SCALABLE HIERARCHICAL GRAPH GENERATION VIA SOFT COMMUNITY STRUCTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating large attributed graphs requires reproducing the topology, generating attributes jointly with the structure, and remaining scalable. Many real-world graphs exist as a single large graph, so a generative model has to generalize from the one graph it is fit on, without independent samples. We present SCHEMA, which recursively decomposes a reference graph into a hierarchy of soft communities, assigning each node a membership distribution. Generation is then split into three stages, each trained independently: (1) synthesizing node attributes conditioned on soft memberships, (2) generating intra-community edges from local structural context, and (3) modeling inter-community connections over bridge nodes whose membership mass is distributed across several communities. No stage forms the full adjacency matrix, and each stage operates on a subgraph bounded by the community size. We also introduce an evaluation protocol that covers structural fidelity, memorization, downstream utility, and scalability. On four real-world attributed graphs, SCHEMA recovers the balance between local and long-range structure more closely than any other model that generates attributes, while reproducing only a small fraction of the reference edges. It retains the downstream accuracy of the reference graph without raising it artificially above that level. Baselines that match its structural fidelity memorize the reference, while those with higher downstream accuracy either exceed the reference accuracy or fail to complete on the larger graphs. We measure scalability on six additional graphs with up to 10 million nodes.

1 INTRODUCTION

Access to real graph data is often restricted by privacy regulations, licensing, or data collection costs, which creates a need for synthetic graphs. Graph generation methods fall into two categories, distinguished by the available data and the generation objective. The first category targets collections of relatively small graphs, where a model learns a distribution over independent graph instances and samples new graphs from it, as in molecular generation. The second category targets a single large graph, such as a social, citation, or transaction network, which may contain millions of nodes. In this setting, the model does not sample from a distribution over graph instances; it generates a graph that preserves the structural and attribute characteristics of the reference while generalizing beyond the specific edges and node identities observed during training.

Existing methods for the single-graph setting face three limitations. The first is scalability: autoregressive methods decode edges sequentially, incurring a cost that typically scales quadratically with the number of nodes (You et al., 2018), whereas methods that operate on full adjacency representations or require repeated passes over the entire graph become memory-limited as the graph grows. The second limitation concerns the coexistence of multiple structural scales. Real-world graphs typically contain dense local communities, sparse inter-community connections, and global topological dependencies (Fortunato, 2010), and generative fidelity depends on preserving all three at once. A single flat decoder has to fit these different densities with one set of parameters, so fitting one density can come at the cost of the others. The third is node features. Several existing methods generate only the topology and omit node features (Bojchevski et al., 2018; Rendsburg et al., 2020), which restricts their use on attributed graphs.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Evaluation in this setting is itself underdeveloped. Distributional metrics such as Maximum Mean Discrepancy (MMD) (Gretton et al., 2012) require samples from a reference distribution and do not generalize to a setting with a single reference graph. Existing benchmarks, therefore, often report narrow fidelity metrics and leave open the question of whether a generated graph merely memorizes the reference or remains useful for downstream tasks.

In this paper, we present SCHEMA (**S**oft **C**ommunity **H**ierarchies for **E**fficient **M**assive **A**ttributed graph generation), which recursively decomposes the reference graph into a hierarchy of communities. The decomposition assigns soft labels: each node carries a distribution over community memberships rather than a single label. Nodes whose membership mass is distributed over several communities are called *bridge nodes*, and the generated graph routes its inter-community edges through them. Generation is factorized into three stages, each operating at the community level. The intra-community and inter-community edge generators specialize in different structural densities: dense structure within a community and sparse structure between communities, respectively. Node features and edges are conditioned on the same soft memberships, so the attributes and the connectivity are tied to the same decomposition.

The main contributions of this work are summarized as follows:

- **Hierarchical factorization.** We formulate attributed graph generation as a hierarchical factorization over recursively constructed soft communities, decomposed into node feature generation, intra-community edge generation, and inter-community edge generation. Each stage operates on a subgraph bounded by the community size and does not form the full adjacency matrix.
- **Soft communities and bridge nodes.** We introduce a learning-free soft-community representation obtained by propagating community memberships over the graph, and an inter-community edge generator that weights candidate pairs by the connectivity observed between communities, so that bridge nodes carry the inter-community structure that hard partitioning removes.
- **Evaluation protocol.** We introduce an evaluation protocol that covers structural fidelity, memorization and generalization, downstream utility, and scalability.
- **Empirical study.** We evaluate SCHEMA against nine baselines on attributed graphs ranging from a few thousand to 10 million nodes. Among the models that generate attributes, it is the only one that is competitive on structure, keeps memorization low, preserves downstream accuracy without artificially raising it above the reference graph, and completes on every graph.

2 RELATED WORK

Generation over Graph Collections. Most existing graph generation methods are designed for the multi-graph setting, where models are trained on collections of relatively small graphs and generate new samples from the learned distribution, using autoregressive (You et al., 2018; Dai et al., 2020; Liao et al., 2019), variational (Simonovsky & Komodakis, 2018), adversarial (De Cao & Kipf, 2018), or diffusion-based (Vignac et al., 2023; Huang et al., 2022a; Chen et al., 2023) mechanisms. Even the methods that address scalability within this setting are typically demonstrated on graphs with only a few thousand nodes.

Generation from a Single Graph. Classical models generate graphs from a small number of parameters fit to the reference, such as edge-independent (Erdős & Rényi, 1960), small-world (Watts & Strogatz, 1998), and preferential attachment (Barabási & Albert, 1999) models. The configuration model (Molloy & Reed, 1995) reproduces the degree sequence exactly by construction, at the cost of all higher-order structure, while DC-SBM (Karrer & Newman, 2011) fits community-level connectivity but remains constrained by its block-level parametric form. GAE and VGAE (Kipf & Welling, 2016) were proposed for link prediction rather than generation, but are commonly used as generative baselines by sampling edges from the decoder applied to the learned embeddings. NetGAN (Bojchevski et al., 2018) models graph generation as learning a distribution over biased random walks with a generative adversarial network, and reconstructs the edge set from sampled walks. Building upon NetGAN, CELL (Rendsburg et al., 2020) reformulates this random walk objective as a low-rank matrix approximation problem, improving training speed. None of these methods generates node features.

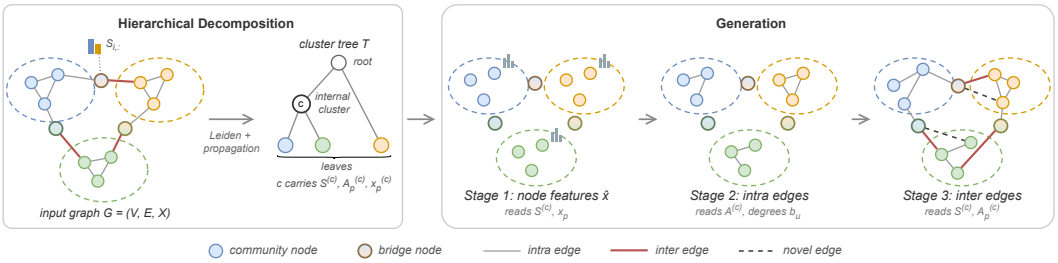


Figure 1: Overview of SCHEMA. The reference graph is recursively decomposed into soft communities. Generation then proceeds in three stages: node features, intra-community edges, and inter-community edges through the bridge nodes.

Among the methods that generate attributes alongside structure, GraphMaker (Li et al., 2024) employs discrete diffusion in two variants that differ in whether attributes and edges are denoised jointly or asynchronously, with edge mini-batching for scalability. CGT (Yoon et al., 2023) models per-node computation graphs with a Transformer and does not generate a graph directly. GenCAT (Maekawa et al., 2023) takes an algorithmic approach, reproducing the joint distribution of node classes, attributes, and topology via user-specified characteristic matrices that explicitly control the class-attribute and class-topology relationships. SynGen (Darabi et al., 2025) generates structure with a generalized stochastic Kronecker model and features with a separate model, and then maps features onto the generated structure with a gradient-boosting aligner intended to preserve feature-structure correlation.

Hierarchical and Subgraph-Based Generation. Hierarchical and subgraph-based approaches decompose graphs into smaller units to control the size of each generation subproblem, either by constructing an explicit hierarchy or by sampling subgraphs from the reference graph. In the multi-graph domain, HiGen (Karami, 2024) constructs coarsening hierarchies, while HiCGG (Razavi et al., 2026) trains community-level generators integrated via graph encoders. In the single-graph setting, SaGess (Limnios et al., 2024) partitions the graph into manageable subgraphs and generates their edges with a sampling procedure built on DiGress (Vignac et al., 2023). LGSG (Tuna & Soares, 2025) extends SaGess with learned embeddings in place of node identifiers, and aggregates the generated subgraphs through embedding distance, not shared identifiers. SANGEA (Lemaire et al., 2024) partitions an attributed graph into subcommunities with the Louvain algorithm, trains independent generative models for each partition, and links them to preserve topology and features.

SCHEMA differs from these methods in three ways. It decomposes the graph recursively, not into a flat set of partitions. It assigns each node a distribution over communities instead of a single label. It generates connectivity between communities at a dedicated stage, whereas the other methods link independently generated partitions.

Evaluation of Graph Generation. Evaluation differs between the two settings. In the multi-graph setting, structural fidelity is traditionally assessed using MMD (Gretton et al., 2012) over graph statistics (You et al., 2018), whose sensitivity to kernel choice is well documented (O'Bray et al., 2022; Thompson et al., 2022). In the single-graph setting, evaluation relies on direct statistical comparisons, class homophily metrics (Lim et al., 2021; Platonov et al., 2023), and downstream utility via synthetic-to-real transferability (Li et al., 2024). A separate line of work establishes memorization as a third dimension of evaluation. Chanpuriya et al. (2021) show that edge-independent models cannot reproduce the triangle density of real graphs without high overlap with the reference, and Chanpuriya et al. (2024) extend this to a hierarchy of edge, node, and fully dependent models.

3 METHODOLOGY

SCHEMA consists of a hierarchical decomposition module and a three-stage generator. Figure 1 illustrates the overall architecture.

3.1 Preliminaries and Problem Statement

An attributed graph is $G = (V, E, \mathbf{X})$ with a node set V of N nodes, an edge set E, and a feature matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ whose i-th row holds the features of node i. The objective is to generate a new graph $\hat{G} = (V, \hat{E}, \hat{\mathbf{X}})$ that reflects the characteristics of G. Appendix A.7 extends this to generated graphs of any size.

An effective generative model must satisfy three requirements. **Structural fidelity** requires that the structural properties of $\hat{G}$, such as its degree distribution and clustering patterns, match those of G. **Joint attribute generation** requires that attributes and edges are conditioned on the same representation and not sampled independently. **Generalization** requires that the model generates new graphs and does not reproduce G. For example, a naive baseline that inserts or removes a few random edges in G satisfies the first requirement while failing the third.

Scope and assumptions. Three properties of the formulation determine what SCHEMA generates and what it omits. First, the generated graph is defined over the node set of the reference graph, preserving node identities. This has two consequences. Node labels are inherited through those identities, so SCHEMA does not model the label distribution, and the shared node set allows memorization measures such as edge overlap. Second, inter-community edges are generated only between bridge nodes, assuming these edges attach to nodes near community boundaries, thereby keeping the candidate set small and allowing this stage to scale. Third, both edge stages score each node pair independently, given only the two nodes and their cluster context, and do not condition on edges that have already been sampled.

3.2 Hierarchical Soft-Community Decomposition

Leaf partition and cluster tree. We partition G once into leaf communities with the Leiden algorithm (Traag et al., 2019) under the modularity objective. Each community is then contracted into a supernode, and the weight between two supernodes is the number of reference edges between the corresponding communities. The cluster tree T is built top down over this fixed set of supernodes: its root is the complete set, and Leiden is applied recursively to partition each set into smaller sets. A set holding more than b supernodes is split again by the same procedure. A set holding at most b supernodes terminates the recursion, and each of its supernodes becomes a leaf child of the current cluster. When Leiden cannot split a set, either because its supernodes share no edges or because it returns the set unpartitioned, the set is divided into groups with at most b supernodes. Every leaf of T is one of the communities found in the first partition. Because the splits follow modularity rather than a fixed branching factor, the tree can be unbalanced.

Soft assignments via propagation. Nodes differ in how strongly they belong to their community. Some lie in its core, and others at the border between two or more communities. Assigning a single label cannot express this difference. We therefore extend the hard labels to soft assignments $\mathbf{S}^{(c)}$, one matrix per internal cluster c, where each row is a distribution over the K_c child clusters of c. The row of a node at the core is almost one-hot, and the row of a node at a border carries mass from more than one child cluster. Let $\mathbf{S}_0$ be the one-hot indicator of the child cluster whose subtree contains each node, and let $\mathbf{A}^{(c)}$ be the adjacency matrix of the subgraph induced by the nodes under c. At the root, this is the adjacency matrix of G itself. The memberships are updated by iterative propagation with row normalization at each step,

$$\mathbf{S}_{t+1} = \mathrm{rownorm}\big(\alpha\, \tilde{\mathbf{A}}\, \mathbf{S}_t + (1 - \alpha)\, \mathbf{S}_0\big), \tag{1}$$

for $t = 0, \ldots, \tau - 1$, where $\tilde{\mathbf{A}}$ is the symmetrized, self-looped, and degree-normalized form of $\mathbf{A}^{(c)}$, and the product $\tilde{\mathbf{A}}\mathbf{S}_t$ is computed over the edge list, so no dense matrix is formed. Here, τ is the number of propagation steps, and the coefficient $\alpha \in (0, 1)$ controls how far the membership mass spreads. The final matrix $\mathbf{S}_\tau$ defines $\mathbf{S}^{(c)}$. Equation (1) is a row-normalized, truncated form of the personalized propagation scheme of Gasteiger et al. (2019). The propagation has no learned parameters, so the decomposition is computed once per graph and reused across training runs.

Bridge nodes and pooled representations. A node whose membership mass in a second child cluster exceeds a threshold is a bridge node. The threshold and the number k of candidates selected per cluster pair are hyperparameters, so the candidate set remains bounded, independent of community size. Each internal cluster also stores two pooled matrices that summarize its child clusters, using

the soft assignment as a weight matrix,

$$\mathbf{x}_{\mathrm{p}}^{(c)} = \left(\mathbf{S}^{(c)}\right)^{\top}\mathbf{X}^{(c)} \in \mathbb{R}^{K_c \times F}, \qquad \mathbf{A}_{\mathrm{p}}^{(c)} = \left(\mathbf{S}^{(c)}\right)^{\top}\mathbf{A}^{(c)}\mathbf{S}^{(c)} \in \mathbb{R}^{K_c \times K_c}. \tag{2}$$

Each row of $\mathbf{x}_{\mathrm{p}}^{(c)}$ is a membership-weighted sum over the nodes of one child cluster. The diagonal entries of $\mathbf{A}_{\mathrm{p}}^{(c)}$ hold the edge mass within each child cluster, and the off-diagonal entries hold the edge mass between pairs of child clusters.

Edge assignment. An edge with both endpoints in the same leaf community is an intra-community edge, and an edge whose endpoints lie in different leaf communities is an inter-community edge, owned by the lowest common ancestor of the two communities in T. Every edge falls into exactly one of these two categories, which are modeled by separate stages.

3.3 Stage 1: Node Feature Generation

Stage 1 generates node features for each leaf, conditioned on the soft assignment and the corresponding row of $\mathbf{x}_{\mathrm{p}}^{(c)}$, where c is the parent cluster of the leaf. We instantiate the generator as a Transformer (Vaswani et al., 2017) over the member nodes of the leaf, where each node attends to the other members and cross-attends to the pooled row. The features of all members of a leaf are therefore generated jointly so that the model can capture correlations among them. We train with a masked reconstruction loss that handles leaves of varying sizes. The reconstruction term follows the feature type of the reference graph, detected once from $\mathbf{X}$.

3.4 Stage 2: Intra-Community Edge Generation

Stage 2 generates the edges within each leaf, which we formulate as an edge prediction problem inside the leaf. Every pair of member nodes is a candidate, the reference edges of the leaf are positives, and the remaining pairs are negatives. A single model is trained over all leaves, which are batched as a disjoint union so that no candidate pair crosses a leaf boundary.

