Barking up the right tree: an approach to search over molecule synthesis DAGs

Anonymous Authors[1]

Abstract

When suggesting new molecules with particular properties to a chemist, it is not only important what to make but crucially *how to make it*. These instructions form a synthesis directed acyclic graph (DAG), describing how a large vocabulary of simple building blocks can be recursively combined through chemical reactions to create more complicated molecules of interest. In contrast, many current deep generative models for molecules ignore synthesizability. We therefore propose a deep generative model that better represents the real world process, by directly outputting molecule synthesis DAGs. We argue that this provides sensible inductive biases, ensuring that our model searches over the same chemical space that a chemist would. We show that our approach models chemical space well, producing a wide range of diverse molecules, and allows for unconstrained optimization of an inherently constrained problem: maximize certain properties such that discovered molecules are synthesizable.

1. Introduction

Designing new molecules is a key step for problems such as medicine development. To address this, there have been many recent exciting developments in ML towards two goals: G1. **Learning generative models of molecules**: that can be used to sample novel molecules, for downstream screening and scoring, and; G2. **Molecular optimization**: how to search for molecules that maximize certain properties (e.g., drug-likeness) (Gómez-Bombarelli et al., 2018; You et al., 2018; Jin et al., 2018; Li et al., 2018; Olivecrona et al., 2017; Segler et al., 2017b; Kadurin et al., 2017; Assouel et al., 2018; Dai et al., 2018; Samanta et al., 2019). However, for most ML approaches there is often no indication that proposed molecules can *be made*.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review at BRIDGE BETWEEN PERCEPTION AND REASONING: GRAPH NEURAL NETWORKS & BEYOND (ICML 2020 Workshop). Do not distribute.

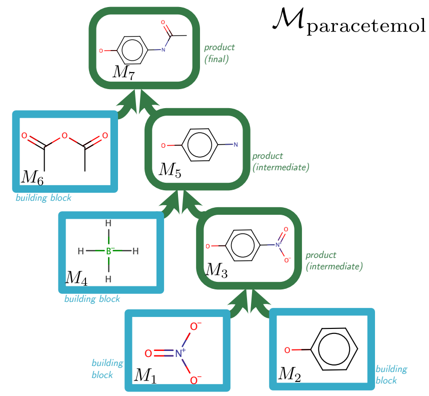

Figure 1. A synthesis DAG for paracetamol (Ellis, 2002).

Recently, approaches to address this have, (a) focused on single-step reactions (Bradshaw et al., 2019), or (b) performed a random walk on a reaction network, deciding which points to assess the properties of using Bayesian optimization (Korovina et al., 2019). The downside of (a) is that most molecules cannot be synthesized in a single step from a fixed set of common reactants (e.g., paracetamol – see Figure 1). Whereas, the downside of (b) is that the walk proceeds in an undirected manner through synthesis space.

In this work to address this gap, we present a new architecture to generate *multi-step molecular synthesis routes*. We represent routes as directed acyclic graphs (DAGs) of graphs (DoGs), and develop encoder and decoder networks around this structure, These can be integrated into widely used frameworks such as latent generative models (G1) (Kingma & Welling, 2013; Tolstikhin et al., 2017), which allow sampling and interpolation within molecular space, or reinforcement learning-based optimization procedures (G2) to optimize molecules for particular tasks. Compared with models not constrained to also generate synthetically tractable molecules, competitive results are obtained.

2. Modeling Synthesis DAGs

In this section we describe how synthesis routes can be defined as DAGs, and our generative model over this structure. We then show how our model can be used as part of a larger framework, such as an autoencoder.

2.1. Synthesis Pathways as DAGs

Consider Figure 1. At a high level, to synthesize a new molecule M_T, such as paracetamol, one needs to per-

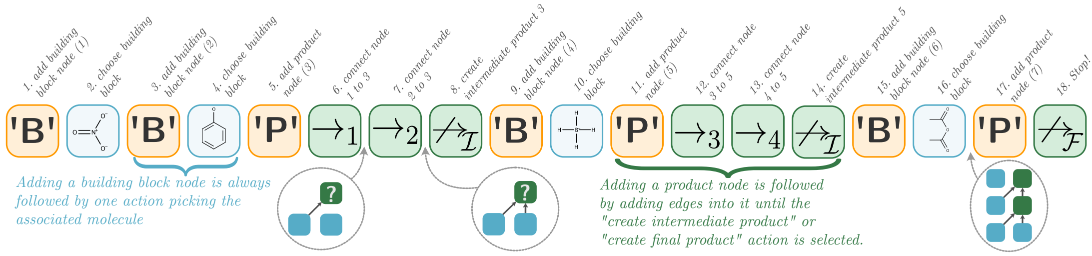

Figure 2. An example of how we can *serialize* the construction of the DAG shown in Figure 1, with the corresponding DAG for three different time-points shown in the gray circles. The serialized construction sequence consists of a sequence of actions, these can be classified into belonging to three different types: (A1) node addition, (A2) building block molecular identity, and (A3) connectivity choice. By convention we start at the furthest building block from the final product node.

form a series of reactions. Each reaction takes a set of molecules (reactants) and physically combines to produce a new molecule (a product), where we make the assumption here that all reactions are deterministic and produce a single primary product. The set of reactants are selected from a pool of available molecules, which includes a large set of easy-to-obtain starting molecules (building blocks), $\mathcal{R}$, and existing intermediate products already created.

In general, this multi-step reaction pathway forms a synthesis DAG, which we shall denote $\mathcal{M}$. Specifically, note that it is directed from reactants to products, each unique molecule maps one-to-one with each node, and it is not cyclic, as we need not consider reactions that produce existing molecules.

2.2. A probabilistic generative model of synthesis DAGs

We need a way to serialize the construction of a DAG such that a ML model can iteratively construct it. Figure 2 shows such an approach. Specifically, we divide actions into three types: A1. *Node-addition (shown in yellow)*: What type of node (building block or product) should be added to the graph?; A2. *Building block molecular identity (in blue)*: Once a building block node is added, what molecule should this node represent?; A3. *Connectivity choice (in green)*: What reactant nodes should be connected to a product node? (i.e., what molecules should be reacted together).

As shown in Figure 2 the construction of a DAG, $\mathcal{M}$, then happens through a sequence of these actions, which we shall denote as $\mathcal{M} = [V^1, V^2, V^3, \ldots, V^L]$. Building block ('B') or product nodes ('P') are selected through action type A1, before the identity of the molecule they contain is specified. For building blocks this consists of choosing the relevant molecule in $\mathcal{R}$, through an action of type A2. Product nodes' molecular identity is instead defined by the reactants that produce them, therefore action type A3 is used repeatedly to either select an incoming reactant edge to an existing molecule in the DAG, or to decide to form an intermediate ($\not\to_{\mathcal{I}}$) or final ($\not\to_{\mathcal{F}}$) product. In forming a final product all the previous nodes without successors are connected up to the final product node, and the sequence is complete.

Defining a probabilistic distribution over actions We propose an auto-regressive factorization over the actions:

$$p_\theta(\mathcal{M}|\boldsymbol{z}) = \prod_{l=1}^{L} p_\theta(V_l|V_{<l}, \boldsymbol{z}) \tag{1}$$

Each $p_\theta(V_l|V_{<l}, \boldsymbol{z})$ is parameterized by a neural network (NN), with weights θ. The structure of this network is shown in Figure 3. It consists of a shared RNN that, given an embedding of the previous action chosen, computes a 'context' vector, which gets fed into a feed forward action-network (specific to the action-type) for predicting each action. The RNN's hidden state is initialized by a latent variable $\boldsymbol{z} \in \mathbb{R}^d$, the setting of which is discussed in §2.3.[1]

Action embeddings We represent actions in our NNs using continuous embeddings. For abstract actions, such as producing a new node ('B', 'P'), these are learnt, and for molecular actions provided by Gated Graph Neural Networks (Li et al., 2016) (GGNNs) run on the molecular graph.

