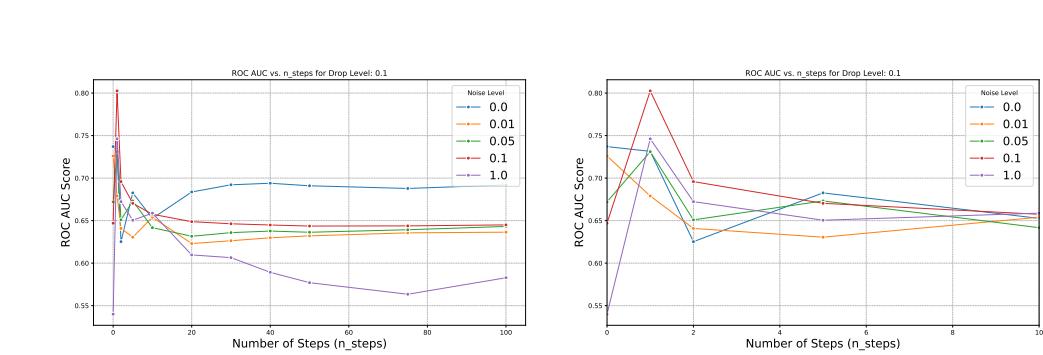


1238 Figure 4: ROC as a function of the noise  $\sigma_s$  in  $p(\mathbf{A}_0)$ . The impact of noise level  $\sigma_s$  on model  
 1239 performance, measured by the best ROC AUC score. Results are shown for two different drop rates:  
 1240 0.1 (blue) and 0.5 (orange). A small amount of noise improves performance for both configurations,  
 1241 after which increasing noise leads to performance degradation.

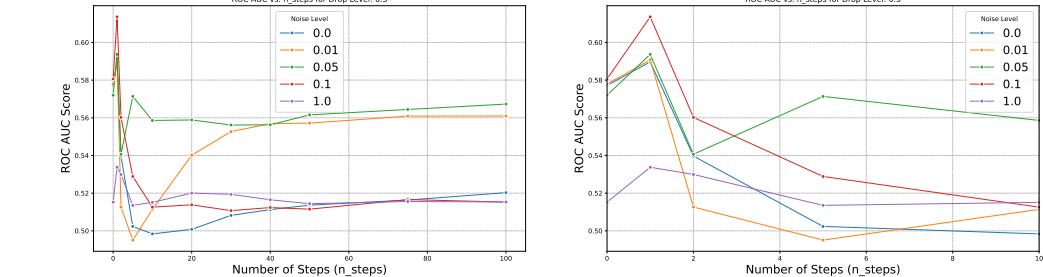
1242 F.2.2 PERFORMANCE AS A FUNCTION OF  $K$   
1243

1244 To determine the optimal number of processing steps,  $K$ , we evaluated model performance while  
1245 varying this parameter from 1 to 100. Figures 5 and 6 shows the results for a fixed drop rate of 0.1  
1246 and 0.5 respectively, across five different noise levels.

1247 A key observation is that peak performance, in terms of AUC-ROC, is achieved within a very small  
1248 number of steps, typically for  $K < 10$ . In particular, the introduction of a moderate noise level allows  
1249 the model to achieve its highest overall score ( $\approx 0.80$  ROC AUC) in a single step ( $K = 1$ ). However,  
1250 this advantage diminishes as the number of steps increases. The model without noise ( $\sigma_s = 0.0$ )  
1251 provides the most stable and consistently high performance for larger  $K$ . Conversely, a high noise  
1252 level ( $\sigma_s = 1$ ) consistently degrades performance regardless of the number of steps. This analysis  
1253 suggests a trade-off: while noise can provide a significant boost for models with very few steps, a  
1254 no-noise configuration is more robust for models with a larger number of steps.



1267 Figure 5: An analysis of the ROC AUC score as a function of the number of processing steps ( $K$ ) for  
1268 a drop rate of 10%. This experiment was conducted with a fixed drop rate of 0.1, while varying the  
1269 noise level,  $\sigma_s$ . The results show that the optimal number of steps is small, typically under 10.



1284 Figure 6: An analysis of the ROC AUC score as a function of the number of processing steps ( $K$ ) for  
1285 a drop rate of 50%. This experiment was conducted with a fixed drop rate of 0.1, while varying the  
1286 noise level,  $\sigma_s$ . The results show that the optimal number of steps is small, typically under 10.

1289 F.3 DISTORTION-PERCEPTION TRADE-OFF.  
1290

1291 Here we expand on the distortion-perception trade-off by computing the MMD. The results are shown  
1292 in Figures 7 and 8. Again, both figures show that the MMD<sup>2</sup> distance decreases as we increase  
1293  $K$ ; this is particular noticeable for  $0 < \sigma_s \leq 0.1$ . In other words, if we aim for a high-quality  
1294 perceptual reconstruction, we should consider  $\sigma_s = 0.01$  or  $0.05$ . However, if we are aiming for high  
1295 reconstruction quality in terms of AUC-ROC, we should use  $\sigma_s = 0.1$  (see Fig.4). In other words,  
1296 the choice of the  $K$  is heavily dependent of the downstream task.

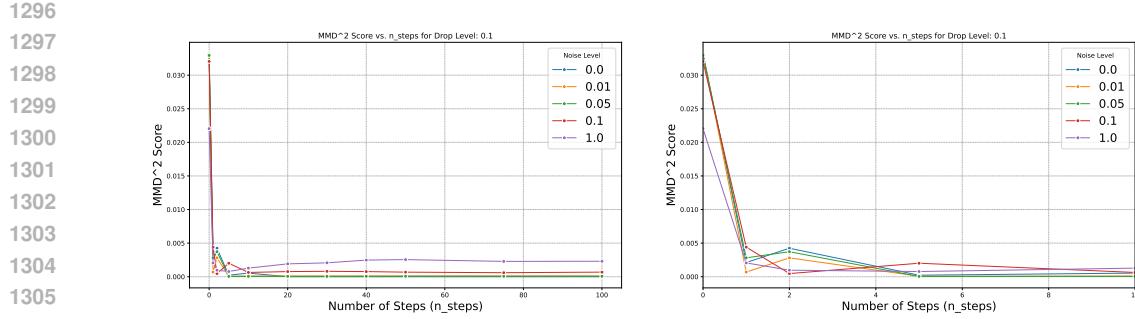


Figure 7: Analysis of the perception component of the distortion-perception trade-off. The plot shows the MMD<sup>2</sup> score (where lower is better) versus the number of steps, K, for a fixed drop rate of 0.1. Each line represents a different noise level  $\sigma_s$ .

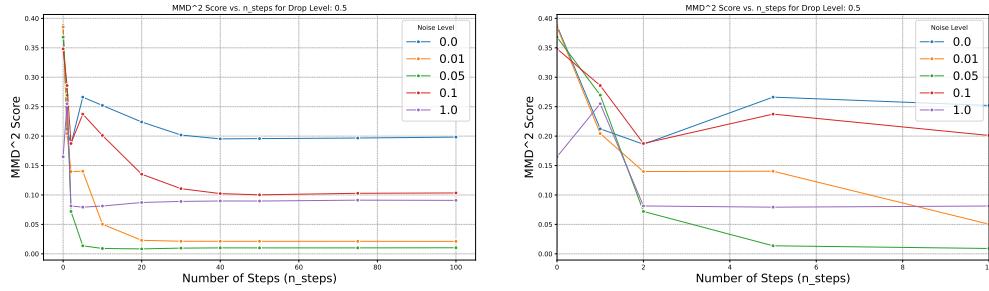
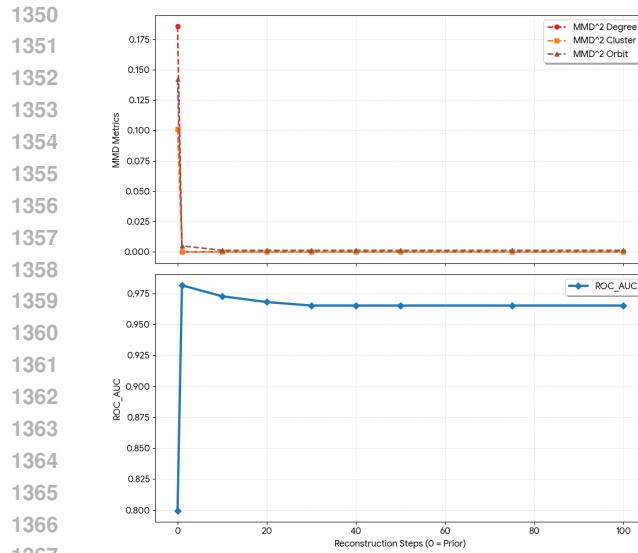


