TRAJECTORY MATCHING: DISCRETE FLOW MATCHING WITH STRUCTURED ACTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The natural units of structured discrete generation are often not independent single-component edits but richer actions: attaching a fragment to a molecule, inserting a node together with an incident edge, or growing a hierarchy by adding a child. Yet most existing discrete flow matching methods (DFM) are restricted to independent component-wise edits over fixed-size states. This restriction stems from endpoint conditioning: endpoints fix the source and target during training but not the action sequence between them, making tractable conditional rates hard to define for complex actions that cannot be factorized component-wise. We propose trajectory matching, which conditions the flow on a full sequence of actions connecting the endpoints. This yields principled, simulation-free training targets for user-defined action spaces, including actions that change state size, modify multiple components, or depend on earlier actions. We prove that the resulting objective recovers the marginal flow and that trajectory matching strictly subsumes factorized DFM, while additionally capturing actions with prerequisite structure such as hierarchy growth or fragment-based molecule generation. Empirically, on tree and molecular graph generation, trajectory matching with domain-aligned actions matches or surpasses factorized DFM baselines (lifting MOSES validity from 92.8% to 99.2%) and transfers without retraining to substructure extension and linking. Trajectory matching enlarges the design space of discrete flow matching by turning the action vocabulary into a modeling choice.

1 INTRODUCTION

Many structured generation problems are most naturally expressed in terms of actions that reflect the structure of the domain rather than the structure of the representation. In molecular graph generation, for example, the natural actions are not single-component edits but operations like attaching a fragment to a partial structure or inserting an atom together with a bond. Such actions can change the size of the state, modify several components at once, and depend on one another—an atom can only be attached to atoms that have already been inserted. The value of such structured actions is well established in autoregressive models: fragment-based molecule generation (Jin et al., 2019), grammar-based code generation (Yin & Neubig, 2017), and insertion-based graph models (You et al., 2018) all exploit domain-aligned actions to achieve strong empirical performance.

However, existing discrete flow matching (DFM) (Campbell et al., 2024; Gat et al., 2024) and related non-autoregressive methods such as discrete diffusion (Austin et al., 2021) cannot easily express such actions. The reason is structural: DFM prescribes a conditional flow between sampled endpoints, and the corresponding rate must be tractable for training. To achieve this over combinatorially large state spaces, existing methods factorize the rate across components. This commits the model to fixed-size states, single-component edits, and independence between actions. Recent extensions (Havasi et al., 2025; Deutschmann et al., 2026; Lüdke et al., 2026) lift the fixed-size constraint by supporting insertions and deletions in specific domains, but they do so by mapping these operations into augmented state spaces where they again reduce to factorized single-component edits. Actions that modify multiple components or depend on one another therefore remain out of reach.

We address this gap with trajectory matching: rather than conditioning on endpoints alone, we condition on a full trajectory of actions connecting them. A trajectory specifies which actions are applied and in what order, leaving the conditional flow to determine only when each action is taken.

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.

This makes the conditional rate straightforward to define for general action spaces, including actions that change state size, modify multiple components, or depend on one another. The resulting training objective is still simulation-free and provably recovers the marginal data-generating flow. To make training efficient, we require an efficiently sampleable distribution over trajectories connecting the endpoints. We give explicit constructions for several settings, including actions with hierarchical dependencies. Our framework also recovers standard factorized DFM as the special case of independent component-wise actions with uniformly sampled orderings, placing factorized updates and actions with prerequisites within a common framework.

Empirically, we show that domain-aligned action spaces translate into concrete gains on tree and molecular graph generation. On tree generation, child-attachment actions yield 100% validity by construction across sampling budgets, while a factorized baseline collapses below 10% at 10 Euler steps. On molecular graphs, we demonstrate the flexibility of trajectory matching across four action spaces, ranging from factorized bond edits, which match existing DFM baselines, to fragment-based insertions, which lift MOSES validity from 92.8% to 99.2%. Additionally, a single variable-size insertion models transfers zero-shot to substructure extension and linking, substantially outperforming a factorized baseline that requires task-specific adaptation.

Our core contributions are:

- **Trajectory matching.** A discrete flow matching framework over user-defined action spaces, including size-changing, multi-component, and dependent actions, obtained by conditioning training on full action trajectories rather than endpoints. The objective recovers the marginal flow (Thm. 3.2); a Rao-Blackwellized variant reduces gradient variance (Prop. B.3).
- **A unifying framework.** Trajectory matching subsumes factorized DFM (Thm. 4.4) while extending to partial-order action systems, capturing hierarchy growth and fragment-based molecule construction.
- **Empirical gains on structured generation.** On tree and molecular graph generation, trajectory matching with domain-aligned action spaces yields concrete gains over factorized DFM baselines: 100% validity by construction on trees, MOSES validity from 92.8% to 99.2% on molecules, and zero-shot transfer to substructure extension and linking.

2 Action-Based Discrete Flow Matching

States and Actions We model generation as a sequence of actions on discrete states $s \in \mathcal{S}$ rather than component-wise updates. This enlarges the design space beyond standard factorized DFM. Let $\mathcal{A}$ be the global action vocabulary and $A(s) \subseteq \mathcal{A}$ the set of actions available at state s; each a produces a new state $a(s) \neq s$. The choice of $\mathcal{A}$ is itself a modeling decision: in molecules, for instance, one might edit bonds on a fixed atom set, attach atoms one at a time, or insert whole ring fragments. Actions may modify multiple components, depend on one another, or change the size of the state. We say actions a_1, a_2 are *independent* if the order of application does not matter, i.e., for every s with $a_1, a_2 \in A(s)$: $a_1 \in A(a_2(s))$, $a_2 \in A(a_1(s))$, and $a_1(a_2(s)) = a_2(a_1(s))$. For example, unmasking two tokens in text is independent, while insertion actions generally are not.

CTMCs Our construction is built on continuous-time Markov chains (CTMC) $(S_t)_{t\in[0,1]}$ on the state space $\mathcal{S}$. Rather than parameterizing the CTMC by transition rates between states, we parameterize it by an *action rate* $u_t(s, a) \geq 0$ of taking action $a \in A(s)$ at state s and time t. The infinitesimal transition probabilities for $s' \neq s$ are

$$\mathbb{P}[S_{t+h} = s' \mid S_t = s] = h \sum_{a(s)=s'} u_t(s, a) + o(h), \quad \mathbb{P}[S_{t+h} = s \mid S_t = s] = 1 - h \sum_{a\in A(s)} u_t(s, a) + o(h).$$

We say u_t *generates* the associated marginal *probability path* $p_t(s) = \mathbb{P}[S_t = s]$.

Discrete Flow Matching The goal of discrete flow matching (Campbell et al., 2024; Gat et al., 2024) is to learn the rate u_t of a CTMC that transports the source distribution p_{src} to the target distribution p_{tgt}. Directly specifying u_t is in general intractable. Instead, flow matching specifies, for each pair of endpoints (s_I, s_F), an endpoint-conditioned CTMC $(S_t \mid s_I, s_F)_t$ with prescribed rates $u_t(s, a \mid s_I, s_F)$ that generate a conditional path $p_t(s \mid s_I, s_F)$ satisfying

$$p_0(s \mid s_I, s_F) = \mathbf{1}[s = s_I], \qquad p_1(s \mid s_I, s_F) = \mathbf{1}[s = s_F].$$

Training uses a coupling distribution $\gamma(s_I, s_F)$ for the endpoints with marginals p_{src} and p_{tgt}, respectively. The marginal rate and path are then given by:

$$u_t(s, a) = \mathbb{E}_{(s_I, s_F)\sim\gamma}[u_t(s, a \mid s_I, s_F) \mid S_t = s], \quad p_t(s) = \mathbb{E}_{(s_I, s_F)\sim\gamma}\, p_t(s \mid s_I, s_F)$$

where by construction $p_0 = p_{\text{src}}$ and $p_1 = p_{\text{tgt}}$. The key insight of flow matching is that the marginal rate u_t generates the marginal path p_t (Lipman et al., 2023). The conditional rate $u_t(\cdot \mid s_I, s_F)$ thus serves as a tractable target for an unconditional model u_t^θ, trained by minimizing a Bregman divergence D (Holderrieth et al., 2024):

$$\mathbb{E}_{(s_I, s_F)\sim\gamma}\, \mathbb{E}_t\, \mathbb{E}_{s\sim p_t(\cdot \mid s_I, s_F)}\, D\big(u_t(s, \cdot \mid s_I, s_F), u_t^\theta(s, \cdot)\big) \tag{1}$$

Factorized Flows Specifying the conditional rate $u_t(s, a \mid s_I, s_F)$ is the central design choice in discrete flow matching: it must be efficiently computable and generate an efficiently sampleable path connecting s_I and s_F. In general action spaces, however, many different action sequences — potentially with dependencies between actions — may connect s_I to s_F, making it difficult to specify such rates from the endpoints alone. Existing discrete flow matching methods therefore make strong assumptions on the action space: 1) initial, final, and all intermediate states decompose into components $s = (s^1, \ldots, s^d)$ with the same fixed size d, 2) each action modifies exactly one component from its initial value s_I^i to its final value s_F^i, and 3) conditioned on the endpoints, components evolve independently. Under these assumptions, the conditional rate factorizes across components: for an action a acting on component i, it suffices to specify a rate $u_t(s, a \mid s_I, s_F)$ that depends only on s^i, s_I^i, and s_F^i. Each component then evolves independently between its initial and final values. A common choice is the linear schedule

$$u_t(s, a \mid s_I, s_F) = \mathbf{1}[a^i(s^i) = s_F^i] \cdot 1/(1 - t), \tag{2}$$

which generates the factorized path

$$p_t(s \mid s_I, s_F) = \prod_{i=1}^{d} p_t^i(s^i \mid s_I^i, s_F^i), \quad \text{with} \quad p_t^i(s^i \mid s_I^i, s_F^i) = (1 - t)\, \mathbf{1}[s^i = s_I^i] + t\, \mathbf{1}[s^i = s_F^i].$$

Other time schedules yield analogous constructions. This is the most widely used discrete flow matching formulation, which we will refer to as *factorized flows*.

3 Trajectory Matching

Factorized flows are simple to define and have shown strong empirical performance on a variety of tasks. However, their underlying assumptions rule out many natural modeling choices in structured domains: actions cannot change the size of the state (e.g., adding an atom to a molecule), cannot modify multiple components at once (e.g., inserting a full fragment into a molecular graph), and cannot depend on one another (e.g., a bond insertion requires both endpoints to already exist). We lift these restrictions by defining flows over general action spaces. As argued in § 2, specifying valid rates from endpoints alone is difficult in this setting, because many action sequences may connect the same endpoint pair. We therefore condition not only on endpoints but a full *trajectory*: a concrete sequence of actions transforming s_I into s_F. Conditioning on a trajectory makes the conditional rate straightforward to define, even for size-changing, multi-component, or dependent actions.

3.1 Trajectory-conditioned flows

A trajectory τ from an initial state s_I to a final state s_F is a sequence of states and actions

$$\tau : s_I = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n = s_F.$$

For ease of notation, we assume that no state s_i appears more than once in one trajectory.

Next, we define a trajectory-conditioned CTMC $(S_t \mid \tau)_{t\in[0,1]}$. The trajectory fixes *the order of actions and states* while the CTMC randomizes *when* each action is taken. We again specify the process via its rate $u_t(s, a \mid \tau)$, requiring it to generate a path with $p_0(s \mid \tau) = \mathbf{1}[s = s_I]$ and $p_1(s \mid \tau) = \mathbf{1}[s = s_F]$. A natural choice is to assume equal expected times between actions, i.e., the

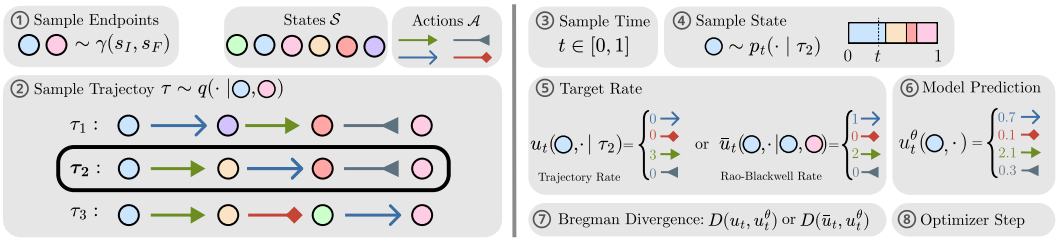


Figure 1: Training step of trajectory matching. Conditioning on a full trajectory (rather than endpoints alone) yields tractable training targets for general action spaces (including multi-component, dependent, or size-changing actions). Pseudocode in Algorithm 1.

execution times t_i of actions a_i ($0 < t_1 < \cdots < t_n < 1$) are distributed as the order statistics of n i.i.d. uniform random variables on $[0, 1]$. This corresponds to the rate

$$u_t(s, a \mid \tau) = \sum_{i=0}^{n-1} \mathbf{1}[s = s_i \wedge a = a_{i+1}] \cdot \frac{n-i}{1-t} \tag{3}$$

in which only the next action in the trajectory has positive rate, proportional to the number of remaining actions. The generated path has a simple closed form:

Proposition 3.1 (Trajectory-conditioned path)**.** *Under rate Eq. (3), the generated probability path is*

$$p_t(s \mid \tau) = \sum_{i=0}^{n} \mathbf{1}[s = s_i] \cdot \mathbb{P}[K_t = i],$$

where $K_t \sim$ Bin(n, t) counts the number of actions taken by time t.

In particular, $p_t(\cdot \mid \tau)$ is easy to sample from. Other choices of conditional rates are discussed in App. C. Importantly, conditioning on the trajectory lets us define a CTMC between initial and final state even for settings with complex actions where the endpoint-conditioned rate cannot be factorized.

3.2 Trajectory Matching

Having defined trajectory-conditioned CTMCs, we now describe how to use their rates as training targets for an unconditional model $u_t^\theta(s, \cdot)$ that depends only on the current state and time. For any endpoint pair (s_I, s_F), many valid trajectories may connect them. Therefore, in addition to the coupling γ over endpoints, we also specify a conditional distribution $q(\tau \mid s_I, s_F)$ over trajectories from s_I to s_F. We assume $q(\cdot \mid s_I, s_F)$ is supported on valid trajectories for all (s_I, s_F) in the support of γ, and that sampling from q is efficient. Intuitively, q encodes an inductive bias over how s_I should be transformed into s_F: it determines which trajectories contribute to training and with what weight. Analogously to endpoint-conditioned discrete flow matching, one can show:

Theorem 3.2. *Let $u_t(s, a \mid \tau)$ be a conditional rate that generates the trajectory-conditioned path $p_t(s \mid \tau)$. Define the marginal probability path and marginal rate as*

$$p_t(s) := \mathbb{E}_{\tau \sim q}\, p_t(s \mid \tau), \qquad u_t(s, a) := \mathbb{E}_{\tau \sim q}[\, u_t(s, a \mid \tau) \mid S_t = s\,],$$

where $\mathbb{E}_{\tau \sim q}$ is shorthand for $\mathbb{E}_{(s_I, s_F) \sim \gamma}\, \mathbb{E}_{\tau \sim q(\cdot \mid s_I, s_F)}$. Then u_t generates p_t.

In particular, we have $p_0 = p_{\text{src}}$ and $p_1 = p_{\text{tgt}}$, so the marginal rate u_t transports noise to data.

Training. Similarly to endpoint-conditioned flow matching, Thm. 3.2 enables efficient simulation-free training using a Bregman divergence D:

$$\mathcal{L}(\theta) := \mathbb{E}_{\tau \sim q}\, \mathbb{E}_t\, \mathbb{E}_{s \sim p_t(\cdot \mid \tau)}\, D\big(u_t(s, \cdot \mid \tau),\, u_t^\theta(s, \cdot)\big)\,. \tag{4}$$

This loss is a valid proxy for the marginal matching loss in two senses. First, its gradients agree, so optimizing $\mathcal{L}(\theta)$ is equivalent to optimizing the marginal target:

Proposition 3.3 (Consistent gradients)**.** *For any parametric family u_t^θ,*

$$\nabla_\theta \mathcal{L}(\theta) \;=\; \nabla_\theta\, \mathbb{E}_t\, \mathbb{E}_{s \sim p_t}\, D\big(u_t(s, \cdot),\, u_t^\theta(s, \cdot)\big)\,.$$

Second, with a sufficiently expressive family, the minimizer recovers the data-generating flow:

Corollary 3.4 (Consistent optima)**.** *If the family u_t^θ contains the marginal rate u_t, then any minimizer of $\mathcal{L}(\theta)$ equals u_t for almost all (s,t) with $p_t(s) > 0$ and generates the marginal probability path p_t.*

Fig. 1 summarizes the resulting training procedure. Importantly, trajectory matching training remains simulation-free as $p_t(s \mid \tau)$ can be efficiently sampled from (Prop. 3.1) and $u_t(s, a \mid \tau)$ can be computed in closed-form. Sampling from γ and q is application-specific; we discuss efficient instantiations in § 4. In our experiments, overlapping preprocessing with GPU training makes the additional wall-clock cost small (App. G.1).

Sampling. Sampling proceeds as in endpoint-conditioned DFM: draw $s_I \sim p_{\text{src}}$ and integrate the learned CTMC with rates u_t^θ over $t \in [0,1]$. With Euler integration, each step of size h samples each available action a independently with probability $h\, u_t^\theta(s,a)$ and applies the sampled compatible actions to the current state; see App. D. Although the trajectory-conditioned targets $u_t(\cdot \mid \tau)$ are sparse—only the next action in the sampled trajectory has positive rate—the learned model u_t^θ approximates the *marginal* rate (Thm. 3.2), which can place mass on many currently available actions. This allows multiple actions per Euler step, so sampling does not degenerate to sequential generation.

In summary, trajectory matching separates three modeling decisions: i) **the state and action space**, which determines what transformations are possible; ii) **the source and target distributions $p_{\text{src}}, p_{\text{tgt}}$ and coupling γ**, which specify *what* is transported; and iii) **the trajectory distribution $q(\tau \mid s_I, s_F)$**, which specifies *how* endpoints are connected through action sequences.

3.3 Variance reduction via Rao–Blackwellization

As we have seen, the trajectory-matching objective Eq. (4) recovers the correct marginal flow by supervising the model on a single next action per sampled trajectory. We can sharpen the training signal by observing that multiple trajectories in $q(\cdot \mid s_I, s_F)$ may pass through the same state s at time t, each prescribing a potentially different next action. We now show how to supervise the model on all such actions simultaneously, reducing gradient variance while preserving the optimum. To this end, define the posterior distribution over trajectories given the current state s and endpoints (s_I, s_F):

$$p_t(\tau \mid s, s_I, s_F) := \frac{p_t(s \mid \tau) \cdot q(\tau \mid s_I, s_F)}{\sum_{\tau'} p_t(s \mid \tau') \cdot q(\tau' \mid s_I, s_F)}$$

The *Rao-Blackwellized target* is then the expectation of $u_t(s, a \mid \tau)$ under this posterior:

$$\bar{u}_t(s, a \mid s_I, s_F) := \mathbb{E}_{\tau \sim p_t(\cdot \mid s, s_I, s_F)}\, u_t(s, a \mid \tau).$$

When this expectation can be computed efficiently (as we show for several trajectory families in § 4), the Rao–Blackwellized rate can replace $u_t(\cdot \mid \tau)$ as the training target in Eq. (4). This preserves the expected gradient and the optimum while reducing gradient variance (Props. B.2 and B.3). Our ablation suggests that this yields faster convergence to comparable final performance (App. G.4).

In summary, conditioning on the full trajectory gives a general recipe for defining valid conditional flows over arbitrary action spaces, including settings where endpoint-conditioned rates cannot be specified directly. This yields a simulation-free training objective whose optimum recovers the marginal data-generating flow. Gradient variance can be further reduced by Rao-Blackwellizing the target over all trajectories consistent with the current state and endpoints.

