A Appendix

A.1 Derivation of modular adjoint

We present a standard adjoint gradient derivation (Bradley, 2019), and show that the adjoint of a graph neighborhood differential is sparse.

For completeness, we define an ODE system

$$\dot{\mathbf{z}}(t) = f(\mathbf{z}, t, \boldsymbol{\theta}) \tag{11}$$

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_0^t f(\mathbf{z}, t, \boldsymbol{\theta}) d\tau, \tag{12}$$

where $\mathbf{z} \in \mathbb{R}^D$ is a state vector, $\dot{\mathbf{z}} \in \mathbb{R}^D$ is the state time differential defined by the vector field function f and parameterised by $\boldsymbol{\theta}$. The starting state is $\mathbf{z}_0$, and $t, \tau \in \mathbb{R}_+$ are time variables. Our goal is to solve a constrained problem

$$\min_{\boldsymbol{\theta}} \qquad G(\boldsymbol{\theta}) = \int_0^T g(\mathbf{z}, t, \boldsymbol{\theta}) dt \tag{13}$$

$$s.t. \qquad \dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta}) = 0, \qquad \forall t \in [0, T] \tag{14}$$

$$\mathbf{z}(0) - \mathbf{z}_0 = 0, \tag{15}$$

where G is the total loss that consists of instant loss functionals g. We desire to compute the gradients of the system $\nabla_{\boldsymbol{\theta}} G$.

We optimise the constrained problem by solving the Lagrangian

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = G(\boldsymbol{\theta}) + \int_0^T \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) \tag{16}$$

$$= \int_0^T \Big[g(\mathbf{z}, t, \boldsymbol{\theta}) + \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) \Big] dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) \, . \tag{17}$$

The constraints are satisfied by the ODE definition. Hence, $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} G$, and we can set values $\boldsymbol{\theta}$ and $\boldsymbol{\mu}$ freely. We use a shorthand notation $\frac{\partial a}{\partial b} = a_b$, and omit parameters from the functions for notational simplicity. Applying the chain rule, we note that the gradient becomes

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} G = \int_0^T \Big[g_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} + g_{\boldsymbol{\theta}} + \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}} \Big] dt \, , \tag{18}$$

where the $\boldsymbol{\mu}$ term drops out since it does not depend on parameters $\boldsymbol{\theta}$. We apply integration by parts to swap the differentials in term $\boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}}$, resulting in

$$\int_0^T \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} dt = \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T} - \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0} - \int_0^T \dot{\boldsymbol{\lambda}}^\top \mathbf{z}_{\boldsymbol{\theta}} dt \, . \tag{19}$$

Substituting this into previous equation and rearranging the terms results in

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T \underbrace{(g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} - \dot{\boldsymbol{\lambda}}^\top) \mathbf{z}_{\boldsymbol{\theta}}}_{0, \text{ if } \dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}} dt + \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt + \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T}}_{0, \text{ if } \boldsymbol{\lambda}(T) = \mathbf{0}} - \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0}}_{0} \, . \tag{20}$$

The last term is removed since $\mathbf{z}(0)$ not depend on $\boldsymbol{\theta}$ as a constant, and thus $\mathbf{z}_{\boldsymbol{\theta}}(0) = 0$. The difficult term in the equation is $\mathbf{z}_{\boldsymbol{\theta}}$. We remove it by choosing

$$\dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}. \tag{21}$$

Finally, we choose $\boldsymbol{\lambda}(T) = 0$ which also removes the second-to-last term. The choices lead to a final term

$$\nabla_{\boldsymbol{\theta}} G = \nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt \tag{22}$$

$$s.t. \quad \dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} \tag{23}$$

$$\boldsymbol{\lambda}(T) = 0. \tag{24}$$

In the derivation the adjoint $\boldsymbol{\lambda}(t) = \frac{\partial \mathcal{L}}{\partial \mathbf{z}(t)} \in \mathbb{R}^D$ represents the change of loss with respect to instant states, and is another ODE system that runs backwards from $\boldsymbol{\lambda}(T) = 0$ until $\boldsymbol{\lambda}(0)$. The final gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ counts all adjoints within $[0, T]$ multiplied by the 'immediate' partial derivatives $f_{\boldsymbol{\theta}}$. The final gradient also takes into account the instant loss parameter derivatives. For simple MSE curve fitting, the instant loss has no parameters.

