STAGE: SCALING NOVEL GRAPH GENERATION VIA LIGHTWEIGHT STRUCTURE-GUIDED AUTOREGRESSIVE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating realistic and diverse graphs is a fundamental challenge because molecules, materials, circuits, and malware can all be represented as graphs. A useful generator must produce novel graphs that follow the underlying data distribution rather than simply reproduce training examples. For instance, regenerating training molecules yields no new drug or material candidates, while replicating known malware graphs provides no new variants for improving detectors. Graph generators must therefore scale to large, sparse graphs while producing novel, diverse, and distributionally consistent samples. Current methods fall short on at least one requirement. Dense autoregressive and diffusion models incur at least quadratic cost in graph size, and sparse autoregressive models scale but, like dense ones, fit the training distribution, so novelty arises only incidentally. To our knowledge, no existing method achieves both scalability and novelty. We introduce STAGE, a lightweight autoregressive graph generator designed for both scalability and novelty. For scalability, STAGE serializes graphs as flat bit-level streams of lexicographically sorted edges, yielding $O(|E| \log |V|)$ generation with any causal sequence backbone; to make this compact stream learnable, STAGE orders nodes by a breadth-first traversal that enqueues neighbors by SIR-GN structural rank, resolving many of the arbitrary choices left by plain BFS. For novelty, STAGE uses a two-phase training strategy that combines exploration-oriented edge perturbations with self-training on generated samples. Samples are retained either by a label-free Gaussian-mixture density in graph-embedding space or, when available, by a domain validity oracle, steering exploration toward plausible regions. With label-free filtering, STAGE achieves the highest joint validity–uniqueness–novelty (VUN) on QM9, QM7x, Transition1x, and ANI-1x (56.8% vs. 55.0% on QM9; 86.4% vs. 59.4% on QM7x), matches the strongest baseline on MOSES, and remains competitive on ZINC250K. With oracle filtering, it reaches 60.9% VUN on QM9 and 95.1% on MOSES, compared with 55.0% and 85.4% for the strongest baselines. Ablations show that ordering is critical, with random ordering sharply degrading validity. Finally, STAGE trains on MalNet-Tiny function-call graphs with a median of 2,272 nodes, a regime in which DiGress, DeFoG, and SparseDiff exhaust GPU memory.

1 INTRODUCTION

Many scientific and engineering systems are naturally represented as graphs: molecules as atoms joined by bonds, circuits as interconnected components, and software as dependency or control-flow graphs. Graph generation enables the synthesis of novel structured objects that preserve the regularities of real data, with applications ranging from molecular discovery to program analysis and cyber-defense. Two properties determine whether a generator is useful in these settings: it must scale to the graph sizes of the domain (Android function-call graphs, for example, routinely exceed thousands of nodes), and it must produce graphs that are genuinely new rather than reproductions of its training data, while remaining plausible. Current generators fall short on at least one.

Despite rapid progress, existing graph generators remain limited in scalability. Dense autoregressive and hybrid approaches incur at least quadratic complexity in graph size, and diffusion-based ap-

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.

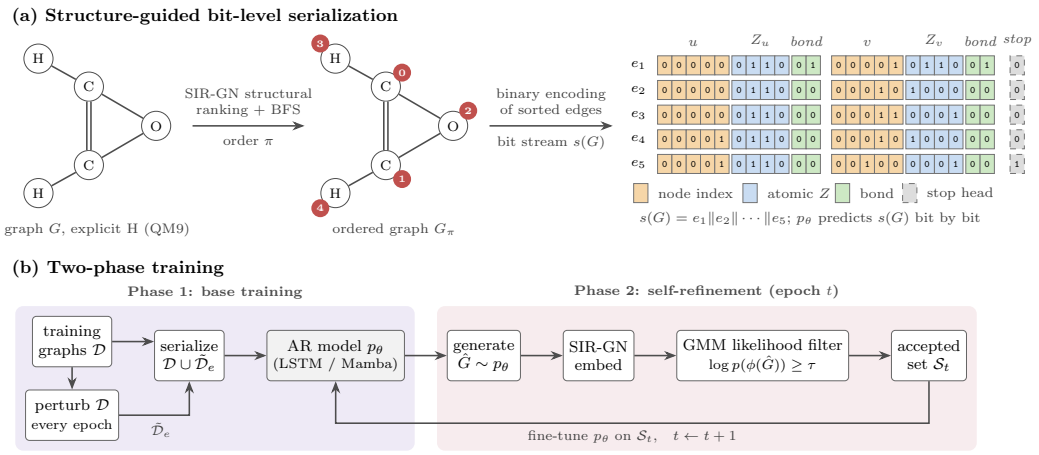


Figure 1: Overview. (a) Serialization of oxirene (QM9, explicit H): SIR-GN ranking guides BFS from the max-degree node to give π; each sorted edge becomes a bit token (Z: atomic number), concatenated into $s(G)$, with a separate stop head. (b) Phase 1 trains on $\mathcal{D}$ plus perturbed copies regenerated each epoch; Phase 2 fine-tunes on generated graphs whose GMM log-likelihood exceeds τ, the q-quantile on training graphs.

proaches rely on costly operations over the full adjacency structure, repeated across iterative denoising chains. In practice, DiGress (Vignac et al., 2023), DeFoG (Qin et al., 2025a), and SparseDiff (Qin et al., 2025b) cannot be trained on MalNet-Tiny (Freitas et al., 2021), whose median graph has 2,272 nodes, while remaining within GPU memory in our evaluated configuration (Appendix C). Sparse autoregressive generators such as BiGG (Dai et al., 2020) are more lightweight, emitting only existing edges, but their performance depends strongly on the serialization strategy: a graph has no canonical order, the same graph maps to many sequences, and the chosen linearization largely determines how hard the sequence is to learn.

More fundamentally, existing distribution-learning graph generators treat novelty as a by-product rather than a design goal. Likelihood, score-matching, flow-matching, and denoising objectives all fit the empirical training distribution, which a perfect fit would simply reproduce; novelty arises only to the extent that the model fits imperfectly. In practice it is often low: on QM7x, 37–51% of the valid, unique molecules produced by the baselines we evaluate already appear in the training set (Table 4).

We introduce STAGE (Structure-guided Two-phase Autoregressive Graph Exploration), a lightweight autoregressive generator designed for both requirements. **For scalability**, STAGE serializes each graph as a flat bit-level stream of sorted edges, encoding each edge by the binary indices of its endpoints and its attributes. The sequence length is $\Theta(|E| \log |V|)$, giving $O(|E| \log |V|)$ generation without ever materializing the adjacency matrix. Unlike BiGG's recursive decomposition, the stream is a single causal sequence, so the backbone is interchangeable; we use an LSTM on molecular benchmarks and Mamba (Gu and Dao, 2023) for long sequences. Bit-level streams are compact but highly sensitive to node order: different serializations of the same graph range from highly regular bit patterns to fragmented sequences that are difficult for autoregressive models to learn. STAGE therefore orders nodes by a breadth-first traversal from the maximum-degree node in which neighbors are enqueued by structural rank, computed from SIR-GN structural node embeddings (Joaristi and Serra, 2021). In plain BFS the enqueue order depends on arbitrary node identifiers, so one graph admits combinatorially many serializations; structural ranking resolves this for structurally distinguishable nodes and yields substantially more regular sequences. **For novelty**, STAGE trains in two phases (Figure 1b). Phase 1 trains on the data together with edge-perturbed copies regenerated every epoch, broadening the generator's support beyond the observed graphs. Phase 2 applies ReST-style self-training (Gulcehre et al., 2023): the generator samples candidates, retains those accepted by a selection criterion, and continues training on them. The default criterion is label-free: a Gaussian mixture fitted to SIR-GN embeddings of the training graphs, which accepts samples above a likelihood quantile and therefore applies to domains without a validity oracle. When an oracle exists, such as chemical validity and stability for molecules, it can replace the mixture. The two phases are

complementary: Phase 1 creates support and Phase 2 retains its plausible part. Ablations confirm that without perturbations novelty decays, and without refinement validity drops sharply.