4 Instantiations of Trajectory Matching

We now instantiate trajectory matching at increasing levels of generality. The two central design choices are the action space and the trajectory distribution q. We illustrate these choices through a running example of molecular graph generation (Fig. 2), moving from independent actions to settings outside the scope of factorized flows. We begin with the most common case: for each endpoint pair, trajectories differ only in the *order* of actions, while the action set itself is fixed.

Definition 4.1 (Fixed-action distribution)**.** q is a *fixed-action distribution* if for every (s_I, s_F), all trajectories $\tau \in \operatorname{supp} q(\cdot \mid s_I, s_F)$ use the same set of actions, denoted $\alpha(s_I, s_F)$.

When the same property holds at intermediate states, we write $\alpha(s, s_F)$ for the set of actions remaining after s: that is, all trajectories passing through s on their way to s_F share the same remaining action set, independent of how s was reached.

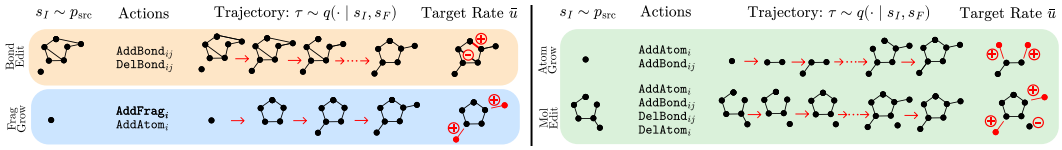


Figure 2: Four trajectory-matching instantiations on molecular graphs, BONDEDIT (factorized bond edits), FRAGGROW (fragment-based growth), ATOMGROW (atom/bond growth from a seed), and MOLEDIT (noisy-initial editing), each shown with a sampled source s_I, action set, example trajectory τ, and Rao-Blackwellized target rate $\bar{u}_t$. Full definitions in App. E, more instantiations in Fig. 6

Independent fixed-action distribution

The simplest case is when the required actions are pairwise independent and the trajectory distribution is uniform over their orderings. For example, in BONDEDIT, which mirrors DeFoG (Qin et al., 2025), atoms are fixed by the coupling γ and only edges can be inserted or deleted. Any ordering of these independent edge edits yields a valid trajectory.

Definition 4.2 (Independent fixed-action distribution)**.** A fixed-action distribution q is *independent* if for every (s_I, s_F), the actions in $\alpha(s_I, s_F)$ are pairwise independent, and $q(\cdot \mid s_I, s_F)$ is uniform over their orderings.

Sampling from q is efficient whenever computing α is. The rate $\bar{u}_t$ also admits a closed form:

Proposition 4.3. *Let q be an independent fixed-action distribution. Then for a state s on a trajectory from s_I to s_F we have: $\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \alpha(s, s_F)] \cdot 1/(1-t)$.*

Thus, all remaining actions receive the same rate, reflecting the order-invariance. Importantly, factorized flow matching arises as the special case of trajectory matching with order-invariant, independent component-wise actions:

Theorem 4.4. *Consider a factorized flow as defined in Eq. (2) with single component actions $a_v^i(s) = (s^1, \ldots, v, \ldots, s^d)$. Let q be the corresponding independent fixed action distribution with $\alpha(s_I, s_F) = \{a_v^i : v = s_F^i \neq s_I^i\}$. Then:*

1. *The trajectory-matching loss (Eq. (4)) and the factorized flow matching loss (Eq. (1)) have the same gradient in θ and hence the same minimizers.*
2. *The Rao-Blackwellized loss (Prop. 4.3, Prop. B.2) is identical to the factorized loss (Eq. (1)).*

Dependent fixed-action distribution

Having established that trajectory matching subsumes factorized flows, we now relax the independence assumption. Dependencies between actions arise naturally in structured domains: in hierarchy generation, adding a child requires its parent to exist; in molecular generation without a fixed atom set, inserting a bond requires both endpoint atoms to exist. A natural way to model such dependencies is through a directed acyclic graph (DAG) over actions.

Definition 4.5. We equip the action set $\alpha(s, s_F)$ with a DAG whose vertices are the actions and whose edges $a_i \to a_j$ indicate that a_i is a prerequisite of a_j. We require two actions to be independent if neither is reachable from the other. A trajectory τ is consistent with $\alpha(s_I, s_F)$ if its action sequence is a topological ordering of this DAG.

Similarly to the independent fixed-action distribution, we define a dependent fixed-action distribution as uniform over all *consistent* trajectories:

Definition 4.6 (Dependent fixed-action distribution)**.** We say q is a *dependent fixed-action distribution* if for all s_I, s_F every trajectory in the support of $q(\cdot \mid s_I, s_F)$ is consistent with the dependency DAG $\alpha(s_I, s_F)$ and $q(\cdot \mid s_I, s_F)$ is uniform over all such trajectories.

This family interpolates between two familiar regimes: an edgeless DAG recovers the independent fixed-action distribution, while a linear chain DAG recovers fixed-order autoregressive generation. General DAGs lie between these extremes and capture partially ordered action systems such as our GROWHIERARCHY (a tree grown by attaching children to existing nodes) and FRAGGROW (a molecule grown by attaching ring fragments to a root atom); see Fig. 2 and App. E.

Unlike the independent case, the Rao–Blackwellized target $\bar{u}_t$ can assign different rates to the remaining actions. Available actions on which many future actions depend receive higher rates, encouraging them to occur earlier. For forest-structured dependency DAGs, trajectories can be sampled and Rao–Blackwellized rates computed efficiently:

Proposition 4.7. *Let q be a dependent fixed-action distribution whose dependency DAG $\alpha(s_I, s_F)$ is a forest. Then for a state s on a trajectory from s_I to s_F we have:*

$$\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \text{roots}(\alpha(s, s_F))] \cdot \text{desc}_{\alpha(s,s_F)}(a) \cdot 1/(1-t)$$

where roots *is the set of roots of the DAG and* $\text{desc}(a)$ *is the size of the tree rooted at a (a included).*

The resulting target interpolates between the independent and fully dependent extremes. With no dependencies between actions, it reduces to the uniform rate from Prop. 4.3, supervising every remaining action with equal weight. With a fully sequential chain of dependencies, all weight concentrates on the single next action, recovering fixed-order autoregressive supervision. Between these extremes, every currently available action receives a rate proportional to the size of its remaining subtree. Rao–Blackwellized rates for general dependency DAGs are given in Prop. B.4, but computing them exactly is intractable in general. For the forest-plus-sink dependencies arising in AtomGrow and MolEdit, we derive an alternative conditional path with exact, tractable rates (App. E.7).

Dependent fixed action distributions extend trajectory matching to partially ordered action systems that factorized endpoint-conditioned flows cannot express. Conditioning on concrete trajectories yields targets that span the full spectrum from unordered to fully sequential actions in a unified way.

Beyond fixed action sets

So far we assumed that all trajectories between a given endpoint pair use the same set of actions. In some settings, however, different trajectories between s_I and s_F involve different action sets, for example because the target admits multiple decompositions. In AtomGrow, a given ring bond may be created when a new atom is attached by `AddAtom` or by a separate `AddBond` action after both endpoints exist. One way to obtain tractable trajectory distributions in such settings is to introduce a latent variable z specifying the decomposition. We express the trajectory distribution as a mixture, which can be sampled efficiently whenever both the latent distribution and the conditional trajectory distribution can be sampled efficiently:

$$q(\tau \mid s_I, s_F) = \mathbb{E}_{z \sim p(z \mid s_I, s_F)}\, q(\tau \mid z, s_I, s_F). \tag{5}$$

For AtomGrow, z specifies a spanning tree rooted at the seed atom: tree edges are created by `AddAtom` actions, while other edges require `AddBond` actions. Conditional on z, the action set and its dependencies are fixed again, forming a forest-plus-sink structure for which we have a tractable conditional path with exact rates (App. E.7). MolEdit uses a similar construction (App. E).

Importantly, we can also understand the existing insertion- and deletion-based text model EditFlows (Havasi et al., 2025) as an instantiation of trajectory matching over a variable action set; see App. F.1.

In summary, trajectory matching provides a flexible and principled framework for designing discrete flows over domain-specific action spaces. It subsumes classical endpoint-conditioned factorized flows as a special case, while naturally extending to complex actions, partial dependencies, and structured generation processes for which direct endpoint-conditioned targets are difficult to specify.

5 Empirical Results

We evaluate trajectory matching on tree and molecular graph generation and show that: (i) domain-aligned action spaces can enforce *structural validity by construction*, which factorized flows cannot; (ii) our framework supports *diverse action spaces* over the same domain, where domain-aligned ones can outperform factorized baselines; and (iii) the resulting models *transfer without retraining* to structured downstream tasks such as substructure extension and linking. All instantiations share the same training and sampling procedure and graph transformer backbone (Vignac et al., 2023); only the action space, source distribution, and trajectory distribution differ. Setup details are in App. G.1.

Hierarchy generation. Our GrowHierarchy instantiation generates rooted trees using a single action type that attaches a new child to an existing node. Because the action space only contains tree-preserving operations, every reachable state is a connected tree and *validity is guaranteed by*

Table 1: Graph generation on Tree. Full results in Tab. 7.

	Tree		
	Model	V.U.N. ↑	Ratio ↓
Fixed-size	DiGress	90.0	**1.6**
	EDGE	0.0	850.7
	BwR	0.0	11.4
	HSpectre (one-shot)	82.5	2.1
	DeFoG	$96.5_{\pm 2.6}$	$\mathbf{1.6}_{\pm 0.4}$
Variable-size	GraphRNN	0.0	607.0
	GRAN	0.0	607.0
	BiGG	75.0	5.2
	GraphGen	95.0	33.2
	HSpectre	**100**	4.0
	GrowHierarchy	$\mathbf{100.0}_{\pm 0.0}$	$2.5_{\pm 0.6}$

Table 2: Comparison of DFM and diffusion molecular graph generative models on ZINC (Irwin et al., 2020) and MOSES (Polykovskiy et al., 2020). Best results in **bold**, second best underlined.

		ZINC					MOSES						
	Model	Val. ↑	Uniq. ↑	FCD ↓	NSPDK ($\times 10^3$) ↓	Scaf ↑	Val. ↑	Uniq. ↑	Nov. ↑	Filt. ↑	FCD ↓	SNN ↑	Scaf ↑
	Training set	100.0	100.0	0.19	0.1	0.48	100.0	100.0	0.0	100.0	0.01	0.64	99.1
Fixed-size	DiGress (Vignac et al., 2023)	95.0	–	3.48	2.1	0.42	85.7	100.0	95.0	97.1	1.19	0.52	14.8
	DisCo (Xu et al., 2024)	–	–	–	–	–	88.3	100.0	97.7	95.6	1.44	0.50	15.1
	Cometh (Siraudin et al., 2024)	–	–	–	–	–	90.5	99.9	92.6	99.1	1.27	0.54	**16.0**
	GBD (Liu et al., 2024)	97.9	–	2.25	1.8	0.50	–	–	–	–	–	–	–
	DeFoG (Qin et al., 2025)	$\mathbf{99.2}_{\pm 0.1}$	$99.99_{\pm 0.01}$	$1.43_{\pm 0.02}$	$0.8_{\pm 0.1}$	$\mathbf{0.59}_{\pm 0.01}$	92.8	99.9	92.1	98.9	1.95	**0.55**	14.4
	BondEdit	$98.8_{\pm 0.1}$	$99.98_{\pm 0.02}$	$1.41_{\pm 0.02}$	$0.5_{\pm 0.0}$	$0.56_{\pm 0.01}$	84.9	100.0	97.0	96.6	1.38	0.50	14.8
Variable-size	GrIDDD (Ninniri et al., 2025)	85.3	99.95	4.30	2.3	0.34	79.6	99.9	98.5	84.7	1.67	0.48	14.9
	MolEdit	$89.6_{\pm 0.1}$	$99.99_{\pm 0.00}$	$3.79_{\pm 0.02}$	$4.9_{\pm 0.0}$	$0.36_{\pm 0.01}$	76.7	100.0	**99.3**	93.1	2.09	0.48	7.7
	AtomGrow	$98.2_{\pm 0.1}$	$\mathbf{100.00}_{\pm 0.00}$	$1.24_{\pm 0.03}$	$0.6_{\pm 0.1}$	$0.52_{\pm 0.01}$	82.0	100.0	94.6	98.1	**0.94**	0.53	13.0
	FragGrow	$98.9_{\pm 0.0}$	$\mathbf{100.00}_{\pm 0.00}$	$\mathbf{1.22}_{\pm 0.03}$	$\mathbf{0.4}_{\pm 0.0}$	$0.51_{\pm 0.01}$	**99.2**	99.9	90.7	**99.3**	1.21	**0.55**	13.7

construction, without filtering or projection. On the Tree dataset (Bergmeister et al., 2024), all evaluated samples are also unique and novel, yielding 100% V.U.N. GROWHIERARCHY achieves a Ratio of 2.5, the best distributional graph-statistics score among the evaluated variable-size methods (Tab. 1). The advantage of our domain-aligned actions is particularly clear at low sampling budgets: at just 10 Euler steps, well below the number of actions per trajectory, GROWHIERARCHY maintains a comparable Ratio and 100% validity, while DeFoG's validity drops to 8% (Figs. 3 and 7).

Molecular graph generation. We instantiate trajectory matching with four different action spaces over the same domain, illustrating the flexibility of the framework: BONDEDIT (bond edits on a fixed-size atom set, mirroring existing factorized approaches), ATOMGROW (atom and bond insertions from a seed atom), FRAGGROW (ring fragment insertions from a dataset-specific vocabulary), and MOLEDIT (insertions and deletions on noisy initial molecules); see Fig. 2. Tab. 2 reports results on ZINC and MOSES against fixed-size and variable-size baselines. QM9 results are in Tab. 8.

Three findings stand out. First, BONDEDIT, our factorized instantiation, matches existing factorized DFM and discrete diffusion baselines (despite using no sampling-time optimizations), showing that the generality of trajectory matching comes at no cost in the factorized regime (Thm. 4.4). Second, ATOMGROW and FRAGGROW outperform the variable-size diffusion baseline GrIDDD (Ninniri et al., 2025) on most metrics, suggesting that directly modeling size-changing actions offers advantages over augmenting factorized diffusion with auxiliary size prediction. Third, FRAGGROW, which encodes the strongest inductive bias by treating ring fragments as atomic actions, achieves very strong DFM results on ZINC and MOSES, most notably lifting MOSES validity from 92.8% to 99.2%. This empirically supports the value of aligning the action space with the structure of the domain.

Downstream applications. Trajectory matching naturally supports zero-shot transfer to structured downstream tasks by allowing generation to start from meaningful partial states. We demonstrate this on two tasks: *substructure extension* (extend a partial substructure into a valid molecule containing it) and *substructure linking* (join two disjoint substructures into a single valid molecule). A single ATOMGROW model trained on ZINC with a random number of root atoms accommodates both directly: extension by setting s_I to the given substructure, and linking by setting s_I to the union of two disconnected substructures (see Apps. E.3 and G.6). While ATOMGROW can generate extensions and linkers without a fixed target size, adapting the factorized BONDEDIT baseline requires sampling a target size in advance and freezing the conditioning substructure(s). ATOMGROW substantially outperforms the factorized baseline on both tasks, achieving a valid-and-connected fraction of 94.6% vs. 33.8% on extension across 11 substructures, including some not present during training, and 78.8% vs. 57.0% on linking across 6 fragment pairs (see Tab. 3). This works because ATOMGROW's trajectories pass through valid partial molecules, precisely the states on which these downstream tasks condition. More generally, trajectory matching makes the path between endpoints a modeling choice: training intermediates can be chosen to align with downstream conditional tasks, rather than being fixed by a factorized endpoint-conditioned construction.

6 RELATED WORK

Action-based generation. Several lines of work generate structured objects through action sequences, using different training objectives and generation paradigms: GFlowNets (Bengio et al., 2021; 2026) learn distributions over action trajectories on a DAG, but target reward-proportional sampling and are trained via balance constraints rather than conditional matching. Autoregressive models handle complex actions naturally by generating in a fixed order (Jin et al., 2019; Kusner et al., 2017; You et al., 2018), but commit to that order and to sequential generation. Non-autoregressive edit-based

Figure 3: Tree validity by sampling steps.

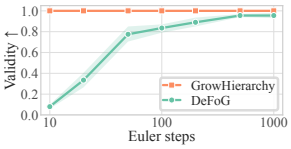


Table 3: Substructure extension and linking. Scores are valid-and-connected percentages (↑). Full results in Figs. 10 and 12.

Substructure extension				Substructure linking			
Seed	BondEdit	AtomGrow	Sample AtomGrow	Seed	BondEdit	AtomGrow	Sample AtomGrow
	50.4	**98.4**			39.8	**71.1**	
	31.3	**91.8**			60.9	**76.6**	
All 11 seeds	33.8	**94.6**		All 6 pairs	57.0	**78.8**	

models such as DiffusER (Reid et al., 2022) predict a restricted set of single-token edit operations in parallel, but rely on heuristic reconstruction targets without a principled guarantee that the learned model samples from the data distribution.

Discrete flow matching. Discrete diffusion (Austin et al., 2021) and DFM (Gat et al., 2024; Campbell et al., 2024) provide a principled framework for non-autoregressive generation that provably matches the data distribution, and have been successfully applied to text (Austin et al., 2021), graphs (Qin et al., 2025; Vignac et al., 2023), and molecules (Poletukhin et al., 2026). However, these methods share a common template restricting the action space: the conditional dynamics are determined by endpoints alone, factorize across components, and are restricted to single-component updates on a fixed-size state. ARDM (Hoogeboom et al., 2022) is an exception to endpoint-conditioning: training over random unmasking orders implicitly conditions on a trajectory, but is restricted to fixed-size, single-token unmasking operations.

A small body of work relaxes the fixed-size constraint: Jump diffusion (Campbell et al., 2023) combines continuous-state diffusion with a separate jump process and an auxiliary insertion head that predicts new component values and locations. GRIDDD (Ninniri et al., 2025) similarly augments fixed-size graph diffusion with an auxiliary model that predicts how many nodes to insert at each step. Edit Flows (Havasi et al., 2025), the closest prior work, handles single-token insertions, substitutions, and deletions for text by lifting them into an augmented state space, where they reduce to factorized single-component updates. Variants of this construction have been applied to temporal point processes (Lüdke et al., 2026), 3D molecule generation (Billera et al., 2026), and protein generation (Deutschmann et al., 2026). None of these approaches supports multi-component actions (e.g., fragment attachment) or general actions with explicit prerequisites (e.g., edge insertions requiring both endpoints). Trajectory matching addresses this gap through full-trajectory conditioning, providing, to our knowledge, the first general DFM construction for such actions. It also recovers factorized DFM and Edit Flow rates as special cases, yielding a unified framework.

7 Limitations

Trajectory matching requires specifying both an action space $\mathcal{A}$ and a trajectory distribution q, making its performance dependent on domain-specific design choices. The framework is most useful when the actions encode meaningful structure, such as fragment insertions in molecular graphs. In domains where independent component-wise edits already capture the relevant structure, complex actions may offer no advantage. In this case, trajectory matching can fall back to factorized DFM.

The tractability of trajectory matching depends on being able to sample from q efficiently and, ideally, to compute the Rao-Blackwellized rates in closed form. For many natural choices—independent fixed-action distributions and forest-structured dependencies—both can be done in linear time. Outside these families, both may require domain-specific constructions (e.g., Eq. (5)), so applying trajectory matching to a new domain can itself require non-trivial methodological work.

Our empirical evaluation covers trees and molecular graphs across four datasets. While these settings exercise diverse action structures, evaluating the framework in further domains remains future work.

8 Conclusion

We introduced trajectory matching, a discrete flow matching framework that operates over user-defined action spaces by conditioning on full action trajectories rather than on endpoints alone. The framework subsumes factorized DFM and extends it to actions that change state size, modify multiple components, or depend on one another. Empirically, domain-aligned action spaces translate into concrete gains across tree and molecular graph generation. By turning the action vocabulary into a modeling choice, trajectory matching opens up the design space of DFM.

