Modular Flows: Differential Molecular Generation

Yogesh Verma, Samuel Kaski, Markus Heinonen
Aalto University
Espoo, Finland
`{yogesh.verma, samuel.kaski, markus.heinonen} @aalto.fi`

Vikas Garg
Aalto University; FCAI; YaiYai Ltd
Espoo, Finland
`vikas.garg@aalto.fi, vikas@yaiyai.fi`

Abstract

Generating new molecules is fundamental to advancing critical applications such as drug discovery and material synthesis. Flows can generate molecules effectively by inverting the encoding process, however, existing flow models either require artifactual dequantization or specific node/edge orderings, lack desiderata such as permutation invariance, or induce discrepancy between encoding and decoding steps that necessitates *post hoc* validity correction. We circumvent these issues with novel continuous normalizing E(3)-equivariant flows, based on a system of node ODEs coupled as a graph PDE, that repeatedly reconcile locally toward globally aligned densities. Our models can be cast as message passing temporal networks, and result in superlative performance on the tasks of density estimation and molecular generation. In particular, our generated samples achieve state of the art on both the standard QM9 and ZINC250K benchmarks.

1 Introduction

Generative models have rapidly become ubiquitous in machine learning with advances from image synthesis (Ramesh et al., 2022) to protein design (Ingraham et al., 2019). Molecular generation (Stokes et al., 2020) has also received significant attention in discovering new drugs and materials. However, searching for valid molecules in large discrete spaces is challenging: drug-like structures range between 10^{23} and 10^{60} but only a tiny fraction ($\sim 10^8$) - has been synthesized (Polishchuk et al., 2013; Merz et al., 2020). Thus, learning representations that exploit appropriate molecular inductive biases (e.g., spatial correlations) becomes crucial. Earlier models focused on generating sequences based on the SMILES notation (Weininger, 1988), which were supplanted by generative models that capture valuable spatial information via embedding molecular with some graph neural network (GNNs) (Scarselli et al., 2009; Garg et al., 2020). This include variants of GANs (Goodfellow et al., 2014; Maziarka et al., 2020) which suffer the problem of mode collapse,

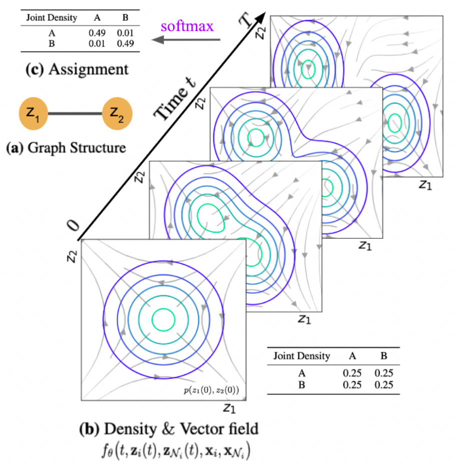


Figure 1: A toy illustration of `ModFlow` in action with a two-node graph. The two local flows - $\mathbf{z}_1$ and $\mathbf{z}_2$ - co-evolve toward a more complex joint density, both driven by the same differential f.

Variational Auto Encoders (VAE) (Kingma and Welling, 2013; Lim et al., 2018; Jin et al., 2018) which are susceptible to a distributional shift between the training data and the generated samples, and Normalizing Flows (Dinh et al., 2014, 2016). Flows are appealing since they enable estimating (and sampling from) complex data distributions using a sequence of invertible transformations from a tractable continuous distribution. Molecules are discrete, so many flow models (Madhawa et al., 2019; Honda et al., 2019; Shi et al., 2020) add noise during encoding leading to *dequantization* procedure which begets distortion and issues related to convergence (Luo et al., 2021). Moreover, many methods employ *post hoc* correction to ensure validity (Zang and Wang, 2020), effecting a discrepancy between the encoding and the decoded distributions.