We model each leaf with a variational graph autoencoder (Kipf & Welling, 2016). The encoder first makes the embeddings degree-aware: the feature embedding of a node is scaled and shifted by learned functions of its log-transformed reference degree, following the FiLM conditioning of Perez et al. (2018), so that each embedding carries the degree budget its node is expected to fill. A graph convolutional network (Kipf & Welling, 2017) then reads these embeddings together with the leaf adjacency and produces a latent embedding $\mathbf{z}_i$ per node, which therefore reflects the features of a node's neighbors and not only its own. The decoder scores a pair by a bilinear form $s_{ij} = \mathbf{z}_i^{\top}\mathbf{W}\mathbf{z}_j$, and the edge probability is $\sigma(s_{ij})$, where σ is the logistic sigmoid. Training minimizes binary cross-entropy over the observed edges of the leaf and sampled non-edges, with a KL term that keeps the latent embeddings close to a standard normal prior. Both are subsampled, with at most P edges per leaf per epoch and νP non-edges, where ν is the negative sampling ratio, so the per-step cost is set by P and not by the leaf size.

At generation, there is no reference adjacency to encode, so the latent embeddings are drawn from the prior, and the decoder scores pairs from them together with the degree conditioning. Edges are selected from the score matrix under a per-node degree budget.

3.5 Stage 3: Inter-Community Edge Generation

Stage 3 generates inter-community edges. Scoring every possible pair between two communities is quadratic in community size, so we restrict generation to a candidate pool of fixed size k. Candidates are selected from the soft assignments, and bridge nodes are natural candidates by definition.

As in Stage 2, we formulate Stage 3 as an edge prediction task. A pair is scored by a bilinear form over the two node embeddings. We add to this score a cluster-affinity term read from the hierarchy, which measures how often the source's cluster links to the clusters of the destination, weighted by the source's membership in its own cluster. The term is computed from $\mathbf{S}^{(c)}$ and $\mathbf{A}_{\mathrm{p}}^{(c)}$ alone. It involves no learned parameters, so the model starts from the connectivity already recorded in the hierarchy and only has to adjust it.

3.6 Training and Generation

The three stages are trained independently, with no joint objective and no gradient flow between them. This keeps memory bounded by the largest community and avoids balancing a shared loss across three objectives with different scales.

Generation traverses the tree once. Stage 1 generates the features of every leaf, Stage 2 generates the intra-community edges of each leaf, and Stage 3 visits each internal cluster and generates the edges between its children. Each stage draws its own noise. The noise scale controls how far a stage departs from what it was trained on, and it is the only source of variation between two graphs generated from the same model.

No stage forms a dense $N \times N$ adjacency matrix. The decomposition is linear in the number of edges of each subgraph, and both edge stages train on sampled pairs. One quadratic term remains at generation: Stage 2 scores every pair inside a leaf, which grows with the square of the leaf size. Since the modularity objective places no bound on the leaf size, scalability and memory usage depend on the graph itself. Generation holds at most one batch of clusters in memory at a time.

4 Experiments and Results

The evaluation covers four aspects. Structural fidelity and memorization test the two requirements stated in Section 3.1. Downstream utility tests whether the generated graph serves as a substitute for the reference. Scalability measures the cost of generation as the graph grows.

4.1 Experimental Setup

Datasets. We evaluate on ten attributed graphs in total. Four are used for the full evaluation: Citeseer (Yang et al., 2016) and Cora ML (Bojchevski & Günnemann, 2018) are citation networks, and Amazon Computers and Amazon Photo (Shchur et al., 2018) are co-purchase networks. The other six datasets are larger and are used in the main text only to measure scalability: Flickr and Yelp (Zeng et al., 2020), Reddit (Hamilton et al., 2017), DGraphFin (Huang et al., 2022b), OGBN Products (Hu et al., 2020), and IGB Medium (Khatua et al., 2023). Dataset details are given in Appendix A.1.

Baselines. We compare against four groups of methods. The classical models include the configuration model (Molloy & Reed, 1995) and DC-SBM (Karrer & Newman, 2011). DC-SBM blocks are obtained from a Leiden partition of the reference. The topology-only deep models generate structure without attributes: NetGAN (Bojchevski et al., 2018) and CELL (Rendsburg et al., 2020). The attributed models generate structure and node features: GenCAT (Maekawa et al., 2023), GraphMaker (Li et al., 2024), and SynGen (Darabi et al., 2025). For GraphMaker, we report the asynchronous variant, which the authors identify as the stronger of the two. The subgraph-based models decompose the reference before generating: SaGess (Limnios et al., 2024) and LGSG (Tuna & Soares, 2025). Further baselines, implementation details, and deviations from the published settings are given in Appendix A.2.

Implementation Details. We implement SCHEMA in Python with PyTorch (Paszke et al., 2019) and PyTorch Geometric (Fey & Lenssen, 2019). All experiments are conducted on a single machine with an NVIDIA GeForce RTX 3090 GPU (24 GB of memory) and 32 GB of system RAM. The code is available at `https://anonymous.4open.science/r/schema-submission`. For each experiment, we set a 12-hour time limit. Methods that run out of memory (OOM) or time out (TO) are reported as such in the corresponding tables.

Reporting. Unless otherwise stated, each reported number is the mean and standard deviation over 10 independently generated graphs. Due to space limitations, the main text reports the key findings of each experiment. The Appendix provides extended versions of every experiment with per-dataset discussion, a full evaluation on Flickr, and an ablation study (Appendix A.8).

Table 1: Structural fidelity. Results are given as the mean over 10 generated graphs with standard deviation.

Model	Citeseer				Cora ML			
	Deg. W1	Tri. Rat.	Assort. Diff.	Inter/Intra	Deg. W1	Tri. Rat.	Assort. Diff.	Inter/Intra
Config	$0.007_{\pm 0.002}$	$0.026_{\pm 0.006}$	$0.059_{\pm 0.011}$	$5.207_{\pm 0.051}$	$0.047_{\pm 0.004}$	$0.097_{\pm 0.008}$	$0.048_{\pm 0.005}$	$6.653_{\pm 0.151}$
DC-SBM	$0.445_{\pm 0.016}$	$0.467_{\pm 0.039}$	$0.061_{\pm 0.023}$	$0.993_{\pm 0.099}$	$0.466_{\pm 0.032}$	$0.382_{\pm 0.011}$	$0.052_{\pm 0.005}$	$0.969_{\pm 0.054}$
GenCAT	$0.027_{\pm 0.004}$	$0.093_{\pm 0.011}$	$0.105_{\pm 0.006}$	$3.022_{\pm 0.194}$	$0.183_{\pm 0.007}$	$0.245_{\pm 0.014}$	$0.010_{\pm 0.007}$	$1.826_{\pm 0.177}$
NetGAN	$0.612_{\pm 0.131}$	$1.024_{\pm 0.084}$	$0.063_{\pm 0.027}$	$1.472_{\pm 0.549}$	$0.875_{\pm 0.034}$	$0.489_{\pm 0.060}$	$0.012_{\pm 0.011}$	$1.810_{\pm 0.425}$
CELL	$0.744_{\pm 0.012}$	$0.088_{\pm 0.008}$	$0.073_{\pm 0.015}$	$3.370_{\pm 0.071}$	$0.961_{\pm 0.030}$	$0.403_{\pm 0.016}$	$0.025_{\pm 0.004}$	$2.436_{\pm 0.108}$
GraphMaker	$9.129_{\pm 0.348}$	$43.349_{\pm 3.082}$	$0.208_{\pm 0.048}$	$0.849_{\pm 0.342}$	$6.560_{\pm 0.480}$	$6.689_{\pm 0.977}$	$0.102_{\pm 0.054}$	$0.296_{\pm 0.031}$
SaGess	$0.138_{\pm 0.008}$	$1.045_{\pm 0.020}$	$0.008_{\pm 0.003}$	$0.943_{\pm 0.069}$	$0.485_{\pm 0.018}$	$1.085_{\pm 0.008}$	$0.009_{\pm 0.002}$	$1.107_{\pm 0.054}$
LGSG	$0.953_{\pm 0.046}$	$0.986_{\pm 0.076}$	$0.184_{\pm 0.010}$	$2.296_{\pm 0.212}$	$3.993_{\pm 0.065}$	$0.258_{\pm 0.016}$	$0.317_{\pm 0.015}$	$2.196_{\pm 0.084}$
SynGen	$0.646_{\pm 0.083}$	$0.059_{\pm 0.037}$	$0.082_{\pm 0.046}$	$6.669_{\pm 1.204}$	$1.484_{\pm 0.064}$	$0.026_{\pm 0.009}$	$0.199_{\pm 0.071}$	$4.532_{\pm 0.548}$
SCHEMA	$0.247_{\pm 0.015}$	$0.363_{\pm 0.021}$	$0.022_{\pm 0.005}$	$0.984_{\pm 0.090}$	$1.075_{\pm 0.024}$	$0.275_{\pm 0.019}$	$0.002_{\pm 0.002}$	$1.030_{\pm 0.083}$
Model	Amazon Photo				Amazon Computers			
	Deg. W1	Tri. Rat.	Assort. Diff.	Inter/Intra	Deg. W1	Tri. Rat.	Assort. Diff.	Inter/Intra
Config	$0.582_{\pm 0.015}$	$0.137_{\pm 0.001}$	$0.010_{\pm 0.001}$	$20.407_{\pm 1.599}$	$0.797_{\pm 0.006}$	$0.199_{\pm 0.001}$	$0.013_{\pm 0.000}$	$9.857_{\pm 0.865}$
DC-SBM	$2.797_{\pm 0.082}$	$0.375_{\pm 0.003}$	$0.103_{\pm 0.002}$	$1.128_{\pm 0.009}$	$2.797_{\pm 0.090}$	$0.388_{\pm 0.002}$	$0.075_{\pm 0.001}$	$0.796_{\pm 0.020}$
GenCAT	$1.233_{\pm 0.046}$	$0.299_{\pm 0.008}$	$0.039_{\pm 0.003}$	$1.692_{\pm 0.261}$	$1.567_{\pm 0.044}$	$0.330_{\pm 0.009}$	$0.044_{\pm 0.003}$	$0.924_{\pm 0.102}$
NetGAN	$26.537_{\pm 0.033}$	$0.877_{\pm 0.003}$	$0.680_{\pm 0.002}$	$17.509_{\pm 1.158}$	OOM			
CELL	$2.584_{\pm 0.055}$	$0.626_{\pm 0.004}$	$0.015_{\pm 0.002}$	$1.105_{\pm 0.020}$	$1.299_{\pm 0.023}$	$0.649_{\pm 0.003}$	$0.008_{\pm 0.001}$	$1.044_{\pm 0.066}$
GraphMaker	$4.961_{\pm 0.448}$	$0.845_{\pm 0.129}$	$0.160_{\pm 0.032}$	$0.629_{\pm 0.039}$	TO			
SaGess	TO				TO			
LGSG	$28.963_{\pm 0.044}$	$0.010_{\pm 0.000}$	$0.288_{\pm 0.006}$	$3.970_{\pm 0.086}$	TO			
SynGen	$7.069_{\pm 1.869}$	$0.088_{\pm 0.050}$	$0.137_{\pm 0.070}$	$4.997_{\pm 1.184}$	$9.535_{\pm 2.205}$	$0.063_{\pm 0.027}$	$0.248_{\pm 0.092}$	$1.650_{\pm 0.471}$
SCHEMA	$5.417_{\pm 0.069}$	$0.294_{\pm 0.003}$	$0.024_{\pm 0.001}$	$1.159_{\pm 0.026}$	$11.084_{\pm 0.023}$	$0.549_{\pm 0.002}$	$0.092_{\pm 0.001}$	$0.945_{\pm 0.085}$

4.2 STRUCTURAL FIDELITY

We report four metrics that cover three structural scales: the Wasserstein-1 degree distance and the assortativity difference at the local scale, the triangle count ratio at the higher-order scale, and the ratio of inter-community to intra-community edges at the mesoscopic scale. Ratios are computed relative to the reference and are ideal at one, whereas differences and distances are ideal at zero.

Structural fidelity results are reported in Table 1. No model matches the reference at every scale, and the models that lead on one metric are weak on another. Among the learning-based baselines, those that match the reference most closely in any single column either reproduce a large part of it or fail to complete on the larger graphs. SCHEMA completes on all four datasets. Among the models that generate attributes, it is the closest to the reference Inter/Intra ratio across all datasets. However, the cost is a larger degree distance, so the gain is at the community scale and not at the level of individual nodes.

The triangle ratio should be read together with the memorization results. SCHEMA scores node pairs independently in both edge stages, and Chanpuriya et al. (2021) show that models of this form cannot reach the triangle density of a real graph without reproducing a large fraction of its edges.

4.3 MEMORIZATION

A generated graph can score well on every structural metric by reproducing the reference, so fidelity metrics alone cannot separate generalization from memorization. We therefore measure how much of the reference is copied and how much the generated samples differ from one another.

Edge overlap (EO) is the Jaccard similarity between the generated and reference edge sets, self-overlap (SO) is the mean Jaccard similarity between pairs of generated graphs, and neighborhood memorization (NM) is the mean per-node Jaccard similarity between generated and reference neighbor sets. These measures require a node correspondence between the reference and generated graphs. The configuration model and GenCAT are aligned by degree sequence, and the remaining models without a correspondence are excluded. Results are in Table 2.

The models with a high triangle ratio in Table 1 show the highest overlap in Table 2. Low overlap alone is not evidence of generalization either. The configuration model and GenCAT sit near zero on all three measures, and both distort the community structure.

Table 2: Memorization. Each result is the mean over 10 generated graphs. NetGAN runs on the largest connected component, not on the full graph.

Model	Citeseer			Cora ML			Amazon Photo			Amazon Computers		
	EO	SO	NM	EO	SO	NM	EO	SO	NM	EO	SO	NM
Config	0.003	0.002	0.001	0.007	0.008	0.003	0.017	0.016	0.007	0.019	0.017	0.006
DC-SBM	0.084	0.063	0.140	0.052	0.046	0.050	0.075	0.066	0.046	0.056	0.051	0.031
GenCAT	0.002	0.004	0.001	0.005	0.014	0.002	0.015	0.028	0.007	0.015	0.029	0.005
NetGAN	0.228	0.142	0.208	0.174	0.094	0.151	0.027	0.801	0.025		OOM	
CELL	0.384	0.258	0.500	0.442	0.284	0.516	0.337	0.237	0.340	0.348	0.232	0.393
SaGess	0.910	0.847	0.904	0.815	0.735	0.795		TO			TO	
SCHEMA	0.102	0.108	0.254	0.038	0.066	0.068	0.040	0.068	0.035	0.032	0.111	0.024

The results show two memorization patterns. SaGess shows near-complete memorization, with high overlap and high NM, so it reconstructs the reference instead of generating from it. CELL and NetGAN show partial memorization, with moderate overlap and NM. Both stop training once the generated graph reaches a preset overlap with the reference, so the amount they copy is set by this stopping criterion.

SCHEMA reproduces a small fraction of the reference edges, and its overlap with the reference is at or below its overlap with its own samples on every dataset. The one high value, NM on Citeseer, comes from nodes in leaf communities of at most two nodes, where the reference neighborhood is the only one the model can produce (Appendix A.4.3).

4.4 DOWNSTREAM UTILITY

A generated graph is useful in practice only if a model trained on it transfers to the reference. We train and test node classifiers on the reference and generated graphs in three settings. TRTR trains and tests on the reference, which serves as the expected ceiling, since a generated graph cannot carry information that the reference does not have. TSTR trains on the generated graph and tests on the reference, and we report its accuracy divided by the TRTR accuracy as utility. TSTS trains and tests on the generated graph, and its difference from TRTR indicates whether the generated task is easier or harder than the reference task. A generator can raise utility by making the task easier, and TSTS detects this. We also compare how TSTS and TRTR rank architectures, using Spearman and Pearson correlations.

For this task, we report only the models that generate node features. Results are listed in Table 3. On every dataset where GraphMaker completes, its generated task is easier than the reference, and on Citeseer, its utility exceeds 1. Further analysis in Appendix A.5 shows that this difference largely comes from inflated homophily.

