Constrained Goal-directed Planar Graph Generation with Grammar-based Reinforcement Learning

Nicolas Hochuli*
Engineering Design and Computing Laboratory
ETH Zürich

Lorenzo Miele*
Engineering Design and Computing Laboratory
ETH Zürich

Kristina Shea
Engineering Design and Computing Laboratory
ETH Zürich

Tino Stankovic
Engineering Design and Computing Laboratory
ETH Zürich

Abstract

Planar graphs are central to applications across science and engineering, yet existing generators provide limited support for goal-directed generation under hard structural and geometric feasibility constraints. We propose a dataset-free method for generating planar graph embeddings by combining parametric graph grammars with safe reinforcement learning to optimize generic task-specific objectives while satisfying constraints *during* construction. We formulate the generation process as a constrained Markov decision process, where the graph grammar defines the state and action spaces. We further introduce an action projection that maps sampled actions toward state-dependent safe sets, improving constraint satisfaction during training. In contrast to classical graph generators and deep generative models, which typically offer limited goal-directed control or rely on weak constraint satisfaction, our method constructs feasible planar graph embeddings directly during generation. We also introduce a benchmark suite for constrained and goal-directed planar graph generation, together with classical and deep generative baselines. Across all benchmark tasks, our method consistently outperforms baselines while satisfying the formulated constraints.

1 Introduction

Goal-directed planar graph generation is a central research topic in computer science, with applications spanning urban planning and road-network design [1, 2, 3], engineering design, including architecture, structural optimization, origami design [4, 5, 6, 7, 8] as well as computational mesh-generation [9]. In many of these applications, graphs are not merely abstract combinatorial representations of relational data, but require explicit *planar embeddings* whose structure must be optimized for task-specific objectives under hard structural and geometric constraints. Despite this need, methods for systematically generating goal-directed embedded planar graphs while enforcing hard constraints remain limited in performance or tailored to domain-specific settings [10].

Classical graph generators [11, 12, 13] and specialized planar graph generators [14, 15] provide tractable random graph models, including samplers for well-defined planar graph families. However, they are not generally designed for goal-directed multi-objective optimization nor do they construct

*Equal contribution. Correspondence to nhochuli@ethz.ch
Code is available at https://github.com/ETHZ-EDAC/constrained-graph-rl.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

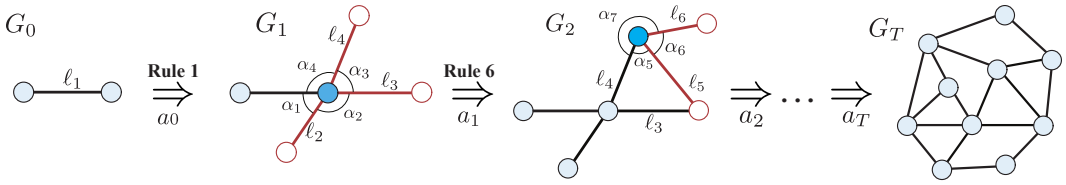


Figure 1: **Sequential planar graph generation.** Starting from an initial graph G_0, a policy samples rule-based actions $a_t \sim \pi(a_t \mid G_t)$ from a safe action space, enabling valid graph generation at each step. Actions modify graphs topologically and geometrically based on a predefined graph grammar.

valid planar graph embeddings under hard constraints. Deep generative models [10, 16, 17] can learn graph distributions and support property-conditioned generation [18], but they require training data which induces distributional biases and typically handle hard structural or geometric feasibility through soft conditioning, rejection sampling or post-processing. In contrast, autoregressive models, reinforcement learning, and hybrids of both have been used for sequential graph construction under hard validity constraints, especially in molecular design [19, 20, 21, 22, 23]. Yet many of these methods rely on discrete topological actions or domain-specific validity rules with relatively small graph instances, leaving generic and larger scale goal-directed generation of embedded planar graphs under hard constraints an open challenge.

Here, we propose a generic dataset-free method for goal-directed embedded planar graph generation across varying graph sizes. The method combines parametric graph grammars with safe reinforcement learning (Figure 1). We formulate the generation process as a Constrained Markov Decision Process (CMDP) whose state and action spaces are induced by the grammar. The grammar defines graph rewriting rules with continuous parameters, while an agent learns to apply these rules sequentially to optimize user-specified multi-objective rewards. Since graph size is controlled directly through sequential rule applications and termination criteria, our method can generate graphs with an exact prescribed number of nodes, a requirement that is difficult to enforce in many distributional graph generators. To improve feasibility during learning, we introduce a differentiable action projection that maps sampled actions toward state-dependent safe sets. The method constructs the planar embedding subject to hard constraints *during* generation, enabling controllable synthesis of planar graph embeddings for settings where multiple optimization objectives jointly matter. We additionally propose a new benchmark suite for constrained goal-directed embedded planar graph generation, that covers a wide variety of graph properties involving geometric, topological and spectral graph metrics.

2 Preliminaries

In this section we introduce parametric graph grammars, as well as their relation to Markov Decision Processes (MDP), to provide the basis for the definition of our method.

2.1 Graph Grammars

Graph grammars provide a rule-based formalism for the generation of structured graphs via sequential graph rewriting [24]. We adopt this formalism to model graph construction in which graph states, admissible rule application sites, and parametric rules are explicitly defined. In the following, we formalize the notion of graph states, introduce a parametric graph grammar, and define the graph transition operator that induces state-to-state evolution.

Graph states. At each discrete step $t \in \mathbb{N}$ in the rewriting process, we define a graph in its respective state as $G_t = (V_t, E_t^d, E_t^b, F_t)$, where V_t is the node set, E_t^d is a set of directed edges forming a directed acyclic graph (DAG), E_t^b is a set of undirected boundary edges encoding the planar embedding, $F_t \in \mathbb{R}^{|V_t| \times m}$ assigns m-dimensional feature vectors to nodes, including planar coordinates. All graphs considered in this work are finite, connected and planar. The primary reason for the DAG formulation is the efficient detection of the *generative frontier* of a graph state G_t which is defined as the set of leaf nodes $v_t^b \in V_t^b$, called *boundary nodes*, with

$$V_t^b = \{\, v \in V_t \mid \deg^-_{E_t^d}(v) \geq 1 \ \wedge \ \deg^+_{E_t^d}(v) = 0 \,\}, \tag{1}$$

The boundary nodes v_t^b are the only locations at which graph rewriting may be applied.

Parametric graph grammar. The graph generation process is directed by a parametric graph grammar $\mathcal{G} = (G_0, \mathcal{P}, \Theta)$, where G_0 is an initial graph state, $\mathcal{P}$ is a finite set of rewriting rules, and Θ assigns to each rule $p \in \mathcal{P}$ a continuous parameter space $\Theta(p)$ that spans the geometry of a graph in terms of edge lengths ℓ and sector angles α. Each rewriting rule p is a production $p : L \to R$ defining a subgraph L to be matched and replaced from a graph in state G_t using the subgraph in R. A *match* $g_t : L \to G_t$ is a graph morphism that identifies a node in the generative frontier such that $g_t(V_L) \subseteq V_t^b$. Since each left-hand side L contains exactly one boundary node, a match g_t uniquely determines the admissible rule application site as a boundary node $v_t^b := g_t(v)$, where $v \in V_L$. The set $\mathcal{P}$ comprises seven rewriting rules, that we detail in Appendix A.

Graph transition operator. At discrete step t, a transition $\mathcal{T}$ is determined by selecting (i) a rule $p_t \in \mathcal{P}$, (ii) an admissible match g_t, and (iii) a continuous parameter vector $\theta_t \in \Theta(p_t)$. These choices induce a direct graph transformation, denoted by $G_t \underset{\mathcal{T}}{\Rightarrow} G_{t+1}$. Equivalently, the successor state can be written as $G_{t+1} = \mathcal{T}(G_t; p_t, g_t, \theta_t)$, where $\mathcal{T}$ denotes the *graph transition operator* (see Figure 1). Operationally, $\mathcal{T}$ introduces new nodes, adds directed edges, updates boundary edges, and assigns planar coordinates to newly created nodes according to the selected rule and its associated parameters. The details regarding the graph transition operator are provided in Appendix A. The graph rewriting process terminates at a finite time T when a termination criterion is satisfied, such as reaching a prescribed number of nodes in G_T.

2.2 Constrained Markov Decision Processes

We briefly review CMDPs, which provide a principled framework for safe sequential decision-making under hard feasibility constraints [25]. A CMDP is defined as the tuple $(\mathcal{S}, \mathcal{A}, P, r, c, \gamma, \rho_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ the transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function, $c : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$ constraints, $\gamma \in [0, 1)$ the discount factor, and ρ_0 the initial-state distribution. A policy $\pi_\phi : \mathcal{S} \to P(\mathcal{A})$ is a mapping from states to a probability distribution over actions, where the objective of RL is to find a policy $\pi_\phi(a_t|s_t)$ as the probability of taking action $a_t \in \mathcal{A}$ given the state $s_t \in \mathcal{S}$ at discrete step $t \in \mathbb{N}$ that maximizes the expected discounted cumulative reward as

$$\max_{\phi} \ \mathbb{E}_{\tau \sim \pi_\phi}\Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\Big] \quad \text{s.t.} \quad c(s_{t+1}) = 0 \ \ \forall c \in \mathcal{C} \subseteq \{0, 1\}. \tag{2}$$

where $\tau = (s_0, a_0, s_1, a_1, ...)$ denotes a trajectory sampled from the policy π_ϕ and $\mathcal{C}$ defines the constraint set. The policy is approximated using neural networks with parametrization ϕ. In this work, constraints are enforced *instantaneously*: an action a_t is admissible at state s_t only if all $c = 0$ and is otherwise disallowed [26]. CMDPs of this form can be equivalently viewed as Markov decision processes with state-dependent safe action sets $\mathcal{A}_{\text{safe}}(s_t) \subseteq \mathcal{A}(s_t)$ that respect all constraints in $\mathcal{C}$ for every action taken, i.e.

$$\mathcal{A}_{\text{safe}}(s_t) = \big\{ a_t \in \mathcal{A}(s_t) \ \big| \ c = 0 \ \forall\, c \in \mathcal{C}(s_{t+1}) \big\}. \tag{3}$$

In this setting, the constrained objective in Equation 2 can be relaxed to an unconstrained formulation, equivalent to standard MDPs.

2.3 Graph Grammars as CMDPs

The graph generation process introduced in Section 2.1 implies a CMDP in which each rewriting step corresponds to a state transition. The CMDP *state space* $\mathcal{S}$ is defined as the set of all admissible intermediate graph states reachable from the initial graph in T discrete steps under the grammar,

$$\mathcal{S} = \{\, G \mid G_0 \overset{T}{\underset{\mathcal{T}}{\Rightarrow}} G_T \,\}, \tag{4}$$

while satisfying the constraints in $\mathcal{C}$ at each successive application of $\mathcal{T}$. At a given state G_t, a safe hybrid action consists of selecting a graph rewriting action $a_t = (g_t, p_t, \theta_t),\ a_t \in \mathcal{A}_{\text{safe}}$ in line with the operator $\mathcal{T}$. The *transition kernel* is deterministic and directed by the graph transition operator $\mathcal{T}$,

$$P(G_{t+1} \mid G_t, a_t) = \mathbb{I}\{\, G_t \underset{\mathcal{T}}{\Rightarrow} G_{t+1} \,\}. \tag{5}$$

which satisfies the necessary Markov property for CMDPs $P(G_{t+1} \mid G_t, a_t, \ldots, G_0, a_0) = P(G_{t+1} \mid G_t, a_t)$, since transitions depend only on the current state and action.

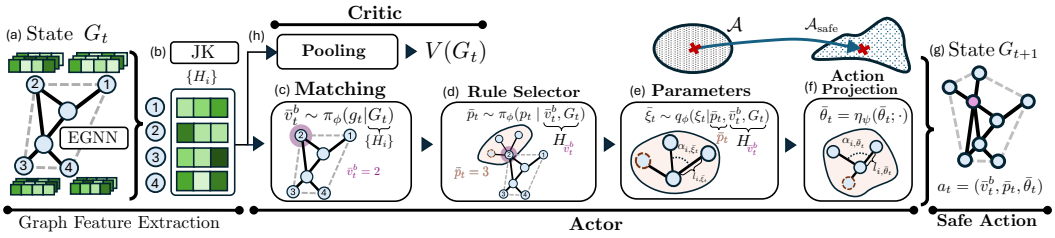


Figure 2: Illustration of the proposed Actor-Critic architecture.

Summary. Under the formulation proposed in this section, the graph generation process using the graph grammar is equivalent to a CMDP subject to hard, instantaneous constraints, where states are represented as graphs and actions are defined as graph rewriting rule applications. We leverage this formulation in Section 3 to develop a safe RL algorithm that preserves feasibility throughout generation.

3 Planar Graph Generation

Building on the CMDP formulation of the parametric graph grammar in Section 2, we formalize planar graph generation as a constrained sequential decision-making problem. The grammar defines admissible matches and available graph transformations, while a learned policy coordinates the choice of specific match, rule selection and assignments of parameter values to generate graphs optimized for user-specified objectives. Therefore, RL with Proximal Policy Optimization (PPO) [27] is employed to learn a policy that maps the current graph state to a distribution over grammar-based actions, enabling informed decisions over long trajectories τ. To efficiently account for constraints during learning and inference, we introduce a differentiable action projection mechanism that maps actions onto a learned safe set, enabling stable and sample-efficient policy optimization.

3.1 General Setup

The environment state at step t is the graph state G_t. The state encodes the full graph topology, the generative frontier, the planar embedding, and node features. The initial state G_0 is defined by two boundary nodes connected by an edge of length 1 (see Figure 1). Based on the preliminaries, the environment dynamics are deterministic for admissible actions and all stochasticity arises from the policy. We adopt an actor-critic architecture in which both the actor and the critic operate on a separate graph state encodings using a geometry-aware Graph Neural Network (GNN)-based feature extractor that maps G_t to node- and graph-level feature representations. An overview of the architecture is shown in Figure 2 and explained in detail, including training, in Appendix B.

3.2 Constraint Taxonomy

The graph generation process is subject to instantaneous feasibility constraints, which we explicitly categorize and relate to state-dependent safe action sets. We define the set of constraints as

$$\mathcal{C}(G_t) = \{c_\ell(G_t), c_\alpha(G_t), c_{\deg}(G_t), c_{\text{pl}}(G_t)\} \tag{6}$$

where the first constraint $c_\ell(G_t) = 0$ if $\ell(e) \in [\ell_{\min}, \ell_{\max}]\ \forall e \in E_t^d \cup E_t^b$, i.e. all edge lengths in the graph are within bounds, otherwise 1. The second constraint $c_\alpha(G_t) = 0$ if all $\alpha_i(v) > \alpha_{\min}\ \forall v \in V_t$ representing minimum bounds on all sector angles α_i between consecutive incident edges around all nodes v of G_t, otherwise 1. The third constraint $c_{\deg}(G_t) = 0$ if $\deg(v) \leq \deg_{\max}\ \forall v \in V_t$, otherwise 1, representing maximum degree bounds on nodes. The last constraint $c_{\text{pl}}(G_t)$ checks whether the graph is drawn as a valid planar embedding, i.e. that no edge intersections occur, returns 0 if so, otherwise 1. Hence, a graph transition $\mathcal{T}$ is only admissible if $c = 0\ \forall\, c \in \mathcal{C}(\mathcal{T}(G_t; a_t))$.