We evaluate STAGE on six molecular benchmarks, QM9 (Ramakrishnan et al., 2014), ZINC250K (Irwin et al., 2012; Gómez-Bombarelli et al., 2018), MOSES (Polykovskiy et al., 2020), ANI-1x (Smith et al., 2020), QM7x (Hoja et al., 2021), and Transition1x (Schreiner et al., 2022), and on MalNet-Tiny. With label-free selection, STAGE achieves the highest joint validity–uniqueness–novelty (VUN) on QM9, QM7x, Transition1x, and ANI-1x (56.8% vs. 55.0% on QM9; 86.4% vs. 59.4% on QM7x), matches the strongest baseline on MOSES, and remains competitive on ZINC250K. With oracle selection, it reaches 60.9% VUN on QM9 and 95.1% on MOSES, against 55.0% and 85.4%. These gains come with lower distributional fidelity on MOSES, which we report alongside them (Section 5.1). On MalNet-Tiny, STAGE trains and generates in a regime where DiGress, DeFoG, and SparseDiff exhaust GPU memory.

Our contributions are:

- **Scalable, learnable serialization.** A flat bit-level edge stream with $O(|E| \log |V|)$ generation cost and an interchangeable causal backbone, made learnable by SIR-GN-ranked BFS ordering. Ablations show that ordering is decisive: random ordering collapses validity, and plain BFS is less stable than structure-guided BFS on harder datasets.
- **Novelty-oriented two-phase training.** Perturbation-based pretraining followed by ReST-style self-training with either a label-free embedding-space criterion or a domain oracle, with ablations isolating the role of each phase and of the selection criterion.
- **Evaluation across molecular and large non-molecular graphs.** Results on six molecular benchmarks and MalNet-Tiny with LSTM and Mamba backbones, together with a memory-scaling study showing where four recent generators exhaust GPU memory.

2 Related Work

Autoregressive and Sequence-Based Graph Generation Autoregressive graph generators construct graphs through a sequence of local decisions, such as adding nodes, predicting edges, or generating blocks of structure. Early models such as DeepGMG and GraphRNN use recurrent networks to generate graphs sequentially, with GraphRNN relying on BFS orderings to reduce the space of possible node permutations (Li et al., 2018; You et al., 2018). GRAN improves efficiency by generating nodes and edges in blocks, while BiGG exploits graph sparsity to avoid dense adjacency-matrix generation, reaching near-log-linear complexity through a recursive binary decomposition of the adjacency structure (Liao et al., 2019; Dai et al., 2020). Our method reaches a comparable asymptotic regime by a different route: rather than decomposing the adjacency structure, we emit a flat bit-level stream of sorted edges under a fixed structural ordering, which keeps the generator a single causal sequence model and makes the backbone interchangeable.

A central limitation of autoregressive graph generation is that graphs do not have a canonical linear order. As a result, the same graph may correspond to many different sequences, and the chosen serialization strongly affects both learning difficulty and generation quality. Recent work has revisited this problem with more compact representations and larger sequence models. GEEL (Jang et al., 2024) uses a gap-encoded edge-list representation to reduce sequence and vocabulary size, while G2PT (Chen et al., 2025b) uses graph tokenizations with a Transformer decoder. AutoGraph (Chen et al., 2025a) similarly casts graph generation as sequence modeling, flattening graphs into token sequences that are generated autoregressively with a Transformer decoder. This provides a particularly relevant comparison to our approach: both reduce graph generation to causal sequence modeling, but our method instead uses structure-guided bit-level edge serialization and lightweight LSTM or state-space backbones.

Ordering and Serialization Because graph likelihoods are permutation-invariant but sequence likelihoods are not, node ordering has become a central issue in autoregressive graph generation. Earlier methods rely on fixed traversal or heuristic orderings, such as BFS, DFS, degree-based orderings, or canonical ordering families. More recent work treats ordering as a modeling problem. Order Matters models the node sequence as a latent variable and uses variational inference to infer likely generation orders (Chen et al., 2021). Latent Sort learns continuous embeddings whose sorted

order induces a graph permutation (Bu et al., 2023). LO-ARM introduces a trainable state-dependent order policy that dynamically chooses the autoregressive order and optimizes a variational lower bound (Wang et al., 2025). Our ordering instead uses SIR-GN structural node embeddings as an explicit ranking signal, combining this structural ranking with a BFS constraint to produce a connectivity-aware serialization (Joaristi and Serra, 2021). This places our approach in the broader line of structural node representation methods, such as struc2vec, GraphWave, and SIR-GN, which represent nodes by structural role rather than only proximity (Ribeiro et al., 2017; Donnat et al., 2018; Joaristi and Serra, 2021).

Other approaches reduce order sensitivity without learning a full order: orderless regularization (Cohen-Karlik et al., 2024) penalizes recurrent hidden states that differ across valid flattenings of the same graph, while MAG (Belkadi et al., 2025) sidesteps next-node and next-edge ordering entirely by generating coarse-to-fine latent scales. In contrast, we regularize the induced sequence through a fixed structural ranking rather than learning or marginalizing over generation orders.

Diffusion and Hybrid Models A second family of graph generators avoids dependence on a fixed serialization by modeling graphs more holistically. GraphVAE, EDP-GNN, GDSS, GraphGDP, and DiGress do so through latent-variable, score-based, or diffusion-based formulations (Simonovsky and Komodakis, 2018; Niu et al., 2020; Jo et al., 2022; Huang et al., 2022; Vignac et al., 2023). More recent diffusion and flow-based models refine this line: DisCo and Cometh formulate diffusion in continuous time over discrete graph states, DeFoG uses discrete flow matching to decouple training from sampling, and SparseDiff targets scalability through sparsity-preserving diffusion (Xu et al., 2024; Siraudin et al., 2024; Qin et al., 2025a;b). These formulations introduce different scaling costs: dense generators may reason over node pairs, giving quadratic dependence on graph size, while diffusion-based generators require repeated graph updates across sampling steps. We quantify these practical scaling limitations in Appendix C, where several recent graph generators encounter GPU-memory limits within the graph-size regime of MalNet-Tiny.

Hybrid models combine autoregressive and diffusion-based ideas. GraphARM adds nodes and incident edges through an autoregressive diffusion process, while PARD defines a structural partial order and generates graphs block by block with a diffusion decoder for each conditional block (Kong et al., 2023; Zhao et al., 2024). These approaches reduce dependence on a single generation order but require graph decompositions, block schedules, or repeated denoising. Our method instead keeps the generator purely autoregressive and lightweight.

Novelty-Oriented Training Most graph generative models are trained to match the empirical training distribution through likelihood maximization, variational objectives, score matching, flow matching, or denoising losses. These objectives are effective for fidelity, but they do not necessarily encourage meaningful novelty. This tension is well known in molecular graph generation, where models are commonly evaluated using validity, uniqueness, novelty, and distributional metrics, which do not always move together. Goal-directed molecular generators do reward novelty or diversity explicitly, for example through diversity filters in reinforcement-learning pipelines (Blaschke et al., 2020), but they optimize task-specific scores rather than learning the data distribution; our setting is unconditional generation, where we are not aware of prior generators that target novelty by design.

Graph perturbations have also been widely used as augmentations in graph representation learning, for example through edge dropping in DropEdge and contrastive views in GraphCL (Rong et al., 2020; You et al., 2020). In contrast, we use edge additions and removals not to learn invariant representations, but to broaden the support of the autoregressive generator during pretraining.

Our refinement stage is inspired by Reinforced Self-Training (ReST), which iteratively generates candidate samples and uses selected samples to improve the model (Gulcehre et al., 2023), and is related to broader self-training and rejection-sampling fine-tuning procedures such as STaR and RAFT, where models are trained on filtered self-generated outputs (Zelikman et al., 2022; Dong et al., 2023). Unlike these methods, which rely on a task reward or correctness label, our default acceptance criterion is label-free: Phase 2 retains generated graphs according to their likelihood under a GMM fitted in graph embedding space, which extends to domains where no domain-specific validity oracle exists. Where such an oracle is available, the same loop accepts domain-defined criteria instead, and we report both settings.