The adjoint depends on the instant loss state derivatives $g_{\mathbf{z}}$. These are often only available for observations $\mathbf{y}_j$ at observed timepoints t_j. This can be represented by having a convenient loss

$$g(\mathbf{z}, t, \boldsymbol{\theta}) = \delta(t = t_j)\tilde{g}(\mathbf{z}, \mathbf{y}_j, t, \boldsymbol{\theta}), \tag{25}$$

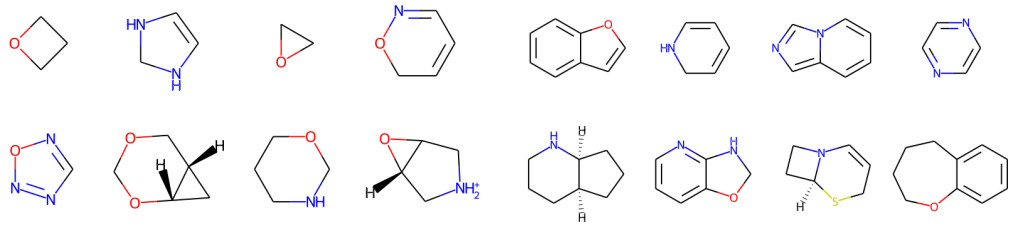
and now the term $g_{\mathbf{z}}$ induces discontinuous jumps at observations. This does not pose problems in practice, since we can integrate the ODE in continuous segments between the observation instants.

The sparsity of the adjoint evolution is evident from Equation 23, where the $\dot{\boldsymbol{\lambda}}_i$ is an inner product between $\boldsymbol{\lambda}$ and one column of $\frac{\partial f}{\partial \mathbf{z}}$, which is invariant to non-neighbors. This gives the result

$$\frac{d\boldsymbol{\lambda}_i}{dt} = -\boldsymbol{\lambda}(t)^\top \frac{\partial f\left(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\right)}{\partial \mathbf{z}} = -\sum_{j \in \mathcal{N}_i \cup \{i\}} \boldsymbol{\lambda}_j(t)^\top \frac{\partial f\left(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\right)}{\partial \mathbf{z}_j}. \tag{26}$$

A.2 Tree Decomposition

For tree decomposition of the molecules, we followed closely the procedure described in Jin et al. (2018). The rings as well as the nodes corresponding to each ring substructure were extracted using RDKit's functions, `GetRingInfo` and `GetSymmSSSR`. We restricted our vocabulary to the unique ring substructures in the molecules. The vocabulary of clusters follows a skewed distribution over the frequency of appearance within the dataset. In particular, only a subset (~ 30) of ring substructures (labels) appear with high frequency in molecules within the vocabulary. Therefore, we simplify the vocabulary by only representing the 30 commonly occurring substructures of $\mathcal{A}_{tree}$. In Figure 8, we show some examples of these ring substructures for the two datasets.



(a) QM9 Dataset (b) ZINC250K Dataset

Figure 8: Examples of frequently occurring ring substructures

A.3 Equivariant Graph Neural Networks

Equivariant Graph Neural Networks (EGNN) (Satorras et al., 2021) are E(3)-equivariant with respect to an input set of points. The E(3) equivariance accounts for translation, rotation, and reflection symmetries, and can be extended to E(n) group equivariance. The inherent dynamics governing the EGNN can be described, for each layer l, as follows. Here, $\mathbf{h}_i^l$ and $\mathbf{x}_i^l$ pertain, respectively, to the embedding for the node i and that for its coordinates; and a_{ij} abstracts the information about the edge between nodes i and j.