We propose a coupled continuous normalizing E(3)-equivariant flows tailored to molecule generation, that bestow generative capabilities from neural partial differential equation (PDE) models on graphs (Chamberlain et al., 2021; Poli et al., 2019; Iakovlev et al., 2020). We seek to bring their efficacy and elegance as tools to generate complex objects, such as molecules. Specifically, a flow is associated with each node, which is conjoined as a joint ODE system conditioned on neighboring nodes. While these flows originate independently, they adjust progressively toward more complex joint distributions via interacting with the neighboring flows. We call the proposed method Modular Flows (`ModFlow`) to underscore that each node can be regarded as a module that coordinates with other modules.

2 Modular Flows

We focus on unsupervised learning of an underlying graph density $p(G)$ using a dataset $\mathcal{D} = \{G_n\}_{n=1}^N$ of observed molecular graphs G_n. We learn a generative flow model $p_\theta(G)$ specified by flow parameters θ and use it to sample novel high-probability molecules.

2.1 Molecular Representation

Graph representation. We represent the molecular graph $G = (V, E)$ as a tuple of vertices $V = (v_1, \ldots, v_M)$ and edges $E \subset V \times V$. Each vertex takes value as: $v \in \mathcal{A} = \{\mathtt{C}, \mathtt{H}, \mathtt{N}, \ldots\}$; while the edges $e \in \mathcal{B} = \{1, 2, 3\}$ represent the type of bond. We assume that conditioned on the edges, the graph likelihood factorizes as a Categorical distribution over vertices given their latent representations:

$$p(G) := p(V|E, \{z\}) = \prod_{i=1}^{M} \mathrm{Cat}(v_i|\sigma(\mathbf{z}_i)), \tag{1}$$

where $\mathbf{z}_i = (z_{i\mathtt{C}}, z_{i\mathtt{H}}, \ldots) \in \mathbb{R}^{|\mathcal{A}|}$ is a set of atom score parameters of node i, and σ is the softmax function. We can obtain an alternative tree representation of molecules, similar to Jin et al. (2018) where we restrict clusters to be ring sub-structures. We obtain an extended alphabet $\mathcal{A}_{\text{tree}} = \{\mathtt{C}, \mathtt{H}, \mathtt{N}, \ldots, \mathtt{C}_1, \mathtt{C}_2, \ldots\}$, where each cluster label $\mathtt{C}_r$ corresponds to the some ring-substructure in the label vocabulary $\mathcal{X}$. For more details see Appendix A.2.

2.2 Differential modular flows

We propose to model the atom scores $\mathbf{z}_i(t)$ as a Continuous-time Normalizing Flow (CNF) (Grathwohl et al., 2018) over time $t \in \mathbb{R}_+$. We assume the initial scores at time $t = 0$ as $\mathbf{z}_i(0) \sim \mathcal{N}(0, I)$ for each node i. Node scores evolve in parallel over time by a differential equation, where $\mathcal{N}_i$ is the set of neighbors, $\mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}$ is the positional information, and θ are the parameters of the flow function. By collecting all node differentials we obtain a *modular* joint, coupled ODE, which is our key contribution:,

$$\dot{\mathbf{z}}(t) = \begin{pmatrix} \dot{\mathbf{z}}_1(t) \\ \vdots \\ \dot{\mathbf{z}}_M(t) \end{pmatrix} = \begin{pmatrix} f_\theta\big(t, \mathbf{z}_1(t), \mathbf{z}_{\mathcal{N}_1}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big) \\ \vdots \\ f_\theta\big(t, \mathbf{z}_M(t), \mathbf{z}_{\mathcal{N}_M}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big) \end{pmatrix} \tag{2}$$

$$\mathbf{z}(T) = \mathbf{z}(0) + \int_0^T \dot{\mathbf{z}}(t)dt. \tag{3}$$

The above system is usually solved via an ODE solver where gradients are computed via adjoint sensitivity method (Kolmogorov et al., 1962), which incurs a low memory cost, and explicitly controls numerical error. Notably, moving towards modular flows translates sparsity also to the adjoints.