SCHEMA retains most of the reference accuracy on every dataset, and its TSTS-TRTR is negative throughout, so its utility does not come from simplifying the task. GenCAT and SynGen also complete on all four datasets but with lower utility. For SCHEMA, the Pearson correlation is high on every dataset, so the accuracy on the generated graph tracks that of the reference. The Spearman correlation is moderate, so the order of the architectures is only partly preserved.

SynGen retains about a quarter of the reference accuracy across all datasets, which is consistent with its design. It generates structure and features with separate models and joins them with a node-wise aligner, so the dependence between labels and connectivity is not modeled.

4.5 SCALABILITY

We measure wall-clock time and peak memory per stage, and report training and generation time separately. Peak memory covers both host memory and GPU memory. For graphs that are not fully connected, we run both the full graph and its largest connected component. The batch size is 16. Figure 2 shows the results.

Table 3: Downstream utility. Utility: TSTR to TRTR ratio, where 1 is ideal.

Model	Citeseer				Cora ML			
	Utility	TSTS-TRTR	Spearman ρ	Pearson r	Utility	TSTS-TRTR	Spearman ρ	Pearson r
GenCAT	$0.752_{\pm 0.298}$	$0.017_{\pm 0.119}$	$0.261_{\pm 0.118}$	$0.682_{\pm 0.087}$	$0.686_{\pm 0.241}$	$-0.050_{\pm 0.150}$	$0.527_{\pm 0.114}$	$0.750_{\pm 0.070}$
GraphMaker	$1.055_{\pm 0.034}$	$0.135_{\pm 0.025}$	$0.673_{\pm 0.081}$	$0.854_{\pm 0.071}$	$0.976_{\pm 0.039}$	$0.086_{\pm 0.009}$	$0.755_{\pm 0.083}$	$0.974_{\pm 0.061}$
SynGen	$0.251_{\pm 0.064}$	$-0.503_{\pm 0.009}$	$0.050_{\pm 0.221}$	$0.176_{\pm 0.278}$	$0.263_{\pm 0.092}$	$-0.835_{\pm 0.017}$	$0.019_{\pm 0.268}$	$0.115_{\pm 0.043}$
SCHEMA	$0.861_{\pm 0.076}$	$-0.132_{\pm 0.037}$	$0.538_{\pm 0.175}$	$0.853_{\pm 0.095}$	$0.816_{\pm 0.062}$	$-0.125_{\pm 0.029}$	$0.492_{\pm 0.107}$	$0.971_{\pm 0.019}$
Model	Amazon Photo				Amazon Computers			
	Utility	TSTS-TRTR	Spearman ρ	Pearson r	Utility	TSTS-TRTR	Spearman ρ	Pearson r
GenCAT	$0.470_{\pm 0.117}$	$-0.218_{\pm 0.177}$	$0.506_{\pm 0.099}$	$0.663_{\pm 0.051}$	$0.422_{\pm 0.065}$	$-0.494_{\pm 0.063}$	$0.268_{\pm 0.275}$	$0.234_{\pm 0.229}$
GraphMaker	$0.975_{\pm 0.011}$	$0.016_{\pm 0.007}$	$0.891_{\pm 0.030}$	$0.993_{\pm 0.009}$	TO			
SynGen	$0.220_{\pm 0.079}$	$-0.901_{\pm 0.028}$	$-0.013_{\pm 0.278}$	$-0.251_{\pm 0.316}$	$0.248_{\pm 0.145}$	$-0.870_{\pm 0.024}$	$0.002_{\pm 0.235}$	$-0.242_{\pm 0.372}$
SCHEMA	$0.805_{\pm 0.100}$	$-0.111_{\pm 0.008}$	$0.443_{\pm 0.127}$	$0.994_{\pm 0.008}$	$0.771_{\pm 0.088}$	$-0.137_{\pm 0.015}$	$0.408_{\pm 0.142}$	$0.974_{\pm 0.028}$

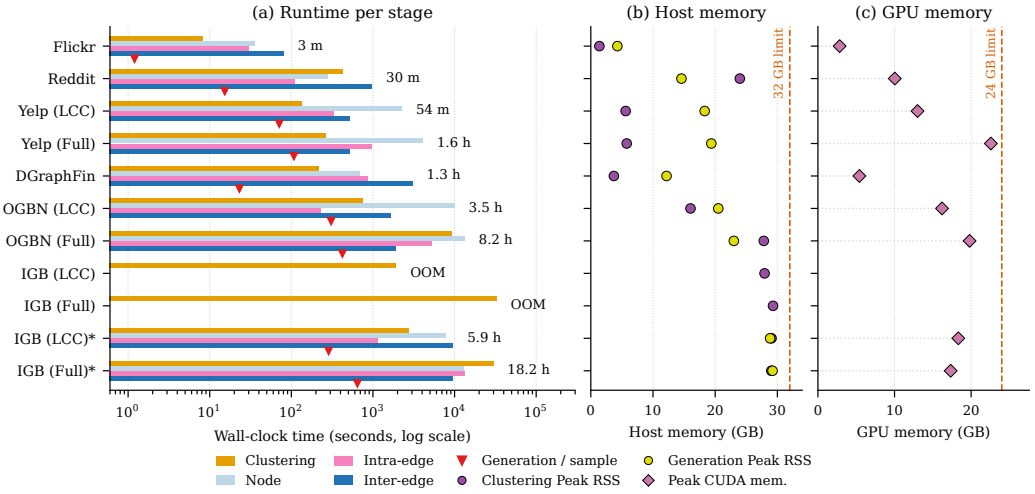


Figure 2: Runtime and peak memory usage of SCHEMA per stage on the six scalability datasets.

The memory and runtime of clustering and training grow with the number of connected components. Stage 1 dominates training time on larger graphs, whereas on graphs with many inter-community edges, Stage 3 takes the largest share. Generation is inexpensive compared to training.

GPU memory remains bounded by the leaf size and the number of pairs scored per step, as stated in Section 3.6. Since leaf sizes are determined by the modularity objective, peak GPU memory scales with the largest leaf, not by the graph size, so the largest graph does not necessarily use the most GPU memory. IGB Medium does not complete generation under the default settings, since its largest community under the modularity objective has more than 1 million nodes and does not fit in memory. With the leaf size capped at 100,000 nodes and a batch size of 8, generation completes, but the cap changes the partition, so this run does not use the same configuration as the other graphs.

5 CONCLUSION

SCHEMA generates a large attributed graph from a single reference by decomposing it into a hierarchy of soft communities and training three generation stages independently. Among the models that generate attributes, SCHEMA is competitive on structural fidelity, keeps memorization low, and retains most of the reference downstream accuracy without making the task easier. Each competitor either memorizes, makes the downstream task easier, fails on larger graphs, or fails to generate features. Generation scales to a graph with 10 million nodes on a single GPU, although this graph requires capping the leaf size. Because both edge stages score node pairs independently, the triangle density that SCHEMA reaches at low memorization is limited, and the gain at the community scale comes with a larger degree distance.

AI USE STATEMENT

We used generative AI tools to improve the manuscript's readability and language, and to aid with formatting and grammatical consistency. Generative AI tools were not used to generate synthetic datasets, formulate or prove mathematical claims, propose research hypotheses, or implement the proposed method.

All AI-assisted content was reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide an anonymous repository containing the source code in the main text. The repository includes a hyperparameter configuration and dedicated installation instructions.

All datasets that are used are already publicly available.

We also provide generated graphs in the repository.

REFERENCES

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286 (5439):509–512, 1999.

Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.

Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian embedding of graphs: Unsupervised inductive learning via ranking. In *International Conference on Learning Representations*, 2018.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. NetGAN: Generating graphs via random walks. In *International Conference on Machine Learning*, pp. 610–619, 2018.

Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. In *SIAM International Conference on Data Mining*, pp. 442–446, 2004.

Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos Tsourakakis. On the power of edge independent graph models. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24418–24429, 2021.

Sudhanshu Chanpuriya, Cameron N. Musco, Konstantinos Sotiropoulos, and Charalampos Tsourakakis. On the role of edge dependency in graph generative models. In *International Conference on Machine Learning*, pp. 6325–6345, 2024.

Xiaohui Chen, Jiaxing He, Xu Han, and Li-Ping Liu. Efficient and degree-guided graph generation via discrete diffusion modeling. In *International Conference on Machine Learning*, pp. 4585–4610, 2023.

Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable deep generative modeling for sparse graphs. In *International Conference on Machine Learning*, pp. 2302–2312, 2020.

Sajad Darabi, Piotr Bigaj, Dawid Majchrowski, Artur Kasymov, Pawel Morkisz, and Alex Fit-Florea. A framework for large-scale synthetic graph dataset generation. *IEEE Transactions on Neural Networks and Learning Systems*, 36(8):14258–14268, 2025.

Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences*, 5(1):17–61, 1960.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *International Conference on Learning Representations*, 2019.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Python in Science Conference*, pp. 11–15, 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 22118–22133, 2020.

Han Huang, Leilei Sun, Bowen Du, Yanjie Fu, and Weifeng Lv. GraphGDP: Generative diffusion processes for permutation invariant graph generation. In *IEEE International Conference on Data Mining*, pp. 201–210, 2022a.

Xuanwen Huang, Yang Yang, Yang Wang, Chunping Wang, Zhisheng Zhang, Jiarong Xu, Lei Chen, and Michalis Vazirgiannis. DGraph: A large-scale financial dataset for graph anomaly detection. In *Advances in Neural Information Processing Systems*, volume 35, pp. 22765–22777, 2022b.

Mahdi Karami. HiGen: Hierarchical graph generative networks. In *International Conference on Learning Representations*, 2024.

Brian Karrer and Mark E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. IGB: Addressing the gaps in labeling, features, heterogeneity, and size of public graph datasets for deep learning research. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

Thomas N. Kipf and Max Welling. Variational graph auto-encoders. In *NeurIPS Bayesian Deep Learning Workshop*, 2016.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Valentin Lemaire, Youssef Achenchabe, Lucas Ody, Houssem Eddine Souid, Gianmarco Aversano, Nicolas Posocco, and Sabri Skhiri. SANGEA: Scalable and attributed network generation. In *Asian Conference on Machine Learning*, pp. 678–693, 2024.

Mufei Li, Eleonora Kreačić, Vamsi K. Potluru, and Pan Li. GraphMaker: Can diffusion models generate large attributed graphs? *Transactions on Machine Learning Research*, 2024.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K. Duvenaud, Raquel Urtasun, and Richard Zemel. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. In *Advances in Neural Information Processing Systems*, volume 34, pp. 20887–20902, 2021.

Stratis Limnios, Praveen Selvaraj, Mihai Cucuringu, Carsten Maple, Gesine Reinert, and Andrew Elliott. SaGess: A sampling graph denoising diffusion model for scalable graph generation. In *European Conference on Artificial Intelligence*, pp. 2950–2957, 2024.

Seiji Maekawa, Yuya Sasaki, George Fletcher, and Makoto Onizuka. GenCAT: Generating attributed graphs with controlled relationships between classes, attributes, and topology. *Information Systems*, 115:102195, 2023.

Joel C. Miller and Aric Hagberg. Efficient generation of networks with given expected degrees. In *International Workshop on Algorithms and Models for the Web-Graph*, pp. 115–126, 2011.

Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6(2-3):161–180, 1995.

Leslie O'Bray, Max Horn, Bastian Rieck, and Karsten Borgwardt. Evaluation metrics for graph generative models: Problems, pitfalls, and practical solutions. In *International Conference on Learning Representations*, 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. Characterizing graph datasets for node classification: Homophily-heterophily dichotomy and beyond. In *Advances in Neural Information Processing Systems*, volume 36, pp. 523–548, 2023.

Masoomeh Sadat Razavi, Abdolreza Mirzaei, and Mehran Safayani. Hierarchical community-based graph generation model for improving structural diversity. *Pattern Recognition*, 172:112320, 2026.

Luca Rendsburg, Holger Heidrich, and Ulrike von Luxburg. NetGAN without GAN: From random walks to low-rank approximations. In *International Conference on Machine Learning*, pp. 8073–8082, 2020.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. In *NeurIPS Workshop on Relational Representation Learning*, 2018.

Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422, 2018.

Rylee Thompson, Boris Knyazev, Elahe Ghalebi, Jungtaek Kim, and Graham W. Taylor. On evaluation metrics for graph generative models. In *International Conference on Learning Representations*, 2022.

Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

Rodrigo Tuna and Carlos Soares. Generating large semi-synthetic graphs of any size, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pp. 6861–6871, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning*, pp. 40–48, 2016.

Minji Yoon, Yue Wu, John Palowitch, Bryan Perozzi, and Russ Salakhutdinov. Graph generative model for benchmarking graph neural networks. In *International Conference on Machine Learning*, pp. 40175–40198, 2023.

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pp. 5708–5717, 2018.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

A Appendix

A.1 Datasets

The four evaluation graphs (Citeseer, Cora ML, Amazon Photo, and Amazon Computers) have the same format. Nodes carry binary bag-of-words vectors, edges are undirected, and every node has exactly one class label. In the two citation networks, nodes are papers, edges are citations, features indicate whether a dictionary word occurs in the paper, and classes are research areas. Amazon Photo and Amazon Computers are co-purchase networks in which nodes are products, edges connect frequently co-purchased products, features are derived from product reviews, and product categories serve as labels. Citeseer and Cora ML have higher-dimensional sparse features, while the Amazon graphs are denser in both attributes and degree.

The six scalability graphs are larger and carry continuous features, which are dense on every graph except Flickr. Flickr is an image network in which nodes are images, edges connect images that share properties such as a common gallery or location, features are built from words that describe the images, and node labels are image categories. Reddit is a social network where nodes are posts, an edge means the same user commented on both posts, features are built from posts and entries, and labels are subreddits. Yelp is a social network in which nodes are users and edges are friendships, features are embeddings of each user's reviews, and labels are the types of businesses the user reviewed. It is the only multi-label graph. OGBN Products is a co-purchase network, with features derived from product descriptions and product categories as labels. DGraphFin is a financial network in which nodes are users of a lending platform, and an edge means one user listed another as an emergency contact. Its features are anonymized profile fields, and its four label values cover two foreground classes, normal and fraudulent, plus two background classes for unlabeled users. IGB Medium is a citation network in which features are language model embeddings of paper abstracts.

Detailed dataset statistics are given in Table 4. Graphs with many connected components are harder for the hierarchy: modularity gains nothing by merging nodes across components, so each component yields at least one community. Citeseer is the extreme case in relative terms: only 63.7% of its nodes lie in the largest component, and 16.4% lie in components of at most two nodes.

A.2 Baselines

A.2.1 Classical models

We implement six random graph models, two of which appear in the main text, and two graph autoencoders adapted for generation. None of them generates node attributes. The random graph models define a distribution over edge sets only, and the autoencoders read node features as encoder input but decode only edges. We therefore use them only in the structural and memorization experiments, as structural baselines and as a check in the evaluation.

Table 4: Dataset statistics. # Con. Comp. is the number of connected components. LCC is the fraction of nodes in the largest connected component. F is the number of node attributes. Feat. den. is the fraction of non-zero entries in the attribute matrix. C is the number of classes.

Dataset	N	$\|E\|$	Avg. deg.	# Con. Comp.	LCC	F	Feat. den.	C
Citeseer	3,327	4,552	2.74	438	63.7%	3,703	0.009	6
Cora ML	2,995	8,158	5.45	61	93.8%	2,879	0.018	7
Amazon Photo	7,650	119,081	31.13	136	97.9%	745	0.347	8
Amazon Computers	13,752	245,861	35.76	314	97.3%	767	0.348	10
Flickr	89,250	449,878	10.08	1	100%	500	0.464	7
Reddit	232,965	57,307,946	491.99	1	100%	602	1.000	41
Yelp	716,847	6,618,986	18.47	6,288	98.2%	300	1.000	100
OGBN Products	2,449,029	61,859,012	50.52	52,658	97.4%	100	0.990	47
DGraphFin	3,700,550	3,997,260	1.08	1	100%	17	0.942	4
IGB Medium	10,000,000	59,997,454	12.00	111,011	98.8%	128†	1.000	-

† The original graph has 1,024 features, of which we keep the first 128 to fit in memory.

