TOPOCOMPILE: MOLECULAR GENERATION WITH EXACT GLOBAL COUNT CONSTRAINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Molecular generation designs novel molecules with desired structures and properties. Global count constraints specify molecular size, elemental composition, and ring content without prescribing detailed structure. Property-conditioned generators steer generation toward such targets but do not guarantee exact satisfaction. We introduce TOPOCOMPILE, which combines an autoregressive component-tree generator with a dynamic-programming compiler for supported component-additive integer counts. TOPOCOMPILE represents molecules as trees of intact ring systems and individual atoms, making each supported count additive over components. At each step, an autoregressive Transformer scores component and attachment actions, while the compiler masks actions with no target-satisfying completion. Within the fixed component vocabulary and typed attachment rules, constrained decoding guarantees exact equality for supported non-negative, component-additive integer counts. Applied at inference time, these checks let the same trained generator support different targets and count combinations without retraining. Across four benchmarks, TOPOCOMPILE achieves 100% Exact-valid yield for Common and Rare four-count target groups; the strongest baseline within each dataset reaches 44.9% or less on Rare targets. In unconditional generation, it achieves the highest validity or valid-unique-novel (V.U.N.) among the compared methods across all four benchmark datasets.

1 INTRODUCTION

Designing molecules with specified structural and physicochemical properties is a central problem in drug discovery and materials design. Neural molecular generators address this problem by learning distributions over molecular structures and using them to produce chemically plausible candidates beyond the training set (Gómez-Bombarelli et al., 2018; Jin et al., 2018; Zang & Wang, 2020; Shi et al., 2020; Luo et al., 2021). Recent advances in diffusion, flow, and autoregressive models have further improved molecular generation across graph and sequence representations (Vignac et al., 2023; Jolicoeur-Martineau et al., 2025; Wang et al., 2025).

In practical molecular design, chemically plausible candidates often also need to meet specific structural requirements. Global count constraints provide a direct way to express these requirements: users can specify molecular size, elemental composition, and ring content without prescribing detailed connectivity. Such constraints arise in molecular structure elucidation, where the molecular formula is known but the atom connectivity remains unresolved (Stravs et al., 2022; Bohde et al., 2025). They also support controlled comparison libraries for virtual screening and molecular-model evaluation (Imrie et al., 2021), while exact ring counts enable targeted exploration of atypical topologies such as high-cycle-rank molecules. The generator chooses detailed structures from the learned distribution while the user controls global molecular characteristics.

Existing approaches differ in how precisely they control generation and what users must specify. Property-conditioned generators steer generation toward multiple requested attributes but generally do not guarantee exact satisfaction (Liu et al., 2024; Jolicoeur-Martineau et al., 2025; Lee et al., 2026). Structure-conditioned methods can provide stronger structural control, but require a user-specified scaffold or partial structure, fixing part of the molecule in advance (Xu & Chen, 2023; Lee et al., 2025). Spectrum-conditioned methods instead require a mass spectrum and molecular formula (Bohde et al., 2025). Hard-constraint methods enforce selected rules through projection, search, or

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

symbolic reasoning (Sharma et al., 2024; Madeira et al., 2024; Saikali & Pesant, 2026; Geng et al., 2026). Existing approaches therefore do not directly address our setting: satisfying multiple count constraints exactly while leaving the scaffold and partial structure unspecified.

To achieve this, we build molecules from scratch and check before each addition whether the requested counts remain achievable. A locally valid addition can still make these targets unreachable, yet enumerating every possible completion would require a combinatorial search over future molecular structures. We address both difficulties with a lossless component-tree representation. We cut each bridge bond, whose removal disconnects the molecular graph, and treat every intact ring system and every atom outside rings as a component. The resulting component-level graph is a tree. Each supported count is the sum of fixed, non-negative integer contributions from these components. Each component also has attachment ports that identify the local atom and bond type for a connection. Under a fixed component vocabulary and attachment rules, whether the generator can complete a partial molecule depends only on the number of components remaining, unmatched ports by bond type, and remaining target counts. A dynamic program can therefore test for an exact completion using this compact state, without enumerating complete molecular graphs.

Building on this representation, we introduce **TopoCompile**, which combines an autoregressive Transformer for component-tree generation with a dynamic-programming compiler. At each step, the Transformer scores component and attachment choices, while the compiler removes choices that leave no target-satisfying completion. The generator then samples from the renormalized probabilities, preserving its relative preferences among the remaining choices. Thus, the generator learns to construct plausible molecules, while the compiler ensures that the requested counts remain achievable. Because we apply constraints only during decoding, the same trained generator supports different target values and count combinations without retraining. Within the fixed component vocabulary and attachment rules, constrained decoding guarantees target counts for every molecule.

Our contributions are:

- **Exact-control interface.** TopoCompile provides exact control over supported global counts while leaving the scaffold, component identities, and component-tree layout unspecified. The same trained generator supports different target values and combinations of supported counts without retraining.
- **Completion-checking compiler.** We derive a dynamic program over the remaining component count, unmatched ports by bond type, and remaining target counts. We prove that it determines exactly whether a target-satisfying completion exists under the fixed vocabulary and attachment rules, enabling exact decoding without rejection or repair.
- **Evaluation.** Across four benchmarks, TopoCompile achieves 100% exact-valid yield for both frequently and infrequently observed four-count targets, whereas the strongest baseline in each dataset reaches at most 44.9% on infrequent targets. In unconditional generation, it achieves the highest validity or yield of valid, unique, novel molecules among the compared methods on all four benchmarks.

2 Related Work

Molecular generation. Neural molecular generators fall into two categories: iterative global-update methods and sequential autoregressive methods. Diffusion models (Jo et al., 2022; Vignac et al., 2023; Liu et al., 2024; Lee et al., 2025) and discrete flow-matching models (Eijkelboom et al., 2024; Qin et al., 2025; Lee et al., 2026) repeatedly refine a global state represented as a molecular graph, sequence, or fragment graph. Autoregressive models instead construct molecules through local decisions, adding atoms and bonds sequentially (Shi et al., 2020; Luo et al., 2021) or optimizing the generation order (Wang et al., 2025). To reduce sequential complexity, several frameworks use trees to organize motifs or guide generation (Jin et al., 2018; 2020; Ahn et al., 2022). Grammar-based models constrain generation through syntactic rules (Kusner et al., 2017), semantic attributes (Dai et al., 2018), or molecular graph grammars (Kajino, 2019). These methods target valid construction; TopoCompile checks whether each partial molecule can meet the requested global counts.

Constrained molecular generation. Existing methods provide different levels of control over generated molecules. Reward-based and property-conditioned approaches bias generation toward

desired molecular properties, but generally do not guarantee exact satisfaction of requested values (You et al., 2018; Liu et al., 2024; Jolicoeur-Martineau et al., 2025). Structure-guided methods provide stronger control by conditioning on user-specified scaffolds or topological templates; for example, Tree-Invent enforces user templates through masked generation (Xu & Chen, 2023). However, such methods fix part of the molecular structure in advance, reducing structural freedom. Hard-constraint methods go further by enforcing selected constraints through projection, constraint programming, or symbolic reasoning (Sharma et al., 2024; Madeira et al., 2024; Saikali & Pesant, 2026; Geng et al., 2026; Omrani & Naanaa, 2020). Related mechanisms such as valence masks (Liu et al., 2018) and valid-by-construction representations (Krenn et al., 2020) guarantee aspects of chemical validity, but do not by themselves guarantee exact satisfaction of a user-specified global count vector. TopoCompile targets a different setting: exact control of multiple global counts while leaving the molecular components and their arrangement unspecified.

3 The TopoCompile Method

Problem formulation. We consider exact constrained molecular generation with a trained generator p_θ. Let $R : \mathcal{G} \to \mathbb{N}_0^d$ map a supported molecular graph G to a vector of discrete global counts, and let $\mathbf{b}^\star \in \mathbb{N}_0^d$ be the user-specified target. The task is to generate $G \in \mathcal{G}$ such that $R(G) = \mathbf{b}^\star$; if the user additionally specifies a total component count M, the generated graph must contain exactly M components. The scaffold, component identities, and component-tree layout remain unspecified, and different target values and combinations of supported counts must be handled by the same trained generator without retraining or post-hoc rejection or repair of completed samples.

Overview. Section 3.1 introduces the component-tree representation and autoregressive generator. Section 3.2 derives a dynamic program for exact completion checking, and Section 3.3 uses it to mask infeasible actions during decoding. **Figure 1** provides an overview of the complete training and constraint-guided sampling pipeline.

3.1 Offline learning of the component-tree generator

Tree representation of molecular graphs. We represent each molecule as a tree of molecular components. This representation reduces autoregressive generation to adding one component to one parent at each step while retaining enough information to reconstruct the original molecular graph. For a connected molecular graph $G = (V, E, \mathbf{x}, \mathbf{y})$, an edge is a bridge if removing it disconnects G. In a molecular graph, bridge edges are exactly the edges that do not belong to any ring. Removing these non-ring edges partitions G into atom-disjoint components $\mathcal{C}(G) = \{C_1, \ldots, C_M\}$. Replacing each component with one node and retaining one edge for each bridge produces a component-level graph $\bar{T}(G)$ that records the connections among components. As shown in Figure 1(A2), its nodes fall into two types: single atoms and ring systems; ring-system nodes include single rings and fused, spiro, or bridged multi-ring structures. By bridge decomposition, $\bar{T}(G)$ is a tree (Tarjan, 1974).

Component vocabulary and encoding. For each component C_i, $\mathcal{P}_i$ lists the ports exposed by removed bridges. Each port records its local atom, bridge-bond type, and symmetry class for equivalent ports. We treat $\widehat{C}_i = (C_i, \mathcal{P}_i)$ as a port-annotated template and write $\widehat{\mathcal{C}}(G) = \{\widehat{C}_i : C_i \in \mathcal{C}(G)\}$. Canonicalization independent of atom ordering gives $\mathcal{V} = \mathrm{Unique}(\bigcup_{G \in \mathcal{D}_{\mathrm{train}}} \widehat{\mathcal{C}}(G))$. Each component template $\tau \in \mathcal{V}$ stores a complete internal graph and a fixed port set $\mathcal{P}(\tau)$, which determines $\boldsymbol{\sigma}(\tau)$. Only canonically identical templates share a vocabulary entry (Figure 1(A2)).

To encode a molecule, we assign the corresponding template $\tau_i \in \mathcal{V}$ to each component C_i, with $\mathcal{P}_i = \mathcal{P}(\tau_i)$. The component tree identifies which components connect, but each tree edge must also identify the atoms that form the connection. For every edge (i, j), we record ports $p_{i\to j}$ and $p_{j\to i}$ on the two components and retain the original bridge attribute $\mathbf{y}_{ij}^{\mathrm{bridge}}$. The arrows distinguish the two ends of an undirected bond; they do not indicate bond directions.

For the ring-system–Br connection in Figure 1(A2), the endpoint ports locate the attachment atoms, and $\mathbf{y}_{ij}^{\mathrm{bridge}}$ records the bond type. The complete encoding is

$$\mathrm{Enc}(G) = \left(\{(\tau_i, \mathcal{P}_i)\}_{i=1}^{M}, \{(i, j, p_{i\to j}, p_{j\to i}, \mathbf{y}_{ij}^{\mathrm{bridge}})\}_{(i,j)\in E_{\bar{T}}} \right). \tag{1}$$

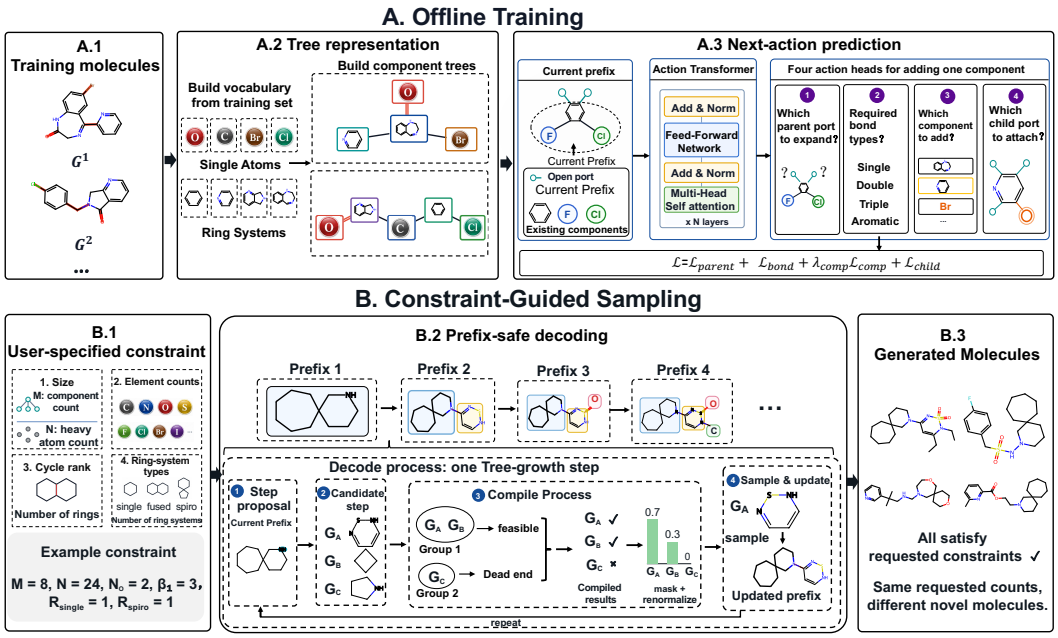


Figure 1: Overview of TOPOCOMPILE. (A) Molecules are encoded as trees of atom and ring-system components to train an autoregressive generator. (B) During sampling, the dynamic program masks additions with no target-satisfying completion; the completed tree is decoded into a molecule.

The decoder retrieves component graphs and restores bridges from their ports and bond attributes.