Proposition 1: *Modular adjoints are sparser than regular adjoints. They can be computed by*

$$\frac{d\boldsymbol{\lambda}_i}{dt} = -\sum_{j\in\mathcal{N}_i\cup\{i\}} \boldsymbol{\lambda}_j(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}_j}, \tag{4}$$

where the partial derivatives $\frac{\partial f}{\partial \mathbf{z}} = [\frac{\partial f_i}{\partial \mathbf{z}_j}]_{ij}$ *are sparse* (See Appendix A.1 for further details).

Equivariant local differential The differential function f must satisfy the natural equivariances and invariances of molecules like translation, rotational (and reflection) equivariant and permutation equivariances. Therefore, we chose to use E(3)-Equivariant GNN (EGNN) (Satorras et al., 2021), which satisfies all the above criteria (See Appendix A.3 for details). The input to the EGNN are the node embeddings and the geometric information (polar coordinates (2D) and spherical polar coordinates (3D)). Interestingly, `ModFlow` can be viewed as a message passing temporal graph network (Rossi et al., 2020) as shown next.

Proposition 2: *Modular Flows can be cast as message-passing Temporal Graph Networks (TGNs). The operations are listed in Table 2, where* `ModFlow` *is subjected to a single layer of EGNN.* (See Appendix A.4 for more details).

2.3 Training objective

Normalizing flows are trained to minimize KL divergence $\mathrm{KL}[p_{\mathrm{data}}||p_\theta]$ between the unknown data distribution p_{data} and the flow-generated distribution p_θ, which is equivalent to maximizing their cross-entropy $\mathbb{E}_{p_{\mathrm{data}}}[\log p_\theta]$ (Papamakarios et al., 2021). However, this requires a mapping from discrete graphs G to continuous atom scores $\mathbf{z}(t)$. We reduce the learning problem to maximizing the score cross-entropy $\mathbb{E}_{\hat{p}_{\mathrm{data}}(\mathbf{z}(T))}[\log p_\theta(\mathbf{z}(T))]$, where we turn the observed set of graphs $\{G_n\}$ into a set of scores $\{\mathbf{z}_n\}$ by using one-hot encoding

$$\mathbf{z}_n(G_n;\epsilon) = (1-\epsilon)\,\mathrm{onehot}(G_n) + \frac{\epsilon}{|\mathcal{A}_f|}\mathbf{1}_{M(n)}\mathbf{1}_{|\mathcal{A}_f|}^\top,$$

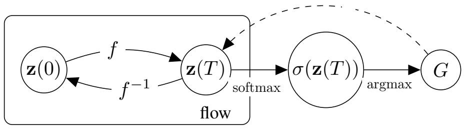


Figure 2: Plate diagram showing both the inference and generative components of `ModFlow`.

where $\mathrm{onehot}(G_n)$ is a matrix of size $M(n)\times|\mathcal{A}_f|$ such that $G_n(i,k) = 1$ if $v_i = a_k \in \mathcal{A}_f$, $\mathbf{1}_q$ is a vector with q entries each set to 1; $\mathcal{A}_f \in \{\mathcal{A}, \mathcal{A}_{\mathrm{tree}}\}$; and $\epsilon\in[0,1]$ is added to model the noise in estimating the posterior $p(\mathbf{z}(T)|G)$ due to short-circuiting the inference process as shown in Fig.2. We exploit the (non-reversible) composition of the argmax and softmax operations to short-circuit the process, which keeps the forward and backward flows aligned. We thus maximize an objective over N training graphs,

$$\arg\max_\theta \quad \mathcal{L} = \mathbb{E}_{\hat{p}_{\mathrm{data}}(\mathbf{z})}\log p_\theta(\mathbf{z}) \quad \approx \frac{1}{N}\sum_{n=1}^{N}\log p_T\big(\mathbf{z}(T) = \mathbf{z}_n\big) \tag{5}$$