3 METHODS

Given a dataset $\mathcal{D} = \{G_i\}_{i=1}^{N}$ of graphs $G = (V, E)$ with discrete node and edge attributes, we seek a generator that produces novel graphs consistent with the structural and attribute patterns of $\mathcal{D}$. Each graph is serialized under a node ordering $\pi(G)$ into a variable-length sequence $s(G) = (x_1, \ldots, x_T)$ and modeled autoregressively as $p(s(G)) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$. Because π determines the sequence the model must learn, its choice strongly affects learnability.

3.1 STRUCTURAL NODE ORDERING

We define a node ordering based on structural properties of the graph. The structural ranking induces a partial ordering over the nodes, since structurally equivalent nodes receive the same rank. Ties are resolved by the underlying input ordering. Our goal is not to compute a fully canonical total ordering, but rather to reduce the ambiguity of graph serialization by preferring node orders that are consistent with node structure.

A purely structure-based ranking, however, does not by itself guarantee that the serialization proceeds coherently within a connected component. Since our generation process is intended to construct one connected component at a time, we combine the structural ordering with a breadth-first search (BFS) traversal. In this way, structural information determines node priority, while BFS enforces a connectivity-aware exploration order.

Algorithm 1 gives the procedure. We compute SIR-GN node representations, rank nodes by them (ties broken by input index), and run BFS from the maximum-degree node of each connected component, enqueuing unvisited neighbors in rank order. In plain BFS, this enqueuing order depends on arbitrary node identifiers, so a single graph admits combinatorially many serializations, and a sequence model trained on such data must spread its capacity across them. Structural ranking removes this arbitrariness for distinguishable nodes; because parents are themselves enqueued in rank order, the effect carries through the entire traversal and grows with the structural diversity of the graph.

This ordering is particularly important in our setting because the downstream generator operates on a binary serialization, whose learnability depends strongly on the regularity of the induced sequence.

Algorithm 1 Structure-guided BFS ordering

Require: Graph $G = (V, E)$ with node attributes x_v, ordering π, index width b
Ensure: Node ordering π
$\{h_v\} \leftarrow \psi(G)$; $r(v) \leftarrow$ rank of h_v
$\pi \leftarrow [\,]$
for each connected component K of G **do**
  $Q \leftarrow [\arg\max_{v \in K} \deg(v)]$
  while $Q \neq \emptyset$ **do**
    $u \leftarrow \textsc{PopFront}(Q)$; append u to π
    for $w \in N(u)$ in increasing $r(w)$ **do**
      if $w \notin \pi$ and $w \notin Q$ **then**
        $\textsc{PushBack}(Q, w)$
      end if
    end for
  end while
end for
return π

3.2 GRAPH SERIALIZATION

Given the structural node ordering described above, each graph is converted into an ordered edge list, which is then flattened into a bit-level representation for autoregressive modeling. After reindexing nodes according to the structural order, the edge list is constructed so that edge pairs appear in strictly increasing lexicographic order. This reduces arbitrariness in the serialization and yields a more predictable sequence for the autoregressive model.

Algorithm 2 Bit-level edge serialization

Require: Graph $G = (V, E)$, ordering π, index width b
Ensure: Bit sequence $s(G)$
Relabel each $v \in V$ by its position in π
$s \leftarrow [\,]$
for all $(u, v, a) \in E$ in lexicographic order of (u, v), $u < v$ **do**
  $\beta_w \leftarrow \textsc{Bin}_b(w) \,\|\, \textsc{Enc}(x_w) \,\|\, \textsc{Enc}(a),\ w \in \{u, v\}$
  Append $\beta_u \,\|\, \beta_v$ to s
end for
return s

Each node appearing in an edge is represented through a fixed binary encoding. The representation of an edge is then obtained by combining the binary encodings of its two endpoints together with the corresponding edge attributes. Concatenating these bit representations over the ordered edge list

yields a variable-length binary sequence $s(G) = (x_1, \ldots, x_T)$, which serves as the target sequence for the autoregressive model.

The autoregressive decoder processes this flattened bit sequence one step at a time. In addition to the previously generated bit, the model receives a positional encoding that indicates which bit of the current edge representation is being predicted. This provides the decoder with local context about its position inside the edge encoding and helps stabilize sequence generation.

Although such binary encodings are often viewed as difficult to model due to their sensitivity to serialization choices, we find that the combination of structure-guided ordering and phased training makes this representation effective in practice.

3.3 Phase 1: Pretraining with Perturbed Graphs

In the first phase, the autoregressive model is trained with teacher forcing on both the original training graphs and dynamically generated perturbations. At each epoch, we construct one perturbed graph per training example by randomly adding and removing edges, and serialize it using the same structure-guided ordering and bit-level encoding as the original graphs. We treat the edge-perturbation rate as a training hyperparameter and select it empirically to introduce structural variation without excessively distorting the original graph distribution. We use a single setting, $p_{drop} = 0.15$ and $p_{add} = 0.15$, across all datasets. We do not derive the rate analytically, and characterizing the optimal rate as a function of graph size or domain is left to future work. The model is therefore trained on a mixture of observed and structurally modified graphs.

The purpose of this augmentation is exploratory rather than simply increasing the training-set size. By exposing the generator to nearby graph configurations beyond the empirical training set, Phase 1 broadens the sequence distribution encountered during training. Phase 2 subsequently provides the exploitative component by retaining generated samples according to an embedding-space notion of plausibility.

3.4 Phase 2: ReST-style Refinement with GMM-based Filtering

After pretraining, we refine the generator through iterative self-training inspired by ReST. At each iteration, the current model generates a set of candidate graphs, which are filtered before being used for subsequent teacher-forced training. During this phase, the original and perturbed Phase-1 graphs are no longer used.

Algorithm 3 Phase 2: ReST-style refinement

Require: Pretrained p_θ, encoder ϕ, data $\mathcal{D}$, iterations K, candidates M, quantile q

Fit a C-component GMM on $\{\phi(G) : G \in \mathcal{D}\}$
$\tau \leftarrow q$-quantile of $\{\log p_{\mathrm{GMM}}(\phi(G)) : G \in \mathcal{D}\}$
for $k = 1, \ldots, K$ **do**
  Sample M sequences from p_θ; decode well-formed ones into distinct $\{\widehat{G}_j\}$
  $\mathcal{C}_k \leftarrow \{\widehat{G}_j : \log p_{\mathrm{GMM}}(\phi(\widehat{G}_j)) \geq \tau\}$ ▷ or oracle
  Train p_θ by teacher forcing on $\{s(\widehat{G}) : \widehat{G} \in \mathcal{C}_k\}$ only
end for
return p_θ

To provide a domain-independent selection criterion, we fit a Gaussian Mixture Model (GMM) to embeddings of the original training graphs. Each generated graph is mapped into the same embedding space and scored under the GMM; samples below a quantile-based likelihood threshold are discarded, while the remaining samples are retained for continued training. This provides the exploitative component of the framework, selectively refining the exploratory generator toward regions considered plausible in the learned embedding space.

4 Experimental Setup

Datasets. We evaluate on six molecular datasets—QM9, ZINC250K, MOSES, ANI-1x, QM7x, Transition1x—and on MalNet-Tiny, a non-molecular benchmark of Android malware function-call graphs. For the quantum-chemical datasets, which may contain multiple conformations of the same molecule, we canonicalize molecules by SMILES with RDKit (Landrum, 2025) and remove duplicate molecular graphs before training and evaluation so that novelty is measured at the graph level. For MalNet-Tiny, we consider one representative family as an initial test of non-molecular generation and scalability to substantially larger sparse graphs.