AI USE STATEMENT

The idea of this work, the general conceptual framework and its specific instantiations was developed by the authors without any substantial use of generative AI. The main text was polished with substantial use of generative AI, which provided feedback on clarity, suggested alternative formulations of sentences and paragraphs, and helped to find typos, inconsistencies and errors. Text in the appendix was for some sections (mainly the experimental details and some proofs) in parts drafted by generative AI based on provided implementation details or proof sketches by the authors. E.g., a draft for the precise description of the experimental setup was generated by generative AI based on the provided implementation. Again, all text generated by generative AI was carefully checked, partly revised, and verified by the authors.

The code for the general framework was written by the authors. The implementation of specific instantiations and experiments was done with the help of AI Coding Assistants. Additionally, we used generative AI to provide feedback on the experiments, find bugs, and interpret some of the results. We did not use AI to generate datasets or clean them. We extensively tested the code and use the existing evaluation framework used by other baselines (e.g. DeFoG).

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies foundational aspects of generative modeling of structured discrete objects. Our experiments include molecular graph generation, which can support beneficial applications such as drug discovery but also carries risks of misuse, including the design of harmful compounds. We do not identify ethical concerns specific to our method beyond those associated with generative modeling of molecules in general.

REPRODUCIBILITY STATEMENT

The mathematical assumptions and proofs supporting our framework are provided in App. B, with the consistency of the implemented decomposed objective established in App. G.1. Training and sampling procedures are described in App. D, and the action spaces, source distributions, and trajectory constructions for each instantiation are specified in App. E. App. G.1 documents the datasets and splits, model architecture, training hyperparameters, hardware, sampling budgets, and evaluation protocols. Additional ablations and the downstream evaluation procedure are provided in App. G. We will make the code for our method and experiments available to reviewers and area chairs through an anonymous repository linked in a private OpenReview comment during the discussion phase, in accordance with the ICLR guidelines. We commit to releasing the code publicly upon publication.

REFERENCES

Michael S. Albergo and Eric Vanden-Eijnden. Building Normalizing Flows with Stochastic Interpolants, March 2023. URL http://arxiv.org/abs/2209.15571. arXiv:2209.15571 [cs].

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation, November 2021. URL http://arxiv.org/abs/2106.04399. arXiv:2106.04399 [cs].

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet Foundations, January 2026. URL http://arxiv.org/abs/2111.09266. arXiv:2111.09266 [cs].

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

Lukas Billera, Hedwig Nora Nordlinder, Jack Collier Ryder, Anton Oresten, Aron Stålmarck, Theodor Mosetti Björk, and Ben Murrell. Branching Flows: Discrete, Continuous, and Manifold Flow Matching with Splits and Deletions, April 2026. URL http://arxiv.org/abs/2511.09465. arXiv:2511.09465 [stat].

Andrew Campbell, William Harvey, Christian Weilbach, Valentin De Bortoli, Tom Rainforth, and Arnaud Doucet. Trans-Dimensional Generative Modeling via Jump Diffusion Models, October 2023. URL http://arxiv.org/abs/2305.16261. arXiv:2305.16261 [stat].

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative Flows on Discrete State-Spaces: Enabling Multimodal Flows with Applications to Protein Co-Design, June 2024. URL http://arxiv.org/abs/2402.04997. arXiv:2402.04997 [stat].

Xiaohui Chen, Jiaxing He, Xu Han, and Li-Ping Liu. Efficient and Degree-Guided Graph Generation via Discrete Diffusion Modeling, May 2023. URL http://arxiv.org/abs/2305.04111. arXiv:2305.04111 [cs].

Chaoran Cheng, Zhanghan Ni, Yanru Qu, Yuxin Chen, Ruihan Guo, Jiajun Fan, and Ge Liu. Variable-Length Generative Protein Design via Generalized Poisson Flow, July 2026. URL http://arxiv.org/abs/2607.09039. arXiv:2607.09039 [cs.LG].

Hanjun Dai, Azade Nazi, Yujia Li, Bo Dai, and Dale Schuurmans. Scalable Deep Generative Modeling for Sparse Graphs, June 2020. URL http://arxiv.org/abs/2006.15502. arXiv:2006.15502 [cs].

Nicolas Deutschmann, Constance Ferragu, Jonathan D. Ziegler, Shayan Aziznejad, and Eli Bixby. EvoFlows: Evolutionary Edit-Based Flow-Matching for Protein Engineering, April 2026. URL http://arxiv.org/abs/2603.11703. arXiv:2603.11703 [cs].

Nathaniel Diamant, Alex M. Tseng, Kangway V. Chuang, Tommaso Biancalani, and Gabriele Scalia. Improving Graph Generation by Restricting Graph Bandwidth, May 2023. URL http://arxiv.org/abs/2301.10857. arXiv:2301.10857 [cs].

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky T. Q. Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete Flow Matching, November 2024. URL http://arxiv.org/abs/2407.15595. arXiv:2407.15595 [cs].

Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. GraphGen: A Scalable Approach to Domain-agnostic Labeled Graph Generation. In *Proceedings of The Web Conference 2020*, pp. 1253–1263, April 2020. doi: 10.1145/3366423.3380201. URL http://arxiv.org/abs/2001.08184. arXiv:2001.08184 [cs].

Marton Havasi, Brian Karrer, Itai Gat, and Ricky T. Q. Chen. Edit Flows: Flow Matching with Edit Operations, October 2025. URL http://arxiv.org/abs/2506.09018. arXiv:2506.09018 [cs].

Peter Holderrieth, Marton Havasi, Jason Yim, Neta Shaul, Itai Gat, Tommi Jaakkola, Brian Karrer, Ricky T. Q. Chen, and Yaron Lipman. Generator Matching: Generative modeling with arbitrary Markov processes, October 2024. URL https://arxiv.org/abs/2410.20587v3.

Emiel Hoogeboom, Alexey A. Gritsenko, Jasmijn Bastings, Ben Poole, Rianne van den Berg, and Tim Salimans. Autoregressive Diffusion Models, February 2022. URL http://arxiv.org/abs/2110.02037. arXiv:2110.02037 [cs].

John J. Irwin, Khanh G. Tang, Jennifer Young, Chinzorig Dandarchuluun, Benjamin R. Wong, Munkhzul Khurelbaatar, Yurii S. Moroz, John Mayfield, and Roger A. Sayle. ZINC20—A Free Ultralarge-Scale Chemical Database for Ligand Discovery. *Journal of Chemical Information and Modeling*, 60(12):6065–6073, December 2020. ISSN 1549-9596. doi: 10.1021/acs.jcim.0c00675. URL https://doi.org/10.1021/acs.jcim.0c00675.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction Tree Variational Autoencoder for Molecular Graph Generation, March 2019. URL http://arxiv.org/abs/1802.04364. arXiv:1802.04364 [cs].

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical Generation of Molecular Graphs using Structural Motifs, April 2020. URL http://arxiv.org/abs/2002.03230. arXiv:2002.03230 [cs].

Jaeyeon Kim, Lee Cheuk-Kit, Carles Domingo-Enrich, Yilun Du, Sham Kakade, Timothy Ngotiaoco, Sitan Chen, and Michael Albergo. Any-Order Flexible Length Masked Diffusion, September 2025. URL http://arxiv.org/abs/2509.01025. arXiv:2509.01025 [cs.LG].

Matt J. Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar Variational Autoencoder, March 2017. URL http://arxiv.org/abs/1703.01925. arXiv:1703.01925 [stat].

Daniel Levy and Jarrid Rector-Brooks. Molecular Fragment-based Diffusion Model for Drug Discovery. May 2023. URL https://openreview.net/forum?id=LPdXni4tC8.

Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient Graph Generation with Graph Recurrent Attention Networks, July 2020. URL http://arxiv.org/abs/1910.00760. arXiv:1910.00760 [cs].

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling, February 2023. URL http://arxiv.org/abs/2210.02747. arXiv:2210.02747 [cs, stat].

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow, September 2022. URL http://arxiv.org/abs/2209.03003. arXiv:2209.03003 [cs].

Xinyang Liu, Yilin He, Bo Chen, and Mingyuan Zhou. Advancing Graph Generation through Beta Diffusion, October 2024. URL http://arxiv.org/abs/2406.09357. arXiv:2406.09357 [cs].

David Lüdke, Marten Lienen, Marcel Kollovieh, and Stephan Günnemann. Edit-Based Flow Matching for Temporal Point Processes, February 2026. URL http://arxiv.org/abs/2510.06050. arXiv:2510.06050 [cs].

Krzysztof Maziarz, Henry Jackson-Flux, Pashmina Cameron, Finton Sirockin, Nadine Schneider, Nikolaus Stiefl, Marwin Segler, and Marc Brockschmidt. Learning to Extend Molecular Scaffolds with Structural Motifs, May 2024. URL http://arxiv.org/abs/2103.03864. arXiv:2103.03864 [cs.LG].

Rocío Mercado, Tobias Rastemo, Edvard Lindelöf, Günter Klambauer, Ola Engkvist, Hongming Chen, and Esben Jannik Bjerrum. Graph networks for molecular design. *Machine Learning: Science and Technology*, 2(2):025023, 2021. doi: 10.1088/2632-2153/abcf91. URL https://doi.org/10.1088/2632-2153/abcf91.

John Nguyen, Marton Havasi, Tariq Berrada, Luke Zettlemoyer, and Ricky T. Q. Chen. OneFlow: Concurrent Mixed-Modal and Interleaved Generation with Edit Flows, December 2025. URL http://arxiv.org/abs/2510.03506. arXiv:2510.03506 [cs].

Matteo Ninniri, Marco Podda, and Davide Bacciu. Graph Diffusion that can Insert and Delete, October 2025. URL http://arxiv.org/abs/2506.15725. arXiv:2506.15725 [cs].

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019. URL http://arxiv.org/abs/1912.01703. arXiv:1912.01703 [cs].

Dhruvesh Patel, Benjamin Rozonoyer, Gaurav Pandey, Tahira Naseem, Ramón Fernandez Astudillo, and Andrew McCallum. Insertion Based Sequence Generation with Learnable Order Dynamics, June 2026. URL http://arxiv.org/abs/2602.18695. arXiv:2602.18695 [cs.LG].

Roman Poletukhin, Marcel Kollovieh, Eike Eberhard, and Stephan Günnemann. 3D Molecule Generation from Rigid Motifs via SE(3) Flows, January 2026. URL http://arxiv.org/abs/2601.16955. arXiv:2601.16955 [cs].

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, December 2020. ISSN 1663-9812. doi: 10.3389/fphar.2020.565644. URL https://www.frontiersin.org/journals/pharmacology/articles/10.3389/fphar.2020.565644/full.

Bo Qiang, Yuxuan Song, Minkai Xu, Jingjing Gong, Bowen Gao, Hao Zhou, Weiying Ma, and Yanyan Lan. Coarse-to-Fine: a Hierarchical Diffusion Model for Molecule Generation in 3D, May 2023. URL http://arxiv.org/abs/2305.13266. arXiv:2305.13266 [q-bio.BM].

Yiming Qin, Manuel Madeira, Dorina Thanou, and Pascal Frossard. DeFoG: Discrete Flow Matching for Graph Generation, June 2025. URL http://arxiv.org/abs/2410.04263. arXiv:2410.04263 [cs].

Raghunathan Ramakrishnan, Pavlo O. Dral, Matthias Rupp, and O. Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1(1):140022, August 2014. ISSN 2052-4463. doi: 10.1038/sdata.2014.22. URL https://www.nature.com/articles/sdata201422.

Machel Reid, Vincent J. Hellendoorn, and Graham Neubig. DiffusER: Discrete Diffusion via Edit-based Reconstruction, October 2022. URL https://arxiv.org/abs/2210.16886v1.

Neta Shaul, Itai Gat, Marton Havasi, Daniel Severo, Anuroop Sriram, Peter Holderrieth, Brian Karrer, Yaron Lipman, and Ricky T. Q. Chen. Flow Matching with General Discrete Paths: A Kinetic-Optimal Perspective, December 2024. URL http://arxiv.org/abs/2412.03487. arXiv:2412.03487 [cs].

Antoine Siraudin, Fragkiskos D. Malliaros, and Christopher Morris. Cometh: A continuous-time discrete-state graph diffusion model, October 2024. URL http://arxiv.org/abs/2406.06449. arXiv:2406.06449 [cs].

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

Zhe Xu, Ruizhong Qiu, Yuzhong Chen, Huiyuan Chen, Xiran Fan, Menghai Pan, Zhichen Zeng, Mahashweta Das, and Hanghang Tong. Discrete-state Continuous-time Diffusion for Graph Generation, November 2024. URL http://arxiv.org/abs/2405.11416. arXiv:2405.11416 [cs].

Pengcheng Yin and Graham Neubig. A Syntactic Neural Model for General-Purpose Code Generation, April 2017. URL http://arxiv.org/abs/1704.01696. arXiv:1704.01696 [cs].

Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5708–5717. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/you18a.html.

Appendix Contents

- Appendix A: Extended Related Work
- Appendix B: Proofs
- Appendix C: General Time Schedules
- Appendix D: Training and Sampling
- Appendix E: Action Space and Trajectory Distribution Instantiations
- Appendix F: Connection to existing methods
- Appendix F.1: Edit Flows as Trajectory Matching
- Appendix G: Additional Experimental Results

A Extended Related Work

Flow and generator matching. Flow matching (Lipman et al., 2023; Albergo & Vanden-Eijnden, 2023; Liu et al., 2022) learns marginal dynamics by regressing against tractable conditional dynamics. Generator Matching (Holderrieth et al., 2024) generalizes this conditional-to-marginal principle to broad classes of Markov processes through their infinitesimal generators, encompassing deterministic flows, diffusions, and continuous-time Markov chains on continuous, discrete, and manifold-valued state spaces. Discrete Flow Matching (DFM) (Campbell et al., 2024; Gat et al., 2024) can be viewed as an instance of this framework on discrete spaces and has been applied to text, proteins, graphs, and molecules (Gat et al., 2024; Campbell et al., 2024; Qin et al., 2025). Widely used DFM constructions define endpoint-conditioned paths on fixed product spaces, factorize the conditional probability paths across components, and allow individual transitions to modify only one categorical component. Within this product-space setting, Shaul et al. (2024) derive kinetic-optimal rates for general categorical probability paths within each component.

Trajectory Matching lies within the Generator Matching framework (Holderrieth et al., 2024), with the complete action trajectory serving as the conditioning variable. Our contribution is a general construction of tractable conditional generators over discrete action spaces without requiring a fixed product-space representation or componentwise actions, together with efficient Rao–Blackwellized targets for several important trajectory families.

Variable-size discrete flows. A few diffusion and flow models relax the fixed-size assumption using specialized size-changing processes. Jump Diffusion (Campbell et al., 2023) combines continuous diffusion with a dimension-reducing deletion process and learns its time reversal, including the rate and transition kernel for inserting new components. GRIDDD (Ninniri et al., 2025) augments graph diffusion with insertions and deletions and an auxiliary model for predicting graph size changes. FlexMDM (Kim et al., 2025) augments masked diffusion for text with a dedicated insertion process: an additional head predicts how many mask tokens to insert between existing positions, while the standard unmasking posterior subsequently predicts their values. Generalized Poisson Flow (Cheng et al., 2026) similarly decomposes variable-size generation into a generalized-Poisson insertion process, which learns the insertion rate and the distribution of newly inserted components, and a within-length flow that refines the current state.

Edit Flows (Havasi et al., 2025), the closest predecessor to our construction, defines a CTMC on variable-length sequences with token insertions, substitutions, and deletions. To obtain tractable conditional dynamics, it lifts sequences to an augmented alignment space, where insertions and deletions are represented as transitions between tokens and a special empty symbol. The resulting conditional probability path factorizes over aligned positions, allowing the standard product-space DFM construction to be applied, with each transition modifying only one component of the augmented state. Notably, trajectory matching recovers the Edit Flow probability path and marginal rate by uniformly ordering the alignment-induced edits for a fixed alignment distribution (Prop. F.1).

Related size-changing constructions have been applied to temporal point processes (Lüdke et al., 2026), proteins (Deutschmann et al., 2026), and 3D molecules (Billera et al., 2026). In particular, Billera et al. (2026) use a specialized atom split-and-deletion process conditioned on a latent forest that determines the ancestry of split atoms. This construction can be seen as a special case of our dependent fixed-action formulation, which allows general action sets and dependency structures.

In summary, these methods provide specialized constructions for domain-specific size-changing operations, rather than a general recipe for training flows with arbitrary multi-component actions and dependencies.

Generation order. Generation order is an important modeling choice in discrete generative models. Autoregressive models typically fix a single generation order, whereas Autoregressive Diffusion Models (Hoogeboom et al., 2022) train over multiple componentwise unmasking orders. Standard factorized DFM similarly permits components to be modified in arbitrary order; in our framework, it corresponds to independent actions ordered uniformly at random. Trajectory Matching makes this trajectory distribution explicit by specifying $q(\tau \mid s_I, s_F)$ over complete domain-level action trajectories, allowing generation orders to encode domain knowledge and dependencies between actions. Jointly learning this trajectory distribution—as recently explored for token insertion orders by LoFlexMDM (Patel et al., 2026)—is a promising direction for future work.

Fragment-based molecular generation. Fragment-based molecular generators construct molecules from chemically meaningful substructures rather than individual atoms. Early approaches include junction-tree and hierarchical variational autoencoders (Jin et al., 2019; 2020) and autoregressive models that select and attach structural motifs (Maziarz et al., 2024). These methods demonstrate that fragment-level representations can improve chemical validity, scalability, and control over molecular structure, but rely on autoregressive or latent-variable constructions.

Fragments have also been incorporated into diffusion and flow-based models. Fragment-based molecular diffusion and flow matching approaches typically generate a fragment graph based on a fixed vocabulary (Levy & Rector-Brooks, 2023; Qiang et al., 2023) and reconstruct atom-level graphs through seperate coarse-to-fine post-processing. Similarly, Poletukhin et al. (2026) generate 3D molecular structures by generating a point cloud of rigid motifs.

These approaches use fragments primarily to define a coarser state representation on which a conventional diffusion or flow process operates. In contrast, our FRAGGROW instantiation retains an atom-level molecular graph and treats attaching an entire fragment as a single structured action, directly producing the updated atom-level graph without a separate coarse-to-fine decoder.

B PROOFS

Throughout the proofs we use the Kolmogorov forward equations, which characterize when a rate u_t generates a probability path p_t. In our action-based parametrization, they take the following form:

Theorem B.1 (Kolmogorov forward equations)**.** *A rate u_t generates a probability path p_t if and only if, for every state s and every $t \in [0, 1]$,*

$$\frac{d}{dt}p_t(s) \;=\; \sum_{\substack{s',\,a:\\ a(s')=s}} p_t(s')\,u_t(s',a) \;-\; p_t(s) \sum_{a\in A(s)} u_t(s,a).$$

Proof. This is the standard Kolmogorov forward equation for CTMCs , specialized to our action parametrization via $Q_t(s', s) = \sum_{a:\, a(s')=s} u_t(s', a)$ for $s' \neq s$. □

B.1 PROOF OF PROPOSITION 3.1

Proposition 3.1 (Trajectory-conditioned path)**.** *Under rate Eq. (3), the generated probability path is*

$$p_t(s \mid \tau) = \sum\nolimits_{i=0}^{n} \mathbf{1}[s = s_i] \cdot \mathbb{P}[K_t = i],$$

where $K_t \sim$ Bin(n, t) counts the number of actions taken by time t.