$$= \frac{1}{N}\sum_{n=1}^{N}\left(\sum_{i=1}^{M(n)}\left[\log p_0(\mathbf{z}_i(0)) - \int_0^T \mathrm{tr}\frac{\partial f_\theta(t,\mathbf{z}_i(t),\mathbf{z}_{\mathcal{N}_i}(t),\mathbf{x}_i,\mathbf{x}_{\mathcal{N}_i})}{\partial \mathbf{z}_i(t)}dt\right]\right), \tag{6}$$

which factorizes over the size $M(n)$ of the n'th training molecule. In practice, we solve ODE integrals using numerical solvers, such as Runge-Kutta. We delegate this task to a general solver of the form $\texttt{ODESolve}(\mathbf{z}, f_\theta, T)$, where map f_θ is applied for T steps starting on $\mathbf{z}$. An optimizer `optim` is also required for updating θ.

2.4 Molecular generation

We generate novel molecules by sampling an initial state $\mathbf{z}(0) \sim \mathcal{N}(0, I)$ based on structure, and running the modular flow forward in time until $\mathbf{z}(T)$. This procedure maps a tractable base distribution p_0 to some more complex distribution p_T. We follow argmax to pick the most probable label assignment for each node (Zang and Wang, 2020).

3 Experiments

We evaluated `ModFlow` models trained, variously, on 2D and 3D coordinates, and their tree representation respectively on the task of a molecular generation. Notably, `ModFlow` achieves state-of-the-art results without validity checks or post hoc correction when evaluated against many existing state-of-the-art methods.

We train and evaluate on ZINC250k (Irwin et al., 2012) and QM9 (Ramakrishnan et al., 2014) datasets. The molecules are in *kekulized* form with hydrogens removed by the RDkit software (Landrum et al., 2013). We used **Validity**, **Uniqueness**, **Novelty** and **Reconstruction** as metrics. These metrics measure the molecules obeying valency rules, non-duplicate generations, not present in the dataset, and the fraction of molecules that can be reconstructed from their encoding. We report the mean and the standard deviation from five different initialization and generate 50,000 molecules for evaluation. All the implementation is done in PyTorch (Paszke et al., 2019). The input concatenates time and scalar vocabulary scores, per node. For more details, see Appendix A.5.

Tables 1 shows the results on QM9 and ZINC250K. `ModFlow` achieves state-of-the-art results across all metrics. Notably, its reconstruction rate is 100% similar to other flow models; novelty and uniqueness scores are also very high. Moreover, `ModFlow` surpassed all early methods on validity (95%-99%). Additional results on property optimization and density estimation are shown in Appendix A.6 and A.7.

Table 1: Random generation on QM9 (top) and ZINC250K (bottom) without post hoc validity corrections. Results with * are taken from Luo et al. (2021). Higher values are better for all columns.

Method	Validity %	Uniqueness %	Novelty %	Reconstruction %
GVAE	60.2	9.3	80.9	96.0
GraphNVP*	83.1	99.2	58.2	100
GRF*	84.5	66	58.6	100
GraphAF*	67	94.2	88.8	100
GraphDF*	82.7	97.6	98.1	100
MoFlow*	89.0	98.5	96.4	100
`ModFlow` (2D-EGNN)	**96.2** $\pm$ 1.7	**99.5**	**100**	100
`ModFlow` (3D-EGNN)	**98.3** $\pm$ 0.7	99.1	**100**	100
`ModFlow` (JT-2D-EGNN)	**97.9** $\pm$ 1.2	99.2	**100**	100
`ModFlow` (JT-3D-EGNN)	**99.1** $\pm$ 0.8	99.3	**100**	100

