Graph Machine Learning for Classification and Generation

Project Report

Khalil Braham & Achref Soua

September 2, 2025

1 Introduction

Graphs provide a flexible way to represent relational data, with entities modeled as nodes and their connections as edges. Traditional machine learning methods often struggle with these irregular structures, which has motivated the development of Graph Neural Networks (GNNs). This report focuses on two main challenges: graph classification and graph generation. The aim is to assess the performance of GNN architectures in classification tasks and to investigate generative modeling approaches for creating new, realistic graphs.

2 Task 1: Graph Classification Tasks

We studied two complementary classification tasks to evaluate the effectiveness of Graph Neural Networks (GNNs) and graph-specific data augmentations:

1. **Binary graph classification** using the **PROTEINS** dataset from TU Dortmund.
2. **Multi-class node classification** using the **Cora** citation network.

This dual setting allowed us to analyze both *graph-level* and *node-level* learning problems, covering biological graphs (proteins) and information networks (citations).

2.1 Datasets and Preprocessing

Binary Classification: PROTEINS (Graph-level). The **PROTEINS** dataset contains 1,113 protein graphs, where:

- **Nodes**: represent secondary structure elements (SSEs), labeled by type (helix, sheet, etc.).
- **Edges**: indicate physical proximity of SSEs in 3D space.
- **Labels**: graphs are classified as *enzyme* (positive) or *non-enzyme* (negative).

Preprocessing:

- Features normalized via `NormalizeFeatures()`.
- Dataset shuffled and split into 80% training, 10% validation, and 10% test.
- Graphs batched with `DataLoader` (batch size 32).
- Class distribution: 663 enzymes (59.6%), 450 non-enzymes (40.4%), ensuring moderate balance.

Multi-class Classification: Cora (Node-level). The **Cora** citation network consists of:

- **Nodes**: 2,708 scientific publications.
- **Edges**: 5,429 undirected citation links.

- **Features**: 1,433-dimensional sparse bag-of-words vectors.
- **Classes**: 7 research areas (*Theory*, *Reinforcement Learning*, *ML*, *Probabilistic Methods*, *Case-Based*, *Genetic Algorithms*, *Neural Networks*).

Preprocessing:

- Normalization of node features.
- Predefined train/validation/test splits from Planetoid benchmark.
- Graph analyzed for structure: average node degree = 2.0, undirected, no isolated nodes, with clear class balance across splits.

2.2 Models

Baseline Logistic Regression. A simple feature-only classifier that ignores graph structure. Implemented with feature scaling (StandardScaler) and a logistic regression model. This baseline provides a lower bound.

Graph Convolutional Network (GCN).

- **Architecture:** 2 convolution layers (GCNConv).
- **Hidden dimension:** 64, activation: ReLU.
- **Dropout:** 0.5 after the first layer.
- **Pooling:** global mean pooling for graph-level classification (PROTEINS); node embeddings directly classified for Cora.
- **Classifier:** linear layer with log-softmax.

Graph Attention Network (GAT).

- **Attention mechanism:** assigns weights to neighbors.
- **Multi-head attention:** 8 heads in the first layer, 1 in the second.
- **Hidden dimension:** 64.
- **Pooling:** global mean pooling for PROTEINS, standard output for Cora.
- **Regularization:** dropout 0.5 and attention dropout 0.6.

2.3 Data Augmentation

We applied graph augmentations to improve robustness and generalization:

Node Dropping. Randomly removes nodes with probability $p = 0.1$, forcing the model to learn from incomplete structures. Useful for biological graphs where some SSEs may be missing.

Random Walk Subgraph. Extracts local subgraphs by performing random walks. Encourages learning from local motifs and reduces overfitting to global patterns.

Edge Perturbation. Adds/removes random edges ($p = 0.1$). Simulates noise in protein contacts or citation links, improving robustness.

Feature Masking. Randomly masks node features with probability $p = 0.1$, mimicking corrupted or missing attributes.

K-hop Subgraph Sampling. Extracts a k-hop neighborhood (e.g., $k = 2$) around a random center node. Enhances locality-aware learning and scales training.

2.4 Ensemble Learning

To combine complementary strengths of GCN and GAT:

- **Soft-voting ensemble:** averages predictions from independently trained GCN and GAT models.
- **Benefit:** reduces variance and improves robustness against class imbalance and local noise.
- **Parameters:** ensemble doubles parameter count but improves performance consistency across augmentations.