Metrics. For molecular datasets, we report Validity, Uniqueness, and Novelty following standard practice. Validity is the fraction of chemically valid generated molecules, Uniqueness the fraction of non-duplicate samples, and Novelty the fraction not present in the training set. On QM9, we additionally report Atom and Molecule Stability. We also report VUN, the fraction of generated samples simultaneously satisfying validity, uniqueness, and novelty. Where applicable, we report standard distributional metrics such as SNN and scaffold similarity (Polykovskiy et al., 2020). Because some metrics are conditional on others (e.g., MOSES Novelty is evaluated over unique valid molecules), we interpret them jointly rather than in isolation. For MalNet-Tiny, we use representation- and classifier-based analogues of these metrics, defined in Section 5.3.

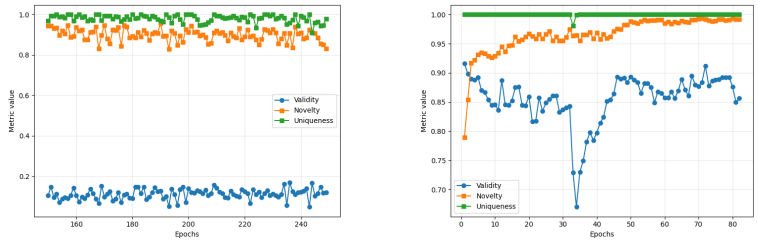
Baselines. We compare against representative autoregressive, diffusion, flow-based, and hybrid graph generators: PARD, DiGress, DeFoG, AutoGraph, and SparseDiff. We additionally perform ablations of the proposed ordering and two-phase training procedure, including random and BFS orderings, training without Phase-1 perturbations, and training without Phase-2 refinement.

5 Results

5.1 Main Results

Tables 1–3 compare our method with recent graph generators on QM9, ZINC250K, and MOSES. On QM9, the labeled variant achieves 60.90% VUN@10k, compared with 54.99% for SparseDiff, while maintaining 97.88% validity, 97.45% uniqueness, and 96.25% molecular stability. On MOSES, it reaches 95.07% VUN@25k, compared with 85.38% for DeFoG. On ZINC250K, both variants maintain high validity, uniqueness, and novelty, with the unlabeled variant reaching 98.29% VUN@10k. The MOSES results also expose a fidelity–exploration tradeoff: the labeled variant achieves higher novelty and scaffold similarity, but lower Filters and SNN than several baselines, whereas the unlabeled variant reaches an SNN of 0.56.

5.2 Ablation Summary



(a) Fully random ordering. (b) Plain BFS, no structural guidance.

Figure 2: **Node-ordering ablation on Transition1x.** Random ordering collapses validity; plain BFS recovers most of it but remains unstable.

We summarize the main ablation findings here and provide full curves and discussion in Appendix F. Removing ReST after perturbed pretraining leads to exploratory but poorly controlled generation, while removing both perturbations and ReST produces a conservative model with declining novelty. These results support the exploration–exploitation interpretation of the two-phase procedure.

Ordering is also critical. Fully random ordering causes a severe validity collapse (Figure 2a), showing that bit-level generation is highly sensitive to serialization. Plain BFS recovers much of the lost structure but is less stable than the full structure-guided traversal on harder datasets (Figure 2b). Appendix D further shows that the proposed structural ranking yields more distinct node ranks than degree and PARD-style orderings.

Finally, removing perturbed graphs from Phase 1 makes refinement predominantly exploitative, further supporting the role of Phase-1 perturbations in encouraging exploration.

Table 1: **QM9 with explicit hydrogens.** VUN@10k denotes the percentage of generated molecules that are simultaneously valid, unique, and novel. Higher is better.

Method	Val. ↑	Uniq. ↑	Nov. ↑	Atom. ↑	Mol. ↑	VUN@10k ↑
PARD	96.73	95.99	37.23	98.40	86.10	34.57
DiGress	95.40	97.60	33.40	98.10	79.80	31.10
DeFoG	**98.23**	96.28	36.00	98.99	91.00	34.04
AutoGraph	97.70	96.70	45.50	98.60	87.30	42.99
SparseDiff	97.21	**97.74**	57.88	99.60	94.73	54.99
STAGE-L (VUNS)	97.88	97.45	**63.85**	**99.71**	**96.25**	**60.90**
STAGE-U	97.76	92.07	63.06	98.65	82.28	56.76

Table 2: **ZINC250K.** Higher is better.

Method	Val. ↑	Uniq. ↑	Nov. ↑	VUN@10k ↑
PARD	98.68	99.95	99.69	98.33
DiGress	78.64	99.99	**99.99**	78.62
DeFoG	**99.13**	**100.0**	99.96	**99.09**
AutoGraph	87.68	**100.0**	95.50	83.73
SparseDiff	83.67	99.98	99.95	83.61
STAGE-L (VUN)	97.19	99.28	99.94	96.43
STAGE-U	98.32	99.99	99.98	98.29

Table 3: **MOSES.** Higher is better. '–' denotes metrics not reported.

Method	Val. ↑	Uniq. ↑	Nov. ↑	Filters ↑	SNN ↑	Scaf. ↑	VUN@25k ↑
PARD	86.80	100.00	78.20	**99.0**	**0.56**	2.2	67.87
DiGress	85.70	100.00	95.00	97.1	0.52	14.8	81.42
DeFoG	92.80	99.90	92.10	98.9	0.55	14.4	85.38
AutoGraph	87.40	100.00	85.90	98.6	0.55	–	75.08
SparseDiff	84.70	100.00	95.10	97.0	0.52	15.5	80.55
STAGE-L (VUN)	**97.81**	98.47	**98.71**	87.64	0.41	**21.8**	**95.07**
STAGE-U	92.56	98.08	94.16	94.50	**0.56**	10.1	85.48

Table 4: **Extended molecular benchmarks.** Higher is better.

Dataset	Method	Val. ↑	Uniq. ↑	Nov. ↑	VUN ↑
ANI-1x	PARD	84.18	98.17	89.70	74.13
	DiGress	96.48	99.02	87.74	83.82
	DeFoG	96.76	98.71	87.25	83.33
	AutoGraph	**97.24**	**99.22**	80.97	78.12
	SparseDiff	95.51	99.09	88.16	83.43
	STAGE-U	95.80	94.65	**99.83**	**90.52**
QM7x	PARD	97.99	96.13	49.24	46.38
	DiGress	98.00	96.72	62.63	59.36
	DeFoG	98.90	95.94	50.78	48.18
	AutoGraph	**98.93**	**97.09**	53.79	51.66
	SparseDiff	97.55	96.99	59.26	56.07
	STAGE-U	96.42	92.38	**96.94**	**86.35**
Transition1x	PARD	92.82	98.15	75.09	68.41
	DiGress	96.07	98.89	82.00	77.90
	DeFoG	93.57	97.84	83.01	75.99
	AutoGraph	97.26	**99.14**	77.93	75.14
	SparseDiff	**97.38**	98.29	88.43	84.64
	STAGE-U	92.86	93.8	**98.94**	**86.18**

5.3 MALNET-TINY TARGET-FAMILY EVALUATION.

Table 5: **Conditional generation on MalNet-Tiny.** Validity is the fraction of generated graphs classified as the target family by an external classifier trained only on real graphs. Uniqueness and Novelty are computed in SIR-GN embedding space: Uniqueness is the fraction of valid generations with distinct signatures, and Novelty is the fraction of unique valid signatures not observed in the real target-family training set. Higher is better.

Graphs / Method	Validity ↑	Uniqueness ↑	Novelty ↑	VUN@100 ↑
Real Family 0	88.0	–	–	–
Real Families 1–4	15.0	–	–	–
AutoGraph	**85.0**	**77.6**	75.8	49.99
STAGE (Phase 1)	74.0	68.9	56.9	29.01
STAGE (Tempo 1)	64.0	68.8	59.1	26.02
STAGE (Tempo 2)	<u>81.0</u>	67.9	81.8	44.99
STAGE (Tempo 3)	70.0	<u>72.9</u>	90.2	46.03
STAGE (Tempo 4)	77.0	72.7	**96.4**	**53.96**

For MalNet-Tiny, we define Validity as the fraction of generated graphs classified as the target family by the external classifier. To define representation-level Uniqueness and Novelty, we mean-pool the frozen SIR-GN node embeddings and quantize each embedding dimension, yielding a discrete embedding signature. Uniqueness is the fraction of valid generations with distinct signatures, while Novelty is the fraction of unique valid signatures not observed among real target-family training graphs. Quantization avoids treating negligible floating-point differences as distinct representations.

