ARTICLES
https://doi.org/10.1038/s42256-021-00410-2
nature machine intelligence

Check for updates

A deep generative model for molecule optimization via one fragment modification

Ziqi Chen[1], Martin Renqiang Min[2], Srinivasan Parthasarathy[1,3] and Xia Ning[1,3,4] ✉

Molecule optimization is a critical step in drug development to improve the desired properties of drug candidates through chemical modification. We have developed a novel deep generative model, Modof, over molecular graphs for molecule optimization. Modof modifies a given molecule through the prediction of a single site of disconnection at the molecule and the removal and/or addition of fragments at that site. A pipeline of multiple, identical Modof models is implemented into Modof-pipe to modify an input molecule at multiple disconnection sites. Here we show that Modof-pipe is able to retain major molecular scaffolds, allow controls over intermediate optimization steps and better constrain molecule similarities. Modof-pipe outperforms the state-of-the-art methods on benchmark datasets. Without molecular similarity constraints, Modof-pipe achieves 81.2% improvement in the octanol–water partition coefficient, penalized by synthetic accessibility and ring size, and 51.2%, 25.6% and 9.2% improvement if the optimized molecules are at least 0.2, 0.4 and 0.6 similar to those before optimization, respectively. Modof-pipe is further enhanced into Modof-pipe^m to allow modification of one molecule to multiple optimized ones. Modof-pipe^m achieves additional performance improvement, at least 17.8% better than Modof-pipe.

Molecule optimization is a critical step in drug discovery for improving the desired properties of drug candidates through chemical modification. For example, in lead optimization[1] (molecules showing both activity and selectivity towards a given target), the chemical structures of the lead molecules can be altered to improve their selectivity and specificity. Conventionally, such a molecule optimization process is planned based on knowledge and experiences from medicinal chemists, and is carried out via fragment-based screening or synthesis[2–5]. Thus, it is not scalable or automated. Recent in silico approaches using deep learning have enabled alternative computationally generative processes to accelerate the conventional paradigm. These deep-learning methods learn from string-based molecule representations (SMILES)[6,7] or molecular graphs[8,9], and generate new ones accordingly (for example, via connecting atoms and bonds) with better properties. Although computationally attractive, these methods do not conform to the in vitro molecule optimization process in one very important aspect: molecule optimization needs to retain the major scaffold of a molecule, but generating entire, new molecular structures may not reproduce the scaffold. Therefore, these methods are limited in their potentials to inform and direct in vitro molecule optimization.

We propose a novel generative model for molecule optimization that better approximates in silico chemical modification. Our method is referred to as 'modifier with one fragment', or Modof. Following the idea of fragment-based drug design[10,11], Modof predicts a single site of disconnection at a molecule and modifies the molecule by changing the fragments (for example, ring systems, linkers and side chains) at that site. Distinctly from existing molecule optimization approaches that encode and decode whole molecular graphs, Modof learns from and encodes the difference between molecules before and after optimization at one disconnection site. To modify a molecule, Modof generates only one fragment that instantiates the expected difference by decoding a sample drawn from the latent 'difference' space. Modof then removes the original fragment at the disconnection site, and attaches the generated fragment at the site. Figure 1 presents an overview of Modof. By sampling multiple times, Modof is able to generate multiple optimized candidates. A pipeline of multiple, identical Modof models, denoted Modof-pipe, is implemented to optimize molecules at multiple disconnection sites through different Modof models iteratively, with the output molecule from one Modof model as the input to the next. Modof-pipe is further enhanced into Modof-pipe^m to allow modification of one molecule into multiple optimized ones as the final output.

Modof has the following advantages:

- It modifies one fragment at a time. It better approximates the in vitro chemical modification and retains the majority of molecular scaffolds. Thus, it potentially better informs and directs in vitro molecule optimization.
- It only encodes and decodes the fragment that needs modification and facilitates better modification performance.
- Modof-pipe modifies multiple fragments at different disconnection sites, iteratively. It enables easier control over and intuitive deciphering of the intermediate modification steps, and facilitates better interpretability of the entire modification process.
- Modof is less complex than the state of the art. It has at least 40% fewer parameters and uses 26% less training data.
- Modof-pipe outperforms the state-of-the-art methods on benchmark datasets in optimizing the octanol–water partition coefficient penalized by synthetic accessibility (SA) and ring size, with 81.2% improvement without molecular similarity constraints on the optimized molecules, and 51.2%, 25.6% and 9.2% improvement if the optimized molecules need to be at least 0.2, 0.4 and 0.6 similar (in Tanimoto coefficient over 2,048-dimensional (2,048D) Morgan fingerprints with radius of 2) to those before optimization, respectively.

[1]Computer Science and Engineering, The Ohio State University, Columbus, OH, USA. [2]Machine Learning Department, NEC Labs America, Princeton, NJ, USA. [3]Translational Data Analytics Institute, The Ohio State University, Columbus, OH, USA. [4]Biomedical Informatics, The Ohio State University, Columbus, OH, USA. ✉e-mail: ning.104@osu.edu

Table 1 | Overall comparison on optimizing plog*P*

Model	$\delta = 0.0$		$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$	
	Imprv ± s.d.	Sim ± s.d.	Imprv ± s.d.	Sim ± s.d.	Imprv ± s.d.	Sim ± s.d.	Imprv ± s.d.	Sim ± s.d.
JT-VAE	1.91 ± 2.04	0.28 ± 0.15	1.68 ± 1.85	0.33 ± 0.13	0.84 ± 1.45	0.51 ± 0.10	0.21 ± 0.71	0.69 ± 0.06
GCPN	4.20 ± 1.28	0.32 ± 0.12	4.12 ± 1.19	0.34 ± 0.11	2.49 ± 1.30	0.47 ± 0.08	0.79 ± 0.63	0.68 ± 0.08
JTNN	-	-	-	-	3.55 ± 1.54	0.46 ± 0.06	2.33 ± 1.19	0.66 ± 0.05
HierG2G	-	-	-	-	3.98 ± 1.46	0.46 ± 0.06	2.49 ± 1.09	0.66 ± 0.05
GraphAF	2.94 ± 1.55	0.31 ± 0.15	2.65 ± 1.29	0.35 ± 0.12	1.62 ± 1.16	0.51 ± 0.10	0.34 ± 0.46	0.69 ± 0.06
MoFlow	2.39 ± 1.47	0.54 ± 0.22	2.26 ± 1.37	0.59 ± 0.17	2.04 ± 1.24	0.65 ± 0.12	1.46 ± 1.09	0.71 ± 0.07
Modof-pipe	7.61 ± 2.30	0.21 ± 0.15	6.23 ± 1.77	0.34 ± 0.12	5.00 ± 1.53	0.48 ± 0.09	2.72 ± 1.53	0.65 ± 0.05
Modof-pipe^m	9.37 ± 2.04	0.12 ± 0.08	7.58 ± 1.65	0.27 ± 0.07	5.89 ± 1.57	0.46 ± 0.06	3.14 ± 1.77	0.65 ± 0.05

Imprv, the average improvement in plog*P*; s.d., standard deviation; sim, similarity between the original molecules M_x and optimized molecules M_y; -, not reported in the literature. We calculated 'sim ± s.d.' for JTNN and HierG2G using the optimized molecules provided by JTNN and our reproduced results for HierG2G, respectively.