We further decompose the constraint set as $\mathcal{C} = \mathcal{C}_T \cup \mathcal{C}_G$ into topological and geometric constraints, respectively. Topological constraints $\mathcal{C}_T$ enforce the structural admissibility of an action and cannot be resolved by changing the values of continuous parameters θ_t after a rule p_t and a match g_t are determined. Such violations are intrinsic to the graph topology generated by a rewriting rule and therefore irreversible in our set-up. Geometric constraints $\mathcal{C}_G$, in contrast, correspond to violations

that can, in principle, be resolved by finding the appropriate values of θ_t given a rule p_t and a match g_t. These violations occur due to boundary edge-length violations in E_t^b or edge intersections. In practice, violations of $\mathcal{C}_T$ occur infrequently (see Appendix C.3) and thus can be simply handled by rejection sampling over the discrete components of the policy. In contrast, violations of $\mathcal{C}_G$ occur at a high rate, particularly edge intersections, and naive rejection sampling would lead to severe sampling inefficiency. This motivates the introduction of an explicit action projection mechanism that maps nominal parametrization of rules directly to a safe set $\mathcal{A}(G_t) \to \mathcal{A}_{\text{safe}}(G_t)$, see Section 3.4.

3.3 State, Feature Extraction and Critic

Graph observation. At each discrete step, the environment returns the current graph state G_t with the corresponding node feature matrix $F_t \in \mathbb{R}^{|V_t| \times m}$. Node features consist of (i) a binary boundary node indicator b_i for each node, (ii) the integer node degree d_i, and (iii) a degree-invariant encoding of local sector angles using spherical harmonics, similar to [28, 29]. These are computed using the first $K = 4$ Fourier harmonics as

$$h_i = (\bar{c}_{i,1}, \bar{s}_{i,1}, \ldots, \bar{c}_{i,K}, \bar{s}_{i,K}) \in \mathbb{R}^8, \tag{7}$$

with $\bar{c}_{i,k} = \frac{1}{d_i} \sum_{m=1}^{d_i} \cos(k\alpha_{i,m})$ and $\bar{s}_{i,k} = \frac{1}{d_i} \sum_{m=1}^{d_i} \sin(k\alpha_{i,m})$. Finally, we add the absolute two-dimensional node coordinates to the node attributes as y_i, and stack $x_i = (b_i, d_i, h_i, y_i)$ into F_t.

Graph feature extraction and Critic. To enable decision-making over graph states, the graph observation is encoded using E(2)-Equivariant Graph Neural Networks (EGNN) [30] that jointly processes graph structure and planar embedding, where we extract the invariant features only. Prior to the EGNN, node attributes (b_i, d_i, h_i) are lifted to a higher dimension via a small Multilayer Perceptron (MLP). To accommodate growing graph sizes, we adopt the Jumping Knowledge (JK) mechanism [31] with attentional aggregation scaled by graph size. The resulting node embeddings are combined with a global graph representation obtained via attention pooling [32] and mean pooling. These are concatenated into a set of node-wise features $\{H_i\}_{i=1}^{|V_t|}$, which serve as input to the downstream agent. Further details on the feature extractor architecture are given in Appendix B.1.

The critic employs an equivalent encoder architecture which is trained separately. In this setting, only the global graph representation is used, augmented with a learned embedding of the graph-size $\log(1 + |V_t|)$, and passed through a small MLP to predict the state value $V(G_t)$.

3.4 Agent Design

We now define the hybrid agent design and the associated conditional policy, aligned with the semantics of the graph transition operator $\mathcal{T}(G_t; g_t, p_t, \theta_t)$. Architectural details on the policy networks are outlined in Appendix B.2.

Conditional hybrid policy. At each discrete step, the agent applies a graph rewriting action

$$a_t = (g_t, p_t, \theta_t) \ \in \ \mathcal{A}(G_t) \subseteq V_t^b \times \mathcal{P} \times \Theta(p_t), \tag{8}$$

where the policy is structured autoregressively. First, the agent samples a boundary node according to $\pi_\phi(g_t \mid G_t)$, implemented as a categorical distribution over boundary nodes masked by b_i and approximated using an MLP with input $\{H_i\}$. This yields $\bar{v}_t^b \sim \pi_\phi(g_t \mid G_t)$.

Conditioned on $\bar{v}_t^b$, the agent samples a rewriting rule from the categorical distribution $\bar{p}_t \sim \pi_\phi(p_t \mid \bar{v}_t^b, G_t)$, which is another MLP with input $H_{\bar{v}_t^b}$, i.e. the node embedding for the selected rule application site. Since violations of $\mathcal{C}_T$ are rare at this stage (see Figure 7) but cannot be ruled out, we apply rejection sampling over p_t, advancing $t \to t + 1$ with zero reward and no state transition, while signaling rejection to PPO through the discount $\gamma < 1$. With this approach, we limit $(\bar{v}_t^b, \bar{p}_t)$ to the safe subset of $V_t^b \times \mathcal{P}$.

Finally, conditioned on $(\bar{v}_t^b, \bar{p}_t)$, the agent samples continuous parameters from $\bar{\theta}_t \ \sim \ \pi_\phi(\theta_t \mid \bar{v}_t^b, \bar{p}_t, G_t)$. This distribution is obtained by transforming a nominal Gaussian $q_\phi(\xi_t \mid \cdot) = \mathcal{N}(\mu_\phi, \sigma_\phi^2)$ via a push-forward $\tanh$-affine map. The Gaussian mean is predicted by an MLP with input $H_{\bar{v}_t^b}$ and $\bar{p}_t$, while the standard deviation is a global trainable parameter. For each rewriting rule,

the dimensionality of the Gaussian $\dim(\Theta(p))$ is different, so we apply masking to retrieve rule-dependent distribution dimensions. The $\tanh$-affine map projection serves two purposes: it enforces the boundedness $\theta_t \in (-1, 1)$, and it enables the approximate projection of nominal samples ξ_t into the geometric safe set $\mathcal{A}_{\text{safe}}$ induced by $\mathcal{C}_G$. We refer to this mechanism as *action projection*.

Action Projection. Nominal parametrizations ξ_t with infinite support are projected using a diffeomorphism $\eta_\psi : \mathbb{R}^{\dim(\Theta(p_t))} \to (-1, 1)^{\dim(\Theta(p_t))}$,

$$\theta_t = \tanh\big(s_\psi(G_t, g_t, p_t) \odot \xi_t + t_\psi(G_t, g_t, p_t)\big), \tag{9}$$

where s_ψ and t_ψ are learned, state-dependent, scaling and shift functions with parameters ψ. This construction is equivalent to a normalizing flow [33] with a single flow step followed by a $\tanh$ squashing. We found this one-step flow to be sufficient empirically. We further explain the construction of s_ψ and t_ψ in Section 3.5. Applying the action projection to the policy's distribution as $\pi_\phi(\theta_t | g_t, p_t, G_t) = \eta_{\psi\#} q_\phi(\xi_t \mid g_t, p_t, G_t)$ results in the push-forward measure

$$\pi_\phi(\theta_t \mid \cdot) = q_\phi(\xi_t \mid \cdot) \left|\det J_\psi(g_t, p_t, G_t)\right|^{-1}, \quad \det J_\psi(g_t, p_t, G_t) = \prod_i \Big(s_{\psi,i}\,(1 - \theta_{t,i}^2)\Big) \tag{10}$$

where the expression follows directly from the change-of-variables correction for densities and J_ψ denotes the Jacobian $\frac{\partial \eta_\psi}{\partial \xi_t}(\xi_t)$, which is diagonal since the projector acts element-wise.

Joint policy. The resulting joint policy $\pi_\phi(a_t \mid G_t)$ after action projection factorizes as

$$\begin{aligned} \pi_\phi(a_t \mid G_t) = \;& \pi_\phi(g_t \mid G_t)\, \pi_\phi(p_t \mid g_t, G_t)\, \pi_\phi(\theta_t \mid g_t, p_t, G_t) \\ =\; & \pi_\phi(g_t \mid G_t)\, \pi_\phi(p_t \mid g_t, G_t)\, q_\phi(\xi_t \mid g_t, p_t, G_t) \left|\det J_\psi(g_t, p_t, G_t)\right|^{-1}, \end{aligned} \tag{11}$$

which is then used in PPO for policy learning. To encourage exploration under PPO via an entropy bonus, we compute the entropy using the same factorization:

$$\mathcal{H}[\pi_\phi] = w_g\, \mathcal{H}[g_t \mid G_t] + w_p\, \mathbb{E}_{g_t \sim \pi_\phi(\cdot)}\big[\mathcal{H}[p_t \mid g_t, G_t]\big] + w_\xi\, \mathbb{E}_{g_t, p_t \sim \pi_\phi(\cdot)}\big[\mathcal{H}[\xi_t \mid p_t, g_t, G_t]\big] \tag{12}$$

We omit the Jacobian term due to missing closed-form entropy and because its Monte Carlo estimation can be noisy, and compute the continuous entropy over the nominal distribution $q_\phi(\xi_t \mid \cdot)$. Categorical and continuous components are weighted by $w_{(\cdot)}$ (set to 0.9 and 0.1, respectively). Policy MLPs are initialized with small final-layer weights and zero bias to encourage near-uniform initial exploration.

3.5 Action Projection Learning

The projection in Equation 9 specifies *how* nominal samples ξ_t are mapped into bounded parameters $\theta_t \in (-1, 1)$, but it does not by itself specify *where* to project a given ξ_t in order to satisfy geometric constraints. To obtain supervision for learning a projector that targets the admissible safe set $\mathcal{A}_{\text{safe}}$, we run an efficient implementation of Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) [34] inside the environment during graph rewriting on the continuous parameters θ'_t, with the projections' initial guess $\bar{\theta}_t$, to resolve potential violations of $\mathcal{C}_G$ using the formulation

$$\theta_t^\star = \arg\min_{\theta'_t} \; \|\theta'_t - \bar{\theta}_t\|_2 \qquad \text{s.t.} \quad c(\mathcal{T}(G_t; g_t, p_t, \theta_t)) = 0 \quad \forall c \in \mathcal{C}_G, \tag{13}$$

which follows the paradigm of safety-layer approaches in safe RL [26, 35]. The key idea is to find a proximal $\theta_t^\star$ to the projection's previous guess $\bar{\theta}_t$, that satisfies the constraints, implemented via an explicit penalty-based formulation (see Equation 28 in the Appendix). The resulting candidate $\theta_t^\star$ is accepted only if all $c = 0$, otherwise we fall back to the rejection sampling scheme. Empirically, CMA-ES succeeds in 70–90% of cases to find such a $\theta_t^\star$. If the projection's guess $\bar{\theta}_t$ is already admissible, we set $\theta_t^\star = \bar{\theta}_t$. Then, using the corrected $\theta_t^\star$ obtained from CMA-ES and nominal samples $\bar{\xi}_t$, we assemble an off-policy replay buffer to train the projection $\eta_\psi : \bar{\xi}_t \mapsto \bar{\theta}_t^\star$ in a supervised setting alongside PPO, which continuously refines the projection through CMA-ES correction. Ultimately, the action projector converges toward the admissible safe-set projection, CMA-ES corrections become in-frequent and act only as a fallback, as shown in Appendix C.3, where we also provide computational overhead insights of CMA-ES. The projector architecture uses an equivalent graph feature extractor (as introduced in Section 3.3) for G_t. Given the node embedding at a selected $\bar{v}_t^b$ and a rule embedding for the selected $\bar{p}_t$, we use two small MLPs to compute the scale s_ψ and shift t_ψ, with $s_\psi > 0$. The projector is trained synchronously with respect to the policy parameters. Details about architecture and training are given in Appendix C.

3.6 Reward Design

The reward function measures how closely the current graph satisfies a set of user-defined objectives, expressed through selected metrics (Section 5). Let $\mathcal{M}$ denote this set. For each metric $m \in \mathcal{M}$, we compute a symmetric relative deviation (lower is better) where $\kappa^\star$ and κ denote the target and current values of metric m, respectively, and $\epsilon > 0$ ensures numerical stability. The multiobjective reward r, which encourages solutions that jointly satisfy all metrics, is then defined as

$$r = 1 - \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} d_m, \qquad d_m = \frac{|\kappa^\star - \kappa|}{|\kappa^\star| + |\kappa| + \epsilon} \in [0, 1]. \tag{14}$$

4 Related Works

Classical graph models such as Erdős-Rényi or Barabási-Albert graphs [11, 12], as well as specialized planar graph generators [14, 15], provide useful baselines and, in some cases, planarity guarantees at the abstract graph level. However, they are not designed for goal-directed multi-objective optimization or for constructing task-optimized planar embeddings under hard geometric constraints. Deep graph generative models can learn graph distributions with high distributional fidelity [10, 16, 17], and conditional or guided variants can bias samples toward desired properties [36, 37]. Models such as DiGress and GDSS with Prodigy guidance generate abstract graph structures and can be adapted to multi-objective and single-objective property-conditioned generation, respectively [38, 39, 40]. However, enforcing exact graph size, planarity, and hard structural constraints through conditioning remains difficult, with validity often relying on rejection sampling, post-processing, or external guidance [18]. TreeDiff [18] extends pretrained graph diffusion models with verifier-guided tree search at inference time but adapting it to embedded planar graphs requires task-specific modifications to the diffusion backbone, intermediate-state verifier, and search procedure. Geometry-aware models such as NeVAE and GeoDiff primarily target molecular domains and rely on chemistry-specific validity assumptions [41, 42]. MuDiff more directly couples discrete structure and continuous coordinates [43], but feasibility is still learned implicitly from data rather than enforced as a construction constraint.

Reinforcement learning provides a natural framework for goal-directed graph construction through sequential decisions [44, 45]. It has been widely used in molecular graph generation, where policies add atoms and bonds under chemistry-specific validity [2, 19, 20, 21]. Grammar- and rule-based RL has been applied to molecular optimization, where policies select productions from molecular grammars to preserve chemical validity [23, 46], to structural design, where grammar actions preserve domain-specific structural constraints [5, 8]. Outside generative settings, many graph-based RL methods focus on tasks such as adversarial attacks, knowledge-graph completion, or relational reasoning rather than generic graph generation [45]. It has also been applied to computation-graph optimization, where RL selects rules for neural-network subgraph rewriting [47]. These works demonstrate the value of structured action spaces. However, they mainly address discrete and structural domain-specific construction. Our distinguishing setting is the goal-directed generation of planar embedded graphs using hybrid discrete-continuous action spaces: the policy jointly selects a graph match, a rewriting rule, and safe continuous geometric parameters, thereby coupling discrete topological graph rewriting with continuous embedding optimization under state-dependent topological and geometric constraints.

5 Experiments

We evaluate our method in two stages. First, we introduce four multi-objective benchmarks to assess the method's performance under varying geometric, topological and spectral objective as well as different constraint tightness. Second, we ablate the proposed action projection on a single-objective graph generation task and perform a scalability study. All results are averaged over 5 seeds with 1-sigma standard deviations reported for the rewards achieved by our method. Training was performed on an NVIDIA RTX PRO 6000 GPU, while the environment ran on an AMD 16-Core CPU; runtimes ranged from 10 minutes to 2 hours depending on benchmark complexity.

5.1 Benchmarks

To the best of our knowledge, no standardized benchmarks have been established for constrained, goal-directed generation of planar embedded graphs. Accordingly, the proposed benchmark suite comprises four heterogeneous tasks combining geometric, topological, and spectral metrics for constrained multi-objective optimization and varying graph sizes $|G|$, where all metrics used are defined in detail in Appendix D. Table 1 reports benchmark results and compares our method against the three best baselines out of all 10 reported in Appendix G. This includes the best results from the *House of Graphs* (HOG) dataset [48], which contains more than $4{,}000$ planar graphs with graph size up to 250. Since HOG graphs are not embedded, we compute embeddings using a force-directed spring layout [49]. Further, we compare against three diffusion-based generators; DiGress, GDSS, and MuDiff [38, 39, 43] on the same dataset and report the strongest contender for each benchmark. These models were selected as they allow for conditional generation, in the case of DiGress and MuDiff, while GDSS can be guided towards graph size using Prodigy [40]. The classical generator baselines used are described in Appendix E. For all classical baselines, we apply ranking-based selection to retrieve graphs satisfying the most constraints and, among them, maximize the reward. For all deep generative baselines, we evaluate from 1000 generated candidates. After retaining the largest connected component and filtering out non-planar graphs, candidates are ranked first by proximity to the target graph size and then by benchmark reward. The remaining geometric and structural constraint statistics are evaluated and reported, but they are not used for candidate selection. This alternative protocol for deep generative models is motivated by the fact that we empirically found diffusion models to rarely satisfy the constraint set, and prioritizing constraint satisfaction would substantially reduce the reward performance of these models. We consequently exclude the constraint satisfaction criterion from the ranking procedure for diffusion baselines to enable a fairer comparison of reward metrics. More details on the evaluation protocol are available in Appendix F. For benchmark 1, 2, 4, we enforce a tight constraint set $\mathcal{C}_2$ with $\ell \in [0.75, 2.5]$, $\alpha_{\min} = 20°$, and $\deg_{\max} = 8$. For benchmark 3, we use the relaxed constraint set $\mathcal{C}_1$ with $\ell \in [0.1, 3.5]$, $\alpha_{\min} = 1°$ and $\deg_{\max} = 15$ to allow for fairer comparison against unconstrained baselines.