Reaction prediction At test time, intermediate or final products may be created using reactions not contained in our training set. Here we use the Molecular Transformer (Schwaller et al., 2019) as a reaction predictor, $\mathsf{Product}(\cdot)$, that given a set of reactants predicts the major product (or randomly selects a reactant if the reaction does not work).

2.3. Variants of our model

Having introduced our general model for generating synthesis DAGs of (molecular) graphs (DoGs), we detail two variants: an autoencoder (DoG-AE) for learning continuous embeddings (G1), and a more basic generator (DoG-Gen) for performing molecular optimization via finetuning (G2).

DoG-AE: Learning a latent space over synthesis DAGs For G1, we are interested in learning latent continuous embeddings of our space, which allows the exploration of latent space through sampling and interpolation. To do this we will use our generative model as a *decoder*, in an autoencoder structure, DoG-AE. We specify a Gaussian prior over our latent variable $\boldsymbol{z}$, where each $\boldsymbol{z} \sim p(\boldsymbol{z})$ can be thought

[1]Further model details can also be found in the Appendix.

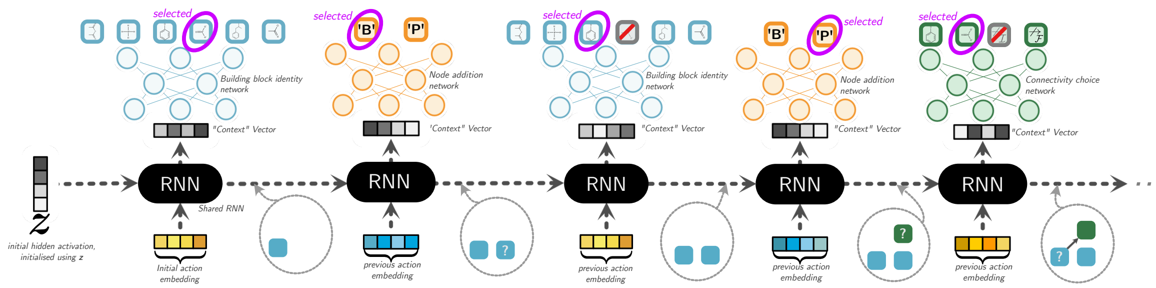

Figure 3. A depiction of how we use neural networks to parameterize the probability of picking actions at stages 1-6 of Figure 2 (note that as stage 1 always suggests a building block node it is automatically completed). A shared RNN for the different action networks receives an embedding of the previous action chosen and creates a context vector for the action network. When using our approach as part of an autoencoder network then the initial hidden layer is parameterized by the latent space sample, $\boldsymbol{z}$. Each type of action network chooses a subsequent action to take (impossible actions are masked out, such as selecting an already existing building block or creating an intermediate product before selecting at least one reactant). The process continues until the create final product node is selected.

of as describing different types of synthesis pathways. We can learn the parameters of our model by optimizing for the Wasserstein autoencoder (WAE) objective with a negative log likelihood cost function (Tolstikhin et al., 2017),

$$\min_{\phi,\theta} \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}\mathbb{E}_{q_\phi(\mathbf{z}\mid\mathcal{M})}\Big[-\log p_\theta(\mathcal{M}\mid\mathbf{z})\Big] + \lambda\mathcal{D}(q_\phi(\mathbf{z}), p(\mathbf{z})),$$

where following Tolstikhin et al. (2017) $\mathcal{D}(\cdot,\cdot)$ is a maximum mean discrepancy divergence measure and $\lambda = 10$.

This leaves us to define our *encoder*, $q_\phi(\mathbf{z}\mid\mathcal{M})$ a stochastic mapping from synthesis DAGs to latent space. Our encoder consists of a two-step hierarchical message passing procedure. Initial DAG node embeddings are calculated using a primary GGNN on the molecular graph associated with each node. These DAG node embeddings are then updated using a secondary GGNN, which passes messages forward on the DAG. Lastly, the updated final product node embeddings parameterize a distribution over latent space.

2.4. DoG-Gen: Molecular optimization via fine-tuning

For molecular optimization, we consider a model trained without a latent space; we use our probabilistic generator of synthesis DAGs and fix $\mathbf{z}=\mathbf{0}$, we call this model DoG-Gen. We then adopt the hill-climbing algorithm from Brown et al. (2019). For this, our model is pre-trained via maximum likelihood to match the training distribution $p(\mathcal{M})$. For optimization, we can then fine-tune the weights θ of the decoder: this is done by sampling a large number of candidate DAGs from the model, ranking them according to a target, and finetuning our model's weights on the top K samples.

3. Experiments

We now evaluate our approach to generating synthesis DAGs on the two goals set out earlier: (G1) can we model the space of synthesis DAGs well, and (G2) can we find optimized molecules for particular properties. To train our models, we create a dataset of synthesis DAGs based on the USPTO reaction dataset (Lowe, 2012) (see appendix for details).

3.1. Generative Modeling of Synthesis DAGs

We begin by assessing properties of the final molecules produced by our models (Table 1). Ignoring the synthesis allows us to compare against previous generative models for molecules including SMILES-LSTM (Segler et al., 2017b), the Character VAE (CVAE) (Gómez-Bombarelli et al., 2018), the Grammar VAE (GVAE) (Kusner et al., 2017), the GraphVAE (Simonovsky & Komodakis, 2018), the Junction Tree Autoencoder (JT-VAE) (Jin et al., 2018), the Constrained Graph Autoencoder (CGVAE) (Liu et al., 2018), and Molecule Chef (Bradshaw et al., 2019).[2] These models cover a wide range of approaches for modeling molecular graphs, however aside from Molecule Chef, which is itself limited to one step reactions, these other baselines do not provide synthetic routes with their output.

As metrics we report those used previously (Liu et al., 2018). Specifically, validity measures how many of the generated molecules can be parsed by the chemoinformatics software RDKit (RDKit, online). Conditioned on validity, we consider the proportions of molecules that are unique and novel (different to those in the training set). These metrics are useful sanity checks, albeit with limitations (Brown et al., 2019), showing that sensible molecules are produced.

3.2. Optimizing Synthesizable Molecules

We next look at using our model for the optimization of molecules with desirable properties. To evaluate our model, we compare its performance on a series of 10 optimization

[2] We reimplemented the CVAE and GVAE models in PyTorch and found that our implementation is significantly better than Kusner et al. (2017)'s published results. We believe this is down to being able to take advantage of some of the latest techniques for training these models (for example β-annealing(Higgins et al., 2017; Alemi et al., 2018)) as well as hyperparameter tuning.