- Modof-pipe^m improves the performance of Modof-pipe by at least 17.8%.
- Modof-pipe^m and Modof-pipe also show superior performance on two other benchmarking tasks, optimizing molecule binding affinities against the dopamine D2 receptor and improving the drug-likeness estimated by quantitative measures.

Related work

A variety of deep generative models have been developed to generate molecules with desired properties. These generative models include reinforcement learning (RL)-based models, generative adversarial networks (GAN)-based models, flow-based generative models and variational autoencoder (VAE)-based models, among others. Among RL-based models, You et al.[9] developed a graph convolutional policy network (GCPN) to sequentially add new atoms and corresponding bonds to construct new molecules. In the flow-based models, Shi et al.[12] developed an autoregressive model (GraphAF), in which they learned an invertible mapping between Gaussian distribution and molecule structures, and applied RL to fine-tune the generation process. Zang and Wang[13] developed a flow-based method (MoFlow) in which they utilized bond flow to learn an invertible mapping between bond adjacency tensors and Gaussian distribution, and then applied a graph conditional flow to generate an atom-type matrix given the bond adjacency tensors. VAE-based generative models are also very popular in molecular graph generation. Jin et al.[8] first decomposed a molecular graph into a junction tree of chemical substructures, and then used a junction tree VAE (JT-VAE) to generate and assemble new molecules. Jin et al.[14] developed a junction tree-based encoder–decoder neural model (JTNN), which learns a translation mapping between a pair of molecules to optimize one into another. Jin et al.[15] replaced the small chemical substructures used in JT-VAE with larger graph motifs, and modified JTNN into an autoregressive hierarchical encoder–decoder model (HierG2G). Additional related work including fragment-based VAE[16], Teacher and Student polish (T&S polish)[17], scaffold-based VAE[18] and other genetic algorithm-based methods[19,20] are discussed in Supplementary Section 1.

The existing generative methods typically encode the entire molecular graphs and generate whole, new molecules from an empty or a randomly selected structure. Unlike these methods, Modof learns from and encodes the difference between molecules before and after optimization. The learning and generative processes are thus less complex and are able to retain major molecular scaffolds.

Problem definition

Following Jin et al.[8], we focused on the optimization of the partition coefficients (log*P*) measured by Crippen log*P* (ref. [21]) and penalized by SA[22] and ring size. Crippen log*P* is a predicted value of the experimental log*P* using the Wildman and Crippen approach[21], and has been demonstrated to have a strong correlation (for example, $r^2 = 0.918$; ref. [21]) with the experimental log*P*. Because it is impractical to measure the experimental log*P* values for a large set of molecules, such as our training set (Supplementary Section 3), or for in silico generated molecules, using Crippen log*P* will enable scalable learning from a large set of molecules, and effective yet accurate evaluation on in silico-optimized molecules. The combined measurement of log*P*, SA and ring size is referred to as penalized log*P*, denoted plog*P*. Higher plog*P* values indicate higher molecule concentrations in the lipid phase with potentially good SA and simple ring structures. Note that Modof can be used to optimize other properties as well, with the property of interest used instead of plog*P*. The optimization of other properties is discussed in Supplementary Section 11. Optimizing multiple properties simultaneously is discussed in Supplementary Section 12. In the rest of this Article, 'property' is by default referred to plog*P*.

Problem definition: Given a molecule M_x, molecule optimization aims to modify M_x into another molecule M_y such that (1) M_y is similar to M_x in its molecular structures (similarity constraint), that is, $\mathrm{sim}(M_x, M_y) \geq \delta$ (δ is a threshold), and (2) M_y is better than M_x in the property of interest (for example, $\mathrm{plog}P(M_y) > \mathrm{plog}P(M_x)$) (property constraint).

Materials

Data. We used the benchmark training dataset provided by Jin and colleagues[15]. This dataset was extracted from the ZINC dataset[23,24] and contains 74,887 pairs of molecules. Every two paired molecules are similar in their molecule structures but different in their plog*P* values. Using the DF-GED[25] algorithm, we extracted 55,686 pairs of molecules from Jin's training dataset such that each extracted pair had only one disconnection site. That is, our training data amount to 26% less than in Jin's dataset. We used these extracted pairs of molecules (104,708 unique molecules) as our training data. Details about training data generation are discussed in the next section. We used Jin's validation set for parameter tuning, and tested on Jin's test dataset of 800 molecules. More details about the training data are provided in Supplementary Section 3.

Training data generation. We used a pair of molecules (M_x, M_y) as a training instance in Modof, where M_x and M_y satisfy both the similarity and property constraints, and M_y is different from M_x in only one fragment at one disconnection site. We constructed such training instances as follows. We first quantified the difference between M_x and M_y using the optimal graph edit distance[26] between their junction tree representations $\mathcal{T}_x$ and $\mathcal{T}_y$, and derived the optimal edit paths to transform $\mathcal{T}_x$ to $\mathcal{T}_y$. Such quantification also identified disconnection sites at M_x during its graph comparison. Details about