$$\begin{aligned}
\mathbf{m}_{ij} &= \phi_e\left(\mathbf{h}_i^l, \mathbf{h}_j^l, \left\|\mathbf{x}_i^l - \mathbf{x}_j^l\right\|^2, a_{ij}\right) \\
\mathbf{x}_i^{l+1} &= \mathbf{x}_i^l + C\sum_{j \neq i}\left(\mathbf{x}_i^l - \mathbf{x}_j^l\right)\phi_x\left(\mathbf{m}_{ij}\right) \\
\mathbf{m}_i &= \sum_{j \neq i} \mathbf{m}_{ij} \\
\mathbf{h}_i^{l+1} &= \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)
\end{aligned}$$

Initially, messages $\mathbf{m}_{ij}$ are computed between the neighboring nodes via ϕ_e. Subsequently, the coordinates of each node i are updated via a weighted sum of relative position vectors $\{(\mathbf{x}_i - \mathbf{x}_j) : j \neq i\}$ with the aid of ϕ_x. Finally, the node embeddings are updated based on the aggregated messages $\mathbf{m}_i$ via ϕ_h. The aggregated message can be computed based on only the neighbors of a node by simply replacing the sum over $j \neq i$ with a sum over $j \in \mathcal{N}_i$ in these equations.

Table 7: `ModFlow` as a temporal graph network (TGN). Adopting notation for TGNs from Rossi et al. (2020) v_i is a node-wise event on i; e_{ij} denotes an (asymmetric) interaction between i and j; $\mathbf{s}_i$ is the memory of node i; and t and t^- denote time with t^- being the time of last interaction before t, e.g., $\mathbf{s}_i(t^-)$ is the memory of i just before time t; and msg and agg are learnable functions (e.g., MLP) to compute, respectively, the individual and the aggregate messages. For `ModFlow`, we use $\mathbf{r}_{ij}$ to denote the spatial distance $\mathbf{x}_i - \mathbf{x}_j$, and a_{ij} to denote the attributes of the edge between i and j. The functions ϕ_e, ϕ_x, and ϕ_h are as defined in Satorras et al. (2021), and summarized in A.3.

Method	TGN layer	`ModFlow`
Edge	$\mathbf{m}'_{ij}(t) = \mathrm{msg}\left(\mathbf{s}_i\left(t^-\right), \mathbf{s}_j\left(t^-\right), \Delta t, \mathbf{e}_{ij}(t)\right)$ $\overline{\mathbf{m}}'_i(t) = \mathrm{agg}\left(\{\mathbf{m}'_{ij}\left(t\right) \mid j \in \mathcal{N}_i\}\right)$	$\mathbf{m}_{ij}(t) = \phi_e\left(\mathbf{z}_i(t), \mathbf{z}_j(t), \left\|\mathbf{r}_{ij}(t)\right\|^2, a_{ij}\right)$ $\mathbf{m}_i(t) = \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_{ij}(t) = \mathbf{r}_{ij}(t) \cdot \phi_x\left(\mathbf{m}_{ij}(t)\right)$ $\hat{\mathbf{m}}_i(t) = C \sum_{j \in \mathcal{N}(i)} \hat{\mathbf{m}}_{ij}(t)$
Memory state	$\mathbf{s}_i(t) = \mathrm{mem}\left(\overline{\mathbf{m}}'_i(t), \mathbf{s}_i\left(t^-\right)\right)$	$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \hat{\mathbf{m}}_i(t)$
Node	$\mathbf{z}'_i(t) = \sum_{j \in \mathcal{N}_i} h\left(\mathbf{s}_i(t), \mathbf{s}_j(t), \mathbf{e}_{ij}(t), \mathbf{v}_i(t), \mathbf{v}_j(t)\right)$	$\mathbf{z}_i(t+1) = \phi_h\left(\mathbf{z}_i(t), \mathbf{m}_i(t)\right)$

A.4 Connection to Temporal Graph Networks

Temporal Graph Networks (Rossi et al., 2020; Souza et al., 2022) are state-of-the-art neural models for embedding dynamic graphs. A prominent class of these models consists of a combination of (recurrent) memory modules and graph-based operators, and rely on message passing for updating the embeddings based on node-wise or edge-wise *events*.