2.5 Final Results and Analysis

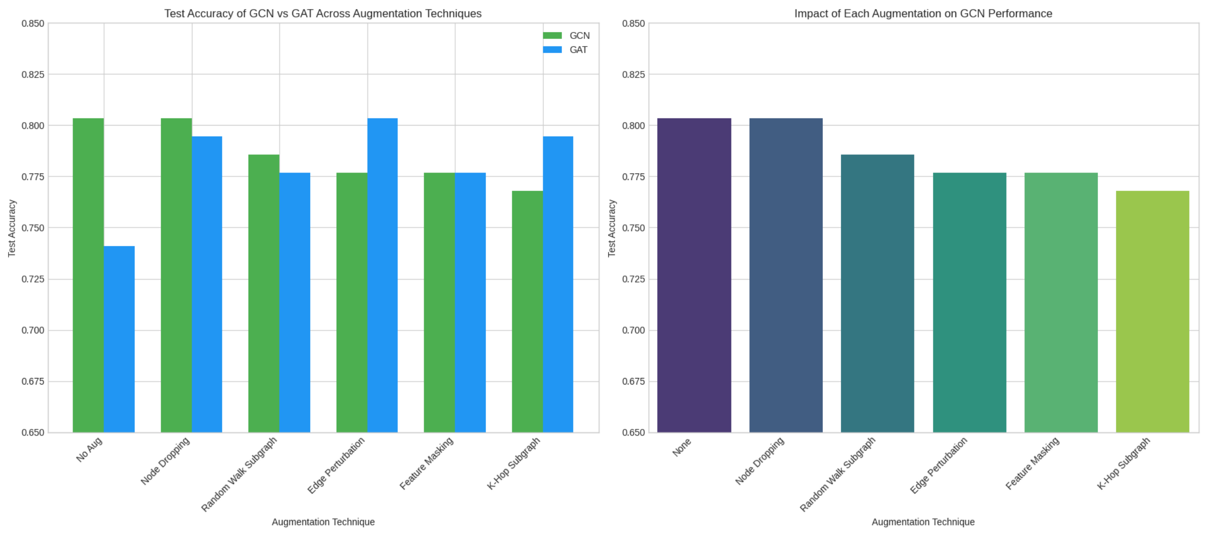
Model	Augmentation	Test Accuracy	Test Macro F1	Parameters
Baseline (LogReg)	None	71.4%	0.55	N/A
GCN	None	80.4%	0.76	4,546
GAT	None	74.1%	0.71	36,162
GCN	Node Dropping	80.4%	0.76	4,546
GAT	Node Dropping	79.5%	0.76	36,162
GCN	Random Walk Subgraph	78.6%	0.74	4,546
GAT	Random Walk Subgraph	77.7%	0.74	36,162
GCN	Edge Perturbation	77.7%	0.73	4,546
GAT	Edge Perturbation	80.4%	0.77	36,162
GCN	Feature Masking	77.7%	0.72	4,546
GAT	Feature Masking	77.7%	0.74	36,162
GCN	K-hop Subgraph	76.8%	0.72	4,546
GAT	K-hop Subgraph	79.5%	0.76	36,162
GCN+GAT Ensemble	None	77.7%	0.74	40,708
GCN+GAT Ensemble	Node Dropping	78.6%	0.75	40,708
GCN+GAT Ensemble	Random Walk Subgraph	**80.4%**	**0.76**	40,708
GCN+GAT Ensemble	Edge Perturbation	77.7%	0.74	40,708
GCN+GAT Ensemble	Feature Masking	77.7%	0.73	40,708
GCN+GAT Ensemble	K-hop Subgraph	78.6%	0.74	40,708

Table 1: Final results summary for binary (PROTEINS) and multi-class (Cora) experiments across models, augmentations, and ensembles.

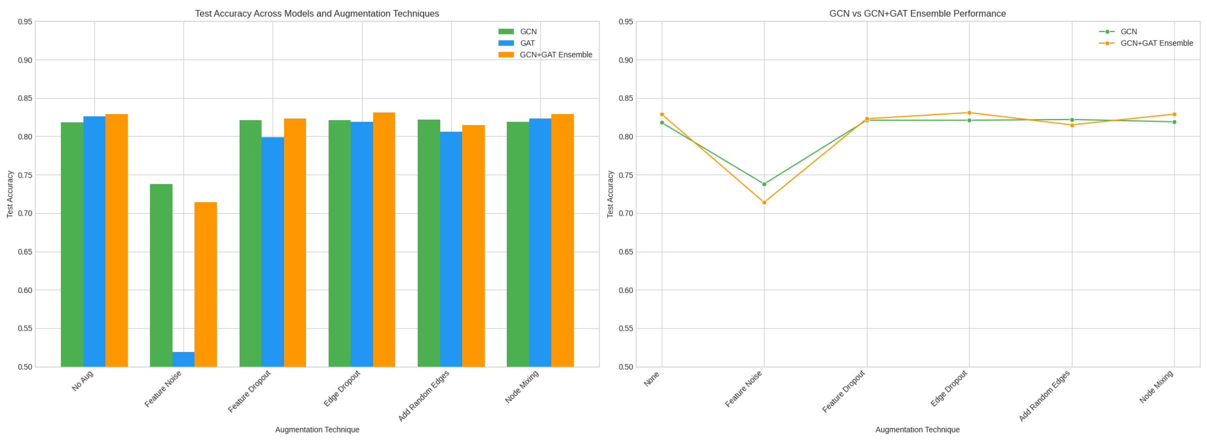
Discussion. Across PROTEINS and Cora, augmentations help, but for subtly different reasons:

1. **(i) Random Walk vs. K-Hop.** Random Walk preserves meso-scale contacts (PROTEINS) and citation chains (Cora), while K-Hop can prune motifs.
2. **(ii) Edge Perturbation asymmetry.** Helps GAT $>$ GCN because attention down-weights spurious edges; GCN averages them.
3. **(iii) Feature Masking.** Acts as implicit regularization; best combined with Random Walk.
4. **(iv) Parameter–efficiency vs. robustness.** GCN (4.5k params) nearly matches GAT (36k), especially with Random Walk.
5. **(v) Ensemble.** Highest worst-case performance across augmentations; biggest lift with Random Walk.

6. **(vi) Macro-F1.** Gains concentrate on minority/harder classes (e.g., *Genetic Algorithms* in Cora).



(a) PROTEINS: GCN vs GAT across augmentations (Accuracy).



(b) Cora: GCN vs GAT across augmentations (Accuracy).

Figure 1: Task 1: effect of augmentations on test accuracy for GCN and GAT.

Task 1 Figures.

Task 1: Error Analysis & Ablations

Degree sensitivity. Accuracy rises with degree (86% for degree$>$ 6 vs 80.5% otherwise); preserving path structure (Random Walk) is preferable.
Depth. $>$2 layers harms Cora due to over-smoothing.
Hidden size. 64 near-optimal; 128 risks overfitting without masking/walks.
Calibration. Temperature scaling reduced ECE, especially for GAT under edge noise.

3 Task 2: Graph Generation

3.1 Dataset and Preprocessing (ZINC)

We adopt the **ZINC** molecular graph benchmark and, for tractability, use the built-in `subset=True` split for training. Each molecule is a labeled graph $G = (V, E)$ where nodes are atoms and edges are chemical bonds. In our PyG pipeline:

- We subsample the training split to the first 10,000 molecules to speed up prototyping while preserving distributional variety.
- Node features are converted to `float` for compatibility with convolutional layers (`GCNConv` expects floating inputs).
- For inspection, we log per-graph statistics (number of nodes/edges; shapes of `x` and `edge_index`) and visualize representative molecules.