Method	Validity %	Uniqueness %	Novelty %	Reconstruction %
MRNN	65	99.89	100	n/a
GraphNVP*	42.6	94.8	100	100
GRF*	73.4	53.7	100	100
GraphAF*	68	99.1	100	100
GraphDF*	89	99.2	100	100
MoFlow*	50.3	**99.9**	100	100
`ModFlow` (2D-EGNN)	**94.8** $\pm$ 1.0	99.4	100	100
`ModFlow` (3D-EGNN)	**95.4** $\pm$ 1.2	99.7	100	100
`ModFlow` (JT-2D-EGNN)	**97.4** $\pm$ 1.4	99.1	100	100
`ModFlow` (JT-3D-EGNN)	**98.1** $\pm$ 0.9	99.3	100	100

4 Conclusion

We proposed `ModFlow`, a new generative flow model where multiple flows interact locally according to a coupled ODE, resulting in accurate modeling of graph densities and high-quality molecular generation without any validity checks or correction. Interesting avenues open up, including the design of (a) more nuanced mappings between discrete and continuous spaces, and (b) extensions of modular flows to (semi-)supervised settings.

5 Acknowledgments

The calculations were performed using resources within the Aalto University Science-IT project. This work has been supported by the Academy of Finland under the *HEALED* project (grant 13342077).

References

Andrew M. Bradley. PDE-constrained optimization and the adjoint method. Technical report, 2019.

Ben Chamberlain, James Rowbottom, Maria I Gorinova, Michael Bronstein, Stefan Webb, and Emanuele Rossi. Grand: Graph neural diffusion. In *International Conference on Machine Learning*, pages 1407–1418. PMLR, 2021.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

Vikas Garg, Stefanie Jegelka, and Tommi Jaakkola. Generalization and representational limits of graph neural networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3419–3430. PMLR, 2020. URL `http://proceedings.mlr.press/v119/garg20c.html`.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. URL `https://arxiv.org/abs/1406.2661`.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Shion Honda, Hirotaka Akita, Katsuhiko Ishiguro, Toshiki Nakanishi, and Kenta Oono. Graph residual flow for molecular graph generation. *arXiv preprint arXiv:1909.13521*, 2019.

Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. *arXiv preprint arXiv:2006.08956*, 2020.

John Ingraham, Vikas Garg, Regina Barzilay, and Tommi Jaakkola. Generative models for graph-based protein design. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/f3a4ff4839c56a5f460c88cce3666a2b-Paper.pdf`.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL `https://arxiv.org/abs/1312.6114`.

Andrei Nikolaevich Kolmogorov, Ye F Mishchenko, and Lev Semenovich Pontryagin. A probability problem of optimal control. Technical report, JOINT PUBLICATIONS RESEARCH SERVICE ARLINGTON VA, 1962.

Greg Landrum et al. Rdkit: A software suite for cheminformatics, computational chemistry, and predictive modeling, 2013.

Jaechang Lim, Seongok Ryu, Jin Woo Kim, and Woo Youn Kim. Molecular generative model based on conditional variational autoencoder for de novo molecular design. *Journal of cheminformatics*, 10(1):1–9, 2018.

Youzhi Luo, Keqiang Yan, and Shuiwang Ji. Graphdf: A discrete flow model for molecular graph generation, 2021. URL `https://arxiv.org/abs/2102.01189`.

Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Lukasz Maziarka, Agnieszka Pocha, Jan Kaczmarczyk, Krzysztof Rataj, Tomasz Danel, and Michał Warchoł. Mol-cyclegan: a generative model for molecular optimization. *Journal of Cheminformatics*, 12(1):1–18, 2020.

Kenneth M. Merz, Gianni De Fabritiis, and Guo-Wei Wei. Generative models for molecular design. *Journal of Chemical Information and Modeling*, 60(12):5635–5636, 2020. doi: 10.1021/acs.jcim.0c01388. URL `https://doi.org/10.1021/acs.jcim.0c01388`. PMID: 33378853.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57): 1–64, 2021.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. Estimation of the size of drug-like chemical space based on gdb-17 data. *Journal of computer-aided molecular design*, 27(8):675–679, 2013.

Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022. URL `https://arxiv.org/abs/2204.06125`.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs, 2020. URL `https://arxiv.org/abs/2006.10637`.

Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E(n) equivariant graph neural networks, 2021. URL `https://arxiv.org/abs/2102.09844`.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Chence Shi, Minkai Xu, Zhaocheng Zhu, Weinan Zhang, Ming Zhang, and Jian Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. *arXiv preprint arXiv:2001.09382*, 2020.

Amauri Souza, Diego Mesquita, Samuel Kaski, and Vikas Garg. Provably expressive temporal graph networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Jonathan M. Stokes, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M. Donghia, Craig R. MacNair, Shawn French, Lindsey A. Carfrae, Zohar Bloom-Ackermann, Victoria M. Tran, Anush Chiappino-Pepe, Ahmed H. Badran, Ian W. Andrews, Emma J. Chory, George M. Church, Eric D. Brown, Tommi S. Jaakkola, Regina Barzilay, and James J. Collins. A deep learning approach to antibiotic discovery. *Cell*, 180(4):688–702.e13, 2020. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2020.01.021. URL `https://www.sciencedirect.com/science/article/pii/S0092867420301021`.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Chengxi Zang and Fei Wang. Moflow: an invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 617–626, 2020.

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

$$\dot{\mathbf{z}}(t) = f(\mathbf{z}, t, \boldsymbol{\theta}) \tag{7}$$

$$\mathbf{z}(t) = \mathbf{z}_0 + \int_0^t f(\mathbf{z}, t, \boldsymbol{\theta}) d\tau, \tag{8}$$

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

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & G(\boldsymbol{\theta}) = \int_0^T g(\mathbf{z}, t, \boldsymbol{\theta}) dt && (9) \\ s.t. \quad & \dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta}) = 0, \qquad \forall t \in [0, T] && (10) \\ & \mathbf{z}(0) - \mathbf{z}_0 = 0, && (11) \end{aligned}$$

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

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= G(\boldsymbol{\theta}) + \int_0^T \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) && (12) \\ &= \int_0^T \Big[g(\mathbf{z}, t, \boldsymbol{\theta}) + \boldsymbol{\lambda}(t)^\top (\dot{\mathbf{z}} - f(\mathbf{z}, t, \boldsymbol{\theta})) \Big] dt + \boldsymbol{\mu}^\top (\mathbf{z}(0) - \mathbf{z}_0) \,. && (13) \end{aligned}$$

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

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} G = \int_0^T \Big[g_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} + g_{\boldsymbol{\theta}} + \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} \mathbf{z}_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}} \Big] dt \,, \tag{14}$$

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

$$\int_0^T \boldsymbol{\lambda}^\top \dot{\mathbf{z}}_{\boldsymbol{\theta}} dt = \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T} - \boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0} - \int_0^T \dot{\boldsymbol{\lambda}}^\top \mathbf{z}_{\boldsymbol{\theta}} dt \,. \tag{15}$$

Substituting this into previous equation and rearranging the terms results in

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T \underbrace{(g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} - \dot{\boldsymbol{\lambda}}^\top) \mathbf{z}_{\boldsymbol{\theta}}}_{0, \text{ if } \dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}} dt + \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt + \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=T}}_{0, \text{ if } \boldsymbol{\lambda}(T) = \mathbf{0}} - \underbrace{\boldsymbol{\lambda}^\top \mathbf{z}_{\boldsymbol{\theta}}|_{t=0}}_{0} \,. \tag{16}$$

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

$$\dot{\boldsymbol{\lambda}}^\top = g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}}. \tag{17}$$