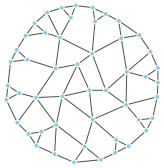
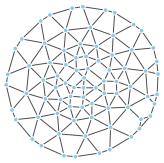
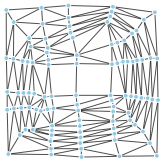
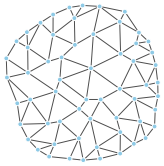
Resilient power grid in circular hull. This benchmark generates a 50-node power-grid-like graph inside a circular hull, measured by the isoperimetric ratio ($r_{\text{iso}} = 1.0$). The objective combines shape matching, low degree inequality ($c_{\text{gini}} = 0.1$), moderate clustering ($c_{\text{clust}} = 0.2$), and normalized spectral gap ($\lambda_2 = 1.0$) measuring overall connectivity. Here, the target $\lambda_2 = 1.0$ is idealistic; planar, bounded-degree, geometrically constrained graphs have much lower attainable spectral gaps, so this term acts primarily as a connectivity pressure. Results show that our method balances these competing metrics best, while satisfying all constraints. Note here, that Delaunay triangulation, representing a maximally connected planar embedded graph, achieves highest spectral gap, which is this baselines natural strength, but does so at the cost of clustering and constraint satisfaction.

Urban road grid in circular hull. This benchmark synthesizes an 80-node road-network-like graph in a circular hull. The objective combines shape matching (r_{iso}), edge-length regularity ($S_l = 0.7$), angular resolution ($\text{AR} = 0.8$), and low degree inequality ($c_{\text{gini}} = 0.2$). Although Delaunay triangulation shows a better match on two individual metrics, it does so by strongly underperforming on angular resolution and falls behind in total reward while also failing to satisfy 2/4 constraints.

Strongly connected mesh in square hull. This benchmark generates a 100-node mesh in a square hull with high connectivity, combining a squareness objective ($r_{\text{square}} = 1.0$) with a spectral-gap objective λ_2. Similar to benchmark 1, the goal of $\lambda_2 = 0.3$ is chosen arbitrarily as a maximization pressure on connectivity. Lattice graphs naturally achieve perfect squareness, but have lower spectral gap. Our method provides the best overall trade-off while satisfying all constraints.

Graph with 40 triangles in circular hull. This benchmark targets a 60-node mesh in a circular hull with exactly 40 triangles. Unlike standard triangulation methods, the task requires a specific combinatorial face. Our method closely matches the triangle target on average and satisfies all constraints, whereas HOG substantially overshoots the triangle count.

Table 1: Benchmark results across four planar graph generation tasks. Each graph corresponds to the result obtained by our method. The table shows averaged multi-objective reward, the individual metrics, graph sizes and number of satisfied constraints for each benchmark. Evaluations for all 10 baselines are reported in Appendix G, complemented with per-baseline graph visualizations. Relative metric deviations from the targets are additionally provided in Appendix Table 12.

Benchmark	Graph	Method	Reward	Metrics				Constr.	
			r	r_{iso}	c_{gini}	c_{clust}	λ_2	$\lvert G\rvert$	$\mathcal{C}_2$
1. Resilient Power Grid		Target	1.0	1.0	0.1	0.2	1.0	50	✓
		Ours	**0.76**± 0.004	**0.96**	**0.09**	**0.2**	0.05	**50**	✓ (4/4)
		HOG	0.38	0.84	0.03	0.0	0.08	47	✗ (3/4)
		Delaunay	0.69	0.93	0.18	0.22	**0.09**	**50**	✗ (2/4)
		GDSS	0.56	0.50	0.2	0.16	0.01	43	✗ (2/4)
			r	r_{iso}	S_l	AR	c_{gini}	$\lvert G\rvert$	$\mathcal{C}_2$
2. Urban Road Grid		Target	1.0	1.0	0.7	0.8	0.2	80	✓
		Ours	**0.94**± 0.01	**0.95**	0.82	**0.76**	0.16	**80**	✓ (4/4)
		HOG	0.78	0.4	0.83	0.49	0.16	**80**	✗ (1/4)
		Delaunay	0.91	**0.95**	**0.76**	0.45	**0.2**	**80**	✗ (2/4)
		MuDiff	0.53	0.34	0.01	0.46	0.25	56	✗ (1/4)
			r	r_{square}	λ_2	–	–	$\lvert G\rvert$	$\mathcal{C}_1$
3. Strongly Connected Mesh		Target	1.0	1.0	0.3	–	–	100	✓
		Ours	**0.65**± 0.01	0.89	**0.06**	–	–	**100**	✓ (4/4)
		HOG	0.36	0.54	0.00	–	–	101	✗ (3/4)
		Lattice	0.59	**1.0**	0.03	–	–	**100**	✓ (4/4)
		GDSS	0.39	0.55	0.01	–	–	72	✗ (3/4)
			r	r_{iso}	T	–	–	$\lvert G\rvert$	$\mathcal{C}_2$
4. Graph with 40 Triangles		Target	1.0	1.0	40	–	–	60	✓
		Ours	**0.98**± 0.02	**0.92**	**39.91**	–	–	**60**	✓ (4/4)
		HOG	0.86	0.89	64	–	–	**60**	✓ (4/4)
		Boltzmann	0.88	0.63	38	–	–	**60**	✗ (0/4)
		DiGress	0.57	0.42	15	–	–	10	✗ (2/4)

5.2 Ablation and Scalability Study

We further evaluate action projection and scalability on a single squareness objective r_{square} over varying graph sizes. Figure 3 shows that CMA-ES-supervised action projection improves terminal reward by steering the policy away from infeasible continuous actions, allowing the agent to focus on reward optimization. This also reduces constraint violations by a factor of 15 compared to the projection-free variant; detailed statistics are provided in Appendix C.3. Figure 4 evaluates the reward of graph sizes up to 150 nodes under two constraint regimes. As expected, tighter constraints and larger graph sizes reduce achieved reward. However, the method still generates feasible square-like graphs at graph sizes substantially larger than common graph diffusion models operating on planar graph datasets, which typically use at most 64 nodes [38, 50].

6 Limitations & Future Work

While our experiments generate graphs with up to 150 nodes (Figure 4), scalability remains limited by the growing combinatorial grammar design space for longer decision horizons. This limitation should be viewed relative to deep generative baselines, which often lose validity or efficiency on larger graphs under strict structural constraints such as planarity, as shown in Appendix G. Tackling this scalability ceiling is left for future work, where a principled approach could be the adoption of hierarchical reinforcement learning [51] to reduce the effective decision horizon. A higher-level

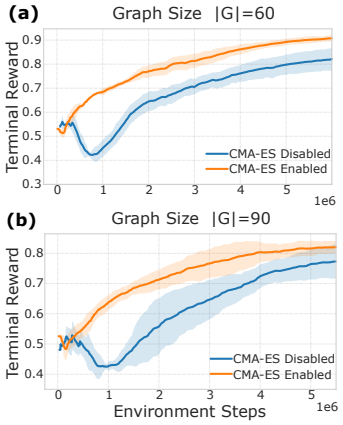


Figure 3: Ablation study of action projection with CMA-ES enabled, vs. disabled for varying target graph sizes.

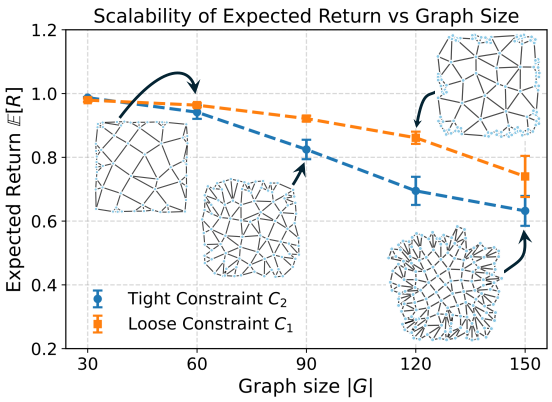


Figure 4: Scalability study of our method for different graph sizes and constraint tightness.

policy would select actions that define larger structural changes, while a lower-level policy would apply the existing grammar to maximize the intrinsic reward obtained by the high-level policy. Such an extension could also facilitate applications to constrained, goal-directed mesh generation, where substantially larger graphs are typically required. A second limitation is the structural prior imposed by the grammar: the method can only generate graphs expressible by the chosen rule set. We therefore do not claim completeness with respect to all planar embedded graphs, and results should be interpreted within the assumptions encoded by the grammar. At the same time, the rule set is also a strength in domain-specific settings, where hand-crafted rules encode system validity by design [52]. However, deriving grammars with improved formal coverage of planar graph embeddings is an important direction for future work.

7 Conclusions

We introduced a grammar-based reinforcement learning method and benchmark suite that, to our knowledge, is the first to combine goal-directed generation of embedded planar graphs with hard structural and geometric constraint enforcement. By formulating graph rewriting as a constrained Markov decision process, the method constructs topology and geometry jointly while enforcing feasibility through grammar-induced admissibility and learned action projection. Across four benchmarks, our method generates graphs with exact prescribed node counts, satisfies all constraints, and achieves the best overall multi-objective reward. Although a few baselines are competitive in the benchmarks, they do not generalize across other multi-objective tasks; by contrast, our method maintains strong rewards while satisfying the full constraint set in all benchmarks, indicating stronger generalization across heterogeneous benchmarks. On the other hand, diffusion-based generators struggle strongly to satisfy constraints while maximizing the reward. Beyond the metrics and constraints used here, the framework is designed to accommodate arbitrary user-specified objectives and constraints through the reward and constraint definitions.

Acknowledgments

This research was supported by the ETH Research Grant 23-1 ETH-35. Research supported by the NVIDIA Academic Grant Program using a NVIDIA RTX PRO 6000 Blackwell Max-Q Workstation Edition GPU.

References

[1] Marc Barthélemy. Spatial networks. *Physics Reports*, 499(1):1–101, 2011. doi: https://doi.org/10.1016/j.physrep.2010.11.002. URL https://www.sciencedirect.com/science/article/pii/S037015731000308X.

[2] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/d60678e8f2ba9c540798ebbde31177e8-Abstract.html.

[3] Geoff Boeing. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017. doi: https://doi.org/10.1016/j.compenvurbsys.2017.05.004. URL https://www.sciencedirect.com/science/article/pii/S0198971516303970.

[4] Reza Kakooee and Benjamin Dillenburger. Reimagining space layout design through deep reinforcement learning. *Journal of Computational Design and Engineering*, 11(3):43–55, June 2024. doi: 10.1093/jcde/qwae025. URL https://doi.org/10.1093/jcde/qwae025.

[5] Maximilian E. Ororbia and Gordon P. Warn. Design Synthesis Through a Markov Decision Process and Reinforcement Learning Framework. *Journal of Computing and Information Science in Engineering*, 22 (021002), July 2021. doi: 10.1115/1.4051598. URL https://doi.org/10.1115/1.4051598.

[6] Nicolas Hochuli and Tino Stankovic. Inverse Design of Origami for Trajectory Following Using Deep Learning. *Journal of Mechanisms and Robotics*, 17(091014), July 2025. ISSN 1942-4302. doi: 10.1115/1.4069199. URL https://doi.org/10.1115/1.4069199.

[7] Andreas Walker and Tino Stankovic. Algorithmic design of origami mechanisms and tessellations. *Communications Materials*, 3(1):4, January 2022. doi: 10.1038/s43246-022-00227-5. URL https://www.nature.com/articles/s43246-022-00227-5.

[8] Lazlo Bleker, Kam-Ming Mark Tam, and Pierluigi D'Acunto. A graph-based grammar for structural design using deep reinforcement learning. In *Proceedings of the IASS 2024 Symposium: Redefining the Art of Structural Design*, pages 1–10, Zurich, Switzerland, 2024. International Association for Shell and Spatial Structures. URL https://www.ingentaconnect.com/contentone/iass/piass/2024/00002024/00000011/art00014.

[9] Will Thacher, Yulong Pan, and Per-Olof Persson. Optimization of a triangular delaunay mesh generator using reinforcement learning. *Computer-Aided Design*, 189:103964, 2025. ISSN 0010-4485. doi: https://doi.org/10.1016/j.cad.2025.103964. URL https://www.sciencedirect.com/science/article/pii/S0010448525001253.

[10] Xiaojie Guo and Liang Zhao. A Systematic Survey on Deep Generative Models for Graph Generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(5):5370–5390, May 2023. doi: 10.1109/TPAMI.2022.3214832. URL https://ieeexplore.ieee.org/abstract/document/9920219.

[11] P Erdös and A Rényi. On random graphs I. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[12] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439): 509–512, 1999. doi: 10.1126/science.286.5439.509.

[13] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393: 440–442, 1998. doi: 10.1038/30918. URL https://www.nature.com/articles/30918. Publisher: Nature Publishing Group.

[14] Gunnar Brinkmann and Brendan D. McKay. Fast generation of planar graphs. In *MATCH-COMMUNICATIONS IN MATHEMATICAL AND IN COMPUTER CHEMISTRY*, volume 58, pages 323–357, 2007. URL http://hdl.handle.net/1854/LU-397541.

[15] Eric Fusy. Uniform random sampling of planar graphs in linear time. *Random Struct. Algorithms*, 35(4): 464–522, December 2009.

[16] Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative Diffusion Models on Graphs: Methods and Applications, August 2023. URL http://arxiv.org/abs/2302.02591. arXiv:2302.02591 [cs].

[17] Manuel Madeira, Clement Vignac, Dorina Thanou, and Pascal Frossard. Generative modelling of structurally constrained graphs, 2024. URL https://arxiv.org/abs/2406.17341.

[18] Jiachi Zhao, Zehong Wang, Yamei Liao, Chuxu Zhang, and Yanfang Ye. Controllable Graph Generation with Diffusion Models via Inference-Time Tree Search Guidance, October 2025. URL http://arxiv.org/abs/2510.10402. arXiv:2510.10402 [cs].

[19] Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation, February 2020. URL http://arxiv.org/abs/2001.09382. arXiv:2001.09382 [cs].

[20] Yash Khemchandani, Stephen O'Hagan, Soumitra Samanta, Neil Swainston, Timothy J. Roberts, Danushka Bollegala, and Douglas B. Kell. DeepGraphMolGen, a multi-objective, computational strategy for generating molecules with desirable properties: a graph convolution and reinforcement learning approach. *Journal of Cheminformatics*, 12(1), December 2020. doi: 10.1186/s13321-020-00454-3. URL https://jcheminf.biomedcentral.com/articles/10.1186/s13321-020-00454-3.

[21] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N. Zare, and Patrick Riley. Optimization of Molecules via Deep Reinforcement Learning. *Scientific Reports*, 9(1):10752, July 2019. doi: 10.1038/s41598-019-47148-x. URL http://arxiv.org/abs/1810.08678. arXiv:1810.08678 [cs].

[22] Gregor N. C. Simm, Robert Pinsler, and José Miguel Hernández-Lobato. Reinforcement Learning for Molecular Design Guided by Quantum Mechanics. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119 of *Proceedings of Machine Learning Research*, pages 8921–8931. PMLR, 2020. URL http://proceedings.mlr.press/v119/simm20b.html.