Specifically, adopting the notation from Rossi et al. (2020), an interaction $\mathbf{e}_{ij}(t)$ between any two nodes i and j at time t triggers an edge-wise event leading to the following steps. First, a message $\mathbf{m}'_{ij}(t)$ is computed based on the memory $\mathbf{s}_i\left(t^-\right)$ and $\mathbf{s}_j\left(t^-\right)$ of the two nodes just before time t via a learnable function msg (such as multilayer perceptron). For each node i, the messages thus accrued over a small period due to interactions of with neighbors j are combined (via agg) into an aggregate message $\overline{\mathbf{m}}'_i(t)$. This message, in turn, is used to update the memory of i to $\mathbf{s}_i(t)$ via mem (implemented e.g., as a recurrent neural network). Finally, the node embedding of i is revised based on its memory $\mathbf{s}_i(t)$, interaction $\mathbf{e}_{ij}(t)$ and memory $\mathbf{s}_j(t)$ of each neighbor $j \in \mathcal{N}_i$, as well as any additional node-wise events $\mathbf{v}_i(t)$ involving i or any node in $\mathcal{N}_i$.

It turns out (see Table 7) that `ModFlow` can be viewed as an equivariant message passing temporal graph network. Interestingly, the coordinate embedding $\mathbf{x}_i$ plays the role of the memory $\mathbf{s}_i$.

A.5 Implementation Details

We implemented the proposed models in PyTorch (Paszke et al., 2019).[1] We used a single layer for EGNN with embedding dimension 32 and aggregated information for each node from only its immediate neighbors. For geometric (spatial) information, we worked with the polar coordinates (2D) or the spherical polar coordinates (3D). We solved the ODE system with the Dormand–Prince adaptive step size scheme (i.e., the `dopri5` solver). The number of function evaluations lay roughly between 70 and 100. The models were trained for 50-100 epochs with the Adam (Kingma and Ba, 2014) optimizer.

Time comparisons. We found the training time of `ModFlow` to be slightly worse than one-shot discrete flow models that characterize the whole system using a single flow (recall that, in contrast, `ModFlow` associates an ODE with each node). However, `ModFlow` is faster to train than the auto-regressive methods.

Note that computation is a crucial aspect of generative modeling for application domains with a huge search space, as is true for the molecules. We report the computational effort (excluding the time for preprocessing) for generating 10000 molecules averaged across 5 independent runs in Table 8. Notably, largely by virtue of being one-shot, `ModFlow` is able to generate significantly faster than the auto-regressive models such as GraphAF and GraphDF. `ModFlow` also owes this speedup, in part, to obviate the need for multiple decoding (unlike, e.g., JT-VAE) as well as any validity checks.

[1] We make the code available at `https://github.com/yogeshverma1998/Modular-Flows-Neurips-2022`.

Table 8: Generation time (in seconds/molecule) on QM9 and ZINC250K.

Method	ZINC250K	QM9
GraphEBM	1.12 ± 0.34	0.53 ± 0.16
GVAE	0.86 ± 0.12	0.46 ± 0.07
GraphAF	0.93 ± 0.14	0.56 ± 0.12
GraphDF	3.12 ± 0.56	1.92 ± 0.42
MoFlow	0.71 ± 0.14	0.31 ± 0.04
`ModFlow` (2D-EGNN)	0.46 ± 0.09	0.16 ± 0.04
`ModFlow` (3D-EGNN)	0.55 ± 0.13	0.24 ± 0.06
`ModFlow` (JT-2D-EGNN)	0.53 ± 0.07	0.21 ± 0.07
`ModFlow` (JT-3D-EGNN)	0.62 ± 0.11	0.28 ± 0.09

A.6 Additional Evaluation Metrics

We invoked additional metrics, namely the MOSES metrics (Polykovskiy et al., 2020), to compare the different models in terms of their ability to generate molecules. These metrics, described below, access the overall quality of the generated molecules.