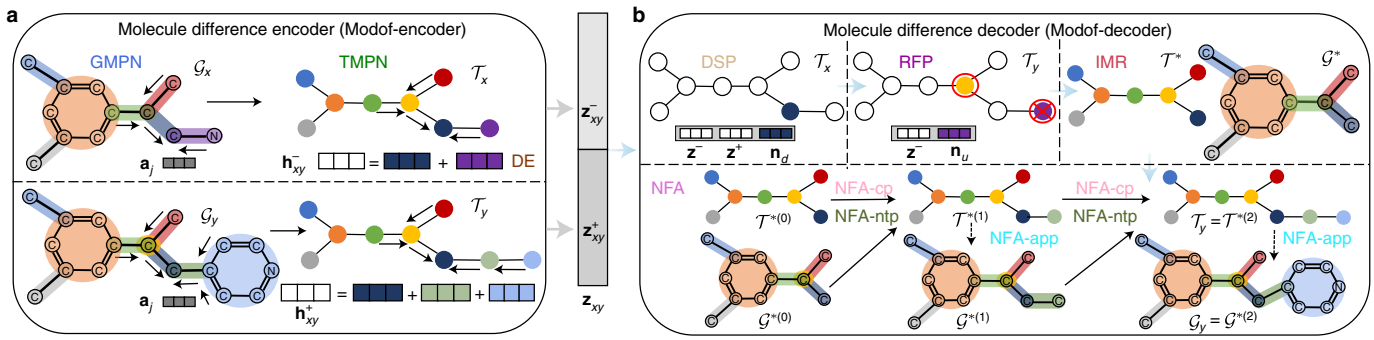

Fig. 1 | Modof model overview. a, The Modof-encoder. Modof first generates atom embeddings of M_x/M_y over molecular graphs $\mathcal{G}_x/\mathcal{G}_y$ using GMPNs, as well as node embeddings over corresponding junction trees $\mathcal{T}_x/\mathcal{T}_y$ using TMPNs. The difference between $\mathcal{T}_x$ and $\mathcal{T}_y$ at the disconnection site (circles in $\mathcal{T}_x/\mathcal{T}_y$) is encoded (DE) into $\mathbf{h}^-_{xy}$ and $\mathbf{h}^+_{xy}$, which then construct two normal distributions $\mathbf{z}^-_{xy}$ and $\mathbf{z}^+_{xy}$. **b**, The Modof-decoder. Using $\mathbf{z}_{xy}$, Modof conducts disconnection site prediction (DSP) to identify site n_d. At neighbours of n_d, Modof conducts removal fragment prediction (RFP) to remove the fragment at n_d. Then, Modof produces an intermediate representation (IMR) of the remaining scaffold ($\mathcal{G}^*$, $\mathcal{T}^*$). Over ($\mathcal{G}^*$, $\mathcal{T}^*$), Modof performs new fragment attachment (NFA) by interactively performing child node connection prediction (NFA-cp), child node type prediction (NFA-ntp) and attachment point prediction (NFA-app) to optimize M_x. In molecule representations, substructures in molecular graphs and their corresponding nodes in junction trees are coded in the same colours.

this process is available in Supplementary Section 4. Identified molecule pairs satisfying similarity and property constraints with only one site of disconnection were used as training instances. For a pair of molecules with a high similarity (for example, above 0.6), it is very likely that they have only one disconnection site, as demonstrated in Supplementary Section 5.

Molecule similarity calculation. We used 2,048D binary Morgan fingerprints with radius of 2 to represent molecules and used the Tanimoto coefficient to measure molecule similarities.

Baseline methods. We compared Modof with state-of-the-art baseline methods for molecule optimization, including JT-VAE[8], GCPN[9], JTNN[14], HierG2G[15], GraphAF[12] and MoFlow[13]:

- JT-VAE encodes and decodes junction trees and assembles new, entire molecular graphs based on decoded junction trees.
- GCPN applies a graph convolutional policy network and iteratively generates molecules by adding atoms and bonds, one by one.
- JTNN learns from molecule pairs and performs molecule optimization to translate molecular graphs.
- HierG2G encodes molecular graphs in a hierarchical fashion, and generates new molecules by generating and connecting structural motifs.
- GraphAF learns an invertible mapping between a prior distribution and molecular structures, and uses reinforcement learning to fine-tune the model for molecule optimization.
- MoFlow learns an invertible mapping between bond adjacency tensors and Gaussian distribution, and then applies a graph conditional flow to generate an atom-type matrix as the representation of a new molecule from the mapping.

Experimental results

Overall comparison on plog*P* optimization. Table 1 presents an overall comparison of Modof-pipe and Modof-pipe^m, both with a maximum of five iterations, and the baseline methods on plog*P* optimization. Note that Modof-pipe^m outputs 20 optimized molecules, as do JTNN and HierG2G. Following GCPN, an additional constraint of molecule size is imposed into Modof-pipe to limit the size of the optimized molecules to be at most 38. As Crippen log*P* tends to be large on large molecules, this additional constraint also prevents Modof-pipe from improving log*P* by simply increasing the molecule size. When there is no similarity constraint ($\delta=0$), that is, it is not required to produce similar molecules out of the optimization, Modof-pipe is able to generate highly optimized molecules with substantially better plog*P* improvement (7.61 ± 2.30), with 81.2% improvement from the best baseline GCPN (4.20 ± 1.28), although with lower similarities between the molecules before and after the optimization. Modof-pipe^m achieves even better performance, with a plog*P* improvement of 9.37 ± 2.04, that is, 123.1% better than GCPN. When the similarity constraint takes effect (for example, $\delta=0.2$, 0.4 and 0.6), Modof-pipe consistently produces molecules that are similar to those before optimization and also with better properties. At $\delta=0.2$, 0.4 and 0.6, Modof-pipe achieves better property improvement (6.23 ± 1.77, 5.00 ± 1.53 and 2.72 ± 1.68, respectively) than all the best baselines (GCPN with 4.12 ± 1.19 at $\delta=0.2$ and HierG2G with 3.98 ± 1.47 at $\delta=0.4$ and 2.49 ± 1.09 at $\delta=0.6$), with 51.2%, 25.6% and 9.2% improvement over the baselines, respectively, although the baselines generate more similar molecules than Modof-pipe. Modof-pipe^m achieves the best performance on property improvement (7.58 ± 1.65, 5.89 ± 1.57 and 3.14 ± 1.77, respectively) with 84.0%, 48.0% and 26.1% improvement over the best baselines, respectively.

When δ is large, we could observe that JTNN and HierG2G tend to decode more aromatic rings, leading to large molecules with over-estimated similarities. However, Modof tends to stop if there are many aromatic rings and thus produces more drug-like molecules[27,28]. Issues related to similarity calculation that will affect optimization performance are discussed in Supplementary Section 7. Still, the overall comparison demonstrates that Modof-pipe and Modof-pipe^m outperform or at least achieve similar performance to state-of-the-art methods.