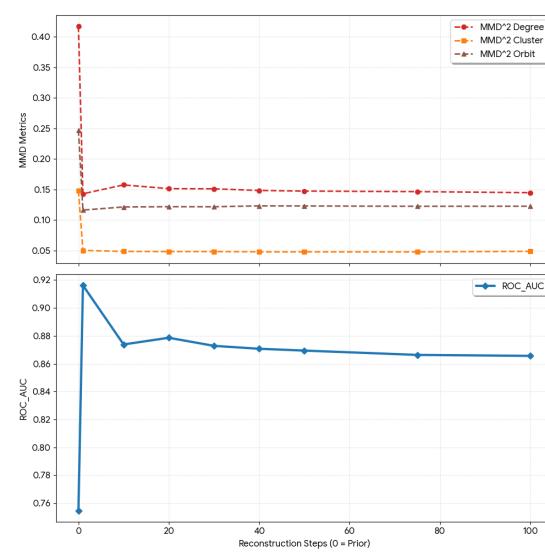
Figure 8: Analysis of the perception component of the distortion-perception trade-off. The plot shows the MMD<sup>2</sup> score (where lower is better) versus the number of steps, K, for a fixed drop rate of 0.5. Each line represents a different noise level  $\sigma_s$ .

### F.3.1 ADDITIONAL METRICS

This section incorporates additional metrics to showcase the observed performance trade-off. The results for IMDB are in Figs. 9 and 10, for PROTEINS in Figs. 11 and 12, and for ENZYMES in Figs. 13 and 14. While an increased number of steps yields an improvement in generating an estimated graph with statistics that more closely align with the ground-truth distribution, the reconstruction performance (measured in terms of AUC) declines relative to the initial step. Critically, the trend is found to be highly contingent on the underlying dataset's sparsity. For the dense case (IMDB), the AUC exhibits a consistent monotonic decrease after the optimal initial guess, independent of drop rates. In contrast, the sparser PROTEINS and ENZYMES datasets demonstrate an intermediate improvement as the number of steps increases, though their overall AUC still trails that achieved at the initial step ( $t = 1$ ).



1368 Figure 9: IMDB dataset, expansion task,  
1369 10% drop rate



1368 Figure 10: IMDB dataset, expansion task,  
1369 50% drop rate

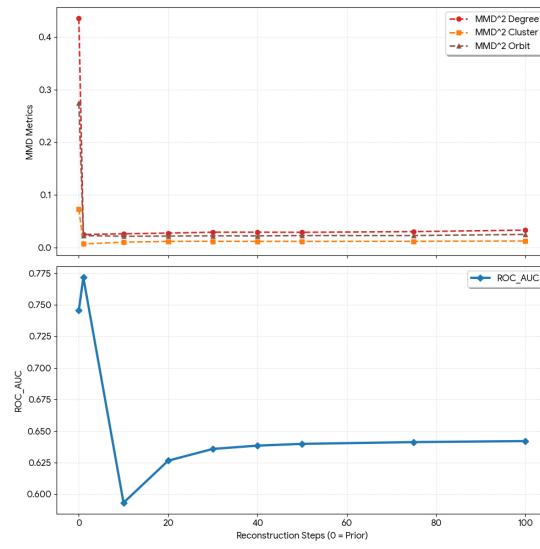


Figure 11: PROTEINS dataset, expansion, 50% drop rate

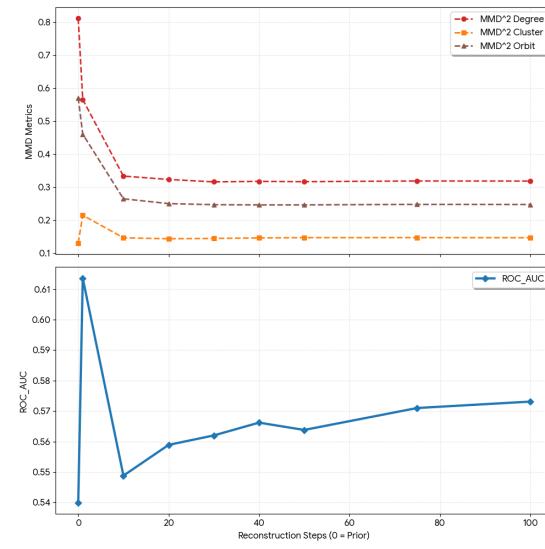


Figure 12: PROTEINS dataset, expansion, 50% drop rate

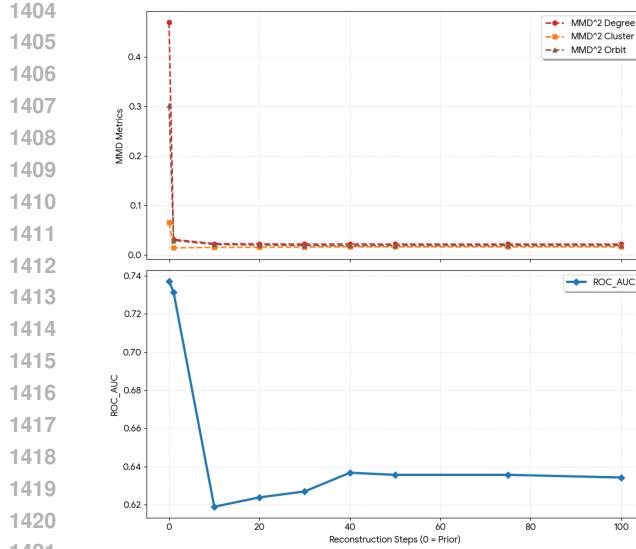


Figure 13: ENZYMES dataset, expansion, 10% drop rate

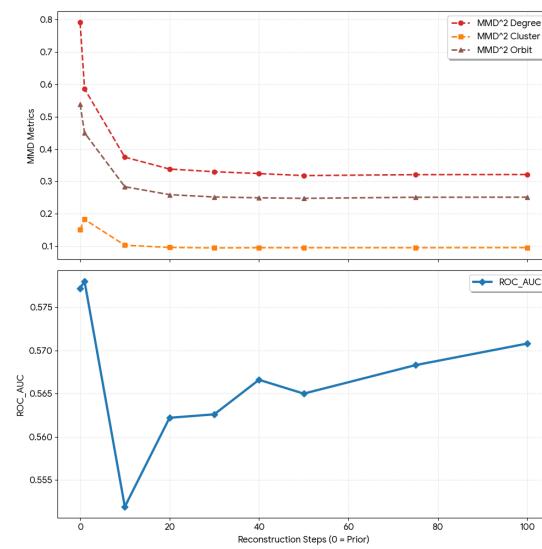


Figure 14: ENZYMES dataset, expansion, 50% drop rate

#### F.4 EXAMPLES OF RECONSTRUCTED GRAPHS

We show here a few samples for the expansion case. We plot the samples from ENZYMES, using a subset of the dataset used in Section 5.3.