- **FCD**: *Fréchet Chemnet Distance* (FCD) (Preuer et al., 2018) is a general purpose metric that measures diversity of the generated molecules, as well as the extent of their chemical and biological property alignment with a reference set of real molecules. Specifically, the last layer activations of ChemNet are used for this purpose. Lower is better.
- **Frag**: *Fragment similarity* (Frag), measures the cosine distance between the fragment frequencies of the generated molecules and a set of reference molecules. Higher is better.
- **SNN**: *Nearest Neighbor Similarity* (SNN) quantifies how close the generated molecules are to the true molecule manifold. Specifically, it computes the average similarity of a generated molecule to the nearest molecule from the reference set. Higher is better.
- **IntDiv**: As the name suggests, *Internal Diversity* (IntDiv) accounts for diversity by computing the average pairwise similarity of the generated molecules. Higher is better.

For our purpose, we evaluated these metrics with QM9 and ZINC250K as the reference sets. As shown in Table 9 and Table 10, `ModFlow` achieves better performance results across all metrics. Notably, `ModFlow` registers lower FCD and higher IntDiv scores compared to other methods, suggesting that `ModFlow` is able to generate diverse set of molecules similar to those present in the real datasets.

Table 9: Evaluation of performance on MOSES metrics on generative models on QM9 dataset. FCD is lower the better, Frag, SNN, and IntDiv higher the better.

Method	FCD (↓)	Frag (↑)	SNN (↑)	IntDiv (↑)
GVAE	0.513	0.821	0.582	0.822
GraphEBM	0.551	0.831	0.547	0.831
GraphAF	0.732	0.863	0.565	0.823
GraphDF	0.683	0.892	0.562	0.839
MoFlow	0.496	0.840	0.502	0.852
`ModFlow` (2D-EGNN)	0.432	0.928	0.608	0.875
`ModFlow` (3D-EGNN)	0.478	0.934	0.613	0.885
`ModFlow` (JT-2D-EGNN)	0.421	0.921	0.595	0.867
`ModFlow` (JT-3D-EGNN)	**0.401**	**0.939**	**0.624**	**0.889**

Table 10: Evaluation of performance on MOSES metrics on generative models on ZINC250K dataset. FCD is lower the better, Frag, SNN, and IntDiv higher the better.

Method	FCD (↓)	Frag (↑)	SNN (↑)	IntDiv (↑)
JTVAE	0.512	0.890	0.5477	0.855
GVAE	0.571	0.871	0.532	0.852
GraphEBM	0.613	0.843	0.487	0.821
GraphAF	0.524	0.803	0.465	0.855
GraphDF	0.658	0.869	0.515	0.829
MoFlow	0.597	0.851	0.452	0.832
`ModFlow` (2D-EGNN)	0.495	0.891	0.570	0.863
`ModFlow` (3D-EGNN)	0.512	0.905	0.584	0.869
`ModFlow` (JT-2D-EGNN)	**0.501**	0.915	0.563	0.857
`ModFlow` (JT-3D-EGNN)	0.523	**0.929**	**0.594**	**0.879**

We also evaluated the generated structures via distributions of their important properties. Specifically, we obtained kernel density estimates of these distributions to aid in visualization. We consider the following standard properties.

- **Weight**: sum of the individual atomic weights of a molecule. The weight provides insight into the bias of the generated molecules toward lighter or heavier molecules.
- **LogP**: ratio of concentration in octanol-phase to the aqueous phase, also known as the octanol-water partition coefficient. It is computed via the Crippen (Wildman and Crippen, 1999) estimation.
- **Synthetic Accessibility (SA)**: an estimate for the synthesizability of a given molecule. It is calculated based on contributions of the molecule fragments Ertl and Schuffenhauer (2009).
- **Quantitative Estimation of Drug-likeness (QED)**: describes the likeliness of a molecule as a viable candidate for a drug. It ranges between [0,1] and captures the abstract notion of aesthetics in medicinal chemistry (Bickerton et al., 2012).