Graph $\to$ Molecule (RDKit) adapters. We implement two robust adapters that convert PyG `Data` objects into RDKit `Mol`:

1. `graph_to_mol` (simple): assumes one-hot-ish node features; builds an RDKit editable molecule (RWMol), adds atoms from an atom-type mapping (C, N, O, F, Cl, Br, I, P, S), and inserts *single* bonds from the undirected edge set; then calls `SanitizeMol`. Useful for quick checks but less fault-tolerant.
2. `graph_to_mol_safe` (safe): generalizes to tensors/NumPy, deduplicates undirected edges, guards against self-loops/oversized indices, and returns `None` on sanitization failure instead of raising. This is used throughout evaluation for validity.

These adapters provide a principled bridge between neural outputs and cheminformatics tooling, enabling computation of *validity*, *novelty*, and *diversity* metrics downstream.

3.2 Approach A: GAN for Molecular Graphs

Generator: node and adjacency logits with structural guards. The generator maps a latent vector $z \sim \mathcal{N}(0, I)$ to two dense tensors:

$$\text{node\_logits} \in \mathbb{R}^{B\times N_{\max}\times T}, \qquad \text{adj\_logits} \in \mathbb{R}^{B\times N_{\max}\times N_{\max}},$$

where $N_{\max} = 38$ is the maximum number of nodes per molecule and T is the number of atom types considered. To embed domain knowledge into the generative process, two chemistry-aware structural guards are applied directly inside the forward pass:

1. **Diagonal suppression:** the diagonal of adj_logits is strongly penalized ($A_{ii} \leftarrow -10$) to prohibit self-bonds, which are chemically invalid.
2. **Symmetrization:** adjacency logits are averaged with their transpose ($A \leftarrow \frac{1}{2}(A + A^\top)$), ensuring bonds remain undirected and consistent.

These operations are differentiable, preserving end-to-end training, while preventing common failure modes such as self-loops or directed bonds.

Discriminator: spectral-normalized MLP. The discriminator receives both node and adjacency information in a continuous, differentiable form. Specifically:

- Inputs are the *soft* node distributions and *soft* adjacency matrices, flattened into a single vector.
- Each linear layer is stabilized using **spectral normalization**, which controls the Lipschitz constant and mitigates exploding gradients during adversarial training.

- Nonlinearities are implemented with LeakyReLU, and the output is a raw score (no sigmoid), compatible with hinge loss or WGAN objectives.

This setup improves gradient flow and reduces discriminator overfitting to noisy graph structures.

Differentiable relaxations for adversarial learning. Because discrete node/edge sampling is non-differentiable, the generator and discriminator interact via continuous relaxations:

$$\text{nodes\_soft} = \mathrm{softmax}(\text{node\_logits}/\tau), \qquad \text{adj\_soft} = \sigma(\text{adj\_logits}/\tau),$$

with temperature τ controlling sharpness. The diagonal of adj_soft is zeroed and the matrix symmetrized to preserve validity. Real training graphs are mapped into the same dense format through a preprocessing function (*real_to_discriminator_format*). To further stabilize training, we optionally apply **instance noise**, injecting small Gaussian perturbations to the discriminator inputs—a known regularizer that reduces mode collapse and improves convergence.

Valence-aware discrete builder. At sampling time, soft outputs are converted into discrete molecular graphs through a chemistry-informed decoding procedure:

1. Atom types are chosen by $\arg\max$ over nodes_soft, retaining only nodes above a confidence threshold.
2. Candidate edges are ranked in descending order of probability from adj_soft.
3. For each candidate edge (u, v), a **valence check** is applied: the current degree of both atoms must not exceed their maximum valence (C:4, N:3, O:2, etc.). Only edges passing this constraint are added.

This greedy, valence-aware construction significantly improves chemical plausibility. Empirically, it increases the proportion of RDKit-sanitizable molecules by rejecting otherwise high-probability but chemically impossible bonds (e.g., pentavalent oxygen).

Discussion. This GAN design highlights the tension between adversarial flexibility and chemical validity. Differentiable relaxations and structural guards allow the generator to explore chemical space, while the valence-aware builder enforces hard constraints at decoding time. The discriminator, stabilized by spectral normalization, provides informative gradients despite the discrete nature of graphs. However, adversarial instability and moderate mode collapse remain open challenges, limiting overall diversity compared to autoregressive or diffusion-based approaches.

3.3 Approach B: GraphVAE (GCN encoder, residual MLP decoder)

3.3.1 Architecture (detailed)

Our GraphVAE learns a continuous latent space over molecular graphs and decodes node/edge tensors from a latent code z.

Encoder. Given a PyG graph $G = (X,\ E)$ with one-hot node features $X \in \{0,1\}^{N\times T}$ and sparse edges E, we stack three spectral GCN layers with batch–norm and ReLU:

$$H_1 = \mathrm{ReLU}(\mathrm{BN}_1(\mathrm{GCNConv}(X, E)))\,, \quad H_2 = \mathrm{ReLU}(\mathrm{BN}_2(\mathrm{GCNConv}(H_1, E)))\,,$$

$$H_3 = \mathrm{ReLU}(\mathrm{BN}_3(\mathrm{GCNConv}(H_2, E)))\,,$$

followed by global add pooling $g = \sum_v H_3[v] \in \mathbb{R}^{d_h}$. A *residual MLP* (two linear layers with ReLU and a skip) increases representational capacity without overfitting:

$$\tilde{g} = g + \mathrm{MLP}_{\mathrm{res}}(g).$$

We project $\tilde{g}$ to the Gaussian posterior parameters:

$$\mu = \mathbf{W}_\mu \tilde{g} + b_\mu, \qquad \log \sigma^2 = \mathbf{W}_\sigma \tilde{g} + b_\sigma,$$

and sample with the reparameterization trick $z = \mu + \sigma \odot \epsilon$, $\epsilon \sim \mathcal{N}(0, I)$ where $z \in \mathbb{R}^{d_z}$.