Let $\mathcal{G}_{\text{rep}}$ contain connected graphs after canonicalization that omits stereochemical and isotopic annotations. Their atom states, bond states, and port-local indices must lie in the declared finite encoding sets, and their port-annotated templates must belong to $\mathcal{V}$.

Proposition 1 (Lossless component-tree encoding)**.** *For every $G \in \mathcal{G}_{\text{rep}}$, $\text{Dec}(\text{Enc}(G)) \simeq G$, where $\simeq$ preserves the stored atom and bond attributes.*

Appendix A.2 proves the proposition. Generation recombines fixed component templates rather than creating new ring interiors or port configurations; whole-molecule test coverage ranges from 93.3% to 99.3% (Appendix F).

Construction sequences. We randomize the root and use parent-before-child traversals to expose the generator to multiple valid construction sequences for each molecule. The traversal also ensures that every non-root component has an existing parent when the model adds it. Each template τ has a bond-type vector $\boldsymbol{\sigma}(\tau) = (n_{\text{single}}, n_{\text{double}}, n_{\text{triple}}, n_{\text{aromatic}})$ that counts its attachment ports by bridge-bond type. Let h_t be the partial tree before step t. A non-root action $a_t = (p_t, \boldsymbol{\sigma}_t, \tau_t, q_t)$ selects an open parent port p_t, a component τ_t with $\boldsymbol{\sigma}_t = \boldsymbol{\sigma}(\tau_t)$, and a child port q_t of matching bond type. Adding the component joins the selected ports with a bridge, consumes both ports, and leaves its other ports open. The root action selects only a bond-type vector and component, with $p_1 = q_1 = \text{null}$.

Next-action prediction and training. We train an autoregressive Transformer for $p_\theta(a_t \mid h_t, M)$, where M is the total component count. Four prediction heads factor each addition, allowing compatibility to narrow successive choices without a separate class for every parent-port, bond-type vector, component, and child-port tuple. The parent-port and bond-vector heads predict where to extend the tree and the new component's connection requirements. The component head selects from $\mathcal{V}(\boldsymbol{\sigma}_t) = \{\tau \in \mathcal{V} : \boldsymbol{\sigma}(\tau) = \boldsymbol{\sigma}_t\}$, and the child-port head selects a port matching the parent bond type. Under teacher forcing, each head predicts its corresponding field of $a_t^\star$ from the ground-truth partial tree $h_t^\star$, using the weighted cross-entropy loss $\mathcal{L} = \mathcal{L}_{\text{bond}} + \lambda_{\text{comp}}\mathcal{L}_{\text{comp}} + \mathcal{L}_{\text{parent}} + \mathcal{L}_{\text{child}}$, where $\lambda_{\text{comp}} > 0$ weights component prediction (Appendix B.2). At the root, only bond-vector and

component losses apply because no parent attachment exists. Unconditional generation draws M from its empirical training distribution and samples actions autoregressively from p_θ; constrained generation reuses this model with the completion checks.

3.2 Compiling exact global constraints

The learned generator scores plausible component additions, whereas exact control requires knowing whether each addition leaves a target-satisfying completion. Model probabilities alone do not provide this feasibility test. A dynamic program performs the test from a compact summary of the remaining requirements without enumerating future molecular structures.

Targets and component contributions. The user specifies a vector of target counts $\mathbf{b}^\star$, without specifying components or their tree layout. For a fixed total component count, we write $\mathbf{c}^\star = (M, \mathbf{b}^\star)$. When the user omits M, we check $M = 1, \ldots, M_{\max}$ and sample among the feasible values using the restricted empirical prior in Appendix B.4. Here, $M_{\max}$ is the generator's component-count limit.

We call $r : \mathcal{V} \to \mathbb{N}_0^d$ a supported count map for R if it assigns each template a non-negative integer contribution, attachment leaves these contributions unchanged, and every constructible molecule satisfies $R(G) = \sum_{i=1}^{M} r(\tau_i)$. These conditions allow us to update the remaining targets by subtracting $r(\tau)$ after adding a component. Heavy-atom and element counts satisfy them directly. Ring-system count and cycle rank do as well because bridge attachments do not create cycles between components. Appendix A.5 derives this additivity. The current compiler supports exact equalities over these non-negative, component-additive integer counts. Quantities that depend on component connections, such as cross-component motifs, require a different state or acceptance rule. The formulation excludes continuous descriptors, inequalities, and optimization objectives.

A compact state for completion checking. A partial tree's completion feasibility depends on its remaining component budget, open attachment ports, and residual target counts. After root selection, we collect them in $z = (k, \mathbf{f}, \mathbf{b})$: k counts the components still to add, $\mathbf{f} \in \mathbb{N}_0^4$ counts open ports by bond type, and $\mathbf{b}$ contains the counts that future components must supply. The coordinates of $\mathbf{f}$ match $\boldsymbol{\sigma}(\tau)$ but summarize the entire partial tree.

The compiler treats two ports as compatible exactly when their bridge-bond types match. Atom locations change the molecule but not compatibility, and each template fixes its count contribution. Partial trees with the same $(k, \mathbf{f}, \mathbf{b})$ share the same completion-feasibility result.

Grouping additions with the same effect. For an action $a = (p, \boldsymbol{\sigma}, \tau, q)$, the state update depends only on $\gamma(a) = (\boldsymbol{\sigma}(\tau), r(\tau), \ell(q))$, where $\ell(q)$ is the child port's bond type. We group actions by this tuple and denote the finite group set by Γ. For $\gamma = (\boldsymbol{\sigma}_\gamma, r_\gamma, \ell_\gamma)$, let $\mathbf{e}_{\ell_\gamma}$ denote the unit vector for bond type ℓ_γ. The transition is

$$\mathrm{Next}(z, \gamma) = \big(k - 1,\ \mathbf{f} + \boldsymbol{\sigma}_\gamma - 2\mathbf{e}_{\ell_\gamma},\ \mathbf{b} - r_\gamma\big). \tag{2}$$

The new component supplies r_γ and its ports; the bridge consumes one existing and one new open port of type ℓ_γ. Grouping shares only feasibility computation: the generator still scores concrete components and attachment sites.

Joint feasibility DP. A feasible continuation must use one sequence of additions to close every open port and reduce every residual count to zero. Let $\Gamma(z)$ contain groups satisfying $\mathbf{f}_{\ell_\gamma} \geq 1$, $(\boldsymbol{\sigma}_\gamma)_{\ell_\gamma} \geq 1$, and $r_\gamma \leq \mathbf{b}$ coordinatewise. Define $F_{\mathbf{c}^\star}(z)$ to indicate whether exactly k further additions can close all ports and supply the remaining counts:

$$F_{\mathbf{c}^\star}(0, \mathbf{f}, \mathbf{b}) = \mathbf{1}[\mathbf{f} = \mathbf{0} \wedge \mathbf{b} = \mathbf{0}], \qquad F_{\mathbf{c}^\star}(z) = \bigvee_{\gamma \in \Gamma(z)} F_{\mathbf{c}^\star}(\mathrm{Next}(z, \gamma)), \quad k > 0. \tag{3}$$

For example, with one component left, the same template must close all open ports and supply the residual counts; different templates satisfying these requirements separately do not suffice. Separate structural and count reachability checks may succeed through different continuations, so the compiler uses them only to prune states. It evaluates the joint recurrence on demand and memoizes results across construction histories. Appendix A.8 gives the algorithm and pruning rules.

3.3 CONSTRAINED SAMPLING

Constrained decoding. For fixed $\mathbf{c}^\star = (M, \mathbf{b}^\star)$, a root component τ with $r(\tau) \le \mathbf{b}^\star$ leaves state $z_{\mathrm{root}}(\tau) = (M-1, \boldsymbol{\sigma}(\tau), \mathbf{b}^\star - r(\tau))$. We retain roots satisfying $F_{\mathbf{c}^\star}(z_{\mathrm{root}}(\tau)) = 1$ and sample from their renormalized model probabilities. When the user does not specify M, these checks test all allowed values before selecting among feasible roots. At each non-root step, the partial component tree h_t provides the generator's molecular context, while z_t specifies the completion requirements. For each queried group, we cache $\Lambda_{\mathbf{c}^\star}(z, \gamma) = F_{\mathbf{c}^\star}(\mathrm{Next}(z, \gamma))$. Provided the retained choices have positive total model probability, we sample from

$$\widetilde{p}_\theta(a \mid h_t, z_t, \mathbf{c}^\star) = \frac{p_\theta(a \mid h_t, M)\Lambda_{\mathbf{c}^\star}(z_t, \gamma(a))}{\sum_{a'} p_\theta(a' \mid h_t, M)\Lambda_{\mathbf{c}^\star}(z_t, \gamma(a'))}. \tag{4}$$

This preserves probability ratios among retained actions. It does not in general equal the original model distribution conditioned on the targets because Boolean feasibility records the existence of a matching continuation, not its model probability. Thus, exactness here refers to satisfaction of the requested counts, not exact sampling from $p_\theta(\cdot \mid R = \mathbf{b}^\star)$. After sampling, we attach the component, update z_t using Equation equation 2, and continue until $k = 0$. We then decode the completed component tree into a molecular graph.

Theorem 1 (Prefix safety and exact completion)**.** *Fix a finite vocabulary $\mathcal{V}$, the bond-type matching attachment rules defined above, a supported count map $r : \mathcal{V} \to \mathbb{N}_0^d$, and a target $\mathbf{c}^\star = (M, \mathbf{b}^\star)$. Assume that the feasible roots have positive total model probability and that the retained actions do so at every nonterminal state reached during sampling. Then every sampled partial tree admits a target-satisfying continuation, and sampling can continue until $k = 0$. Every terminal sample has exactly M components, all ports matched, and $R(G) = \mathbf{b}^\star$.*

Proof sketch. The root check starts from a feasible state, and each retained action preserves completion feasibility. At $k = 0$, the terminal condition $\mathbf{f} = \mathbf{b} = \mathbf{0}$ gives port closure and exact target counts. Appendix A.7 gives the full proof.

Scope and failure behavior. The guarantee is relative to the component vocabulary and the bond-type matching rules above. The compiler certifies infeasibility only if every allowed M compiles and none admits a feasible root. A joint-DP state-limit error makes no infeasibility claim. Zero model probability across retained choices leaves the sampling distribution undefined despite DP feasibility. Exact count satisfaction and port closure do not establish chemical validity or synthesizability; Section 4 evaluates validity separately.

Time complexity. Let d count controlled quantities, S all DP states evaluated during compilation or sampling, and $G_{\max}$ the maximum groups examined per state. $T_{\mathrm{dec}}(M)$ covers model prediction, candidate processing, cached checks, and reconstruction. After preprocessing, compiling one target and generating B molecules costs $O(dSG_{\max} + B\,T_{\mathrm{dec}}(M))$. All outputs share the compiled table, giving an amortized compilation cost of $O(dSG_{\max}/B)$ per molecule. Its pseudo-polynomial dependence on target values can make large joint requests expensive. Appendix A.3 derives the bound and covers the case where the user does not specify M.

4 EXPERIMENTS

We evaluate whether TOPOCOMPILE delivers exact counts while preserving generation quality and practical cost. We study four questions: **RQ1:** How reliably does TOPOCOMPILE generate molecules with requested counts? **RQ2:** What do joint completion checking and learned preferences contribute? **RQ3:** How competitive is TOPOCOMPILE in unconditional generation? **RQ4:** How do methods compare in exact-output delivery efficiency?

4.1 EXPERIMENTAL SETUP

Datasets and tasks. We evaluate on MOSES (Polykovskiy et al., 2020), ZINC250K (Sterling & Irwin, 2015), GuacaMol (Brown et al., 2019), and NPGen (Lee et al., 2026). TopoCompile supports different combinations of component-additive count constraints. We evaluate eight constraint

schemas over heavy-atom count N, cycle rank β, oxygen count N_{O}, and ring-system count R_{sys}. A four-count request specifies target values for all four quantities $(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$, whereas the other schemas specify only one, two, or three of them. The user does not specify the component count M; TopoCompile instead considers supported M values and selects among those admitting a feasible completion. The main text focuses on the four-count schema, while results for the subset schemas are reported in Appendix C. For unconditional generation, where no count targets are given, TopoCompile samples M directly from its empirical training distribution and generates autoregressively. Detailed task settings and evaluation protocols are provided in Appendices B.4 and C.

Baselines. For exact-control generation, we compare TOPOCOMPILE with Graph DiT (Liu et al., 2024), STGG+ (Jolicoeur-Martineau et al., 2025), FragFM+Guidance (Lee et al., 2026), GenMol+Filter (Lee et al., 2025), and STGG+ Count Mask under the same target requests and attempt budgets. Appendix B.3 details baseline implementations and configurations. For unconditional generation, Table 2 lists the established molecular generators compared with TOPOCOMPILE.

Evaluation metrics. For exact-control generation, **Valid** measures chemical validity, while **Exact-valid** additionally requires every requested count to be satisfied exactly. **Novelty** and **internal diversity** assess whether exact outputs are new and structurally diverse. In the sampling-policy ablation, where all compared policies use the same exact completion check, conditional FCD measures how closely their exact outputs match the target-specific reference distribution. For unconditional generation, we follow each benchmark's standard Validity, valid-unique-novel (V.U.N.) yield, and FCD protocol. Appendices B.4 and C detail the settings.

4.2 RQ1: HOW RELIABLY DOES TOPOCOMPILE MEET REQUESTED COUNT CONSTRAINTS?

Table 1: Exact four-count generation $(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$. **(A)** Exact-valid yield on Common and Rare targets. **(B)** Novelty and internal diversity of matched Common exact outputs.