It is worth noting that our performance is reported on the exact benchmark test set. In our study, we observed some issues of unfair comparison in the existing baseline methods. For example, some baseline methods compared and reported results on a test set other than the benchmark test set. Some reinforcement learning methods used the test molecules to either directly train a model or fine-tune a pre-trained model to optimize the test molecules, which could lead to artificially high performance[29,30]. Detailed discussions on comparison fairness are provided in Supplementary Section 8.

Table 2 | Overall comparison on optimizing DRD2 and QED

Model	Optimizing DRD2						Optimizing QED					
	OM-pic (DRD2(M_y) ≥ 0.5)			OM-trn (imprv ≥ 0.2)			OM-pic (QED(M_y) ≥ 0.9)			OM-trn (imprv ≥ 0.1)		
	Rate (%)	Imprv ± s.d.	Sim ± s.d.	Rate (%)	Imprv ± s.d.	Sim ± s.d.	Rate (%)	Imprv ± s.d.	Sim ± s.d.	Rate (%)	Imprv ± s.d.	Sim ± s.d.
JTNN	78.10	0.83 ± 0.17	0.44 ± 0.05	78.30	0.83 ± 0.17	0.44 ± 0.05	60.50	0.17 ± 0.03	0.47 ± 0.06	67.38	0.17 ± 0.03	0.47 ± 0.07
HierG2G	82.00	0.83 ± 0.16	0.44 ± 0.05	84.00	0.82 ± 0.18	0.44 ± 0.05	**75.12**	0.18 ± 0.03	0.46 ± 0.06	82.38	0.17 ± 0.03	0.46 ± 0.06
JTNN(m)	43.50	0.77 ± 0.15	0.49 ± 0.08	61.60	0.65 ± 0.24	0.49 ± 0.08	40.50	0.17 ± 0.03	0.54 ± 0.09	68.50	0.15 ± 0.03	0.54 ± 0.09
HierG2G(m)	51.80	0.78 ± 0.15	0.49 ± 0.08	70.20	0.66 ± 0.24	0.49 ± 0.08	37.12	0.17 ± 0.03	0.52 ± 0.09	65.88	0.15 ± 0.03	0.53 ± 0.10
Modof-pipe	74.90	0.83 ± 0.14	0.48 ± 0.07	89.00	0.75 ± 0.22	0.48 ± 0.07	40.00	0.17 ± 0.03	0.51 ± 0.08	70.00	0.16 ± 0.03	0.51 ± 0.08
Modof-pipe^m	**88.60**	0.88 ± 0.12	0.46 ± 0.05	**95.90**	0.84 ± 0.18	0.46 ± 0.05	66.25	0.18 ± 0.03	0.48 ± 0.07	**87.62**	0.17 ± 0.03	0.48 ± 0.07

OM-pic: the optimized molecules that achieve a certain property improvement: (1) for DRD2, the optimized molecules M_y should have DRD2 score no less than 0.5; (2) for QED, the optimized molecules M_y should have QED score no less than 0.9. OM-trn: the optimized molecules that achieve a property improvement in a similar degree as in training data: (1) for DRD2, the optimized molecules M_y should satisfy DRD2(M_y) − DRD2(M_x) ≥ 0.2; (2) for QED, the optimized molecules M_y should satisfy QED scores QED(M_y) − QED(M_x) ≥ 0.1. Rate (%) indicates the percentage of optimized molecules in each group (OM, OM-pic, OM-trn) over all test molecules; imprv, average property improvement; s.d., standard deviation; sim, similarity between the original molecules M_x and optimized molecules M_y. Best rate values are shown in bold.

Additional experimental results are provided in Supplementary Section 9, such as overall Modof-pipe performance, transformation over chemical spaces and retaining of molecule scaffolds. Specifically, we compared model complexities (Supplementary Section 9.7), showing that Modof uses at least 40% fewer parameters and 26% less training data but outperforms or achieves results that are comparable to these state-of-the-art baselines. For reproducibility purposes, detailed parameters are reported in Supplementary Section 9.8.

Case study. Among training molecules, the top-five most popular fragments that have been removed from M_x are presented in Fig. 2a with their canonical SMILE strings. The top-five most popular fragments to be attached to generate M_y are presented in Fig. 2b. Overall, the removal fragments in training data comprise, on average, 2.85 atoms and the new attached fragments comprise 7.55 atoms; that is, the optimization is typically done by removing small fragments and then attaching larger fragments.

Figure 2c presents an example of molecule M_x (that is, $M_x^{(0)}$) being optimized via four iterations in Modof-pipe into another molecule $M_x^{(4)}$ under $\delta = 0.4$. At each iteration, only one small fragment (highlighted in red in the figure) is modified from its input, and the plog$P$ value (below each molecule) is improved. In the first iteration, $M_x^{(1)}$ is modified from $M_x^{(0)}$ by removal of the hydroxyl group in $M_x^{(0)}$ and addition of the 2-chlorophenyl group. The hydroxyl group is polar and tends to increase the water solubility of the molecules, while the 2-chlorophenyl group is non-polar and thus more hydrophobic. In addition, the increase in molecular weight brought by the chlorophenyl substituent would contribute to the lower water solubility as well. Thus, the modification from the hydroxyl group to the chlorophenyl group induces the log$P$ increase (from 0.5226 to 3.9465). Meanwhile, the introduction of the 2-chlorophenyl group to the cyclobutyl group adds complexity to the synthesis, in addition to possible steric effects due to the *ortho*-substitution on the aromatic ring, and induces a decrease in SA (from −4.8669 to −4.9955). In the second iteration, the methyl group in $M_x^{(1)}$ is replaced by a trifluoromethyl group. The trifluoromethyl group is more hydrophobic than the methyl group and thus increases the log$P$ value of $M_x^{(2)}$ over $M_x^{(1)}$ (from 3.9465 to 4.6569). Meanwhile, the slightly larger molecule $M_x^{(2)}$ has slightly worse SA (from −4.9955 to −5.1168). If log$P$ is preferred to be lower than 5 as proposed in the Lipinski's rule of five[31], Modof-pipe can be stopped at this iteration. Otherwise, in the following two iterations, more halogens are added to the aromatic ring, which could make the aromatic ring less polar and further decrease water solubility and increase the log$P$ values[32]. These four iterations highlight the interpretability of Modof-pipe corresponding to chemical knowledge. Please note that all the modifications in Modof are learned in an end-to-end fashion from data, without any chemical rules or templates imposed a priori, emphasizing the power of Modof in learning from molecules.