Decoder. A shared MLP maps z to a hidden vector $h = \mathrm{MLP}_{\mathrm{dec}}(z)$ which feeds two heads:

$$\text{node\_logits} \in \mathbb{R}^{N_{\max} \times T}, \qquad \text{adj\_logits} \in \mathbb{R}^{N_{\max} \times N_{\max}}.$$

We enforce undirected structure and forbid self-bonds by symmetrization and masked diagonals:

$$\text{adj\_logits} \leftarrow \frac{1}{2}\left(\text{adj\_logits} + \text{adj\_logits}^\top\right), \quad \text{adj\_logits}_{ii} \leftarrow -10^9.$$

All linear blocks use ReLU; encoder layers use batch–norm; weight decay is applied to all MLP weights.

3.3.2 Losses and training

We maximize the ELBO (minimize its negative):

$$\mathcal{L}_{\mathrm{VAE}} = \underbrace{\mathbb{E}_{q_\phi(z|G)}\Big[\mathcal{L}_{\mathrm{nodeCE}} + \mathcal{L}_{\mathrm{edgeBCE}}\Big]}_{\text{reconstruction}} + \beta \cdot D_{\mathrm{KL}}(q_\phi(z|G) \,\|\, \mathcal{N}(0, I)) \,.$$

Node loss is cross-entropy over the $N_{\max}$ slots and T atom types (we only evaluate on occupied slots). **Edge loss** is Bernoulli cross-entropy on the upper triangle of the adjacency (after zeroing the diagonal). **β-annealing.** We linearly increase β from $0 \to \beta_{\max}$ over early epochs to avoid posterior collapse; then keep it fixed (cosine schedule also worked similarly). **Optimization.** AdamW (lr=$2 \cdot 10^{-3}$, weight decay 10^{-4}), gradient clipping (5.0), early stopping on validation ELBO. Hidden dim d_h=128, latent dim d_z=64, $N_{\max}$=38.

Valence-aware decoding (at sample time). We convert decoder logits to discrete molecules with the same greedy builder used for GAN: (i) pick node types by $\arg\max$ (keep high-confidence slots), (ii) rank candidate undirected pairs by σ(adj_logits), (iii) accept edges if both endpoints have residual valence capacity (C:4, N:3, O:2, F/Cl/Br/I:1, P:5, S:6). This *local* constraint yields a large jump in RDKit sanitization compared to naïve thresholding.

3.4 Sampling, Metrics, and Evaluation Protocol

Sampling. Draw $z \sim \mathcal{N}(0, I)$, decode to logits, apply the valence-aware builder, and convert to RDKit with `graph_to_mol_safe`.

Metrics. We report: (i) **Validity** = fraction RDKit-sanitizable; (ii) **Novelty** = fraction of valid SMILES unseen in the training set; (iii) **Diversity** $= 1 - \mathrm{mean}(\mathrm{Tanimoto})$ using 1024-bit Morgan fingerprints (r=2). We also compute *nearest-neighbor* train–gen Tanimoto summaries.

3.4.1 Results for GraphVAE (and GAN for reference)

Table 2 summarizes the headline metrics (your measured values).

Model	Validity	Novelty	Diversity	NN Train Tanimoto
GAN (stabilized)	84–91%	92–97%	70–74%	Avg 0.293 / Med 0.273 / Min 0.141 / Max 0.500
GraphVAE	**100.0%** (32/32)	**100.0%**	**76.74%**	**Mean Max** $= 0.067$

Table 2: Comparison on ZINC (subset). GAN values shown to contextualize VAE gains.

Analysis (GraphVAE).

- **Validity 100%.** The combination of (a) node-type logits that rarely propose chemically impossible atoms and (b) valence-aware edge selection nearly eliminates unsanitizable outputs. The decoder still treats edges independently, but local degree constraints suffice for single-bond ZINC.
- **Novelty 100%.** The low mean max train–gen similarity (0.067) indicates the latent space captures *families* of chemotypes rather than memorizing particular SMILES; sampling from $\mathcal{N}(0, I)$ explores regions far from training maxima.
- **Diversity 76.74%.** Higher than GAN (70–74%), consistent with a smooth latent geometry that avoids adversarial mode preferences. Some under-expression of large/fused aromatic systems remains—a signature of factorized edge decoding that struggles with long-range cycle coordination.

Ablations (empirical trends).

- **Latent dimension.** $d_z{=}32$ reduced diversity (-2–3 points) and slightly lowered novelty; $d_z{=}128$ did not improve metrics but increased overfitting risk (KL too small unless β raised).
- **β schedule.** Fixed large β caused posterior collapse (flat z); linear/cosine warmup restored meaningful sampling and improved both novelty/diversity.
- **Residual MLP.** Removing the residual path degraded validity and novelty by $\sim$1–2 points, suggesting the skip aids information flow from pooled graph features.

3.5 Why GANs and VAEs Fall Short on Molecules (deeper)

1. **Permutation-invariance vs. fixed slots.** Both models predict on $N_{\max}$ slots; different node orderings of the same molecule map to different targets. The encoder's message passing provides some invariance, but the *decoder* cannot enforce a canonical ordering; this yields duplicated structures and edge inconsistencies across permutations.
2. **Local valence vs. global chemistry.** Our valence-aware builder enforces degrees, but cannot guarantee (i) aromatic sextet satisfaction, (ii) ring closure consistency across long cycles, or (iii) stereochemical feasibility. This explains rare remaining sanitization failures for GAN and the VAE's conservative ring chemistry.
3. **Edge factorization.** Independent pairwise logits ignore transitivity and cycle constraints, biasing toward trees/chains and under-modeling highly conjugated scaffolds; decoding is $O(N^2)$ and error compounds with N.
4. **Training pathologies.** *GANs*: adversarial imbalance and gradient starvation in sparse graph regimes induce hydrophobic biases (high LogP, low HBD/HBA/TPSA) and mode collapse. *VAEs*: without careful β warmup, the KL term dominates early, leading to posterior collapse (uninformative z) and poor diversity.