A. Exact-valid yield (%)

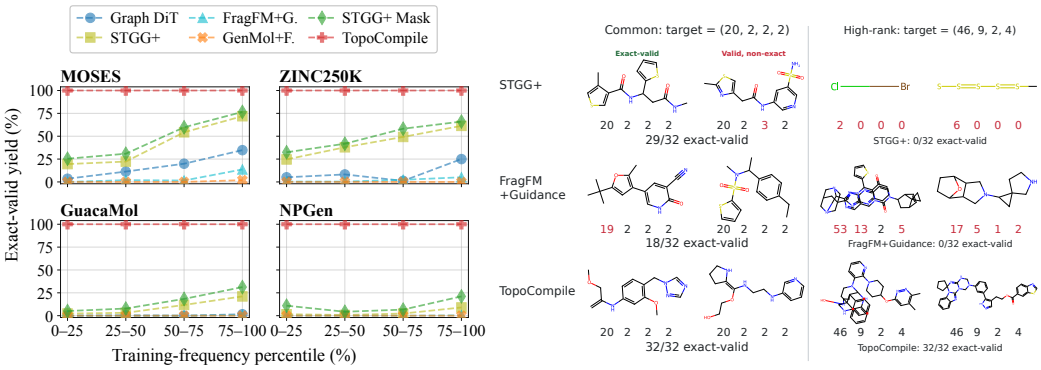
Method	MOSES		ZINC250K		GuacaMol		NPGen	
	Common	*Rare*	*Common*	*Rare*	*Common*	*Rare*	*Common*	*Rare*
Graph DiT	56.64	5.57	43.36	8.40	0.49	4.30	OOM	
STGG+	80.86	29.00	79.59	37.50	34.86	13.87	16.89	3.71
FragFM+Guidance	22.07	0.20	11.04	0.00	4.98	0.00	1.56	0.00
GenMol+Filter	1.86	0.00	1.46	0.00	1.66	0.00	0.00	0.00
STGG+ Count Mask	83.59	35.55	82.71	44.92	46.88	15.53	27.44	14.06
TopoCompile (ours)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**

B. Matched exact-output novelty and diversity

Method	MOSES		ZINC250K		GuacaMol		NPGen	
	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑
STGG+	0.946	0.845	**1.000**	0.860	0.994	0.867	**0.978**	0.883
FragFM+Guidance	0.899	0.846	0.987	0.852	0.988	0.858	0.689	0.860
STGG+ Count Mask	0.952	0.847	**1.000**	0.861	0.994	0.869	**0.978**	0.883
TopoCompile (ours)	**0.990**	**0.862**	**1.000**	**0.872**	**1.000**	**0.882**	0.956	**0.900**

Overall performance. For each dataset, observed four-count tuples with $R_{\mathrm{sys}} \geq 1$ are **Common** at or above the 75th percentile of training frequency and **Rare** at or below the 25th. We select up to 32 targets per group. Table 1(A) reports Exact-valid yield over 32 attempts per target. Table 1(B) compares novelty and internal diversity using equal numbers of exact-valid molecules per Common target from each reported method's first 100 attempts; targets with fewer than two from any method are excluded (Appendix B.4). Figure 2(A) ranks tuples by frequency, divides them into quartiles, and evaluates eight targets per quartile; Figure 2(B) shows Common and high-cycle-rank outputs.

As shown in Table 1, TOPOCOMPILE achieves 100% Exact-valid yield on both Common and Rare four-count targets across all four datasets, whereas the strongest baseline in each dataset reaches only 14.1–44.9% on Rare targets. On matched Common exact outputs, TOPOCOMPILE also maintains high novelty and achieves the highest internal diversity on all four datasets. **Figure 2(A)** shows that baseline Exact-valid yield generally increases with frequency, while TOPOCOMPILE remains at 100% across all four quartiles. **Figure 2(B)** shows that competing methods can produce valid



(A) Training-frequency sensitivity **(B) Common and high-rank four-count outputs**

Figure 2: Exact control across target frequencies and representative outputs. (A) Exact-valid yield across training-frequency quartiles. (B) Representative Common and high-cycle-rank outputs.

molecules while missing target counts, whereas TopoCompile satisfies all four, including a high-cycle-rank request beyond the MOSES training maximum.

4.3 RQ2: What do joint checks and learned preferences contribute?

We conduct two ablations on MOSES and ZINC250K across five constraint schemas, pooling Common and Rare requests. First, with the generator and sampling policy fixed, we vary the completion check: Local rejects immediate structural or count violations; Factorized additionally checks structural and count completion separately, potentially using different continuations; Joint requires one continuation satisfying both. Second, with Joint fixed, Uniform assigns equal probabilities to legal options for each action field, Frequency uses their training frequencies, and Learned uses Transformer probabilities. Figure 3(a,b) reports Exact-valid yield, and Figure 3(c,d) reports conditional FCD. Appendix E provides details.

As shown in Figure 3(a,b), Joint achieves 100% Exact-valid yield across all evaluated settings, versus 67.1–86.2% for Factorized and 12.4–56.2% for Local, showing that separate feasibility checks cannot replace joint completion checking. Under Joint, all three sampling policies retain 100% Exact-valid yield, but Learned achieves the lowest conditional FCD in every schema (Figure 3(c,d)). Learned therefore improves alignment with target-specific references, although Uniform has higher internal diversity in some settings (Appendix E).

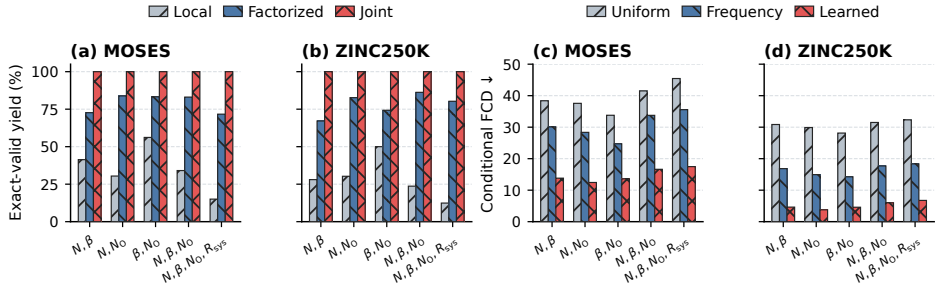


Figure 3: Schema-level ablations on MOSES and ZINC250K with Common and Rare requests pooled. **(a,b)** Exact-valid yield under Local, Factorized, and Joint completion checks. **(c,d)** Conditional FCD under Uniform, Frequency, and Learned sampling with Joint checking fixed.

4.4 RQ3: How competitive is TopoCompile in unconditional generation?

As shown in Table 2, TopoCompile achieves the highest V.U.N. yield on GuacaMol, MOSES, and NPGen, and the highest validity on ZINC250K. On MOSES and NPGen, it improves V.U.N.

Table 2: Unconditional molecular generation. GuacaMol reports FCD score (↑); other datasets report FCD distance (↓). Bold indicates the best result in each column.

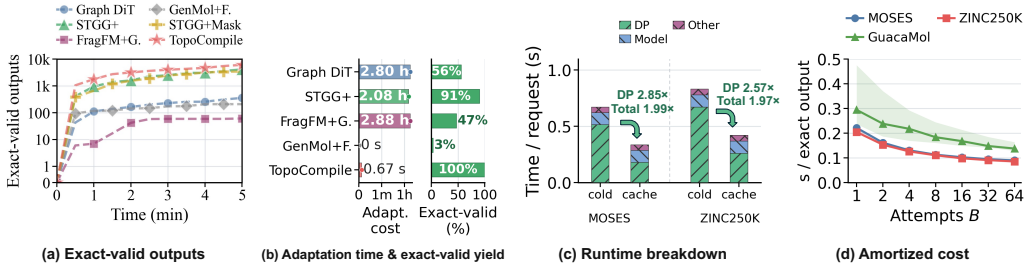
Method	GuacaMol		MOSES		NPGen		ZINC250K	
	V.U.N. ↑	FCD score ↑	V.U.N. ↑	FCD ↓	V.U.N. ↑	FCD ↓	Validity ↑	FCD ↓
DiGress (Vignac et al., 2023)	85.1	68.0	81.4	1.19	85.1	2.05	94.98	3.482
DeFoG (Qin et al., 2025)	97.9	73.8	85.4	1.95	83.8	4.46	99.22	1.425
GraphAF (Shi et al., 2020; Lee et al., 2026)	–		85.2	0.523	48.1	25.11	67.92	16.128
Tree-Invent (Xu & Chen, 2023)	86.2	58.1	OOM		75.3	8.20	95.43	5.999
FragFM (Lee et al., 2026)	95.0	**85.8**	86.9	**0.58**	92.6	1.34	99.81	0.630
TopoCompile (ours)	**98.2**	80.8	**92.5**	0.88	**97.4**	**0.841**	**100.00**	**0.520**

over the strongest reported baseline by 5.6 and 4.8 percentage points, respectively. Distributional alignment shows a more mixed picture: TopoCompile achieves the best reported FCD on NPGen and ZINC250K, while FragFM performs better on GuacaMol and MOSES. These results also separate the two roles in TopoCompile: the component-tree generator remains competitive without constraint checks, while the compiler is responsible for exact control.

4.5 RQ4: How do methods compare in exact-output delivery efficiency?

Figure 4(a) measures the number of exact-valid molecules generated within a fixed five-minute budget. Figure 4(b) compares the cost of adapting to a new target and the resulting Exact-valid yield. Figure 4(c) measures the benefit of reusing cached compilation results, and Figure 4(d) examines how generating multiple molecules for the same target amortizes the one-time compilation cost.

As shown in Figure 4, TopoCompile generates more exact-valid molecules within the fixed time budget, demonstrating higher end-to-end efficiency (a). For new targets, its main advantage is that adaptation only requires compilation, with no retraining or tuning of the generator, while filtering avoids adaptation at the cost of much lower exact-valid yield (b). Cache reuse further reduces latency for repeated requests by reusing previously solved feasibility states (c). Finally, more molecules amortize a target's one-time compilation cost, but one-off requests pay the full cost (d). A fixed-M ablation shows that grouping equivalent actions reduces median compilation time by 12.2–13.9× without changing the DP state count (Appendix E).



(a) Exact-valid outputs (b) Adaptation time & exact-valid yield (c) Runtime breakdown (d) Amortized cost

Figure 4: Efficiency of exact constrained generation. (a) Exact-valid molecules generated within a fixed time budget. (b) Target adaptation cost and Exact-valid yield. (c) Cold and cached request time. (d) Amortized time per exact-valid molecule as the number of samples per target increases.

5 Conclusion

We introduce TopoCompile for molecular generation under exact global count constraints. Its autoregressive component-tree generator ranks plausible additions, while dynamic-programming completion checks retain only additions that can reach the requested counts. TopoCompile satisfies supported counts exactly while retaining molecular quality and diversity, and its trained generator also produces high-quality molecules without constraints. Future work will extend exact control to connection-dependent constraints through richer completion states and broaden component support to enlarge the reachable molecular space while preserving the current exact completion guarantees.

Reproducibility Statement

The anonymous repository at anonymous repository contains the source code and experiment scripts. Appendix B documents the data preprocessing, model configuration, and evaluation protocols needed to reproduce the reported results.

Ethics Statement

This work is computational and does not involve human subjects or private data. We acknowledge that molecular generation may have downstream biomedical applications and should be used with appropriate experimental validation.

AI use statement

Generative AI tools were used only to assist with writing and language polishing, including improving clarity, grammar, and presentation. All technical content, methods, experiments, results, analyses, and conclusions were developed and verified by the authors. The authors take responsibility for the final code, experiments, claims, citations, figures, and text.

References

Sungsoo Ahn, Binghong Chen, Tianzhe Wang, and Le Song. Spanning tree-based graph generation for molecules. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=w60btE_8T2m.

Montgomery Bohde, Mrunali Manjrekar, Runzhong Wang, Shuiwang Ji, and Connor W. Coley. DiffMS: Diffusion generation of molecules conditioned on mass spectra. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 4737–4756. PMLR, 2025. URL https://proceedings.mlr.press/v267/bohde25a.html.

Nathan Brown, Marco Fiscato, Marwin H. S. Segler, and Alain C. Vaucher. GuacaMol: Benchmarking models for de novo molecular design. *Journal of Chemical Information and Modeling*, 59(3): 1096–1108, 2019. doi: 10.1021/acs.jcim.8b00839.

Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyqShMZRb.

Floor Eijkelboom, Grigory Bartosh, Christian A. Naesseth, Max Welling, and Jan-Willem van de Meent. Variational flow matching for graph generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 11735–11764, 2024. doi: 10.52202/079017-0374.

Chuqin Geng, Li Zhang, Mark Zhang, Haolin Ye, Ziyu Zhao, and Xujie Si. Neural proposals, symbolic guarantees: Neuro-symbolic graph generation with hard constraints. *arXiv preprint arXiv:2602.16954*, 2026. doi: 10.48550/arXiv.2602.16954. URL https://arxiv.org/abs/2602.16954.

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018. doi: 10.1021/acscentsci.7b00572.

Fergus Imrie, Anthony R. Bradley, and Charlotte M. Deane. Generating property-matched decoy molecules using deep learning. *Bioinformatics*, 37(15):2134–2141, 2021. doi: 10.1093/bioinformatics/btab080.

Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2323–2332, 2018. URL https://proceedings.mlr.press/v80/jin18a.html.

Wengong Jin, Regina Barzilay, and Tommi S. Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *Proceedings of the 37th International Conference on Machine Learn-*

ing, volume 119 of *Proceedings of Machine Learning Research*, pp. 4839–4848, 2020. URL https://proceedings.mlr.press/v119/jin20a.html.

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

Alexia Jolicoeur-Martineau, Aristide Baratin, Kisoo Kwon, Boris Knyazev, and Yan Zhang. Any-property-conditional molecule generation with self-criticism using spanning trees. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=QGZd5Bfb1L.

Hiroshi Kajino. Molecular hypergraph grammar with its application to molecular optimization. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3183–3191, 2019. URL https://proceedings.mlr.press/v97/kajino19a.html.

Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alan Aspuru-Guzik. Self-referencing embedded strings (SELFIES): A 100% robust molecular string representation. *Machine Learning: Science and Technology*, 1(4):045024, 2020. doi: 10.1088/2632-2153/aba947.

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1945–1954, 2017. URL https://proceedings.mlr.press/v70/kusner17a.html.

Joongwon Lee, Seonghwan Kim, Seokhyun Moon, Hyunwoo Kim, and Woo Youn Kim. FragFM: Hierarchical framework for efficient molecule generation via fragment-level discrete flow matching. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/92f556b408cfac2943fb228207487967-Abstract-Conference.html.

Seul Lee, Karsten Kreis, Srimukh Prasad Veccham, Meng Liu, Danny Reidenbach, Yuxing Peng, Saee Gopal Paliwal, Weili Nie, and Arash Vahdat. GenMol: A drug discovery generalist with discrete diffusion. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 33205–33226. PMLR, 2025. URL https://proceedings.mlr.press/v267/lee25o.html.

Gang Liu, Jiaxin Xu, Tengfei Luo, and Meng Jiang. Graph diffusion transformers for multi-conditional molecular generation. In *Advances in Neural Information Processing Systems*, volume 37, pp. 8065–8092, 2024. doi: 10.52202/079017-0260.

Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/b8a03c5c15fcfa8dae0b03351eb1742f-Abstract.html.

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. GraphDF: A discrete flow model for molecular graph generation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7192–7203, 2021. URL https://proceedings.mlr.press/v139/luo21a.html.

Manuel Madeira, Clément Vignac, Dorina Thanou, and Pascal Frossard. Generative modelling of structurally constrained graphs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 137218–137262, 2024. doi: 10.52202/079017-4360.

Alicja Maksymiuk, Alexandre Duplessis, Michael Bronstein, Alexander Tong, Fernanda Duarte, and Ismail Ilkan Ceylan. MacroGuide: Topological guidance for macrocycle generation. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2602.14977.

Virasone Manibod, David Saikali, and Gilles Pesant. Constrained sequential inference in machine learning using constraint programming. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pp. 2628–2636, 2025. doi: 10.24963/ijcai.2025/293.

Mohamed Amine Omrani and Wady Naanaa. Constraints for generating graphs with imposed and forbidden patterns: An application to molecular graphs. *Constraints*, 25(1–2):1–22, 2020. doi: 10.1007/s10601-019-09305-x.

Daniil Polykovskiy, Alexander Zhebrak, Benjamin Sanchez-Lengeling, Sergey Golovanov, Oktai Tatanov, Stanislav Belyaev, Rauf Kurbanov, Aleksey Artamonov, Vladimir Aladinskiy, Mark

Veselov, Artur Kadurin, Simon Johansson, Hongming Chen, Sergey Nikolenko, Alán Aspuru-Guzik, and Alex Zhavoronkov. Molecular sets (MOSES): A benchmarking platform for molecular generation models. *Frontiers in Pharmacology*, 11:565644, 2020. doi: 10.3389/fphar.2020.565644.

Kristina Preuer, Philipp Renz, Thomas Unterthiner, Sepp Hochreiter, and Günter Klambauer. Fréchet ChemNet distance: A metric for generative models for molecules in drug discovery. *Journal of Chemical Information and Modeling*, 58(9):1736–1741, 2018. doi: 10.1021/acs.jcim.8b00234.

Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. DeFoG: Discrete flow matching for graph generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 50269–50326, 2025. URL https://proceedings.mlr.press/v267/qin25d.html.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010. doi: 10.1021/ci100050t.

David Saikali and Gilles Pesant. Constrained molecule generation modelled using the grammar constraint. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 14313–14321, 2026. doi: 10.1609/aaai.v40i17.38446.

Kartik Sharma, Srijan Kumar, and Rakshit S. Trivedi. Diffuse, sample, project: Plug-and-play controllable graph generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 44545–44564. PMLR, 2024. URL https://proceedings.mlr.press/v235/sharma24b.html.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. GraphAF: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1esMkHYPr.

Teague Sterling and John J. Irwin. ZINC 15: Ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. doi: 10.1021/acs.jcim.5b00559.

Michael A. Stravs, Kai Dührkop, Sebastian Böcker, and Nicola Zamboni. MSNovelist: De novo structure generation from mass spectra. *Nature Methods*, 19(7):865–870, 2022. doi: 10.1038/s41592-022-01486-3.

Robert Endre Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2 (6):160–161, 1974. doi: 10.1016/0020-0190(74)90003-9.

Clément Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. DiGress: Discrete denoising diffusion for graph generation. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UaAD-Nu86WX.

Zhe Wang, Jiaxin Shi, Nicolas Heess, Arthur Gretton, and Michalis K. Titsias. Learning-order autoregressive models with application to molecular graph generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 64282–64301. PMLR, 2025. URL https://arxiv.org/abs/2503.05979.

Mingyuan Xu and Hongming Chen. Tree-Invent: A novel multipurpose molecular generative model constrained with a topological tree. *Journal of Chemical Information and Modeling*, 63(22): 7067–7082, 2023. doi: 10.1021/acs.jcim.3c01626.

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, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/d60678e8f2ba9c540798ebbde31177e8-Abstract.html.

Chengxi Zang and Fei Wang. MoFlow: An invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 617–626. ACM, 2020. doi: 10.1145/3394486.3403104.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. doi: 10.52202/079017-3670.

A Proofs and Implementation Details

A.1 Bridge-component tree

Proof. Let $G = (V, E)$ be connected, let $\mathcal{B}$ be its bridges, and let $C_1, \ldots, C_M$ be the connected components of $(V, E \setminus \mathcal{B})$. Construct $\bar{T}(G)$ with one vertex per component and one edge per bridge. Contracting connected subgraphs preserves connectivity, so $\bar{T}(G)$ is connected. Suppose $\bar{T}(G)$ contained a cycle and choose an edge on that cycle corresponding to bridge (u, v). The other cycle edges correspond to a sequence of components connected by bridges without using (u, v). Paths inside these components then connect u to v without that edge, contradicting that (u, v) is a bridge. Hence $\bar{T}(G)$ is acyclic and connected, and therefore is a tree with $M - 1$ edges. □

A.2 Exact inverse

Proof. The components C_i partition V. For each C_i, template τ_i stores one atom state for every $v \in C_i$ and one bond state for every edge whose endpoints both lie in C_i. Every remaining edge is a bridge. Its tree-edge record stores the two component indices, one port slot per endpoint, and the bridge-bond state; each port slot stores the component-local endpoint atom. The decoder first creates disjoint copies of all template atoms and internal bonds, then adds exactly one bridge for each tree-edge record between the atoms named by its ports. Thus every source vertex and edge is created exactly once, and no additional vertex or edge is introduced. The source-to-decoded local index maps define a feature-preserving graph isomorphism.

This argument applies to connected inputs after non-isomeric canonicalization when all atom states, bond states, and port-local indices belong to the declared representation. Stereochemical and isotopic identity is not stored. The implementation uses RDKit 2025.09.2; different RDKit versions can assign different component identities and are therefore outside the reproducibility claim. □

A.3 Time complexity and computational efficiency

We analyze inference for a fixed component vocabulary and resource map. Integer coordinates are assumed to fit in machine words, and table access is constant-time after constructing the key. The cost of constructing a state key is included in the vector-processing cost below.

DP compilation. Let S count the structurally admissible DP states evaluated, including one additional state for root selection, and let $G_{\max} \geq 1$ bound the number of groups examined at any state. States rejected by an immediate structural check are charged to the transition that produces them.

Each transition updates four port counts and d resource counts. For $d \geq 1$, this takes $O(d)$ time, including compatibility checks and construction of the successor key. Memoization evaluates each state once, and each evaluation examines at most $G_{\max}$ groups. The total DP time is therefore

$$T_{\mathrm{DP}} = O(dSG_{\max}).$$

This bound includes states first evaluated during sampling if compilation is performed lazily. Such evaluations contribute to S rather than being charged as a new search at every generation step.

To bound the state count, consider a partial tree with k components left to add. Each currently open port must eventually connect to a new component. Since one addition connects a new component to only one existing port, a structurally feasible state must satisfy

$$\sum_{\ell=1}^{4} f_\ell \leq k.$$

There are $\binom{k+4}{4}$ non-negative four-dimensional vectors satisfying this inequality. Each residual resource coordinate also satisfies $0 \le b_j \le b_j^\star$. Consequently,

$$S \le 1 + \sum_{k=0}^{M-1} \binom{k+4}{4} \prod_{j=1}^{d} (b_j^\star + 1)$$
$$= 1 + \binom{M+4}{5} \prod_{j=1}^{d} (b_j^\star + 1). \tag{5}$$

The leading one accounts for root selection. Thus, for fixed d, the bound is polynomial in M and the numerical target values, but not in their binary encoding lengths. The product over target coordinates also explains why increasing the number of jointly controlled counts can be expensive.

This is a worst-case bound, not the number of states necessarily visited. The actual computation follows reachable transitions and uses the pruning rules in Appendix A.8. Grouping reduces repeated work across actions with the same effect, while memoization reduces repeated work across construction histories that reach the same state. Neither guarantees a small state space for every target.

Generation with cached completion checks. At generation step t, let $C_\theta(t)$ be the cost of computing the candidate probabilities used by the sampler, including the prediction heads. Let A_t be the number of concrete candidate scores processed at that step.

Once the feasibility results are cached and indexed, checking one candidate takes constant time. Constructing the mask, normalizing the retained scores, and sampling therefore take $O(A_t)$ time in total, not $O(1)$. Updating the selected DP state adds $O(d)$ work. After M additions, reconstructing the atom–bond graph takes $O(|V| + |E|)$ time. Hence, the decoding cost is bounded by

$$T_{\mathrm{dec}}(M) = O\left(\sum_{t=1}^{M} \big(C_\theta(t) + A_t + d \big) + |V| + |E| \right).$$

The bound covers graph construction, not chemical sanitization or benchmark evaluation.

For a fixed vocabulary, component sizes are bounded, so $|V| + |E| = O(M)$. For fixed d, state updates also take $O(M)$ time across the trajectory. These terms can therefore be absorbed into the candidate-processing cost, since at least one candidate is processed at every step. The substantial decoding costs are model evaluation and handling its candidate scores.

Importantly, $G_{\max}$ and A_t describe different computations. The former bounds the grouped Boolean feasibility checks; the latter counts the exact choices processed by the sampler. Sharing a feasibility result does not remove the cost of scoring different components and attachment sites. We also retain $C_\theta(t)$ explicitly because model evaluation may depend on the growing prefix. Having M generation steps does not, by itself, imply $O(M)$ total neural computation.

Preprocessing and total cost. Let T_{pre} include construction of the grouped vocabulary index and auxiliary pruning tables when these are not already available. These structures can be reused while the vocabulary, count definitions, and supported component horizon remain unchanged.

Given the template annotations, forming the grouped index takes $O(d|\mathcal{V}|)$ time with hashing. The auxiliary table costs follow from their recurrences. Let H be the number of distinct bond-type vectors, U the set of distinct resource vectors, and let P_f and P_b count the frontier and resource-sum entries expanded across all table levels. A recurrence-based construction takes

$$T_{\mathrm{pre}} = O(d|\mathcal{V}| + HP_f + d|U|P_b).$$

Each frontier entry is extended through the available bond-type vectors, while each resource-sum entry is extended through U. Optional table limits can reduce this preprocessing work; disabling a pruning table does not change the exact joint recurrence.

For B molecules sharing a target, $T_{\mathrm{dec}}(M)$ upper-bounds each decoding trajectory:

$$T_{\mathrm{total}}(B) = T_{\mathrm{pre}} + O(dSG_{\max} + B\,T_{\mathrm{dec}}(M)).$$

The main-text expression assumes the reusable preprocessing is already available. If it must be repeated for a new count definition or a larger supported horizon, its cost must be included.

When the user does not specify M, we compile the candidate component counts separately. Writing S_m for the states evaluated for component count m, a conservative bound on the total DP work is

$$O\left(dG_{\max}\sum_{m=1}^{M_{\max}} S_m\right).$$

Generation then pays the decoding cost for each selected component count. This bound does not assume additional sharing between the fixed-M compilations.

Which costs dominate. For a new target with few requested samples, compilation can dominate. Its cost depends on how many distinct states and transitions are explored, and the state-space bound grows multiplicatively with the resource ranges. Low resource dimension helps, but does not alone guarantee a cheap target.

For repeated sampling, the DP computation is shared across outputs. Its amortized contribution is $O(dSG_{\max}/B)$, while the decoding cost must be paid for every molecule. With lazy compilation, S may increase as new states are encountered, but previously solved states are not recomputed. Once the relevant cache is populated, further generation requires model prediction and candidate processing rather than another search through possible completions.

This differs from count-based rejection sampling. If independent unconstrained attempts satisfy a requested target with probability $p_{\text{hit}} > 0$, rejection requires $1/p_{\text{hit}}$ attempts on average for one matching output. TopoCompile avoids this repetition due to count mismatches, provided compilation succeeds and feasible choices have positive model probability. Its advantage is therefore reusable exact completion checking, not a guarantee of faster generation for every target. Expensive compilation with little reuse can outweigh the savings from avoiding rejection.

Relation to measured efficiency. The reported MOSES scaling experiments illustrate both costs. First compilation takes 0.034 seconds in the one-count configuration and 19.011 seconds in the three-count configuration. Their generation throughputs are respectively 7.12% and 28.96% below the corresponding port-only decoder. These results show that compilation latency and the remaining sampling overhead must be considered separately. Figure 4(c) assigns sampler/compiler setup, target compilation, reachability, and cached parent-option checks to DP. The Model category combines model execution, reconstruction, and final exactness checks; Other overhead contains the remaining action/state processing and delivery bookkeeping. Cache reuse reduces DP time from 0.515 to 0.181 seconds on MOSES and from 0.670 to 0.261 seconds on ZINC250K, corresponding to $2.85\times$ and $2.57\times$ speedups. Total request time falls from 0.671 to 0.337 seconds and from 0.831 to 0.421 seconds, respectively. The measured cold/cache Other overhead estimates differ by less than 1% within each dataset, so the figure uses their request-weighted pooled value and moves the compensating difference to DP, preserving every measured total.