In Fig. 2c, the molecule similarities between $M_x^{(t)}$ ($t = 1, \ldots, 4$) and $M_x^{(0)}$ are 0.630, 0.506, 0.421 and 0.411, respectively. This example also shows that Modof is able to retain the major scaffold of a molecule and optimizes at different disconnection sites during the iterative optimization process. Additional analysis on fragments is provided in Supplementary Section 10.

Performance on DRD2 and QED optimization. In addition to improving plog$P$, another two popular benchmarking tasks for molecule optimization include improving molecule binding affinities against the dopamine D2 receptor (DRD2) and improving the drug-likeness estimated by quantitative measures (QED)[33]. Specifically, given a molecule that does not bind well to the DRD2 receptor (for example, with low binding affinities), the objective of optimizing the DRD2 property is to modify the molecule into another one that will better bind to DRD2. In the QED task, given

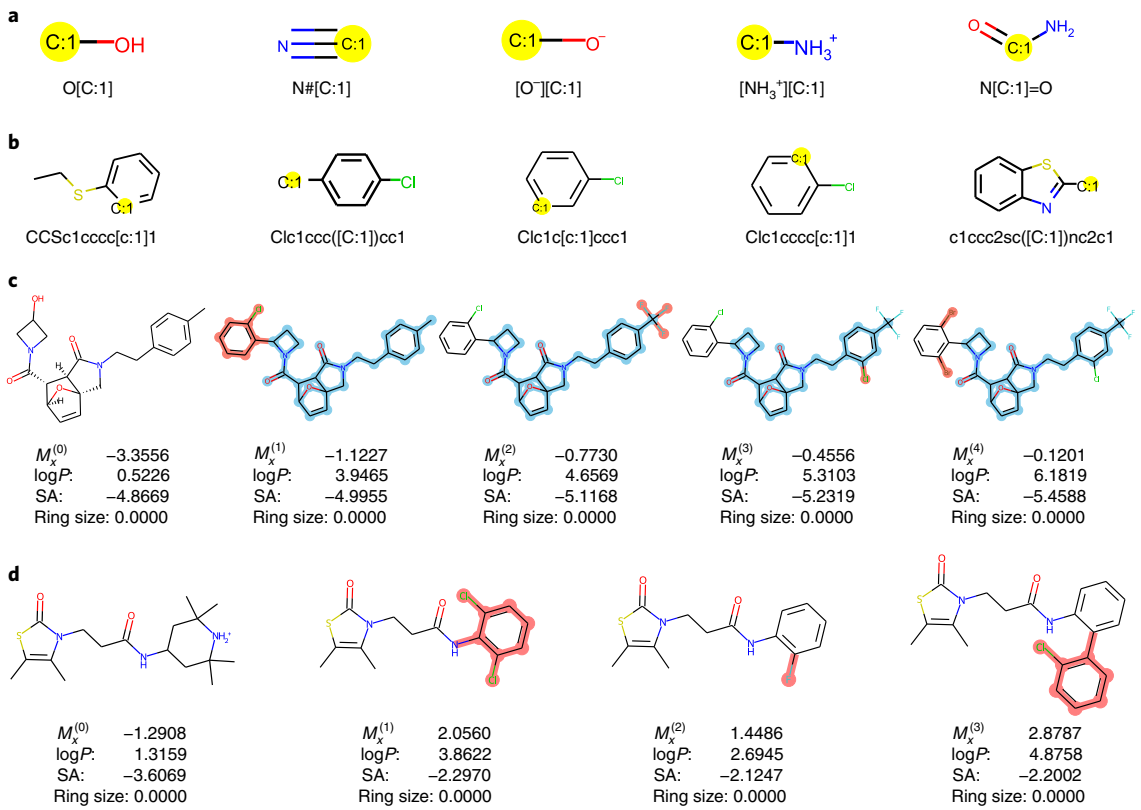

Fig. 2 | Modof-pipe examples for plog*P* optimization. a, Visualization of popular removal fragments. **b**, Visualization of popular attaching fragments. **c**, Modof-pipe optimization example with multiple disconnection sites and multiple Modof iterations. **d**, Local optimization. Disconnection sites are highlighted in yellow. Modified fragments in each step are highlighted in red. Retained scaffolds after Modof-pipe optimization are highlighted in sky blue. Numbers associated with M_x are the corresponding plog*P* values.

a molecule that is not very drug-like, the objective of optimizing the QED property is to modify this molecule into a more 'drug-like' molecule. Table 2 presents the major results in success rates, property improvement and similarity comparison under the similarity constraint $\delta = 0.4$. The results demonstrate that Modof-pipe^m substantially outperforms or is comparable to the baseline methods in optimizing DRD2 and QED, when the success rates are measured either using the benchmark metrics[14,15] (OM-pic in Table 2) or based on training data (OM-trn in Table 2). Figure 3a,b presents two examples of molecule optimization for DRD2 and QED property improvement. In Fig. 3b, in the first iteration, a 4-methoxyphenyl group is removed and a small chain of 2-fluoroethyl group is added, so the number of aromatic rings and the number of hydrogen-bond acceptors are reduced, which makes the compound more drug-like than its predecessor. In the second iteration, a cyclooctyl group is removed from $M_x^{(1)}$ and a 2-fluorophenyl group is added. This modification may induce reduced flexibility—another preferred property of a successful drug. In the following iterations, some commonly used fragments in drug design are used to further modify the molecule into more drug-likeness. Note that, again, QED optimization is completely learned from data in an end-to-end fashion without any medicinal chemistry knowledge imposed by experts. The meaningful optimization in the example in Fig. 3b demonstrates the interpretability of Modof-pipe. More details about these two optimization tasks and results are provided in Supplementary Section 11.

We also conducted experiments to optimize both DRD2 and QED properties of molecules simultaneously, that is, a multi-property optimization task. Details on this multi-property task and results are provided in Supplementary Section 12. Figure 3c presents an example of multi-property molecule optimization, in which both the DRD2 and QED scores of the molecule are consistently increased with the iterations of optimization.