**Binary comparison.** In this case, we first compare the thresholded versions (with 0.5) of the mean matrices. We compute this for 3 graphs in the test set (3, 7 and 29). The plots are in Figs. 15, 16 and 17

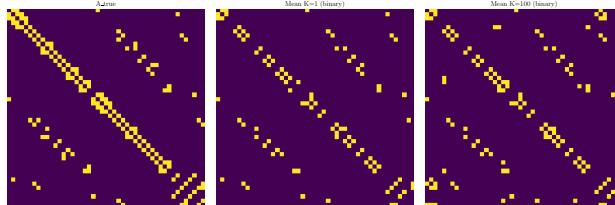


Figure 15: Graph reconstruction for sample 3, thresholded with 0.5

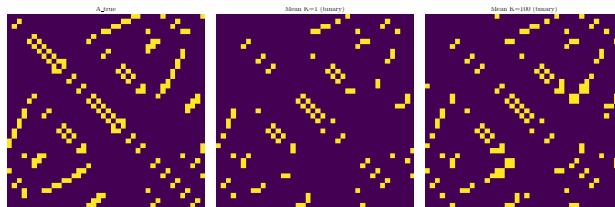


Figure 16: Graph reconstruction for sample 7, thresholded with 0.5

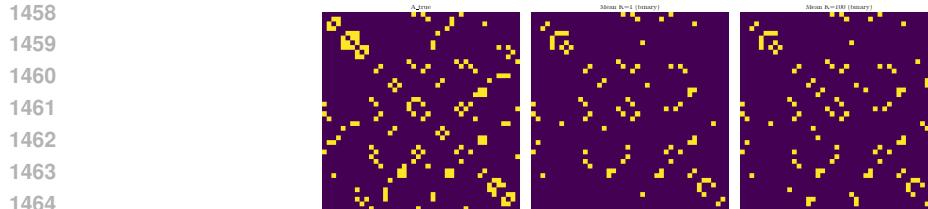


Figure 17: Graph reconstruction for sample 30, thresholded with 0.5

1469  
1470  
1471  
1472  
1473

**Raw comparison - Mean.** In this case, we compare the raw versions of the mean matrices. We compute this for 3 graphs in the test set (3, 7 and 29). The plots are in Figs. 18, 19 and 20. Notice that the mean reconstructions for  $K = 100$  have values that are between 0 and 1; this can be explained by looking at individual samples (see below), which are more diverse, and therefore, they have non-overlapping set of existing edges.

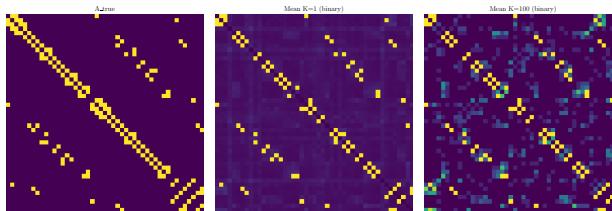


Figure 18: Graph reconstruction for sample 3, mean raw

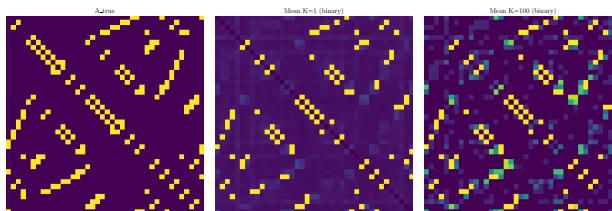


Figure 19: Graph reconstruction for sample 7, mean raw

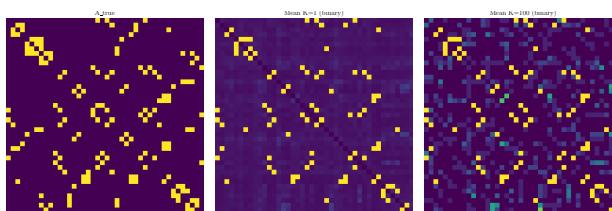


Figure 20: Graph reconstruction for sample 30, mean raw

**Raw comparison - Median.** In this case, we compare the raw versions of the median matrices. We compute this for 3 graphs in the test set (3, 7 and 29). The plots are in Figs. 21, 22 and 23.

1512

1513

1514

1515

1516

1517

1518

1519

1520

1521

1522

1523

1524

1525

1526

1527

1528

1529

1530

1531

1532

1533

1534

1535

1536

1537

1538

1539

1540

1541

1542

1543

1544

1545

1546

1547

1548

**Individual samples for each graph.** Lastly, we show the raw versions of different realizations (individual samples) for each graph. We compute this for 3 graphs in the test set (3, 7 and 29). Interesting, the samples for  $K = 100$  are more diverse (similar to the case of images in Ohayon et al. (2025)); this diversity explains why the raw mean in Figs. 18, 19 and 20 have values that are not exactly 0 or 1 (which means that there are non-overal between samples).

1553

1554

1555

1556

1557

1558

1559

1560

1561

1562

1563

1564

1565

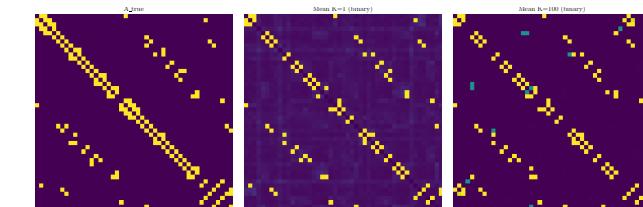


Figure 21: Graph reconstruction for sample 3, median raw

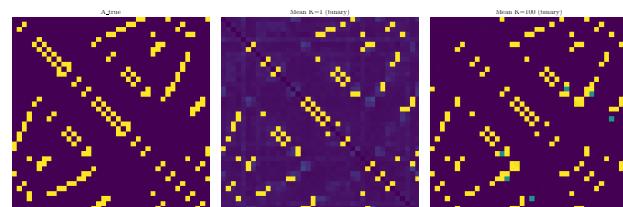


Figure 22: Graph reconstruction for sample 7, median raw

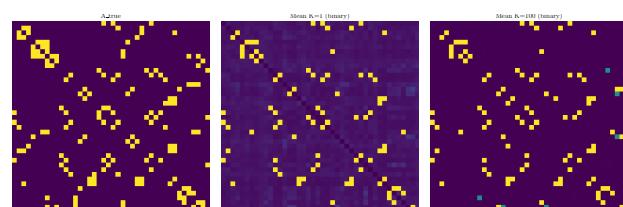
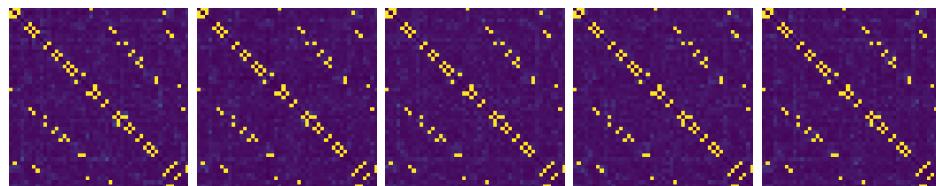