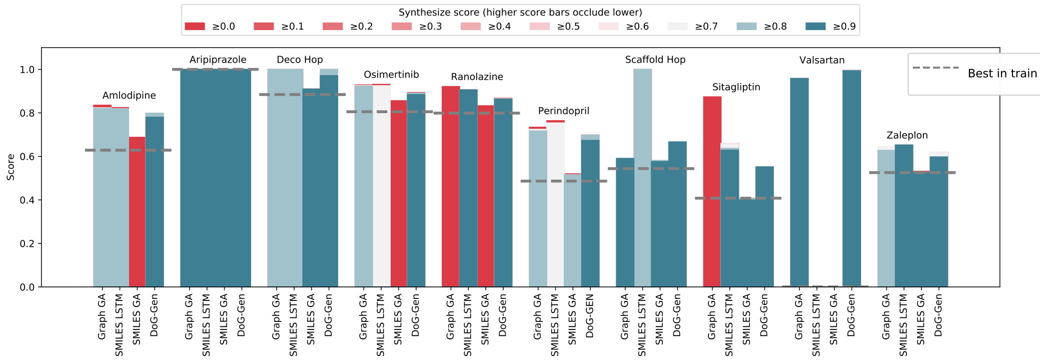

Figure 4. The score of the best molecule found by the different approaches over a series of ten Guacamol tasks (Brown et al., 2019, §3.2). Benchmark scores (y-axis) range between 0 and 1, with 1 being the best. We also differentiate between the synthesizability of the different best molecules found by using colors to indicate the synthesizability score (higher better) of the best molecule found. Note that bars representing a molecule within a higher synthesizability score bucket (eg blue) will occlude lower synthesizability score bars (eg red). The dotted gray lines represent the scores of the best molecule in our training set.

Table 1. Table showing the percentage of valid molecules generated and then conditioned on this the uniqueness and novelty (within the sample). For each model we generate the molecules by decoding from 20k prior samples from the latent space.

Model Name	Validity (↑)	Uniqueness (↑)	Novelty (↑)
DoG-AE	100.0	98.3	92.9
DoG-Gen	100.0	97.7	88.4
Training Data	100.0	100.0	0.0
SMILES LSTM	94.8	95.5	74.9
CVAE	96.2	97.6	76.9
GVAE	74.4	97.8	82.7
GraphVAE	42.2	57.7	96.1
JT-VAE	100.0	99.2	94.9
CGVAE	100.0	97.8	97.9
Molecule Chef	98.9	96.7	90.0

tasks from Brown et al. (2019, §3.2) against the three best reported models Brown et al. (2019, Table 2) found: (1) SMILES LSTM (Segler et al., 2017b), which does optimization via fine-tuning; (2) GraphGA (Jensen, 2019), a graph genetic algorithm (GA); and (3) SMILES GA (Yoshikawa et al., 2018), a SMILES based GA. We train all methods on the same data, which derived from the USPTO dataset, should give a strong bias for synthesizability.

We note that we should not expect our model to find the best molecule if judged solely on molecular property score; our model has to build up molecules from reactions, which although better reflecting reality, means that it is more constrained. However, the final property score is not everything, molecules also need to: (i) be sufficiently stable, and (ii) be able to actually be created in practice (synthesizable). To quantify (i) we use the quality filters proposed in Brown et al. (2019, §3.3). To quantify (ii) we use Computer-Aided Synthesis Planning (Boda et al., 2007; Segler et al., 2017a; 2018; Gao & Coley, 2020). Specifically, we run a retrosynthesis tool (Segler et al., 2018) on each molecule to see if a synthetic route can be found, and how many steps are involved. We also measure an aggregated synthesizability score over each step (see Appendix). All results are calculated on the top 100 molecules found by each method for each task.

The results are shown in Figure 4 and Table 2 (see also appendix). Figure 4 shows when disregarding synthesis, that generally Graph GA and SMILES LSTM produce molecules with the best property scores. However, corroborating with (Gao & Coley, 2020, FigureS6), we find GA methods regularly produce unsynthesizable molecules. Our model, consistently finds high-scoring molecules while maintaining high synthesis scores. Furthermore, a high fraction of our model's molecules pass the quality checks (Table 2).

Table 2. Using the top 100 molecules suggested by each method aggregated over all tasks: the fraction for which a synthetic route is found, the mean synthesizability score , and the fraction that pass the quality filters from Brown et al. (2019, §3.3).

	Frac. Synthesizable	Avg. Synth. Score	Quality
DoG-Gen	0.9	0.76	0.75
Graph GA	0.42	0.33	0.36
SMILES LSTM	0.48	0.39	0.49
SMILES GA	0.29	0.25	0.39

4. Conclusions

In this work, we introduced a novel neural architecture component for molecule design, which by directly generating synthesis DAGs, captures how molecules are made in the lab. We showcase how the component can be used in different paradigms, such as WAEs and RL, demonstrating competitive performance on various benchmarks.

References

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. Fixing a broken ELBO. In *International Conference on Machine Learning*, pp. 159–168, 2018.

Assouel, R., Ahmed, M., Segler, M. H., Saffari, A., and Bengio, Y. Defactor: Differentiable edge factorization-based probabilistic graph generation. 2018. arXiv:1811.09766.

Boda, K., Seidel, T., and Gasteiger, J. Structure and reaction based evaluation of synthetic accessibility. *Journal of computer-aided molecular design*, 21(6):311–325, 2007.

Bradshaw, J., Paige, B., Kusner, M. J., Segler, M. H., and Hernández-Lobato, J. M. A model to search for synthesizable molecules. In *NeurIPS*, 2019.

Brown, N., Fiscato, M., Segler, M. H., and Vaucher, A. C. Guacamol: benchmarking models for de novo molecular design. *Journal of chemical information and modeling*, 59(3):1096–1108, 2019.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. Syntax-directed variational autoencoder for structured data. In *International Conference on Learning Representations*, 2018.

Ellis, F. *Paracetamol: a curriculum resource*. Royal Society of Chemistry, 2002.

Gao, W. and Coley, C. W. The synthesizability of molecules proposed by generative models. *Journal of Chemical Information and Modeling*, 2020.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a Data-Driven continuous representation of molecules. *ACS Cent Sci*, 4(2):268–276, February 2018.

Grzybowski, B. A., Bishop, K. J. M., Kowalczyk, B., and Wilmer, C. E. The 'wired' universe of organic chemistry. *Nat. Chem.*, 1(1):31–36, April 2009.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

Jacob, P.-M. and Lapkin, A. Statistics of the network of organic chemistry. *React. Chem. Eng.*, 3(1):102–118, February 2018.

Jensen, J. H. A graph-based genetic algorithm and generative model/monte carlo tree search for the exploration of chemical space. *Chemical science*, 10(12):3567–3572, 2019.

Jin, W., Coley, C. W., Barzilay, R., and Jaakkola, T. Predicting organic reaction outcomes with Weisfeiler-Lehman network. In *Advances in Neural Information Processing Systems*, 2017.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, 2018.

Kadurin, A., Aliper, A., Kazennov, A., Mamoshina, P., Vanhaelen, Q., Khrabrov, K., and Zhavoronkov, A. The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883, 2017.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Korovina, K., Xu, S., Kandasamy, K., Neiswanger, W., Poczos, B., Schneider, J., and Xing, E. P. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. *arXiv preprint arXiv:1908.01425*, 2019.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *International Conference on Machine Learning*, 2017.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *International Conference on Learning Representations*, 2016.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, March 2018.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. L. Constrained graph variational autoencoders for molecule design. In *Advances in neural information processing systems*, 2018.

Lowe, D. M. *Extraction of chemical structures and reactions from the literature*. PhD thesis, University of Cambridge, 2012.

Olivecrona, M., Blaschke, T., Engkvist, O., and Chen, H. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

RDKit, online. RDKit: Open-source cheminformatics. http://www.rdkit.org. [Online; accessed 01-February-2018].

Samanta, B., Abir, D., Jana, G., Chattaraj, P. K., Ganguly, N., and Rodriguez, M. G. Nevae: A deep generative model for molecular graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1110–1117, 2019.

Schneider, N., Lowe, D. M., Sayle, R. A., and Landrum, G. A. Development of a novel fingerprint for chemical reactions and its application to large-scale reaction classification and similarity. *J. Chem. Inf. Mod.*, 55(1):39–53, 2015.