Discussion and conclusions

Molecule optimization using simulated properties. Most of the molecule properties considered in our experiments are based on simulated or predicted values rather than experimentally measured. That is, an independent simulation or machine learning model is first used to generate the property values for the benchmark dataset. For example, Crippen log*P* is estimated via the Wildman and Crippen approach[21], synthesis accessibility is calculated using a scoring function over predefined fragments[22], the DRD2 property is predicted using a support vector machine classifier[34], and the QED property is predicted using a nonlinear classifier combining multiple desirability functions of molecular properties[33]. Although all the existing generative models for molecule optimization[8,9,13–16,19,35–37] use such simulated properties, there are both challenges and opportunities. Challenges arise when the simulation or machine learning models for those property predictions are not sufficiently accurate for various reasons (for example, limited or biased training molecules) and the generative models learned from the inaccurate property values would also be inaccurate or incorrect, resulting in generated molecules that could negatively impact the downstream drug development tasks. However, the opportunities due to the property simulation or prediction can be immense in fully unleashing the power of large-scale, data-driven learning paradigms to stimulate drug development as we continue to improve these simulations and predictions. Specifically, most deep learning-based models for drug development purposes, many of which have been demonstrated to be very promising[38], are not possible without large-scale training

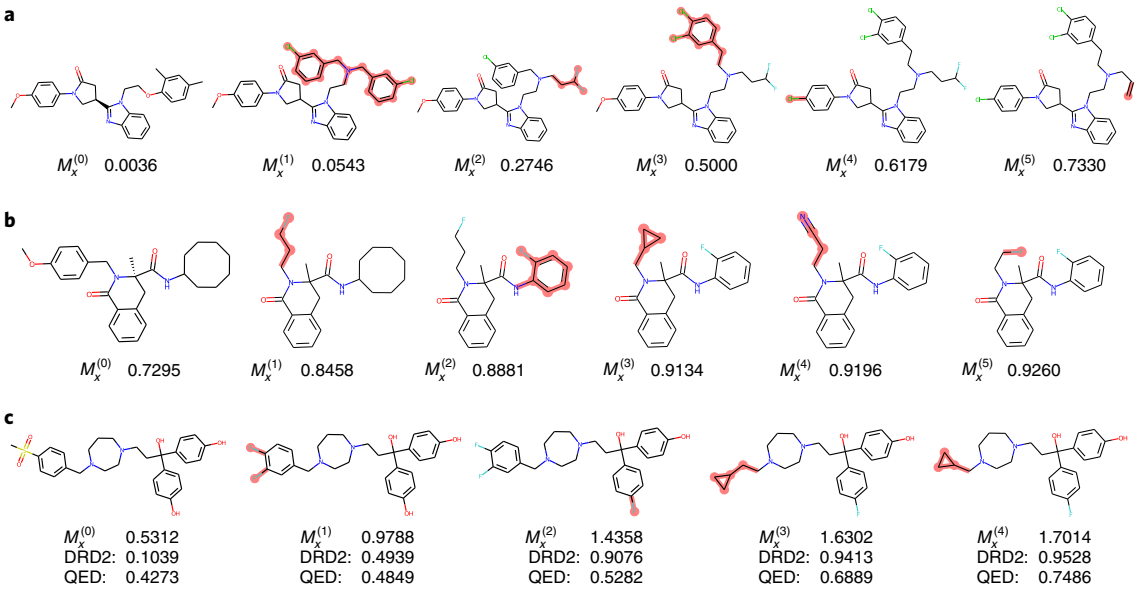

Fig. 3 | Modof-pipe examples for DRD2, QED and multi-property optimization. a, Modof-pipe examples for DRD2 optimization. **b**, Modof-pipe examples for QED optimization. **c**, Modof-pipe examples for multi-property optimization of DRD2 and QED. Modified fragments in each step are highlighted in red. Numbers associated with M_x are the corresponding property values.

data. Although it is impractical, if even possible, to experimentally measure the properties of interest for a large set of molecules (for example, more than 100,000 molecules as in our benchmark training data), the property simulation or prediction of the molecules enables large training data and makes the development of such deep learning methodologies possible. Fortunately, property prediction simulations or models have become more accurate (for example, 98% accuracy for DRD2[34]) due to the accumulation of experimental measurements[39] and the strong learning power of innovative computational approaches. The accurate property simulation or prediction over large-scale molecule data and the powerful learning capability of generative models from such molecule data will together have strong potential to further advance in silico drug development.

Synthesizability and retrosynthesis. Our experiments show that Modof is also able to improve synthesis accessibility (Supplementary Section 9.4). However, it does not necessarily mean that the generated molecules can be easily synthesized. This limitation of Modof is actually common to almost all the computational approaches for molecule generation. A recent study has shown that many molecules generated via deep learning are not easily synthesizable[40], which limits the translational potentials of the generative models in making real impacts in drug development. On the other hand, retrosynthesis prediction via deep learning, which aims to identify a feasible synthesis path for a given molecule through learning and searching from a large collection of synthesis paths, has been an active research area[41,42]. Optimizing molecules towards not only better properties but also better synthesizability, particularly with explicit synthesis paths identified simultaneously, could be a highly interesting and challenging future research direction. Ultimately, we would like to develop a comprehensive computational framework that could generate synthesizable molecules with preferable properties. This would require not only a substantial amount of data to train sophisticated models, but also necessary domain knowledge and human experts looped into the learning process.

In vitro validation. Ultimately, testing of the in silico-generated molecules in a laboratory will be needed to validate the computational methods. Although most existing computational methods are developed in academic environments and thus cannot be easily tested on purchasable or proprietary molecule libraries, and their generated molecules cannot be easily synthesized as we discussed earlier, a few successful stories[43] have demonstrated that powerful computational methods have great potential to truly make new discoveries that can succeed in laboratory validation. Analogous to this molecule optimization and discovery process using deep learning approaches is AlphaFold[44], a deep learning method that predicts protein folding structures. The breakthrough from AlphaFold in solving a 50-year-old grand challenge in biology offers strong evidence of the tremendous power of modern learning approaches, which should not be underestimated. Still, collaborations with the pharmaceutical industry and in vitro testing are very much needed to truly translate the progress with computational methods into a real impact. In addition, effective sampling and/or prioritization of generated molecules to identify a feasible, small set of molecules for small-scale in vitro validation could be a practical solution. This will require the development of new sampling schemes over molecule subspaces and/or the learning of molecule prioritization[45,46] within the molecule generation process. Meanwhile, large-scale in vitro validation of in silico-generated molecules is a challenging but interesting future research direction.