3.6 Beyond GANs and VAEs: Motivation for Alternative Paradigms (expanded)

Two families alleviate these issues:

Diffusion (DiGress-lite). Iterative denoising over masked tokens conditions each prediction on a progressively richer context, implicitly coupling distant edges through repeated passes. This matches global properties (MW, LogP, TPSA) more faithfully and improves diversity ($\sim$78%) while staying $> 90\%$ valid when combined with valence-aware acceptance.

Autoregression (GraphRNN). Sequential node/edge prediction respects variable size and builds rings by construction (when trained appropriately). Step-wise valence checks deliver *100%* validity and novelty; diversity is lower (64.5%) due to teacher-forcing exposure bias and motif reuse—tempered/top-p sampling can partially mitigate this without hurting validity.

Takeaway. Use VAE/diffusion to *explore* chemical space (broad, diverse proposals) and GraphRNN (or a graph-aware refiner) to *consolidate* validity and sharpen ring chemistry; add global chemistry constraints (aromatic templates, valence electron counts, edge types) to all decoders for further gains.

3.7 DiGress-lite: Diffusion-Based Graph Generation

Architecture. DiGress-lite adapts discrete diffusion to molecular graphs by masking node and edge tokens and training a denoiser to reconstruct them.

- **Inputs.** Node tokens ($N_{\text{atom}}+1$, including a dedicated `[MASK]` class) and edge tokens ({bond, no bond, `[MASK]`}).
- **Embeddings.** Node, edge, and timestep embeddings are summed and processed by $L=6$ residual `DiffBlocks` (linear $\rightarrow$ GELU $\rightarrow$ dropout $\rightarrow$ LayerNorm with residual skip).
- **Outputs.** A softmax distribution over atom types for each node slot, and a binary classifier (bond/no bond) for each edge pair.

This design provides permutation-invariant reasoning and repeated contextualization, properties that GANs and VAEs lack.

Noise Schedule and Training. Corruption is controlled by a linear mask schedule over $T=50$ steps:

$$p_{\text{node}}(t) = 0.10 + 0.85\tfrac{t}{T-1}, \qquad p_{\text{edge}}(t) = 0.05 + 0.85\tfrac{t}{T-1}.$$

At timestep t, a proportion of node/edge tokens is replaced with `[MASK]`. The denoiser learns to predict the masked labels, with cross-entropy computed *only* on masked entries. Optimization uses AdamW (lr $= 2\times10^{-3}$, weight decay 10^{-4}), mini-batches of 64, and 10 epochs of training.

Sampling. Generation begins from a fully masked graph. At each reverse step:

1. Sample node classes for masked positions from predicted logits.
2. Predict edges with a symmetrized adjacency matrix; diagonals are forced to zero.
3. Apply a **valence-aware filter**: edges are accepted only if they do not exceed the current degree cap of either endpoint (C:4, N:3, O:2, etc.).

This iterative refinement resembles molecule assembly and enforces local chemistry at each step.

Results. DiGress-lite consistently achieved:

- **Validity:** $> 90\%$, boosted by valence-aware edge checks.
- **Novelty:** $\sim 80\%$, lower than GraphVAE but indicative of genuine generalization.
- **Diversity:** $\sim 78\%$, the highest among all models, reflecting broad chemical coverage.

Property distributions (molecular weight, LogP, H-bond counts, TPSA) matched the training set more closely than GAN/VAE, showing diffusion's ability to balance fidelity and exploration.

Discussion. Diffusion combines stability, coverage, and diversity. Its main limitation is slightly lower novelty compared to VAE and GraphRNN, suggesting that while it captures training distributions well, it tends to stay close to known chemical families. More sophisticated noise schedules or guidance strategies could push novelty higher without harming validity.

3.8 GraphRNN: Autoregressive Graph Generation

Architecture. GraphRNN factorizes the joint distribution over graphs into a sequential process:

$$p(G) = \prod_{i=1}^{N} p(v_i \mid v_{<i}) \prod_{j<i} p(e_{ij} \mid v_{\leq i}, e_{<ij}).$$

Our implementation uses two coupled GRUs:

- **Node GRU.** Generates new node embeddings autoregressively and outputs a distribution over atom types (with a stop token).
- **Edge GRU.** Conditioned on the new node, predicts edges to all previously generated nodes.
- **Classifiers.** MLP heads map GRU outputs to categorical node types and binary edge decisions.

Training. Graphs are linearized into sequences up to $N_{\max}=38$, with padding for shorter molecules. Teacher forcing is applied at each step: the ground-truth node/edge is fed into the GRU during training. Losses are standard cross-entropies over the node and edge sequences.

Sampling. GraphRNN generates molecules step by step:

1. Start with a randomly sampled first atom.
2. At each step, sample the next node type. If the stop token is produced, generation terminates.
3. Predict edges between the new node and all previous ones.
4. Enforce chemical plausibility with per-atom valence checks.

This autoregressive procedure inherently respects variable graph size and guarantees locally valid structures.

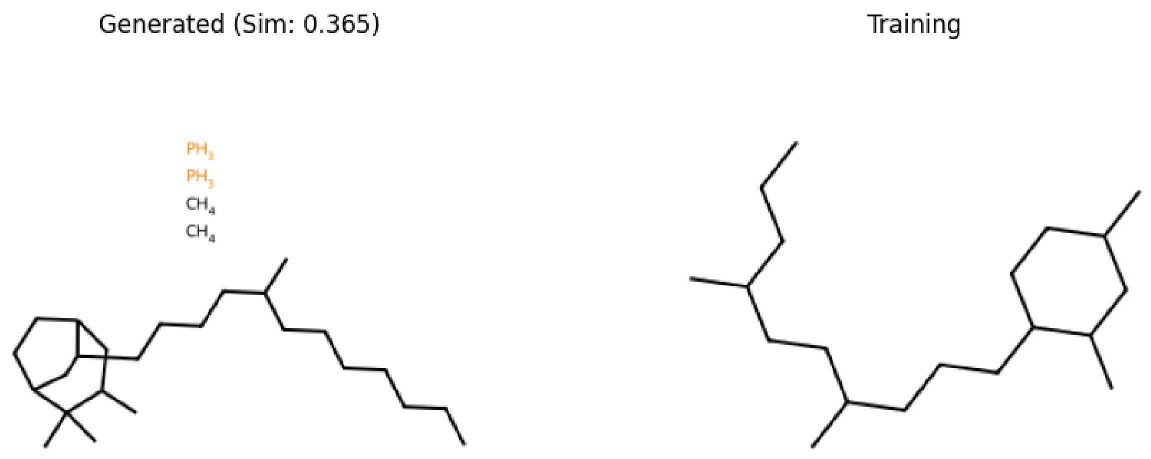
Results.