Schwaller, P., Gaudin, T., Lanyi, D., Bekas, C., and Laino, T. "found in translation": predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. *Chemical science*, 9(28):6091–6098, 2018.

Schwaller, P., Laino, T., Gaudin, T., Bolgar, P., Hunter, C. A., Bekas, C., and Lee, A. A. Molecular transformer: A model for uncertainty-calibrated chemical reaction prediction. *ACS Central Science*, 5(9):1572–1583, 2019. doi: 10.1021/acscentsci.9b00576.

Segler, M., Preuß, M., and Waller, M. P. Towards alphachem: Chemical synthesis planning with tree search and deep neural network policies. *arXiv preprint arXiv:1702.00020*, 2017a.

Segler, M. H., Kogej, T., Tyrchan, C., and Waller, M. P. Generating focused molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017b.

Segler, M. H., Preuss, M., and Waller, M. P. Planning chemical syntheses with deep neural networks and symbolic ai. *Nature*, 555(7698):604–610, 2018.

Simonovsky, M. and Komodakis, N. GraphVAE: Towards generation of small graphs using variational autoencoders. In Kůrková, V., Manolopoulos, Y., Hammer, B., Iliadis, L., and Maglogiannis, I. (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2018*, pp. 412–422, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01418-6.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Yoshikawa, N., Terayama, K., Sumita, M., Homma, T., Oono, K., and Tsuda, K. Population-based de novo molecule generation, using grammatical evolution. *Chemistry Letters*, 47(11):1431–1434, 2018.

You, J., Liu, B., Ying, R., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems*, 2018.

A. Appendix

Our appendix contains the following sections:

Section A.1 Provides further experimental results, for instance we show examples of the synthesis DAGs decoded whilst interpolating in the latent space of DoG-AE in Figure 6.

Section A.2 Provides further details about our model, for instance it includes an algorithm of our generative process (Alg. 1).

Section A.3 Provides further details about our experiments, such as details on how we create a dataset of synthesis DAGs, the definition of the synthesis score used in the main paper, and implementation details of the models and baselines we use.

A.1. Further experimental results

Interpolation in DoG-AE's latent space The advantage of our model over others is that it directly generates synthesis DAGs, indicating how a generated molecule could be made. To visualize the latent space of DAGs we start from a training synthesis DAG and walk randomly in latent space until we have output five different synthesis DAGs. We plot the combination of these DAGs, which can be seen as a reaction network, in Figure 5. We see that as we move around latent space many of the synthesis DAGs have subgraphs that are isomorphic, resulting in similar final molecules.

Further results for the Guacamol optimization tasks Figure 6 shows the fraction of the top 100 molecules proposed by each method for each task for which a synthetic route can be found. Figure 7 shows the average synthesis score over the 100 best molecules proposed by each method for each task.

A.2. Further details about our model

In this section we provide further details of our model. Our explanation is further broken down into three subsections. In the first we provide more details on our generative model for synthesis DAGs, including pseudocode for the full generative process. In the second subsection we provide further details on how we use the Molecular Transformer for reaction prediction to fill in the products of reactions at test time. In the third and final subsection we provide further information on the finetuning setup. The description of the hyperparameters and specific architectures used in our models are given in the next section.

A.2.1. A GENERATIVE MODEL OF SYNTHESIS DAGS

In this subsection we provide a more thorough description of our generative model for synthesis DAGs. We first recap and expand upon the notation that we use in the main paper. We formally represent the DAG, $\mathcal{M}$, as a sequence of actions, with $\mathcal{M} = [V^1, \dots, V^L]$. Alongside this we denote the associated action types as $\boldsymbol{A} = [A^1, \dots, A^L]$. The action type entries A^l take values in $\{\text{A1}, \text{A2}, \text{A3}\}$, corresponding to the three action types. The action type entry at a particular step, A^l, is fully defined by the actions (and action types) chosen previously to this time l, the exact details of which we shall come back to later. Finally the set of molecules existing in the DAG at time l are denoted (in an abuse of our notation) by $\mathcal{M}_{<l}$.

Actions and the values that they can take We now describe the potential values that the actions can take. These depend on the action type at the step, and we denote this conditioning as $V^l_{|A^l}$. For example for node addition actions $A^l = \text{A1}$, the possible values of V^l (ie $V^l_{|A_l=\text{A1}}$) are either 'B' for creating a new building block node, or 'P' for a new product node. Building-block actions $A^l = \text{A2}$ have corresponding values $V^l \in \mathcal{R}$, which determine which building block becomes a new 'leaf' node in the DAG. Connectivity choice actions $A^l = \text{A3}$ have values $V^l \in \mathcal{M}_{<l} \cup \{\not\rightarrow_{\mathcal{I}}, \not\rightarrow_{\mathcal{F}}\}$, where $\mathcal{M}_{<l}$ denotes the current set of all molecules present in the DAG; selecting one of these molecules adds an edge into the new product node. The symbol $\not\rightarrow_{\mathcal{I}}$ is an intermediate product stop symbol, indicating that the new product node has been connected to all its reactants (ie an intermediate product has been formed); the symbol $\not\rightarrow_{\mathcal{F}}$ is a final stop symbol, which triggers production of the final product and the completion of the generative process.

As hinted at earlier, the action type, A^l is defined by the previous actions $V^1, \dots, V^{l-1}$ and action types (see also Figure 2 in the main paper). More specifically, this happens as follows:

$V^{l-1} =$ 'B', then the next action type is building block selection, $A^l = \text{A2}$.

$A^{l-1} = \text{A2}$, then the next action type is again node addition, $A^l = \text{A1}$ (as you will have selected a building block on the previous step).

$V^{l-1} =$ 'P', then the next action type is connectivity choice, $A^l = \text{A3}$, to work out what to connect up to the product node previously selected.

$A^{l-1} = \text{A3}$ then:

- if $V^{l-1} = \not\rightarrow_{\mathcal{I}}$ then the next action type is to choose a new node again, ie $A^l = \text{A1}$;
- if $V^{l-1} = \not\rightarrow_{\mathcal{F}}$ the generation is finished;
- if $V^{l-1} \in \mathcal{M}_{<l}$ then connectivity choice continues, ie $A^l = \text{A3}$.

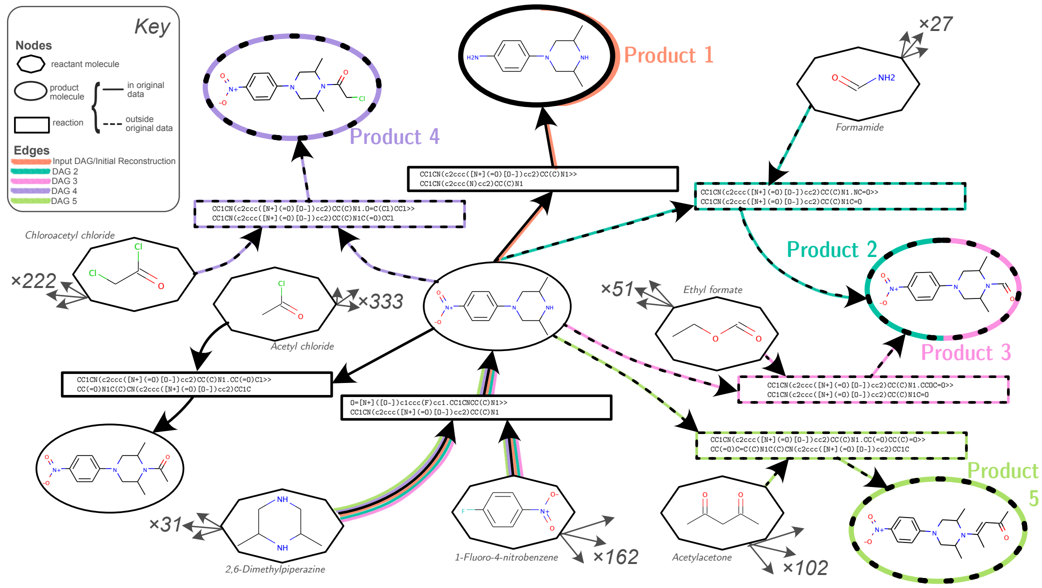

Figure 5. We randomly walk in the latent space of a DoG-AE model and we decode out to similar DAGs nearby, unseen in training. Reactions and nodes that exist in our training dataset are outlined in solid lines, whereas those that have been discovered by our model are shown in dashed line.