[23] Chencheng Xu, Qiao Liu, Minlie Huang, and Tao Jiang. Reinforced Molecular Optimization with Neighborhood-Controlled Grammars, November 2020. URL http://arxiv.org/abs/2011.07225. arXiv:2011.07225 [cs].

[24] Hans-Jorg Kreowski, Renate Klempien-Hinrichs, and Sabine Kuske. Some Essentials of Graph Transformation. *Studies in Computational Intelligence*, 2006. doi: 10.1007/978-3-540-33461-3_9.

[25] Eitan Altman. *Constrained Markov Decision Processes: Stochastic Modeling*. Routledge, Boca Raton, 1 edition, December 2021. ISBN 978-1-315-14022-3. doi: 10.1201/9781315140223. URL https://www.taylorfrancis.com/books/9781315140223.

[26] Yongshuai Liu, Avishai Halev, and Xin Liu. Policy learning with constraints in model-free reinforcement learning: A survey. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4508–4515. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi: 10.24963/ijcai.2021/614. URL https://doi.org/10.24963/ijcai.2021/614. Survey Track.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL http://arxiv.org/abs/1707.06347.

[28] Yi Liu, Limei Wang, Meng Liu, Xuan Zhang, Bora Oztekin, and Shuiwang Ji. Spherical Message Passing for 3D Graph Networks, 2022. URL http://arxiv.org/abs/2102.05013. arXiv:2102.05013 [cs].

[29] Johannes Gasteiger, Janek Groß, and Stephan Günnemann. Directional Message Passing for Molecular Graphs, April 2022. URL http://arxiv.org/abs/2003.03123. arXiv:2003.03123 [cs].

[30] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) Equivariant Graph Neural Networks, 2022. URL http://arxiv.org/abs/2102.09844. arXiv:2102.09844 [cs].

[31] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks, 2018. URL http://arxiv.org/abs/1806.03536. arXiv:1806.03536 [cs].

[32] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph Matching Networks for Learning the Similarity of Graph Structured Objects, May 2019. URL http://arxiv.org/abs/1904.12787. arXiv:1904.12787 [cs].

[33] Danilo Jimenez Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows, June 2016. URL http://arxiv.org/abs/1505.05770. arXiv:1505.05770 [stat].

[34] Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, June 2001. doi: 10.1162/106365601750190398. URL https://ieeexplore.ieee.org/document/6790628.

[35] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe Exploration in Continuous Action Spaces, January 2018. URL http://arxiv.org/abs/1801.08757. arXiv:1801.08757 [cs].

[36] Jérôme Kunegis, Jun Sun, and Eiko Yoneki. Guided Graph Generation: Evaluation of Graph Generators in Terms of Network Statistics, and a New Algorithm, March 2023. URL http://arxiv.org/abs/2303.00635. arXiv:2303.00635 [cs].

[37] Andreas Bergmeister, Karolis Martinkus, Nathanaël Perraudin, and Roger Wattenhofer. Efficient and Scalable Graph Generation through Iterative Local Expansion, May 2024. URL http://arxiv.org/abs/2312.11529. arXiv:2312.11529 [cs].

[38] Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete Denoising diffusion for graph generation, May 2023. URL http://arxiv.org/abs/2209.14734. arXiv:2209.14734 [cs].

[39] Jaehyeong Jo, Seul Lee, and Sung Ju Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations, 2022. URL https://arxiv.org/abs/2202.02514.

[40] Kartik Sharma, Srijan Kumar, and Rakshit Trivedi. Diffuse, sample, project: Plug-and-play controllable graph generation. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 44545–44564. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/sharma24b.html.

[41] Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs, 2019. URL https://arxiv.org/abs/1802.05283.

[42] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: a geometric diffusion model for molecular conformation generation, 2022. URL https://arxiv.org/abs/2203.02923.

[43] Chenqing Hua, Sitao Luan, Minkai Xu, Rex Ying, Jie Fu, Stefano Ermon, and Doina Precup. Mudiff: Unified diffusion for complete molecule generation, 2024. URL https://arxiv.org/abs/2304.14621.

[44] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Goal-directed graph construction using reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2254):20210168, October 2021. doi: 10.1098/rspa.2021.0168. URL http://arxiv.org/abs/2001.11279. arXiv:2001.11279 [cs].

[45] Mingshuo Nie, Dongming Chen, and Dongqi Wang. Reinforcement Learning on Graphs: A Survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(4):1065–1082, August 2023. doi: 10.1109/TETCI.2022.3222545. URL https://ieeexplore.ieee.org/document/10121200/.

[46] Egor Kraev. Grammars and reinforcement learning for molecule optimization, November 2018. URL http://arxiv.org/abs/1811.11222. arXiv:1811.11222 [cs].

[47] Guoliang He, Sean Parker, and Eiko Yoneki. X-RLflow: Graph Reinforcement Learning for Neural Network Subgraphs Transformation.

[48] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. House of Graphs: A database of interesting graphs. *Discrete Applied Mathematics*, 161:311–314, 2013. doi: 10.1016/j.dam.2012.07.018. URL https://www.sciencedirect.com/science/article/pii/S0166218X12002831.

[49] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21(11):1129–1164, 1991. doi: https://doi.org/10.1002/spe.4380211102. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.4380211102.

[50] Karolis Martinkus, Andreas Loukas, Nathanaël Perraudin, and Roger Wattenhofer. SPECTRE: Spectral Conditioning Helps to Overcome the Expressivity Limits of One-shot Graph Generators, June 2022. URL http://arxiv.org/abs/2204.01613. arXiv:2204.01613 [cs].

[51] Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

[52] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 301–308, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 158113374X. doi: 10.1145/383259.383292. URL https://doi.org/10.1145/383259.383292.

[53] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[54] Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, April 2019. URL http://arxiv.org/abs/1903.02428. arXiv:1903.02428 [cs].

[55] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[56] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

[57] Robert Tjarko Lange. evosax: JAX-based Evolution Strategies, December 2022. URL http://arxiv.org/abs/2212.04180. arXiv:2212.04180 [cs].

[58] Mikhail Drobyshevskiy and Denis Turdakov. Random Graph Modeling: A survey of the concepts. *ACM Computing Surveys*, 52(6):1–36, November 2019. doi: 10.1145/3369782. URL http://arxiv.org/abs/2403.14415.

[59] Gavin J. Mooney, Helen C. Purchase, Michael Wybrow, and Stephen G. Kobourov. The Multi-Dimensional Landscape of Graph Drawing Metrics. In *2024 IEEE 17th Pacific Visualization Conference (PacificVis)*, pages 122–131, April 2024. doi: 10.1109/PacificVis60374.2024.00022. URL https://ieeexplore.ieee.org/document/10541779.

[60] Gavin J. Mooney, Tim Hegemann, Alexander Wolff, Michael Wybrow, and Helen C. Purchase. Universal Quality Metrics for Graph Drawings: Which Graphs Excite Us Most? In Vida Dujmović and Fabrizio Montecchiani, editors, *33rd International Symposium on Graph Drawing and Network Visualization (GD 2025)*, volume 357, pages 30:1–30:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi: 10.4230/LIPIcs.GD.2025.30. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.GD.2025.30.

[61] Helen C. Purchase. Metrics for Graph Drawing Aesthetics. *Journal of Visual Languages & Computing*, 13 (5):501–516, October 2002. doi: 10.1006/jvlc.2002.0232. URL https://www.sciencedirect.com/science/article/pii/S1045926X02902326.

[62] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, July 1983. doi: 10.1109/TIT.1983.1056714. URL https://ieeexplore.ieee.org/document/1056714/.

[63] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring Network Structure, Dynamics, and Function using NetworkX. pages 11–15, Pasadena, California, June 2008. doi: 10.25080/TCWV9851. URL https://doi.curvenote.com/10.25080/TCWV9851.

[64] Tom AB Snijders. Markov chain Monte Carlo estimation of exponential random graph models. *Journal of Social Structure*, 3(2):1–40, 2002. URL http://hbanaszak.mjr.uw.edu.pl/TempTxt/Mcpstar.pdf.

[65] Tobias Winkler. boltzmann-planar-graph: Python implementation of uniform random planar graph sampler, 2020. URL https://github.com/towink/boltzmann-planar-graph. Accessed: 2026-01-21.

A Grammar Rules

This appendix specifies the seven graph rewriting rules defined in the grammar $\mathcal{G}$. Each rule $p \in \mathcal{P}$ acts on a single boundary node $\bar{v}_t^b$ determined by the match g_t and is applied via the graph transition operator $\mathcal{T}$ to the graph state G_t. A rule may introduce new embedded nodes and edges connecting those to $\bar{v}_t^b$, parameterized geometrically by a rule-specific continuous vector $\theta_t \in \Theta(p_t)$ that encodes normalized sector angles and edge lengths (cf. Appendix A.3). All rules update both the directed edge set E_t^d and the node set V_t. Prior to each rule application, the boundary edges E_t^b in G_t are removed as part of $\mathcal{T}$, and reintroduced afterwards according to the connecting procedure (cf. Appendix A.2). This connecting procedure identifies how the leaf nodes in V_t^b in the current graph state are connected using boundary edges, such that the resulting graph is planar and geometrically closed. These boundary edges then populate the edge set E_t^b. The rule semantics and notations are introduced in Appendix A.1. A schematic overview of all rules in $\mathcal{P}$ is shown in Figure 5 and Table 2.

A.1 Rule semantics and notation in Figure 5 and Table 2.

Each rule $p^{(\cdot)} \in \mathcal{P}$. With reference to Section 2.1, a rule $p^{(\cdot)} : L \to R$ rewrites a local neighborhood (left-hand side L) in the graph state G_t around a boundary node $\bar{v}_t^b$ identified by the match g_t with a new subgraph R. In the visualizations of all rules in Figure 5, the top row depicts the graph state G_t prior to a rule application with the highlighted subgraph L (i.e. the match $\bar{v}_t^b$), which can be rewritten with a rule and the graph transition operator $\mathcal{T}$. Additionally, we depict optional incoming directed edges to $\bar{v}_t^b$ using dashed edges. This represents edges that may, or may not be present at the rule application site, implying that any number of existing incoming directed edges may be valid at a selected match $\bar{v}_t^b$, as long as the degree constraint $c_{\deg}$ (see Section 3.2) is satisfied.

The bottom row in Figure 5 depicts the graph state G_{t+1} where the right-hand side R of the rules has been applied. The newly added edges are depicted as outward directed edges from $\bar{v}_t^b$ in red with new nodes with a red outline. Several rules allow to merge an edge to an existing node in the graph G_t (i.e. rules 2,3,4,6,7). This merging mechanism is enabled by the connecting procedure (cf. Appendix A.2). Figure 5 highlights the merged nodes in R as dashed-outlined nodes.

Each rule is associated with a parameter space $\theta_t \in \Theta(p_t)$ that defines the geometric parametrization of angles α_i (sector angles) and lengths ℓ_i (line segments) in R. In the bottom row of Figure 5, we depict the set of sector angles occupied by edges that are present at $\bar{v}_t^b$ prior to rule applications using the set $\mathcal{B}$. Hence, the angular parameter space of $\Theta(p_t)$ is restricted to the remaining angular budget $B = 2\pi - \sum_{k \in \mathcal{B}} \alpha_k$.

Boundary edges E_t^b are not depicted in Figure 5, as they are removed before any rule application by the transition operator $\mathcal{T}$ and reintroduced afterward by use of the connecting procedure (cf. Appendix A.2). Note that, when discarding the undirected boundary edge set, the described rule-set creates Directed Acyclic Graphs (DAG) by design, since rules are applied uniquely at leaf nodes in a directed graph. Additionally, merge-able nodes, identified by the connecting procedure are uniquely leaf nodes, too, therefore, the application of the rewriting rules cannot introduce directed cycles and the grammar $\mathcal{G}$ will always produce a DAG.

Table 2: Summary of grammar rules $\mathcal{P}$. "Merge" refers to connections identified by the connecting procedure (cf. Figure 6) and "Sprout" refers to newly added nodes. The table additionally shows the dimensionality of the geometric parametrization.

Rule set $\mathcal{P}$	Description	New nodes	Merges	$(\#\alpha, \#\ell)$	$\dim \Theta$
$p^{(1)}$	Triple node sprout	3	0	$(4, 3)$	7
$p^{(2)}$	Single sprout + two merges	1	2	$(2, 1)$	3
$p^{(3)}$	Merge left + double sprout	2	1	$(3, 2)$	5
$p^{(4)}$	Merge right + double sprout	2	1	$(3, 2)$	5
$p^{(5)}$	Double node sprout	2	0	$(3, 2)$	5
$p^{(6)}$	Merge left + single sprout	1	1	$(2, 1)$	3
$p^{(7)}$	Merge right + single sprout	1	1	$(2, 1)$	3

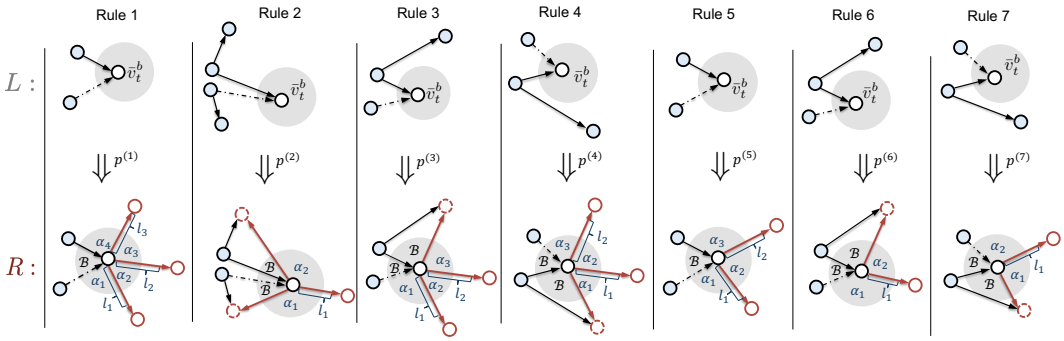


Figure 5: Schematic illustration of the seven grammar rules in $\mathcal{P}$. The top row represent a graph state G_t and the highlighted left-hand side L (i.e. the match $\bar{v}_t^b$) of a rule $p^{(\cdot)}$ prior to a rule application. The bottom row shows the graph state G_{t+1} after a rule has been applied, with the right-hand side R shown as red outward directed edges, the red outlined newly added nodes, and merged nodes as dashed outlined nodes. Geometric parameters are indicated in the bottom row.

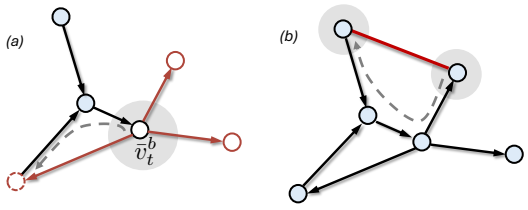


Figure 6: Visualization of the connecting procedure. (a) The connecting procedure identifies a merge-able node, starting from $\bar{v}_t^b$. The merge-able node is depicted as the dashed-outlined node. (b) The connecting procedure identifies a boundary edge, depicted by the undirected edge between the highlighted nodes.

A.2 Connecting Procedure cf. Figure 6

The *connecting procedure* is a deterministic geometric routine that serves two purposes: (i) it identifies merge-able nodes that can be connected to from a match $\bar{v}_t^b$ by rules with merging semantics, while ensuring topological planarity (cf. Figure 6(a)), (ii) it reconstructs the graph's boundary edges after each rule application so that the updated graph remains planar and geometrically closed (cf. Figure 6(b)).

The procedure operates exclusively on the current embedded graph state and depends only on the node positions and the directed edge configuration E_t^d. In case (i), the procedure is initialized at the match $\bar{v}_t^b$ and geometrically traverses the connected edges in E_t^d, starting from $\bar{v}_t^b$, in either clock-wise or counter-clock-wise direction, depending on the selected rule. The traversal terminates upon finding a boundary node, which is then labeled as the node to merge to. This is indicated in the figure using the dashed-curved arrow (traversal process) and the dashed-outlined node (merge-able node) in Figure 6(a). In case (ii), the procedure is initialized at any boundary node and follows the same traversal principle until another boundary node is found. Then, a boundary edge connects both boundary nodes, depicted as the undirected edge between the highlighted nodes in Figure 6(b). Applying this process to all boundary nodes guarantees that the resulting boundary edge set E_t^b forms a non-self-intersecting polygonal chain that encloses the graph.