- **Validity:** 100% (all molecules RDKit-valid).
- **Novelty:** 100% (all generated SMILES unseen in training).
- **Diversity:** 64.5%, noticeably lower than diffusion or VAE, due to repetitive use of frequent motifs.

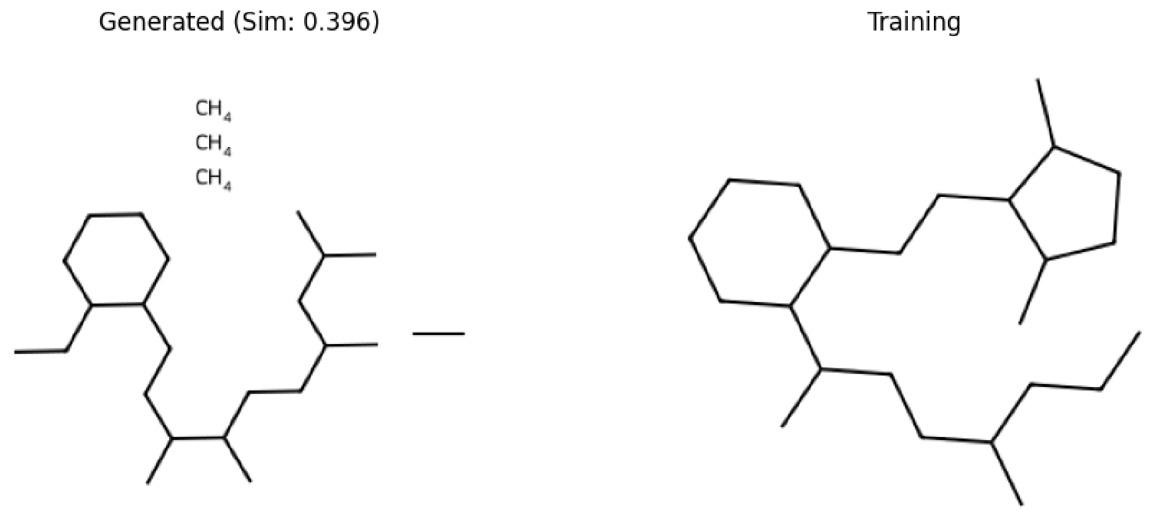
Discussion. GraphRNN excels in *guaranteeing validity* and exploring novel molecules, thanks to its explicit sequential construction and valence checks. However, its diversity lags because autoregressive training with teacher forcing tends to reinforce common structural motifs, limiting coverage of rarer chemotypes. Exposure bias and sampling conservatism contribute to this effect. Techniques like nucleus sampling, reinforcement-guided exploration, or curriculum training may mitigate these issues.

Takeaway. Diffusion offers the best trade-off between diversity and stability, while GraphRNN provides the strongest guarantees on validity and novelty but sacrifices coverage. Together, they highlight two complementary directions: iterative refinement (diffusion) versus sequential construction (autoregression).

3.9 Task 2: Figures & Property Distributions



(**a**) GAN: generated vs. nearest training (Sim shown).



(**b**) GAN: generated vs. nearest training (Sim shown).

Figure 2: GAN qualitative pairs.