Our generative process over these actions Our model is shown at a high level in Figure 8 (see also Figure 3 of the main paper), which serves to provide an intuitive understanding of the generative process. The overall structure of the probabilistic model is rather complex, as it depends on a series of branching conditions: we therefore give pseudocode for the entire generative procedure in detail as a probabilistic program in Algorithm 1. The program described in Alg. 1 defines a distribution over DAG serializations; running it forward will sample from the generative process, but it can equally well be used to evaluate the probability of a DAG $\mathcal{M}$ of interest by instead accumulating the log probability of the sequence at each distribution encountered during the execution of the program. Note that given our assumption that all reactions are deterministic and produce a single primary product, product molecules do not appear in our decomposition.

A.2.2. Reaction prediction

As described in the main paper we use the Molecular Transformer (Schwaller et al., 2019) for reaction prediction. We use pre-trained weights (trained on a processed USPTO (Jin et al., 2017; Lowe, 2012) dataset without reagents). Furthermore, we treat the transformer as a black box oracle and so make no further adjustments to these weights when training our model. We take the top one prediction from the transformer as the prediction for the product, and if this is not a valid molecule (determined by RDkit) then we instead pick one of the reactants randomly.

When running our model at prediction time there is the possibility of getting loops (and so no longer predicting a DAG) if the output of a reaction (either intermediate or final) creates a molecule which already exists (in the DAG) as a predecessor of one of the reactants. A principled approach one could use to deal with this when using a probabilistic reaction predictor model, such as the Molecular Transformer, is to mask out the prediction of reactions that cause loops in the reaction predictor's beam search. However, in our experiments we want to keep the reaction predictor as a black box oracle for which we send reactants and for which it sends us back a product. Therefore, to deal with any prediction-time loops we go back through the DAG, before and after predicting the final product node, and remove any loops we have created by choosing the first path that was predicted to each node.

A.2.3. Finetuning

The algorithm we use for finetuning is given in Algorithm 2.

A.3. Further experimental details

This section provides further details about aspects of our experiments. We start by describing how we create a dataset of synthesis DAGs for training. We then describe how the synthesis score we use in the optimization experiments is calculated. Finally, the latter subsections provide specific details on the hyperparameters we use.

Algorithm 1 Probabilistic simulator for serialized DAGs

Require: Action networks for node addition, building block molecular identity, and connectivity choice: $\mathsf{na}(\cdot)$, $\mathsf{bbmi}(\cdot)$, $\mathsf{cc}(\cdot)$;
Require: Reaction predictor: $\mathsf{Product}(\cdot)$;
Require: Context RNN: $\mathbf{c}^l = r(\mathbf{c}^{l-1}, \boldsymbol{e}^l)$;
Require: Continuous latent variable: $\boldsymbol{z}$
Require: Linear projection for mapping continuous latent to RNN initial hidden: $\mathsf{Lin}(\cdot)$.
Require: Gated graph neural network, $\mathsf{GGNN}(\cdot)$, for computing molecule embeddings
Require: Learnable embeddings for abstract actions: $\boldsymbol{h}_{\text{'B'}}$, $\boldsymbol{h}_{\text{'P'}}$, $\boldsymbol{h}_{\not\to\mathcal{I}}$, and $\boldsymbol{h}_{\not\to\mathcal{F}}$.

Initialize DAG $\mathcal{M} \leftarrow [V^1 = \text{'B'}]$, Initialize $\boldsymbol{A} \leftarrow [A^1 = \mathsf{A1}, A^2 = \mathsf{A2}]$
Initialize molecule set $M \leftarrow \{\}$ and set of unused reactants $U \leftarrow \{\}$ {Track all / all unused molecules}
$\mathbf{c}^1 \leftarrow \mathsf{Lin}(\boldsymbol{z})$ {Z initializes the first hidden state of the recurrent NN}
$\boldsymbol{e}^2 \leftarrow \boldsymbol{h}_{\text{'B'}}$ {Initial input into RNN reflects that new node added on first step.}
while $V^{|\mathcal{M}|} \neq \not\to_{\mathcal{F}}$ **do**
  $l \leftarrow |\mathcal{M}| + 1$
  $\mathbf{c}^l \leftarrow r(\mathbf{c}^{l-1}, \boldsymbol{e}^l)$ {Update context}
  if $A^l = \mathsf{A1}$ **then**
    $\boldsymbol{w} \leftarrow \mathsf{na}(\mathbf{c}^l)$; $\boldsymbol{B} \leftarrow \text{STACK}([\boldsymbol{h}_{\text{'B'}}, \boldsymbol{h}_{\text{'P'}}])$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^T$
    $V^l \sim \text{softmax}(\boldsymbol{w}\boldsymbol{B}^T)$
    if $V^l = \text{'B'}$ **then**
      $A^{l+1} \leftarrow \mathsf{A2}$; $\boldsymbol{e}^{l+1} \leftarrow \boldsymbol{h}_{\text{'B'}}$
    else if $V^l = \text{'P'}$ **then**
      $A^{l+1} \leftarrow \mathsf{A3}$; $\boldsymbol{e}^{l+1} \leftarrow \boldsymbol{h}_{\text{'P'}}$
      Initialize intermediate reactant set $R \leftarrow \{\}$ {Will temporarily store *working* reactants}
      stop_actions $\leftarrow [\boldsymbol{h}_{\not\to\mathcal{F}}]$ {You cannot stop for intermediate product until at least one reactant}
    end if
  else if $A^l = \mathsf{A2}$ **then**
    $\boldsymbol{w} \leftarrow \mathsf{bbmi}(\mathbf{c}^l)$; $\boldsymbol{B} \leftarrow \text{STACK}([\mathsf{GGNN}(g) \text{ for } g \text{ in } \mathcal{R} \setminus M])$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^T$
    $V^l \sim \text{softmax(logits)}$ {Pick building block molecule}
    $A^{l+1} \leftarrow \mathsf{A1}$; $\boldsymbol{e}^{l+1} \leftarrow \mathsf{GGNN}(V^l)$
    $M \leftarrow M \cup \{V^l\}, U \leftarrow U \cup \{V^l\}$
  else if $A^l = \mathsf{A3}$ **then**
    $\boldsymbol{w} \leftarrow \mathsf{cc}(\mathbf{c}^l)$; $\boldsymbol{B} \leftarrow \text{STACK}([\mathsf{GGNN}(g) \text{ for } g \text{ in } M \setminus R] + \text{stop\_actions})$
    logits $\leftarrow \boldsymbol{w}\boldsymbol{B}^T$
    $V^l \sim \text{softmax(logits)}$ {Pick either (i) molecule to connect to, or (ii) to end and create product}
    if $V^l = \not\to_{\mathcal{I}}$ **then**
      $M^{\text{new}} \leftarrow \mathsf{Product}(R)$
      $M \leftarrow M \cup \{M^{\text{new}}\}, U \leftarrow U \cup \{M^{\text{new}}\}$
      $A^{l+1} \leftarrow \mathsf{A1}$; $\boldsymbol{e}^{l+1} \leftarrow \boldsymbol{h}_{\not\to\mathcal{I}}$
    else if $V^l \in M$ **then**
      $R \leftarrow R \cup \{V^l\}$ {Update reactant set}
      $U \leftarrow U \setminus \{V^l\}$ {Remove from pool of "unused" molecules}
      $A^{l+1} \leftarrow \mathsf{A3}$; $\boldsymbol{e}^{l+1} \leftarrow \mathsf{GGNN}(V^l)$
      stop_actions $\leftarrow [\boldsymbol{h}_{\not\to\mathcal{I}}, \boldsymbol{h}_{\not\to\mathcal{F}}]$ {Now you can stop for both final or intermediate product}
    end if
  end if
  Update $\mathcal{M} \leftarrow [V^1, \ldots, V^l]$; $\boldsymbol{A} \leftarrow [A^1, \ldots, A^l]$
end while
Predict final product $M^T \leftarrow \mathsf{Product}(R \cup U)$ {The final product considers both R and U}
return $\mathcal{M}$, M^T