A.3 Rule Parametrization

In the implementation, $\mathcal{T}(G_t; g_t, p_t, \theta_t)$ maps the normalized parameters $\theta_t \in (-1, 1)^{\dim(\Theta(p_t))}$ to Cartesian coordinates. For each rule p_t, the parameter space $\Theta(p_t)$ is decomposed into angular

parameters $\Theta_\alpha(p_t)$ and length parameters $\Theta_\ell(p_t)$, which are denormalized independently using the procedures described below.

Sector angles. Rewrite rules introduce a finite set of new sector angles around $\bar{v}_t^b$. Let $\mathcal{B}$ denote the set of angles already occupied by incoming directed edges at $\bar{v}_t^b$ (cf. Figure 5), and let $B = 2\pi - \sum_{k\in\mathcal{B}} \alpha_k$ be the remaining angular budget. For a rule that requires explicit angles n plus one closing angle, let $m = n + 1$ and let $\theta_\alpha \in (0,1)^m$ denote the normalized angular parameters obtained by rescaling the policy outputs from $(-1,1)$. A minimum angle $\alpha_{\min} > 0$ is enforced by reserving $\alpha_{\min}$ per angle, yielding free budget $B_{\text{free}} = \max(B - m\alpha_{\min}, 0)$. Denormalized angles are computed as

$$u_i = \theta_{\alpha_i}^2, \qquad \tilde{u}_i = \frac{u_i}{\sum_j u_j + \delta} B_{\text{free}}, \qquad \alpha_i = \alpha_{\min} + \tilde{u}_i, \tag{15}$$

with $\delta > 0$ a small constant. This guarantees $\alpha_i \geq \alpha_{\min}$ and $\sum_i \alpha_i \leq 2\pi$, with the final angle acting as a closing angle.

Edge lengths. Each rule specifies a fixed number $\hat{L}$ of newly introduced edges and thus $\hat{L}$ corresponding length parameters $\theta_\ell \in (0,1)^{\hat{L}}$ obtained by rescaling the policy outputs from $(-1,1)$. Physical edge lengths are given by

$$\ell = \ell_{\min} + \theta_\ell(\ell_{\max} - \ell_{\min}), \tag{16}$$

where $\ell_{\min}$ and $\ell_{\max}$ are global bounds from the constraint set $\mathcal{C}$.

Geometric realization. Let $\hat{r}$ denote the incoming reference direction at $\bar{v}_t^b$, conventionally taken as the first incident directed edge in counter-clockwise direction that sits within the polygon formed by the boundary edges E_t^b, and $R(\alpha)$ a planar rotation by angle α with respect to the last incident edge in counter-clockwise direction within the polygon E_t^b. All newly introduced vertices are placed in the Euclidean plane as

$$y_{\text{new}} = y_{\bar{v}_t^b} + \ell \, \frac{R(\alpha)\hat{r}}{\|R(\alpha)\hat{r}\|_2}. \tag{17}$$

A.4 Implementation

The complete grammar and graph rewriting logic is implemented in JAX [53], enabling compiled and efficient implementation of the transition operator. Low-level geometric operations, in particular planarity checks based on pairwise edge-intersection tests, are formulated in a vectorized manner and evaluated in parallel. The grammar execution runs on the CPU, as the control flow of the grammar is predominantly sequential with sparse data structures that benefit more from high multi-core CPU throughput than from GPU parallelism.

B Actor–Critic Architecture

In this appendix, we describe the complete actor–critic architecture in detail. The whole architecture is parametrized with neural networks, denoted with ϕ.

B.1 Graph Feature Extractor

First we describe the geometry-aware graph feature extractor used by the actor–critic networks. The extractor maps an embedded graph state G_t to node-level and graph-level representations suitable for decision-making, while respecting the geometric structure of the graph.

Input representation. The extractor operates on a batched graph representation. Each node $v \in V_t$ is associated with a feature vector

$$x_v = [\, b_v,\ d_v,\ h_v,\ y_v \,], \tag{18}$$

where $b_v \in \{0,1\}$ is a boundary indicator, $d_v \in \{0,\dots,D_{\max}\}$ is the (discrete) degree, and $h_v \in \mathbb{R}^{2K}$ is a precomputed sector-angle embedding using $K = 4$ Fourier harmonics (cf. Figure 3.3). Each node additionally carries planar coordinates $y_v \in \mathbb{R}^2$. Edges are treated as undirected for the purpose of message passing.

B.1.1 Node-level Embeddings

Discrete and continuous feature lifting. Discrete degree values are embedded via a learned embedding (lookup table) $e(d_v) \in \mathbb{R}^{\mathcal{E}_d}$. The discrete block $[\, b_v,\ e(d_v)\,]$ is concatenated with the continuous sector-angle features h_v and mapped to the hidden dimension Z using a two-layer perceptron ϕ_{lift},

$$\tilde{h}_v^{(1)} = \phi_{\text{lift}}(h_v \,\|\, b_v \,\|\, e(d_v)) \in \mathbb{R}^Z, \tag{19}$$

while remaining E(2)-invariant. Initial coordinates are taken as $y_v^{(1)} = y_v$.

EGNN backbone. The lifted features are processed by a stack of three E(2)-equivariant graph neural network (EGNN) layers [30]. At each layer l, E(2)-invariant node features $\tilde{h}_v^{(l)}$ and E(2)-equivariant coordinates $y_v^{(l)}$ are jointly updated using relative positions encoded via a fixed Fourier encoding with four frequencies. Each layer applies residual connections followed by layer normalization on the invariant features, yielding updated node features $\tilde{h}_v^{(l+1)}$. Coordinate updates $y_v^{(l+1)}$ are not affected by residual connections or normalization. The backbone produces a sequence of node-feature tensors $\{\tilde{H}^{(1)}, \tilde{H}^{(2)}, \tilde{H}^{(3)}\}$, where $\tilde{H}^{(l)} \in \mathbb{R}^{|V_t| \times Z}$ and $|V_t|$ vary with the current graph size. Coordinate updates are equivariant under global planar rigid motions, while node and graph features are invariant to translations and rotations. All global readouts are permutation-invariant with respect to node ordering.

Graph-size-aware jumping knowledge. To adapt representations across varying graph sizes, the extractor employs a graph-size-aware jumping-knowledge mechanism [31]. For each layer l, mean-pooled features $g_{\text{pool}}^{(l)} = \frac{1}{|V_t|} \sum_v \tilde{h}_v^{(l)}$ are concatenated with a log-scaled graph-size signal $\log(1 + |V_t|)$ that automatically bounds the graph size for a well-behaved MLP input. A two-layer MLP produces normalized layer weights $w_{\text{gate}}^{(l)}$, which are used to fuse per-layer node feature tensors into the final node embedding tensor

$$\hat{H} = \sum_{l=1}^{3} w_{\text{gate}}^{(l)}\, \tilde{H}^{(l)} \in \mathbb{R}^{|V_t| \times Z}, \tag{20}$$

where each row $\hat{H}_v$ in $\hat{H}$ corresponds to a node embedding at node $v \in V_t$.

B.1.2 Graph-level Embeddings

Graph-level embeddings $\hat{g} \in \mathbb{R}^Z$ are obtained by summing attentional aggregation [32], implemented via a two-layer gating MLP, and mean pooling over the set of node embeddings $\{\hat{H}_v\}_{v=1}^{|V_t|}$.

B.1.3 Feature Aggregation

As described in Section 3.3, for downstream policy network inputs we concatenate each node embedding $\hat{H}_v$ with the graph-level embedding $\hat{g}$

$$H_v = [\hat{H}_v \| \hat{g}] \in \mathbb{R}^{2Z} \tag{21}$$

to form a final set of fused node-level representations $\{H_v\}_{v=1}^{|V_t|}$. Including graph-level information enables the agent to make globally informed decisions using only node-level features H_v while preserving the Markov property.

For the critic, only the graph-level representation $\hat{g}$ is used as input, and node-level representations are discarded.

B.2 Policy Networks

We describe the neural networks used to model the policy distributions $\pi_\phi(g_t \mid G_t)$ (matching), $\pi_\phi(p_t \mid g_t, G_t)$ (rule selection), and $q_\phi(\xi_t \mid g_t, p_t, G_t)$ (parametrization), as introduced in Section 3.4.

B.2.1 Matching

The matching distribution $\pi_\phi(g_t \mid G_t)$ is approximated by a two-layer MLP with domain $\{\mathbb{R}^{2Z}\}^{|V_t|} \to \mathbb{R}$, taking as input the node representations $\{H_v\}_{v=1}^{|V_t|}$ and producing one logit per node. By definition of admissible matches (cf. Figure 2.1), only boundary nodes are valid; all non-boundary nodes are therefore masked by assigning $-\infty$ logits. The resulting logits define the categorical distribution $\pi_\phi(g_t \mid G_t)$, from which a boundary node $\bar{v}_t^b$ is sampled.

B.2.2 Rule Selection

The rule-selection distribution $\pi_\phi(p_t \mid g_t, G_t)$ is modeled by a two-layer MLP $\mathbb{R}^{2Z} \to \mathbb{R}^{P_{\max}}$ with input $H_{\bar{v}_t^b}$. Here, $P_{\max} = 7$, corresponding to the number of rules implemented (cf. Table 2). Conditioning on (g_t, G_t) is thus realized by evaluating the node embedding at the sampled match location $\bar{v}_t^b$ with $H_{\bar{v}_t^b}$. The network outputs one logit per rule in $\mathcal{P}$, defining the categorical distribution $\pi_\phi(p_t \mid g_t, G_t)$ from which a rule $\bar{p}_t$ is sampled.

B.2.3 Parametrization

The continuous parametrization $q_\phi(\xi_t \mid g_t, p_t, G_t) = \mathcal{N}(\mu_\phi, \sigma_\phi^2)$ is modeled as an isotropic Gaussian. The mean μ_ϕ is conditioned on the sampled rule via a learned rule embedding $e(p_t) \in \mathbb{R}^{\mathcal{E}_p}$ and predicted by a two-layer MLP with domain $\mathbb{R}^{2Z} \times \mathbb{R}^{\mathcal{E}_p} \to \mathbb{R}^{D_{\max}}$, taking inputs $H_{\bar{v}_t^b}$ and $e(p_t)$. Here, $D_{\max} = \max_{p \in \mathcal{P}} \dim \Theta(p) = 7$, corresponding to the maximum number of continuous parameters for all rules. For any rule that requires fewer parameters, we apply masking to filter unused dimensions. The log-variance $\log \sigma_\phi^2$ is a global trainable vector of dimension seven with no state-dependent inputs. This Gaussian represents the nominal parametrization ξ_t with infinite support prior to the action projection.

B.3 Critic Network

The critic predicts a scalar state value from the graph-level embedding $\hat{g}_t \in \mathbb{R}^Z$ and discards node-level representations. Note that the critics feature extractor does *not* share the feature-extractor parameters with the policies feature extractor. Each uses an architecturally identical but independently trained extractor, to disentangle learning signals for actor and critic. To provide an explicit graph-size signal, a learned embedding $e(\log(1 + |V_t|)) \in \mathbb{R}^Z$ is added to form $\tilde{g}_t = \hat{g}_t + e(\log(1 + |V_t|))$. The state value $V(G_t)$ is then predicted from $\tilde{g}_t \in \mathbb{R}^Z$ using a two-layer MLP.

B.4 Implementation, Network Sizes, and Training

The complete actor–critic architecture is implemented using PyTorch and PyTorch-Geometric [54], while the environment is implemented in JAX [53]. The PPO implementation is inspired by, but independently re-implemented from StableBaselines3 [55] to support graph data structures.

Network Sizes. All mentioned actor–critic MLPs use two-layers (`Linear - SiLU - Linear`) (including MLPs within the EGNN layers) with `SiLU` activations. The hidden dimension $Z = 32$ is used throughout, except for the critic MLP, which uses dimension 64. The degree and rule embeddings are set to $E_d = E_p = 20$. In total, the architecture contains $66{,}972$ trainable parameters, illustrating the expressivity of the model with only a few tens of thousands of parameters.

PPO. PPO optimization uses the Adam optimizer [56] with a cosine-annealed learning-rate schedule. Training is performed on an NVIDIA RTX PRO 6000 Blackwell Max-Q GPU, while the environment runs on a 16-core AMD CPU with multiprocessing. Training time ranges from 10 minutes to 2 hours depending on the task complexity. Hyperparameters are listed in Table 3. All policy networks, including feature extractors, are orthogonally initialized with gain $\sqrt{2}$; final linear layers of matching, rule-selection, and parametrization networks use gain 0.01 and zero bias to encourage early exploration. The log-variance vector $\log \sigma_\phi^2$ is initialized to zero ($\sigma = 1$), and critic networks are initialized with gain 1.0.

Table 3: PPO hyper-parameters chosen consistently for all experiments.

Parameter	Value
Discount factor γ	0.99
GAE parameter λ	0.95
Clip range ϵ_{PPO}	0.2
Entropy coefficient c_{ent}	0.01
PPO epochs per update	5
Mini-batch size	675
Number of mini-batches	4
Number of parallel environments	30
Number of steps per rollout	90
Time limit for episode	400
Learning rate	4×10^{-4} (decay to 10^{-5})
Target KL-divergence	0.02

C Action Projection

This appendix describes the learnable action projection module η_ψ used to map nominal continuous action parameters to a state-dependent feasible set, as introduced in Section 3.4 and Section 3.5. We detail (i) the action projection architecture, (ii) the supervised training procedure, and (iii) the generation of supervision signals for action projection training. We additionally show that the policy space is preserved up to a reparameterization using the action projection in Appendix C.1.3

C.1 Action Projection Architecture

At time step t, the actor produces a nominal hybrid action $a_t = (g_t, p_t, \xi_t)$, where g_t denotes the selected boundary node (match), $p_t \in \mathcal{P}$ the selected grammar rule, and ξ_t a nominal continuous parameter vector. The action projection is then conditioned on the local node-level embedding at the selected match and on the discrete rule choice using a learned embedding $e(p_t) \in \mathbb{R}^{\mathcal{E}_p}$. The sampled node-level embedding $\hat{H}_{\bar{v}_t^b} \in \mathbb{R}^Z$ is extracted using the equivalent feature extractor (cf. Appendix B.1 and Section 3.3) as for the actor–critic networks, but trained separately in terms of trainable parameters ψ.

C.1.1 Squashed Affine Map

Conditioned on the concatenation of a sampled node embedding $\hat{H}_{\bar{v}_t^b}$ and $e(\bar{p}_t)$ the projection uses an element-wise affine transformation followed by a $\tanh$ squashing nonlinearity, explicitly formulated as

$$\theta_t = \eta_\psi(\xi_t; G_t, g_t, p_t) = \tanh\Big(s_\psi(H_{\bar{v}_t^b} \| e(\bar{p}_t)) \odot \xi_t + t_\psi(H_{\bar{v}_t^b} \| e(\bar{p}_t))\Big) \tag{22}$$

where $s_\psi(\cdot) > 0$ and $t_\psi(\cdot)$ denote learned scale and shift functions, respectively, and $\odot$ denotes element-wise multiplication.

The scale and shift functions $s_\psi(\cdot) > 0$ and $t_\psi(\cdot)$ are modeled as a 2-layer MLP $\mathbb{R}^{Z+\mathcal{E}_p} \to \mathbb{R}^{D_{\max}}$ with input $[\, H_{\bar{v}_t^b} \,\|\, e(p_t)\,]$ each. Additionally, the output of the scale MLP $s_\psi(\cdot)$ is $\log s_\psi$ bounded via $\log s_\psi = \tanh(\cdot)\,\kappa$ with $\kappa = 2$, yielding element-wise scale values $s_\psi \in [\exp(-2), \exp(2)]$.