Scalability. MalNet-Tiny also exposes practical memory limitations of recent graph generators. On a GH200 with 95 GiB available GPU memory, DiGress exceeds memory at 1,750 nodes and SparseDiff at 2,999 nodes under the evaluated training configurations, while PARD inference fails at 5,000 nodes. Approximately 70% of our MalNet-Tiny split exceeds 1,500 nodes, with a median training size of 2,272 nodes. Full configurations and stress-test results are reported in Appendix C.

6 CONCLUSION

We introduced a lightweight autoregressive graph generator based on structure-guided bit-level edge serialization and two-phase refinement. Combining SIR-GN structural rankings with BFS traversal produces regular sparse edge sequences modeled at $O(|E| \log |V|)$ complexity, while perturbation-based pretraining and ReST-style GMM filtering balance exploration and plausibility. Across molecular benchmarks, the framework achieves competitive joint validity, uniqueness, and novelty, including 60.90% VUN@10k on QM9 and 95.07% VUN@25k on MOSES. Experiments on MalNet-Tiny further demonstrate generation in a substantially larger sparse-graph regime where several recent graph generators encounter practical memory limitations. These results highlight graph serialization as an important design axis for scalable, novelty-oriented autoregressive generation.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the writing and for coding assistance. We have not used generative AI tools for experimental design or interpretation of results. All AI-assisted content was reviewed and verified by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Samuel Belkadi, Steve Hong, and Marian Chen. Make autoregressive great again: Diffusion-free graph generation with next-scale prediction. *arXiv preprint arXiv:2503.23612*, 2025. URL https://arxiv.org/abs/2503.23612.

Thomas Blaschke, Ola Engkvist, Jürgen Bajorath, and Hongming Chen. Memory-assisted reinforcement learning for diverse molecular de novo design. *Journal of Cheminformatics*, 12:68, 2020.

Jie Bu, Kazi Sajeed Mehrab, and Anuj Karpatne. Let there be order: Rethinking ordering in autoregressive graph generation. *arXiv preprint arXiv:2305.15562*, 2023. URL https://arxiv.org/abs/2305.15562.

Dexiong Chen, Markus Krimmel, and Karsten Borgwardt. Flatten graphs as sequences: Transformers are scalable graph generators. In *Advances in Neural Information Processing Systems*, volume 38, 2025a. URL https://arxiv.org/abs/2502.02216.

Xiaohui Chen, Xu Han, Jiajing Hu, Francisco J. R. Ruiz, and Liping Liu. Order matters: Probabilistic modeling of node sequence for graph generation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1630–1639. PMLR, 2021. URL https://proceedings.mlr.press/v139/chen21j.html.

Xiaohui Chen, Yinkai Wang, Jiaxing He, Yuanqi Du, Soha Hassoun, Xiaolin Xu, and Li-Ping Liu. Graph generative pre-trained transformer. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*. PMLR, 2025b. URL https://proceedings.mlr.press/v267/chen25bl.html.

Edo Cohen-Karlik, Eyal Rozenberg, and Daniel Freedman. Overcoming order in autoregressive graph generation. *arXiv preprint arXiv:2402.03387*, 2024. URL https://arxiv.org/abs/2402.03387.

Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable deep generative modeling for sparse graphs. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2302–2312. PMLR, 2020. URL https://proceedings.mlr.press/v119/dai20b.html.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023. URL https://arxiv.org/abs/2304.06767.

Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1320–1329. ACM, 2018. doi: 10.1145/3219819.3220025. URL https://doi.org/10.1145/3219819.3220025.

Scott Freitas, Yuxiao Dong, Joshua Neil, and Duen Horng Chau. A large-scale database for graph representation learning. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023. URL https://arxiv.org/abs/2308.08998.

Johannes Hoja, Leonardo Medrano Sandonas, Brian G. Ernst, Alvaro Vazquez-Mayagoitia, Robert A. DiStasio Jr., and Alexandre Tkatchenko. QM7-X, a comprehensive dataset of quantum-mechanical properties spanning the chemical space of small organic molecules. *Scientific Data*, 8:43, 2021.

Han Huang, Leilei Sun, Bowen Du, Yanjie Fu, and Weifeng Lv. GraphGDP: Generative diffusion processes for permutation invariant graph generation. In *2022 IEEE International Conference on Data Mining*, pages 201–210. IEEE, 2022. doi: 10.1109/ICDM54844.2022.00030. URL https://arxiv.org/abs/2212.01842.

John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A free tool to discover chemistry for biology. *Journal of Chemical Information and Modeling*, 52 (7):1757–1768, 2012.

Yunhui Jang, Seul Lee, and Sungsoo Ahn. A simple and scalable representation for graph generation. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=nO344avRib.

Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 10362–10383. PMLR, 2022. URL https://proceedings.mlr.press/v162/jo22a.html.

Mikel Joaristi and Edoardo Serra. SIR-GN: A fast structural iterative representation learning approach for graph nodes. In *ACM Transactions on Knowledge Discovery from Data*, volume 15, pages 1–39. ACM, 2021. doi: 10.1145/3450315. URL https://doi.org/10.1145/3450315.

Lingkai Kong, Jiaming Cui, Haotian Sun, Yuchen Zhuang, B. Aditya Prakash, and Chao Zhang. Autoregressive diffusion model for graph generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 17391–17408. PMLR, 2023. URL https://proceedings.mlr.press/v202/kong23b.html.

Greg Landrum. RDKit: Open-source cheminformatics. https://www.rdkit.org, 2025. Version 2025.03.2.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hy1d-ebAb.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, William L. 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. URL https://proceedings.neurips.cc/paper/2019/hash/d0921d442ee91b896ad95059d13df618-Abstract.html.

Chenhao Niu, Yang Song, Jiaming Song, Shengjia Zhao, Aditya Grover, and Stefano Ermon. Permutation invariant graph generation via score-based generative modeling. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 4474–4484. PMLR, 2020. URL https://proceedings.mlr.press/v108/niu20a.html.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, et al. Molecular sets (MOSES): A benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11:565644, 2020.

Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. DeFoG: Discrete flow matching for graph generation. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=KPRIwWhqAZ.

Yiming Qin, Clement Vignac, and Pascal Frossard. SparseDiff: Sparse discrete diffusion for scalable graph generation. *Transactions on Machine Learning Research*, 2025b. URL https://openreview.net/forum?id=kuJ3lpxnVC.

Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1:140022, 2014.

Leonardo F. R. Ribeiro, Pedro H. P. Saverese, and Daniel R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394. ACM, 2017. doi: 10.1145/3097983.3098061. URL https://doi.org/10.1145/3097983.3098061.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. DropEdge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=Hkx1qkrKPr.

Mathias Schreiner, Arghya Bhowmik, Tejs Vegge, Jonas Busk, and Ole Winther. Transition1x – a dataset for building generalizable reactive machine learning potentials. *Scientific Data*, 9:779, 2022.

Martin Simonovsky and Nikos Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *Artificial Neural Networks and Machine Learning – ICANN 2018*, pages 412–422. Springer, 2018. URL https://arxiv.org/abs/1802.03480.

Antoine Siraudin, Fragkiskos D. Malliaros, and Christopher Morris. Cometh: A continuous-time discrete-state graph diffusion model. *arXiv preprint arXiv:2406.06449*, 2024. URL https://arxiv.org/abs/2406.06449.

Justin S. Smith, Roman Zubatyuk, Benjamin Nebgen, Nicholas Lubbers, Kipton Barros, Adrian E. Roitberg, Olexandr Isayev, and Sergei Tretiak. The ANI-1ccx and ANI-1x data sets, coupled-cluster and density functional theory properties for molecules. *Scientific Data*, 7:134, 2020.