We generate the Erdős–Rényi (ER), Watts–Strogatz (WS), Barabási–Albert (BA), Chung–Lu (CL), and configuration model (Config) with the corresponding NetworkX generators (Hagberg et al., 2008), two of which use the linear-time samplers of Batagelj & Brandes (2005) for ER $G(n, p)$ and Miller & Hagberg (2011) for CL instead of iterating over all pairs. The degree-corrected stochastic block model (DC-SBM) is implemented in NumPy (Harris et al., 2020). Every model is fitted to the same reference graph as the other baselines, and all of them operate on the simple undirected edge list with self-loops and duplicate edges removed. Each model generates 10 samples per dataset with independent seeds, and the same 10 seeds are used across models and datasets.

The models differ in how much information they receive from the reference. In order of increasing information: ER $G(n, p)$ receives the node count and the edge density $p = M/\binom{N}{2}$, where $M = |E|$ is the reference edge count. ER $G(n, m)$ receives the node count and the exact edge count. WS receives the node count and the mean degree k, rounded to the nearest integer, and its rewiring probability is not fitted but fixed at $\beta = 0.1$. BA receives the node count and the per-node attachment count m, which we set to the reference edge count divided by the node count, rounded to the nearest integer.

CL receives the full reference degree sequence and treats it as a sequence of expected degrees, so the generated degrees match it only in expectation. Config receives the same sequence and matches it exactly, but samples a multi-graph. Reducing parallel edges to a single edge and removing self-loops lowers the realized degrees.

Our DC-SBM implementation follows Karrer & Newman (2011). The model is a Poisson multi-graph in which the expected number of edges between nodes i and j is $\theta_i\theta_j\omega_{b_ib_j}$, where b_i is the block of node i, ω_{rs} is the edge budget between blocks r and s, and $\theta_i = d_i / \sum_{j:b_j=b_i} d_j$ is a within-block degree parameter, with d_i the degree of node i. Under the maximum-likelihood fit for known blocks, ω_{rs} is the observed edge count m_{rs} between blocks r and s, and twice the internal edge count when $r = s$.

The θ_i sum to one within each block, so the per-pair Poisson means add up to ω_{rs}: the total number of edges between two blocks is itself Poisson with that mean. Conditional on that total, the two endpoints of each edge are independent draws proportional to θ within their blocks. We therefore sample one Poisson count per occupied block pair and then place the endpoints, which costs $O(M)$ instead of the $O(N^2)$ of a per-pair formulation. This sampler is exact, not an approximation. The sampled multi-graph is projected to a simple undirected graph in the same way as the configuration model.

The blocks come from a Leiden partition (Traag et al., 2019) of the reference at resolution 1.0, not from the node labels, so the model receives no label information. The same Leiden setting defines the reference communities in the mesoscopic metrics, so DC-SBM's mesoscopic scores measure how well the Poisson sampler preserves the given partition, not whether it can find community structure. SCHEMA builds its leaves from the same partition. DC-SBM therefore shares SCHEMA's communities and replaces the learned edge model and the hierarchy with a Poisson block model.

Generated edge counts. Four of the classical models systematically miss the reference edge count, and ER $G(n, p)$ and CL match it only in expectation. BA and WS quantize the target density to an integer parameter, so their realized edge counts are $m(N - m)$ and $\lfloor k/2 \rfloor N$. Config loses edges when parallel edges and self-loops are removed. DC-SBM loses edges when the Poisson multi-graph is projected onto a simple graph. We do not correct any of these, and the Deg. Rat. column of the structural fidelity tables shows the resulting deviation.

A.2.2 GRAPH AUTOENCODERS

We implement the graph autoencoder (GAE) and the variational graph autoencoder (VGAE) (Kipf & Welling, 2016) as two-layer graph convolutional encoders with an inner-product decoder, trained on positive edges and an equal number of negative edges. Both read node features as encoder input and decode only edges. We change the standard setup in two ways.

First, we divide the KL term by the number of nodes. Without this normalization, the KL term dominates the objective and the posterior collapses to the prior, so the latents carry almost no information about the graph.

Second, the decoder scores every node pair, so it needs a rule for turning scores into edges. We give it N and the reference edge count M, as for ER $G(n, m)$, and keep the M highest-scoring pairs after a Gumbel perturbation at temperature 0.2. The edge count is then fixed, and the node that receives those edges is determined by the decoder.

Since GAE defines no prior, its only valid generation setting is reconstruction from the encoded reference. VGAE supports both reconstruction and prior sampling, but we report only the prior version.

A.2.3 PUBLISHED IMPLEMENTATIONS

For the remaining baselines, we use the authors' released code with the hyperparameters reported in their papers or source code. The notes below list where we deviate from the published setting, or where the released code forces a choice, because each affects how the corresponding results should be interpreted.

- **CELL and NetGAN.** Both stop training when the generated graph reaches a preset edge overlap with the reference, so their edge overlap is fixed in advance. We use the published stopping criteria on every dataset. Their memorization results, therefore, reflect the stopping rule rather than generalization.
- **NetGAN.** NetGAN works on the largest connected component, so its generated graphs cover only the nodes in that component. All NetGAN results are computed against the largest connected component of the reference. Size-dependent quantities such as the triangle count and the Jaccard denominators are therefore taken over a smaller graph than for the other rows.
- **GenCAT.** GenCAT takes a class mixing matrix and class-wise attribute statistics as input, so it receives more information from the reference than most baselines. It also takes the reference degree sequence as input, which keeps its degree distance low. Its generated features are continuous and nearly uniformly dense, so feature-marginal comparisons against the binary references are not meaningful.
- **GraphMaker.** The model is label-conditional, so like GenCAT, it receives label information. The released implementation stores a dense $N \times N$ adjacency matrix, which limits the size of the graph it can handle. The main text reports the asynchronous variant, and the appendix reports both.
- **SaGess.** We run SaGess as published, with no deviations.
- **SynGen.** The model is unnamed in the paper; we use the name from the repository. We use the KDE feature generator, the tool's default, which has the highest feature correlation in the authors' ablation study. Two properties of the method directly affect the generated graphs. First, SynGen's R-MAT (Chakrabarti et al., 2004) structural generator uses $\lceil \log_2 N \rceil$ bits per node identifier, so the node count rises to $2^{\lceil \log_2 N \rceil}$ minus any unused identifiers. The edge count is matched exactly, so the generated graphs are sparser than the reference. Second, structure and features are generated independently and joined by a node-wise aligner, which is trained to predict a node's feature row from single-node structural summaries. The aligner, therefore, cannot represent homophily.
- **LGSG.** The model generates many small subgraphs with node embeddings and assembles them into one graph. The authors' threshold-based assembler produces graphs with varying sizes across datasets, so we use the node assembler, which targets the reference node count. The target is not always met, and the edge count is not controlled. Each of the 10 generated graphs is assembled from a disjoint slice of the generated subgraphs.
- **SANGEA.** No implementation is publicly available, so we discuss it in the related work and do not include it in the experiments.
- **CGT.** The model learns and generates a separate computation graph per node and does not generate a single graph as the other models do. None of the structural or attribute metrics is defined on its output, so we exclude it.

Table 5: Citeseer structural fidelity results.

Model	Deg. Rat.	Deg. W1	Tri. Rat.	Clus. C. Rat.	Assort. Diff.	Mod. Diff.	Inter/Intra	CPL Rat.
ER $G(n,p)$	$0.992_{\pm 0.013}$	$0.834_{\pm 0.018}$	$0.004_{\pm 0.001}$	$0.008_{\pm 0.003}$	$0.045_{\pm 0.022}$	$0.194_{\pm 0.004}$	$6.404_{\pm 0.137}$	$0.852_{\pm 0.010}$
ER $G(n,m)$	$1.000_{\pm 0.000}$	$0.846_{\pm 0.016}$	$0.004_{\pm 0.002}$	$0.008_{\pm 0.005}$	$0.041_{\pm 0.007}$	$0.195_{\pm 0.003}$	$6.409_{\pm 0.102}$	$0.851_{\pm 0.004}$
WS	$0.731_{\pm 0.000}$	$1.415_{\pm 0.012}$	$0.000_{\pm 0.000}$	$0.001_{\pm 0.002}$	$0.091_{\pm 0.021}$	$0.072_{\pm 0.001}$	$0.298_{\pm 0.012}$	$14.745_{\pm 5.090}$
BA	$0.731_{\pm 0.000}$	$0.852_{\pm 0.026}$	$0.000_{\pm 0.000}$	$0.000_{\pm 0.000}$	$0.137_{\pm 0.015}$	$0.065_{\pm 0.001}$	$0.344_{\pm 0.017}$	$0.886_{\pm 0.053}$
CL	$0.997_{\pm 0.010}$	$0.441_{\pm 0.018}$	$0.040_{\pm 0.008}$	$0.035_{\pm 0.006}$	$0.057_{\pm 0.006}$	$0.285_{\pm 0.004}$	$7.939_{\pm 0.153}$	$0.577_{\pm 0.007}$
Config	$0.997_{\pm 0.001}$	$0.007_{\pm 0.002}$	$0.026_{\pm 0.006}$	$0.027_{\pm 0.006}$	$0.059_{\pm 0.011}$	$0.185_{\pm 0.002}$	$5.207_{\pm 0.051}$	$0.654_{\pm 0.004}$
DC-SBM	$0.854_{\pm 0.012}$	$0.445_{\pm 0.016}$	$0.467_{\pm 0.039}$	$0.576_{\pm 0.042}$	$0.061_{\pm 0.023}$	$0.008_{\pm 0.004}$	$0.993_{\pm 0.099}$	$0.815_{\pm 0.032}$
GenCAT	$0.990_{\pm 0.001}$	$0.027_{\pm 0.004}$	$0.093_{\pm 0.011}$	$0.097_{\pm 0.011}$	$0.105_{\pm 0.006}$	$0.124_{\pm 0.007}$	$3.022_{\pm 0.194}$	$0.737_{\pm 0.009}$
GAE	$1.000_{\pm 0.000}$	$2.156_{\pm 0.007}$	$11.630_{\pm 0.062}$	$2.357_{\pm 0.015}$	$0.181_{\pm 0.003}$	$0.075_{\pm 0.002}$	$0.909_{\pm 0.058}$	$0.740_{\pm 0.041}$
VGAE	$1.000_{\pm 0.000}$	$0.969_{\pm 0.019}$	$0.528_{\pm 0.018}$	$0.407_{\pm 0.014}$	$0.179_{\pm 0.012}$	$0.305_{\pm 0.006}$	$9.932_{\pm 0.304}$	$0.532_{\pm 0.005}$
NetGAN	$0.850_{\pm 0.000}$	$0.612_{\pm 0.131}$	$1.024_{\pm 0.084}$	$1.279_{\pm 0.276}$	$0.063_{\pm 0.027}$	$0.038_{\pm 0.039}$	$1.472_{\pm 0.549}$	$0.679_{\pm 0.111}$
CELL	$1.000_{\pm 0.000}$	$0.744_{\pm 0.012}$	$0.088_{\pm 0.008}$	$0.184_{\pm 0.018}$	$0.073_{\pm 0.015}$	$0.118_{\pm 0.003}$	$3.370_{\pm 0.071}$	$0.946_{\pm 0.006}$
GraphMaker S	$0.864_{\pm 0.036}$	$0.464_{\pm 0.047}$	$0.036_{\pm 0.020}$	$0.072_{\pm 0.031}$	$0.180_{\pm 0.071}$	$0.146_{\pm 0.011}$	$3.717_{\pm 0.370}$	$0.830_{\pm 0.021}$
GraphMaker A	$4.284_{\pm 0.125}$	$9.129_{\pm 0.348}$	$43.349_{\pm 3.082}$	$2.167_{\pm 0.071}$	$0.208_{\pm 0.048}$	$0.131_{\pm 0.005}$	$0.849_{\pm 0.342}$	$0.570_{\pm 0.010}$
SaGess	$0.995_{\pm 0.001}$	$0.138_{\pm 0.008}$	$1.045_{\pm 0.020}$	$0.986_{\pm 0.015}$	$0.008_{\pm 0.003}$	$0.006_{\pm 0.001}$	$0.943_{\pm 0.069}$	$0.901_{\pm 0.014}$
LGSG	$1.113_{\pm 0.015}$	$0.953_{\pm 0.046}$	$0.986_{\pm 0.076}$	$0.044_{\pm 0.005}$	$0.184_{\pm 0.010}$	$0.114_{\pm 0.013}$	$2.296_{\pm 0.212}$	$0.450_{\pm 0.018}$
SynGen	$0.813_{\pm 0.001}$	$0.646_{\pm 0.083}$	$0.059_{\pm 0.037}$	$0.065_{\pm 0.033}$	$0.082_{\pm 0.046}$	$0.247_{\pm 0.048}$	$6.669_{\pm 1.204}$	$0.668_{\pm 0.096}$
SCHEMA	$0.996_{\pm 0.005}$	$0.247_{\pm 0.015}$	$0.363_{\pm 0.021}$	$0.368_{\pm 0.020}$	$0.022_{\pm 0.005}$	$0.018_{\pm 0.001}$	$0.984_{\pm 0.090}$	$0.884_{\pm 0.015}$

Table 6: Cora ML structural fidelity results.

Model	Deg. Rat.	Deg. W1	Tri. Rat.	Clus. C. Rat.	Assort. Diff.	Mod. Diff.	Inter/Intra	CPL Rat.
ER $G(n,p)$	$0.994_{\pm 0.010}$	$2.448_{\pm 0.031}$	$0.004_{\pm 0.001}$	$0.014_{\pm 0.003}$	$0.067_{\pm 0.010}$	$0.335_{\pm 0.003}$	$7.241_{\pm 0.086}$	$0.934_{\pm 0.005}$
ER $G(n,m)$	$1.000_{\pm 0.000}$	$2.459_{\pm 0.026}$	$0.005_{\pm 0.001}$	$0.015_{\pm 0.002}$	$0.074_{\pm 0.012}$	$0.336_{\pm 0.001}$	$7.245_{\pm 0.106}$	$0.932_{\pm 0.001}$
WS	$0.734_{\pm 0.000}$	$3.278_{\pm 0.011}$	$0.413_{\pm 0.009}$	$3.076_{\pm 0.072}$	$0.049_{\pm 0.019}$	$0.099_{\pm 0.005}$	$0.800_{\pm 0.043}$	$2.008_{\pm 0.045}$
BA	$1.100_{\pm 0.000}$	$1.078_{\pm 0.052}$	$0.068_{\pm 0.006}$	$0.067_{\pm 0.005}$	$0.018_{\pm 0.008}$	$0.364_{\pm 0.002}$	$7.278_{\pm 0.206}$	$0.734_{\pm 0.002}$
CL	$1.002_{\pm 0.015}$	$0.451_{\pm 0.064}$	$0.147_{\pm 0.009}$	$0.141_{\pm 0.005}$	$0.053_{\pm 0.005}$	$0.379_{\pm 0.005}$	$7.431_{\pm 0.141}$	$0.750_{\pm 0.006}$
Config	$0.991_{\pm 0.001}$	$0.047_{\pm 0.004}$	$0.097_{\pm 0.008}$	$0.104_{\pm 0.009}$	$0.048_{\pm 0.005}$	$0.342_{\pm 0.002}$	$6.653_{\pm 0.151}$	$0.790_{\pm 0.003}$
DC-SBM	$0.925_{\pm 0.010}$	$0.466_{\pm 0.032}$	$0.382_{\pm 0.011}$	$0.524_{\pm 0.012}$	$0.052_{\pm 0.005}$	$0.004_{\pm 0.003}$	$0.969_{\pm 0.054}$	$0.910_{\pm 0.007}$
GenCAT	$0.966_{\pm 0.001}$	$0.183_{\pm 0.007}$	$0.245_{\pm 0.014}$	$0.349_{\pm 0.019}$	$0.010_{\pm 0.007}$	$0.130_{\pm 0.010}$	$1.826_{\pm 0.177}$	$0.911_{\pm 0.004}$
GAE	$1.000_{\pm 0.000}$	$3.361_{\pm 0.016}$	$6.098_{\pm 0.038}$	$2.092_{\pm 0.018}$	$0.220_{\pm 0.002}$	$0.015_{\pm 0.002}$	$0.517_{\pm 0.074}$	$0.828_{\pm 0.010}$
VGAE	$1.000_{\pm 0.000}$	$1.169_{\pm 0.040}$	$0.399_{\pm 0.010}$	$0.546_{\pm 0.010}$	$0.057_{\pm 0.016}$	$0.319_{\pm 0.002}$	$6.275_{\pm 0.140}$	$0.780_{\pm 0.003}$
NetGAN	$0.850_{\pm 0.000}$	$0.875_{\pm 0.034}$	$0.489_{\pm 0.060}$	$0.634_{\pm 0.118}$	$0.012_{\pm 0.011}$	$0.088_{\pm 0.037}$	$1.810_{\pm 0.425}$	$0.850_{\pm 0.049}$
CELL	$1.000_{\pm 0.000}$	$0.961_{\pm 0.030}$	$0.403_{\pm 0.016}$	$0.643_{\pm 0.024}$	$0.025_{\pm 0.004}$	$0.115_{\pm 0.005}$	$2.436_{\pm 0.108}$	$0.933_{\pm 0.004}$
GraphMaker S	$0.640_{\pm 0.027}$	$1.961_{\pm 0.145}$	$0.021_{\pm 0.006}$	$0.116_{\pm 0.022}$	$0.203_{\pm 0.062}$	$0.104_{\pm 0.006}$	$2.119_{\pm 0.111}$	$1.160_{\pm 0.031}$
GraphMaker A	$2.170_{\pm 0.089}$	$6.560_{\pm 0.480}$	$6.689_{\pm 0.977}$	$1.917_{\pm 0.131}$	$0.102_{\pm 0.054}$	$0.025_{\pm 0.005}$	$0.296_{\pm 0.031}$	$0.882_{\pm 0.007}$
SaGess	$0.994_{\pm 0.001}$	$0.485_{\pm 0.018}$	$1.085_{\pm 0.008}$	$0.983_{\pm 0.018}$	$0.009_{\pm 0.002}$	$0.017_{\pm 0.003}$	$1.107_{\pm 0.054}$	$0.939_{\pm 0.011}$
LGSG	$0.536_{\pm 0.009}$	$3.993_{\pm 0.065}$	$0.258_{\pm 0.016}$	$0.012_{\pm 0.001}$	$0.317_{\pm 0.015}$	$0.250_{\pm 0.017}$	$2.196_{\pm 0.084}$	$0.455_{\pm 0.008}$
SynGen	$0.731_{\pm 0.000}$	$1.484_{\pm 0.064}$	$0.026_{\pm 0.009}$	$0.077_{\pm 0.023}$	$0.199_{\pm 0.071}$	$0.235_{\pm 0.013}$	$4.532_{\pm 0.548}$	$1.061_{\pm 0.025}$
SCHEMA	$1.032_{\pm 0.004}$	$1.075_{\pm 0.024}$	$0.275_{\pm 0.019}$	$0.202_{\pm 0.013}$	$0.002_{\pm 0.002}$	$0.042_{\pm 0.003}$	$1.030_{\pm 0.083}$	$0.845_{\pm 0.011}$