Proof. Let $\beta_i(t) := \binom{n}{i} t^i (1-t)^{n-i} = \Pr[K_t = i]$. We show that

$$p_t(s \mid \tau) \;=\; \sum_{i=0}^{n} \mathbf{1}[s = s_i]\,\beta_i(t)$$

satisfies the Kolmogorov forward equations of Thm. B.1 with the trajectory-conditioned rate Eq. (3) and initial condition $p_0(\cdot \mid \tau) = \mathbf{1}[\cdot = s_0]$; the claim then follows by uniqueness.

Since the states $s_0, \dots, s_n$ in τ are distinct, $p_t(s \mid \tau) = 0$ for $s \notin \{s_0, \dots, s_n\}$, and the forward equation holds trivially at such s. For $s = s_i$, the only action flowing in is a_i from s_{i-1} (when $i \geq 1$) and the only action with positive outflow rate is a_{i+1} (when $i \leq n-1$). The forward equation thus reduces to

$$\frac{d}{dt}\beta_i(t) \;=\; \frac{n-i+1}{1-t}\,\beta_{i-1}(t) \;-\; \frac{n-i}{1-t}\,\beta_i(t),$$

with the convention $\beta_{-1} \coloneqq \beta_{n+1} \coloneqq 0$. Using the identity $(n-i+1)\binom{n}{i-1} = i\binom{n}{i}$, both sides equal

$$\binom{n}{i}\Big[\, i\, t^{i-1}(1-t)^{n-i} \;-\; (n-i)\, t^i (1-t)^{n-i-1}\Big],$$

which verifies the equation. The initial condition $\beta_i(0) = \mathbf{1}[i=0]$ also holds. □

Trajectories with repeated states. The assumption that a trajectory never revisits a state is only for notational convenience. For repeated states, we additionally condition on the progress index $K_t = i$, distinguishing visits through the augmented state (s_i, i). The next action a_{i+1} then has rate $(n-i)/(1-t)$, with zero outgoing rate at $i = n$. Averaging these rates over the conditional distribution of K_t given $S_t = s$ and τ yields a rate on the original state space. The resulting rate defines a process on the original state space with the same probability of each state at every time. In particular, it still starts at s_I and ends at s_F.

Theorem 3.2. *Let $u_t(s, a \mid \tau)$ be a conditional rate that generates the trajectory-conditioned path $p_t(s \mid \tau)$. Define the marginal probability path and marginal rate as*

$$p_t(s) \coloneqq \mathbb{E}_{\tau\sim q}\; p_t(s \mid \tau), \qquad u_t(s,a) \coloneqq \mathbb{E}_{\tau\sim q}[\, u_t(s,a \mid \tau) \mid S_t = s\,],$$

where $\mathbb{E}_{\tau\sim q}$ is shorthand for $\mathbb{E}_{(s_I,s_F)\sim\gamma}\; \mathbb{E}_{\tau\sim q(\cdot\mid s_I,s_F)}$. Then u_t generates p_t.

Proof. By Thm. B.1, since $u_t(\cdot \mid \tau)$ generates $p_t(\cdot \mid \tau)$ for each τ,

$$\frac{d}{dt}p_t(s \mid \tau) \;=\; \sum_{\substack{s',\,a:\\ a(s')=s}} p_t(s' \mid \tau)\, u_t(s', a \mid \tau) \;-\; p_t(s \mid \tau) \sum_{a\in A(s)} u_t(s, a \mid \tau).$$

Taking $\mathbb{E}_{\tau\sim q}$ on both sides,

$$\frac{d}{dt}p_t(s) \;=\; \sum_{\substack{s',\,a:\\ a(s')=s}} \mathbb{E}_\tau[p_t(s' \mid \tau)\, u_t(s', a \mid \tau)] \;-\; \sum_{a\in A(s)} \mathbb{E}_\tau[p_t(s \mid \tau)\, u_t(s, a \mid \tau)]\,.$$

By the definition of $u_t(s,a)$ as a conditional expectation,

$$\mathbb{E}_\tau[p_t(s \mid \tau)\, u_t(s, a \mid \tau)] \;=\; p_t(s)\, u_t(s,a),$$

where on $\{p_t(s) = 0\}$ both sides vanish and $u_t(s,a)$ may be defined arbitrarily. Substituting yields the Kolmogorov forward equation for p_t with rate u_t, so by Thm. B.1, u_t generates p_t. □

B.2 Proof of Prop. 3.3

Proposition 3.3 (Consistent gradients)**.** *For any parametric family u_t^θ,*

$$\nabla_\theta \mathcal{L}(\theta) \;=\; \nabla_\theta\, \mathbb{E}_t\; \mathbb{E}_{s\sim p_t}\; D\big(u_t(s,\cdot),\, u_t^\theta(s,\cdot)\big)\,.$$

Proof. Expanding $\mathcal{L}(\theta)$ and the marginal matching loss, the claim is

$$\nabla_\theta\, \mathbb{E}_\tau\; \mathbb{E}_t\; \mathbb{E}_{s\sim p_t(\cdot\mid\tau)}\; D\big(u_t(s,\cdot \mid \tau),\, u_t^\theta(s,\cdot)\big) \;=\; \nabla_\theta\, \mathbb{E}_t\; \mathbb{E}_{s\sim p_t}\; D\big(u_t(s,\cdot),\, u_t^\theta(s,\cdot)\big)\,.$$

Since D is a Bregman divergence, $D(a,b) = \phi(a) - \phi(b) - \langle a-b,\, \nabla\phi(b)\rangle$ for some strictly convex ϕ. The terms $\phi(a)$ on each side do not depend on θ and vanish under ∇_θ. The remaining θ-dependent terms are

$$-\phi(u_t^\theta(s,\cdot)) - \langle a - u_t^\theta(s,\cdot),\, \nabla\phi(u_t^\theta(s,\cdot))\rangle,$$

whose θ-gradient is affine in a for fixed (s, t). By Thm. 3.2,

$$\mathbb{E}[u_t(s, \cdot \mid \tau) \mid S_t = s, t] = u_t(s, \cdot).$$

Consequently,

$$\mathbb{E}\big[\nabla_\theta D\big(u_t(s, \cdot \mid \tau),\, u_t^\theta(s, \cdot)\big) \,\big|\, S_t = s, t\big] = \nabla_\theta D\big(u_t(s, \cdot),\, u_t^\theta(s, \cdot)\big)\,.$$

Taking expectations over t and $s \sim p_t$ proves the claim, with differentiation and expectation interchanged under the usual regularity conditions. □

B.3 Proof of Corollary 3.4

Corollary 3.4 (Consistent optima)**.** *If the family u_t^θ contains the marginal rate u_t, then any minimizer of $\mathcal{L}(\theta)$ equals u_t for almost all (s, t) with $p_t(s) > 0$ and generates the marginal probability path p_t.*

Proof. By Prop. 3.3, $\mathcal{L}(\theta)$ and the marginal matching loss have the same gradient in θ, hence the same minimizers. The marginal matching loss is the expectation under $\mathbb{E}_t\, \mathbb{E}_{s\sim p_t}$ of $D(u_t(s, \cdot),\, u_t^\theta(s, \cdot))$. Since D is a Bregman divergence, $D(a, b) \geq 0$ with equality iff $a = b$, so the integrand is pointwise minimized by $u_t^\theta(s, \cdot) = u_t(s, \cdot)$ for almost every (s, t) with $p_t(s) > 0$.

If the family contains u_t, this pointwise minimum is attained, so any minimizer $\theta^\star$ satisfies $u_t^{\theta^\star}(s, \cdot) = u_t(s, \cdot)$ for almost every (s, t) with $p_t(s) > 0$. Consequently, the probability fluxes $p_t(s)u_t^{\theta^\star}(s, a)$ and $p_t(s)u_t(s, a)$ agree almost everywhere in time. Therefore, p_t also satisfies the Kolmogorov forward equations, in integral form, for $u_t^{\theta^\star}$. Since both processes start from p_{src}, uniqueness implies that $u_t^{\theta^\star}$ generates the same marginal path p_t. □

B.4 Rao–Blackwellization: consistency and variance reduction

We establish the guarantees for the Rao–Blackwellized training target introduced in § 3.3. Averaging over trajectories conditional on the current state, time, and endpoints preserves the expected gradient and the optimum of the trajectory-matching objective, while reducing gradient variance.

Proposition B.2 (Consistency of the Rao-Blackwellized loss)**.** *Prop. 3.3 and Corollary 3.4 continue to hold when $u_t(\cdot \mid \tau)$ is replaced by $\bar{u}_t(\cdot \mid s_I, s_F)$ in Eq. (4).*

Proof. By the tower property, $u_t(\cdot \mid \tau)$ and $\bar{u}_t(\cdot \mid s_I, s_F)$ share the same conditional expectation given $S_t = s$, namely the marginal rate $u_t(s, \cdot)$. The proof of Prop. 3.3 uses $u_t(\cdot \mid \tau)$ only through this conditional expectation, so the same gradient identity holds with $\bar{u}_t(\cdot \mid s_I, s_F)$ in place of $u_t(\cdot \mid \tau)$. Corollary 3.4 then follows by the same argument. □

Proposition B.3 (Variance reduction)**.** *Let $\hat{g}_{TM}(\theta) = \nabla_\theta D(u_t(s, \cdot \mid \tau), u_t^\theta(s, \cdot))$ and $\hat{g}_{RB}(\theta) = \nabla_\theta D(\bar{u}_t(s, \cdot \mid s_I, s_F), u_t^\theta(s, \cdot))$ be the gradient estimators under the trajectory-conditioned and Rao-Blackwellized targets, sampled from the distribution defining $\mathcal{L}(\theta)$. Then*

$$\mathbb{E}[\hat{g}_{RB}] = \mathbb{E}[\hat{g}_{TM}] \quad \textit{and} \quad \mathrm{Var}(\hat{g}_{RB}) \leq \mathrm{Var}(\hat{g}_{TM}) \textit{ componentwise}.$$

Proof. Since D is a Bregman divergence, $\nabla_\theta D(a, u_t^\theta(s, \cdot))$ is linear in a. Therefore, conditioning on (s, t, s_I, s_F) and using the definition of the Rao-Blackwellized target $\bar{u}_t(s, a \mid s_I, s_F) = \mathbb{E}[u_t(s, a \mid \tau) \mid s, t, s_I, s_F]$,

$$\mathbb{E}[\hat{g}_{\text{TM}} \mid s, t, s_I, s_F] \;=\; \nabla_\theta D\big(\mathbb{E}[u_t(s, \cdot \mid \tau) \mid s, t, s_I, s_F],\, u_t^\theta(s, \cdot)\big) \;=\; \hat{g}_{\text{RB}}.$$

Equality of expectations follows by the tower property: $\mathbb{E}[\hat{g}_{\text{TM}}] = \mathbb{E}[\mathbb{E}[\hat{g}_{\text{TM}} \mid s, t, s_I, s_F]] = \mathbb{E}[\hat{g}_{\text{RB}}]$. The variance inequality follows from the law of total variance applied componentwise:

$$\mathrm{Var}(\hat{g}_{\text{TM}}) \;=\; \mathbb{E}\big[\mathrm{Var}(\hat{g}_{\text{TM}} \mid s, t, s_I, s_F)\big] + \mathrm{Var}\big(\mathbb{E}[\hat{g}_{\text{TM}} \mid s, t, s_I, s_F]\big) \;\geq\; \mathrm{Var}(\hat{g}_{\text{RB}}). \qquad \square$$

B.5 Proof of Prop. 4.3

Proposition 4.3. *Let q be an independent fixed-action distribution. Then for a state s on a trajectory from s_I to s_F we have: $\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \alpha(s, s_F)] \cdot 1/(1-t)$.*

Proof. Fix s_I, s_F and a state s on some trajectory from s_I to s_F in the support of q. Let $m := |\alpha(s, s_F)|$ be the number of actions remaining at s, and let $\ell := |\alpha(s_I, s_F)| - m$ be the number of actions already taken to reach s. Since q is an independent fixed-action distribution, every trajectory passing through s on its way to s_F takes the same set $\alpha(s, s_F)$ of remaining actions; only their ordering varies, and the distribution over orderings is uniform.

By symmetry, the posterior $p_t(\tau \mid s, s_I, s_F)$ is uniform over orderings of $\alpha(s, s_F)$, so each $a \in \alpha(s, s_F)$ is equally likely to be the next action: $\Pr[a_{\ell+1} = a \mid s, s_I, s_F] = 1/m$.

The trajectory-conditioned rate at s (position ℓ on a trajectory of length $\ell + m$) is, by Eq. (3),

$$u_t(s, a \mid \tau) = \mathbf{1}[a = a_{\ell+1}] \cdot \frac{m}{1-t}.$$

Taking the posterior expectation,

$$\begin{aligned} \bar{u}_t(s, a \mid s_I, s_F) &= \mathbb{E}_{\tau \sim p_t(\cdot \mid s, s_I, s_F)}\, u_t(s, a \mid \tau) \\ &= \mathbf{1}[a \in \alpha(s, s_F)] \cdot \frac{1}{m} \cdot \frac{m}{1-t} \\ &= \mathbf{1}[a \in \alpha(s, s_F)] \cdot \frac{1}{1-t}. \end{aligned}$$

□

B.6 Proof of Thm. 4.4

Theorem 4.4. *Consider a factorized flow as defined in Eq. (2) with single component actions $a_v^i(s) = (s^1, \ldots, v, \ldots, s^d)$. Let q be the corresponding independent fixed action distribution with $\alpha(s_I, s_F) = \{a_v^i : v = s_F^i \neq s_I^i\}$. Then:*

1. *The trajectory-matching loss (Eq. (4)) and the factorized flow matching loss (Eq. (1)) have the same gradient in θ and hence the same minimizers.*
2. *The Rao-Blackwellized loss (Prop. 4.3, Prop. B.2) is identical to the factorized loss (Eq. (1)).*

Proof. We first prove Part 2; Part 1 then follows from Prop. B.2. **Part 2.** By Prop. 4.3, the Rao-Blackwellized rate is

$$\bar{u}_t(s, a_v^i \mid s_I, s_F) = \mathbf{1}[a_v^i \in \alpha(s, s_F)] \cdot \frac{1}{1-t} = \mathbf{1}[s^i \neq v = s_F^i] \cdot \frac{1}{1-t},$$

which is exactly the factorized flow rate from Eq. (2). It remains to check that the sampling distribution over (s, t, s_I, s_F) agrees in the two losses. Both draw $(s_I, s_F) \sim \gamma$ and $t \sim U[0, 1]$; it suffices to show the conditional paths agree, i.e.,

$$\mathbb{E}_{\tau \sim q(\cdot \mid s_I, s_F)}\, p_t(s \mid \tau) = \prod_{i=1}^{d} p_t^i(s^i \mid s_I^i, s_F^i).$$

Fix (s_I, s_F) and let $F = \{i : s_I^i \neq s_F^i\}$ with $n_F = |F|$. Every τ in the support of $q(\cdot \mid s_I, s_F)$ is an ordering of $\{a_{s_F^i}^i : i \in F\}$. By Prop. 3.1, conditional on τ and $K_t = k$, the current state has flipped exactly the components corresponding to the first k actions in τ. Averaging over τ uniformly over orderings, the identity of the k flipped components is uniform over size-k subsets of F. Since $K_t \sim \mathrm{Bin}(n_F, t)$, this is equivalent to flipping each $i \in F$ independently with probability t, which yields the factorized path $\prod_{i=1}^{d} p_t^i(s^i \mid s_I^i, s_F^i)$.

Part 1. Follows from Part 2 and Prop. B.2. □

B.7 Rao-Blackwellized rates for general dependent fixed-action distributions

We will first prove a more general version of Prop. 4.7, which applies to any dependent fixed-action distribution, not just those with forest-structured dependency DAGs.

Let $L(G)$ denote the number of linear extensions (topological orderings) of a DAG G, and let $G \setminus a$ denote the DAG obtained by removing vertex a together with its incident edges.

Proposition B.4. *Let q be a dependent fixed-action distribution with dependency DAG on $\alpha(s_I, s_F)$. For a state s on a trajectory from s_I to s_F,*

$$\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \text{roots}(\alpha(s, s_F))] \cdot \frac{L(\alpha(s, s_F) \setminus a)}{L(\alpha(s, s_F))} \cdot \frac{m}{1-t},$$

where $m = |\alpha(s, s_F)|$ is the number of remaining actions.

Proof. Fix s_I, s_F and a state s on a trajectory in the support of $q(\cdot \mid s_I, s_F)$. Let $\ell = |\alpha(s_I, s_F)| - m$ be the number of actions already taken to reach s, so every trajectory through s has length $\ell + m$ and reaches s at position ℓ.

Since q is uniform over topological orderings of $\alpha(s_I, s_F)$, the posterior $p_t(\tau \mid s, s_I, s_F)$ is uniform over those orderings whose first ℓ actions reach s. Each such ordering decomposes into a prefix of length ℓ (an ordering of the actions already taken) and a suffix of length m (a topological ordering of $\alpha(s, s_F)$). Marginalizing out the prefix, the suffix is uniform over linear extensions of $\alpha(s, s_F)$. Hence for any root a of $\alpha(s, s_F)$,

$$\Pr[a_{\ell+1} = a \mid s, s_I, s_F] = \frac{L(\alpha(s, s_F) \setminus a)}{L(\alpha(s, s_F))},$$

since orderings with a first are in bijection with linear extensions of $\alpha(s, s_F) \setminus a$. For non-roots this probability is zero.

The trajectory-conditioned rate at s is, by Eq. (3), $u_t(s, a \mid \tau) = \mathbf{1}[a = a_{\ell+1}] \cdot m/(1-t)$. Taking the posterior expectation,

$$\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \text{roots}(\alpha(s, s_F))] \cdot \frac{L(\alpha(s, s_F) \setminus a)}{L(\alpha(s, s_F))} \cdot \frac{m}{1-t}. \qquad \square$$

We note that counting linear extensions of a general DAG is computationally intractable in general, so Prop. B.4 does not yield an efficient training target in general. However, if the dependency DAG is a forest, the linear-extension ratio admits a closed form, recovering:

Proposition 4.7. *Let q be a dependent fixed-action distribution whose dependency DAG $\alpha(s_I, s_F)$ is a forest. Then for a state s on a trajectory from s_I to s_F we have:*

$$\bar{u}_t(s, a \mid s_I, s_F) = \mathbf{1}[a \in \text{roots}(\alpha(s, s_F))] \cdot \text{desc}_{\alpha(s, s_F)}(a) \cdot 1/(1-t)$$

where roots *is the set of roots of the DAG and* $\text{desc}(a)$ *is the size of the tree rooted at a (a included).*

Proof. By Prop. B.4, it suffices to show that for any root a of a forest G on m vertices, the probability that a comes first in a uniform linear extension of G equals $\text{desc}(a)/m$.

Let $T_1, \ldots, T_r$ be the trees of G with roots $\rho_1, \ldots, \rho_r$ and sizes $m_1, \ldots, m_r$. Any linear extension of G is uniquely determined by a linear extension of each T_j together with an interleaving specifying, for each of the m output positions, which tree contributes that position. The interleaving is a sequence in $[r]^m$ containing m_j copies of j, and conversely every such interleaving combined with linear extensions of the T_j yields a valid linear extension of G. Under a uniform linear extension of G, both the per-tree extensions and the interleaving are independently uniform.

By symmetry over the m positions, the interleaving assigns position 1 to tree T_j with probability m_j/m. Conditional on position 1 coming from T_j, the first vertex is the minimum of T_j's linear extension, which is the root ρ_j (the unique minimum of any rooted tree's partial order). Hence

$$\Pr[\rho_j \text{ first}] = \frac{m_j}{m} = \frac{\text{desc}(\rho_j)}{m}.$$

Since a is the root of some T_j, substituting into Prop. B.4 gives the claim. $\square$

For the forest-plus-sink dependencies in AtomGrow and MolEdit, App. E.7 gives an alternative conditional path with exact tractable rates.