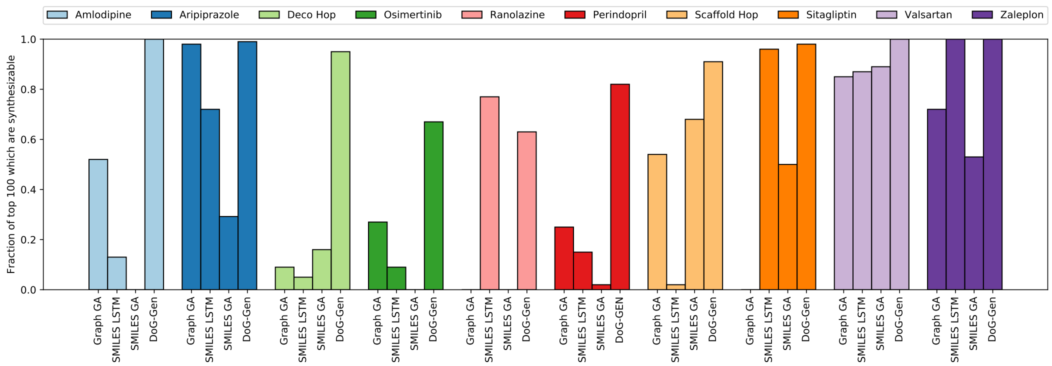

Figure 6. The fraction of the top 100 molecules proposed that for which a synthetic route can be found, over a series of ten Guacamol tasks (Brown et al., 2019, §3.2).

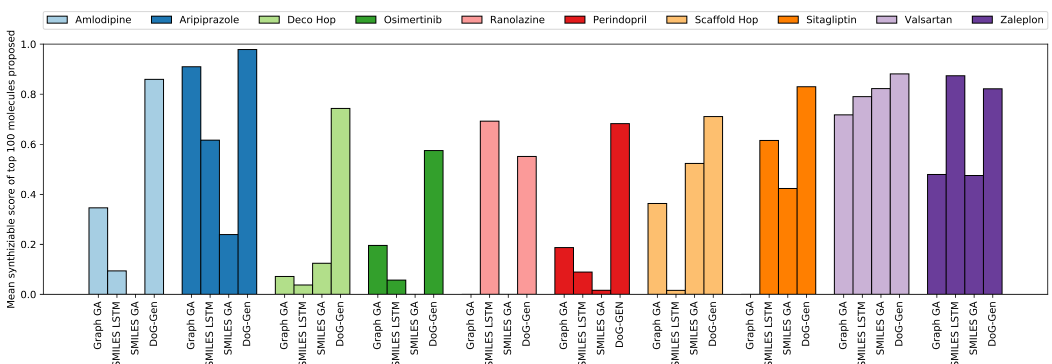

Figure 7. The mean of the synthesis score of the top 100 molecules proposed by each method, over a series of ten Guacamol tasks (Brown et al., 2019, §3.2).

A.3.1. Creating a dataset of synthesis DAGs

In this subsection we describe how we create a dataset of synthesis DAGs, with a high level illustration of the process given in Figure 9. The creation of our synthesis DAG dataset starts by collecting the reactions from the USPTO dataset (Lowe, 2012), using the processed and cleaned version of this dataset provided by (Jin et al., 2017, §4). We filter out reagents (molecules that do not contribute any atoms to the final product) and multiple product reactions (97% of the dataset is already single product reactions) using the approach of Schwaller et al. (2018, §3.1).

This processed reaction data is then used to create a reaction network (Jacob & Lapkin, 2018; Grzybowski et al., 2009). To be more specific, we start from the reactant building blocks specified in Bradshaw et al. (2019, §4) as initial molecule nodes in our network, and then iterate through our list of processed reactions adding any reactions (and the associated product molecules) (i) that depend only molecule nodes that are already in our network, and (ii) where the product is not an initial building block. This process repeats until we can no longer add any of our remaining reactions.

This reaction network is then used to create one synthesis DAG for each molecule. To this end, starting from each possible (non building block) molecule node in our reaction network, we step backwards through the network until we find a sub-graph of the reaction network (without any loops) with initial nodes that are from our collection of building blocks. When there are multiple possible routes we pick one. This leaves us with a dataset of 72008 synthesis DAGs, which we use approximately 90% of as training data and split the remainder into a validation dataset (of 3601