A.3 FULL STRUCTURAL FIDELITY

Structural fidelity metrics follow three conventions. Ratios divide the value of a metric on the generated graph $\hat{G}$ by its value on the reference graph G. Differences are absolute, $|m(G) - m(\hat{G})|$ for a metric m, and distribution distances use the first-order Wasserstein distance (W_1).

At the local scale, we report the average degree ratio (Deg. Rat.), the W_1 degree distance (Deg. W1), and the assortativity difference (Assort. Diff.). The average degree ratio is used instead of the edge ratio because some models generate a different number of nodes. The W_1 degree distance compares the degree distributions of the generated and reference graphs. Degree assortativity is the correlation between the degrees at the two ends of an edge. It is positive when high-degree nodes connect to each other and negative when they connect to low-degree nodes.

At the higher-order scale, we report the triangle count ratio (Tri. Rat.) and the ratio of global clustering coefficients, i.e., transitivity (Clus. C. Rat.).

At the mesoscopic scale, we report the modularity difference (Mod. Diff.) and the ratio of inter-community to intra-community edge counts (Inter/Intra), $\frac{\hat{G}(\texttt{Inter/Intra})}{G(\texttt{Inter/Intra})}$. Both are computed from a Leiden partition of each graph, independently obtained for the reference and for each sample.

Modularity measures how strongly the graph separates into communities, and the Inter/Intra ratio measures how its edges split between and within communities.

At the global scale, we report the characteristic path length ratio (CPL Rat.), where the characteristic path length is the mean shortest-path distance over all reachable node pairs.

A good generator does not need a perfect score on every metric. Rewiring a few edges of the reference scores nearly perfectly on all of them, and it is not a generative model. Some models are also built toward particular metrics: a model conditioned on the reference degree sequence will score well on degree-based measures by construction. We therefore read the metrics as a profile of the generated graph, not as a single objective to maximize.

Structural fidelity results are given in Tables 5, 6, 7, 8, and 9, for the datasets Citeseer, Cora ML, Amazon Photo, Amazon Computers, and Flickr, respectively.

Among the classical models, the mesoscopic metrics separate DC-SBM from the rest. ER, CL, and Config reach Inter/Intra ratios between 2.4 and 20.5, and BA exceeds 2.4 on every graph except Citeseer. CL and Config match the degree distribution and still lose the community structure. DC-SBM is the only classical model that receives a community partition as input. A degree-based model, therefore, does not recover the mesoscopic scale on its own.

DC-SBM is the strongest classical baseline. Its Inter/Intra ratio ranges from 0.80 to 1.13 across all five datasets, and it has the lowest modularity difference on three of them. This is expected, since the model is given the Leiden partition of the reference directly and reproduces the community structure it was handed. SCHEMA starts from the same Leiden partition, so the comparison isolates the edge model: SCHEMA reaches a comparable Inter/Intra ratio on the four evaluation graphs with a learned edge model and also generates attributes.

Among the models that generate attributes, GenCAT has the lowest degree distance across all datasets, and it completes on all five. On Flickr, its degree distance is 0.020, and its assortativity difference is 0.002. Its Inter/Intra ratio is between 1.7 and 3.0 on four of the five datasets, so the edges it places correctly at the level of individual nodes are misdistributed between communities. GenCAT takes a class mixing matrix, class-wise attribute statistics, and the degree sequence from the reference as input, so part of what it reproduces is what it was given.

The autoencoder baselines are inconsistent across datasets. GAE produces triangle ratios of 11.6, 6.1, and 9.0 across three of the datasets and 579 on Flickr, and its degree ratio is exactly one by construction. On those four datasets, GAE matches the edge count and places those edges in dense clusters that the reference does not have. VGAE is closer to triangles but reaches Inter/Intra ratios of 9.9 on Citeseer and 13.0 on Amazon Photo.

Completion rates fall with scale. All models run on the two citation graphs. One fails on Amazon Photo, five fail on Amazon Computers, and six fail on Flickr, either due to memory exhaustion or to reaching the time limit. The models that fail either produce a full score matrix or adopt a diffusion process.

The random-walk models trade one scale against another. CELL matches the edge count exactly, and NetGAN is at 0.85 of it on every graph where it runs. NetGAN's triangle ratio on Citeseer (1.024) is among the closest to one, but the Inter/Intra ratios of the two models range from 1.04 to 17.5. NetGAN runs out of memory on Amazon Computers and CELL on Flickr.

The diffusion models follow two designs. GraphMaker runs a diffusion process over the whole graph and is inconsistent across its variants: the synchronous one is the better of the two on Citeseer, with a degree ratio of 0.864 versus 4.284, and the worse on Amazon Photo, at 5.178 versus 0.921. Neither variant completes on Amazon Computers or Flickr. SaGess and LGSG instead diffuse over sampled subgraphs and reassemble them. SaGess is among the closest to the reference in most columns where it runs, and the memorization results show that it does so by reproducing most of the reference edge set. LGSG reassembles by merging nodes with similar embeddings. Its degree ratios of 0.117 on Amazon Photo and 0.536 on Cora ML indicate that the merge discards a large part of the edge set, while on Citeseer it overshoots at 1.113.

SynGen differs from the other baselines in how it sizes the output. It matches the reference edge count but rounds up the node count to the next power of 2, so the same edges are spread over more

Table 7: Amazon Photo structural fidelity results.

Model	Deg. Rat.	Deg. W1	Tri. Rat.	Clus. C. Rat.	Assort. Diff.	Mod. Diff.	Inter/Intra	CPL Rat.
ER $G(n,p)$	$0.999_{\pm 0.003}$	$18.734_{\pm 0.055}$	$0.007_{\pm 0.000}$	$0.023_{\pm 0.000}$	$0.043_{\pm 0.003}$	$0.576_{\pm 0.002}$	$14.867_{\pm 4.432}$	$0.716_{\pm 0.000}$
ER $G(n,m)$	$1.000_{\pm 0.000}$	$18.750_{\pm 0.029}$	$0.007_{\pm 0.000}$	$0.023_{\pm 0.000}$	$0.045_{\pm 0.004}$	$0.576_{\pm 0.001}$	$15.704_{\pm 3.271}$	$0.716_{\pm 0.000}$
WS	$0.964_{\pm 0.000}$	$21.240_{\pm 0.014}$	$0.817_{\pm 0.002}$	$2.968_{\pm 0.008}$	$0.041_{\pm 0.003}$	$0.104_{\pm 0.001}$	$1.153_{\pm 0.008}$	$0.899_{\pm 0.001}$
BA	$1.026_{\pm 0.000}$	$7.820_{\pm 0.020}$	$0.067_{\pm 0.001}$	$0.099_{\pm 0.001}$	$0.034_{\pm 0.003}$	$0.586_{\pm 0.001}$	$19.438_{\pm 1.753}$	$0.688_{\pm 0.001}$
CL	$0.997_{\pm 0.003}$	$0.593_{\pm 0.052}$	$0.191_{\pm 0.003}$	$0.195_{\pm 0.002}$	$0.018_{\pm 0.001}$	$0.597_{\pm 0.001}$	$20.538_{\pm 2.191}$	$0.698_{\pm 0.001}$
Config	$0.981_{\pm 0.000}$	$0.582_{\pm 0.015}$	$0.137_{\pm 0.001}$	$0.157_{\pm 0.001}$	$0.010_{\pm 0.001}$	$0.594_{\pm 0.002}$	$20.407_{\pm 1.599}$	$0.704_{\pm 0.000}$
DC-SBM	$0.910_{\pm 0.003}$	$2.797_{\pm 0.082}$	$0.375_{\pm 0.003}$	$0.585_{\pm 0.003}$	$0.103_{\pm 0.002}$	$0.005_{\pm 0.001}$	$1.128_{\pm 0.009}$	$0.911_{\pm 0.005}$
GenCAT	$0.960_{\pm 0.001}$	$1.233_{\pm 0.046}$	$0.299_{\pm 0.008}$	$0.365_{\pm 0.009}$	$0.039_{\pm 0.003}$	$0.100_{\pm 0.006}$	$1.692_{\pm 0.261}$	$0.779_{\pm 0.002}$
GAE	$1.000_{\pm 0.000}$	$5.150_{\pm 0.044}$	$0.811_{\pm 0.005}$	$0.911_{\pm 0.003}$	$0.112_{\pm 0.001}$	$0.258_{\pm 0.004}$	$3.752_{\pm 0.246}$	$0.750_{\pm 0.001}$
VGAE	$1.000_{\pm 0.000}$	$5.006_{\pm 0.190}$	$0.249_{\pm 0.003}$	$0.436_{\pm 0.003}$	$0.085_{\pm 0.004}$	$0.482_{\pm 0.003}$	$13.042_{\pm 1.025}$	$0.740_{\pm 0.001}$
NetGAN	$0.850_{\pm 0.000}$	$26.537_{\pm 0.033}$	$0.877_{\pm 0.003}$	$0.042_{\pm 0.000}$	$0.680_{\pm 0.002}$	$0.615_{\pm 0.002}$	$17.509_{\pm 1.158}$	$0.561_{\pm 0.001}$
CELL	$1.000_{\pm 0.000}$	$2.584_{\pm 0.055}$	$0.626_{\pm 0.004}$	$0.840_{\pm 0.005}$	$0.015_{\pm 0.002}$	$0.009_{\pm 0.001}$	$1.105_{\pm 0.020}$	$0.842_{\pm 0.002}$
GraphMaker S	$5.178_{\pm 0.169}$	$130.322_{\pm 5.262}$	$25.411_{\pm 2.984}$	$1.519_{\pm 0.055}$	$0.636_{\pm 0.018}$	$0.158_{\pm 0.030}$	$0.274_{\pm 0.011}$	$0.661_{\pm 0.002}$
GraphMaker A	$0.921_{\pm 0.038}$	$4.961_{\pm 0.448}$	$0.845_{\pm 0.129}$	$1.230_{\pm 0.076}$	$0.160_{\pm 0.032}$	$0.009_{\pm 0.006}$	$0.629_{\pm 0.039}$	$0.936_{\pm 0.006}$
SaGess	TO					TO		
LGSG	$0.117_{\pm 0.001}$	$28.963_{\pm 0.044}$	$0.010_{\pm 0.000}$	$0.005_{\pm 0.000}$	$0.288_{\pm 0.006}$	$0.216_{\pm 0.006}$	$3.970_{\pm 0.086}$	$0.541_{\pm 0.002}$
SynGen	$0.934_{\pm 0.000}$	$7.069_{\pm 1.869}$	$0.088_{\pm 0.050}$	$0.181_{\pm 0.066}$	$0.137_{\pm 0.070}$	$0.335_{\pm 0.043}$	$4.997_{\pm 1.184}$	$0.762_{\pm 0.008}$
SCHEMA	$0.966_{\pm 0.002}$	$5.417_{\pm 0.069}$	$0.294_{\pm 0.003}$	$0.273_{\pm 0.003}$	$0.024_{\pm 0.001}$	$0.043_{\pm 0.003}$	$1.159_{\pm 0.026}$	$0.829_{\pm 0.003}$

nodes, and the degree ratio falls between 0.68 and 0.93. Its assortativity difference ranges from 0.08 to 0.25, among the highest of the models that complete on all five datasets.

On the two citation graphs and Amazon Photo, SCHEMA's degree ratio stays near one and its Inter/Intra ratio near the reference. Its assortativity difference on Cora ML is 0.002, the lowest in that table. Amazon Computers is its weakest result, with a degree ratio of 1.277, likely the per-node budget bounds the edges a node selects but not the edges it receives, so high-degree nodes gain edges. Across the four datasets, the triangle ratio ranges from 0.28 to 0.55, which is consistent with scoring node pairs independently (Section 4.2). The characteristic path length is between 0.83 and 0.98 of the reference, so the generated graphs are slightly more compact. Flickr is an exception to both and is discussed below.

On Flickr, SCHEMA uses a candidate pool of 500 nodes per cluster pair. Its degree distance of 2.287 is below that of every learned baseline that completes there. The Inter/Intra ratio, however, falls to 0.097. The pool covers only a small fraction of the nodes with inter-community edges in the reference, and an edge whose endpoint is outside the pool cannot be generated. Enlarging the pool raises the Inter/Intra ratio but distorts other metrics. At 4000 nodes, it reaches 0.414, while the degree distance rises to 10.772 and the assortativity difference to 0.181, because the additional edges concentrate on the nodes already in the pool. We therefore report the setting that preserves the degree distribution.

Two measurements reverse on Flickr. The triangle ratio is 4.579, and the characteristic path length ratio exceeds 1, whereas both are below 1 for the other four datasets. A candidate pool of 2000 nodes increases the triangle ratio to 122.9. The reference has $63{,}941$ triangles over $89{,}250$ nodes, against about 1.5 million over $13{,}752$ nodes for Amazon Computers, so a small absolute excess moves the ratio by a large factor. GAE shows the same sensitivity here, at 579, compared to 6 to 12 on Citeseer, Cora ML, and Amazon Computers.

A.4 MEMORIZATION EVALUATION

In the multi-graph setting, a generated graph is memorized if it is isomorphic or nearly isomorphic to a training graph. With a single reference, this definition does not apply: the model is trained to produce a graph that resembles the reference, so some overlap is expected. A model can also score perfectly on every criterion except memorization by copying the reference.

Even a model that generates every edge independently shares some edges with the reference if it reproduces the reference degree sequence and community structure, so a non-zero overlap alone does not indicate memorization.

Table 8: Amazon Computers structural fidelity results.