C General Time Schedules

Factorized flows are commonly used with a general time schedule $\kappa : [0,1] \to [0,1]$ satisfying $\kappa_0 = 0$, $\kappa_1 = 1$, and $\dot{\kappa}_t > 0$. The conditional rate of Eq. (2) then takes the form

$$u_t(s, a \mid s_I, s_F) \;=\; \mathbf{1}[a^i(s^i) = s_F^i] \cdot \frac{\dot{\kappa}_t}{1 - \kappa_t}, \tag{6}$$

which generates the factorized path with marginals $\kappa_t \mathbf{1}[s^i = s_F^i] + (1 - \kappa_t)\mathbf{1}[s^i = s_I^i]$. This general schedule is used, for example, by Edit Flows (Havasi et al., 2025) and schedules like the polydec schedule have been shown to outperform the linear schdedule for some tasks (Qin et al., 2025).

Reinterpretation as a time reparametrization. The factor $\dot{\kappa}_t/(1 - \kappa_t)$ is precisely what one obtains by substituting $t \mapsto \kappa_t$ in the linear-schedule rate $1/(1 - t)$ and applying the chain rule. More generally, if u_t generates a probability path p_t on $[0, 1]$, then by the chain rule applied to the Kolmogorov forward equation (Thm. B.1),

$$\tilde{u}_t(s, a) := \dot{\kappa}_t \cdot u_{\kappa_t}(s, a) \quad \text{generates} \quad \tilde{p}_t(s) := p_{\kappa_t}(s),$$

and endpoints are preserved since $\kappa_0 = 0$ and $\kappa_1 = 1$. Applying this transformation to the linear-schedule trajectory rate of Eq. (3) yields the natural κ-scheduled trajectory rate.

κ-scheduled trajectory rate. Probabilistically, this corresponds to drawing the action execution times $0 < t_1 < \cdots < t_n < 1$ as the order statistics of n i.i.d. random variables with CDF κ_t (rather than $\mathrm{U}[0, 1]$). The resulting trajectory-conditioned rate is

$$u_t(s, a \mid \tau) \;=\; \sum_{i=0}^{n-1} \mathbf{1}[s = s_i \wedge a = a_{i+1}] \cdot (n - i) \cdot \frac{\dot{\kappa}_t}{1 - \kappa_t}, \tag{7}$$

which generates the conditional path

$$p_t(s \mid \tau) \;=\; \sum_{i=0}^{n} \mathbf{1}[s = s_i] \cdot \mathbb{P}[K_t = i], \qquad K_t \sim \mathrm{Bin}(n, \kappa_t). \tag{8}$$

The linear schedule $\kappa_t = t$ recovers Eq. (3) and Prop. 3.1.

All results extend. Because the schedule enters Eq. (7) only through the scalar prefactor $\dot{\kappa}_t/(1-\kappa_t)$, every result in §§ 3 and 4 carries over by replacing $1/(1 - t)$ with $\dot{\kappa}_t/(1 - \kappa_t)$ throughout. The κ schedule slots into trajectory matching exactly as it does into factorized flows, without changing the structure of any proof.

D Training and Sampling

Algorithms 1 and 2 give the training and sampling procedures for trajectory matching. We note a few practical considerations.

Training. The training loop is written in terms of the trajectory τ to make the connection to the theory explicit, but the trajectory itself does not necessarily need to be materialized. In practice, we use a single sampler that takes $(s_I, s_F) \sim \gamma$ and directly returns a tuple $(t, s, u_t(s, \cdot \mid \tau))$ — or $(t, s, \bar{u}_t(s, \cdot \mid s_I, s_F))$ when using the Rao-Blackwellized target. Depending on the chosen trajectory distribution, the sampler can often be efficiently implemented without explicitly constructing the full trajectory τ. For most instantiations of interest, the target rate at s depends only on quantities such as the remaining action set $\alpha(s, s_F)$ or the residual dependency DAG, which can be computed directly from the endpoints and the sampled state.

Because the sampling routine is independent of the model, it is well suited to run in CPU dataloader workers in parallel with the forward and backward passes on the GPU. In our QM9 experiments, the observed loader wait is below 1 ms per training step, so sampling and target construction add very little exposed wall-clock time (App. G.1).

Algorithm 1. Training procedure for trajectory matching

Require: Model u_t^θ, coupling γ, trajectory distribution q

while u_t^θ not converged **do**
  Sample $(s_I, s_F) \sim \gamma$
  Sample $\tau \sim q(\cdot \mid s_I, s_F)$
  Sample time $t \in [0, 1]$
  Sample $s_t \sim p_t(\cdot \mid \tau)$
  $\mathcal{L} \leftarrow D\big(u_t(s_t, \cdot \mid \tau), u_t^\theta(s_t, \cdot)\big)$
  Update θ with optimizer step on $\mathcal{L}$

Algorithm 2. Sampling procedure for trajectory matching

Require: Trained model u_t^θ, source distribution p_{src}, number of Euler steps N

$h \leftarrow 1/N$
Sample $s \sim p_{\text{src}}$ ▷ initial state
for $k = 0, 1, \ldots, N-1$ **do**
  $t \leftarrow k\,h$
  $\mathcal{B} \leftarrow \emptyset$
  for each available action $a \in \mathcal{A}(s)$ **do**
    With prob. $\min\{u_t^\theta(s, a) \cdot h,\, 1\}$: $\mathcal{B} \leftarrow \mathcal{B} \cup \{a\}$
  end for
  Apply actions in $\mathcal{B}$ to s
end for
return s

Figure 4: Pseudocode for training and sampling in trajectory matching.

Sampling. At each Euler step, the inner loop over $\mathcal{A}(s)$ in Algorithm 2 can be executed in parallel: each available action is independently retained with probability $\min\{u_t^\theta(s, a) \cdot h, 1\}$, and these draws are independent Bernoullis. In most of our settings the sampled actions are also *compatible* — they can be applied simultaneously to s without conflict — either by design of the action space or as the typical case. For BONDEDIT, ATOMGROW, FRAGMENTGROW, and GROWHIERARCHY, every action in $A(s)$ is independent of every other, so any subset can be applied in parallel and the result does not depend on the application order.

For MOLEDIT, the action space mixes deletions and additions and a sampled subset can occasionally be incompatible — for example, an `AddBond` action targeting an atom that another sampled `DelAtom` action removes. In practice this is rare, and we resolve it deterministically by applying all deletions before all additions, and skipping any action that is no longer in $A(s)$ by the time it would be applied. This keeps the parallel structure of the inner loop while guaranteeing a well-defined update.

E ACTION SPACE AND TRAJECTORY DISTRIBUTION INSTANTIATIONS

In this appendix we give precise definitions of the action spaces, source distributions, couplings, and trajectory distributions for the four settings visualized in Fig. 1: BONDEDIT, FRAGMENTGROW, ATOMGROW, and MOLEDIT. We deliberately abstract away from atom and bond features in this beginning of this section to simplify the expostion and treat all graphs as unlabeled. The extension to feature-labeled graphs is discussed subsequently.

Atom-id correspondence. Throughout, we identify the correspondence between atoms in s_I and s_F by index. Atoms with the same index are assumed to transform into each other along a training trajectory. Thus, the atom ids given by the coupling γ decide what unambiguously what atom is "kept", "deleted", or "added", and what bond is "already present in s_I" or "required in s_F". This is the standard setting in discrete graph diffusion and flow matching approaches (Qin et al., 2025; Vignac et al., 2023).

We represent a (partial) molecular graph as $s = (V(s), E(s))$ with $V(s)$ a set of atom ids and $E(s)$ a set of unordered pairs $\{i, j\} \subseteq V(s)$.

Two recurring source distributions. Two source distributions appear repeatedly across the four settings:

- *Single-atom source* ($p_{\text{src}}^{\text{atom}}$): samples a graph $(\{i_0\}, \emptyset)$ consisting of a single atom. Used by FRAGMENTGROW and ATOMGROW.
- *Marginal-graph source* ($p_{\text{src}}^{\text{marg}}$): first samples the number of atoms n from the empirical distribution of $|V(\cdot)|$ in the training set, then samples each potential edge $\{i, j\}$ independently with probability $\bar{p}_{\text{edge}}$, the average edge density in the training set. Used by BONDEDIT and MOLEDIT.

E.1 BONDEDIT

BONDEDIT is the simplest setting: the atom set is fixed by the coupling and only edges are inserted or deleted. It mirrors the action space of DeFoG (Qin et al., 2025).

States. For every (s_I, s_F) in the support of γ we have $V(s_I) = V(s_F)$. Every intermediate state s on a trajectory satisfies $V(s) = V(s_I) = V(s_F)$.

Source distribution. We use the marginal-graph source $p_{\text{src}} = p_{\text{src}}^{\text{marg}}$.

Coupling. γ is the size-matched coupling: $s_I \sim p_{\text{src}}^{\text{marg}}$ and s_F is sampled uniformly from training molecules with $|V(s_F)| = |V(s_I)|$, with atom ids matched arbitrarily (e.g. in the order they appear in each graph).

Action space. At a state s, the available actions are

$$\mathcal{A}(s) \;=\; \big\{\, \texttt{AddBond}_{ij} \;:\; \{i,j\} \notin E(s) \,\big\} \;\cup\; \big\{\, \texttt{DelBond}_{ij} \;:\; \{i,j\} \in E(s) \,\big\},$$

with $\texttt{AddBond}_{ij}(s) = (V(s), E(s) \cup \{\{i,j\}\})$ and $\texttt{DelBond}_{ij}(s) = (V(s), E(s) \setminus \{\{i,j\}\})$.

Required actions. The set of actions transforming s into s_F is

$$\alpha(s, s_F) \;=\; \big\{\, \texttt{AddBond}_{ij} : \{i,j\} \in E(s_F) \setminus E(s) \,\big\} \cup \big\{\, \texttt{DelBond}_{ij} : \{i,j\} \in E(s) \setminus E(s_F) \,\big\}.$$

So the adding all bonds that are still missing and deleting all bonds that are not present in s_F.

Trajectory distribution. Any two actions in $\alpha(s_I, s_F)$ act on disjoint edges, so they are pairwise independent in the sense of § 2. We take $q(\cdot \mid s_I, s_F)$ to be the *independent fixed-action distribution* (Definition 4.2) over $\alpha(s_I, s_F)$ and use the Rao-Blackwellized rates of Prop. 4.3.

E.2 FRAGMENTGROW

FRAGMENTGROW produces a molecule from a single seed atom by attaching fragments which capture all ring substructures and individual atoms. We assume a fixed fragment library $\mathcal{F}$ (rings and ring systems extracted from the training set; in practice $|\mathcal{F}| < 300$). Molecules whose ring system cannot be expressed using $\mathcal{F}$ are skipped during training. The fragment vocabulary covers 94% of QM9, 99% of ZINC, and 90% of MOSES training molecules; the remaining 6%, 1%, and 10%, respectively, are skipped during FragGrow training. Validation and test sets are not filtered: all evaluations use the complete standard reference splits, including molecules not representable by the fragment vocabulary.

States. The source s_I is a single-atom graph $(\{i_0\}, \emptyset)$ for some seed atom i_0 that is also present in s_F. Every intermediate state is a connected subgraph of s_F containing i_0.

Source distribution. We use the single-atom source $p_{\text{src}} = p_{\text{src}}^{\text{atom}}$.

Coupling. s_F is sampled from the training distribution and the seed atom i_0 is sampled uniformly from $V(s_F)$.

Action space. At a state s, the available actions are

- $\texttt{AddAtom}_i$ for each $i \in V(s)$, which adds a new atom $j \notin V(s)$ together with a single bond $\{i,j\}$ to the parent atom i.
- $\texttt{AddFragment}_i^F$ for each $i \in V(s)$ and each fragment $F \in \mathcal{F}$ that admits an atom-attachment, which inserts a copy of F joined to s at atom i.
- $\texttt{AddFragment}_{ij}^F$ for each $\{i,j\} \in E(s)$ and each fragment $F \in \mathcal{F}$ that admits a bond-attachment, which inserts a copy of F joined to s along the bond $\{i,j\}$.

Each of these actions strictly grows the state.



(a) Decomposition (b) Dependency tree

Figure 5: FragmentGrow decompostion and dependency tree: Given the decomposition of s_F into fragments and atom/bonds, there is one unique set of actions to generate s_F from s_I (marked in red), and the dependencies between these actions form a tree.

Decomposition. Given s_F and a chosen root atom $i_0 \in V(s_F)$, there is a *unique hierarchical decomposition* of s_F into a structural action tree, computed as follows:

1. Initialize the ring units as the SSSR (smallest set of smallest rings) rings of s_F.
2. Repeatedly merge ring units whose overlap is more than a single shared atom or a single shared bond, and merge any remaining cycle in the ring-overlap graph into a single larger ring fragment. After this step, distinct ring fragments share at most one atom or one bond.
3. Build the structural action tree by a BFS-like traversal from i_0: non-ring bonds become `AddAtom` children, and ring fragments incident to the current frontier become `AddFragment` children attached either at an atom (if the ring shares one atom with the parent) or at a bond (if the ring shares one bond with the parent).

Required actions. The structural action tree above determines $\alpha(s_I, s_F)$ (one action per non-root unit of the decomposition). The structural action tree also indicated the dependency relationships. This procedure is the key to obtaining a tree of actions: because each ring system is absorbed into a single atomic action, no action has two prerequisites and there is one unique set of actions to generate s_F from s_I.

Trajectory distribution. We take $q(\cdot \mid s_I, s_F)$ to be the *dependent fixed-action distribution* (Definition 4.6) over $\alpha(s_I, s_F)$ with the dependency tree described above. Then, we can derive the Rao-Blackwellized rates efficiently and use for training

E.3 AtomGrow

AtomGrow also grows from a single seed atom, but uses only single-atom and single-bond actions; rings are produced by `AddBond` actions that close cycles between previously inserted atoms.

States. As in FragmentGrow, $s_I = (\{i_0\}, \emptyset)$ for a seed atom $i_0 \in V(s_F)$, and every intermediate state is a connected subgraph of s_F containing i_0.

Source distribution. We use the single-atom source $p_{\text{src}} = p_{\text{src}}^{\text{atom}}$.

Coupling. As in FragmentGrow: s_F is sampled from the training distribution, i_0 uniformly from $V(s_F)$, and $s_I = (\{i_0\}, \emptyset)$.

Action space. At state s,

$$\mathcal{A}(s) = \{\texttt{AddAtom}_i : i \in V(s)\} \cup \{\texttt{AddBond}_{ij} : i, j \in V(s),\ \{i, j\} \notin E(s)\}.$$

$\texttt{AddAtom}_i$ adds a new atom j together with the bond $\{i, j\}$; $\texttt{AddBond}_{ij}$ inserts the bond $\{i, j\}$ between two existing atoms.

Latent decomposition. Different orderings of `AddAtom` / `AddBond` actions can yield the same final molecule via genuinely different action sets — for example, the same edge of a ring may be

inserted either by an AddAtom that introduces its second endpoint or by an AddBond that closes the cycle. To resolve this, we introduce a latent

$$z \;=\; T,$$

a rooted spanning tree of s_F rooted at the seed atom i_0. We use the factored form of § 4 with $p(z \mid s_I, s_F)$ a distribution over rooted spanning trees of s_F with root i_0. In practice we use for z a random BFS tree of s_F rooted at i_0, which can be sampled efficiently and yields good results.

Required actions given z. Given $z = T$, the action set decomposes into two groups:

- For each non-root atom $v \in V(s_F) \setminus \{i_0\}$, an action $\texttt{AddAtom}_{\text{parent}_T(v)}$ that produces v.
- For each non-tree edge $\{i, j\} \in E(s_F) \setminus E(T)$, an action $\texttt{AddBond}_{ij}$.

Dependency structure. The AddAtom actions form a tree mirroring T: $\texttt{AddAtom}_{\text{parent}_T(v)}$ is a prerequisite of $\texttt{AddAtom}_{\text{parent}_T(w)}$ whenever $\text{parent}_T(w) = v$. Each $\texttt{AddBond}_{ij}$ has *two* prerequisites — the AddAtom actions producing i and j — and has no successors.

Trajectory distribution. Conditional on z, we use the forest-plus-sink construction of App. E.7 for the DAG above. It specifies the trajectory distribution $q(\cdot \mid z, s_I, s_F)$ together with the conditional timing law and exact rates.

Multi-root variant. The construction extends naturally to sources s_I consisting of k disjoint seed atoms (used in our substructure-linking experiments, App. G.6). The coupling samples s_F from the training distribution, runs BFS from k random seeds $i_1, \ldots, i_k \in V(s_F)$ to partition $V(s_F)$ into k components, and sets $s_I = (\{i_1, \ldots, i_k\}, \emptyset)$. The latent z becomes a rooted spanning forest of s_F with one tree per seed; everything else is unchanged. The choice of k (or a distribution over k) can be set flexibly. Empirically, we found that this matches single-root performance on standard generation while enabling zero-shot transfer to downstream tasks like substructure linking.

E.4 MolEdit

MolEdit is the most general setting: the source s_I is itself a (noisy) molecule, and trajectories edit it into s_F using both deletions and additions of edges *and* nodes.

States. s_I and s_F are molecular graphs related by the atom-id correspondence: an atom $i \in V(s_I)$ is *kept* if $i \in V(s_F)$ and *deleted* otherwise. We write

$$V_{\text{kept}} = V(s_I) \cap V(s_F), \quad V_{\text{del}} = V(s_I) \setminus V(s_F), \quad V_{\text{new}} = V(s_F) \setminus V(s_I),$$

and assume $V_{\text{kept}} \neq \emptyset$.

Source distribution. We use the marginal-graph source $p_{\text{src}} = p_{\text{src}}^{\text{marg}}$.

Coupling. The base coupling γ is independent: $s_I \sim p_{\text{src}}^{\text{marg}}$ and $s_F \sim p_{\text{tgt}}$ are drawn independently. In practice, however, we found it beneficial to maximize the edge overlap between s_I and s_F to reduce the number of required edits. Concretely, after sampling $(s_I, s_F) \sim p_{\text{src}}^{\text{marg}} \otimes p_{\text{tgt}}$, we relabel the atoms of s_I by greedily aligning them to the atoms of s_F so as to maximize the size of the edge overlap $|E(s_I) \cap E(s_F)|$ (under the resulting atom-id correspondence). This relabeling does not change p_{src} or p_{tgt}, but it concentrates γ on endpoint pairs with shorter trajectories.

Action space. At state s,

$$\begin{aligned}\mathcal{A}(s) \;=\; &\{\texttt{AddAtom}_i : i \in V(s)\} \;\cup\; \{\texttt{AddBond}_{ij} : i, j \in V(s),\, \{i, j\} \notin E(s)\} \\ &\cup\; \{\texttt{DelBond}_{ij} : \{i, j\} \in E(s)\} \;\cup\; \{\texttt{DelAtom}_i : i \in V(s),\, \deg_s(i) = 0\}.\end{aligned}$$

$\texttt{DelAtom}_i$ is only available once atom i has no incident bonds.

Latent decomposition. We introduce a latent

$$z = T,$$

a rooted spanning forest of s_F whose roots are exactly V_{kept}. Each component contains one kept atom, and every new atom $v \in V_{\text{new}}$ has a unique parent $\text{parent}_T(v)$ to which it attaches when inserted. Thus, the forest edges $E(T)$ are exactly the bonds created by atom additions. Such a forest can be constructed by a multi-source BFS initialized with all kept atoms.

Required actions given z. Given $z = T$, $\alpha(s_I, s_F \mid z)$ consists of:

- **Bond deletions.** $\texttt{DelBond}_{ij}$ for every $\{i,j\} \in E(s_I)$ that is either incident to a deleted atom ($\{i,j\} \cap V_{\text{del}} \neq \emptyset$) or is a kept-kept bond not in s_F ($\{i,j\} \subseteq V_{\text{kept}}$ and $\{i,j\} \notin E(s_F)$).
- **Atom deletions.** $\texttt{DelAtom}_i$ for every $i \in V_{\text{del}}$.
- **Atom additions.** $\texttt{AddAtom}_{\text{parent}_T(v)}$ for every $v \in V_{\text{new}}$.
- **Bond additions.** $\texttt{AddBond}_{ij}$ for every

  $$\{i,j\} \in \big(E(s_F) \setminus E(s_I)\big) \setminus E(T).$$

  These are all missing target bonds except those created by atom additions, including missing bonds between kept atoms and bonds connecting different forest components.