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

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} G = \nabla_{\boldsymbol{\theta}} \mathcal{L} &= \int_0^T (g_{\boldsymbol{\theta}} - \boldsymbol{\lambda}^\top f_{\boldsymbol{\theta}}) dt && (18) \\ s.t. \quad \dot{\boldsymbol{\lambda}}^\top &= g_{\mathbf{z}} - \boldsymbol{\lambda}^\top f_{\mathbf{z}} && (19) \\ \boldsymbol{\lambda}(T) &= 0. && (20) \end{aligned}$$

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

$$g(\mathbf{z}, t, \boldsymbol{\theta}) = \delta(t = t_j)\tilde{g}(\mathbf{z}, \mathbf{y}_j, t, \boldsymbol{\theta}), \tag{21}$$

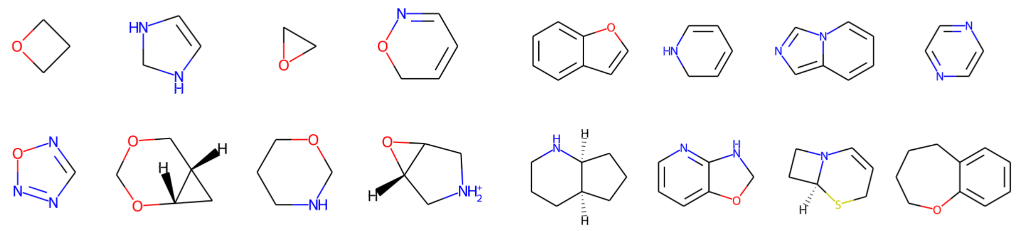
and now the term $g_{\mathbf{z}}$ induces discontinuous jumps at observations. This does not pose problems in practice, since we can integrate the ODE in continuous segments between the observation instants.

The sparsity of the adjoint evolution is evident from Equation 19, 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

$$\frac{d\boldsymbol{\lambda}_i}{dt} = -\boldsymbol{\lambda}(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}} = -\sum_{j \in \mathcal{N}_i \cup \{i\}} \boldsymbol{\lambda}_j(t)^\top \frac{\partial f\big(t, \mathbf{z}_i(t), \mathbf{z}_{\mathcal{N}_i}(t), \mathbf{x}_i, \mathbf{x}_{\mathcal{N}_i}\big)}{\partial \mathbf{z}_j}. \tag{22}$$

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 ($\sim$ 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 3, we show some examples of these ring substructures for the two datasets.



(a) QM9 Dataset (b) ZINC250K Dataset

Figure 3: Examples of frequently occurring ring substructures

A.3 Equivariant Graph Neural Networks (EGNN)

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.

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 $\mathcal{N}_i$.

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

It turns out (see Table 2) 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 3. 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.

A.6 Density Estimation

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

Table 3: 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

We generated our synthetic data in the following way. We considered two variants of a chessboard pattern, namely, (i) 4×4 grid where every node takes a binary value, 0 or 1, and neighboring nodes have different values; and (ii) 16×16 grid where nodes in each block of 4×4 all take the same value (0 or 1), different from the adjacent blocks. We also experimented with a 20×20 grid describing alternating stripes of 0s and 1s.

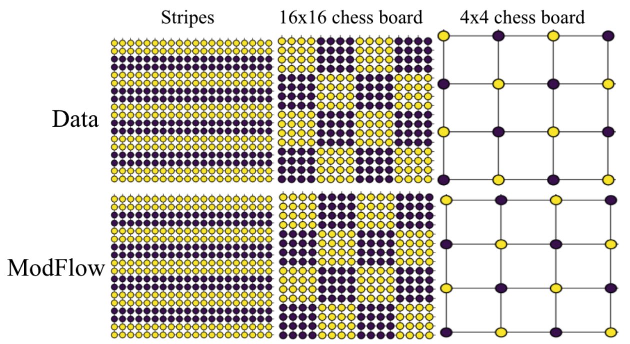


Figure 4: `ModFlow` can accurately learn to reproduce complex, discontinuous graph patterns.