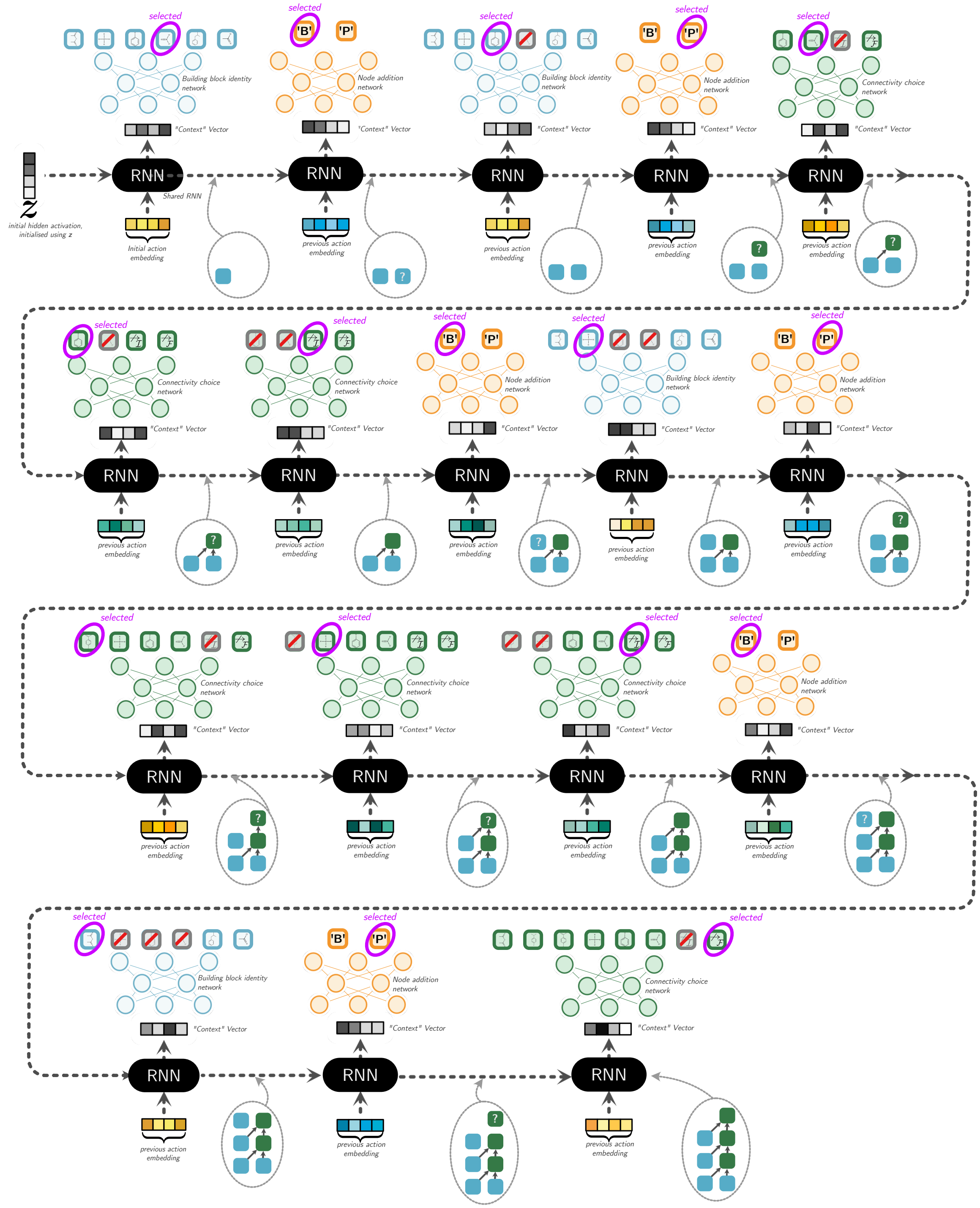

Figure 8. This is an expanded version of Figure 3 in the main paper showing all the actions required to produce the DAG for Paracetemol (see also Figure 1 and 2 in the main paper). A shared RNN (recurrent neural network) provides a context vector for the different action networks. Based on this context vector, each type of action network chooses an action to take (some actions are masked out as they are not allowed, for instance suggesting a building block already in the graph, or choosing to make an intermediate product before choosing at least one reactant). Note embeddings of molecular graphs are computed using a GNN (graph neural network). The initial hidden vector of the shared RNN is initialized using a latent vector z in our autoencoder model DoG-AE; in DoG-Gen it is set to a constant. The state of the DAG at each stage of the generative process is indicated in the dotted gray circles.

Algorithm 2 Synthesis DAG Fine-Tuning. Note that for finetuning we use the model DoG-Gen, in which $\boldsymbol{z}$ is always set at $\mathbf{0}$, hence we drop our specific dependence on $\boldsymbol{z}$ in this algorithm.

Require: Initial model $p_\theta(\mathcal{M})$, iterations I, threshold K, sample size N, objective $h(\cdot)$, pool of seen synthesis DAGs (initially empty), P.

1: Compute the score $h(\cdot)$ of all products in the initial training set and add to P
2: **for** $i = 1, \ldots, I$ **do**
3:     Sample N DAGs from $p_\theta(\mathcal{M})$
4:     Compute the score $h(\cdot)$ of the N products and add to pool, P
5:     Select the K DAGs with the highest score from P
6:     Run two training epochs on θ using these K DAGs as training data
7: **end for**
8: **return** updated distribution $p_\theta(\mathcal{M})$, pool of all seen synthesis DAGs (ranked) P

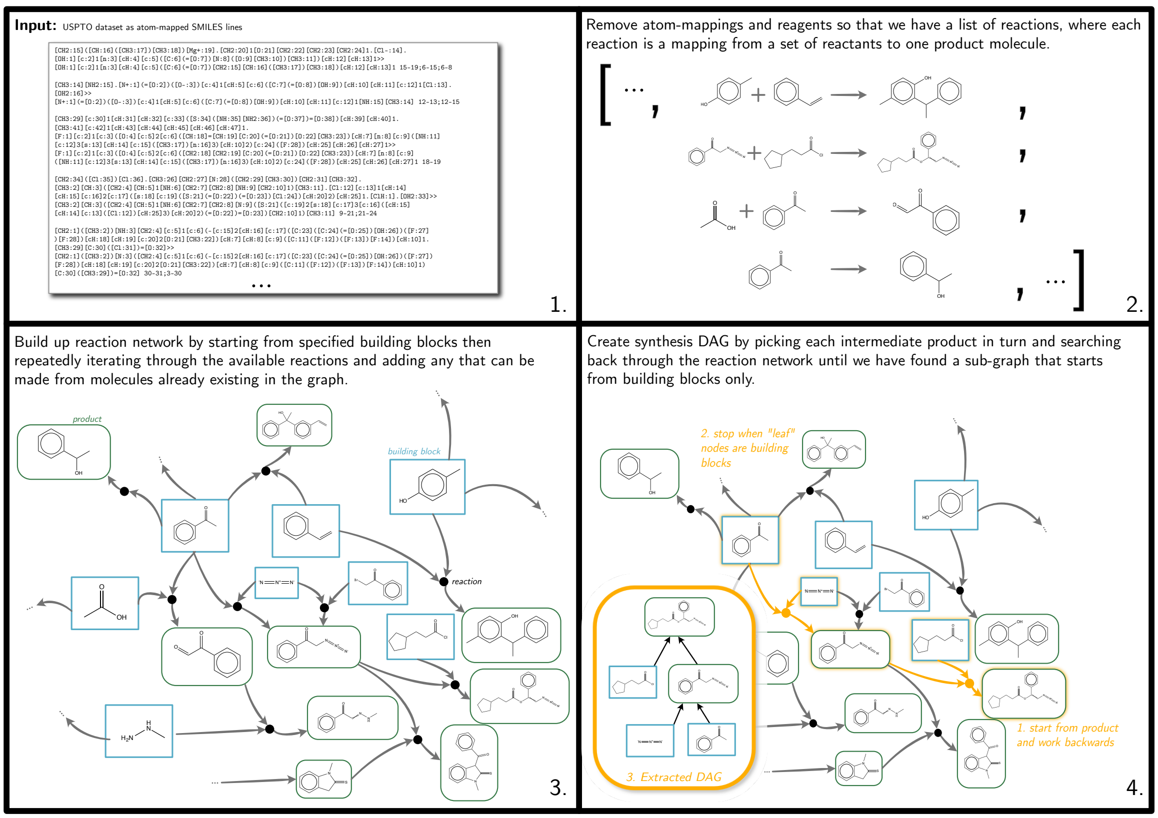

Figure 9. An illustration of how we create a dataset of synthesis DAGs from a dataset of reactions. We first clean up the reaction dataset by removing reagents (molecules which do not contribute atoms to the final product) and any reactions which lead to more than one product. We then form a modified reaction network (we do not allow loops back to building block molecules), which is a directed graph showing how molecules are linked to others through reactions. This process starts by adding molecule nodes corresponding to our initial building blocks. We then repeatedly iterate through our list of reactions and gradually add reaction nodes (and their associated product nodes) to the graph if both (i) the corresponding reaction's reactants are a subset of the molecule nodes already in the graph, and (ii) the product is not a building block. Finally for each possible product node we iterate back through the directed edges until we have selected a subgraph without any loops, where the initial nodes are members of our set of building blocks.

synthesis DAGs) and test dataset (of 3599 synthesis DAGs).