Dependency structure. The dependencies are:

- For each $i \in V_{\text{del}}$, every $\texttt{DelBond}_{ij}$ incident to i is a prerequisite of $\texttt{DelAtom}_i$.
- For each $v \in V_{\text{new}}$, the action inserting v depends on the action inserting $\text{parent}_T(v)$ if that parent is also new; otherwise, it has no prerequisites.
- Each required $\texttt{AddBond}_{ij}$ depends on the atom-insertion action for each endpoint in V_{new}. Endpoints in V_{kept} introduce no prerequisites.

The atom-insertion dependencies form a forest rooted at insertions whose parents are kept atoms. Bond additions and atom deletions have no successors, so the resulting action DAG retains the forest-plus-sink structure.

Trajectory distribution. Conditional on z, we use the forest-plus-sink construction of App. E.7 for the DAG above, which specifies the trajectory distribution, conditional timing law, and exact rates.

E.5 GrowHierarchy

To illustrate that trajectory matching applies beyond molecular graph generation, we include a non-molecular setting: generating rooted trees by repeatedly attaching children to existing nodes. This is the simplest non-trivial instance of a forest-structured dependent fixed-action distribution — a single action type, no latent, and a closed-form Rao-Blackwellized rate from Prop. 4.7.

States. A state is a rooted tree $s = (V(s), E(s), r)$ with $V(s)$ a set of node ids, $E(s) \subseteq V(s) \times V(s)$ a set of directed parent-to-child edges, and $r \in V(s)$ the root. The intermediate states along a trajectory are subtrees of s_F containing its root.

Source distribution. p_{src} samples a single-node tree $(\{r\}, \emptyset, r)$. If the training distribution is over rooted trees, r is the designated root of the sampled s_F; otherwise r is sampled uniformly from $V(s_F)$.

Coupling. s_F is sampled from the training distribution and s_I is the single-node tree at its root.

Action space. At a state s,

$$\mathcal{A}(s) = \{\texttt{AddChild}_i : i \in V(s)\},$$

where $\texttt{AddChild}_i$ adds a new node $j \notin V(s)$ together with the directed edge (i,j).

Required actions. The set of actions transforming s_I into s_F is

$$\alpha(s_I, s_F) \;=\; \big\{\, \texttt{AddChild}_{\text{parent}_{s_F}(v)} \;:\; v \in V(s_F) \setminus \{r\} \,\big\},$$

i.e. one action per non-root node of s_F.

Dependency structure. The dependency DAG is exactly s_F: the action producing v is a prerequisite of the action producing any child of v. This DAG is a tree (in particular a forest), so Prop. 4.7 applies.

Trajectory distribution. $q(\cdot \mid s_I, s_F)$ is the dependent fixed-action distribution over $\alpha(s_I, s_F)$, uniform over the topological orderings of the tree.

E.6 Handling Node and Edge Features

The instantiations above abstract away from node and edge features (atom types, bond orders, charges, etc.). Two natural strategies extend them to feature-labeled graphs.

Strategy A: joint prediction at insertion. Insertion actions are parameterized by features: $\texttt{AddAtom}_i(v)$ inserts a new atom with feature v in a single action, and analogously for $\texttt{AddBond}_{ij}(b)$ and `AddFragment`. The action vocabulary grows by a factor of the feature alphabet, but no separate feature-edit actions are needed.

Strategy B: separate feature-edit actions. Insertion actions produce nodes and bonds with a random feature drawn from a fixed distribution; features are then modified by dedicated $\texttt{ChangeAtomFeature}_i(v)$ and $\texttt{ChangeBondFeature}_{ij}(b)$ actions. For settings such as BondEdit that already work with a fixed atom set, `ChangeAtomFeature` and `ChangeBondFeature` become additional edits on the existing structure.

Tradeoffs and our default. Strategy A reduce the number of actions needed, but lock features in at insertion time. Strategy B keeps features editable throughout the trajectory, which lets the model correct earlier feature choices during sampling, and it composes cleanly with all instantiations above without enlarging fragment libraries by feature combinations. The best choice depend on the task and setting, but we found strategy B to perform best in most experiments.

The dependency structures can be easily adapted to accomodate these additional feature-wise edit actions.

E.7 Exact conditional paths for forest-plus-sink dependencies

The dependency structures of AtomGrow and MolEdit motivate a tractable construction beyond forests. Required `AddBond` actions may require both endpoint atoms to be inserted, and MolEdit's `DelAtom` actions require their incident bonds to be deleted. These multi-parent actions are sinks: they have prerequisites but no successors. We construct a conditional path by evolving the forest independently and scheduling each sink after its prerequisites.

Dependency structure. Fix endpoints (s_I, s_F) and a latent decomposition z, and write $G := \alpha(s_I, s_F \mid z)$ for the resulting action dependency DAG. Every topological ordering of G defines a valid action sequence between these endpoints. We consider DAGs consisting of a forest F and a set Z of multi-parent sinks whose prerequisites lie in F. This partition is fixed throughout the construction.

Sampling the conditional path. We use the linear time schedule and sample firing times as follows:

1. Sample a uniform topological ordering of F and assign its actions the ordered values of $|F|$ independent $\mathrm{Uniform}(0,1)$ draws. This is the existing forest construction underlying Prop. 4.7; denote the resulting firing times by t_a.
2. For each sink $b \in Z$, let $t_b^{\min} = \max_{a \in \mathrm{prereq}(b)} t_a$ and sample
$$t_b = t_b^{\min} + (1 - t_b^{\min})U_b, \qquad U_b \sim \mathrm{Uniform}(0,1),$$
using draws U_b that are independent across sinks and independent of the forest draws.

3. Obtain S_t by applying all actions with firing time at most t to s_I, in increasing firing-time order.

This procedure enables efficient sampling of intermediate states. When $Z = \varnothing$, it recovers exactly the forest construction of Prop. 4.7.

Exact conditional rates. We now show that the constructed path admits tractable, closed-form conditional rates. Fixing the endpoints and latent decomposition z, let $G_s := \alpha(s, s_F \mid z)$ denote the remaining action dependency DAG at state s. As in the preceding fixed-action constructions, we assume that s determines the remaining actions under this conditioning. Let F_s be the forest induced by the remaining forest actions, and let $\mathrm{desc}_{F_s}(a)$ denote the size of the subtree rooted at a, including a. The rates are given by the following proposition.

Proposition E.1 (Exact forest-plus-sink path)**.** *The sampling construction defines a conditional probability path starting from s_I and ending at s_F, generated, for $0 \le t < 1$, by*

$$u_t^{\mathrm{alt}}(s, a \mid s_I, s_F, z) = \frac{1}{1-t} \begin{cases} \mathrm{desc}_{F_s}(a), & a \in \mathrm{roots}(F_s), \\ 1, & a \in Z \cap \mathrm{roots}(G_s), \\ 0, & \textit{otherwise}. \end{cases} \tag{9}$$

Proof. The forest process has the rates of Prop. 4.7: every available forest action a fires at rate $\mathrm{desc}_{F_s}(a)/(1-t)$. Sinks do not affect this process. Their firing histories depend only on past forest events and independent sink draws, so observing them does not change the forest's future rates given its current completed actions.

A sink b cannot fire before $t_b^{\min}$. Conditional on its prerequisites' firing times, t_b is uniform on $[t_b^{\min}, 1]$. Once enabled and while unexecuted, its hazard is

$$\frac{1/(1-t_b^{\min})}{(1-t)/(1-t_b^{\min})} = \frac{1}{1-t}.$$

This expression is independent of $t_b^{\min}$, so the same rate holds after averaging over past prerequisite times. Thus all transition intensities depend only on the completed action set and t, yielding the stated CTMC and rates.

Forest actions respect their topological ordering and fire before 1 almost surely. Each sink fires strictly after its latest prerequisite and before 1 almost surely. The full firing order is therefore a topological ordering of G and executes every required action. By assumption it transforms s_I into s_F. □

Connection to trajectory matching. Trajectory matching permits any valid conditional path between the prescribed endpoints. Since Eq. (9) gives exact rates for such a path, the conditional-to-marginal argument of Thm. 3.2 applies with the endpoints and z as conditioning variables. Under realizability, matching these rates recovers the corresponding marginal rate and transports the source distribution to the target distribution.

Comparison with the uniform-topological path. The difference between the two paths lies in the back-coupling of sinks to their prerequisites. In the uniform-topological construction, all actions share a joint timing law: a sink must occur after its prerequisites, so its presence can accelerate those prerequisites, leaving time for the sink to execute. Our construction removes this back-coupling by sampling the forest independently and then scheduling sinks after their prerequisites. Once a sink is enabled, its rate is $1/(1-t)$ in both constructions.

For an available forest action, the relative rate difference is controlled by the number of still-disabled sinks with prerequisites in its residual subtree, relative to the size of that subtree. If there are no such sinks, the rates agree exactly. Both constructions respect the same dependencies and reach the same endpoints. Their difference concerns the intermediate probability path, which is a modeling choice in trajectory matching.

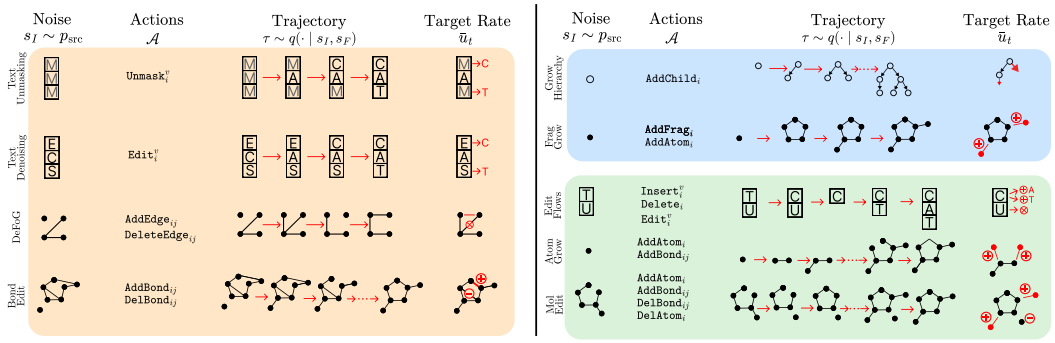


Figure 6: Examples of different existing methods that can be seen as instances of Trajectory Matching (text unmasking, text denoising, DeFog GraphGeneration Qin et al., 2025 and EditFlows Havasi et al., 2025) and our new instantiations of Trajectory Matching (AtomGrow, FragGrow, MolEdit, BondEdit, and GrowHierarchy). for each instantiation, we show an initial state $s_I \sim p_{\text{src}}$, the action space $\mathcal{A}$,a sample trajectory τ from s_I to a final state $s_F \sim p_{\text{tgt}}$ and the Rao-Blackwellized target rate $\bar{u}_t(s, a \mid s_I, s_F)$ of an intermediate state s.

F Connection to existing methods

We know show that Trajectory Matching is a general framework that encompasses many existing methods as special cases. In Fig. 6, we give an overview of how different existing methods can be seen as instances of Trajectory Matching, and how our new instantiations of Trajectory Matching fit into this picture. We discuss these connections in more detail in the following sections.

Any factorized flow matching method, including text unmasking (Austin et al., 2021), text denoising and fixed size graph generation (Qin et al., 2025) can be directly seen as instance of Trajectory Matching with an independent fixed-action distribution.

We will next show that we can also see Edit Flows (Havasi et al., 2025) as an instance of Trajectory Matching.

F.1 Edit Flows as Trajectory Matching

We finally relate our framework to Edit Flows. Edit Flows define a CTMC directly on variable-length sequences using edit operations: insertions, deletions, and substitutions. The main difficulty is that, for two sequences, there is in general no unique sequence of edits transforming one into the other. Edit Flows handle this by introducing an auxiliary alignment. Given a source sequence and a target sequence, an alignment produces two padded sequences (z_0, z_1) of the same length, where the blank symbol ϵ records insertions and deletions. Each aligned coordinate j with $z_0^j \neq z_1^j$ then specifies one elementary edit: if $z_0^j = \epsilon$ the edit inserts z_1^j, if $z_1^j = \epsilon$ the edit deletes z_0^j, and otherwise the edit substitutes z_0^j by z_1^j.

This fits naturally into trajectory matching by viewing the alignment as an additional latent variable. In the notation of Sec. 4, we take

$$z := (z_0, z_1),$$

and first sample z from the alignment distribution conditional on the endpoint pair. Conditional on z, define the set of indices that need to change as

$$E(z) := \{j : z_0^j \neq z_1^j\}.$$

A trajectory is then a permutation of this index set,

$$\tau = (j_1, \ldots, j_m), \qquad m := |E(z)|,$$

and represents the order in which the alignment-induced edits are performed. For a subset $S \subseteq E(z)$ of edits that have already been applied, let s_S be the current observable sequence obtained by applying the edits in S and removing blanks. For each remaining index $j \in E(z) \setminus S$, let

$$A_z(j, S)$$

denote the edit action on the current sequence s_S corresponding to applying the aligned edit at coordinate j. We define

$$q(\tau \mid z)$$

to be uniform over all permutations of $E(z)$. Equivalently, conditional on (z, S), the next index in the trajectory is uniform over the remaining set $E(z) \setminus S$.

It is important that this is not, in general, a fixed-action distribution in the sense of independent component-wise edits. Although the latent index set $E(z)$ is fixed once the alignment is fixed, the concrete edit action induced by an index can depend on the set S of edits that have already been applied, because insertions and deletions change positions in the observable sequence. For example, starting from the aligned source $\epsilon\,\epsilon$ and target $A\,B$, the ordering that inserts A before B corresponds to

$$\mathrm{insert}_0(A) \;\to\; \mathrm{insert}_1(B),$$

whereas the other ordering is

$$\mathrm{insert}_0(B) \;\to\; \mathrm{insert}_0(A).$$

Thus the same latent coordinate system defines the remaining edits, but the actual observable edit action $A_z(j, S)$ is state-dependent. We next show that, despite this dependence, marginalizing over the uniform ordering recovers exactly the Edit Flow probability path and marginal rate.

Proposition F.1 (Edit Flows are uniform-order trajectory matching)**.** *For a fixed alignment distribution over (z_0, z_1), let u_t^{EF} denote the Edit Flow rate with a linear time schedule, and let u_t^{TM} denote the marginal trajectory-matching rate obtained by sampling, conditional on (z_0, z_1), a uniform ordering of all alignment-compatible edits. Then Edit Flows and this trajectory-matching construction induce the same marginal probability path and the same marginal rate:*

$$p_t^{\mathrm{EF}}(s) = p_t^{\mathrm{TM}}(s), \qquad u_t^{\mathrm{EF}}(s, a) = u_t^{\mathrm{TM}}(s, a).$$

Proof. Fix an alignment (z_0, z_1) and abbreviate

$$E := \{j : z_0^j \neq z_1^j\}, \qquad m := |E|.$$

For $S \subseteq E$, let s_S be the observable state obtained by applying the edits in S and removing blanks. We do not assume that the map $(z_0, z_1, S) \mapsto s_S$ is injective.

Probability paths. In trajectory matching, a trajectory is a uniform ordering $\tau = (j_1, \ldots, j_m)$ of the edits in E, and under the linear schedule the firing times are i.i.d. uniform on $[0, 1]$ and independent of τ. In particular, the number of edits completed by time t satisfies $K_t \sim \mathrm{Binomial}(m, t)$ independently of τ. Hence, for any fixed $S \subseteq E$ with $|S| = k$,

$$\begin{aligned}
\mathbb{P}^{\mathrm{TM}}(S_t = S \mid z_0, z_1) &= \mathbb{P}(K_t = k)\,\mathbb{P}(\{j_1, \ldots, j_k\} = S) \\
&= \binom{m}{k} t^k (1-t)^{m-k} \cdot \frac{1}{\binom{m}{k}} \\
&= t^k (1-t)^{m-k}.
\end{aligned}$$

In Edit Flows, each aligned edit is independently applied with probability t, so

$$\mathbb{P}^{\mathrm{EF}}(S_t = S \mid z_0, z_1) = t^{|S|}(1-t)^{m-|S|}.$$

Therefore the two constructions induce the same distribution over (z_0, z_1, S_t). Since the observable state is the deterministic pushforward $X_t = s_{S_t}$ in both,

$$p_t^{\mathrm{TM}}(s \mid z_0, z_1) = p_t^{\mathrm{EF}}(s \mid z_0, z_1), \qquad p_t^{\mathrm{TM}}(s) = p_t^{\mathrm{EF}}(s).$$

Marginal rates. Fix (z_0, z_1, S) and write $R := E \setminus S$. For $j \in R$, let $A(j, s_S)$ denote the $\mathcal{S}$-edit action at s_S corresponding to applying the aligned edit j. The Edit Flow conditional rate is (in our action-based formulation)

$$u_t^{\mathrm{EF}}(s_S, a \mid S, z_0, z_1) = \sum_{j \in R} \frac{1}{1-t}\, \mathbf{1}[a = A(j, s_S)].$$

The trajectory-matching rate, conditioned on an ordering τ compatible with S at s_S (i.e. $\{j_1, \ldots, j_k\} = S$ where $k = |S|$), places all rate on the next edit:

$$u_t^{\mathrm{TM}}(s_S, a \mid \tau, z_0, z_1) = \frac{|R|}{1-t}\, \mathbf{1}[a = A(j_{k+1}, s_S)].$$

Conditioning a uniform ordering of E on the event that the prefix-set is S leaves the suffix uniform over orderings of R, so $\mathbb{P}(j_{k+1} = j \mid S, z_0, z_1) = 1/|R|$ for every $j \in R$. Averaging over the trajectory posterior gives

$$\begin{aligned}\mathbb{E}_{\tau|S,z_0,z_1}\left[u_t^{\mathrm{TM}}(s_S, a \mid \tau, z_0, z_1)\right] &= \frac{|R|}{1-t} \sum_{j \in R} \frac{1}{|R|}\, \mathbf{1}[a = A(j, s_S)] \\ &= u_t^{\mathrm{EF}}(s_S, a \mid S, z_0, z_1).\end{aligned}$$

This equality holds pointwise for every latent triple (z_0, z_1, S).

To obtain the marginal rate at an observable state s, we average over the posterior of latent triples compatible with $X_t = s$. By definition,

$$u_t^{\mathrm{EF}}(s, a) = \mathbb{E}^{\mathrm{EF}}_{z_0,z_1,S|X_t=s}\left[u_t^{\mathrm{EF}}(s, a \mid S, z_0, z_1)\right],$$

$$u_t^{\mathrm{TM}}(s, a) = \mathbb{E}^{\mathrm{TM}}_{z_0,z_1,S,\tau|X_t=s}\left[u_t^{\mathrm{TM}}(s, a \mid \tau, z_0, z_1)\right].$$

Applying the tower property,

$$u_t^{\mathrm{TM}}(s, a) = \mathbb{E}^{\mathrm{TM}}_{z_0,z_1,S|X_t=s}\left[\mathbb{E}^{\mathrm{TM}}_{\tau|S,z_0,z_1,X_t=s}\left[u_t^{\mathrm{TM}}(s, a \mid \tau, z_0, z_1)\right]\right].$$

Given (z_0, z_1, S), the suffix ordering $\sigma_{k+1:m}$ is independent of both K_t and the prefix-set under uniform σ and the linear schedule; the event $X_t = s$ is a function of (z_0, z_1, S) alone (it just selects latent triples with $s_S = s$), so conditioning further on it does not change the suffix law. Hence

$$\mathbb{E}^{\mathrm{TM}}_{\tau|S,z_0,z_1,X_t=s}\left[u_t^{\mathrm{TM}}(s, a \mid \tau, z_0, z_1)\right] = u_t^{\mathrm{EF}}(s, a \mid S, z_0, z_1).$$

By the path equivalence already shown, the posterior of (z_0, z_1, S) given $X_t = s$ is the same under TM and EF, so

$$u_t^{\mathrm{TM}}(s, a) = \mathbb{E}^{\mathrm{EF}}_{z_0,z_1,S|X_t=s}\left[u_t^{\mathrm{EF}}(s, a \mid S, z_0, z_1)\right] = u_t^{\mathrm{EF}}(s, a). \qquad \square$$

The equivalence extends to any shared time schedule κ satisfying the conditions in App. C, by applying the same time reparametrization to both constructions: $\tilde{p}_t = p_{\kappa_t}$ and $\tilde{u}_t = \dot{\kappa}_t u_{\kappa_t}$.