A.4 Representation implementation and verification

The implementation first applies the pinned non-isomeric RDKit canonicalization and serializes atom states, internal bond states, bridge-bond states, component-local endpoint indices, and port symmetry classes. We independently recompute depth-first-search bridges (Tarjan, 1974) and checks agreement with the RDKit-derived component partition, the bridge–port bijection, and reconstruction of all stored atom and bond attributes. Thus the validation does not rely on one ring-perception routine. Disconnected inputs and bond states outside the representation are excluded.

A.5 Additivity of cycle rank

Proof. The components are atom-disjoint, and assembling their bridge tree adds exactly $M-1$ inter-component edges. Hence $|V| = \sum_i |V_i|$ and $|E| = \sum_i |E_i| + M - 1$, which gives

$$\beta(G) = |E| - |V| + 1 = \sum_i \big(|E_i| - |V_i| + 1\big) = \sum_i \beta(\tau_i). \tag{6}$$

Thus bridge attachment creates no cross-component cycle, making cycle rank component-additive. □

A.6 Structural frontier reachability

Let $\boldsymbol{\sigma}(\tau) \in \mathbb{N}^4$ count the attachment ports of each bridge-bond type on template τ. Starting from $\mathcal{R}_0 = \{\mathbf{0}\}$, define

$$\mathbf{f} \in \mathcal{R}_{r+1} \iff \exists\, \ell, \tau :\ f_\ell, \sigma_\ell(\tau) \geq 1, \quad \mathbf{f}' = \mathbf{f} + \boldsymbol{\sigma}(\tau) - 2\mathbf{e}_\ell \in \mathcal{R}_r. \tag{7}$$

Lemma 1 (Frontier reachability)**.** $\mathbf{f} \in \mathcal{R}_r$ *if and only if exactly* r *legal child actions can consume* $\mathbf{f}$ *and terminate at the empty frontier.*

Proof. We prove the lemma by induction on r. For $r = 0$, no child action is available and the frontier can finish if and only if it is $\mathbf{0}$. Assume the result for r. If $\mathbf{f} \in \mathcal{R}_{r+1}$, the recurrence supplies a type ℓ and template τ whose attachment reaches $\mathbf{f}' \in \mathcal{R}_r$; the inductive hypothesis supplies the remaining r actions. Conversely, any legal first child action that admits an r-action completion reaches such a $\mathbf{f}'$, which the inductive hypothesis places in $\mathcal{R}_r$. The recurrence therefore places $\mathbf{f}$ in $\mathcal{R}_{r+1}$. □

A.7 Joint feasibility DP: soundness and relative completeness

Fix a finite vocabulary $\mathcal{V}$, the typed-port transition system in Equation 2, and an admissible map $r : \mathcal{V} \to \mathbb{N}^d$. Equation 3 defines $F_{\mathbf{c}^\star}$ by its terminal condition and Boolean recurrence.

Soundness. For soundness, induct on k. The $k = 0$ state accepts only $(\mathbf{0}, \mathbf{0})$. For $k > 0$, every retained action subtracts exactly $r(\tau)$ and takes the state to a reachable $(k-1)$ suffix. The induction hypothesis closes that suffix with zero frontier and resource residual. Telescoping the transitions therefore consumes the initial frontier and sums the selected component contributions to the initial $\mathbf{b}$. Applying the root check with $\mathbf{b} = \mathbf{b}^\star - r(\tau_1)$ proves exact target equality and structural completion for every sequence accepted by the joint DP. □

Relative completeness. For relative completeness, consider a feasible k-component suffix in the grammar. Its first component and attachment induce a legal non-negative successor, whose remaining $k-1$ components form a feasible suffix. Induction gives $F_{\mathbf{c}^\star} = 1$ for that successor, so the recurrence retains the first action; repetition retains the whole sequence. The result is relative to the finite catalog and transition language. It assumes that every admissible resource is component-local, non-negative and integer-valued, and unchanged by bridge attachment. It neither creates support for unseen component interiors nor guarantees sanitization, synthesis, or learned probability mass. We report infeasible catalog targets before sampling. □

A.8 Joint feasibility DP: full algorithm

For implementation, let Γ_{comp} collect the distinct $(\boldsymbol{\sigma}(\tau), r(\tau))$ pairs, Γ_ℓ those with an available type-ℓ port, and $\mathcal{U}$ the unique resource vectors used below:

$$\mathcal{B}_0 = \{\mathbf{0}\}, \qquad \mathcal{B}_k = \{\mathbf{z} + \mathbf{u} : \mathbf{z} \in \mathcal{B}_{k-1},\ \mathbf{u} \in \mathcal{U}\}. \tag{8}$$

Algorithm 1 gives the complete execution order for terminal checks, structural and resource pruning, compatible-group enumeration, and memoization. If the auxiliary exact-sum cache reaches its cap, $\mathrm{SumReach}$ conservatively returns TRUE.

Algorithm 1: Joint feasibility DP with safe pruning

Data: target $(M, \mathbf{b}^\star)$; catalog index $\mathcal{I} = (\Gamma_{\mathrm{tok}}, \{\Gamma_\ell\}_{\ell=1}^4, \mathbf{r}_{\min}, \mathbf{r}_{\max})$; structural tables $\{\mathcal{R}_k\}_{k=0}^{M-1}$
Result: feasible root groups Γ_{root}

```
memo ← ∅
Function CanFinish(k, f, b):
    z ← (k, f, b)
    if z ∈ memo then
        return memo[z]
    if k = 0 then
        memo[z] ← [f = 0 ∧ b = 0]
        return memo[z]
    if k < 0 ∨ f ∉ R_k then
        memo[z] ← FALSE; return FALSE
    if k r_min ⋠ b ∨ b ⋠ k r_max then
        memo[z] ← FALSE; return FALSE
    if ¬ SumReach(k, b) then
        memo[z] ← FALSE; return FALSE
    memo[z] ← FALSE
    foreach ℓ such that f_ℓ > 0 do
        foreach γ ∈ Γ_ℓ do
            b' ← b − r_γ
            if b' ⋡ 0 then
                continue
            f' ← f + σ_γ − 2e_ℓ
            if f' ⋡ 0 ∨ f' ∉ R_{k−1} then
                continue
            if CanFinish(k − 1, f', b') then
                memo[z] ← TRUE; return TRUE
    return FALSE
Γ_root ← ∅
foreach γ ∈ Γ_tok do
    b' ← b* − r_γ
    if b' ⪰ 0 ∧ σ_γ ∈ R_{M−1} ∧ CanFinish(M − 1, σ_γ, b') then
        Γ_root ← Γ_root ∪ {γ}
return Γ_root
```

If the auxiliary cache reaches 1,000,000 states at any level, the implementation disables only this optional pruning check and continues with the exact joint DP; this can increase computation but does not remove feasible actions. Reaching the 2,000,000-state limit of the joint DP instead raises an explicit compilation error rather than returning an incomplete feasible set. Let $\mathcal{F}_k$ be the possible frontier vectors with k components remaining. The target-conditioned state count satisfies

$$S \le \sum_{k=0}^{M} |\mathcal{F}_k| \prod_{j=1}^{d} (b_j^\star + 1), \tag{9}$$

so the joint DP uses $O(SG_{\max}d)$ worst-case time and $O(S)$ memo memory, where $G_{\max}$ is the maximum number of compatible groups examined at one state. Its dependence on target magnitudes and resource dimension is pseudo-polynomial.

B Additional experimental settings

B.1 Datasets and preprocessing

We evaluate TopoCompile on MOSES (Polykovskiy et al., 2020), ZINC250K, GuacaMol (Brown et al., 2019), and NPGen (Lee et al., 2026). For TopoCompile, preprocessing uses RDKit 2025.09.2 and retains connected molecules within the supported non-isomeric atom–bond domain.

Stereochemical and isotopic identities are not represented. The component vocabulary is constructed from training molecules only, while representation coverage is evaluated separately on the test sets. Before selecting exact-control targets, we remove cross-split molecular overlaps. Molecular identity in the exact-control evaluation is defined by canonical non-isomeric connected SMILES, consistently with the training-exclusion set.

B.2 Model architecture and training

Architecture. The action Transformer has eight layers, eight attention heads, hidden size 256, and feed-forward size 1,024. Atom and bond embeddings have dimensions 256 and 128, respectively, and component encoding uses a five-layer fragment encoder. Four prediction heads model the bond-type vector, exact component, parent port, and child port.

Training. We use AdamW with coefficients $(0.9, 0.999)$, learning rate 5×10^{-4}, 10,000 warm-up updates, no weight decay, exponential moving average decay 0.999, and gradient clipping at 8. Each model is trained for 30,000 updates with an effective batch size of 256 and eight data workers. All local training runs reported in this paper use a single Tesla H800 PCIe GPU, and all experiments report averages over three runs with seeds 123, 456, and 789.

Each randomized rooted traversal produces a ground-truth action sequence. Under teacher forcing, the model receives the ground-truth partial tree and predicts the corresponding next action. Parent candidates are unused ports in the partial tree, component candidates are restricted to the selected bond-type vector, and child ports must match the parent bond type. The loss weights for the bond vector, component, parent port, and child port are $(1, 5, 1, 1)$; only bond-vector and component losses apply at the root. Structural reachability masks are used during both training and sampling.

Unconditional and constrained generation. For unconditional generation, TopoCompile first samples the component count M from its empirical distribution over the training molecules and then generates a molecule autoregressively conditioned on M. Exact constrained generation reuses the same checkpoint and applies the joint DP during sampling, without retraining the generator.

B.3 Baseline implementations

Graph DiT. We use the released conditional diffusion checkpoint of Graph DiT (Liu et al., 2024) and provide the active resource counts as conditioning inputs for the primary exact-control comparison. When a request specifies heavy-atom count N, we use the requested node count to constrain graph size; otherwise, we retain the model's learned node-count distribution. We preserve the released connection repair, valence repair, and largest-component handling before converting the output to a molecule. The resulting molecule is assessed by the shared external validity and exact-count evaluator. Each target uses the fixed attempt budget, including generation, repair, and count-mismatch failures. Graph DiT receives requested counts without stepwise completion checks.

STGG+ and STGG+ Count Mask. We use the released property-conditional spanning-tree generator STGG+ (Jolicoeur-Martineau et al., 2025), conditioning on the active counts in the primary comparison. Unspecified properties are handled by its native missing-property mask, and its structural and valence masks are retained. Guidance strength is selected on separate validation targets. Each attempt uses one internal candidate, without best-of-k selection or additional resampling.

STGG+ Count Mask uses the same checkpoint, conditions, guidance, and sampling budget, adding only a process-local, token-level count mask around the decoder. After each accepted token, the mask updates the accumulated heavy-atom count, oxygen count, and cycle rank. Candidate tokens that would exceed any active target count are masked, and EOS is allowed only when these three counts match their targets. This mask is combined with the native structural and valence masks; it performs local count control without checking future completion feasibility. An empty action set terminates the attempt as a failure. Both variants use the shared external evaluator.

FragFM+Guidance. We use the released FragFM molecular generator (Lee et al., 2026) with a scalar property predictor for each active resource count. For a requested target, predictor deviations are normalized using validation-set scales, and the resulting prediction energies are combined

to guide both fragment-bag generation and molecular graph generation. Bag and graph guidance strengths are selected separately for each dataset using validation targets. We retain the official fragment reconstruction procedure without prescribing a component count, tree layout, or exact completion check. The shared external validity and exact-count evaluator then assesses each molecule.

GenMol+Filter. We use the GenMol V2 unconditional de novo checkpoint (Lee et al., 2025) without supplying target counts to the model. For each target, GenMol independently proposes molecules, and the shared validity and count checks determine which proposals are exact-valid. There is no retraining, guidance, or target-specific decoding. Each proposal counts as one attempt, including invalid outputs and filter rejections; we do not generate replacement proposals to fill a quota of valid or exact outputs. This implements generation followed by filtering under a fixed proposal budget.

Tree-Invent. For unconditional generation, we use the paper-designated RBFS traversal of Tree-Invent (Xu & Chen, 2023). The GuacaMol row is taken from the original paper's 50,000-sample evaluation, with V.U. and V.U.N. derived from its reported validity, uniqueness, and novelty. For MOSES, ZINC250K, and NPGen, we train the released implementation from scratch for up to six passes over a deterministic 95% partition of the training set, reserving the remaining 5% for validation. Batch sizes are 320, 144, and 64, respectively, instead of the released default of 1,000. Each run has an eight-hour training limit and a four-hour generation-and-evaluation limit. ZINC250K and NPGen complete, whereas MOSES fails before the first optimizer update and is reported as OOM. Unconditional-generation tables distinguish local reproductions from published results.

Other unconditional-generation baselines. The unconditional-generation tables also include published results for DiGress (Vignac et al., 2023), DeFoG (Qin et al., 2025), GraphAF (Shi et al., 2020), and other cited generators. These entries retain their source attributions and are distinguished from our local Tree-Invent reproductions.

Guidance selection. We select configurations on 32 fixed validation triples disjoint from the final evaluation targets, using 32 attempts per target. Selection maximizes exact-valid yield, with validity and exact-output diversity as tie breakers. For STGG+, we search guidance strengths $\{1.0, 1.2, 1.5, 2.0\}$ at temperature 1. The selected strength is 1.5 on MOSES and GuacaMol and 2.0 on ZINC250K and NPGen; Count Mask uses the same setting. For FragFM+Guidance, we search bag/graph strength pairs $(0, 2)$, $(0.2, 2)$, $(0.4, 2)$, and $(1, 20)$. Dataset-specific selection yields $(1, 20)$ on all four datasets. No final evaluation target is used to tune these configurations.