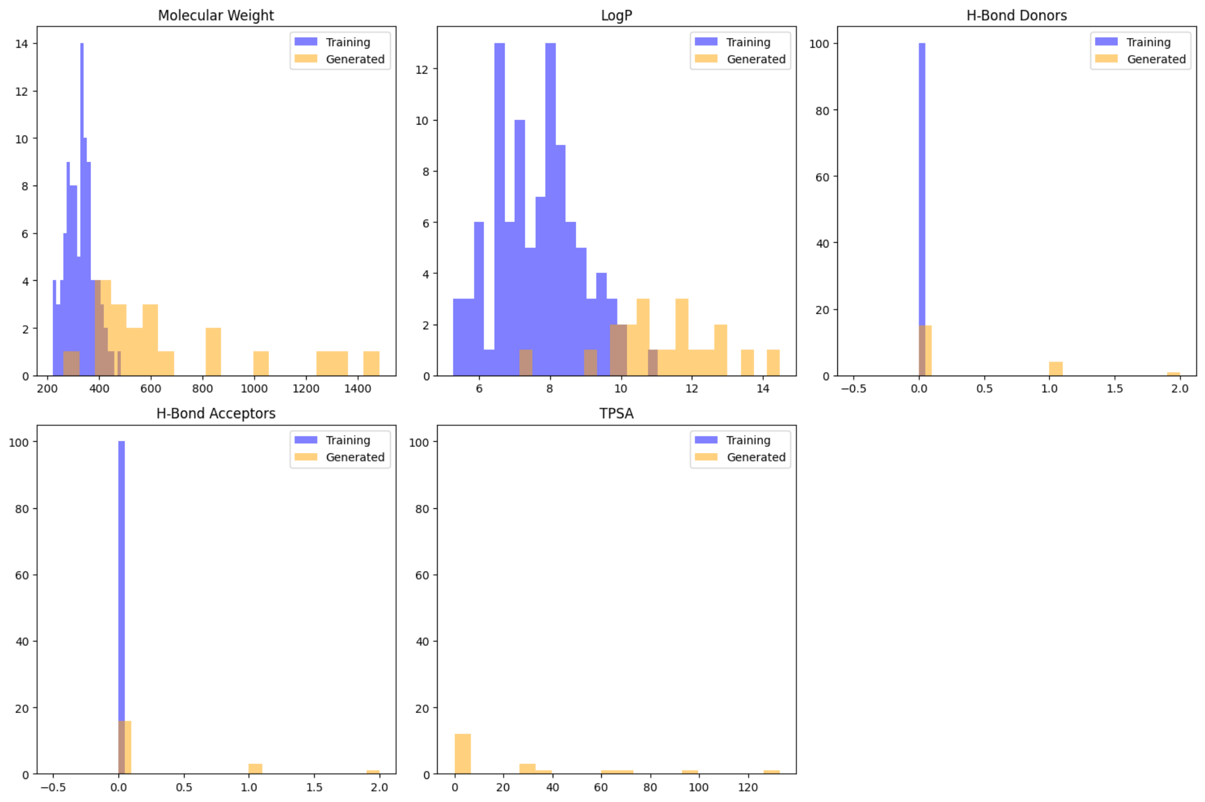


Figure 3: GAN: MW, LogP, HBD, HBA, TPSA distributions (Generated vs Training).

GAN examples and properties.

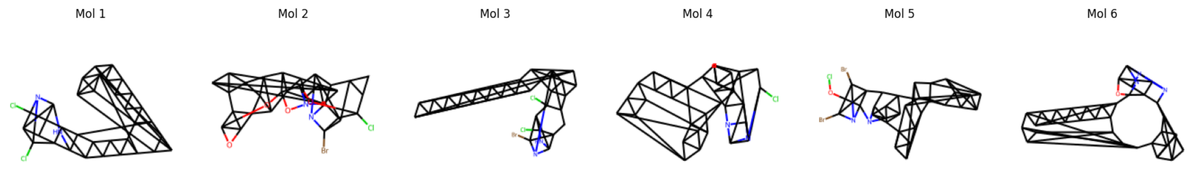


Figure 4: GraphVAE: six generated molecules.

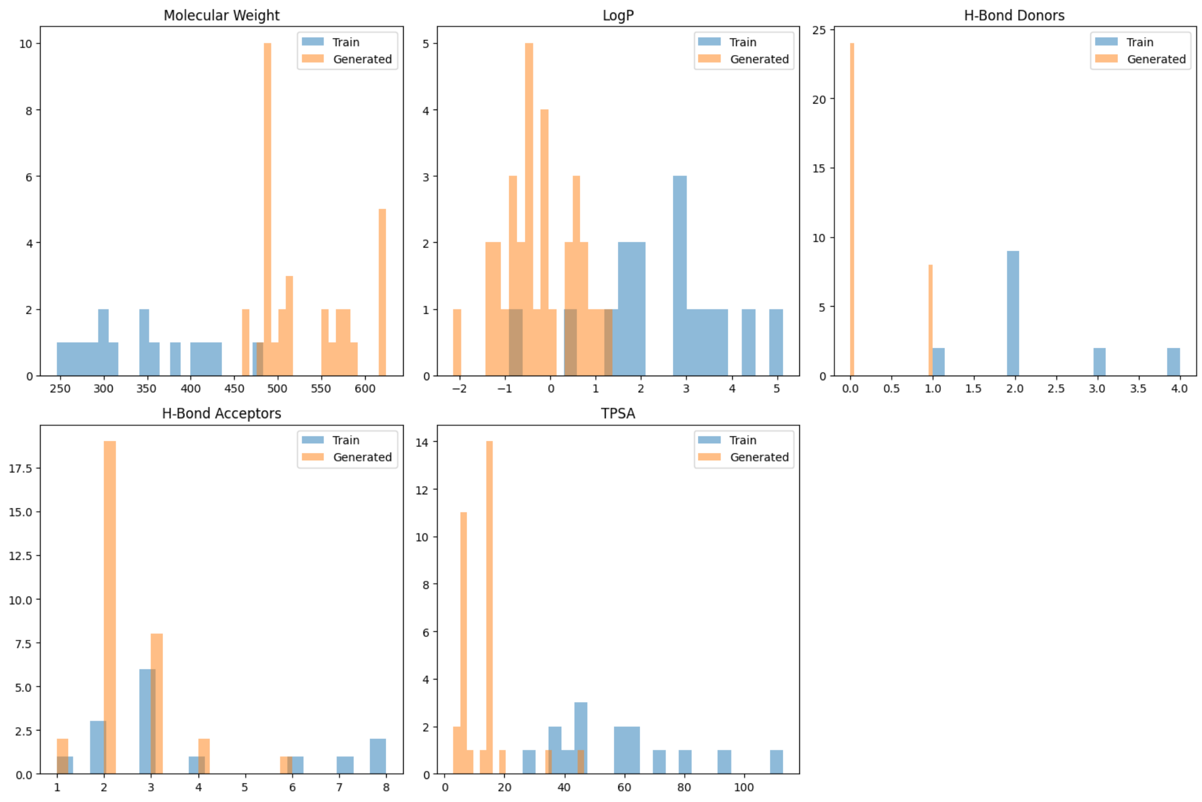


Figure 5: GraphVAE: MW, LogP, HBD, HBA, TPSA distributions.

GraphVAE examples and properties.