A.3.2. Synthesizability Score

The synthesizability score is defined as the geometric mean of the nearest neighbor reaction similarities:

$$\sqrt[|R|]{\prod_{r \in R} \kappa(r, \mathrm{nn}(r))} \tag{2}$$

where R is the list of reactions making up a synthesis DAG, $r \in R$ are the individual reactions in the DAG, $\mathrm{nn}(r)$ is the nearest neighbor reaction in the chemical literature in Morgan fingerprint space, and $\kappa(\cdot, \cdot)$ is Tanimoto similarity over Morgan reaction fingerprints (Schneider et al., 2015).

A.3.3. Atom features used in DoG models

The atom features we use as input to our graph neural networks (GNNs) operating on molecules are given in Table 3. These features are chosen as they are used in Gilmer et al. (2017, Table 1) (we make the addition of an expanded one-hot atom type feature, to cover the greater range of elements present in our molecules).

Table 3. Atom features we use as input to the GGNN. These are calculated using RDKit.

Feature	Description
Atom type	72 possible elements in total, one hot
Atomic number	integer
Acceptor	boolean (accepts electrons)
Donor	boolean (donates electrons)
Hybridization	One hot (SP, SP2, SP3)
Part of an aromatic ring	boolean
H count	integer

A.3.4. Implementation details for DoG-AE

In this subsection we describe specifics of our DoG-AE model used to produce the results in Table 1 of the main paper.

Forming molecule embeddings For forming molecule embeddings we use a GGNN (Gated Graph Neural Network) (Li et al., 2016); this operates on the atom features described in Table 3. This graph neural network (GNN) was run for 4 propagation steps to update the node embeddings, before these embeddings were projected down to a 50 dimensional space using a learnt linear projection. The node embeddings were then combined to form molecule embeddings through a weighted sum. The same GNN architecture was shared between the encoder and the decoder.

Encoder The encoder (shown in Figure 10) consists of two GGNNs. The first, described above, creates molecule embeddings which are then used to initialize the node embeddings in the synthesis DAG. The synthesis DAG node embeddings, which are 50 dimensional, are further updated using a second GGNN. Seven propagation steps of message passing are carried out on the DAG, where the messages are passed forward on the DAG from the 'leaf' nodes to the final product node. Finally, the node embedding of the final product molecule node in the DAG is passed through an additional linear projection to parameterize the mean and log variance of independent Gaussian distributions over each dimension of the latent variable, $\boldsymbol{z}$.

Decoder For the decoder we use a 3 layer GRU RNN (Cho et al., 2014) to compute the context vector. The hidden layers have a dimension of 200 and whilst training we use a dropout rate of 0.1. For initializing the hidden layers of the RNN we use a linear projection (the parameters of which we learn) of z. The action networks are feedforward neural networks with one hidden layer (dimension 28) and ReLU activation functions. For the abstract actions (such as 'B'or 'P') we learn 50 dimensional embeddings, such that these embeddings have the same dimensionality as the molecule embeddings we compute.

Training We train our model, with a 25 dimensional latent space, using the Adam optimizer (Kingma & Ba, 2015), an initial learning rate of 0.001, and a batch size of 64. We train the autoencoder using the Wasserstein autoencoder loss (Tolstikhin et al., 2017), with $\lambda = 10$ and an inverse multi-quadratics kernel for computing the MMD-based penalty, as this is what is used in Tolstikhin et al. (2017, §4).

Our model, DoG-AE, is trained using teacher forcing for 400 epochs (each epoch took approximately 7 minutes) and we multiplied the learning rate by a factor of 0.1 after 300 and 350 epochs. DoG-AE obtains a reconstruction accuracy (on our held out test set) of 65% when greedily decoding (greedy in the sense of picking the most probable action at each stage of decoding).

A.3.5. Implementation details for DoG-Gen

For DoG-Gen we also used a GGNN to create molecule embeddings in a similar way to DoG-AE. The GGNN was run for 5 rounds of message passing to form 80 dimensional node embeddings; these node embeddings were agglomerated into a 160 dimensional molecule embedding through a linear projection and weighted sum. For generating the context vector we use a 3 layer GRU RNN with 512 dimensional hidden layers. The action networks used were feed-forward neural networks with one hidden layer of dimension 28 and ReLU activation functions. We trained our model for 30 epochs.

For optimization we start by evaluating the score on every synthesis DAG in our training and validation datasets; we then run 30 stages of finetuning, sampling 7000 synthesis DAGs at each stage and updating the weights of our model using the best 1500 DAGs seen at that point as a finetuning dataset.

A.3.6. Details of baselines for generation tasks

We used the following implementations for the baselines:

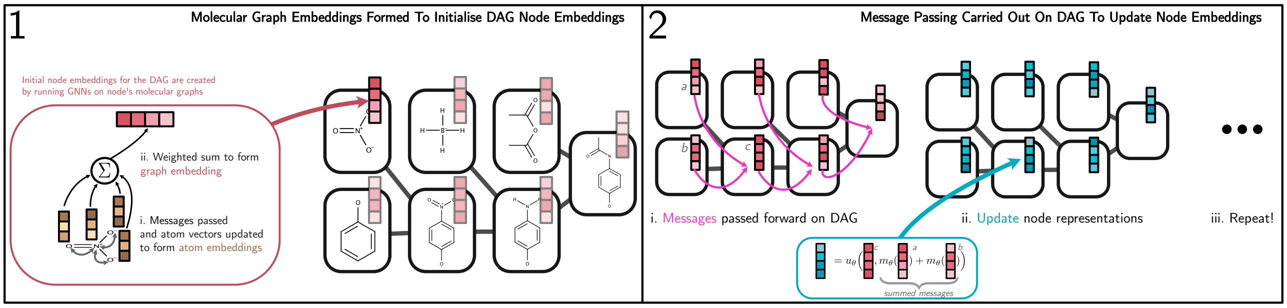

Figure 10. The encoder embeds the DAG of Graphs (DoG) into a continuous latent space. It does this in a two step process. In step 1 it computes initial embeddings for the DAG nodes by forming graph-level embeddings using a GNN on the molecule associated with each node. In step 2 a message-passing algorithm is again used, however, this time on the DAG itself, passing messages forward. The final representation is taken from the node embedding of the final product node.

- SMILES LSTM (Segler et al., 2017b): `https://github.com/benevolentAI/guacamol_baselines`.
- JT-VAE (Jin et al., 2018): `https://github.com/wengong-jin/icml18-jtnn` (we used the updated version of their code, ie the `fast_jtnn` version)
- CGVAE (Liu et al., 2018): `https://github.com/Microsoft/constrained-graph-variational-autoencoder`
- Molecule Chef (Bradshaw et al., 2019): `https://github.com/john-bradshaw/molecule-chef`

For the CVAE, GVAE and GraphVAE baselines we used our own implementations. We tuned the hyperparameters of these models on the ZINC or QM9 datasets so that we were able to get at least similar (and often better) results compared to those originally reported in Kusner et al. (2017); Simonovsky & Komodakis (2018).

When training the GraphVAE on our datasets we exclude any molecules with greater than 20 heavy atoms, as this procedure was found in the original paper to give better performance when training on ZINC (Simonovsky & Komodakis, 2018, §4.3). We use a 40 dimensional latent space, a GGNN (Li et al., 2016) for the encoder, and use max-pooling graph matching during training.

For the CVAE and GVAE we use 72 dimensional latent spaces. We multiply the KL term in the VAE loss by a parameter β (Higgins et al., 2017; Alemi et al., 2018); this β term is then gradually annealed in during training until it reaches a final value of 0.3. We use a 3 layer GRU RNN (Cho et al., 2014) for the decoder with 384 dimensional hidden layers. The encoder is a 3 layer bidirectional GRU RNN also with 384 dimensional hidden layers.