Shared comparison protocol. All exact-control methods use the same Common/Rare target lists, main attempt budgets, external validity and exact-count evaluator, and denominator rules. Each variant retains its native generation mechanism with only the control adaptations described above. The comparison therefore measures exact-valid yield under shared requests and sampling budgets rather than imposing the same internal architecture or constraint mechanism on every method.

B.4 Evaluation protocols

Metric definitions. For fixed attempts, Valid is the fraction of chemically valid molecules, and Exact-valid additionally requires every requested count. For an exact-output pool, Novelty is the fraction of unique canonical identities absent from training; internal diversity is one minus mean pairwise Tanimoto similarity of radius-2, 1024-bit Morgan fingerprints (Rogers & Hahn, 2010). Conditional FCD is the Fréchet ChemNet distance (Preuer et al., 2018) between matched exact outputs and target-matched references. For unconditional generation, Validity is the valid fraction under each benchmark denominator, and V.U.N. is validity $\times$ uniqueness $\times$ novelty. FCD compares ChemNet embeddings of valid outputs with the benchmark reference set; GuacaMol reports the corresponding score, where higher is better.

Unconditional-generation budgets and replication. MOSES and ZINC250K each use 25,000 generation attempts, and NPGen uses 30,000. GuacaMol uses an 11,000-molecule pool to construct its 10,000-molecule report. TopoCompile results on MOSES, ZINC250K, GuacaMol, and NPGen average three independently trained models. Each table or caption states its repetition unit.

Benchmark-specific evaluation. Validity on MOSES, ZINC250K, and NPGen uses all attempts. GuacaMol validity uses the first 10,000 attempts; its FCD uses the first 10,000 pooled-valid molecules, while KL, novelty, and V.U.N. use the first 10,000 pooled unique molecules. Other FCD and ring metrics use valid canonical molecules. MOSES FCD/TestSF uses the 176,225-molecule scaffold test split, and GuacaMol uses its training reference and published score definition. ZINC250K uses the final 5,000 test molecules with a local implementation of the benchmark evaluator. NPGen uses its released test embedding.

Exact-control requests and yields. The primary comparison evaluates $(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$ without supplying M; lower-dimensional schemas are reported in the appendix. TOPOCOMPILE checks supported candidate component counts, retains those admitting a compiler-feasible completion, and samples from the restricted empirical prior $p(M \mid N)$ when N is given, or $p(M)$ otherwise. Exact-control experiments use frozen checkpoints and fixed sampling streams, with $K = 32$ attempts per target unless stated otherwise.

Exact-valid yield (Exact) is the fraction of attempts producing valid molecules that satisfy all requested counts. Novel unique exact yield (NUE) counts the exact-valid canonical molecules that are absent from training and deduplicated within each target and sampling run, using the same attempt denominator. We average target yields by schema and tier, then macro-average schemas.

Matched-output quality. For the main exact-control quality comparison, we use preselected Common targets and the first 100 attempts per target. Compared methods contribute equal-sized exact-output pools for each target, with survivors selected in attempt order. A target is excluded when any compared method provides fewer than two exact outputs. Novelty is the fraction of unique matched identities absent from training. Internal diversity is one minus the mean pairwise Tanimoto similarity of radius-2, 1024-bit Morgan fingerprints. Graph DiT and GenMol+Filter are unreported where their exact outputs do not support a compatible matched pool.

Ring-based metrics. For a parsed molecular graph, cycle rank is $|E| - |V| + c$, where c is the number of connected components and equals one in the supported compiler domain. Ring systems follow the maximal non-bridge component definition used by the representation, and ring size is the multiset of RDKit atom-ring sizes. The topology signature comprises nonexclusive fused, spiro, and bridgehead indicators: fused denotes a bond belonging to multiple perceived atom rings, while spiro and bridgehead indicators use RDKit's canonical counters. Core identity is the port-free canonical component SMILES, and a core pair is an unordered pair of core identities within one molecule. For each of these six statistics, we compute $S_k = 1 - \mathrm{JSD}_{\mathrm{nats}}(P_k, Q_k)/\log 2$; RingSF is their mean.

NPGen proxy metrics. NPGen's NP-score and classifier outputs are learned proxies. Its released classifier KLD discards generated-only labels and renormalizes on reference support. We retain this definition for benchmark compatibility and also report KLD over the union support after probability normalization, together with generated-only mass. The reported benchmark proxy values reproduce within 10^{-6} using the released weights and source files.

B.5 REPRESENTATION AND COMPILER VERIFICATION

We verify the implementation on all 4,243,193 connected records retained after preprocessing the training, validation, and test splits of the four datasets. Independent Tarjan bridge detection (Tarjan, 1974) is checked against RDKit-derived ring grouping. The checks cover component-node types, component-tree connectivity and edge count, cycle localization, the bridge–port bijection, template identity, feature-preserving graph reconstruction, and catalog consistency. No violations are found. Testing 4,096 atom-order permutations confirms preservation of topology, ports, and reconstruction.

The joint DP is independently checked against exhaustive enumeration on five small vocabularies for each resource dimension $d \in \{1, 2, 3\}$. These comparisons remove no feasible completions and retain no infeasible completions. Independent recounting finds no count mismatch in 10,000 MOSES and 10,000 ZINC250K molecules.

C EXACT-CONTROL SETTINGS AND SCHEMA-SPECIFIC RESULTS

This section specifies how we construct exact-control requests, configure the compared methods, and compute the reported success and quality metrics. It then presents the schema-specific results and examines how exact-control performance changes with target frequency.

Requests and target groups. Each request specifies a subset of heavy-atom count N, cycle rank β, oxygen count N_{O}, and ring-system count R_{sys}, but does not specify the component count M. TopoCompile therefore enumerates candidate values of M, retains those with a compiler-feasible completion, and samples from $p(M \mid N)$ when the request specifies N or from $p(M)$ otherwise. We remove cross-split molecular overlaps before selecting targets, so the resulting Common and Rare groups preserve their observed target identities without sharing molecules across splits. For the frequency analysis, we assign equal-frequency tuples their average rank among all distinct targets. Figure 2(A) divides these ranks into four percentile ranges and selects eight four-count tuples, spaced uniformly by rank, from each range.

Baselines and configuration selection. We compare TopoCompile with Graph DiT, STGG+, FragFM+Guidance, GenMol+Filter, and STGG+ Count Mask under a shared external count check. Graph DiT forces the requested node count only for requests that specify N and retains its released connection/valence repair and largest-component handling. STGG+ retains its missing-property and structural/valence masks, whereas FragFM+Guidance applies normalized predictor energies for the active properties during bag and graph generation. GenMol+Filter instead generates unconditional candidates and filters them independently. Finally, STGG+ Count Mask reuses the STGG+ checkpoint and blocks count overshoots and premature EOS; an exhausted action set terminates as a failed attempt. Because guidance strength directly affects these comparisons, we select STGG+ guidance from $\{1.0, 1.2, 1.5, 2.0\}$ at temperature 1 and FragFM bag/graph strengths from $(0, 2)$, $(0.2, 2)$, $(0.4, 2)$, and $(1, 20)$ on separate validation targets. Selection maximizes exact-valid yield at a fixed candidate budget, using validity and exact-output diversity as tie breakers. We select one configuration per dataset, and STGG+ uses one internal candidate per attempt. This procedure selects STGG+ guidance 1.5 for MOSES and GuacaMol and 2.0 for ZINC250K and NPGen; STGG+ Count Mask uses the corresponding STGG+ setting. FragFM uses bag/graph strengths 1/20 on every dataset, selected using 32 attempts on each of 32 fixed validation triples disjoint from the final requests.

Metrics, denominators, and aggregation. We first evaluate every generated molecule with the same external validity and count checks. Valid yield is the fraction of attempts that produce a valid molecule, while Exact-valid yield is the fraction that are both valid and exact; both metrics use the full attempt count as their denominator. Consequently, an invalid output remains in the denominator, whereas a request that cannot finish compilation produces no sampled candidates; we record it separately as request noncompletion. We also distinguish state-limit exhaustion, timeout, and certified infeasibility. Molecular identity throughout is canonical non-isomeric connected SMILES, consistent with the training exclusion set. After computing the joint events per output, we average targets within each schema and macro-average schemas where reported. For matched quality, we use the same preselected Common targets and the first 100 attempts per target. Every method contributes the same number of exact survivors in attempt order, and we exclude a target when any compared method supplies fewer than two. Novelty is then the fraction of unique matched identities absent from training, and internal diversity is one minus mean pairwise Tanimoto similarity using radius-2, 1024-bit Morgan fingerprints. Graph DiT and GenMol+Filter remain unreported when their exact-output counts do not support a compatible pool.

All selected TopoCompile requests entered sampling and completed the full evaluation pipeline. Tables S2a–S2c provide lower-dimensional complements to the main four-count comparison for three schemas with near-complete comparison grids. The sparser N_{O}, (N, β), and (N, N_{O}) grids are not repeated as standalone tables. All four tables use the fixed sampling protocol. Panel A reports Valid and Exact-valid in percent for the first $K = 32$ Common and Rare attempts. In Tables S2a– S2c, Panel B reports Novelty and internal diversity from matched exact outputs among the first 100 Common attempts; Table S2d uses the same budget. For the four-count table, the MOSES/ZINC250K/GuacaMol/NPGen comparable-target and matched-sample counts are 32/690, 32/374, 32/172, and 15/45, respectively. In MOSES/ZINC250K/GuacaMol/NPGen order, dashes

denote missing entries and the comparable target/sample counts are 6/122, 9/104, 20/127, and 23/127 for N; 5/434, 5/396, 5/257, and 7/345 for β; and 18/1,179, 21/1,177, 32/840, and 32/429 for (β, N_{O}).

Table S1a: Numbers of shared Common and Rare targets by dataset and constraint schema.

Dataset	N	β	N_{O}	(N, β)	(N, N_{O})	(β, N_{O})	$(N, \beta, N_{\mathrm{O}})$
MOSES	6/6	5/4	6/6	26/26	32/32	18/18	32/32
ZINC250K	9/9	5/4	6/5	32/32	32/32	21/22	32/32
GuacaMol	21/21	5/5	8/8	32/32	32/32	32/32	32/32
NPGen	25/25	7/7	13/13	32/32	32/32	32/32	32/32

Table S2a: Valid and E.V (Exact-valid) yields (%) for heavy-atom count N.

Method	**MOSES**				**ZINC250K**				**GuacaMol**				**NPGen**			
	Common		*Rare*		*Common*		*Rare*		*Common*		*Rare*		*Common*		*Rare*	
	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V
Graph DiT	67.19	67.19	31.77	31.77	61.81	61.81	46.18	46.18	12.80	12.80	8.48	8.48	44.63	44.63	35.88	35.88
STGG+	100.00	93.06	100.00	21.01	100.00	92.71	100.00	30.90	100.00	77.98	100.00	7.14	100.00	61.63	99.88	10.88
FragFM+Guidance	97.92	26.82	40.89	4.43	100.00	15.97	97.22	1.74	93.30	5.51	91.67	0.15	98.25	5.13	92.50	0.13
GenMol+Filter	100.00	16.67	100.00	0.00	100.00	11.11	100.00	0.00	100.00	5.80	100.00	0.00	100.00	3.88	100.00	0.00
STGG+ Count Mask	96.35	96.35	24.48	24.48	93.40	93.40	46.53	46.53	83.18	83.18	25.15	25.15	70.00	70.00	27.13	27.13
TopoCompile (ours)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**

B. Matched exact-output novelty and diversity

Method	**MOSES**		**ZINC250K**		**GuacaMol**		**NPGen**	
	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑
STGG+	0.910	0.854	0.990	0.870	0.984	0.876	0.992	0.882
FragFM+Guidance	0.893	0.863	0.981	0.870	0.937	0.876	0.866	0.886
STGG+ Count Mask	0.910	0.854	1.000	0.872	0.984	0.877	0.992	0.883
TopoCompile (ours)	**0.951**	**0.864**	**0.990**	**0.868**	**0.992**	**0.878**	**0.984**	**0.889**

Table S2b: Valid and E.V (Exact-valid) yields (%) for cycle rank β.

Method	**MOSES**				**ZINC250K**				**GuacaMol**				**NPGen**			
	Common		*Rare*		*Common*		*Rare*		*Common*		*Rare*		*Common*		*Rare*	
	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V
Graph DiT	54.17	41.25	40.62	5.21	45.00	29.58	35.42	10.68	23.12	5.63	69.38	1.25	41.52	9.38	15.62	0.00
STGG+	100.00	94.38	100.00	32.55	100.00	95.00	100.00	65.63	100.00	98.75	100.00	4.38	100.00	95.54	–	–
FragFM+Guidance	98.75	88.44	65.23	15.23	100.00	86.25	94.53	26.56	90.00	49.38	71.25	2.50	96.88	53.57	41.07	0.45
GenMol+Filter	100.00	18.75	100.00	0.00	100.00	16.88	100.00	0.00	100.00	20.00	100.00	0.00	100.00	14.29	100.00	0.00
STGG+ Count Mask	96.25	95.63	33.59	32.81	95.00	95.00	71.88	67.97	98.75	98.75	23.75	16.25	95.54	95.54	41.07	27.23
TopoCompile (ours)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**

B. Matched exact-output novelty and diversity

Method	**MOSES**		**ZINC250K**		**GuacaMol**		**NPGen**	
	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑
STGG+	0.938	0.871	1.000	0.879	0.996	0.878	0.855	0.882
FragFM+Guidance	0.896	0.864	0.995	0.872	0.957	0.881	0.913	0.884
STGG+ Count Mask	0.931	0.871	0.997	0.880	0.996	0.878	0.855	0.882
TopoCompile (ours)	**0.975**	**0.868**	**0.995**	**0.878**	**0.996**	**0.884**	**0.977**	**0.896**

Table S2c: Valid and E.V (Exact-valid) yields (%) for cycle rank and oxygen count (β, N_{O}).