Model	Deg. Rat.	Deg. W1	Tri. Rat.	Clus. C. Rat.	Assort. Diff.	Mod. Diff.	Inter/Intra	CPL Rat.
ER $G(n,p)$	$1.000_{\pm 0.002}$	$24.577_{\pm 0.033}$	$0.005_{\pm 0.000}$	$0.024_{\pm 0.000}$	$0.056_{\pm 0.002}$	$0.487_{\pm 0.002}$	$11.048_{\pm 1.701}$	$0.872_{\pm 0.000}$
ER $G(n,m)$	$1.000_{\pm 0.000}$	$24.562_{\pm 0.022}$	$0.005_{\pm 0.000}$	$0.024_{\pm 0.000}$	$0.056_{\pm 0.003}$	$0.487_{\pm 0.001}$	$10.704_{\pm 1.886}$	$0.872_{\pm 0.000}$
WS	$1.007_{\pm 0.000}$	$27.549_{\pm 0.013}$	$1.005_{\pm 0.002}$	$4.918_{\pm 0.012}$	$0.053_{\pm 0.001}$	$0.215_{\pm 0.001}$	$0.611_{\pm 0.008}$	$1.081_{\pm 0.001}$
BA	$1.005_{\pm 0.000}$	$11.047_{\pm 0.020}$	$0.052_{\pm 0.001}$	$0.113_{\pm 0.001}$	$0.049_{\pm 0.001}$	$0.494_{\pm 0.001}$	$10.036_{\pm 1.078}$	$0.839_{\pm 0.000}$
CL	$0.996_{\pm 0.002}$	$0.606_{\pm 0.038}$	$0.287_{\pm 0.002}$	$0.307_{\pm 0.002}$	$0.016_{\pm 0.001}$	$0.508_{\pm 0.001}$	$11.379_{\pm 0.784}$	$0.836_{\pm 0.001}$
Config	$0.978_{\pm 0.000}$	$0.797_{\pm 0.006}$	$0.199_{\pm 0.001}$	$0.246_{\pm 0.001}$	$0.013_{\pm 0.000}$	$0.506_{\pm 0.001}$	$9.857_{\pm 0.865}$	$0.843_{\pm 0.001}$
DC-SBM	$0.922_{\pm 0.003}$	$2.797_{\pm 0.090}$	$0.388_{\pm 0.002}$	$0.668_{\pm 0.003}$	$0.075_{\pm 0.001}$	$0.004_{\pm 0.001}$	$0.796_{\pm 0.020}$	$0.966_{\pm 0.001}$
GenCAT	$0.956_{\pm 0.001}$	$1.567_{\pm 0.044}$	$0.330_{\pm 0.009}$	$0.480_{\pm 0.012}$	$0.044_{\pm 0.003}$	$0.074_{\pm 0.004}$	$0.924_{\pm 0.102}$	$0.917_{\pm 0.002}$
GAE	$1.000_{\pm 0.000}$	$30.242_{\pm 0.014}$	$9.043_{\pm 0.008}$	$1.936_{\pm 0.003}$	$0.439_{\pm 0.000}$	$0.233_{\pm 0.000}$	$1.398_{\pm 0.004}$	$0.665_{\pm 0.000}$
VGAE	$1.000_{\pm 0.000}$	$6.971_{\pm 0.116}$	$0.275_{\pm 0.001}$	$0.639_{\pm 0.002}$	$0.083_{\pm 0.003}$	$0.389_{\pm 0.004}$	$7.505_{\pm 0.528}$	$0.902_{\pm 0.001}$
NetGAN	OOM					OOM		
CELL	$1.000_{\pm 0.000}$	$1.299_{\pm 0.023}$	$0.649_{\pm 0.003}$	$0.812_{\pm 0.003}$	$0.008_{\pm 0.001}$	$0.025_{\pm 0.001}$	$1.044_{\pm 0.066}$	$0.982_{\pm 0.001}$
GraphMaker S	TO					TO		
GraphMaker A	TO					TO		
SaGess	TO					TO		
LGSG	TO					TO		
SynGen	$0.839_{\pm 0.000}$	$9.535_{\pm 2.205}$	$0.063_{\pm 0.027}$	$0.223_{\pm 0.078}$	$0.248_{\pm 0.092}$	$0.236_{\pm 0.034}$	$1.650_{\pm 0.471}$	$0.970_{\pm 0.015}$
SCHEMA	$1.277_{\pm 0.001}$	$11.084_{\pm 0.023}$	$0.549_{\pm 0.002}$	$0.498_{\pm 0.002}$	$0.092_{\pm 0.001}$	$0.073_{\pm 0.003}$	$0.945_{\pm 0.085}$	$0.978_{\pm 0.001}$

Table 9: Flickr structural fidelity results.

Model	Deg. Rat.	Deg. W1	Tri. Rat.	Clus. C. Rat.	Assort. Diff.	Mod. Diff.	Inter/Intra	CPL Rat.
ER $G(n,p)$	$1.000_{\pm 0.002}$	$5.127_{\pm 0.014}$	$0.003_{\pm 0.000}$	$0.029_{\pm 0.002}$	$0.044_{\pm 0.001}$	$0.248_{\pm 0.002}$	$3.551_{\pm 0.151}$	$1.221_{\pm 0.001}$
ER $G(n,m)$	$1.000_{\pm 0.000}$	$5.127_{\pm 0.005}$	$0.003_{\pm 0.000}$	$0.029_{\pm 0.002}$	$0.044_{\pm 0.002}$	$0.249_{\pm 0.002}$	$3.549_{\pm 0.196}$	$1.221_{\pm 0.000}$
WS	$0.992_{\pm 0.000}$	$6.020_{\pm 0.002}$	$10.167_{\pm 0.019}$	$122.913_{\pm 0.231}$	$0.035_{\pm 0.002}$	$0.353_{\pm 0.000}$	$0.218_{\pm 0.001}$	$1.815_{\pm 0.003}$
BA	$0.992_{\pm 0.000}$	$1.565_{\pm 0.008}$	$0.063_{\pm 0.003}$	$0.211_{\pm 0.006}$	$0.030_{\pm 0.001}$	$0.243_{\pm 0.001}$	$2.468_{\pm 0.139}$	$1.002_{\pm 0.002}$
CL	$0.999_{\pm 0.001}$	$1.004_{\pm 0.008}$	$1.271_{\pm 0.015}$	$1.313_{\pm 0.017}$	$0.026_{\pm 0.000}$	$0.252_{\pm 0.001}$	$2.473_{\pm 0.064}$	$0.897_{\pm 0.001}$
Config	$0.996_{\pm 0.000}$	$0.043_{\pm 0.001}$	$0.877_{\pm 0.009}$	$0.980_{\pm 0.011}$	$0.023_{\pm 0.000}$	$0.246_{\pm 0.000}$	$2.403_{\pm 0.032}$	$0.892_{\pm 0.000}$
DC-SBM	$0.976_{\pm 0.001}$	$1.152_{\pm 0.011}$	$1.443_{\pm 0.019}$	$2.065_{\pm 0.034}$	$0.024_{\pm 0.000}$	$0.021_{\pm 0.008}$	$0.811_{\pm 0.087}$	$0.951_{\pm 0.001}$
GenCAT	$0.998_{\pm 0.000}$	$0.020_{\pm 0.001}$	$0.083_{\pm 0.009}$	$0.083_{\pm 0.009}$	$0.002_{\pm 0.000}$	$0.213_{\pm 0.008}$	$2.297_{\pm 0.250}$	$0.942_{\pm 0.001}$
GAE	$1.000_{\pm 0.000}$	$12.832_{\pm 0.002}$	$579.284_{\pm 0.231}$	$33.066_{\pm 0.041}$	$0.067_{\pm 0.000}$	$0.011_{\pm 0.002}$	$0.228_{\pm 0.049}$	$0.535_{\pm 0.000}$
VGAE	$1.000_{\pm 0.000}$	$6.172_{\pm 0.017}$	$4.120_{\pm 0.034}$	$8.705_{\pm 0.054}$	$0.110_{\pm 0.006}$	$0.208_{\pm 0.003}$	$1.910_{\pm 0.109}$	$0.951_{\pm 0.001}$
NetGAN	OOM					OOM		
CELL	OOM					OOM		
GraphMaker S	OOM					OOM		
GraphMaker A	OOM					OOM		
SaGess	OOM					OOM		
LGSG	TO					TO		
SynGen	$0.681_{\pm 0.000}$	$3.885_{\pm 0.254}$	$0.069_{\pm 0.044}$	$0.451_{\pm 0.220}$	$0.223_{\pm 0.064}$	$0.112_{\pm 0.039}$	$1.154_{\pm 0.511}$	$1.276_{\pm 0.057}$
SCHEMA	$0.831_{\pm 0.000}$	$2.287_{\pm 0.005}$	$4.579_{\pm 0.114}$	$3.333_{\pm 0.084}$	$0.026_{\pm 0.000}$	$0.273_{\pm 0.001}$	$0.097_{\pm 0.001}$	$1.259_{\pm 0.003}$

We also measure self-overlap, the mean overlap between pairs of samples of the same model. It exposes a second failure mode: low edge overlap with high self-overlap means the model produces nearly the same graph every time.

A.4.1 ALIGNMENT

Memorization metrics require node correspondence between graphs. Graph structure is invariant under node permutations, so that the same graph can be represented in different node orderings. The overlap between $G(\pi_1)$ and $G(\pi_2)$ therefore depends on the permutations π_1 and π_2, although both represent the same graph.

The models fall into three alignment classes. Unaligned models (ER, WS, BA, VGAE, GraphMaker, LGSG, and SynGen) assign node identities arbitrarily, so their overlaps carry no meaning, and we exclude them. Degree-aligned models (CL, Config, and GenCAT) are aligned by matching nodes of equal degree. This alignment comes from the degree sequence, not from the model, so it gives a lower bound on their overlap. Node-aligned models (DC-SBM, GAE, CELL, NetGAN, SaGess, and SCHEMA) keep the reference node identities, and all measures are defined directly.

A.4.2 MEMORIZATION METRICS

Let E be the undirected edge set of the reference, $\hat{E}^{(i)}$ that of the i-th generated sample, and S the number of samples ($S = 10$ throughout). All sets exclude self-loops and duplicate edges.

Edge overlap is the Jaccard similarity

$$\mathrm{EO} = \frac{1}{S} \sum_{i=1}^{S} \frac{|E \cap \hat{E}^{(i)}|}{|E \cup \hat{E}^{(i)}|}.$$

Self-overlap is the same similarity taken over sample pairs,

$$\mathrm{SO} = \binom{S}{2}^{-1} \sum_{i<j} \frac{|\hat{E}^{(i)} \cap \hat{E}^{(j)}|}{|\hat{E}^{(i)} \cup \hat{E}^{(j)}|}.$$

Neighborhood memorization is computed per node. Let $\mathcal{N}(v)$ and $\hat{\mathcal{N}}(v)$ be the neighbor sets of v in the reference and in a sample, and

$$J(v) = \frac{|\mathcal{N}(v) \cap \hat{\mathcal{N}}(v)|}{|\mathcal{N}(v) \cup \hat{\mathcal{N}}(v)|}.$$

We report the mean of $J(v)$ over nodes and samples (NM) and the fraction of nodes with $J(v) \geq 0.9$ (Frac.). Nodes isolated in both the reference and the sample are excluded, since $J(v)$ is undefined for them.

Attribute-level memorization. The four metrics above measure structural memorization. For attributes, we compute for each generated node the Euclidean distances d_1 and d_2 to its nearest and second-nearest reference nodes. We report the median of d_1/d_2 as the nearest-neighbor distance ratio (NNDR), and the fraction of generated nodes whose d_1 is below a threshold (Leak), over 10 samples per model. A ratio near zero means the node sits on one reference node and not between several, which indicates copying at the attribute level. The threshold is set to the fifth percentile of the reference nearest-neighbor distances. Hence, a model whose generated attributes follow the same distribution as the reference yields a fraction near 0.05 by construction. Since the threshold is derived from the reference, the fraction does not depend on the scale of the attribute space. It does depend on the generated attributes following the same marginal distribution. When they do not, the fraction collapses toward zero, or toward one when the generated attributes fall inside a dense region of the reference, and the measure no longer says anything about copying. We therefore give two marginal statistics alongside it, the mean number of non-zero attributes per node and the Spearman correlation between the per-attribute frequencies of the generated and reference graphs. The measurement requires no node identity, as it compares two sets of attribute points.

A.4.3 RESULTS

Structural memorization results are in Table 10 and attribute memorization results in Table 11.

Structural memorization. The alignment classes separate clearly. The degree-aligned models score below 0.03 on every measure and dataset, and none has a memorized node. This is expected from models that reproduce a degree sequence and nothing else, and the same models distort the community structure in the fidelity experiments. Low overlap alone is therefore not evidence of a good generator.

Self-overlap exposes two models. GAE has an edge overlap between 0.008 and 0.227 and a self-overlap between 0.743 and 0.951 across the five datasets, indicating that its samples are close copies of each other. This is expected, since GAE has no prior and generates by reconstructing the reference. NetGAN shows the same pattern on Amazon Photo, with an overlap of 0.027 against a self-overlap of 0.801, though not on the citation graphs. A measure of overlap with the reference alone would have scored both of these as generalizing.

SaGess memorizes directly. Its edge overlap is 0.910 on Citeseer and 0.815 on Cora ML, and 0.818 and 0.524 of its nodes keep their reference neighborhood almost exactly. It reconstructs the

Table 10: Structural memorization results. †NetGAN is trained and evaluated on the largest connected component, so its values are computed over a smaller node set than the other rows. EO is edge overlap, SO is self-overlap, and NM is neighborhood memorization. Frac. is the fraction of nodes with $J(v) \geq 0.9$.

Model	Alignment	Citeseer				Cora ML				Amazon Photo			
		EO	SO	NM	Frac.	EO	SO	NM	Frac.	EO	SO	NM	Frac.
CL	Degree	0.003	0.003	0.001	0.000	0.007	0.009	0.003	0.000	0.019	0.020	0.007	0.000
Config	Degree	0.003	0.002	0.001	0.000	0.007	0.008	0.003	0.000	0.017	0.016	0.007	0.000
GenCAT	Degree	0.002	0.004	0.001	0.000	0.005	0.014	0.002	0.000	0.015	0.028	0.007	0.000
DC-SBM	Node	0.084	0.063	0.140	0.085	0.052	0.046	0.050	0.015	0.075	0.066	0.046	0.002
GAE	Node	0.227	0.750	0.125	0.017	0.191	0.749	0.106	0.004	0.174	0.743	0.075	0.000
CELL	Node	0.384	0.258	0.500	0.198	0.442	0.284	0.516	0.129	0.337	0.237	0.340	0.007
NetGAN†	Node	0.228	0.142	0.208	0.034	0.174	0.094	0.151	0.015	0.027	0.801	0.025	0.000
SaGess	Node	0.910	0.847	0.904	0.818	0.815	0.735	0.795	0.524	TO			
SCHEMA	Node	0.102	0.108	0.254	0.183	0.038	0.066	0.068	0.034	0.040	0.068	0.035	0.005

Model	Alignment	Amazon Computers				Flickr			
		EO	SO	NM	Frac.	EO	SO	NM	Frac.
CL	Degree	0.021	0.022	0.007	0.000	0.001	0.004	0.001	0.000
Config	Degree	0.019	0.017	0.006	0.000	0.001	0.003	0.001	0.000
GenCAT	Degree	0.015	0.029	0.005	0.000	0.001	0.001	0.001	0.000
DC-SBM	Node	0.056	0.051	0.031	0.002	0.007	0.009	0.005	0.000
GAE	Node	0.102	0.820	0.030	0.000	0.008	0.951	0.002	0.000
CELL	Node	0.348	0.232	0.393	0.018	OOM			
NetGAN†	Node	OOM				OOM			
SaGess	Node	TO				OOM			
SCHEMA	Node	0.032	0.111	0.024	0.004	0.003	0.121	0.004	0.000

reference, which explains its structural fidelity. CELL sits between the two extremes, with an edge overlap between 0.337 and 0.442 and a mean neighborhood Jaccard of about 0.5 on the citation graphs, an amount set by its stopping rule.