The final linear layers of both MLPs are initialized to zero, resulting in an initial near-identity mapping (unit scale and zero shift) up to the outer $\tanh$. This enforces unbiased initial mapping before training the action projection and reflects a standard squashed Gaussian projection. Hidden layer dimensions are set to 32.

C.1.2 Jacobian determinant

Since Equation 22 is element-wise, the Jacobian is diagonal and the log absolute determinant is available in closed form,

$$\log\left|\det J_{\eta_\psi}\right| = \sum_{i=1}^{D_{\max}} \mathrm{mask}_i(p_t)\,\Big(\log s_{\psi,i} + \log\big(1-\theta_{t,i}^2\big)\Big), \tag{23}$$

with numerical stabilization obtained by clamping $\theta_{t,i}$ away from ± 1 with a small ε. $\mathrm{mask}_i(p_t)$ represents masking for unused rule-dependent dimensions. During PPO optimization, this term is used to apply the standard change-of-variables correction (cf. Section 3.4)

$$\log \pi_\phi(\theta_t \mid \cdot) = \log q_\phi(\xi_t \mid \cdot) - \log\left|\det J_{\eta_\psi}\right|. \tag{24}$$

C.1.3 Policy Space Preservation

We show that the policy space Θ is preserved up to a reparameterization using the action projection η_ψ.

Let $\Xi = \mathbb{R}^d$ and $\Theta = (-1,1)^d$. The map $\eta_\psi : \Xi \to \Theta$, as specified in Equation 22, is a C^1 diffeomorphism.

Define the push-forward operator

$$\eta_{\psi\#} : \hat{\mathcal{P}}(\Xi) \to \hat{\mathcal{P}}(\Theta), \qquad \eta_{\psi\#}\mu := \mu \circ \eta_\psi^{-1}, \tag{25}$$

where $\hat{\mathcal{P}}(\cdot)$ denotes the set of probability measures. Then, $\eta_{\psi\#}$ is a bijection with inverse $(\eta_\psi^{-1})_\#$. Consequently, the policy classes

$$\{\ q_\phi \in \hat{\mathcal{P}}(\Xi)\ \} \quad \text{and} \quad \{\ \pi_\phi = \eta_{\psi\#} q_\phi \in \hat{\mathcal{P}}(\Theta)\ \} \tag{26}$$

are isomorphic via the push-forward induced by the diffeomorphism η_ψ, and the policy space is preserved up to reparameterization.

C.2 Supervised Training Procedure

The action projection is trained using a short FIFO replay buffer that stores recent graph transitions collected during PPO rollouts. Since the action projection is off-policy in parameters ψ, we store the last ten rollouts in the replay buffer and train it synchronously alongside PPO. Each entry in the buffer consists of graph states together with nominal continuous parameters ξ_t and the corresponding environment-projected targets $\theta_t^\star$ using Covariance Matrix Adapted Evolutionary Strategy (CMA-ES) (see Appendix C.2.1).

Training minimizes a mean-squared error loss over the continuous parameters,

$$\mathcal{L}_{\mathrm{proj}} = \mathbb{E}\Big[w_t\ \big\|\mathrm{mask}(p_t) \odot (\eta_\psi(\xi_t) - \theta_t^\star)\big\|_2^2\Big], \tag{27}$$

where the sample weight w_t is set to 1 if the environment modified any continuous parameter, and to 0.5 otherwise. This biases learning toward the cases where geometric correction is required.

The projection is optimized using the same optimizer and learning-rate schedule as PPO. The projection is then trained for a fixed number of 7 supervised epochs per PPO iteration. After a prescribed number of 200 PPO iterations, the projection training is terminated, and the projection remains fixed during subsequent policy optimization.

C.2.1 Environment-Side Supervision via CMA-ES

For each step, the actor produces nominal samples ξ_t and projects them using the action projection η_ψ to θ_t. The environment then assesses the feasibility of the proposed continuous parameters θ_t using geometric $\mathcal{C}_G$ and topological $\mathcal{C}_T$ constraints checks (cf. Section 3.2). If an action is topologically admissible but geometrically infeasible (e.g., due to edge intersections or boundary length violations), the CMA-ES procedure is invoked (cf. Section 3.5) to further correct the proposed θ_t. If CMA-ES

finds an admissible solution $\theta_t^\star$, the action projection will be updated and trained using the corrected solution in the next training iteration.

CMA-ES is initialized at the proposed parameters and minimizes an objective (cf. Equation 28) that encourages proximity to the original parameters while satisfying $\mathcal{C}_G$. If a feasible $\theta_t^\star$ is found, it is applied to advance the environment state, otherwise we fall back to the rejection-sampling scheme while advancing the time step $t \to t+1$ without changing the environment state. Advancing the time step informs the actor that the action was infeasible implicitly, due to a reduced terminal reward according to the discount factor $\gamma < 1$.

The optimization problem stated in Equation 13 is implemented using an explicit penalty objective as

$$J(\theta_t') = \lambda_{\text{geom}}\big(c_{\text{pl}}(\theta_t') + c_\ell(\theta_t')\big) + \|\theta_t' - \bar{\theta}_t\|_2, \tag{28}$$

with penalty weight $\lambda_{\text{geom}} = 10$, c_{pl} representing the planarity constraint in the form of edge-intersections and c_ℓ the boundary edge length constraint. Other constraints in $\mathcal{C}_G$ do not appear explicitly, as they are either enforced by the rule parametrization (sector angles, internal edge lengths), or correspond to violations introduced by merging rules, which are classified by topological constraints $\mathcal{C}_T$ and cannot be resolved by continuous reparameterization. Since CMA-ES operates in an unconstrained space, optimization is performed over $z \in \mathbb{R}^{\dim\Theta(p_t)}$ using the sigmoid reparameterization $\theta_t' = \sigma(z) \in (0,1)^{\dim\Theta(p_t)}$ to operate in the rule parameterization space (cf. Appendix A.3), yielding

$$\min_{z\in\mathbb{R}^{\dim\Theta(p_t)}} J\big(\sigma(z)\big). \tag{29}$$

CMA-ES is implemented using EVOSAX [57] and run using a population size 100 with 4 iterations in total, an initial search standard deviation of 1.

C.3 CMA-ES & Action Projection Results

We further analyze the effect of CMA-ES and action projection on rollout wall-clock time in Figure 7(a,b), and report the number of CMA-ES invocations during training across graph sizes and constraint regimes in Figure 7(c). Figures 7(a) shows that, after 50% of training, the computational overhead introduced by CMA-ES is negligible relative to the remaining rollout computations, while (b) shows the fractional proportion of CMA-ES during the whole training. This low overhead is due in part to the action projection mechanism, which progressively approximates the safe set during training, and renders CMA-ES only as a fallback mechanism, and to the efficient EVOSAX implementation. Consistently, Figure 7(c) shows that CMA-ES invocations rapidly decrease over training and become rare, partly due to the learned action projection. This effect is further supported by Figure 7(e), where we compare geometric constraint violations with action projection enabled against a static projection variant obtained by setting $s_\psi = 1$ and $t_\psi = 0$. Figure 7(d) reports the CMA-ES success rate and shows that, even for large graphs under tight constraints, the optimizer finds feasible parameters $\theta_t^\star$ satisfying $\mathcal{C}_G = 0$ at a high rate. Further, as discussed in Section 3.5, topological violations are substantially less frequent than geometric violations, as shown on the logarithmic scale in Figure 7(f). Finally, Figure 7(g) and (h) shows the overall rejection sampling rate, which comprises both topological and geometric constraint violations after action projection, which then lead to rejection sampling. For smaller graph sizes this rate quickly converges to zero, while larger graphs with tighter constraints more often reside to rejection sampling. Nevertheless, this rate still remains low.

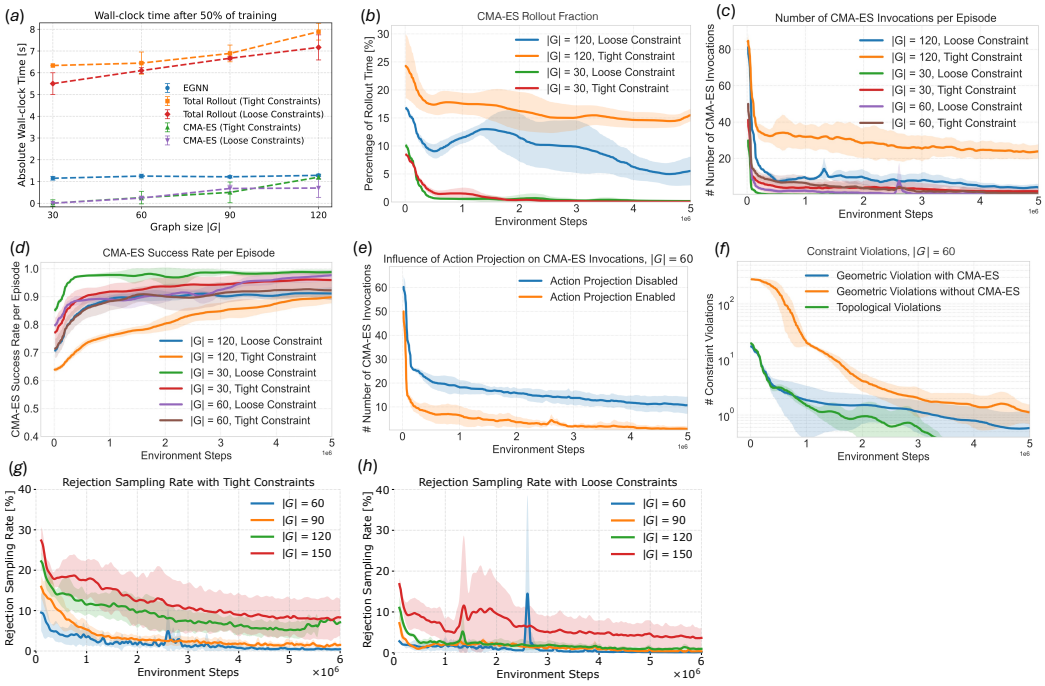


Figure 7: Impact of CMA-ES optimization with action projection on training performance. Constraint tightness refers to the same constraint set as described in Section 5. All error-bars refer to 1-sigma standard deviations accross 5 seeds. (a) Wall-clock time per rollout at 50% of training for different graph sizes, with contributions from feature extraction (EGNN) and CMA-ES under loose and tight constraints. (b) Percentage of rollout time spent in CMA-ES during training. (c) Number of CMA-ES invocations per episode for different graph sizes and constraint tightness. (d) Success rate of CMA-ES in resolving geometric constraints. (e) CMA-ES invocations with learned vs. static projection ($s_\psi = 1, t_\psi = 0$). (f) Number of topological (C_T) and geometric (C_G) constraint violations during PPO (log scale). (g, h) Rejection sampling rate during training using tight and loose constraints.

D Graph Metrics

We describe the graph metrics used in the experiments in Section 5. Metrics are categorized as topological (adjacency-based), geometry-based (drawing-dependent) or spectral (Laplacian-based). All metrics are summarized in Table 4.

D.1 Topological and Spectral Metrics

The topological and spectral metrics described here are enumerated and discussed with more detail in [36, 58]

For all metrics, we represent the union of internal and boundary edges $E_t^d \cup E_t^b$ as an undirected graph with adjacency matrix $A \in \{0,1\}^{n \times n}$.

Degree inequality (Gini coefficient). Given node degrees sorted as $d_1 \leq \cdots \leq d_n$ for n nodes in V_t with mean $\bar{d}$, the definition of the gini coefficient is

$$c_{\text{gini}} = \frac{1}{2n^2\bar{d}} \sum_{i=1}^{n} \sum_{j=1}^{n} |d_i - d_j|. \tag{30}$$

In applications, a c_{gini} helps quantify hub dominance, load balancing concerns, and whether a model produces unrealistically concentrated connectivity.

Triangle count. The number of triangles is computed as

$$T = \frac{\mathrm{tr}(A^3)}{6}. \tag{31}$$

The triangle count is a key indicator of local closure and community structure. Generators often match triangle statistics less accurately than degree distributions, exposing mode collapse.

Global clustering coefficient. The total number of connected triplets is

$$\text{Triplets} = \sum_{i=1}^{n} \binom{d_i}{2} = \sum_{i=1}^{n} \frac{d_i(d_i-1)}{2}. \tag{32}$$

The clustering coefficient is then

$$c_{\text{cluster}} = \frac{3T}{\text{Triplets}}. \tag{33}$$

This captures how frequently a node's neighbors are themselves connected. It differentiates real-world networks from random baselines and is critical for evaluating realism in graph generation.

Spectral gap (or algebraic connectivity). Let $L = I - D^{-1/2}AD^{-1/2}$ be the normalized Laplacian. Then, the spectral gap is given by the second eigenvalue of L

$$\lambda_2(L) \in [0, 2]. \tag{34}$$

Higher values indicate strong connectivity, robustness, and fast mixing. This metric is often maximized in resilient network design.

Table 4: Graph- and drawing-level metrics used for evaluation.

Metric	**Input**	**Typical Use**
Gini coefficient c_{gini}	adjacency	hub dominance / inequality
Triangles $T = \mathrm{tr}(A^3)/6$	adjacency	closure / community signal
Clustering $c_{\text{cluster}} = 3T/\text{Triplets}$	adjacency	local redundancy
Algebraic connectivity $\lambda_2(L)$	adjacency	connectivity strength
Angular resolution AR	drawing angles	readability at vertices
Edge uniformity score S_ℓ	drawing lengths	regularity / aesthetics
Circularity r_{iso}	boundary polygon	roundness of layout
Squareness r_{square}	boundary polygon	alignment with square shape

D.2 Geometric Metrics

The geometric metrics assume a planar drawing with node positions $y_i \in \mathbb{R}^2$. The described planar graph metrics are based on [59, 60, 61]

Angular resolution. This metric is central in graph drawing aesthetics, where evenly spaced edge orientations improve visual clarity. Angular resolution AR also correlates with mesh quality in finite-element analysis. For any node i in V_t with degree $d_i \geq 2$, let $\alpha_{\min}(i)$ be the minimum angle between incident edges. First, the ideal angle is defined as

$$\alpha_{\text{ideal}}(i) = \frac{2\pi}{d_i}. \tag{35}$$

Then, the per-node angular resolution score is

$$\mathrm{AR}(i) = 1 - \frac{|\alpha_{\text{ideal}}(i) - \alpha_{\min}(i)|}{\alpha_{\text{ideal}}(i)}. \tag{36}$$

The global angular resolution is

$$\mathrm{AR} = \frac{1}{|\{i : d_i \geq 2\}|} \sum_{i:d_i \geq 2} \mathrm{AR}(i). \tag{37}$$

Edge-length uniformity. Let ℓ_e be the Euclidean length of edge e and $\bar{\ell}$ the mean edge length. We define

$$\Delta_\ell = \frac{1}{m} \sum_{e \in E} \frac{|\ell_e - \bar{\ell}|}{\bar{\ell}}, \qquad S_\ell = \frac{1}{1 + \Delta_\ell}, \tag{38}$$

with S_ℓ representing the edge-length score metric, where a higher score indicates better edge-length uniformity. Uniform edge lengths enhance readability and fairness in graph layouts. This metric supports assessments of isotropic meshing or layout aesthetics, and is often jointly optimized with angular resolution.

Circularity. Recalling that $V_t^b \subseteq V_t$ denote boundary nodes, let A_b be the area enclosed by the polygon formed by the boundary nodes, and P_b its perimeter. The circularity score is defined as the isoperimetric ratio

$$r_{\text{iso}} = \frac{4\pi A_b}{P_b^2}. \tag{39}$$

This value is maximized (equal to 1) for a perfect circle of the graph boundary hull and decreases with elongation or irregularity of the boundary shape. For all baseline methods, the notion of boundary nodes does not exist, hence we use alpha shapes [62] to estimate the concave hull of baseline graphs. The value of α is estimated using bisection, with a maximum value $\alpha = 5$ and a maximum of 10 bisection steps. The bisection selects the largest α for which all nodes are still included in the concave hull and the hull forms a single connected polygonal domain.