Figure 23: Graph reconstruction for sample 30, median raw

Figure 24: Individual samples for  $K = 1$  and for sample 3

1566

1567

1568

1569

1570

1571

1572

1573

1574

1575

1576

1577

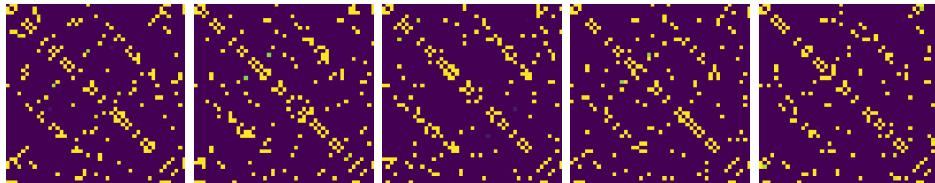


Figure 25: Individual samples for  $K = 100$  and for sample 3

1578

1579

1580

1581

1582

1583

1584

1585

1586

1587

1588

1589

1590

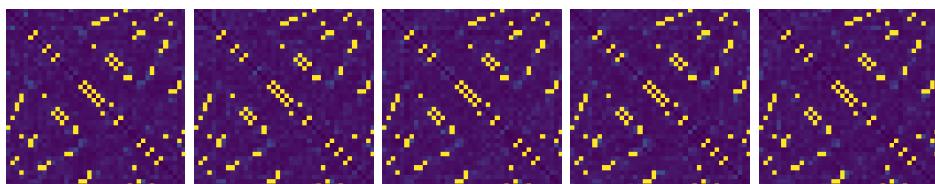


Figure 26: Individual samples for  $K = 1$  and for sample 7

1591

1592

1593

1594

1595

1596

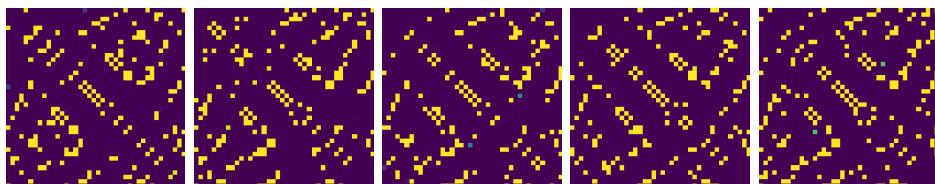


Figure 27: Individual samples for  $K = 100$  and for sample 7

1606

1607

1608

1609

1610

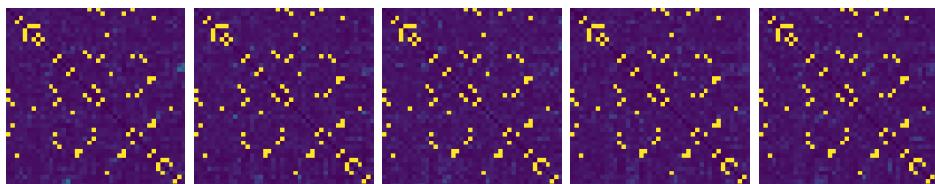
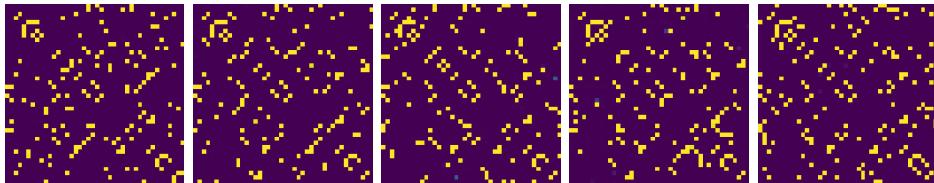


Figure 28: Individual samples for  $K = 1$  and for sample 30

Figure 29: Individual samples for  $K = 100$  and for sample 30

## F.5 PIFM ON LARGE-SCALE GRAPHS

In this section, we train PIFM on large-scale graphs. In particular, we focus on CORA Yang et al. (2016).

**Building the dataset.** To enable scalable diffusion training on Cora while maintaining full-graph link prediction capabilities, we introduce a subgraph-based variant of PIFM following Limnios et al. (2023). We instantiate this via an edge-centered sampling scheme, where each subgraph represents a  $k$ -hop ego-network (capped at a maximum node count) centered around a seed edge from the training split. To ensure reproducibility, we sample a fixed set of seed edges and corresponding subgraphs which remain constant throughout each run.

Following a standard 85/5/10 (train/validation/test) edge split, we evaluate link prediction on held-out edges using the protocol established in the main paper. Furthermore, we utilize purely structural node features. By default, we concatenate: (i) Laplacian positional encodings derived from the smallest  $k$  generalized eigenvectors of  $Lv = \lambda Dv$  computed on the training adjacency; and (ii) a 2-dimensional local context vector comprising both raw and normalized node degrees.

**Training.** To initialize PIFM, we first pre-train structural priors (GraphSAGE and Node2Vec) on the full Cora graph using only the training edge split (85% of edges). These learned embeddings are subsequently used to initialize PIFM for inference on the test split (10% of edges).

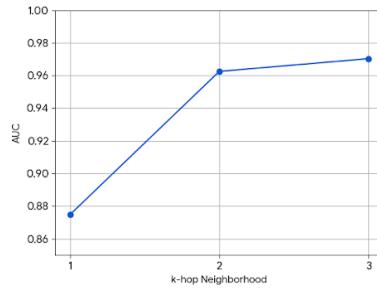
During PIFM training, each training edge seeds a unique edge-centered subgraph. Within each subgraph, we define the observed context as all other training edges, and a hidden region comprising all remaining node pairs (including non-edges and edges outside the context). The seed edge is explicitly masked from the context and designated as the sole supervision target. Consequently, each training example tasks the model with reconstructing a single missing edge within its local neighborhood.

We construct the initial state matrix  $\mathbf{A}_0$  by combining the observed context with structural prior predictions in the hidden region. The flow model is then trained to denoise  $\mathbf{A}_0$  toward the ground-truth local adjacency. While the flow ODE updates all entries within the hidden region, the gradient is computed exclusively from the seed edge. This adapts the global training procedure of Algorithm 1 to a localized, subgraph-based regime

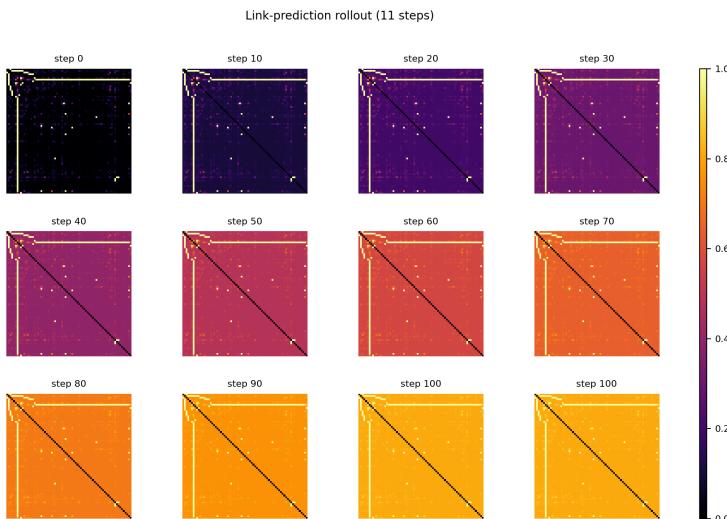
**Inference.** For inference, every held-out validation or test edge seeds a  $k$ -hop subgraph. We define the observed context using the training edges, remove the centered test edge from the mask, and apply diffusion to the resulting hidden region. We strictly evaluate the prediction for the centered held-out edge in each subgraph. To resolve potential overlapping, we aggregate predictions via logit averaging across all subgraphs where a specific edge is present, producing a single probability matrix over all node pairs. We then compute metrics on the held-out positive and negative edges. Since PIFM and the structural prior baseline are evaluated on the exact same set of pairs, the results isolate the specific benefits of the diffusion process