Other issues in computational molecule optimization. A limitation of Modof-pipe is that it employs a local greedy optimization strategy: in each iteration, the input molecules to Modof will be optimized to the best, and if the optimized molecules do not have better properties, they will not go through additional Modof iterations. Detailed discussions about local greedy optimization are provided in Supplementary Section 13.1. In addition to the partition coefficient, there are a lot of factors (for example, toxicity and synthesizability) that need to be considered to develop a molecule into a drug. Discussions about multi-property optimization are available in Supplementary Section 13.2. Target-specific molecule optimization is also discussed in Supplementary Section 13.3. The Modof framework could also be used for compounds or substance property optimization in other application areas (for example, melting or boiling points for volatiles). Related discussions are presented in Supplementary Section 13.4.

Table 3 | Notation

Notation	Meaning
$M = (\mathcal{G}, \mathcal{T})$	Molecule represented by $\mathcal{G}$ and $\mathcal{T}$
$\mathcal{G} = (\mathcal{A}, \mathcal{B})$	Molecular graph with atoms $\mathcal{A}$ and bonds $\mathcal{B}$
$\mathcal{T} = (\mathcal{V}, \mathcal{E})$	Junction tree with nodes $\mathcal{V}$ and edges $\mathcal{E}$
a	An atom in $\mathcal{G}$
b_{ij}	A bond connecting atoms a_i and a_j in $\mathcal{G}$
n	A node in $\mathcal{T}$
e_{uv}	An edge connecting nodes n_u and n_v in $\mathcal{T}$
n_d	Site of disconnection
$\mathcal{A}(n)$, $\mathcal{N}(n)$	Atoms included in a tree node n, n's neighbours
$\mathbf{x}$	Atom type embedding
$\mathbf{m}^{(1\cdots t)}$	Concatenation of $\mathbf{m}^{(1)}$, $\mathbf{m}^{(2)}$, ... $\mathbf{m}^{(t)}$

Conclusions. Modof optimizes molecules at one disconnection site at a time by learning the difference between molecules before and after optimization. With a much less complex model, it achieves substantially better or similar performance to state-of-the-art methods. In addition to the limitations and corresponding future research directions that have been discussed above, another limitation with Modof is that, in Modof, the modification happens at the periphery of molecules. Although this is very common in in vitro lead optimization, we are currently investigating how Modof can be enhanced to modify the internal regions of molecules, if needed, by learning from proper training data with such regions. Additionally, we hope to integrate domain-specific knowledge in the Modof learning process to facilitate increased explainability in the learning and generative process.

Methods

Modof modifies one fragment (for example, a ring system, a linker, a side chain) of a molecule at a time, and thus only encodes and decodes the fragment that needs modification. The site of M where the fragment is modified is referred to as the site of disconnection and denoted n_d, which corresponds to a node in the junction tree representation (discussed in the section Molecule representations and notations). Figure 1 presents an overview of Modof. All the algorithms are presented in Supplementary Section 14. Discussions on the single-disconnection-site rationale are presented in Supplementary Section 5.

Molecule representations and notations. We represent a molecule M_x using a molecular graph $\mathcal{G}_x$ and a junction tree $\mathcal{T}_x$. $\mathcal{G}_x$ is denoted $\mathcal{G}_x = (\mathcal{A}_x, \mathcal{B}_x)$, where $\mathcal{A}_x$ is the set of atoms in M_x, and $\mathcal{B}_x$ is the set of corresponding bonds. In the junction tree representation, $\mathcal{T}_x = (\mathcal{V}_x, \mathcal{E}_x)$ (ref. [8]), all the rings and bonds in M_x are extracted as nodes in $\mathcal{V}_x$, and nodes with common atoms are connected with edges in $\mathcal{E}_x$. Thus, each node $n \in \mathcal{V}_x$ is a substructure (for example, a ring, a bond and its connected atoms) in $\mathcal{G}_x$. We denote the atoms included in node n as $\mathcal{A}_x(n)$ and refer to the nodes connected to n in $\mathcal{T}_x$ as its neighbours, denoted $\mathcal{N}_x(n)$. Thus, each edge $(n_u, n_v) \in \mathcal{E}_x$ actually corresponds to the common atoms $\mathcal{A}_x(n_u) \cap \mathcal{A}_x(n_v)$ between n_u and n_v. When no ambiguity arises, we will eliminate subscript x in the notations. Note that atoms and bonds are the terms used for molecular graph representations, and nodes and edges are used for junction tree representations. In this Article, all the embedding vectors are by default column vectors, represented by lower-case bold letters; all the matrices are represented by upper-case letters. Key notations are listed in Table 3.

Molecular difference encoder (Modof-encoder). Given two molecules (M_x, M_y), Modof (algorithm 1 in Supplementary Section 14) learns and encodes the difference between M_x and M_y using message passing networks[47] over graphs $\mathcal{G}_x$ and $\mathcal{G}_y$, denoted graph message passing networks (GMPNs), and over junction trees $\mathcal{T}_x$ and $\mathcal{T}_y$, denoted tree message passing networks (TMPNs), in three steps.

Step 1 Atom embedding over graphs (GMPN). Modof first represents atoms using embeddings to capture atom types and their local neighbourhood structures by propagating messages along bonds over molecular graphs. Modof uses a one-hot encoding $\mathbf{x}_i$ to represent the type of atom a_i, and a one-hot encoding $\mathbf{x}_{ij}$ to represent the type of bond b_{ij} connecting a_i and a_j. Each bond b_{ij} is associated with two messages $\mathbf{m}_{ij}$ and $\mathbf{m}_{ji}$ encoding the messages propagating from atom a_i to a_j and vice versa. The $\mathbf{m}_{ij}^{(t)}$ in the tth iteration of the GMPN is updated as follows:

$$\mathbf{m}_{ij}^{(t)} = \mathrm{ReLU}\left(W_1^a \mathbf{x}_i + W_2^a \mathbf{x}_{ij} + W_3^a \sum_{a_k \in \mathcal{N}(a_i)\setminus\{a_j\}} \mathbf{m}_{ki}^{(t-1)}\right),$$

where $\mathbf{m}_{ki}^{(0)}$ is initialized as zero, and W_i^a ($i = 1, 2, 3$) are the learnable parameter matrices. Thus, the message $\mathbf{m}_{ij}^{(t)}$ encodes the information of all length-t paths passing through b_{ij} to a_j in the graph. After t_a iterations of message passing, the atom embedding $\mathbf{a}_j$ is updated as follows:

$$\mathbf{a}_j = \mathrm{ReLU}\left(U_1^a \mathbf{x}_j + U_2^a \sum_{a_i \in \mathcal{N}(a_j)} \mathbf{m}_{ij}^{(1\cdots t_a)}\right),$$

where $\mathbf{m}_{ij}^{(1\cdots t_a)}$ is the concatenation of message vectors from all iterations, and U_1^a and U_2^a are learnable parameter matrices. Thus, the atom embedding $\mathbf{a}_j$ aggregates information from a_j's t_a-hop neighbours, as in Xu et al.[48], to improve the atom embedding representation power.