Squareness. To compute squareness of a graphs' hull, we consider all possible planar rotations of the boundary region and identify the orientation that minimizes the area of the axis-aligned bounding square. Let A_{box} be the area of this minimum-area enclosing square. The squareness coefficient is

$$r_{\text{square}} = \frac{A_b}{A_{\text{box}}}. \tag{40}$$

This value lies in the interval $[0, 1]$, with higher values indicating that the boundary closely conforms to a square, up to rotation. This metric is rotationally invariant and captures alignment with square layouts.

E Graph Generators

We evaluate graph algorithms using synthetic datasets generated from well-established random graph models, focusing on classical, planar, and diffusion-based generators. Additionally, we include the House of Graphs (HOG) [48] dataset in the benchmarks, which includes, to the best of our knowledge, the most comprehensive connected planar graph dataset, albeit drawn without an embedding. Across all generators, benchmark-favorable settings and method-specific adaptations described throughout this chapter are applied to align each approach with the benchmark objectives. While computational budgets are kept as comparable as possible, diffusion-based models still require substantially higher training time. For all methods that do not provide an embedding by default, we use the force-directed spring layout to pre-process the graph to be embedded [49] using the NETWORKX library [63]. This library is also used in the following described graph generators.

E.1 Classical Random Graph Models

Erdős–Rényi (ER) Graphs. We use the $G(n, p)$ model, where each edge between the n nodes is independently included with probability $p = \frac{2m}{n(n-1)}$. This model serves as a uniform random baseline and produces graphs with a binomial degree distribution, implemented using `networkx.erdos_renyi_graph` [11].

For benchmarking, we have generated 10,000 graphs per configuration, with edge probabilities ranging from 0.04 to 0.10. Connectivity is enforced, and disconnected graphs are filtered out. Higher edge densities substantially reduce the proportion of planar graphs, reflecting the well-known trade-off between connectivity and planarity in sparse regimes. Specifically, this method does not produce any planar graph for the second and third benchmark, see Appendix G. Despite extensive exploration of this parameter space, Erdős–Rényi graphs never achieve the best performance in our evaluations. This behavior is expected because the model is constraint-free and not goal-directed.

Exponential Random Graph Models (ERGM). Graphs are sampled from an exponential distribution over structural features using a Metropolis–Hastings MCMC sampler:

$$P(G) \propto \exp\left(\beta_{edge} \cdot \#\text{edges} + \beta_{triangle} \cdot \#\text{triangles} + \beta_{star} \cdot \#\text{2-stars}\right). \tag{41}$$

We use the implementation from [64] performing 1000 MCMC steps after burn-in. Parameters are selected from $\beta_{\text{edge}} \in \{-2.5, -2.0, -1.5\}$ and $\beta_{\text{triangle}} \in \{0.0, 0.1\}$ with fixed $\beta_{\text{star}} = 0.0$. Compared to other models, ERGMs involve a large hyperparameter space and are more difficult to tune, as certain parameter settings lead to disconnected graphs, while others produce dense, non-planar structures. As ERGMs do not enforce planarity, many sampled graphs violate planarity.

E.2 Planar Graph Generators

Plantri. We generate triangulated planar graphs using the PLANTRI software [14], which guarantees planarity and non-redundancy via combinatorial generation and isomorphism rejection. Due to expensive I/O and limited scaling beyond 15–20 nodes, we evaluate Plantri only on the first 2,000 sampled graphs.

Boltzmann Sampler. Uniform planar graphs are sampled using a Boltzmann sampler based on analytic combinatorics [15], implemented via the `planar_graph_sampler` library [65]. This approach generates unbiased planar graphs with expected quadratic time complexity. Due to the relatively high computational cost of this approach, we evaluate 50 graphs per benchmark, as larger sample sizes are a lot more expensive compared to our method.

Delaunay Triangulations. A set of $|G| - b > 0$ points are sampled uniformly from circular and square domains, while an additional b boundary points placed along the domain boundary *equidistantly*. The resulting point set is triangulated according to the Delaunay criterion using `scipy.spatial.Delaunay`. For circular target metrics, boundary points are sampled along a circle to enforce a larger isoperimetric ratio, while an analogous construction is used for the squareness metric.

Lattice Graphs. Planar grid graphs are generated as regular 2D lattices using `networkx.grid_2d_graph`. For a target graph size, we select an approximately square grid with 4-neighbor connectivity; in the circular setting, rows are wrapped into rings. These graphs provide deterministic spatial baselines with highly regular structure and no triangles, although the final node count may differ slightly from the target size.

E.3 Deep Generative Graph Models

We further include representative modern Deep Generative Models (DGM) for graph generation as baselines, including DiGress, GDSS and MuDiff [38, 39, 43]. These models are distribution-learning approaches that generate graphs by sampling from learned probabilistic processes. While DiGress and MuDiff provide a conditional sampling mechanisms towards target properties, they do not enforce hard constraints during generation. The same applies to GDSS, which is a purely unconditional model, but can be enhanced with PRODIGY [40] guidance towards single-objective target properties. Due to this, all models are evaluated by selecting candidates according to the procedure described in Section F to adhere to as many constraints as possible for a fair comparison.

DiGress. DiGress is a discrete diffusion model for categorical graph generation with optional conditional guidance [38]. The model iteratively denoises graph structures in discrete space and enables conditioning through learned property predictors. In our evaluation, DiGress is first trained once in an unconditional manner on the full HOG dataset to learn a general distribution over connected graphs. Subsequently, for each benchmark, we train a dedicated property regressor that estimates the respective benchmark-specific objective (e.g., number of triangles and isoperimetric ratio in Benchmark G.4) which is trained using a per-metric standardized mean-squared error loss, so that metrics on different scales contribute comparably to the objective. This regressor is then used within the conditional generation mechanism of DiGress to guide sampling toward graphs that optimize the target objective.

All architectural settings follow the original DiGress implementation. Minor adjustments to the number of diffusion steps and training epochs were applied to ensure stable convergence on the HOG dataset. More detailed hyperparameters are shown in Table 5.

Table 6: Hyperparameters for GDSS with PRODIGY guidance.

Hyperparameter	**Value**
Training epochs	2000
Batch size	64
Learning rate	10^{-2}
EMA decay	0.999
Diffusion steps	1000
Predictor / Corrector	Euler / Langevin

We also note that the original DiGress planar benchmark was evaluated on the SPECTRE planar dataset, consisting of 200 graphs of fixed size (64 nodes) generated exclusively as Delaunay triangulations. This dataset exhibits limited structural diversity compared to HOG, which includes 4000 graphs of variable size (up to 250 nodes) and unrestricted topology. Under this more challenging regime, only a subset of unconditionally generated samples satisfies the planarity constraint (33%), while the original work of Vignac et al. [38] report 75% validity of planar graphs in the simpler dataset regime used.

Table 5: Hyperparameters for DiGress with property-regressor guidance.

Hyperparameter	**Value**
Unconditional epochs	1000
Unconditional batch size	8
Unconditional learning rate	$2 \cdot 10^{-4}$
Diffusion steps	500
Regressor hidden dimension	128
Regressor dropout	0.1
Regressor epochs	200
Regressor batch size	32
Regressor learning rate	10^{-3}
Regressor weight decay	10^{-5}
Regressor scheduler	CosineAnnealingLR
Inference Guidance scale	2.0

GDSS with PRODIGY Guidance. GDSS is a score-based generative model that learns continuous diffusion dynamics over adjacency matrices and node features [39]. In our evaluation, an unconditional GDSS model is first trained on the HOG dataset to learn a general distribution over graph structures. Because GDSS does not natively support conditional generation, we apply guidance using the PRODIGY framework [40], which steers sampling toward specified single-objective target properties. We select the target node number as the conditioning signal during graph generation with Prodigy. More detailed hyperparameters are shown in Table 6 to enhance reproducibility.

MuDiff. MuDiff is a diffusion-based model originally developed for joint structure–geometry generation in molecular graphs [43]. Although designed for three-dimensional molecular systems, the architecture can be adapted to general graph generation tasks. For compatibility with our planar graph setting, we preprocess HOG graphs using force-directed spring layout and embed them in three-dimensional space by assigning zero values to the third coordinate. Benchmark-specific conditional models are trained following the original architectural specifications. Accordingly, four conditional models are trained independently, one for each benchmark scenario. Hyperparameters are tuned separately for each benchmark, in Table 7 we report the ranges. We further note that, unlike the other deep generative methods considered in this study, MuDiff directly generates graphs together with their geometric embeddings, rather than requiring a separate layout or embedding procedure, making it a relevant comparison to state of the art conditional graph generation methods in unconstrained settings.

Table 7: Hyperparameters for MuDiff.

Hyperparameter	Value
Diffusion steps	500
Noise schedule	`polynomial_2`
Diffusion loss	`l2`
Learning rate	$[5 \cdot 10^{-4}, 5 \cdot 10^{-5}]$
Batch size	[2,4]
Training epochs	[400,600]
EMA decay	0.999
Encoder layers	6
Embedding dimension	128
Edge embedding dimension	128
Graph embedding dimension	32
Attention heads	8

F Ranking-Based Graph Selection for Baseline Evaluations

We consider a set of candidate graphs $\tilde{G}_i$ produced by a given method. Candidates are first filtered for planarity (this can still contain graphs with edge intersections). Only planar graphs are retained for further processing through the benchmark pipeline, resulting in the set of graphs G_i.

For each retained graph, we compute benchmark metrics $\mathcal{M}(G_i)$ and the final reward r_i(see Section 3.6). The additional quantities used for ranking are split according to the generator class and are described separately for classical and diffusion-based methods below.

Classical Generators For classical generators, we randomly sample candidates according to the generator specifications and report metrics, and constraint satisfaction. For each generator, we retain only the graphs that satisfy most constraints. If multiple graphs satisfy equally many constraints, we report the graph with highest reward among them. As visible in the benchmarks (cf. Appendix G), classical generators rarely satisfy all constraints.

Deep Generative Models For deep generative models, we sample approximately $M \approx 1000$ candidate graphs per generator. Since generated samples may contain small disconnected components, we retain only the largest connected component G_i with size $|G_i|$ for subsequent evaluation, ensuring that otherwise valid planar structures are not discarded. Graph-size proximity is prioritized to ensure comparability with the target benchmark configuration: since DGMs do not explicitly control the number of nodes during generation, samples can vary substantially in size, and differences in graph size can strongly affect both evaluation metrics and reward. We therefore first rank candidates by their distance to the target size. Let $|G^\star|$ denote the target size and define $d_i = \big|\, |G_i| - |G^\star| \,\big|$ and

$$b_i = \begin{cases} 0, & d_i \leq 10, \\ 1 + \left\lfloor \frac{d_i - 10}{10} \right\rfloor, & \text{otherwise.} \end{cases} \tag{42}$$

The smaller b_i is for a graph, the higher the graph is ranked. We further compute the force-directed spring layout embedding (omitted for MuDiff) and evaluate the constraints satisfaction. Constraint statistics are recorded but not used for ranking, as prioritizing them was empirically found to strongly disadvantage these models compared to other generators.

The final selected graph is the highest-scoring candidate among those with the closest graph-size match to the target benchmark configuration, and is reported together with its constraint statistics.

G Extended Results on Benchmarks

This section reports the full benchmark results (Tables 8–11) and complements them with additional qualitative comparisons (Figures 8–11).

Some benchmark families, such as Delaunay triangulations and lattice grids, provide explicit planar embeddings by construction. For generators without intrinsic geometric embeddings, including the HOG dataset, node coordinates for visualization and geometric evaluation are computed using a force-directed spring layout [49]. Each table presents the overall reward (cf. Equation 14), which quantifies the multi-objective optimization score, together with the individual metrics and constraint satisfaction for all baselines, including both conventional algorithms and deep generative models. The constraint ℓ_1 denotes whether all edge lengths in a graph satisfy $\ell \in [0.75, 2.5]$, where for ℓ_2 the interval is $[0.1, 3.5]$. We additionally include 1-sigma standard deviations for our method in all tables, as well as constraint satisfaction for each baseline. We further denote with *Emb*., whether a baseline produces an embedding by default, or if the graph is post-processed using the force-directed spring layout. Table 12 additionally reports the per-metric relative deviations across all four benchmarks, marking each metrics' contribution to the aggregate reward explicit (Equation 14).

G.1 Benchmark 1: Power Grid in Circular Hull

Table 8: Extended results for power grid in circular hull with 50 nodes. Note that although MuDiff achieves highest spectral gap, this is achieved by not respecting planarity constraints and target graph size. Generally, for smaller graph the spectral gap tends to be higher, as well as for non-planar graphs.

	Reward	Geom.	Topological		Spec.	Constraints					
Method	r	r_{iso}	c_{gini}	c_{clust}	$\lambda_2(L)$	$\lvert G\rvert$	$\alpha_{\min}$	c_ℓ	$\deg_{\max}$	c_{pl}	Emb.
Target	1.0	1.0	0.1	0.2	1.0	50	20°	ℓ_1	8	True	True
Ours	**0.756** $\pm$0.004	**0.960** $\pm$0.020	**0.090** $\pm$0.002	**0.200** $\pm$0.004	0.050 $\pm$0.003	**50**	✓	✓	✓	✓	✓
HOG	0.382	0.841	0.031	0.000	0.075	47	✓	✓	✓	✗	✗
Delaunay	0.698	0.932	0.182	0.217	**0.087**	**50**	✗	✗	✓	✓	✓
Plantri	0.445	0.377	0.034	0.417	0.039	**50**	✗	✓	✓	✗	✗
Boltzmann	0.545	0.468	0.279	0.248	0.064	52	✗	✗	✗	✗	✗
Erdős–Rényi	0.467	0.697	0.195	0.043	0.006	**50**	✓	✓	✓	✗	✗
ERGM	0.526	0.669	0.272	0.114	0.020	42	✓	✓	✓	✗	✗
Lattice	0.431	0.932	0.051	0.000	0.043	56	✓	✗	✓	✓	✓
DiGress	0.671	0.458	0.181	0.456	0.579	8	✗	✓	✓	✗	✗
GDSS-Prodigy	0.557	0.495	0.2	0.16	0.006	43	✓	✗	✓	✗	✗
MuDiff	0.520	0.140	0.496	0.516	0.889	19	✗	✗	✓	✗	✓

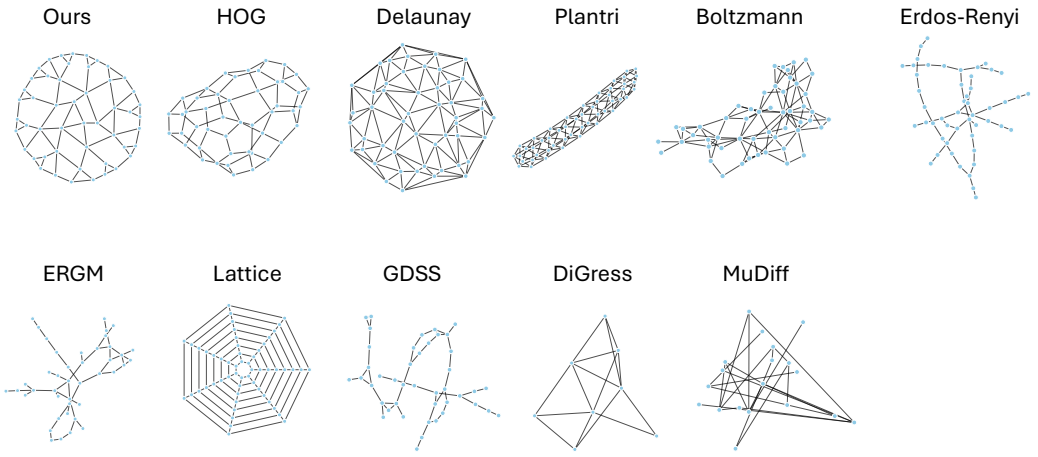


Figure 8: Results for the power grid case for our method and all baselines.