Importantly, this shows that the Edit Flow loss and our trajectory matching loss share the same optimum and gradients.

G Additional Experimental Results

This appendix first describes the empirical setup. Addtitionally it complements the empirical results of § 5 with: (i)extended baseline comparisons on the Tree (App. G.2) and molecular graph benchmarks (App. G.3); (ii) experiments on the validity–compute trade-off as a function of the Euler-integration budget; (iii) an ablation of the trajectory-conditioned versus Rao-Blackwellized training target predicted by Prop. B.3 (App. G.4); and (iv) an ablation of the trajectory distribution (App. G.5). Throughout, we use the same training hyperparameters, model backbone, and sampling procedure as described in App. G.1; only the quantity being varied changes.

G.1 Experimental Setup

All instantiations of trajectory matching presented in this work share a common training and sampling pipeline. The differences between instantiations are isolated to the action space, source distribution, and trajectory distribution as described in App. E; the architecture, loss function, datasets, and hyperparameters described in this section are held fixed across instantiations.

Model architecture. We use a graph-transformer backbone poposed by DiGress (Vignac et al., 2023) to predict a rate for every node and every node pair in the current state s, processed jointly by a stack of attention layers with node and pair representations updated in tandem. We use the additional positional encoding by DeFoG (Qin et al., 2025). We adopt the graph transformer unchanged in structure, with two practical adaptations: (i) we additionally include the current node count $|V(s)|$ as an input feature, encoded with a radial-basis-function (RBF) embedding, because our variable-size instantiations operate over states of varying size and the appropriate rate depends on how much of the generation has already been completed; and (ii)for the Tree benchmark we use a substantially smaller backbone than for the molecular datasets, while all molecular datasets share the same backbone size as in DeFoG. The architecture is permutation-equivariant on nodes and node pairs by construction, so the predicted rates are also permutation-equivariant.

Loss decomposition. The trajectory-conditioned target $u_t(\cdot \mid \tau)$ is sparse: only the next action in τ has positive rate at the current state s, while the Rao-Blackwellized target $\bar{u}_t(\cdot \mid s_I, s_F)$ has positive rate only on the small set of remaining actions $\alpha(s, s_F)$. To predict such a sparse rate vector, we follow OneFlow (Nguyen et al., 2025) and decompose the parametric rate into a positivity probability and a (log-)rate magnitude:

$$u_t^\theta(s,a) \;=\; h_t^\theta(s,a) \cdot \exp\big(g_t^\theta(s,a)\big), \tag{10}$$

where $h_t^\theta(s,a) := \sigma(f_t^\theta(s,a)) \in [0,1]$ is produced by a sigmoid head and represents the probability that a has positive rate at (s,t), and $g_t^\theta(s,a) \in \mathbb{R}$ is a raw log-rate head. Both heads share the same graph-transformer backbone.

We decompose the loss correspondingly. Fix (t,s,a), write $m^\theta = \exp(g_t^\theta(s,a))$, and let R denote the training rate target, either trajectory-conditioned or Rao–Blackwellized. With $Y = \mathbf{1}[R>0]$, the per-action objective is

$$\ell_{\mathrm{dec}}(\theta) = \mathbb{E}\big[\mathrm{BCE}(Y, h^\theta) + Y D(R, m^\theta) \mid t,s,a\big]\,. \tag{11}$$

Importantly, this practical implementation choice retains all central consistency guarantees of trajectory matching. In particular the population optimum of Eq. (11) recovers the same marginal probability path as the original Bregman rate-matching objective, as formalized below:

Proposition G.1 (Consistency of the decomposed objective)**.** *Define* $h_* = \Pr(R>0 \mid t,s,a)$ *and, when* $h_*>0$, $m_* = \mathbb{E}[R \mid t,s,a,R>0]$. *Then*

$$\nabla_\theta \ell_{\mathrm{dec}}(\theta) = \nabla_\theta \big[\mathrm{BCE}(h_*, h^\theta) + h_* D(m_*, m^\theta)\big]\,.$$

Under head-wise realizability, the population optimum recovers $u_t^\theta(s,a) = h_* m_* = \mathbb{E}[R \mid t,s,a] = u_t(s,a)$ *on the support of the training distribution, and hence generates the same marginal probability path. When* $h_* = 0$*, the magnitude target may be chosen arbitrarily and the optimal rate is zero.*

Proof. BCE is affine in its target, and the model gradient of a Bregman divergence is affine in its first argument. Taking conditional expectations therefore gives the gradient identity. The corresponding marginal objective is minimized at $h^\theta = h_*$ and $m^\theta = m_*$ on positive support. Their product equals the marginal rate by conditional expectation, for either choice of R; the path claim follows from Thm. 3.2. □

Tab. 4 compares the decomposed objective with direct Bregman rate matching on GROWHIERARCHY. Both objectives achieve similar performance in this experiment.

Table 4: Objective ablation on Tree with GROWHIERARCHY. Entries report means and standard deviations over sampling seeds from a single trained model per objective.

Objective	Degree ↓	Spectral ↓	Wavelet ↓	Validity (%) ↑
Decomposed	0.0006 ± 0.0002	0.0072 ± 0.0006	0.0033 ± 0.0002	100.0 ± 0.0
Direct Bregman	0.0008 ± 0.0002	0.0085 ± 0.0009	0.0034 ± 0.0004	100.0 ± 0.0

Datasets. We use the standard dataset splits as in DeFoG (Qin et al., 2025) to ensure direct comparability with that line of work; we summarize each dataset briefly here and refer to DeFoG App. F.2 for full preprocessing details. FragGrow uses the representable subset of each training split, as detailed in App. E.2. We explicitly do not filter the validation or test splits, so evaluation uses the same complete standard reference splits as the other models. For the molecular benchmarks, ZINC is distributed under a custom license and is free to use for everyone, while MOSES is released under the MIT license.

Tree. The synthetic random-tree dataset introduced in Bergmeister et al. (2024), consisting of trees with 64 nodes. Used in § 5 and App. G.2 to evaluate hierarchy generation. Validity here means that the generated graph is a connected acyclic graph; the Ratio metric captures distributional fit through normalized distribution-style graph statistics. We note that the Ratio metric is very unstable as it divides through a very small number.

QM9 (without hydrogens). QM9 (Ramakrishnan et al., 2014) contains small organic molecules with up to 9 heavy atoms (C, N, O, F), with hydrogens removed.

ZINC250K. A subset of approximately 250K drug-like molecules from the ZINC database. Tests scaling to larger and more chemically diverse molecules and is the substrate for our downstream substructure extension and linking experiments (App. G.6).

MOSES. A standardized benchmark for de novo molecule generation comprising approximately 1.9M drug-like molecules, with the standard MOSES train/test split as used by DeFoG. Includes explicit metrics for distributional fit such as FCD, SNN similarity, and scaffold similarity, and is the most challenging of our molecular benchmarks for both quality and coverage.

Training. We implement all experiments in PyTorch (Paszke et al., 2019). All experiments use a single NVIDIA A100 GPU per training run. We train with Adam at learning rate $2 \cdot 10^{-4}$, no weight decay, and a fixed (no schedule) learning rate. Batch sizes are 64 for QM9 and ZINC and 32 for MOSES (smaller due to memory). Training durations are 5 000 steps for Tree, 200 000 steps for QM9 and ZINC, and 500 000 steps for MOSES. We use exponential moving average (EMA) on model weights with decay 0.9999, started after step 1 000 and updated every step. Gradients are clipped at L_2-norm 1.0. The training-time distribution over t is uniform on $[0, 1]$; the conditional-flow time schedule κ (App. C) is either linear or the polydec schedule of DeFoG, with polydec yielding better results on most datasets. Checkpoints are selected by lowest validation FCD for the molecular datasets and lowest ratio on tree.

Computational cost. We compare the computational cost of trajectory matching with the factorized DFM baseline DeFoG on QM9, with measurements averaged after warmup. Tab. 5 reports both raw CPU time for trajectory sampling and target construction and the loader wait observed by the training loop. These preprocessing operations run in parallel CPU dataloader workers, overlapping with GPU training (App. D). Despite differences in raw construction time, the observed loader wait remains below 1 ms per training step, making preprocessing essentially free in exposed wall-clock cost in this setup. FragGrow's CPU preprocessing cost could be reduced further by precomputing and caching the training molecules' fragment decompositions (App. E.2), although this need not yield a noticeable end-to-end speedup when preprocessing is already overlapped.

Table 5: CPU preprocessing time and observed loader wait on QM9. Preprocessing overlaps with GPU training; loader wait is measured per training step and is not the total overhead relative to DeFoG.

Method	Trajectory sampling + target construction (ms)	**Observed loader wait (ms)**
AtomGrow	4.56	0.75
FragGrow	188.83	0.94
MolEdit	30.55	0.91

Tab. 6 compares end-to-end training and sampling throughput. Relative to DeFoG, training throughput is approximately 10–14% lower and sampling throughput is 22–25% lower at the same 500-step sampling budget.

Table 6: End-to-end throughput on QM9 using one A100 GPU, averaged after warmup. Sampling uses 500 integration steps for every method.

Method	Training throughput (steps/s)	Sampling throughput (samples/s)
DeFoG	13.79	10.22
AtomGrow	11.89	7.96
FragGrow	12.01	7.63
MolEdit	12.45	7.69

The inductive bias of structured actions can offset these higher per-step training and sampling costs by enabling smaller models, fewer training steps, or fewer sampling steps. On Tree, training takes less than 1 h, compared with approximately 8 h reported for DeFoG (Qin et al., 2025). We attribute this advantage in part to the tree-preserving inductive bias, which allows us to use a substantially smaller model and train for fewer steps (5 000 in our setup). Separately, our sampling-budget experiments (Figs. 7 and 8) show that structured actions can preserve generation quality with fewer integration steps, potentially reducing total sampling time. On the molecular datasets, approximate per-instantiation training times are ~ 8 h on QM9, ~ 24 h on ZINC, and ~ 48 h on MOSES.

Sampling. We use the Euler integration scheme of Algorithm 2 with $N = 1000$ steps for the Tree benchmark and $N = 500$ steps for QM9, ZINC, and MOSES, matching DeFoG's defaults. We do not employ any of DeFoG's sampling-time optimizations (target guidance, stochasticity, time distortion). For evaluation we generate 40 samples per run on Tree, $10\,000$ samples on QM9 and ZINC, and $25\,000$ samples on MOSES. Step-budget sweeps in Apps. G.2 and G.3 vary N around these defaults to study the quality–compute trade-off.

Reported metrics and variability. Unless otherwise stated, results are means and standard deviations over 5 independent sampling seeds from a single trained model; for MOSES, we follow the evaluation setup of DeFoG and DiGress (Qin et al., 2025; Vignac et al., 2023) and report a single run of $25\,000$ samples. To assess sampling variability, we additionally evaluated AtomGrow and FragGrow using three independent sampling seeds per fixed trained model, obtaining validity standard deviations of 0.1 and 0.02 percentage points and FCD standard deviations of 0.008 and 0.006, respectively. These are small relative to FragGrow's validity improvement and AtomGrow's FCD improvement over DeFoG, supporting that these gains are unlikely to be explained by sampling randomness.

G.2 Hierarchy Generation: Extended Results

Full baseline comparison. The expanded tree table extends Tab. 1 from the main text with additional baselines. The picture is consistent with the main-text findings: GrowHierarchy achieves 100% V.U.N., with validity guaranteed by construction, while methods that do not encode tree structure in their action space either underperform on V.U.N. or collapse entirely. The only exception is HSpectre, which combines a diffusion and an autoregressive approach but cannot guarantee validity by construction.

While our main comparisons focus on diffusion and flow-matching methods to isolate the effect of structured actions, Table 7 also includes four purely autoregressive baselines: GraphRNN, GRAN, BiGG, and GraphGen. GrowHierarchy achieves 100% V.U.N., compared with the best autoregressive result of 95%, and a Ratio of 2.5, compared with 5.2.

Validity–compute trade-off. A single-step-count comparison can mask large differences in how a model degrades when the sampling budget is reduced. Fig. 7 sweeps the number of Euler steps from 10 to 10^3 for both GrowHierarchy and DeFoG, reporting validity (left) and the Ratio metric (right). The validity panel (left) shows the central effect of a domain-aligned action space. GrowHierarchy maintains 100% validity across the entire range, including regimes where the step budget is far smaller than the number of actions per trajectory. The Ratio metric shows similar results for both methods across the step sizes. However, DeFoG's validity falls steeply as the budget shrinks:

Table 7: Graph generation results on the Tree dataset.

Model	Class	Tree Dataset									
		Deg.↓	Clus.↓	Orbit↓	Spec.↓	Wavelet↓	Ratio↓	Valid↑	Unique↑	Novel↑	V.U.N.↑
Train set	—	0.0001	0.0000	0.0000	0.0075	0.0030	1.0	100	100	0.0	0.0
GraphRNN (You et al., 2018)	Autoregressive	—	—	—	—	—	607.0	—	—	—	0.0
GRAN (Liao et al., 2020)	Autoregressive	0.1884	0.0080	0.0199	0.2751	0.3274	607.0	0.0	100	100	0.0
DiGress (Vignac et al., 2023)	Diffusion	0.0002	0.0000	0.0000	0.0113	0.0043	**1.6**	90.0	100	100	90.0
EDGE (Chen et al., 2023)	Diffusion	0.2678	0.0000	0.7357	0.2247	0.4230	850.7	0.0	7.5	100	0.0
BwR (Diamant et al., 2023)	Diffusion	0.0016	0.1239	0.0003	0.0480	0.0388	11.4	0.0	100	100	0.0
BiGG (Dai et al., 2020)	Autoregressive	0.0014	0.0000	0.0000	0.0119	0.0058	5.2	100	87.5	50.0	75.0
GraphGen (Goyal et al., 2020)	Autoregressive	0.0105	0.0000	0.0000	0.0153	0.0122	33.2	95.0	100	100	95.0
HSpectre (one-shot) (Bergmeister et al., 2024)	Diffusion	0.0004	0.0000	0.0000	0.0080	0.0055	2.1	82.5	100	100	82.5
HSpectre (Bergmeister et al., 2024)	Autoregressive + Diffusion	0.0001	0.0000	0.0000	0.0117	0.0047	4.0	100	100	100	100
DeFoG (Qin et al., 2025)	Flow	0.0002 $\pm$0.0001	0.0000 $\pm$0.0000	0.0000 $\pm$0.0000	0.0108 $\pm$0.0028	0.0046 $\pm$0.0004	**1.6** $\pm$0.4	96.5 $\pm$2.6	100.0 $\pm$0.0	100.0 $\pm$0.0	96.5 $\pm$2.6
GrowHierarchy	Flow	0.0006 $\pm$0.0002	0.0000 $\pm$0.0000	0.0000 $\pm$0.0000	**0.0072** $\pm$0.0006	**0.0033** $\pm$0.0002	2.5 $\pm$0.6	**100.0** $\pm$0.0	100.0 $\pm$0.0	100.0 $\pm$0.0	**100.0** $\pm$0.0

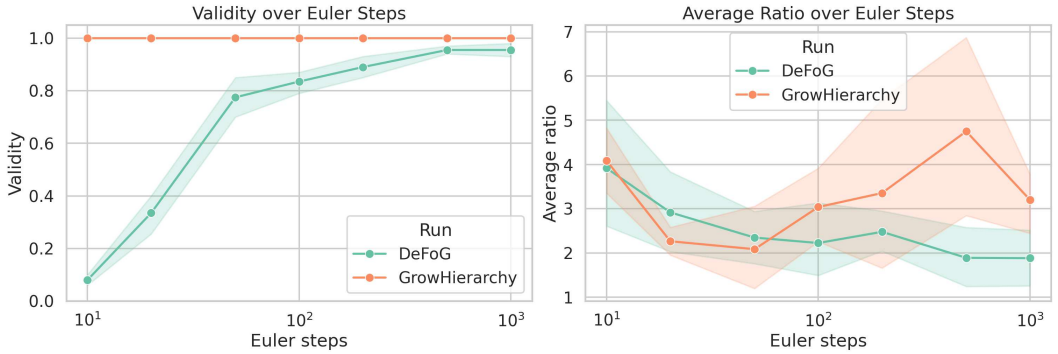


Figure 7: Validity (left) and Ratio (right) on the Tree benchmark as a function of the number of Euler integration steps. GROWHIERARCHY preserves 100% validity across three orders of magnitude of step budgets, while DeFoG's validity collapses below $\sim$10% at 10 steps. Average Ratio metric is comparable for both.

from $\sim$96% at 10^3 steps to $\sim$8% at 10 steps. This underlines the advantage that domain-aligned actions can bring in low-sampling regions.

G.3 MOLECULAR GRAPH GENERATION: EXTENDED RESULTS

Qm9 datset. We include additional results on the QM9 dataset in Tab. 8 which shows similar performance trends as seen for MOSES and ZINC in the main text.

Table 8: Comparison of DFM and diffusion molecular graph generative models on QM9 (without hydrogens). Best results in **bold**, second best underlined.

	Model	Val.↑	R. Val.↑	Uniq.↑	FCD↓
	Training set	99.3	99.5	99.2	0.03
Fixed-size	DiGress (Vignac et al., 2023)	99.0$_{\pm 0.0}$	–	96.2$_{\pm 0.1}$	–
	DisCo (Xu et al., 2024)	99.3$_{\pm 0.6}$	–	–	–
	Cometh (Siraudin et al., 2024)	99.6$_{\pm 0.1}$	–	96.8$_{\pm 0.2}$	0.25$_{\pm 0.01}$
	GBD (Liu et al., 2024)	**99.8**	–	–	0.093
	DeFoG (Qin et al., 2025)	99.3$_{\pm 0.0}$	99.4$_{\pm 0.1}$	96.3$_{\pm 0.3}$	0.12$_{\pm 0.00}$
	BondEdit	99.0$_{\pm 0.0}$	99.2$_{\pm 0.0}$	96.6$_{\pm 0.0}$	**0.085**$_{\pm 0.00}$
Variable-size	GrIDDD (Ninniri et al., 2025)	97.0	–	96.2	0.11
	MolEdit	98.8$_{\pm 0.1}$	99.2$_{\pm 0.1}$	96.0$_{\pm 0.1}$	0.17$_{\pm 0.01}$
	AtomGrow	98.8$_{\pm 0.0}$	99.2$_{\pm 0.0}$	**97.0**$_{\pm 0.0}$	0.096$_{\pm 0.00}$
	FragGrow	99.3$_{\pm 0.0}$	**99.7**$_{\pm 0.0}$	96.3$_{\pm 0.0}$	0.14$_{\pm 0.01}$

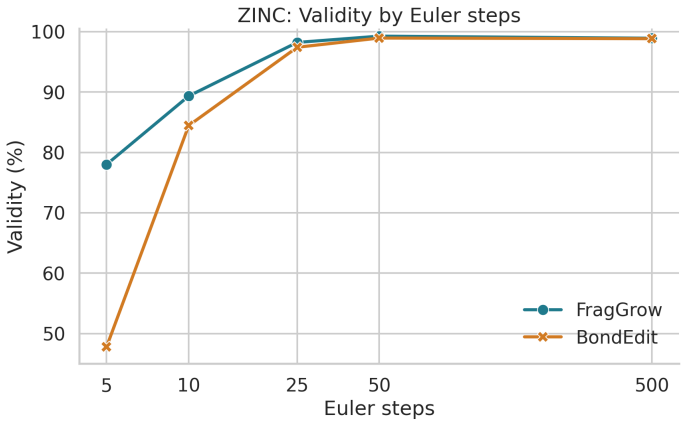


Figure 8: ZINC validity as a function of the number of Euler steps for the fragment-based FRAGGROW and the factorized BONDEDIT. FRAGGROW maintains substantially higher validity in the low-step regime (e.g., 78% vs. 48% at 5 steps); both methods saturate at $\geq 98\%$ for ≥ 25 steps.