1674  
 1675  
 1676  
 1677  
 1678  
 Table 6: Link prediction on CORA (hidden edges) with a GraphSAGE structural prior. We report AUC, Average  
 1679  
 1680  
 1681  
 1682  
 1683  
 1684  
 1685  
 1686  
 1687  
 1688  
 1689  
 1690  
 1691  
 1692  
 1693  
 1694  
 1695  
 1696  
 1697  
 1698  
 1699  
 1700  
 1701  
 1702  
 1703  
 1704  
 1705  
 1706  
 1707  
 1708  
 1709  
 1710  
 1711  
 1712  
 1713  
 1714  
 1715  
 1716  
 1717  
 1718  
 1719  
 1720  
 1721  
 1722  
 1723  
 1724  
 1725  
 1726  
 1727  
 We report PIFM's performance on Cora for different  $k$ -hop neighborhoods during subgraph sampling, and the  
 1679  
 1680  
 1681  
 1682  
 1683  
 1684  
 1685  
 1686  
 1687  
 1688  
 1689  
 1690  
 1691  
 1692  
 1693  
 1694  
 1695  
 1696  
 1697  
 1698  
 1699  
 1700  
 1701  
 1702  
 1703  
 1704  
 1705  
 1706  
 1707  
 1708  
 1709  
 1710  
 1711  
 1712  
 1713  
 1714  
 1715  
 1716  
 1717  
 1718  
 1719  
 1720  
 1721  
 1722  
 1723  
 1724  
 1725  
 1726  
 1727  
 embedding dimension of the laplacian positional encodings. We include also the average node counts for each  
 1679  
 1680  
 1681  
 1682  
 1683  
 1684  
 1685  
 1686  
 1687  
 1688  
 1689  
 1690  
 1691  
 1692  
 1693  
 1694  
 1695  
 1696  
 1697  
 1698  
 1699  
 1700  
 1701  
 1702  
 1703  
 1704  
 1705  
 1706  
 1707  
 1708  
 1709  
 1710  
 1711  
 1712  
 1713  
 1714  
 1715  
 1716  
 1717  
 1718  
 1719  
 1720  
 1721  
 1722  
 1723  
 1724  
 1725  
 1726  
 1727  
 subgraph for each case.

Method	AUC↑	AP↑	FPR↓	FNR↓
GraphSAGE	95.61	47.76	0.59	23.79
1hop 2dim-embed (~ 16.7 nodes)	87.47	61.07	1.00	0.00
2hop 2dim-embed (~ 58.0 nodes)	96.25	75.39	1.00	0.00
3hop 2dim-embed (~ 154.5 nodes)	<b>97.03</b>	33.71	1.00	0.00
4hop 8dim-embed (~ 210.4 nodes)	96.14	21.53	1.00	0.00
NCNC Wang et al. (2024) (from Li et al. (2023))	96.90	-	-	-
NCN Wang et al. (2024) (from Li et al. (2023))	96.76	-	-	-



1705  
 1706  
 1707  
 Figure 30: AUC vs  $k$ -hop neighborhood. We observe that adding more hops enhances the performance  
 1708  
 1709  
 1710  
 1711  
 1712  
 1713  
 1714  
 1715  
 1716  
 1717  
 1718  
 1719  
 1720  
 1721  
 1722  
 1723  
 1724  
 1725  
 1726  
 1727  
 of PIFM, as it exploits more information.