Figure 4 shows that `ModFlow` can learn neural differential functions f_θ that reproduce the patterns almost perfectly, indicating sufficient capacity to model complex patterns. That is, `ModFlow` is able to transform the initial Gaussian distribution into different multi-modal and discontinuous distributions

A.7 Property-targeted Molecular Optimization

The task of molecular optimization is to search for molecules that have better chemical properties. We choose the standard quantitative estimate of drug-likeness (QED) as our target chemical property. QED measures the potential of a molecule to be characterized as a drug. We used a pre-trained ModFlow model f to encode molecules $\mathcal{M}$ into their embeddings $\mathcal{Z} = f(\mathcal{M})$, and applied linear regression to obtain QED scores $\mathcal{Y}$ from these embeddings. We then interpolate in the latent space of each molecule along the direction of increasing QED via several gradient ascent steps, i.e., updates of the form $\mathcal{Z}' = \mathcal{Z} + \lambda * \frac{d\mathcal{Y}}{d\mathcal{Z}}$, where λ denotes the length of the search step. The final embedding thus obtained is decoded as a new molecule via the reverse mapping f^{-1}.

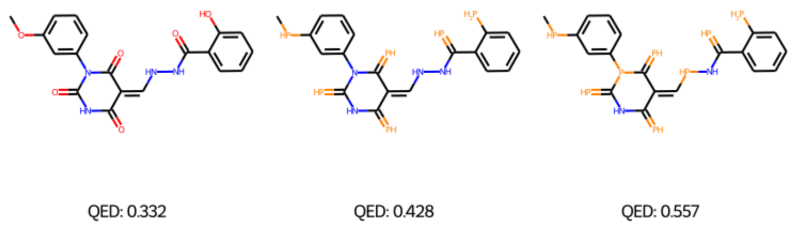


Figure 5: Example of chemical property optimization on the ZINC250K dataset. Given the left-most molecule, we interpolate in latent space along the direction which maximizes its QED property.

Figure 5 and Figure 6 show examples of the molecules decoded from the learned latent space using this procedure, starting with molecules having a low QED score. Note that the number of valid molecules decoded back varies on the query molecule. We report the discovered novel molecules sorted by their QED scores in Table 4. Clearly, `ModFlow` is able to find novel molecules with high QED scores.

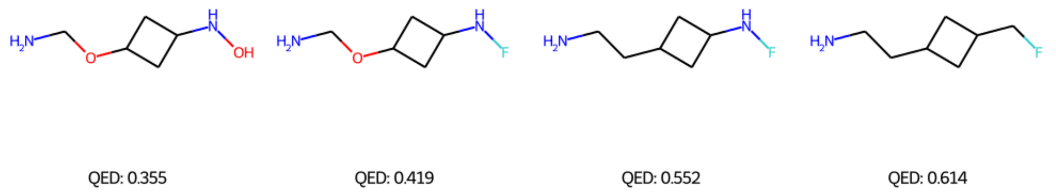


Figure 6: Example of chemical property optimization on the QM9 dataset. Given the left-most molecule, we interpolate in latent space along the direction which maximizes its QED property.

Table 4: Performance in terms of the best QED scores (baselines are taken from Luo et al. (2021)).

Method	1st	2nd	3rd
ZINC (dataset)	0.948	0.948	0.948
JTVAE	0.925	0.911	0.910
GCPN	0.948	0.947	0.945
MRNN	0.844	0.799	0.736
GraphAF	0.948	0.948	0.947
GraphDF	0.948	0.948	0.948
MoFlow	0.948	0.948	0.948
`ModFlow` (2D-EGNN)	0.948	0.941	0.937
`ModFlow` (3D-EGNN)	0.948	0.937	0.931
`ModFlow` (JT-2D-EGNN)	0.947	0.941	0.939
`ModFlow` (JT-3D-EGNN)	0.948	0.948	0.945