Method	**MOSES**				**ZINC250K**				**GuacaMol**				**NPGen**			
	Common		*Rare*		*Common*		*Rare*		*Common*		*Rare*		*Common*		*Rare*	
	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V
Graph DiT	57.52	36.69	53.24	1.27	53.32	24.31	61.93	3.22	21.00	1.37	35.35	0.00	35.16	2.25	35.84	0.00
STGG+	100.00	97.51	100.00	9.14	100.00	96.43	100.00	14.77	100.00	96.09	100.00	0.68	100.00	89.55	92.09	1.17
FragFM+Guidance	99.74	66.15	75.78	5.90	100.00	55.80	91.48	5.68	91.11	26.27	70.51	0.10	98.14	14.45	66.11	0.00
GenMol+Filter	100.00	5.03	100.00	0.00	100.00	5.36	100.00	0.00	100.00	3.13	100.00	0.00	100.00	1.37	100.00	0.00
STGG+ Count Mask	97.40	97.22	44.79	39.06	97.47	97.32	49.57	47.16	97.07	96.88	17.68	14.45	93.75	93.46	62.50	55.57
TopoCompile (ours)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	–	–

B. Matched exact-output novelty and diversity

Method	**MOSES**		**ZINC250K**		**GuacaMol**		**NPGen**	
	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑
STGG+	0.933	0.865	0.997	0.878	0.992	0.882	0.944	0.887
FragFM+Guidance	0.922	0.860	0.996	0.873	0.989	0.878	0.911	0.880
STGG+ Count Mask	0.932	0.865	0.998	0.881	0.992	0.882	0.944	0.887
TopoCompile (ours)	**0.990**	**0.871**	**1.000**	**0.883**	**0.998**	**0.883**	**0.995**	**0.894**

Table S2d: Valid and E.V (Exact-valid) yields (%) for the four-count request $(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$. Dashes in the STGG+ Count Mask row denote Valid values not retained by the compact export; its exact-valid values are complete.

Method	MOSES				ZINC250K				GuacaMol				NPGen			
	Common		*Rare*		*Common*		*Rare*		*Common*		*Rare*		*Common*		*Rare*	
	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V	Valid	E.V
Graph DiT	69.34	56.64	41.60	5.57	66.02	43.36	51.07	8.40	3.61	0.49	49.51	4.30	OOM			
STGG+	100.00	80.86	100.00	29.00	100.00	79.59	100.00	37.50	100.00	34.86	100.00	13.87	100.00	16.89	100.00	3.71
FragFM+Guidance	97.85	22.07	69.34	0.20	99.12	11.04	97.46	0.00	93.16	4.98	90.53	0.00	98.34	1.56	97.46	0.00
GenMol+Filter	100.00	1.86	100.00	0.00	100.00	1.46	100.00	0.00	100.00	1.66	99.90	0.00	100.00	0.00	100.00	0.00
STGG+ Count Mask	–	83.59	–	35.55	–	82.71	–	44.92	–	46.88	–	15.53	–	27.44	–	14.06
TopoCompile (ours)	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**

B. Matched exact-output novelty and diversity from the first 100 attempts

Method	MOSES		ZINC250K		GuacaMol		NPGen	
	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑	Novelty ↑	Div. ↑
STGG+	0.946	0.845	**1.000**	0.860	0.994	0.867	**0.978**	0.883
FragFM+Guidance	0.899	0.846	0.987	0.852	0.988	0.858	0.689	0.860
STGG+ Count Mask	0.952	0.847	**1.000**	0.861	0.994	0.869	**0.978**	0.883
TopoCompile (ours)	**0.990**	**0.862**	**1.000**	**0.872**	**1.000**	**0.882**	0.956	**0.900**

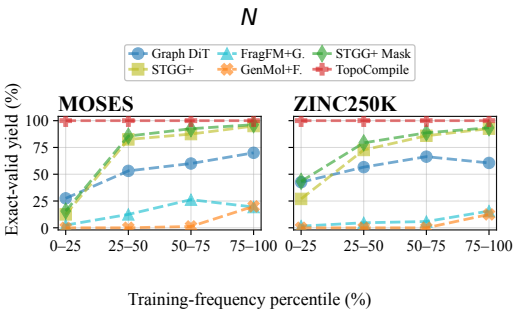


Figure S1: Exact-valid yield for the N schema on MOSES and ZINC250K.

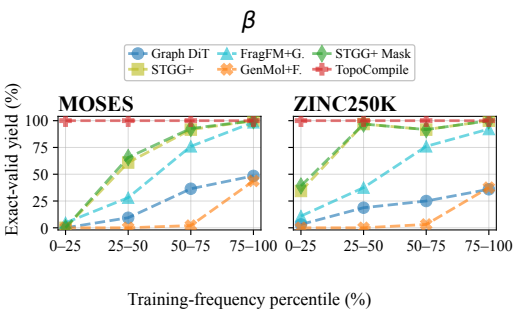


Figure S2: Exact-valid yield for the β schema on MOSES and ZINC250K.

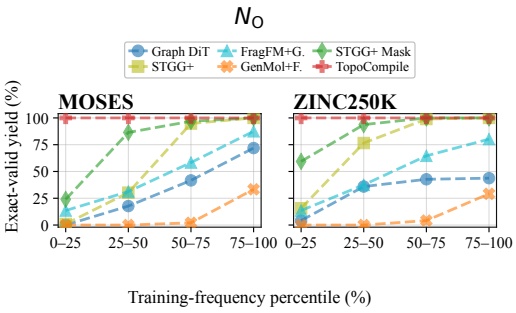


Figure S3: Exact-valid yield for the N_{O} schema on MOSES and ZINC250K.

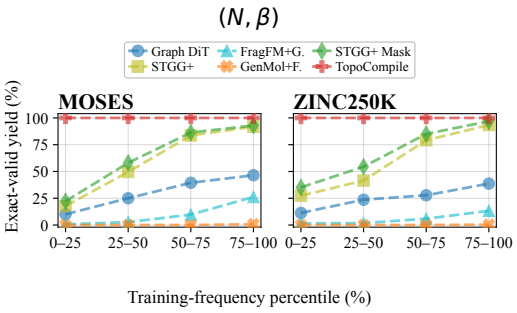


Figure S4: Exact-valid yield for the (N, β) schema on MOSES and ZINC250K.

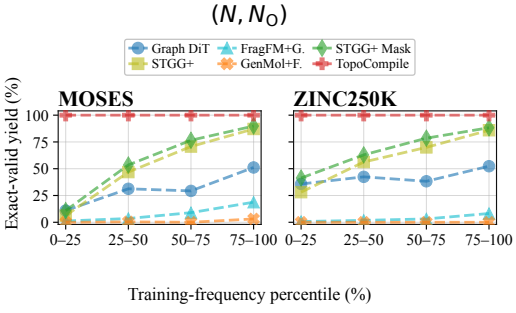


Figure S5: Exact-valid yield for the (N, N_{O}) schema on MOSES and ZINC250K.

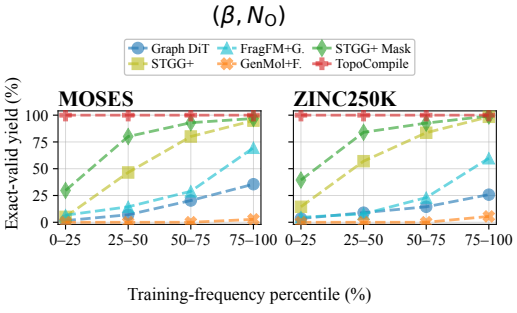


Figure S6: Exact-valid yield for the (β, N_{O}) schema on MOSES and ZINC250K.

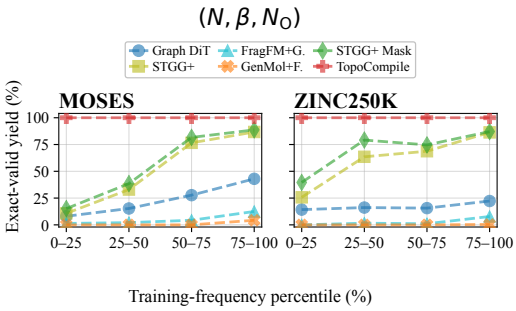


Figure S7: Exact-valid yield for the (N, β, N_O) schema on MOSES and ZINC250K.

Results analysis. Tables S2a–S2d show a consistent separation between molecular validity and exact delivery. TopoCompile attains 100.00% exact-valid yield in every reported dataset–tier cell for N, β, (β, N_O), and (N, β, N_O, R_{sys}), whereas baseline yields generally decrease from Common to Rare targets. Across the four available matched-quality panels, TopoCompile retains novelty of 0.951–1.000 and internal diversity of 0.864–0.896. The completion check preserves exactness across count vectors while the learned policy distinguishes feasible completions.

The schemas differ in how strongly they restrict baseline success and novel output multiplicity. Size-only control remains broadly accessible to the learned baselines, whereas Rare cycle-rank and joint cycle–oxygen requests produce sharper yield losses, especially on GuacaMol and NPGen. For the four-count Rare requests on NPGen, STGG+ remains 100.00% Valid but reaches 3.71% Exact-valid, while FragFM+Guidance remains 97.46% Valid and reaches 0.00% Exact-valid. This gap shows that validity alone does not establish constraint satisfaction. Figures S1–S7 provide a complementary frequency breakdown: baseline yields often increase from the lowest to the highest training-frequency ranges, while TopoCompile remains at 100% across the observed schema–frequency cells.

D Application: High-Rank Molecule Generation

Here, high rank refers to a large molecular cycle rank β. Generating molecules in this regime expands the tested topology beyond the frequencies commonly observed during training and requires multiple ring-forming decisions to satisfy one global structural request. TopoCompile supports this application because cycle rank is additive over components and the joint DP removes partial constructions that cannot complete to the requested β; the user does not need to specify molecular size N or component count M.

The experiments request only $\beta = 4, \ldots, 11$; no method receives N or M. They test whether each method continues to satisfy larger cycle-rank requests and whether its valid outputs become larger as β increases. All requests completed with 1,000 attempts per method–rank cell.

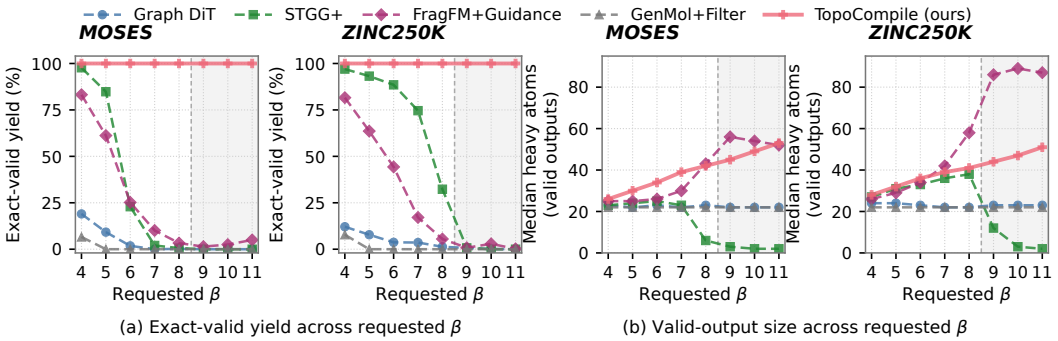


Figure S8: Exact-valid yield (a) and valid-output size (b) by cycle rank for MOSES and ZINC250K.

TopoCompile maintains 100.0% exact-valid yield across all eight requested cycle ranks in both datasets. The baseline yields fall sharply over the same range, with small nonmonotone fluctuations once they approach zero. Within the unobserved range $\beta = 9, 10, 11$, the highest baseline yield is 5.0% on MOSES and 2.9% on ZINC250K, both attained by FragFM+Guidance.

TopoCompile's median heavy-atom count increases from 26 to 53 on MOSES and from 28 to 51 on ZINC250K, while its median component count remains 10 and 11, respectively. Across these tested requests, higher cycle rank is therefore associated with larger TopoCompile outputs without an increase in their median component count. Smaller baseline medians at high β generally coincide with zero or near-zero exact-valid yield; because the size curves include all valid outputs, they do not indicate more compact exact solutions.

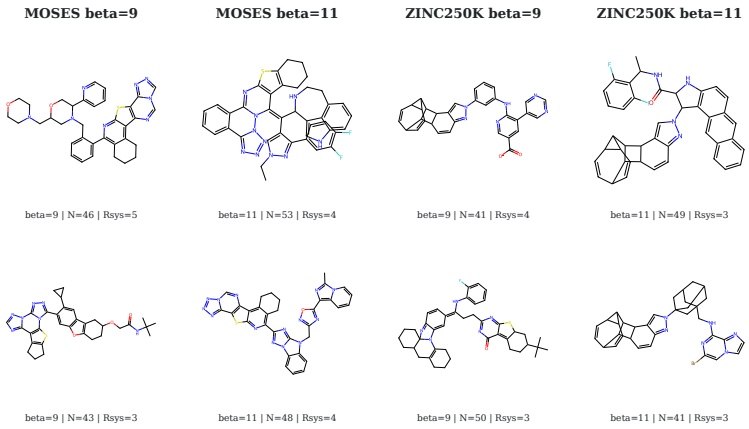


Figure S9: TopoCompile examples for unobserved high-cycle requests on MOSES and ZINC250K, with recomputed structural counts.

Figure S9 shows that the same high cycle-rank request admits visibly different component arrangements. Across the eight fixed examples, heavy-atom count ranges from 41 to 53 and ring-system count from 3 to 5 while every molecule realizes the requested $\beta \in \{9, 11\}$. These examples illustrate structural variation among exact high-rank outputs; they do not estimate the output distribution.