Autoregressive comparison on MOSES. We additionally compare with GraphINVENT (Mercado et al., 2021), an autoregressive molecular graph generator. As shown in Tab. 9, FRAGGROW achieves higher validity (99.2% versus 96.4%) and passes molecular filters more frequently (99.3% versus 95.0%), while achieving comparable FCD (1.21 versus 1.22).

Table 9: Autoregressive comparison on MOSES. GraphINVENT results are taken from DiGress (Vignac et al., 2023). Results for all flow-matching/diffusion methods in Tab. 2.

Model	Validity ↑	Uniqueness ↑	Filters ↑	FCD ↓	SNN ↑	Scaffold ↑
GraphINVENT	96.4	99.8	95.0	1.22	0.54	12.7
FRAGGROW	99.2	99.9	99.3	1.21	0.55	13.7

Sampling-budget efficiency on ZINC. Unlike the Tree setting, the molecular action space does not yield validity by construction: even FRAGGROW can produce invalid molecules (e.g. valence violations). Nevertheless, Fig. 8 shows that a domain-aligned action space still pays off in the low-budget regime. At 5 Euler steps, FRAGGROW retains $\sim 78\%$ validity against $\sim 48\%$ for the factorized baseline BONDEDIT; at 10 steps the gap narrows to 89% vs. 84%, and both saturate at $\geq 98\%$ from 25 steps onward. Each fragment action commits an entire ring system in a single step, so the same molecule is assembled in far fewer actions making FRAGGROW especially attractive whenever sampling cost matters, e.g. as a subroutine in downstream search or screening loops.

G.4 TRAJECTORY-CONDITIONED VS. RAO-BLACKWELLIZED TARGETS

Trajectory matching admits two training targets: the trajectory-conditioned rate $u_t(\cdot \mid \tau)$ defined in Eq. (3), and its Rao-Blackwellized counterpart $\bar{u}_t(\cdot \mid s_I, s_F)$ obtained by marginalizing over trajectories consistent with the current state and endpoints. Prop. B.2 guarantees that the two targets share the same optimum and expected gradient, while Prop. B.3 predicts that the Rao-Blackwellized estimator has component-wise lower or equal variance. We test the empirical impact of this variance reduction on training dynamics.

Fig. 9 compares the two targets on ZINC under ATOMGROW, plotting validity over the first 40 training epochs while keeping all other hyperparameters fixed. The Rao-Blackwellized estimator reaches $\geq 95\%$ validity within the first few epochs and exhibits less epoch-to-epoch noise; the trajectory-conditioned estimator requires roughly twice as many epochs to reach the same threshold and remains noisier throughout. Both targets converge to comparable final validity ($\sim$ 97–99%), consistent with Prop. B.2: the optimum is unchanged, only the trajectory toward it is smoother.

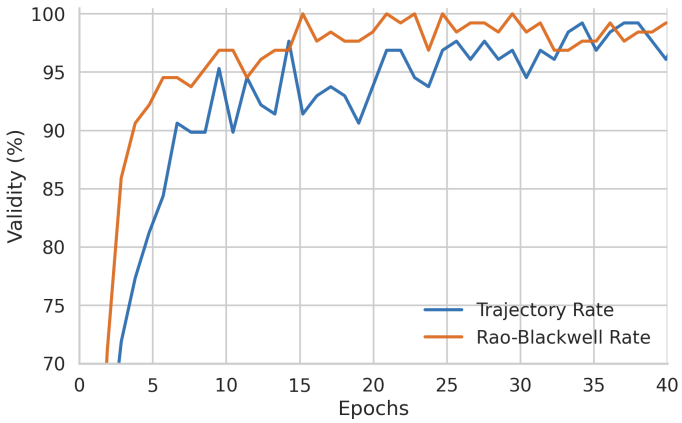


Figure 9: Training dynamics on ZINC under AtomGrow: the Rao-Blackwellized target $\bar{u}_t$ converges faster and with lower epoch-to-epoch variance than the trajectory-conditioned target u_t, while reaching the same final validity. The behavior matches the variance-reduction guarantee of Prop. B.3.

G.5 Effect of the Trajectory Distribution

We study the effect of the training trajectory distribution q using AtomGrow on QM9. We compare three policies for the order of node insertions: random, breadth-first (BFS), and depth-first (DFS). These policies jointly determine the insertion order and the spanning tree, so the experiment varies their combined effect through q. All three variants use single-trajectory training targets. The action space, source distribution, architecture, optimization settings, and sampling procedure are held fixed; non-tree edges are treated in the same way across variants.

Table 10: Effect of the trajectory distribution on AtomGrow on QM9, using single-trajectory training targets for all three policies. Results are means and standard deviations over five independent sampling runs from one trained model per policy.

Policy	Validity (%) ↑	Relaxed validity (%) ↑	Uniqueness (%) ↑	FCD ↓
Random	98.8 ± 0.0	99.2 ± 0.0	97.0 ± 0.0	0.096 ± 0.00
BFS	98.9 ± 0.1	99.2 ± 0.1	96.6 ± 0.2	0.092 ± 0.00
DFS	98.8 ± 0.1	99.2 ± 0.1	96.8 ± 0.2	0.098 ± 0.00

Tab. 10 shows similar performance across the three choices of q, suggesting that AtomGrow is robust to these tested trajectory policies on QM9.

G.6 Substructure Extension and Linking

A practical advantage of variable-size, order-invariant trajectory matching is that the same trained model can address structured downstream tasks without retraining or task-specific finetuning. We illustrate this on two such tasks, substructure extension and substructure linking, and show that AtomGrow accommodates them out of the box, while the factorized BondEdit baseline requires task-specific patches and underperforms substantially.

Setup. For all downstream evaluations and qualitative examples, we use a single AtomGrow model trained on ZINC with one root atom with probability 0.5 and two randomly chosen root atoms with probability 0.5 (App. E.3). The same frozen checkpoint is used for extension and linking, without downstream supervision or fine-tuning. In unconditional generation on ZINC, this randomized-root model achieves 98.7% validity, 0.53 scaffold similarity, and 1.46 FCD. BondEdit also uses a fixed ZINC checkpoint. For AtomGrow, both tasks reduce to a choice of s_I: extension corresponds to setting s_I to the given substructure, and linking corresponds to setting s_I to the union of two disconnected substructures, both treated as roots of the action DAG.

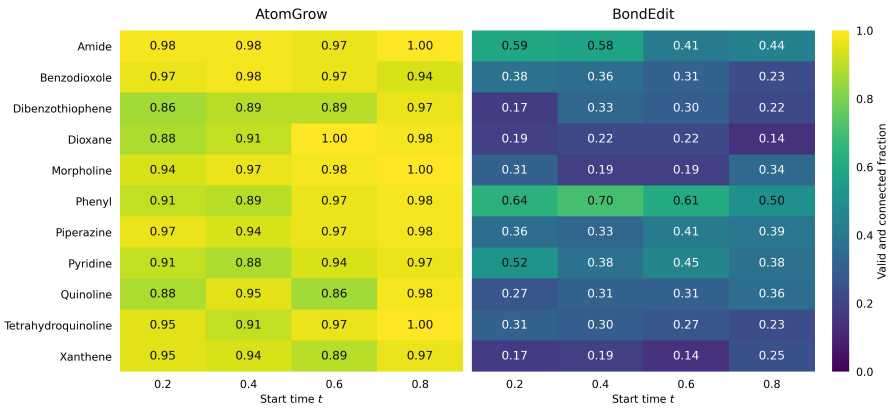


Figure 10: Substructure extension on eleven seed substructures (rows) and four Euler start times (columns), reporting the valid-and-connected fraction of generated completions. AtomGrow (left) accommodates the task by setting s_I to the seed; BondEdit (right) is adapted by sampling a target size and freezing the seed during generation. AtomGrow substantially outperforms the factorized baseline across all seeds and start times, including substructures absent from ZINC training molecules.

The factorized BondEdit operates on a fixed atom set. We sample a number of additional atoms, with their features and the bonds between them drawn from p_{src}, and combine them with the supplied substructure(s) to form s_I. During generation, each substructure's internal structure is frozen.

We sweep the Euler start time $t \in \{0.2, 0.4, 0.6, 0.8\}$ for both methods, which controls how much of the trajectory remains and hence the typical extent of generation. Each evaluation uses one sampling run on the original 500-step Euler time grid, starting at the selected t. We sample $N = 64$ completions per (substructure, t) cell and report the fraction that are simultaneously chemically valid and yield a connected molecule. Aggregate scores pool all samples across cells, equivalently averaging the unrounded cell fractions with equal weight.

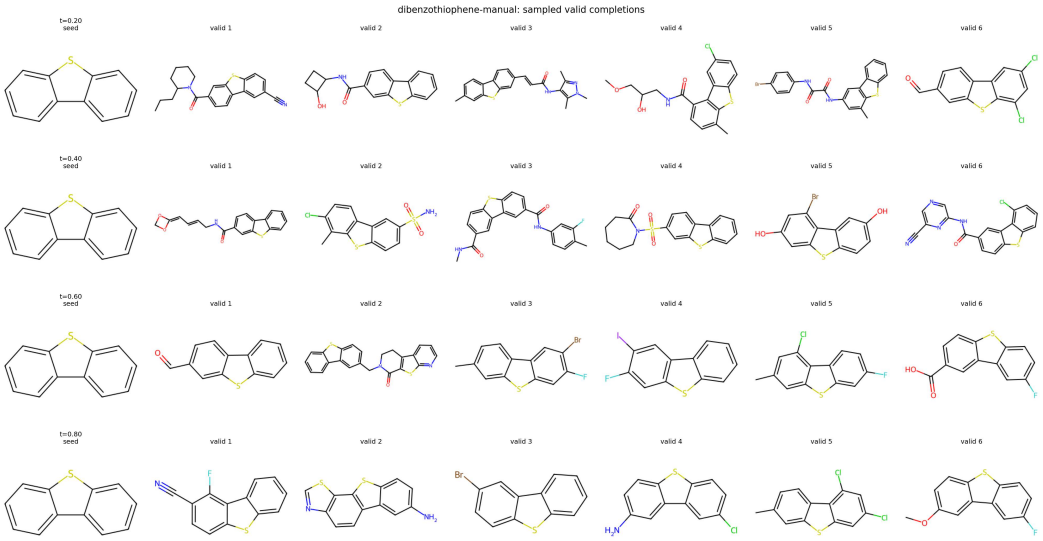
Substructure extension. We evaluate on eleven chemically meaningful seed substructures, deliberately including substructures that do not appear in any ZINC training molecule to test out-of-distribution generalization. Fig. 10 reports valid-and-connected fractions across all seeds and start times.

AtomGrow achieves valid-and-connected fractions in the 0.86–1.00 range across (seed, t) cells, including the out-of-distribution substructures. BondEdit, despite the same backbone and training procedure, plateaus at 0.14–0.70 — well below AtomGrow for every setting.

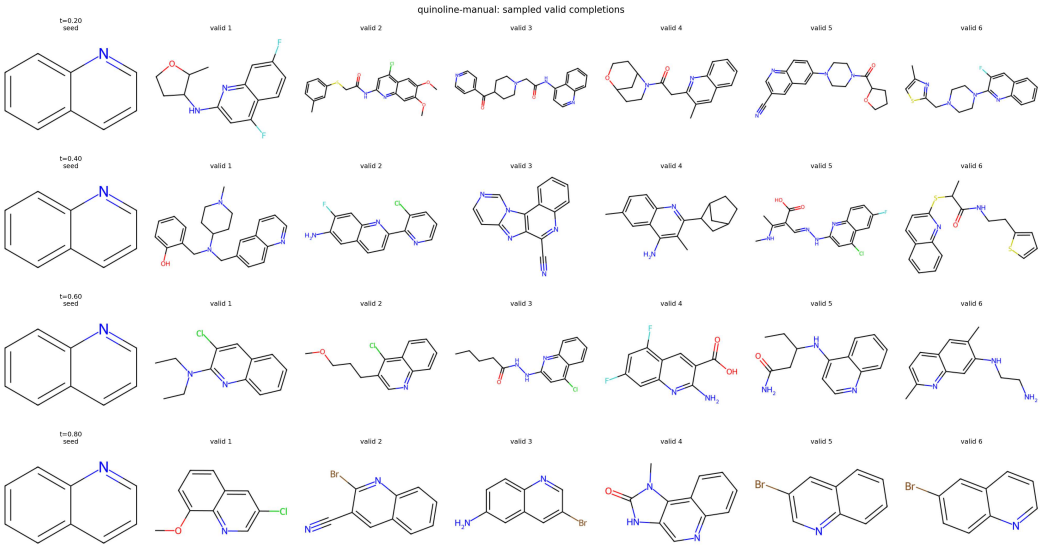
Beyond the aggregate fractions, the start time provides a natural, training-free knob over the size of the extension. Fig. 11 shows valid completions sampled by AtomGrow for two representative seeds (dibenzothiophene and quinoline) at four start times: at $t = 0.2$ many atoms remain to be added and the model produces large, decorated extensions; at $t = 0.8$ only a few atoms are added and completions stay close to the seed (e.g. small substituents on the aromatic core). This gives a single user-controllable lever for trading off seed-faithfulness against decoration extent — without retraining or any modification to the model.

Substructure linking. Linking is a strictly harder task: starting from two disjoint substructures, the model must produce a single molecule containing both as substructures, which in particular requires forming at least one connecting path between them. AtomGrow accommodates this by setting s_I to the union of the two substructures and treating both as roots of the action DAG; the remaining atoms and bonds, including any connecting path, are generated by the standard rollout. BondEdit adapts as in the extension setting.

Fig. 12 reports valid-and-connected fractions on six seed pairs across the same start-time sweep. AtomGrow attains 0.58–0.91 across all cells and outperforms BondEdit on every (pair, t) combi-



(a) Dibenzothiophene seed (out-of-distribution: not present in ZINC).



(b) Quinoline seed.

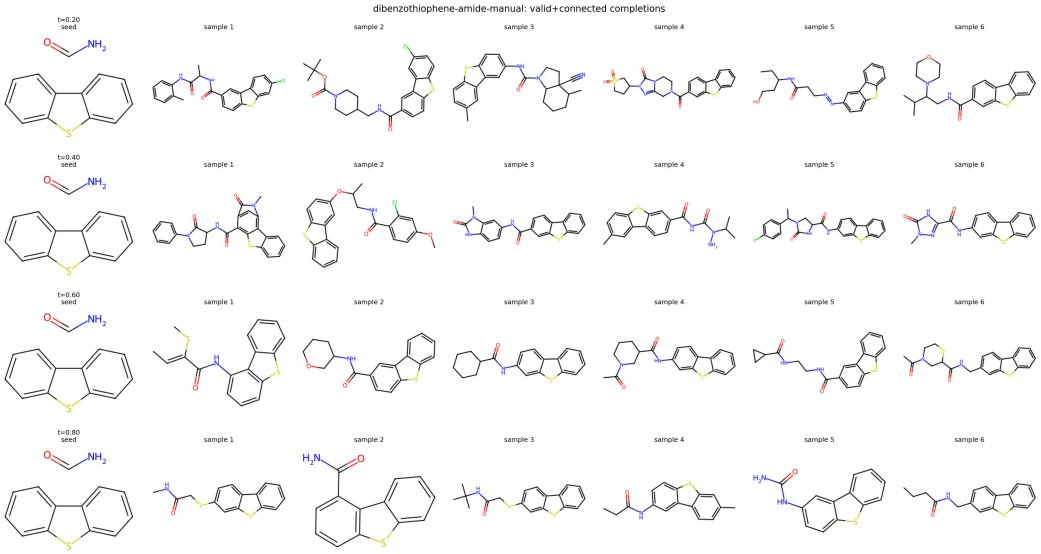
Figure 11: Valid completions sampled by AtomGrow for two seed substructures at four Euler start times $t \in \{0.2, 0.4, 0.6, 0.8\}$ (one row each). The seed (leftmost column) is preserved as a substructure of every completion. Lower t leaves more of the trajectory remaining and produces larger, more elaborately decorated extensions; higher t yields small decorations on the seed.

nation. For example, dibenzothiophene + formamide reaches 0.81 vs. 0.33 at $t = 0.8$, and pyridine + morpholine reaches 0.91 vs. 0.56 at $t = 0.6$.

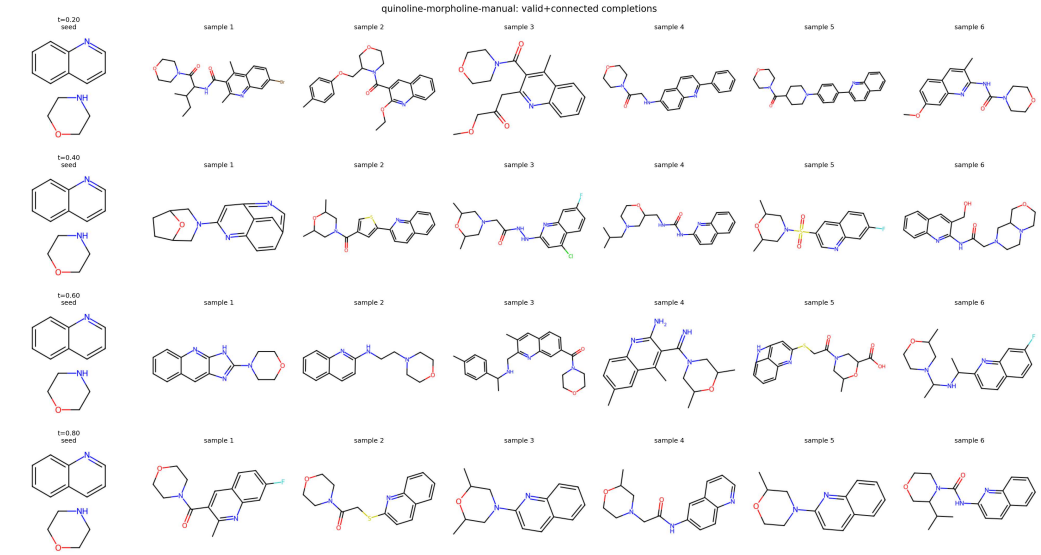
Fig. 13 shows qualitative successful linkings sampled by AtomGrow for two of these seed pairs (dibenzothiophene + formamide and quinoline + morpholine). Both seeds are preserved as substructures, and the linker between them ranges from a single bond or short chain at high t to multi-atom paths with additional functional groups at low t, mirroring the start-time control observed for extension.



Figure 12: Substructure linking on six seed pairs (rows) and four Euler start times (columns), reporting the valid-and-connected fraction of generated molecules. ATOMGROW (left) sets s_I to the union of both seeds and uses both as roots of the action DAG; BONDEDIT (right) is adapted by sampling a target size, freezing both seeds, and requiring inserted bonds to link them. ATOMGROW substantially outperforms the factorized baseline across all pairs and start times.



(a) Dibenzothiophene + formamide.



(b) Quinoline + morpholine.

Figure 13: Valid, connected linkings sampled by AtomGrow for two seed pairs at four Euler start times $t \in \{0.2, 0.4, 0.6, 0.8\}$ (one row each). Both seed substructures (leftmost column) appear in every generated molecule, joined by a linker whose complexity is controlled by t.