1727  
 1728  
 1729  
 1730  
 1731  
 1732  
 1733  
 1734  
 1735  
 1736  
 1737  
 1738  
 1739  
 1740  
 1741  
 1742  
 1743  
 1744  
 1745  
 1746  
 1747  
 1748  
 1749  
 1750  
 1751  
 1752  
 1753  
 1754  
 1755  
 1756  
 1757  
 1758  
 1759  
 1760  
 1761  
 1762  
 1763  
 1764  
 1765  
 1766  
 1767  
 1768  
 1769  
 1770  
 1771  
 1772  
 1773  
 1774  
 1775  
 1776  
 1777  
 1778  
 1779  
 1780  
 1781  
 1782  
 1783  
 1784  
 1785  
 1786  
 1787  
 1788  
 1789  
 1790  
 1791  
 1792  
 1793  
 1794  
 1795  
 1796  
 1797  
 1798  
 1799  
 1800  
 1801  
 1802  
 1803  
 1804  
 1805  
 1806  
 1807  
 1808  
 1809  
 1810  
 1811  
 1812  
 1813  
 1814  
 1815  
 1816  
 1817  
 1818  
 1819  
 1820  
 1821  
 1822  
 1823  
 1824  
 1825  
 1826  
 1827  
 1828  
 1829  
 1830  
 1831  
 1832  
 1833  
 1834  
 1835  
 1836  
 1837  
 1838  
 1839  
 1840  
 1841  
 1842  
 1843  
 1844  
 1845  
 1846  
 1847  
 1848  
 1849  
 1850  
 1851  
 1852  
 1853  
 1854  
 1855  
 1856  
 1857  
 1858  
 1859  
 1860  
 1861  
 1862  
 1863  
 1864  
 1865  
 1866  
 1867  
 1868  
 1869  
 1870  
 1871  
 1872  
 1873  
 1874  
 1875  
 1876  
 1877  
 1878  
 1879  
 1880  
 1881  
 1882  
 1883  
 1884  
 1885  
 1886  
 1887  
 1888  
 1889  
 1890  
 1891  
 1892  
 1893  
 1894  
 1895  
 1896  
 1897  
 1898  
 1899  
 1900  
 1901  
 1902  
 1903  
 1904  
 1905  
 1906  
 1907  
 1908  
 1909  
 1910  
 1911  
 1912  
 1913  
 1914  
 1915  
 1916  
 1917  
 1918  
 1919  
 1920  
 1921  
 1922  
 1923  
 1924  
 1925  
 1926  
 1927  
 1928  
 1929  
 1930  
 1931  
 1932  
 1933  
 1934  
 1935  
 1936  
 1937  
 1938  
 1939  
 1940  
 1941  
 1942  
 1943  
 1944  
 1945  
 1946  
 1947  
 1948  
 1949  
 1950  
 1951  
 1952  
 1953  
 1954  
 1955  
 1956  
 1957  
 1958  
 1959  
 1960  
 1961  
 1962  
 1963  
 1964  
 1965  
 1966  
 1967  
 1968  
 1969  
 1970  
 1971  
 1972  
 1973  
 1974  
 1975  
 1976  
 1977  
 1978  
 1979  
 1980  
 1981  
 1982  
 1983  
 1984  
 1985  
 1986  
 1987  
 1988  
 1989  
 1990  
 1991  
 1992  
 1993  
 1994  
 1995  
 1996  
 1997  
 1998  
 1999  
 2000  
 2001  
 2002  
 2003  
 2004  
 2005  
 2006  
 2007  
 2008  
 2009  
 2010  
 2011  
 2012  
 2013  
 2014  
 2015  
 2016  
 2017  
 2018  
 2019  
 2020  
 2021  
 2022  
 2023  
 2024  
 2025  
 2026  
 2027  
 2028  
 2029  
 2030  
 2031  
 2032  
 2033  
 2034  
 2035  
 2036  
 2037  
 2038  
 2039  
 2040  
 2041  
 2042  
 2043  
 2044  
 2045  
 2046  
 2047  
 2048  
 2049  
 2050  
 2051  
 2052  
 2053  
 2054  
 2055  
 2056  
 2057  
 2058  
 2059  
 2060  
 2061  
 2062  
 2063  
 2064  
 2065  
 2066  
 2067  
 2068  
 2069  
 2070  
 2071  
 2072  
 2073  
 2074  
 2075  
 2076  
 2077  
 2078  
 2079  
 2080  
 2081  
 2082  
 2083  
 2084  
 2085  
 2086  
 2087  
 2088  
 2089  
 2090  
 2091  
 2092  
 2093  
 2094  
 2095  
 2096  
 2097  
 2098  
 2099  
 2100  
 2101  
 2102  
 2103  
 2104  
 2105  
 2106  
 2107  
 2108  
 2109  
 2110  
 2111  
 2112  
 2113  
 2114  
 2115  
 2116  
 2117  
 2118  
 2119  
 2120  
 2121  
 2122  
 2123  
 2124  
 2125  
 2126  
 2127  
 2128  
 2129  
 2130  
 2131  
 2132  
 2133  
 2134  
 2135  
 2136  
 2137  
 2138  
 2139  
 2140  
 2141  
 2142  
 2143  
 2144  
 2145  
 2146  
 2147  
 2148  
 2149  
 2150  
 2151  
 2152  
 2153  
 2154  
 2155  
 2156  
 2157  
 2158  
 2159  
 2160  
 2161  
 2162  
 2163  
 2164  
 2165  
 2166  
 2167  
 2168  
 2169  
 2170  
 2171  
 2172  
 2173  
 2174  
 2175  
 2176  
 2177  
 2178  
 2179  
 2180  
 2181  
 2182  
 2183  
 2184  
 2185  
 2186  
 2187  
 2188  
 2189  
 2190  
 2191  
 2192  
 2193  
 2194  
 2195  
 2196  
 2197  
 2198  
 2199  
 2200  
 2201  
 2202  
 2203  
 2204  
 2205  
 2206  
 2207  
 2208  
 2209  
 2210  
 2211  
 2212  
 2213  
 2214  
 2215  
 2216  
 2217  
 2218  
 2219  
 2220  
 2221  
 2222  
 2223  
 2224  
 2225  
 2226  
 2227  
 2228  
 2229  
 2230  
 2231  
 2232  
 2233  
 2234  
 2235  
 2236  
 2237  
 2238  
 2239  
 2240  
 2241  
 2242  
 2243  
 2244  
 2245  
 2246  
 2247  
 2248  
 2249  
 2250  
 2251  
 2252  
 2253  
 2254  
 2255  
 2256  
 2257  
 2258  
 2259  
 2260  
 2261  
 2262  
 2263  
 2264  
 2265  
 2266  
 2267  
 2268  
 2269  
 2270  
 2271  
 2272  
 2273  
 2274  
 2275  
 2276  
 2277  
 2278  
 2279  
 2280  
 2281  
 2282  
 2283  
 2284  
 2285  
 2286  
 2287  
 2288  
 2289  
 2290  
 2291  
 2292  
 2293  
 2294  
 2295  
 2296  
 2297  
 2298  
 2299  
 2300  
 2301  
 2302  
 2303  
 2304  
 2305  
 2306  
 2307  
 2308  
 2309  
 2310  
 2311  
 2312  
 2313  
 2314  
 2315  
 2316  
 2317  
 2318  
 2319  
 2320  
 2321  
 2322  
 2323  
 2324  
 2325  
 2326  
 2327  
 2328  
 2329  
 2330  
 2331  
 2332  
 2333  
 2334  
 2335  
 2336  
 2337  
 2338  
 2339  
 2340  
 2341  
 2342  
 2343  
 2344  
 2345  
 2346  
 2347  
 2348  
 2349  
 2350  
 2351  
 2352  
 2353  
 2354  
 2355  
 2356  
 2357  
 2358  
 2359  
 2360  
 2361  
 2362  
 2363  
 2364  
 2365  
 2366  
 2367  
 2368  
 2369  
 2370  
 2371  
 2372  
 2373  
 2374  
 2375  
 2376  
 2377  
 2378  
 2379  
 2380  
 2381  
 2382  
 2383  
 2384  
 2385  
 2386  
 2387  
 2388  
 2389  
 2390  
 2391  
 2392  
 2393  
 2394  
 2395  
 2396  
 2397  
 2398  
 2399  
 2400  
 2401  
 2402  
 2403  
 2404  
 2405  
 2406  
 2407  
 2408  
 2409  
 2410  
 2411  
 2412  
 2413  
 2414  
 2415  
 2416  
 2417  
 2418  
 2419  
 2420  
 2421  
 2422  
 2423  
 2424  
 2425  
 2426  
 2427  
 2428  
 2429  
 2430  
 2431  
 2432  
 2433  
 2434  
 2435  
 2436  
 2437  
 2438  
 2439  
 2440  
 2441  
 2442  
 2443  
 2444  
 2445  
 2446  
 2447  
 2448  
 2449  
 2450  
 2451  
 2452  
 2453  
 2454  
 2455  
 2456  
 2457  
 2458  
 2459  
 2460  
 2461  
 2462  
 2463  
 2464  
 2465  
 2466  
 2467  
 2468  
 2469  
 2470  
 2471  
 2472  
 2473  
 2474  
 2475  
 2476  
 2477  
 2478  
 2479  
 2480  
 2481  
 2482  
 2483  
 2484  
 2485  
 2486  
 2487  
 2488  
 2489  
 2490  
 2491  
 2492  
 2493  
 2494  
 2495  
 2496  
 2497  
 2498  
 2499  
 2500  
 2501  
 2502  
 2503  
 2504  
 2505  
 2506  
 2507  
 2508  
 2509  
 2510  
 2511  
 2512  
 2513  
 2514  
 2515  
 2516  
 2517  
 2518  
 2519  
 2520  
 2521  
 2522  
 2523  
 2524  
 2525  
 2526  
 2527  
 2528  
 2529  
 2530  
 2531  
 2532  
 2533  
 2534  
 2535  
 2536  
 2537  
 2538  
 2539  
 2540  
 2541  
 2542  
 2543  
 2544  
 2545  
 2546  
 2547  
 2548  
 2549  
 2550  
 2551  
 2552  
 2553  
 2554  
 2555  
 2556  
 2557  
 2558  
 2559  
 2560  
 2561  
 2562  
 2563  
 2564  
 2565  
 2566  
 2567  
 2568  
 2569  
 2570  
 2571  
 2572  
 2573  
 2574  
 2575  
 2576  
 2577  
 2578  
 2579  
 2580  
 2581  
 2582  
 2583  
 2584  
 2585  
 2586  
 2587  
 2588  
 2589  
 2590  
 2591  
 2592  
 2593  
 2594  
 2595  
 2596  
 2597  
 2598  
 2599  
 2600  
 2601  
 2602  
 2603  
 2604  
 2605  
 2606  
 2607  
 2608  
 2609  
 2610  
 2611  
 2612  
 2613  
 2614  
 2615  
 2616  
 2617  
 2618  
 2619  
 2620  
 2621  
 2622  
 2623  
 2624  
 2625  
 2626  
 2627  
 2628  
 2629  
 2630  
 2631  
 2632  
 2633  
 2634  
 2635  
 2636  
 2637  
 2638  
 2639  
 2640  
 2641  
 2642  
 2643  
 2644  
 2645  
 2646  
 2647  
 2648  
 2649  
 2650  
 2651  
 2652  
 2653  
 2654  
 2655  
 2656  
 2657  
 2658  
 2659  
 2660  
 2661  
 2662  
 2663  
 2664  
 2665  
 2666  
 2667  
 2668  
 2669  
 2670  
 2671  
 2672  
 2673  
 2674  
 2675  
 2676  
 2677  
 2678  
 2679  
 2680  
 2681  
 2682  
 2683  
 2684  
 2685  
 2686  
 2687  
 2688  
 2689  
 2690  
 2691  
 2692  
 2693  
 2694  
 2695  
 2696  
 2697  
 2698  
 2699  
 2700  
 2701  
 2702  
 2703  
 2704  
 2705  
 2706  
 2707  
 2708  
 2709  
 2710  
 2711  
 2712  
 2713  
 2714  
 2715  
 2716  
 2717  
 2718  
 2719  
 2720  
 2721  
 2722  
 2723  
 2724  
 2725  
 2726  
 2727  
 2728  
 2729  
 2730  
 2731  
 2732  
 2733  
 2734  
 2735  
 2736  
 2737  
 2738  
 2739  
 2740  
 2741  
 2742  
 2743  
 2744  
 2745  
 2746  
 2747  
 2748  
 2749  
 2750  
 2751  
 2752  
 2753  
 2754  
 2755  
 2756  
 2757  
 2758  
 2759  
 2760  
 2761  
 2762  
 2763  
 2764  
 2765  
 2766  
 2767  
 2768  
 2769  
 2770  
 2771  
 2772  
 2773  
 2774  
 2775  
 2776  
 2777  
 2778  
 2779  
 2780  
 2781  
 2782  
 2783  
 2784  
 2785  
 2786  
 2787  
 2788  
 2789  
 2790  
 2791  
 2792  
 2793  
 2794  
 2795  
 2796  
 2797