SCHEMA has an edge overlap between 0.003 and 0.102, and its self-overlap exceeds its edge overlap on all five datasets. The reference is therefore no closer to a generated graph than two generated graphs are to each other, and the gap generally widens with graph size: on Flickr, the edge overlap is 0.003 against a self-overlap of 0.121. Its memorized-node fraction is at most 0.034 on four datasets. Citeseer is the exception at 0.183, and the analysis below indicates that this largely follows from its component structure.

Attribute memorization. NNDR does not discriminate between models. Every model on every dataset returns a value between 0.969 and 1.000, including the one with a leak fraction of 0.273. In a sparse high-dimensional binary attribute space, a generated node shares almost no attributes with any reference node, so the Euclidean distance is set by the number of non-zero attributes rather than by the overlap, and the nearest and second-nearest reference nodes sit at nearly the same distance. Therefore, the ratio carries no information here. We report it for completeness.

The leak fraction is interpretable only when the generated attribute marginals match the reference, and two models fall outside that condition. GenCAT generates continuous attributes with a frequency correlation below 0.3, so its zero leak reflects the mismatch. On the citation graphs, SynGen generates about 50 times (Citeseer) and 30 times (Cora ML) as many non-zero attributes per node as the reference, with frequency correlations of 0.465 and 0.389, respectively. On the Amazon graphs, the marginals are closer (about twice the density, correlation above 0.93). GenCAT's zero leak on every dataset and SynGen's zero leak on the citation graphs therefore carry no information about copying.

Table 11: Attribute memorization. The first row of each block is the reference graph. Feat. Den. is the mean number of non-zero attributes per node for binary attributes and the mean L1 norm per node for continuous attributes; it is left empty where the generated attribute type differs from the reference, as the two quantities are then on different scales. Feat. ρ is the Spearman correlation between the column means of the generated and the reference attribute matrix. For binary attributes, the column mean is the frequency of the attribute. The reference row carries no Feat. ρ, NNDR, or Leak value, as all three are defined between a generated graph and the reference. * Leak value not interpretable because the generated attribute marginals do not match the reference.

Model	Feat. Type	Feat. Den.	Feat. ρ	NNDR	Leak	Feat. Type	Feat. Den.	Feat. ρ	NNDR	Leak
	Citeseer					Cora ML				
Reference	binary	31.6	–	–	–	binary	50.5	–	–	–
GenCAT	cont.	–	0.298	1.000	0.000*	cont.	–	0.082	0.999	0.000*
GraphMaker S	binary	23.1	0.873	1.000	0.012	binary	33.8	0.922	0.988	0.000
GraphMaker A	binary	21.1	0.860	1.000	0.273	binary	26.1	0.898	0.969	0.074
SynGen	binary	1561.9	0.465	1.000	0.000*	binary	1584.7	0.389	0.999	0.000*
SCHEMA	binary	28.8	0.858	1.000	0.000	binary	53.2	0.949	0.992	0.000
	Amazon Photo					Amazon Computers				
Reference	binary	258.8	–	–	–	binary	267.2	–	–	–
GenCAT	cont.	–	0.013	0.999	0.000*	cont.	–	0.006	0.999	0.000*
GraphMaker S	binary	263.0	0.979	0.996	0.000			TO		
GraphMaker A	binary	235.4	0.989	0.990	0.004			TO		
SynGen	binary	496.0	0.935	0.998	0.000	binary	510.9	0.964	0.998	0.000
SCHEMA	binary	270.5	0.963	0.996	0.002	binary	264.4	0.955	0.997	0.002
	Flickr									
Reference	cont.	431.0	–	–	–					
GenCAT	cont.	250.0	0.002	0.999	1.000*					
GraphMaker S			OOM							
GraphMaker A			OOM							
SynGen	cont.	656.0	0.916	0.994	0.000*					
SCHEMA	cont.	328.7	0.981	0.981	1.000*					

Among the models whose marginals match the reference, the asynchronous variant of GraphMaker has the highest leak, at 0.273 on Citeseer and 0.074 on Cora ML, against 0.012 and 0.000 for the synchronous variant. Its feature-only accuracy is also above the reference (Table 13), which is consistent with attributes close to individual reference nodes.

SCHEMA reproduces the attribute marginals closely on every dataset. Its non-zero attribute count is within a tenth of the reference on the four binary datasets, and its attribute-frequency correlation ranges from 0.858 to 0.981, the highest of all models on Cora ML and Flickr. The leak remains at or below 0.002 everywhere except Flickr, so the attributes it generates are close to the reference distribution but not to any individual reference node.

On Flickr, the attributes are continuous, and the leak fraction inverts: SCHEMA and GenCAT both reach 1.000, while SynGen stays at 0.000. The reference has a mean L1 norm of 431.0 per node, against 328.7 for SCHEMA and 250.0 for GenCAT, so both generate smaller vectors. Vectors with a smaller norm lie closer to the center of the reference point cloud. A point there is closer to its nearest reference node than most reference nodes are to their own nearest neighbor, so every generated node falls below the threshold. SynGen generates larger vectors (656.0) and has zero leak. This is the marginal mismatch described above in the other direction: shrunken attributes cross the threshold everywhere, and inflated ones never do. The NNDR of SCHEMA on Flickr is 0.981, far from the value near zero that copying produces, which supports this reading. We mark the Flickr leak values as uninterpretable.

Citeseer neighborhood memorization analysis. Citeseer has 3,327 nodes and 4,552 undirected edges in 438 connected components. The largest component covers 63.7% of the nodes, 16.4% of the nodes are in components of at most two nodes, and 40.0% have degree one. SCHEMA generates intra-community edges under a per-node degree budget inside the node's own leaf community. Let $C(i)$ be the leaf community of node i and let d_i be its number of neighbors inside $C(i)$. The generator selects those d_i neighbors from the $|C(i)| - 1$ other members of the community, so the

node admits $\binom{|C(i)|-1}{d_i}$ distinct intra-community neighborhoods. When $d_i = |C(i)|-1$, this count is one, and the reference neighborhood is the only one the model can produce. A two-node component is the extreme case. Each node has one candidate, so the reference edge is the only edge SCHEMA can place, and both nodes score a Jaccard similarity of 1.

Because SCHEMA's neighborhood memorization on Citeseer is far above its value on the other four datasets, we analyze it further alongside CELL and DC-SBM. For each node, we compute the probability that a generator reproduces the reference neighborhood by random chance. A generator that picks d_i of the $m_i = |C(i)| - 1$ candidates at random does so with probability $1/\binom{m_i}{d_i}$. We call a node constrained when this exceeds 0.05. We then shuffle and rewire the reference graph itself while keeping each node's degree unchanged and each edge in the same pair of leaf communities. Isolated nodes are excluded, and we average over 20 shuffles. Of the 3,279 evaluated nodes, 1,064 are constrained. These rewired graphs reach a memorized-node fraction of 0.286 ± 0.003, above SCHEMA's 0.183. On the constrained nodes, the memorized-node fraction is 0.858 for the rewired graphs and 0.561 for SCHEMA. Leaf communities of at most two nodes contribute 82.9% of SCHEMA's memorized neighborhoods.

Splitting the nodes this way separates the three models. Over the 2,215 unconstrained nodes, SCHEMA reproduces 4 neighborhoods, DC-SBM 7, and CELL 299. Restricted to the unconstrained nodes of the largest component, SCHEMA reproduces 4 neighborhoods, DC-SBM 5, and CELL 292.

DC-SBM has a lower memorized-node fraction, 0.085, but the whole difference lies in the constrained nodes, where SCHEMA scores 0.561 and DC-SBM 0.257. On the unconstrained nodes, both reproduce fewer than 10 neighborhoods.

A.5 DOWNSTREAM EVALUATION

We test whether a generated graph can serve as a substitute for the reference in node classification. Three settings are measured. TRTR trains and tests on the reference, setting the expected ceiling for the downstream task. TSTR trains on the generated graph and tests on the reference graph, measuring whether a model trained on generated data transfers to the reference. TSTS trains and tests on the generated graph, which measures how hard the generated task is.

Only three baselines generate attributes (GenCAT, SynGen, and GraphMaker, in two variants), and only these are included. We do not let models inherit reference attributes through the node alignment, since that would give them real features and an unfair advantage.

We use a $60/20/20$ train/validation/test split, drawn with a fixed seed. Every model is evaluated on the same split, and all models use the same training seeds. SynGen generates additional nodes, so its masks are drawn class by class to match the size and class distribution of the reference split.

Utility is the TSTR divided by TRTR, averaged over four two-layer architectures, GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and SGC (Wu et al., 2019), and five seeds. For the architecture rankings, we train 20 configurations under both TSTS and TRTR: GCN, GraphSAGE, GAT, GIN (Xu et al., 2019), and SGC at one to four layers, each over three seeds. Spearman and Pearson correlations are computed between the TSTS and TRTR accuracies of the 20 configurations. All runs use a hidden dimension of 64, Adam optimizer at a learning rate of 0.01 with weight decay of 5×10^{-4}, and at most 200 epochs, with early stopping on validation accuracy and a patience of 50 epochs.

Absolute accuracies. Table 12 lists absolute accuracies next to the majority-class rate of each graph. The utility ratio alone hides class imbalance: when the reference accuracy is close to the majority rate, a model that predicts the majority class still achieves a high ratio. Flickr is such a case. The reference reaches 0.50 against a majority rate of 0.42, and every generated graph returns exactly the majority rate under both TSTR and TSTS. The task does not separate the models on Flickr, so we report no further downstream results for it.

SynGen falls below the majority rate under both TSTR and TSTS on all four evaluation datasets, which is consistent with the design discussed in Section 4.4. Its TSTS on Cora ML, Amazon Photo, and Amazon Computers is also below the uniform chance.

Table 12: Absolute node classification accuracy, averaged over architectures and seeds. The majority-class rate is the accuracy of always predicting the most frequent class.

Model		Citeseer	Cora ML	A. Photo	A. Computers	Flickr
Majority class		0.21	0.30	0.27	0.37	0.42
Reference	TRTR	0.69	0.86	0.94	0.90	0.50
GenCAT	TSTR	0.52	0.59	0.44	0.38	0.42
	TSTS	0.71	0.81	0.72	0.41	0.42
GraphMaker S	TSTR	0.71	0.85	0.80	TO	OOM
	TSTS	0.62	0.83	0.99		
GraphMaker A	TSTR	0.73	0.84	0.92	TO	OOM
	TSTS	0.82	0.94	0.95		
SynGen	TSTR	0.17	0.22	0.21	0.22	0.42
	TSTS	0.19	0.02	0.04	0.03	0.42
SCHEMA	TSTR	0.59	0.70	0.76	0.69	0.42
	TSTS	0.56	0.73	0.83	0.77	0.42

Homophily and the TSTS gap. TSTS trains and tests on the same generated graph. A generator that adds same-class edges makes this easier for message-passing models, which raises TSTS without bringing the graph closer to the reference. On Citeseer, Cora ML, and Amazon Photo, the three datasets on which GraphMaker completes, we measure how much of each TSTS−TRTR gap homophily explains. We use adjusted homophily (Platonov et al., 2023),

$$h_{\mathrm{adj}} = \frac{h_{\mathrm{edge}} - \sum_c p_c^2}{1 - \sum_c p_c^2},$$

where h_{edge} is the fraction of edges that join nodes of the same class and $p_c = D_c/2|E|$ is the degree share of class c, with D_c the total degree of its nodes.

For each target level of h_{adj}, we rewire the reference with degree-preserving double edge swaps. The swap is valid only when it drives the number of same-class edges toward that level. With degrees and labels fixed, h_{adj} is a linear function of this number, so a target homophily fixes a target count. The targets are 0, $h_{\mathrm{ref}}/2$, five steps from h_{ref} to 0.95, and the h_{adj} of each generated model, over two rewiring seeds. Training the downstream models of Section 4.4 on each rewired graph yields a curve $C(h)$ from homophily to accuracy. For a model with homophily h_m, $C(h_m) - \mathrm{TRTR}$ is the gain the reference itself shows at that homophily, and $\mathrm{TSTS} - C(h_m)$ is the excess that is not caused by homophily. We also rewire each generated graph to h_{ref}, measure TSTS again, and train a logistic regression on the attributes alone.

GenCAT is excluded from this analysis. Part of its GraphSAGE and GAT fits collapse to the majority rate on all three datasets, so its TSTS is not a stable estimate.

Table 13 lists the homophily, edge count, and feature-only (without graph structure) accuracy of each graph, and Table 14 splits each TSTS−TRTR gap into its homophily and the parts that cannot be explained without homophily. Whenever a generated graph is easier in the downstream task, its homophily is also higher than the reference. For these graphs, once the reference is rewired to the generated graph's homophily, the remaining difference is between -0.033 and $+0.005$. Which GraphMaker variant inflates homophily depends on the dataset: the asynchronous one on Citeseer and Cora ML, and both on Amazon Photo. On Citeseer and Cora ML, the inflating variant also generates two to four times as many edges as the reference. In contrast, the asynchronous variant on Amazon Photo raises homophily at 0.92 of the reference edge count and still widens the gap, so the inflation does not depend on the extra edges alone.

GraphMaker also generates attributes that separate the classes better than the real ones. In all six cases, its feature-only accuracy is 0.05 to 0.12 above the reference, reaching 0.995 for the synchronous variant on Amazon Photo. This makes the task easier independently of homophily. Rewiring the asynchronous variant to h_{ref} lowers its TSTS by only 0.01 to 0.02, and it stays 0.10 above TRTR on Citeseer. The synchronous variant on Citeseer shows the attribute difference alone: generated homophily is below the reference, yet in the calibration run its TSTS matches TRTR (Table 14), with an excess of $+0.062$.

Table 13: Properties of the reference and generated graphs. h_{adj}: adjusted homophily. Feature-only: accuracy of a logistic regression trained on node attributes without edges.

Dataset	Model	h_{adj}	$\lvert E\rvert/\lvert E_{\mathrm{ref}}\rvert$	Feature-only
Citeseer	Reference	0.671	1.00	0.686
	SCHEMA	0.512	1.00	0.529
	GraphMaker S	0.516	0.86	0.766
	GraphMaker A	0.870	4.28	0.810
Cora ML	Reference	0.749	1.00	0.798
	SCHEMA	0.503	1.03	0.482
	GraphMaker S	0.639	0.64	0.849
	GraphMaker A	0.926	2.17	0.844
A. Photo	Reference	0.785	1.00	0.913
	SCHEMA	0.586	0.97	0.621
	GraphMaker S	0.935	5.18	0.995
	GraphMaker A	0.900	0.92	0.968

Table 14: Homophily calibration of TSTS. TSTS$-$TRTR is split into $C_{\mathrm{ref}}(h_m)-\mathrm{TRTR}$, the change in accuracy when the reference is rewired to the model's homophily h_m (Table 13), and the residual $\mathrm{TSTS}-C_{\mathrm{ref}}(h_m)$. $C_{\mathrm{gen}}(h_{\mathrm{ref}})$: TSTS after rewiring the generated graph to the reference homophily h_{ref}. TRTR is 0.744 on Citeseer, 0.857 on Cora ML, and 0.938 on Amazon Photo, each under a separate draw of the default split; on Citeseer, it differs from Table 12, and so do the TSTS$-$TRTR values. Rewiring alone raises accuracy by 0.013 to 0.025, so rewired differences smaller than that are not interpretable.

			Ref. rewired to h_m		Gen. rewired
Dataset	Model	TSTS$-$TRTR	$C_{\mathrm{ref}}(h_m)-$TRTR	TSTS$-C_{\mathrm{ref}}(h_m)$	$C_{\mathrm{gen}}(h_{\mathrm{ref}})-$TRTR
Citeseer	SCHEMA	-0.069	-0.068	-0.001	-0.019
	GraphMaker S	-0.005	-0.067	+0.062	+0.070
	GraphMaker A	+0.126	+0.137	-0.011	+0.102
Cora ML	SCHEMA	-0.129	-0.076	-0.053	+0.042
	GraphMaker S	-0.026	-0.028	+0.003	+0.029
	GraphMaker A	+0.084	+0.117	-0.033	+0.063
A. Photo	SCHEMA	-0.108	-0.025	-0.084	+0.007
	GraphMaker S	+0.054	+0.048	+0.005	+0.051
	GraphMaker A	+0.020	+0.043	-0.023	+0.012

For SCHEMA, generated homophily is 0.16 to 0.25 below the reference and feature-only accuracy 0.16 to 0.32 below, and its TSTS is 0.07 to 0.13 below TRTR. Rewired to h_{ref}, it comes within 0.05 of TRTR. Its unexplained remainder ranges from -0.001 on Citeseer to -0.084 on Amazon Photo.