Figure 9 and Figure 10 show that barring some dispersion in QED and logP (especially on Zinc250K), the property distributions of the molecules generated by `ModFlow` generally match the corresponding distributions on the reference datasets quite closely. These results demonstrate the effectiveness of `ModFlow` in generating molecules that have properties similar to the molecules in the reference set.

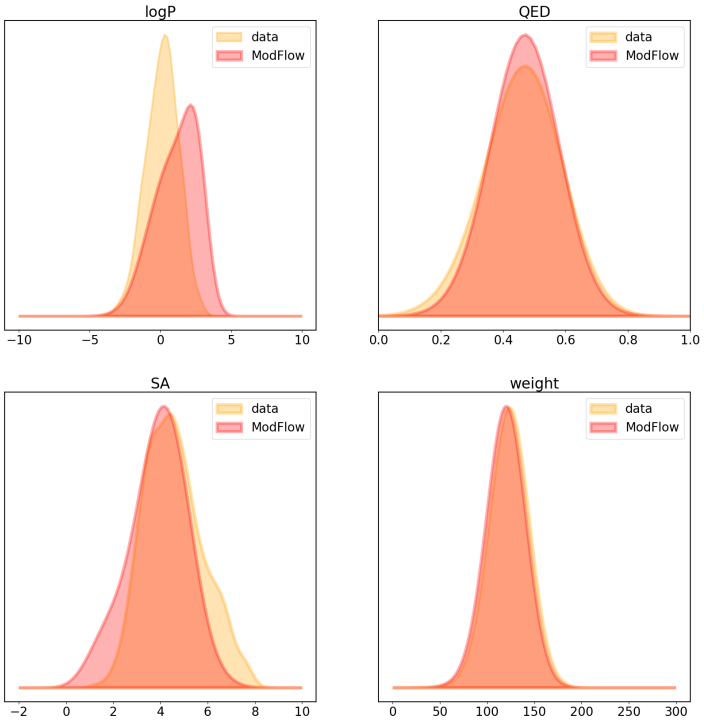


Figure 9: **(QM9)** Distributions of the chemical properties.

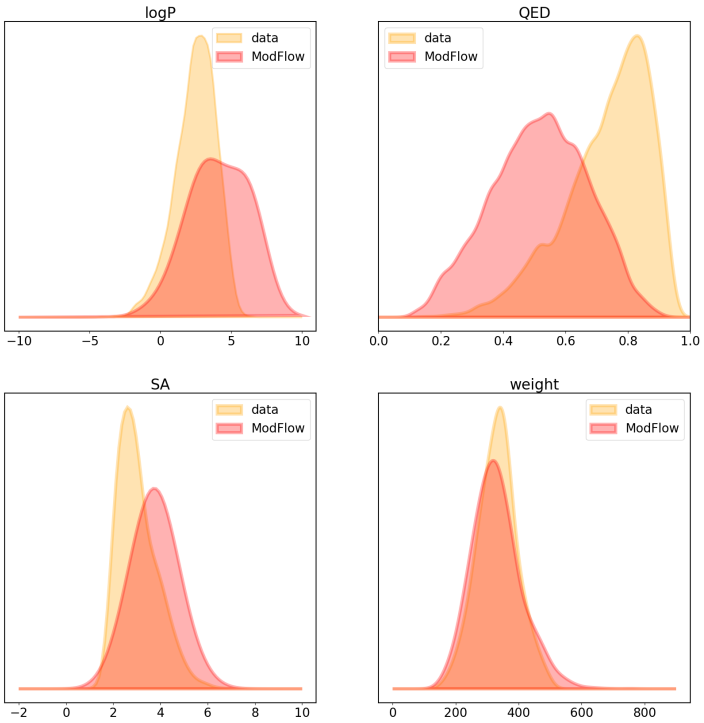


Figure 10: **(ZINC250K)** Distributions of the chemical properties.

A.7 Additional examples of molecules generated by ModFlow .