G.2 Benchmark 2: Urban Road Grid in Circular Hull

Table 9: Extended result for urban road grid in circular hull with 80 nodes. Note that although some baselines marginally improve individual metrics, the combined multi-objective reward is highest for our method that balances the individual metrics closest to the target on average. Additionally, the full constraint set is not respected by most baselines, e.g. ERGM beats the angular resolution metric, but does not satisfy target graph size.

	Reward	Geometric			Topo.	Constraints					
Method	r	r_{iso}	S_l	AR	c_{gini}	$\lvert G\rvert$	$\alpha_{\min}$	c_ℓ	$\deg_{\max}$	c_{pl}	Emb.
Target	1.0	1.0	0.7	0.8	0.2	80	20°	ℓ_1	8	True	True
Ours	**0.942** ±0.011	**0.952** ±0.027	0.82 ±0.02	**0.755** ±0.019	0.163 ±0.01	**80**	✓	✓	✓	✓	✓
HOG	0.782	0.403	0.832	0.486	0.158	**80**	✗	✗	✓	✗	✗
Delaunay	0.914	**0.952**	**0.758**	0.445	**0.200**	**80**	✗	✗	✓	✓	✓
Plantri	–	–	–	–	–	–	–	–	–	–	✗
Boltzmann	0.798	0.649	0.772	0.364	0.282	81	✗	✗	✗	✗	✗
Erdős–Rényi	–	–	–	–	–	–	–	–	–	–	✗
ERGM	0.874	0.574	0.797	**0.816**	0.276	45	✓	✓	✓	✓	✗
Lattice	0.799	0.948	0.578	0.750	0.043	**80**	✓	✗	✓	✓	✓
DiGress	0.751	0.479	0.823	0.336	0.274	8	✗	✓	✓	✗	✗
GDSS-Prodigy	0.834	0.291	0.802	**0.790**	0.185	76	✗	✗	✓	✗	✗
MuDiff	0.534	0.338	0.006	0.462	0.254	56	✗	✗	✓	✗	✓

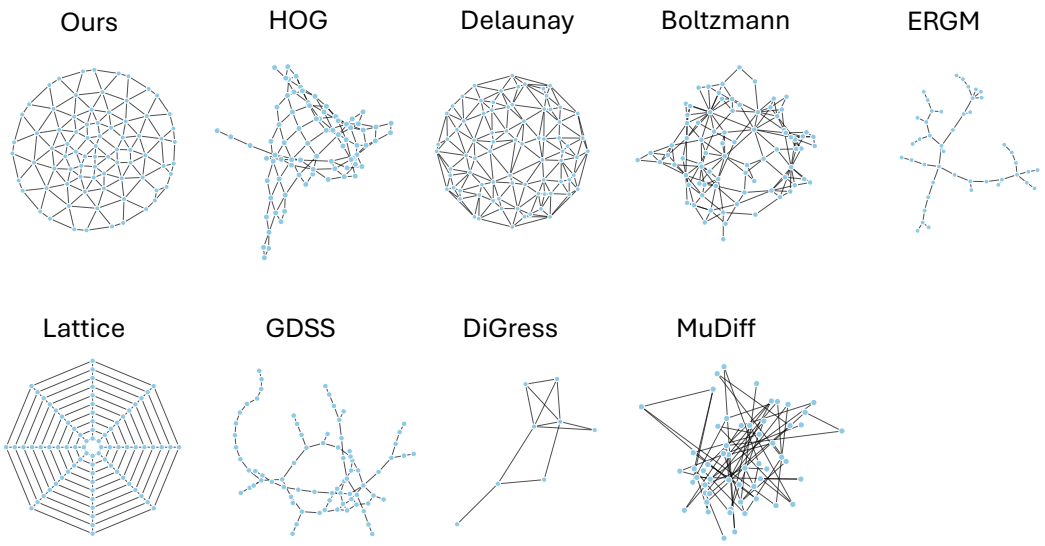


Figure 9: Results for the urban road grid case for our method and all baselines.

G.3 Benchmark 3: Strongly Connected Mesh in Square Hull

Table 10: Extended results for highly connected mesh in square hull with 100 nodes. This benchmark relaxes the constraint set compared to the other benchmarks, providing insights to graph generation closer to an unconstrained setting. Note that in this benchmark, MuDiff achieves highest spectral gap, but does so by strongly dissatisfying planarity constraints.

	Reward	Geometric	Spectral	Constraints					
Method	r	r_{square}	$\lambda_2(L)$	$\lvert G\rvert$	$\alpha_{\min}$	c_ℓ	$\deg_{\max}$	c_{pl}	Emb.
Target	1.0	1.0	0.300	100	1°	ℓ_2	15	True	True
Ours	**0.647** $\pm$0.01	0.887 $\pm$0.06	**0.064** $\pm$0.006	**100**	✓	✓	✓	✓	✓
HOG	0.360	0.543	0.002	101	✓	✓	✓	✗	✗
Delaunay	0.578	0.863	0.039	**100**	✗	✓	✓	✓	✓
Plantri	–	–	–	–	–	–	–	–	✗
Boltzmann	0.502	0.627	0.040	**100**	✗	✓	✓	✗	✗
Erdős–Rényi	–	–	–	–	–	–	–	–	✗
ERGM	0.487	0.486	0.057	17	✓	✓	✓	✓	✗
Lattice	0.587	**1.000**	0.029	**100**	✓	✓	✓	✓	✓
DiGress	0.699	0.385	0.216	9	✗	✓	✓	✗	✗
GDSS-Prodigy	0.386	0.545	0.01	72	✓	✓	✓	✗	✗
MuDiff	0.410	0.197	0.918	85	✗	✗	✓	✗	✓

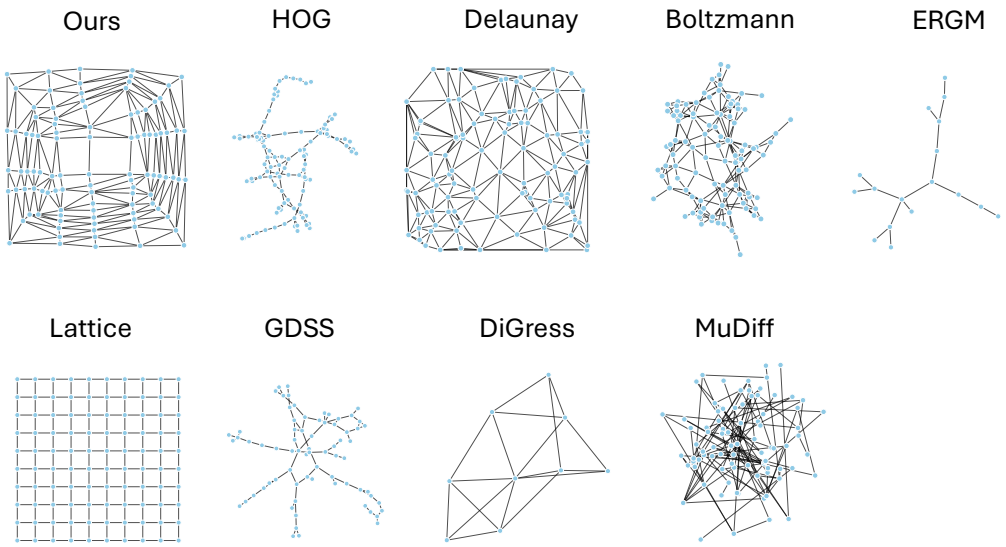


Figure 10: Results for the mesh in square hull high spectral gap for our method and all baselines.

G.4 Benchmark 4: Graph with 40 triangles in circular hull

Table 11: Extended results for graph in circular hull with 60 nodes and 40 triangles.

	Reward	Geom.	Topological	Constraints					
Method	r	r_{iso}	T	$\lvert G\rvert$	$\alpha_{\min}$	c_ℓ	$\deg_{\max}$	c_{pl}	Emb.
Target	1.0	1.0	40	60	20°	ℓ_1	8	True	True
Ours	**0.979** $\pm$0.022	0.920 $\pm$0.080	**39.91** $\pm$0.09	**60**	✓	✓	✓	✓	✓
HOG	0.856	0.892	64	**60**	✓	✓	✓	✓	✗
Delaunay	0.759	**0.949**	107	**60**	✗	✗	✓	✓	✓
Plantri	0.494	0.312	116	**60**	✗	✓	✓	✗	✗
Boltzmann	0.875	0.633	38	56	✗	✗	✗	✗	✗
Erdős–Rényi	0.415	0.709	0	**60**	✗	✓	✓	✗	✗
ERGM	0.333	0.500	0	29	✓	✓	✓	✗	✗
Lattice	0.482	0.932	0	63	✓	✗	✓	✗	✓
DiGress	0.569	0.422	15	10	✗	✓	✓	✗	✗
GDSS-Prodigy	0.552	0.473	12	50	✗	✗	✗	✗	✗
MuDiff	0.433	0.173	16	42	✗	✗	✗	✗	✓

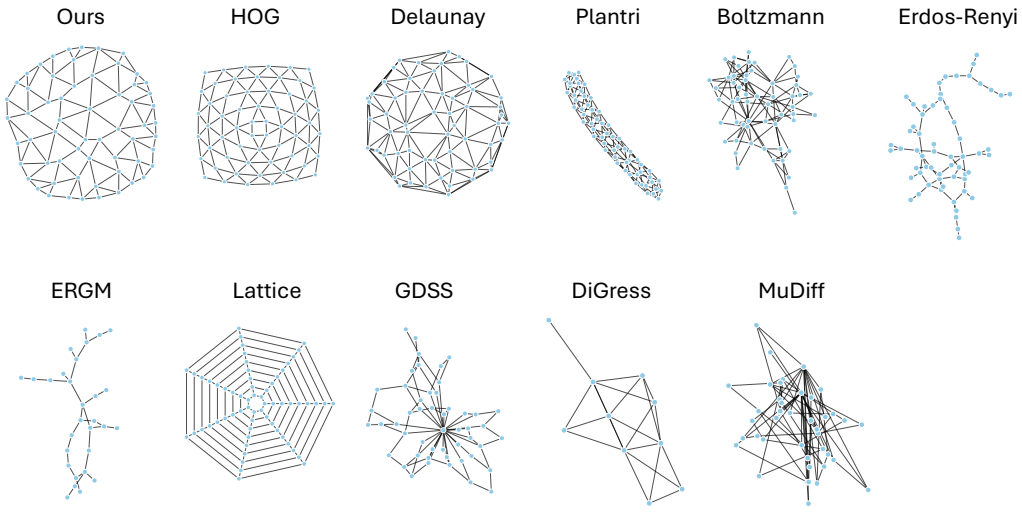


Figure 11: Results for the graph in a circular hull with 40 triangles case for our method and all baselines.

Table 12: Per-metric relative deviations d_m reconstructed from the rounded metrics in Tables 8–11 in addition to the absolute reward. Lower is better for the deviation metrics; each metric contributes $d_m/|\mathcal{M}|$ to the reward.

Benchmark 1: Resilient Power Grid

Method	Reward	d_{iso}	d_{gini}	d_{clust}	d_{λ_2}
Ours	**0.756** $\pm$0.004	**0.020** $\pm$0.010	**0.053** $\pm$0.011	**0.000** $\pm$0.010	0.905 $\pm$0.005
HOG	0.382	0.0864	0.5267	1.0000	0.8605
Delaunay	0.698	0.0352	0.2908	0.0408	**0.8399**
Plantri	0.445	0.4524	0.4925	0.3517	0.9249
Boltzmann	0.545	0.3624	0.4723	0.1071	0.8797
Erdős–Rényi	0.467	0.1786	0.3220	0.6461	0.9881
ERGM	0.526	0.1983	0.4624	0.2739	0.9608
Lattice	0.431	0.0352	0.3245	1.0000	0.9175
DiGress	0.671	0.3717	0.2883	0.3902	0.2666
GDSS-Prodigy	0.557	0.3378	0.3333	0.1111	0.9881
MuDiff	0.520	0.7544	0.6644	0.4413	0.0588

Benchmark 2: Urban Road Grid

Method	Reward	d_{iso}	d_{S_l}	d_{AR}	d_{gini}
Ours	**0.942** $\pm$0.011	**0.025** $\pm$0.014	0.079 $\pm$0.012	**0.029** $\pm$0.013	0.102 $\pm$0.030
HOG	0.782	0.4255	0.0862	0.2442	0.1173
Delaunay	0.914	**0.0246**	**0.0398**	0.2851	**0.0000**
Plantri	—	—	—	—	—
Boltzmann	0.798	0.2129	0.0489	0.3746	0.1701
Erdős–Rényi	—	—	—	—	—
ERGM	0.874	0.2706	0.0648	**0.0099**	0.1597
Lattice	0.799	0.0267	0.0955	0.0323	0.6461
DiGress	0.751	0.3523	0.0808	0.4085	0.1561
GDSS-Prodigy	0.834	0.5492	0.0679	**0.0063**	0.0390
MuDiff	0.534	0.4948	0.9830	0.2678	0.1189

Benchmark 3: Strongly Connected Mesh

Method	Reward	d_{square}	d_{λ_2}
Ours	**0.647** $\pm$0.010	0.060 $\pm$0.034	**0.648** $\pm$0.027
HOG	0.360	0.2962	0.9868
Delaunay	0.578	0.0735	0.7699
Plantri	—	—	—
Boltzmann	0.502	0.2293	0.7647
Erdős–Rényi	—	—	—
ERGM	0.487	0.3459	0.6807
Lattice	0.587	**0.0000**	0.8237
DiGress	0.699	0.4440	0.1628
GDSS-Prodigy	0.386	0.2945	0.9355
MuDiff	0.410	0.6708	0.5074

Benchmark 4: Graph with 40 Triangles

Method	Reward	d_{iso}	d_T
Ours	**0.979** $\pm$0.022	0.042 $\pm$0.043	**0.0011** $\pm$0.0011
HOG	0.856	0.0571	0.2308
Delaunay	0.759	**0.0262**	0.4558
Plantri	0.494	0.5244	0.4872
Boltzmann	0.875	0.2247	0.0256
Erdős–Rényi	0.415	0.1703	1.0000
ERGM	0.333	0.3333	1.0000
Lattice	0.482	0.0352	1.0000
DiGress	0.569	0.4065	0.4545
GDSS-Prodigy	0.552	0.3578	0.5385
MuDiff	0.433	0.7050	0.4286

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We believe that the abstract and introduction provide an accurate summary of the paper's contributions and scope. The claims are supported by the theoretical and empirical results presented and are stated within the limitations discussed in the manuscript.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We believe to have discussed the primary limitations of the proposed method, including scalability considerations, dependence on the choice of grammar and constraint design. These aspects are explicitly addressed in the limitations section of the paper.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We believe that the set of assumptions are clearly stated in the Methods and Background sections, while additional information and further clarifications are available in the appendix A, appendix B and appendix C sections.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide a complete description of the proposed method, including the parametric graph grammar specification, constrained Markov decision process formulation, reward structure, constraint definitions, and training procedure. Experimental protocols, benchmark configurations, and implementation details are documented in the Appendix to enable independent replication of the reported results. In addition, the full source code used to run the experiments will be released to further facilitate reproducibility.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The codebase is available with the submission and includes a README file and comments to facilitate review. Further READMEs files are available to easily reproduce the experiments of baselines.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The most relevant hyperparameters are released in the paper; for PPO training the hyperparameters are reported in Appendix B.4 and additional training and evaluation details are provided throughout the appendix. The complete set of hyperparameters of the benchmark are reported explicitly in Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The reported results include error bars computed over multiple independent runs with different random seeds. We report the mean performance together with the standard deviation across runs in all the relevant figures and tables of the manuscript.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We report the hardware and the time required to run the experiments in section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We confirm that the research complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not involve high-risk data or models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

Justification: We believe to have properly cited related works and software libraries used in this work (including URL where necessary), including but not limited to JAX, Stable-Baselines3, GDSS, DiGress, MuDiff and Prodigy. The licenses and terms of use of these assets have been respected in accordance with their respective documentation.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The released codebase is documented and includes a README file describing usage and training procedures.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The study does not involve human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are not used as part of the core methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.