E Additional Results for Ablation Studies

E.1 Experimental Settings

We ablate three decisions: the completion check, the sampling policy over feasible actions, and the compiler's computational reductions. The completion- checking and policy studies use MOSES and ZINC250K with five schemas: (N, β), (N, N_{O}), (β, N_{O}), $(N, \beta, N_{\mathrm{O}})$, and $(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$.

For completion checking, we fix the checkpoint, vocabulary, requests, component count M, target identities, attempt order, Learned policy, and 32 attempts per target. Local enforces immediate structural and count consistency; Factorized adds exact count reachability independently of structural reachability; Joint requires one continuation to satisfy both. Exact-valid yield and novel unique exact yield (NUE) retain all attempts in their denominators, while NUE removes training identities and deduplicates within each target.

For the policy study, we fix Joint checking and compare Uniform, Frequency, and Learned over the first 100 attempts per target. Uniform assigns equal probability to legal values of each predicted action field, Frequency uses their training frequencies, and Learned uses the Transformer probabilities. FCD, internal diversity, scaffold diversity, Filters, and NSPDK use exact-output pools matched across policies by target and attempt order. The compiler study compares full compilation with grouping, optional-bound, count-sum, and structural-pruning ablations on fixed-M cases.

Each compiler variant has three fresh-process repeats; the table reports median time, states, and transitions, together with peak process memory and completed compilations.

Table S4a: Completion-checking ablation results for exact-valid yield and novel unique exact yield on MOSES and ZINC250K.

Dataset	Schema	Local	Factorized	Joint
		Exact / NUE (%)		
MOSES	(N, β)	41.35 / 39.90	72.66 / 70.43	**100.00 / 95.79**
	(N, N_O)	30.42 / 29.93	83.84 / 83.01	**100.00 / 99.12**
	(β, N_O)	56.16 / 55.56	83.25 / 82.20	**100.00 / 98.78**
	(N, β, N_O)	34.08 / 33.64	83.06 / 82.18	**100.00 / 98.83**
	(N, β, N_O, R_{sys})	15.04 / 14.84	71.68 / 69.82	**100.00 / 97.66**
ZINC250K	(N, β)	28.03 / 27.73	67.14 / 66.60	**100.00 / 98.78**
	(N, N_O)	30.27 / 30.13	82.67 / 82.18	**100.00 / 99.51**
	(β, N_O)	50.00 / 50.00	74.27 / 74.20	**100.00 / 99.93**
	(N, β, N_O)	23.68 / 23.29	86.23 / 84.62	**100.00 / 97.17**
	(N, β, N_O, R_{sys})	12.40 / 11.96	80.27 / 77.59	**100.00 / 93.51**

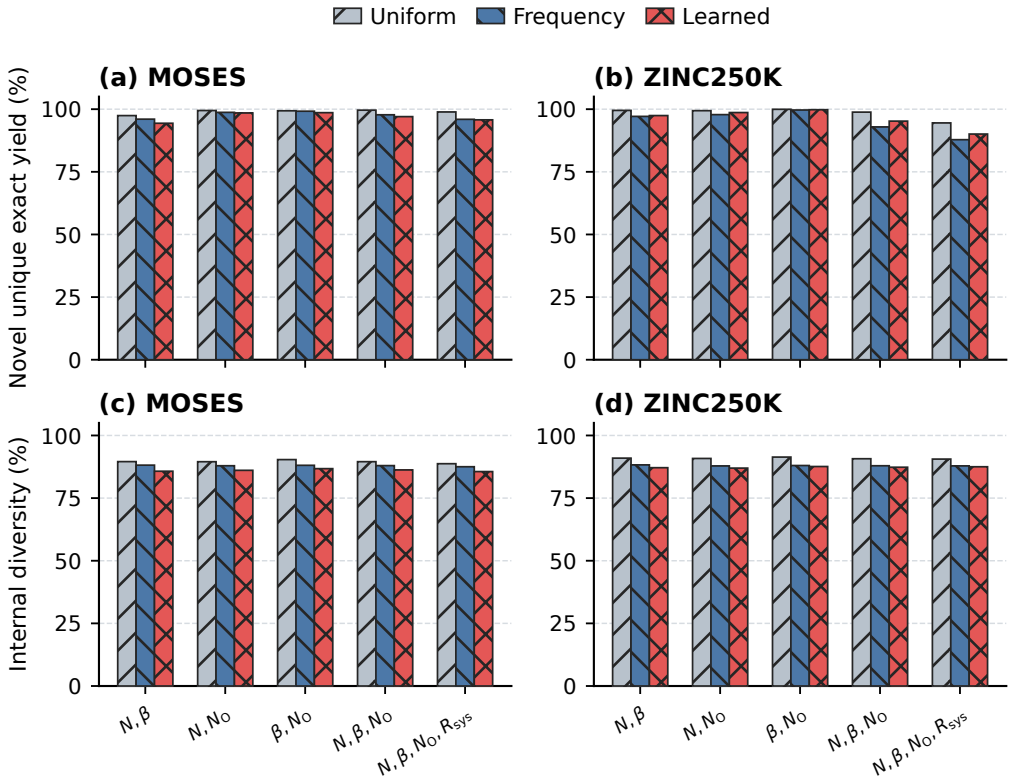


Figure S10: Joint ablations: NUE (a,b) and matched-output internal diversity (c,d).

E.2 Results

Completion checking. Table S4a reports the completion-checking results.

The table also reports NUE. Its close agreement with Exact-valid yield for Local and Factorized shows that most exact outputs are absent from training and unique within their targets.

Sampling policy. Figure S10 and Table S4c compare sampling policies under Joint checking.

Table S4c: Dataset-specific Filters and NSPDK results under different sampling policies.

Dataset	Schema	Metric	Uniform	Frequency	Learned
MOSES	(N, β)	Filters (%) ↑	66.39	73.97	**97.51**
	(N, N_{O})	Filters (%) ↑	66.91	76.27	**95.66**
	(β, N_{O})	Filters (%) ↑	65.00	77.84	**94.74**
	$(N, \beta, N_{\mathrm{O}})$	Filters (%) ↑	64.66	75.19	**96.16**
	$(N, \beta, N_{\mathrm{O}}, R_{\mathrm{sys}})$	Filters (%) ↑	69.61	74.77	**96.38**
ZINC250K	(N, β)	NSPDK ↓	0.0739	0.0373	**0.0239**
	(N, N_{O})	NSPDK ↓	0.0808	0.0207	**0.0046**
	(β, N_{O})	NSPDK ↓	0.0666	0.0181	**0.0045**
	$(N, \beta, N_{\mathrm{O}})$	NSPDK ↓	0.0791	0.0412	**0.0242**

Table S6: Compiler-computation ablation results on matched fixed-M conditions.

Dataset / variant	Time (s)	States	Transitions	Peak (GiB)	Complete / total
moses/full	0.044	1.35e+03	2.18e+04	4.04	4314/4314
moses/without grouping	0.611	1.35e+03	3.84e+05	4.04	4314/4314
moses/without optional bounds	0.0473	1.35e+03	2.18e+04	4.04	4314/4314
moses/without count sum	1.51	3.88e+03	1.06e+06	4.04	4314/4314
moses/without structural pruning	9.4	3.32e+04	5.56e+06	4.04	3722/4314
zinc250k/full	0.0488	1.5e+03	2.22e+04	1.36	5448/5448
zinc250k/without grouping	0.596	1.5e+03	3.44e+05	1.36	5448/5448
zinc250k/without optional bounds	0.0522	1.5e+03	2.22e+04	1.36	5448/5448
zinc250k/without count sum	1.8	4.37e+03	1.26e+06	1.36	5442/5448
zinc250k/without structural pruning	0.495	5.64e+03	2.57e+05	1.36	5058/5448

Compiler computation. The compiler ablation distinguishes reductions that change the reachable state set from those that avoid repeated work. Without grouping, the state count does not change, but transitions increase by 17.6× on MOSES and 15.5× on ZINC250K, explaining the corresponding 13.9× and 12.2× time increases. Removing count-sum or structural pruning also expands the state and transition sets and can prevent completion within the resource limits. Removing optional bounds leaves states, transitions, and completion unchanged and increases median time by at most 7.5%, so these bounds have a limited effect under the tested conditions.

F Vocabulary Support and Component Recombination

Table S8a: Component-vocabulary coverage results on the four test sets.

Dataset	Vocabulary size	Mol. cov.	Component cov.	Whole	Mean M/N	Core occ./unique
MOSES	14,798	99.29	99.93	0.0023	0.454	99.88/82.56
ZINC250K	10,651	97.76	99.79	0.0200	0.476	99.67/94.52
GuacaMol	57,662	97.33	99.79	0.0582	0.453	99.65/81.58
NPGen	70,591	93.25	99.64	0.1397	0.518	98.80/83.21

Table S8b: Ring-distribution results for TopoCompile on MOSES, ZINC250K, and GuacaMol.

Dataset	Ring systems	β	Ring size	Topology	Core	Core pair	Top core (%)	Distinct cov. (%)	Valid/attempts
MOSES	0.9634	0.9354	0.9992	0.9976	0.9778	0.8912	33.85	19.77	25000/25000
ZINC250K	0.9945	0.9946	0.9998	0.9990	0.9689	0.8351	36.04	35.62	25000/25000
GuacaMol	0.9970	0.9973	0.9988	0.9973	0.9262	0.8157	43.67	8.29	10000/10000

Singleton component types account for 43.1%, 43.8%, 52.0%, and 51.2% of the MOSES, ZINC250K, GuacaMol, and NPGen vocabularies, respectively. The 1,000 most frequent types cover 98.9%, 98.3%, 97.2%, and 97.4% of training component occurrences. Occurrence coverage and type-level coverage therefore describe different aspects of the support boundary. Component coverage defines the support of the representation. New molecules can combine known components

in previously unobserved multisets and bridge attachments; this does not create a cyclic interior absent from the vocabulary. For the MOSES sample, 23,146 of 24,978 distinct generated canonical SMILES are absent from training. Among all 25,000 valid generated molecules, 76.61% use a new component multiset and 97.96% a new attachment-bag signature. The latter records endpoint slots and bond types while ignoring connectivity among multiple instances of the same component template.

G SUPPLEMENTARY MOLECULES

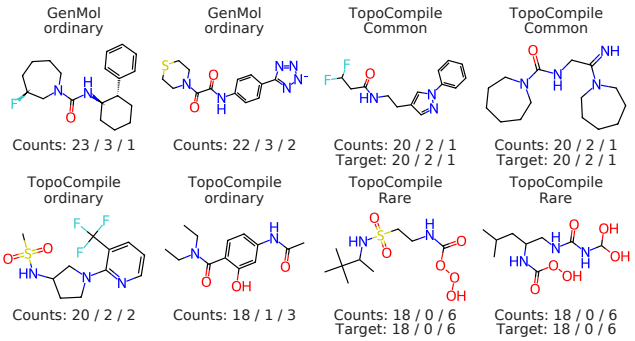


Figure S11: MOSES examples from unconditional generation and shared exact-control requests.

H RELATION TO HARD-CONSTRAINED MOLECULAR GENERATION

TopoCompile addresses a restricted but exact problem: equalities over non-negative component-additive integer counts under a finite component vocabulary and typed attachment grammar. Table S10 separates this contribution from exact constrained generation in general. Projection methods enforce supported graph properties during diffusion; CP-based methods solve or propagate symbolic sequence models; and SMT-based methods assemble neural proposals under logical constraints. TopoCompile instead specializes completion reasoning to the aggregate state $(k, \mathbf{f}, \mathbf{b})$ and queries exact suffix existence before every concrete component-and-attachment action. Its claim is therefore not the first use of hard constraints or neuro-symbolic generation, but an exact and reusable prefix-completion test induced by the component-tree representation.

Classical constraint programming constructs molecular graphs with required and forbidden fragments (Omrani & Naanaa, 2020). Beyond molecules, Ctrl-G combines a distilled hidden Markov model with deterministic finite automata to guarantee logical constraints during text generation (Zhang et al., 2024). TopoCompile instead derives a Boolean completion test directly from molecular count and attachment requirements, without a distilled sequence model. MacroGuide uses persistent-homology guidance to promote macrocycle generation, addressing a different topological objective from exact cycle-rank control (Maksymiuk et al., 2026).

Table S10: Mechanistic comparison of hard-constrained generators.

Method	Specification	Enforcement point	Relation to TOPOCOMPILE
PRODIGY / ConStruct (Sharma et al., 2024; Madeira et al., 2024)	Supported hard graph properties	Projection during diffusion	Exactness depends on a suitable property-specific projector or closure condition, rather than suffix feasibility for an additive count vector
Saikali & Pesant (Saikali & Pesant, 2026)	SMILES grammar with physicochemical and structural constraints	CP construction and search, guided by belief propagation	General symbolic search rather than a compact component-tree completion state
GeAI-BLAnC (Manibod et al., 2025)	Fixed-length sequence constraints, including SMILES grammar and molecular-weight ranges	CP/BP modifies the next-token distribution	Non-backtracking generation may fail to complete; full completion can use backtracking
Geng et al. (Geng et al., 2026)	Chemical, structural, and user logical constraints	SMT assembly from neural proposals	The solver guarantees assembled outputs rather than every neural construction prefix
TOPOCOMPILE	Multiple component-additive count equalities	Exact DP on $(k, \mathbf{f}, \mathbf{b})$ before every action	Every retained prefix is completable relative to the vocabulary and grammar, without trajectory backtracking or completed-sample rejection/repair

The published evaluations of these methods are not numerically interchangeable with Table 1. GeAI-BLAnC studies fixed-length SMILES under a molecular-weight interval, projection methods use their own constraint families and diffusion checkpoints, and Geng et al. (Geng et al., 2026) evaluate SMT-backed logical constraints. We therefore compare these systems mechanistically and reserve Table 1 for methods evaluated under the shared four-count requests and attempt budgets.