Clement 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. URL https://openreview.net/forum?id=UaAD-Nu86WX.

Zhe Wang, Jiaxin Shi, Nicolas Heess, Arthur Gretton, and Michalis K. Titsias. Learning-order autoregressive models with application to molecular graph generation. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=EY6pXIDi3G.

Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state continuous-time diffusion for graph generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/91813e5ddd9658b99be4c532e274b49c-Abstract-Conference.html.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5708–5717. PMLR, 2018. URL https://proceedings.mlr.press/v80/you18a.html.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/3fe230348e9a12c13120749e3f9fa4cd-Abstract.html.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Lingxiao Zhao, Xueying Ding, and Leman Akoglu. PARD: Permutation-invariant autoregressive diffusion for graph generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0d89cf183391e12063cb63ff0d75ed95-Abstract-Conference.html.

A Novelty–Stability Selection Ablation

We further examine the trade-off induced by the selection criterion used during refinement. Selecting samples based on novelty alone encourages aggressive exploration, but can substantially reduce molecular stability. Incorporating stability into the selection criterion preserves most of the novelty gain while producing substantially more stable molecules. Tables 6 and 7 report the corresponding results.

B Complexity Analysis

We analyze the per-graph time complexity of the proposed pipeline. Let $n = |V|$ and $m = |E|$. We focus on sparse graphs with $m = O(n)$, as in the molecular and structural benchmarks considered in this work, and treat the LSTM hidden size, SIR-GN iteration count, edge-attribute width, and graph embedding dimension as constants.

Ordering. Computing structural node representations with SIR-GN requires linear work in the graph size per iteration, hence $O(n + m)$ for a constant number of iterations. Ranking the resulting node representations costs $O(n \log n)$. The structure-guided BFS traversal uses a FIFO queue and visits each node and edge a constant number of times; structural ranking is used only to order neighbors before they are enqueued. Sorting neighbor lists contributes $\sum_v O(\deg(v) \log \deg(v)) \leq O(m \log n)$. Thus, ordering costs $O(n \log n + m \log n)$, which is $O(n \log n)$ for sparse graphs.

Serialization. Each node index requires $b = \lceil \log_2 n \rceil$ bits, so each edge contributes $O(\log n)$ bits to the sequence. Lexicographic edge sorting costs $O(m \log m)$, and writing the bit sequence costs $O(m \log n)$. The resulting sequence length is $T = \Theta(m \log n)$, which is $O(n \log n)$ for sparse graphs.

Sequence model. A forward or sampling pass through the LSTM costs $O(Td^2)$ for hidden size d. Treating d as fixed and using $T = O(n \log n)$ on sparse graphs, training and autoregressive generation are near-log-linear per graph. This avoids constructing or attending over the full $n \times n$ adjacency matrix, a common bottleneck in dense graph generators. For the Mamba variant, the causal state-space backbone has linear dependence on sequence length for fixed hidden dimension, so the sequence-length dependence remains proportional to $T = O(m \log n)$.

Phase 1. At each epoch, the model trains on the original graphs and one perturbed graph per original graph. The perturbation step removes $O(p_{\text{drop}} n)$ edges and adds $O(p_{\text{add}} n)$ candidate edges for constant perturbation rates. Since added edges are sampled by rejection rather than by constructing the full non-edge set, this step is linear in expectation for sparse graphs. The dominant per-epoch cost is therefore $O(|\mathcal{D}|\, n_{\max} \log n_{\max})$.

Phase 2. Each ReST iteration samples M candidate graphs, scores them under the fitted GMM, and trains on the accepted subset $\mathcal{C}_k$. Generating one candidate costs $O(n_{\max} \log n_{\max})$. Computing a linear-time graph embedding adds $O(n_{\max})$, and GMM scoring with C components costs $O(C)$ for fixed embedding dimension. Since $|\mathcal{C}_k| \leq M$, the dominant term per iteration is $O(M\, n_{\max} \log n_{\max})$.

Fitting the GMM. The GMM is fit once before Phase 2 on training graph embeddings. Standard EM with C components and $|\mathcal{D}|$ samples costs $O(I\, C\, |\mathcal{D}|)$ for I iterations when the embedding dimension is fixed. This is a one-time preprocessing cost.

Comparison with prior work. Table 8 summarizes representative generation costs. Dense adjacency-based autoregressive and diffusion models scale at least quadratically in graph size, with diffusion or flow-based methods also requiring multiple sampling steps. Sparse autoregressive models avoid this by exploiting edge sparsity. Our method follows this sparse regime through a bit-level edge stream, giving $O(m \log n)$ generation cost in general and $O(n \log n)$ on sparse graphs.

Table 6: Ablation of novelty (N) and stability (S) selection on QM9 after 20 Phase-2 epochs. All methods include validity (V) and uniqueness (U) selection.

Method	Val. ↑	Uniq. ↑	Nov. ↑	Atom Stab. ↑	Mol. Stab. ↑
VU	99.21	**92.95**	98.67	80.57	13.88
VUN	**99.74**	92.92	**99.94**	71.48	2.77
VUS	99.67	88.20	87.02	**99.96**	**99.56**
VUNS	99.20	88.55	97.89	99.89	98.87

Table 7: Evolution of novelty and molecular stability during Phase-2 training. Novelty selection rapidly increases exploration, while stability selection prevents the associated collapse in molecular stability.

	Epoch 1		Epoch 3		Epoch 10		Epoch 20	
Method	Nov.	Mol. Stab.	Nov.	Mol. Stab.	Nov.	Mol. Stab.	Nov.	Mol. Stab.
VU	28.90	83.21	38.92	80.02	73.98	55.61	98.67	13.88
VUN	43.30	78.85	96.95	7.84	99.65	3.78	99.94	2.77
VUS	25.98	92.77	32.66	97.12	59.11	98.56	87.02	99.56
VUNS	39.10	92.62	63.85	96.25	92.11	98.26	97.89	98.87

Table 8: Per-graph generation complexity of representative graph generative models on sparse graphs ($m = O(n)$). T_{steps} denotes the number of denoising, diffusion, or flow-matching sampling steps. Our general cost is $O(m \log n)$; the table reports the sparse case.

Family	Representative method	Generation cost
Dense diffusion / flow	DiGress, GDSS, DeFoG	$O(T_{\text{steps}} \cdot n^2)$
Hybrid AR–diffusion	PARD, GraphARM	$O(T_{\text{steps}} \cdot n^2)$
Dense AR	GraphRNN, GRAN	$O(n^2)$
Sparse AR	BiGG	$O((n+m)\log n)$
Sparse AR bit stream	**STAGE**	$O(n \log n)$

C Competitor Scalability on MalNet-Tiny

DiGress. We evaluated the original DiGress Graph Transformer with batch size 1 on an NVIDIA GH200 with 95 GiB of available GPU memory. A forward/backward pass required 83.64 GiB at 1,500 nodes and exceeded available memory at 1,750 nodes. 69.8% (698/1,000) of graphs in our MalNet-Tiny splits contain more than 1,500 nodes, including 69.9%, 71.0%, and 69.0% of the training, validation, and test sets, respectively. Moreover, the median training graph contains 2,272 nodes. The benchmark excludes computation of DiGress's spectral and cycle-count auxiliary features and therefore provides a favorable lower bound on the complete training pipeline's memory requirements.

DeFoG. DeFoG was not benchmarked on the GH200 because its reference environment targets AMD64/x86-64, whereas our GH200 host uses ARM64/aarch64. We therefore evaluated the DeFoG graph-transformer backbone on an NVIDIA RTX 4090 (23.64 GiB) using its configured architecture (5 layers, hidden node/edge dimensions 256/64, 8 attention heads), batch size 1, and FP32. A complete forward/backward training step succeeded for a 750-node graph, requiring 20.97 GiB peak allocated GPU memory. At 775 nodes, the forward pass succeeded (22.21 GiB peak), but the backward pass produced CUDA OOM. Thus, the observed training-memory limit was between 750 and 775 nodes. Approximately 70% of MalNet-Tiny graphs exceed 1,500 nodes, making training on the dataset infeasible under this setup.