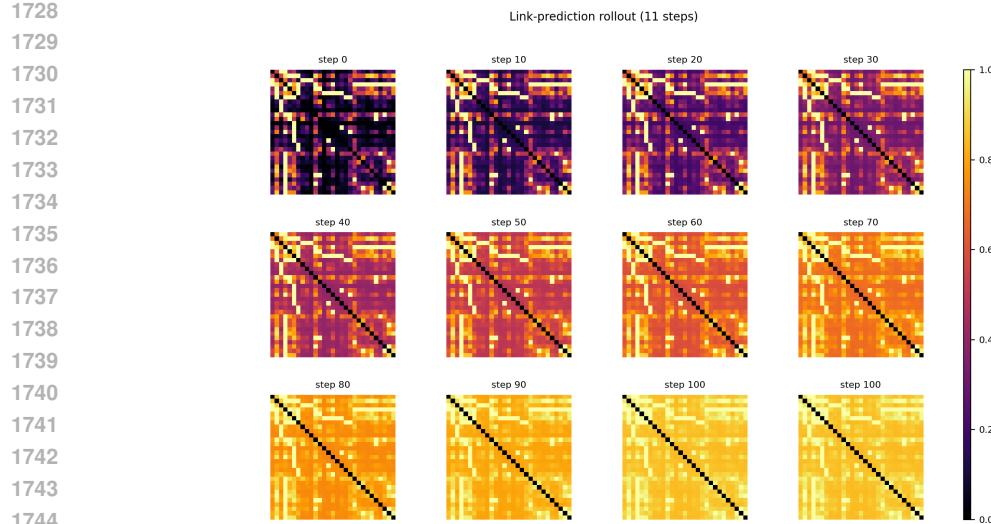


Figure 32: Visualization of the intermediate adjacency matrices of a sagraph for CORA.

## F.6 INTERMEDIATE ADJACENCY MATRICES

In Figs. 33- 35, we show visualizations of the diffusion trajectory of the link-prediction sampler by "snapshotting" the predicted adjacency matrix at steps 0 to 100 in 10-step-increment of a 100-step-total sample path. Each panel shows the raw adjacency values, zeroed on the diagonal, rendered with the colormap with black associated to 0 and yellow to 1. The most bottom-right panel is the ground-truth adjacency for comparison.

To see the sampling process, progress from left-to-right and top-to-bottom shows how the sampler denoises toward the final reconstruction (steps=100). From those images we could see the smooth transitions along the full reconstruction trajectory of PIFM.

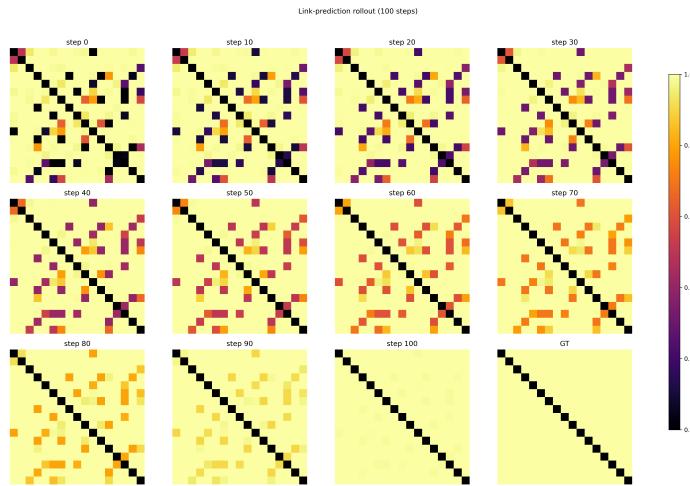


Figure 33: Visualization of IMDB 50% drop rate reconstruction. (Graph 1)

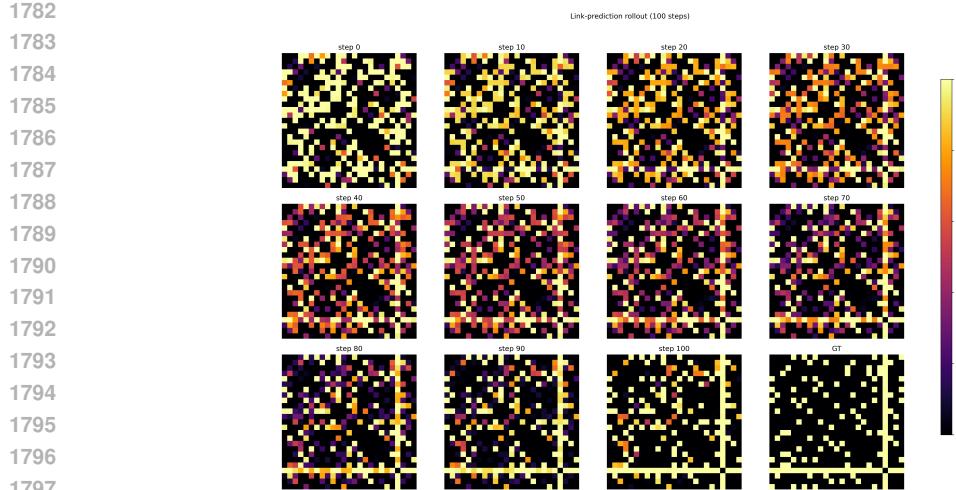


Figure 34: Visualization of IMDB 50% drop rate reconstruction. (Graph 2)

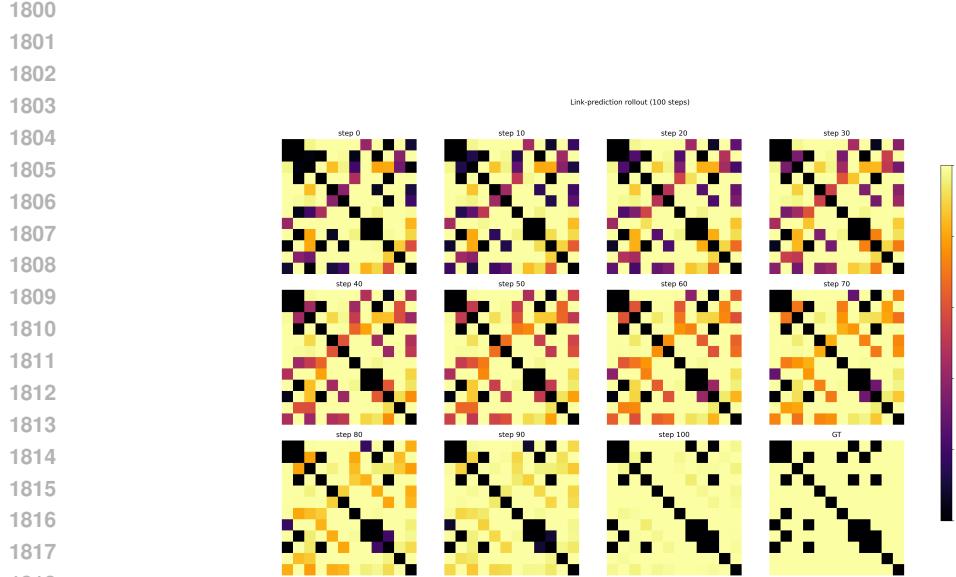


Figure 35: Visualization of IMDB 50% drop rate reconstruction. (Graph 3)

## F.7 TRAINING COST

All the models are trained on NVIDIA A100-SXM4-80GB GPUs. Fitting the Node2Vec prior takes roughly 1 to 1.5 hours depending on the datasets. Once the prior is fixed, the flow model itself has essentially the same training loop and cost as when using a Gaussian prior: end-to-end training of the flow model itself takes about 2 hours on ENZYMEs and IMDB-B, and about 4 to 5 hours on PROTEINS. We did not observe a noticeable difference in training behavior between the Gaussian and prior-informed variants, and the model behaves similarly under different edge-drop rates.