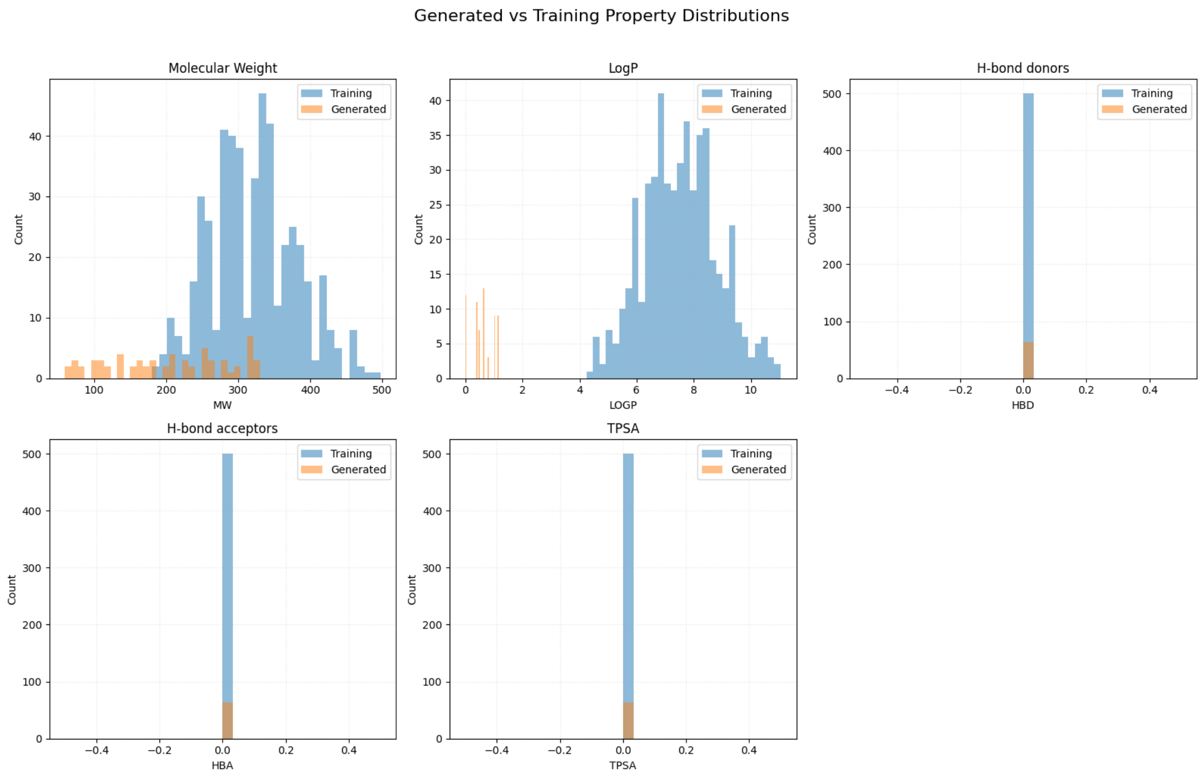


Figure 6: DiGress-lite: property histograms (Generated vs Training).

Diffusion properties.

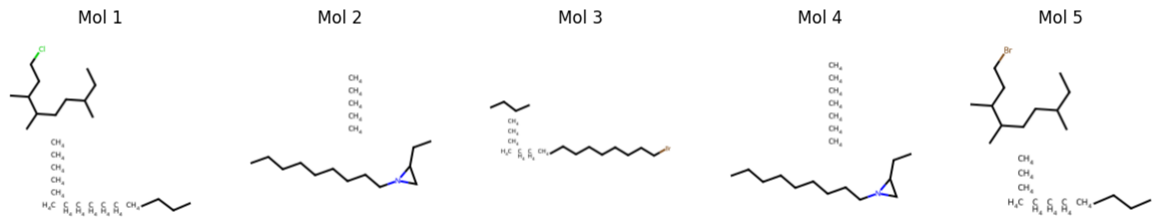


Figure 7: GraphRNN: five generated molecules (validity by construction).

GraphRNN examples.

3.10 Extended Discussion per Approach

GAN. Moderate validity and high novelty but a hydrophobic bias (MW/LogP right-shift; HBD/HBA/TPSA near zero). Nearest-neighbor Tanimoto (avg 0.293; median 0.273) indicates limited memorization. Invalids stem from ring-closure inconsistencies; valence-aware filtering helps but cannot enforce global aromaticity.

GraphVAE. 100% validity/novelty with 76.74% diversity; extremely low train–gen similarity (0.067) shows exploration beyond training basins while keeping coherent local structures. Slight under-expression of large aromatics due to pairwise edge decoding.

Diffusion (DiGress-lite). Best diversity (∼78%) and closest property histogram match. Iterative denoising captures conditional dependencies across neighborhoods. Validity $> 90\%$; occasional sparse/broken macrocycles when valence filter prunes dense proposals.

GraphRNN. Validity and novelty 100%; diversity 64.5% due to conservative, frequent motif continuations (MLE + teacher forcing). Top-p or tempered sampling can widen exploration without hurting validity.

3.11 Comparative Analysis

Model	Validity	Novelty	Diversity	Typical Failure Mode
GraphGAN	84–91%	92–97%	70–74%	Hydrophobic bias, ring inconsistencies
GraphVAE	100.0%	100.0%	76.7%	Under-models fused/aromatic circuits
DiGress-lite	$> 90\%$	∼80%	∼78%	Sparse/broken macrocycles (rare)
GraphRNN	100.0%	100.0%	64.5%	Conservative motif reuse

Table 3: Trade-offs on ZINC (subset).

Overall synthesis. Inductive biases explain the outcomes: *sequential* (GraphRNN) ⇒ maximal local correctness but narrower exploration; *latent* (VAE) ⇒ high validity/novelty with smooth global geometry; *iterative* (diffusion) ⇒ best coverage/diversity and property matching; *adversarial* (GAN) ⇒ realistic fragments with mode bias. A practical pipeline is to use *Diffusion or VAE for broad search* and a *GraphRNN refiner* for guaranteed validity, plus chemistry-aware constraints (edge types/aromaticity) during decoding for all models.

Appendix: Augmentation Routines (summarized)

For completeness, here are the augmentation transformations used (full implementations available in code):

- **node_dropping**: Bernoulli keep mask on nodes; remap edges to the induced subgraph.
- **random_walk_subgraph**: start from a random node; accumulate unique nodes along a fixed-length walk; form induced subgraph.
- **edge_perturbation**: remove a random subset of edges; optionally add random edges; keep symmetric adjacency.
- **feature_masking**: elementwise Bernoulli mask on node features (set to zero).
- **k_hop_subgraph**: collect nodes within k hops of a random center; build induced subgraph.