SparseDiff. Exhibited substantially better scalability than dense diffusion baselines, successfully completing forward and backward passes on a 2,000-node MalNet-Tiny graph using 86.8 GiB peak GPU memory on an NVIDIA GH200 (95 GiB available). However, under the evaluated configuration ($lambda$=0.5, batch size 1, FP32), a 2,999-node graph exceeded available memory during the

Table 9: Ordering discriminativeness measured by the average number of distinct structural embeddings/ranks. Higher values indicate fewer ties in the structural ranking before BFS tie-breaking. Values for PARD and our orderings are reported for depths 1–5.

Order	Depth	QM9	ZINC250K	MOSES	ANI-1x	QM7x	Transition1x
Avg. nodes	–	18.07	23.09	21.60	16.57	15.02	13.62
Degree	–	3.57	3.27	3.21	3.61	3.58	3.57
PARD ordering	1	4.32	5.21	5.20	4.15	3.87	3.67
PARD ordering	2	5.59	7.57	7.28	5.61	5.07	4.73
PARD ordering	3	7.02	11.25	10.56	6.84	5.71	5.49
PARD ordering	4	7.47	12.56	11.65	7.29	5.88	5.63
PARD ordering	5	7.52	13.07	12.01	7.39	5.89	5.64
STAGE, no attr.	1	9.12	12.77	15.23	10.11	8.77	9.90
STAGE, no attr.	2	12.48	18.29	19.02	12.49	10.87	9.90
STAGE, no attr.	3	13.50	20.20	19.69	13.14	11.14	10.16
STAGE, no attr.	4	13.62	20.61	19.80	13.26	11.15	10.17
STAGE, no attr.	5	13.63	20.70	19.82	13.28	11.15	10.17
STAGE, attr.	1	9.45	17.53	16.47	10.34	8.91	8.15
STAGE, attr.	2	12.67	21.20	19.32	12.66	11.05	10.01
STAGE, attr.	3	13.57	21.82	19.80	13.26	11.28	10.26
STAGE, attr.	4	13.68	21.93	19.88	13.37	11.29	10.27
STAGE, attr.	5	13.68	21.95	19.89	13.38	11.29	10.27

forward pass. At this setting, the sampled query set grows with the number of possible node pairs: the 2,999-node graph required approximately 2.25M query edges and a 4.50M-edge computational graph.

PARD. We evaluated the PARD PPGN-Transformer backbone using its configured architecture (8 layers, hidden node/edge dimensions 256/64, 8 attention heads) with batch size 1 and FP32 on an NVIDIA GH200 with 95 GiB of available GPU memory. PARD completed inference at 1,500 nodes using 7.85 GiB peak allocated GPU memory, but memory requirements increased rapidly with graph size, reaching 13.83 GiB at 2,000 nodes, 32.63 GiB at 3,000 nodes, 57.97 GiB at 4,000 nodes, and 81.72 GiB at 4,750 nodes. At 5,000 nodes, inference exceeded the available GPU memory, reaching 84.79 GiB peak allocated memory before failing on an additional 23.84 GiB allocation. Thus, the observed inference-memory limit on the 95-GiB GH200 lies between 4,750 and 5,000 nodes. Despite PARD's autoregressive block decomposition, its dense PPGN-Transformer backbone therefore exhibits severe memory growth with graph size. This benchmark evaluates only the neural backbone in inference mode and consequently provides a favorable lower bound on the memory requirements of the complete training pipeline.

D Additional Ordering Analysis

To further characterize the proposed structural ordering, we measure the average number of distinct node embeddings induced by different structural ranking schemes. A larger value indicates that the ordering signal separates more nodes before BFS tie-breaking, reducing the number of structurally tied nodes during serialization. Table 9 compares degree ordering, the PARD structural ordering at depths 1–5, and our SIR-GN-based ordering with and without node/edge attributes. Across datasets, our ordering produces substantially more distinct embeddings than degree and PARD-style ordering, indicating a more discriminative structural ranking signal.

The gap is largest on ZINC250K and MOSES, where degree produces only about three distinct ranks on average, while our attribute-aware depth-5 ordering separates roughly 22 and 20 distinct embeddings, respectively. This suggests that the proposed structural ranking produces a much finer ordering signal than degree and PARD-style structural neighborhoods. Importantly, this analysis does not by itself prove improved generation quality; rather, it supports the mechanism behind the method by showing that the ordering used for serialization contains substantially more discriminative structural information.

E ADDITIONAL ALGORITHMS

This appendix provides detailed pseudocode for structure-guided ordering (Algorithm 4, a detailed version of Algorithm 1), bit-level serialization, edge perturbation, and the two training phases.

Algorithm 4 Structure-Guided BFS Ordering (STRUCTUREGUIDEDBFS)

Require: Graph $G = (V, E)$ with node and edge attributes, frozen SIR-GN encoder ψ
Ensure: Node ordering π over V

$\{h_v\}_{v \in V} \leftarrow \psi(G)$
Sort V by h_v in descending lexicographic order; let $r(v)$ be the position of v ▷ $r(v) = 1$ is highest-ranked; ties by input node index
$\pi \leftarrow [\,]$; $visited \leftarrow \emptyset$
for all connected components K of G **do**
    $v^* \leftarrow \arg\max_{v \in K} \deg(v)$ ▷ BFS root: max-degree node
    $Q \leftarrow [v^*]$; $inQueue \leftarrow \{v^*\}$
    while $Q \neq \emptyset$ **do**
        $u \leftarrow \text{POPFRONT}(Q)$; $inQueue \leftarrow inQueue \setminus \{u\}$
        Append u to π; $visited \leftarrow visited \cup \{u\}$
        for all $w \in \mathcal{N}(u)$ in increasing order of $r(w)$ **do**
            if $w \notin visited$ **and** $w \notin inQueue$ **then**
                $\text{PUSHBACK}(Q, w)$; $inQueue \leftarrow inQueue \cup \{w\}$
            end if
        end for
    end while
end for
return π

Algorithm 5 Bit-Level Edge Serialization

Require: Graph $G = (V, E)$ with node attributes x_v and edge attributes, ordering π, index width b
Ensure: Bit sequence $s(G) = (x_1, \ldots, x_T)$

Relabel each $v \in V$ by its position in π
$E_{\text{sort}} \leftarrow$ edges sorted lexicographically by (u, v) with $u < v$
$s \leftarrow [\,]$
for all $(u, v, a) \in E_{\text{sort}}$ **do**
    $\beta_u \leftarrow \text{BINARY}(u, b) \,\|\, \text{ENCODENODE}(x_u) \,\|\, \text{ENCODEATTR}(a)$
    $\beta_v \leftarrow \text{BINARY}(v, b) \,\|\, \text{ENCODENODE}(x_v) \,\|\, \text{ENCODEATTR}(a)$
    Append $\beta_u \,\|\, \beta_v$ to s
end for
return s

Algorithm 6 Edge-Level Graph Perturbation

Require: Graph $G = (V, E)$, drop rate p_{drop}, add rate p_{add}
Ensure: Perturbed graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$

$\widetilde{G} \leftarrow \text{copy}(G)$
$k_{\text{drop}} \leftarrow \min(\lfloor p_{\text{drop}}|V| \rfloor, |E|)$
$E_{\text{rem}} \leftarrow$ uniformly sample k_{drop} edges from E
Remove all edges in E_{rem} from $\widetilde{G}$
$k_{\text{add}} \leftarrow \lfloor p_{\text{add}}|V| \rfloor$
$a \leftarrow 0$
while $a < k_{\text{add}}$ **do**
    Sample distinct nodes $u, v \in \widetilde{V}$ uniformly at random
    if $(u, v) \notin \widetilde{E}$ **then**
        Add edge (u, v) to $\widetilde{G}$ with the default or empirical edge attribute
        $a \leftarrow a + 1$
    end if
end while
Remove isolated nodes from $\widetilde{G}$
return $\widetilde{G}$