This analysis has three limitations. First, rewiring alone raises TSTS. A control that applies the same number of swaps at fixed homophily scores 0.013 to 0.025 above TRTR, so rewired differences smaller than that are within the effect of rewiring itself. Second, equal homophily does not mean equal structure. The targeted swaps distribute the same-class edges more evenly than in a real graph. Third, rewiring the reference keeps the real attributes, and rewiring a generated graph keeps the generated ones. The two results are therefore reflect different attributes and cannot be combined into one number.

A.6 SCALABILITY

We measure scalability on the six scalability graphs of Table 4. Results are in Table 15. Most baselines do not run at this scale, so we compare against models that generate attributes and complete on Flickr, leaving GenCAT and SynGen. The other models that run on Flickr, including the classical ones, generate structure only and solve a smaller problem, so we leave them out.

Table 15: Training and generation time on the scalability graphs, in seconds (s), minutes (m), and hours (h).

Model		Flickr	Reddit	Yelp	OGBN	DGraphFin	IGB
GenCAT	Fit	0.72 s	11.55 s	2.4 s[§]	20.34 s	2.23 s	30.92 s[*]
	Generate	36.78 s	33 m	24.44 m[§]	OOM	19.91 m	OOM
SynGen	Fit	14.03 s	12.47 s	261.33 s[‡]	50.30 s	603.21 s	332.59 s
	Generate	5.68 s	24.92 s	18.38 s[‡]	16.18 s	4.73 s	53.9 s
SCHEMA	Clustering	8.2 s	423.3 s	261.9 s	151.86 m	214.0 s	31.91 m[†]
	Train	145.3 s	22.71 m	92.46 m	5.69 h	72.42 m	5.16 h[†]
	Generate	1.2 s	15.3 s	107.5 s	422.7 s	23.1 s	286.5 s[†]

GenCAT runs on CPU only, and SynGen and SCHEMA use a single GPU. OOM for GenCAT means the run exhausted the 32 GB of host memory.

[§] GenCAT supports only single-label graphs, so on Yelp, we give it the first label of each node. Therefore, its Yelp input differs from that of the other models.

[*] IGB Medium has no node labels, and GenCAT requires them, so we assign labels uniformly at random. GenCAT does not complete generation here, and its fit time on random labels may not be representative.

[‡] Yelp's 100 binary labels were written as 100 categorical columns for SynGen.

[†] SCHEMA runs on the largest connected component of IGB Medium (98.8% of its nodes), with leaf communities capped at $100{,}000$ nodes.

We report training (or fitting) time and generation time separately. For GenCAT, fit time is the time required to estimate its parameters from the reference. For SynGen, fit time covers fitting the structure generator, the feature generator, and the aligner. For SCHEMA, the training time includes training across all three stages, and we also report the clustering time separately. Generation time is the wall-clock time to produce one graph. All runs use the hardware described in Section 4.1.

GenCAT fits in under a minute everywhere, but each sample then takes much longer to generate: 33 minutes on Reddit and 20 minutes on DGraphFin, and it exhausts host memory on OGBN Products and IGB Medium. SCHEMA spends most of its time before generation. Clustering and training together take up to 8.2 hours, on OGBN Products, and generation takes between one and seven minutes on Yelp, OGBN Products, and IGB Medium and under 30 seconds on the others. One generated graph costs between 0.5% and 2% of the clustering and training time on every graph, so additional samples are cheap.

SynGen has the lowest total cost on every graph. It fits a small set of structural parameters and fits the attributes and their alignment separately, so its cost is closer to that of a classical generator than a learned one. Its fidelity and downstream results in Sections 4.2 and 4.4 reflect the same design.

A.7 Generating Graphs of Any Size

SCHEMA matches the reference node count by design, but its learned weights do not depend on N. The node count comes from the tree: each stage reads the leaves, their soft assignments, and the pooled matrices from disk in batches. Changing these inputs changes N without retraining.

We change the graph size by replicating or subsampling communities. For a scale factor s, the target node count is sN. If s is an integer, the generated graph contains s copies of every community. If $s > 1$ is not an integer, it contains $\lfloor s \rfloor$ copies of every community and a greedily chosen subset of additional communities. If $s < 1$, it contains only a greedily chosen subset.

Stages 1 and 2. Each leaf of each copy runs through Stage 1 and Stage 2 with its own reference inputs. Copies draw independent noise. Neither stage needs a change.

Stage 3. If Stage 3 were left unchanged, it would connect communities only within each copy, and the result would be a disjoint union of copies, disconnected even when the reference is connected. We therefore generate Stage 3 edges in two steps. First, Stage 3 runs inside each copy exactly as in the standard generation. Second, each of these edges is reassigned to a random pair of copies of its two endpoints, drawn from two independent random permutations of the copies. This keeps every node's degree, and changes only which copies of an edge join.

Table 16: Generation at different scale factors on Flickr. Scale is the scale factor s, N_{exp} is the expected number of nodes, and N_{gen} is the generated number of nodes. Edge rat. (gen) is the number of generated edges divided by s times the edge count SCHEMA generates at scale one. Edge rat. (ref) divides it by s times the reference edge count.

Scale	N_{exp}	N_{gen}	Edge rat. (gen)	Edge rat. (ref)
0.5	44,625	45,225	1.040 ± 0.044	0.864 ± 0.036
1	89,250	89,250	1.000 ± 0.001	0.831 ± 0.001
1.5	133,875	135,058	0.992 ± 0.014	0.824 ± 0.012
2	178,500	178,500	0.987 ± 0.001	0.820 ± 0.001
2.5	223,125	223,433	0.985 ± 0.005	0.818 ± 0.004
5	446,250	446,250	0.992 ± 0.000	0.824 ± 0.000
10	892,500	892,500	0.995 ± 0.000	0.827 ± 0.000
100	8,925,000	8,925,000	0.999 ± 0.000	0.830 ± 0.000

Two reassigned edges can land on the same pair of copies, and the duplicate is dropped, which lowers the edge count. The loss depends on the scale factor and is largest at scale factors 2 and 2.5 in Table 16.

We run this study on Flickr, sweeping scale factors of 0.5, 1, 1.5, 2, 2.5, 5, 10, and 100. Table 16 reports the expected and generated node counts and two edge ratios: against the reference, and against the graph SCHEMA generates at scale one. Each entry is the mean and standard deviation over 3 samples.

For integer scale factors, the generated node count is exact. For fractional ones, it deviates slightly because communities are added whole. Edge rat. (gen) stays within 1.5% of one for every scale factor above one and approaches one as the scale grows, since the fraction of colliding edges falls. At $s = 0.5$, it is 1.040, because the chosen subset holds more nodes than the target. Edge rat. (ref) stays near 0.83, which is the ratio SCHEMA already produces at scale one. Hence, the scale factor does not change the edge ratio relative to the reference. At scale 100, generation completes, producing a graph with 8.9M nodes and 37.3M edges.

A.8 Ablation Study

We run ablation studies on Cora ML because the sweeps produce many generated graphs, and Cora ML is one of the smallest references. We do not use Citeseer: its many small components fall into separate leaves whose candidate sets are too small for the edge-stage variants to differ. Cora ML's leaves are large enough to exercise both edge stages. We do not repeat the studies on the larger graphs.

We report the W_1 degree distance, the triangle ratio, and the Inter/Intra ratio for structural fidelity, edge overlap, and self-overlap for memorization, and TSTS−TRTR for downstream utility.

Each ablation changes one part of the pipeline while keeping the rest at the same configuration.

Stage-wise ablation. Each stage draws noise during generation, and each can be switched off independently, yielding eight combinations. With no noise, Stage 2 encodes the observed adjacency of each leaf, and Stage 3 reads the observed inter-community edges, so the result follows the reference. Noise in all three stages is the configuration reported throughout the paper.

Partitioning. We replace Leiden with Louvain (Blondel et al., 2008) in both the leaf partition and the recursive tree construction, and separately with a random partition. Random partitioning keeps the community sizes Leiden produced and assigns nodes to them at random, so the leaves have the same sizes but no structural meaning.

Hard assignment. This ablation replaces the soft assignments with one-hot matrices over the same leaf partition. No node then carries mass in more than one child, and no bridge nodes will be found. Making the candidate pool drawn at random. The pooled connectivity and the cluster-affinity term remain.

Branching cap. The branching cap b bounds how many supernodes a tree node holds before the recursion splits it again. At $b = 2$, the recursion builds a binary tree. At $b = 1000$, it stops at the root, since Cora ML has fewer leaves than that, so every leaf community is a direct child of the root, and the hierarchy has one level.

Joint attribute generation. At generation, we replace the attributes produced by Stage 1 with the reference attributes, to test whether the edge stages depend on the generated attributes.

Stage architectures. We replace the architecture of one stage at a time. Stage 1 uses an MLP in place of the Transformer, so the attributes of nodes in the same leaf are generated independently. Stage 2 uses an inner-product decoder or an MLP edge predictor. Stage 3 uses an attention scorer in place of the bilinear scorer. The bilinear scorer is already the simpler option, so this row tests whether a more complex scorer helps.

Cluster affinity. Stage 3 adds a cluster-affinity term to each pair score from the hierarchy. We compare three settings. Without the term, the scorer learns from the observed inter-community edges alone. In the reported setting, the term is added to the score with a learned weight, so the model starts from the connectivity that hierarchy contains. In the third setting, the term is a KL target during training and is not added to the scores at generation.

Results. Table 17 separates the three stages. Edge overlap falls from 0.149 with no noise to 0.038 with noise in all stages, and the drop comes from Stage 3: the four combinations that read the observed inter-community edges sit between 0.111 and 0.149, and the four that generate them sit between 0.037 and 0.074. Triangle counts move in the opposite direction and with the intra-edge stage. When that stage encodes the observed adjacency of each leaf, the triangle ratio is between 0.417 and 1.360. When it draws from the prior instead, it is between 0.091 and 0.275. The model reproduces the triangle density of the reference only when it reads the reference adjacency. This is the trade-off from Section 4.2, measured within our own pipeline instead of across models.

Node noise reaches the structure only through the encoder. With Stage 2 in its deterministic mode, switching node noise on changes the degree distance from 1.502 to 0.590 and the triangle ratio from 1.171 to 0.417. With Stage 2 generating and Stage 3 deterministic, the same switch changes both by less than 0.01. The encoder reads the attributes, and the prior does not, so once Stage 2 samples from the prior, Stage 1 has little effect on the structure.

Table 18 shows an ablation study on components. Replacing Leiden with random communities of the same sizes changes the degree distance from 1.075 to 4.026, the triangle ratio from 3.515 to 3.515, and the Inter/Intra ratio from 7.516 to 7.516. Louvain, which optimizes the same objective, gives a lower degree distance (0.933) but a higher Inter/Intra ratio (1.263). The cluster-affinity term also matters: without it, the Inter/Intra ratio rises from 1.030 to 1.247 and the degree distance to 1.432. Using it as a KL target instead of adding it to the scores recovers part of this.

The remaining components have smaller effects on this graph. Hard assignments reach an Inter/Intra ratio of 0.838 against 1.030 with soft ones, so removing bridge nodes that Stage 3 draws its end-

Table 17: Stage-wise ablation on Cora ML. A check mark means the stage draws noise at generation. The last row is the configuration reported in the main text. SO is undefined in the first row, since all samples are identical without noise.

Noise			Structure			Memorization		Downstream
Node	Intra	Inter	Deg. W1	Tri. Rat.	Inter/Intra	EO	SO	TSTS−TRTR
			1.502	1.171	0.871	0.149	n/a	-0.080 ± 0.012
✓			0.590 ± 0.062	0.417 ± 0.051	0.837 ± 0.035	0.113	0.104	-0.118 ± 0.019
	✓		1.170 ± 0.013	0.095 ± 0.004	0.992 ± 0.060	0.112	0.105	-0.082 ± 0.013
✓	✓		1.177 ± 0.018	0.091 ± 0.005	0.986 ± 0.064	0.111	0.105	-0.126 ± 0.024
		✓	1.587 ± 0.045	1.360 ± 0.048	0.989 ± 0.028	0.074	0.183	-0.095 ± 0.013
✓		✓	1.011 ± 0.011	0.577 ± 0.048	0.899 ± 0.087	0.044	0.066	-0.126 ± 0.020
	✓	✓	1.038 ± 0.030	0.246 ± 0.009	0.902 ± 0.087	0.037	0.061	-0.097 ± 0.013
✓	✓	✓	1.075 ± 0.024	0.275 ± 0.019	1.030 ± 0.083	0.038	0.066	-0.125 ± 0.029

Table 18: Component ablations on Cora ML. Each row changes one part of the pipeline and leaves the rest at the reported configuration, given in the first row.

		Structure			Memorization		Downstream
Ablation	Setting	Deg. W1	Tri. Rat.	Inter/Intra	EO	SO	TSTS−TRTR
Reported		1.075 ± 0.024	0.275 ± 0.019	1.030 ± 0.083	0.038	0.066	-0.125 ± 0.029
Partitioning	Louvain	0.933 ± 0.037	0.277 ± 0.015	1.263 ± 0.088	0.037	0.061	-0.107 ± 0.021
	Random	4.026 ± 0.034	3.515 ± 0.069	7.516 ± 0.472	0.017	0.170	n/a
Assignment	Hard	1.226 ± 0.026	0.208 ± 0.007	0.838 ± 0.048	0.033	0.044	-0.142 ± 0.027
Branching cap	$b = 2$	1.365 ± 0.027	0.332 ± 0.015	0.946 ± 0.032	0.036	0.071	-0.127 ± 0.026
	$b = 1000$	0.842 ± 0.019	0.278 ± 0.007	1.044 ± 0.029	0.039	0.069	-0.124 ± 0.027
Attribute source	Reference	1.036 ± 0.020	0.238 ± 0.011	0.942 ± 0.148	0.037	0.062	-0.134 ± 0.024
Architecture	Stage 1, MLP	1.128 ± 0.033	0.284 ± 0.013	0.864 ± 0.099	0.038	0.067	-0.135 ± 0.028
	Stage 2, inner product	1.252 ± 0.050	1.042 ± 0.042	1.073 ± 0.118	0.071	0.127	-0.129 ± 0.025
	Stage 2, MLP	1.703 ± 0.051	1.196 ± 0.061	0.930 ± 0.054	0.079	0.131	-0.121 ± 0.024
	Stage 3, att.	0.917 ± 0.032	0.181 ± 0.007	0.827 ± 0.137	0.037	0.051	-0.134 ± 0.027
Affinity	KL	1.305 ± 0.037	0.238 ± 0.011	1.083 ± 0.042	0.033	0.048	-0.135 ± 0.025
	None	1.432 ± 0.021	0.353 ± 0.016	1.247 ± 0.059	0.033	0.056	-0.132 ± 0.025

points from degrades the result without breaking it for this dataset. A binary tree and a single-level hierarchy both stay within 0.09 of the reported Inter/Intra ratio. Hence, the depth of the recursion matters little on a graph of this size, although the binary tree raises the degree distance to 1.365. Replacing the Transformer of Stage 1 with an MLP keeps the degree distance, the triangle ratio, and the downstream gap close to the reported values and lowers the Inter/Intra ratio to 0.864, within about two standard deviations. Replacing the bilinear scorer of Stage 3 with an attention scorer lowers the degree distance to 0.917 but moves the Inter/Intra ratio further from one, to 0.827, and lowers the triangle ratio to 0.181.

The two simpler decoders for Stage 2 show the same trade-off as the stage-wise table. An inner-product decoder reaches a triangle ratio of 1.042 and an MLP predictor 1.196, both closer to the reference than the 0.275 of the reported configuration, but their edge overlap rises to 0.071 and 0.079 and their self-overlap to 0.127 and 0.131. They recover the triangle density by staying closer to the reference edge set.

Replacing the generated attributes with the reference attributes changes the structural metrics by at most about two standard deviations. At generation, Stage 2 draws its latents from the prior, and Stage 3 reads memberships and pooled connectivity, so neither edge stage reads the generated attributes. Attributes and edges are tied through the shared soft memberships, not through the realized attributes.