## F.8 TRANSFERABILITY

We have experimented the 10% and 50%-drop-rate IMDB-B checkpoints for PIFM and used on PROTEINS and ENZYMEs dataset with the same drop rates, and the best results and the results at the end of the 100 sampling steps of each run are as shown below.

1836 Table 7: Transferability of IMDB-B PIFM checkpoints (with graphSAGE priors) to PROTEINS  
 1837 and ENZYMEs. We use the 10% and 50%-drop-rate checkpoints trained on IMDB-B and evaluate  
 1838 them on the corresponding 10% and 50% drop-rate settings of the target datasets. For each setting,  
 1839 we report the best value over 100 sampling steps and the value at the final step ( $t = 100$ ). Metrics  
 1840 are Average Precision (AP), AUC, False Negative Rate (FNR), and False Positive Rate (FPR). All  
 1841 metrics are reported in percent.

1842

Source checkpoint	Target dataset	AP $\uparrow$	AUC $\uparrow$	FNR $\downarrow$	FPR $\downarrow$
<i>Structural priors (no flow)</i>					
PROTEINS (10% drop)	ENZYMEs (10% drop)	41.28	73.70	13.49	60.59
IMDB-B (10% drop)	PROTEINS (10% drop)	46.36	74.58	11.00	63.50
ENZYMEs (10% drop)	PROTEINS (10% drop)	46.36	74.58	11.00	63.50
PROTEINS (50% drop)	ENZYMEs (50% drop)	23.08	57.72	40.02	52.16
IMDB-B (50% drop)	PROTEINS (50% drop)	27.71	53.99	32.16	66.86
ENZYMEs (50% drop)	PROTEINS (50% drop)	27.71	53.99	32.16	66.86
<i>PIFM</i>					
PROTEINS (10% drop)	ENZYMEs (10% drop)	43.87	76.96	71.96	4.24
IMDB-B (10% drop)	PROTEINS (10% drop)	46.79	75.53	47.07	11.62
ENZYMEs (10% drop)	PROTEINS (10% drop)	49.63	76.60	59.02	6.63
PROTEINS (50% drop)	ENZYMEs (50% drop)	25.68	61.88	95.37	1.70
IMDB-B (50% drop)	PROTEINS (50% drop)	24.88	59.02	61.02	24.32
ENZYMEs (50% drop)	PROTEINS (50% drop)	26.81	55.80	86.13	9.97

1850

1851

1852

1853

1854

1855

1856

1857

1858

1859

From the table we can see that the model degrades the predictions a lot at lower (10%) drop rates, making the results incomparable when comparing to results in Table 1. It can be seen that with more steps integrated, at the final step  $t = 100$  the metrics are much worse than the best result over all sampling steps or the structural prior. Although at higher (50%) drop rates the best results are more comparable to the best results in Table 2, this is due to the graphs being very corrupted under 50% drop rate and hence the observed graphs are much more degraded than they used to be.

1860

1861

1862

1863

1864

1865

1866

## F.9 THRESHOLDING LEVELS

1867

1868

1869

1870

1871

1872

1873

1874

1875

1876

1877

1878

1879

1880

1881

1882

1883

1884

1885

1886

1887

1888

1889

We investigated the impact of alternative thresholding levels—specifically 0.3 and 0.7, beyond the standard 0.5—on the evaluation metrics, with results presented in Figures 36 and 37. We observed that adjusting the threshold yields a marginal increase in the Area Under the Curve (AUC). We hypothesize this is partly driven by the model’s inherent behavior of actively pushing output values towards the binary extremes (0 or 1). However, we noted a slightly larger, albeit still minimal, improvement for lower values of K, where the influence of the prior is greater and output values are typically further away from 0 and 1. Crucially, despite these dynamics, the use of an alternative threshold does not lead to a statistically significant improvement in overall performance.

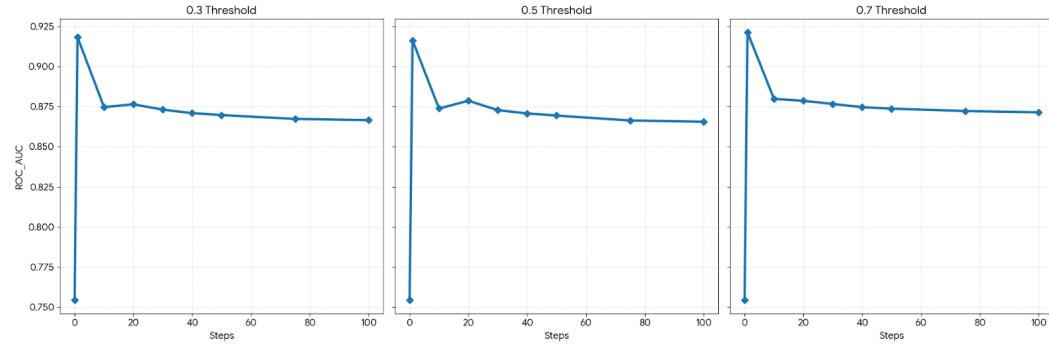


Figure 36: ROC-AUC vs number of steps sampled for IMDB dataset, link prediction task, 50% drop rate under 0.3, 0.5, and 0.7 thresholding levels.

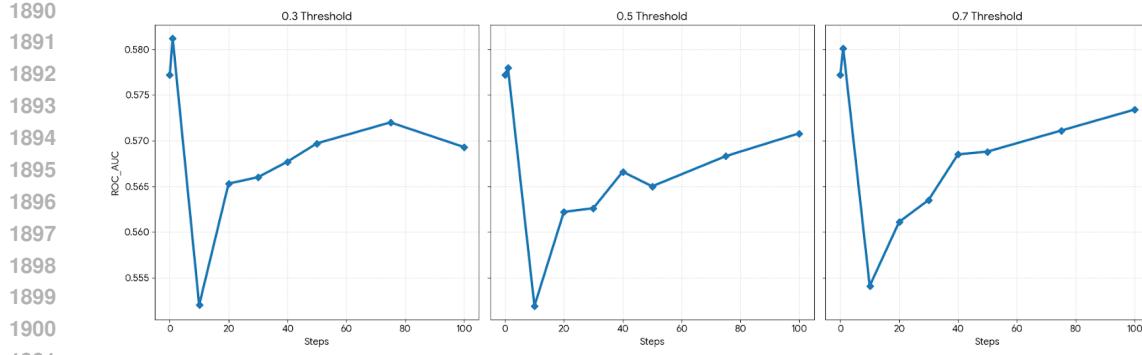


Figure 37: ROC-AUC vs number of steps sampled for ENZYMES dataset, link prediction task, 50% drop rate under 0.3, 0.5, and 0.7 thresholding levels.

## G LIMITATIONS

Despite these advantages, PIFM has apparent limitations. It's performance heavily depends on the quality of prior estimation, as shown by the gap between the node2vec-prior and graphon-prior versions of PIFM. Moreover, graphons may not be the most suitable prior in practice: they are fundamentally limit objects defined for limits of dense graphs, which restricts their applicability to sparse real-world networks. Graphons also does not capture dependencies between edges beyond what can be explained through the latent coordinates. Additionally, the current formulation is restricted to undirected and unweighted graphs, and the training overhead is higher than one-shot priors.

Looking forward, promising directions include extending PIFM to incorporate node and edge attributes for richer graph inference tasks, scaling the method to larger and more complex real-world networks, and enhancing the graphon prior by learning a dictionary of graphons from which the model can adaptively select during sampling. Such a design would provide more faithful prior initialization for datasets containing diverse graph types.