Algorithm 7 Phase 1: Exploration Pretraining with Perturbed Graphs

Require: Training graphs $\mathcal{D}$, precomputed sequences $\{s(G) : G \in \mathcal{D}\}$, epochs E_1, perturbation rates $p_{\text{drop}}, p_{\text{add}}$
Ensure: Pretrained generator p_θ

Initialize generator parameters θ
for $e = 1, \ldots, E_1$ **do**
    $\widetilde{\mathcal{D}}_e \leftarrow \emptyset$
    for all $G \in \mathcal{D}$ **do**
        $\widetilde{G} \leftarrow \textsc{Perturb}(G, p_{\text{drop}}, p_{\text{add}})$
        $\widetilde{\mathcal{D}}_e \leftarrow \widetilde{\mathcal{D}}_e \cup \{\widetilde{G}\}$
    end for
    $\widetilde{\mathcal{S}}_e \leftarrow \{\textsc{BitSerialize}(\widetilde{G}, \textsc{StructureGuidedBFS}(\widetilde{G})) : \widetilde{G} \in \widetilde{\mathcal{D}}_e\}$
    $\mathcal{S}_e \leftarrow \{s(G) : G \in \mathcal{D}\} \cup \widetilde{\mathcal{S}}_e$
    for all mini-batches $\mathcal{B} \subset \mathcal{S}_e$ **do**
        Update θ by teacher forcing on $\mathcal{B}$
    end for
end for
return p_θ

Algorithm 8 Phase 2: ReST-Style Refinement with GMM Filtering

Require: Pretrained generator p_θ, encoder ϕ, training graphs $\mathcal{D}$, iterations K, candidates M, GMM components C, training quantile q
Ensure: Refined generator p_θ

Fit GMM with C components on $\{\phi(G) : G \in \mathcal{D}\}$
$\tau \leftarrow q$-quantile of $\{\log p_{\mathrm{GMM}}(\phi(G)) : G \in \mathcal{D}\}$
for $k = 1, \ldots, K$ **do**
    Sample M bit sequences from p_θ and decode well-formed ones into graphs $\{\widehat{G}_j\}$
    Remove duplicates: keep one $\widehat{G}_j$ per distinct $\phi(\hat{G}_j)$
    $\mathcal{C}_k \leftarrow \emptyset$
    for all $\widehat{G}_j$ **do**
        $\ell_j \leftarrow \log p_{\mathrm{GMM}}(\phi(\widehat{G}_j))$
        if $\ell_j \geq \tau$ **then**
            $\mathcal{C}_k \leftarrow \mathcal{C}_k \cup \{\widehat{G}_j\}$
        end if
    end for
    $\mathcal{C}_k \leftarrow$ first N graphs of $\mathcal{C}_k$ ▷ $N = 10{,}000$
    $\mathcal{S}_k \leftarrow \{\textsc{BitSerialize}(\widehat{G}, \textsc{StructureGuidedBFS}(\widehat{G})) : \widehat{G} \in \mathcal{C}_k\}$
    for all mini-batches $\mathcal{B} \subset \mathcal{S}_k$ **do**
        Continue teacher-forced training of p_θ on $\mathcal{B}$
    end for
end for
return p_θ

F Additional Ablations

We ablate the two training phases and the node ordering. Curves are shown for Transition1x, the structurally hardest of our molecular benchmarks (Figures 3 and 2); we note where QM9 behaves differently. Throughout, Novelty is computed over valid, unique samples and must be read jointly with Validity.

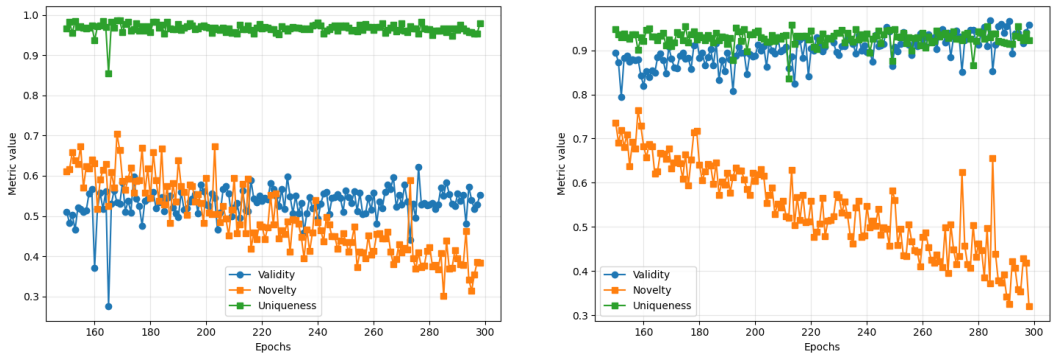
F.1 Phase 1 and the Role of ReST

The training pipeline is built around an exploration–exploitation decomposition: Phase 1 broadens the generator's support through perturbed graphs, and Phase 2 filters that support back toward plausible regions. We remove one ingredient at a time.

Perturbations without ReST. Training on original and perturbed graphs without Phase 2 isolates the exploratory component. The broadened support is never filtered, so the generator explores without specializing: Validity stays around 50–60% throughout training, and Novelty also declines (Figure 3a). On QM9, Novelty remains high, but Validity falls to a similar 50–60% range. Perturbation-based pretraining alone therefore does not yield controlled novelty; Phase 2 is needed to turn exploration into valid samples.

Neither perturbations nor ReST. Standard training on the original graphs alone keeps Validity high, but Novelty falls steadily from roughly 70% to about 30% over training (Figure 3b): without an exploratory signal, the model converges toward the empirical training distribution. On QM9 the effect is milder, with Validity and Uniqueness near-perfect throughout.

Together, these ablations support the complementary roles of the two phases: perturbations supply exploration, refinement supplies selection, and neither alone produces novel samples that remain valid. The contrast is sharper on Transition1x than on QM9, suggesting that the two-phase scheme matters most on structurally harder datasets.



(a) Phase 1 with perturbed graphs, without ReST.

(b) Phase 1 without perturbed graphs, without ReST.

Figure 3: **Ablation of the two-phase training scheme on Transition1x.** (a) With perturbed graphs but without ReST, the model explores more broadly but fails to maintain high Validity. (b) Without perturbed graphs and without ReST, the model remains conservative and Novelty declines steadily.

F.2 Node Ordering

We compare the structure-guided ordering against (i) a fully random node ordering and (ii) plain BFS without structural ranking. In all variants the final edge list is sorted lexicographically after relabeling, so only the node ordering changes.

Fully random ordering. Assigning node indices by a random permutation removes both the structural ranking and the BFS constraint. Validity collapses to roughly 10–15% (Figure 2a), and the same collapse occurs on QM9. Novelty and Uniqueness remain high, but because Novelty is computed only over valid, unique samples, this reflects the small fraction of samples that survive the validity check rather than successful generation. Without a meaningful ordering, the bit-level sequence is too irregular for the decoder to model.

Plain BFS. Retaining BFS but enqueuing neighbors in arbitrary identifier order recovers much of the lost validity, confirming that traversal locality is itself a strong inductive bias. On Transition1x, however, Validity remains lower and less stable than under structure-guided ordering, fluctuating between roughly 70% and 90% with transient drops (Figure 2b). On QM9, plain BFS performs comparably to structure-guided ordering, so the benefit of structural ranking emerges on structurally harder datasets. Structural ranking thus strengthens BFS rather than replacing it, by removing the arbitrary enqueue choices that plain BFS leaves to node identifiers.

F.3 Removing Perturbed Graphs Before Refinement

Finally, we pretrain on the original graphs only and then apply the same Phase 2 refinement, testing whether refinement alone can produce novelty. During Phase 1, Validity remains high while Novelty declines, as in Figure 3b. Phase 2 then further improves plausibility-related metrics but drives Novelty lower still, with only occasional spikes. Refinement can only select among what the generator already produces; without the support created by perturbed pretraining, it specializes the model at the expense of novelty.