Step 2 Node embedding over junction trees (TMPN). Modof encodes nodes in junction trees into embeddings to capture their local neighbourhood structures by passing messages along the tree edges. To produce rich representations of nodes, Modof first aggregates the information of atoms within a node n_u into an embedding $\mathbf{s}_u$, and the information of atoms shared by a tree edge e_{uv} into an embedding $\mathbf{s}_{uv}$ through the following pooling:

$$\mathbf{s}_u = \sum_{a_i \in \mathcal{A}(n_u)} \mathbf{a}_i, \tag{1}$$

$$\mathbf{s}_{uv} = \sum_{a_i \in \mathcal{A}(n_u) \cap \mathcal{A}(n_v)} \mathbf{a}_i. \tag{2}$$

Modof also uses a learnable embedding $\mathbf{x}_u$ to represent the type of node n_u. Thus, $\mathbf{m}_{uv}^{(t)}$ from node n_u to n_v in the tth iteration of TMPN is updated as follows:

$$\mathbf{m}_{uv}^{(t)} = \mathrm{ReLU}\left(W_1^n \mathrm{ReLU}\left(W_2^n[\mathbf{x}_u;\mathbf{s}_u]\right) + W_3^n \mathbf{s}_{uv} + W_4^n \sum_{n_w \in \mathcal{N}(n_u)\setminus\{n_v\}} \mathbf{m}_{wu}^{(t-1)}\right),$$

where $[\mathbf{x}_u; \mathbf{s}_u]$ is a concatenation of $\mathbf{x}_u$ and $\mathbf{s}_u$ so as to represent comprehensive node information, and W_i^n ($i = 1, 2, 3, 4$) are learnable parameter matrices. Similarly to the messages in GMPNs, $\mathbf{m}_{uv}^{(t)}$ encodes the information of all length-t paths passing through edge e_{uv} to n_v in the tree. After t_n iterations, the node embedding $\mathbf{n}_v$ is updated as follows:

$$\mathbf{n}_v = \mathrm{ReLU}\left(U_1^n \mathrm{ReLU}\left(U_2^n[\mathbf{x}_v;\mathbf{s}_v]\right) + U_3^n \sum_{n_u \in \mathcal{N}(n_v)} \mathbf{m}_{uv}^{(1\cdots t_n)}\right), \tag{3}$$

where U_i^n ($i = 1, 2, 3$) are the learnable parameter matrices.

Step 3 Difference embedding. The difference embedding between M_x and M_y is calculated by pooling the node embeddings from $\mathcal{T}_x$ and $\mathcal{T}_y$ as follows:

$$\mathbf{h}_{xy}^- = \sum_{n_x \in \{\mathcal{V}_x \setminus \mathcal{V}_y\} \cup \{n_d \in \mathcal{V}_x\}} \mathbf{n}_x,$$

$$\mathbf{h}_{xy}^+ = \sum_{n_y \in \{\mathcal{V}_y \setminus \mathcal{V}_x\} \cup \{n_d \in \mathcal{V}_y\}} \mathbf{n}_y,$$

where $\mathbf{n}_x/\mathbf{n}_y$ are the embeddings of nodes only appearing in and learned from $\mathcal{T}_x/\mathcal{T}_y$ via the TMPN. Note that n_d in the above equations is the site of disconnection, and both $\mathcal{T}_x$ and $\mathcal{T}_y$ have the common node n_d. Thus, $\mathbf{h}_{xy}^-$ essentially represents the fragment that should be removed from M_x at n_d and $\mathbf{h}_{xy}^+$ represents the fragment that should be attached to M_x at n_d afterwards so as to modify M_x into M_y. We will discuss how to identify n_d, and the removed and new attached fragments at n_d in M_x and M_y in the section 'Molecular difference decoder (Modof-decoder)'.

As in VAE[49], we map the two difference embeddings $\mathbf{h}_{xy}^-$ and $\mathbf{h}_{xy}^+$ into two normal distributions by computing the mean and log variance with fully connected layers $\mu(\cdot)$ and $\Sigma(\cdot)$. We then sample the latent vectors $\mathbf{z}_{xy}^-$ and $\mathbf{z}_{xy}^+$ from these two distributions and concatenate them into one latent vector $\mathbf{z}_{xy}$, that is:

$$\mathbf{z}_{xy}^- \sim N(\mu^-(\mathbf{h}_{xy}^-), \Sigma^-(\mathbf{h}_{xy}^-)),$$

$$\mathbf{z}_{xy}^+ \sim N(\mu^+(\mathbf{h}_{xy}^+), \Sigma^+(\mathbf{h}_{xy}^+)),$$

$$\mathbf{z}_{xy} = [\mathbf{z}_{xy}^-; \mathbf{z}_{xy}^+]. \tag{4}$$

Thus, $\mathbf{z}_{xy}$ encodes the difference between M_x and M_y.

Molecular difference decoder (Modof-decoder). Following the autoencoder idea, Modof decodes the difference embedding $\mathbf{z}_{xy}$ (equation (4)) into edit operations that change M_x into M_y. Specifically, Modof first predicts a node n_d in $\mathcal{T}_x$ as the disconnection site. This node will split $\mathcal{T}_x$ into several fragments, and the number

of resulting fragments depends on the number of n_d's neighbouring nodes $\mathcal{N}(n_d)$. Modof then predicts which frag ments to remove from M_x and merges the remaining fragments with n_d into an intermediate representation $M^* = (\mathcal{G}^*, \mathcal{T}^*)$. After that, Modof attaches new fragments sequentially, starting from n_d to $(\mathcal{G}^*, \mathcal{T}^*)$. The decoding process (algorithm 2 in Supplementary Section 14) has four steps.

Step 1 Disconnection site prediction. Modof predicts a disconnection score for each $\mathcal{T}_x's$ node n_u as follows:

$$f_d(n_u) = (\mathbf{w}^d)^T \tanh(W_1^d \mathbf{n}_u + W_2^d \mathbf{z}), \forall\, n_u \in \mathcal{V}_x, \quad (5)$$

where $\mathbf{n}_u$ is n_u's embedding (equation (3)) in $\mathcal{T}_x$, and $\mathbf{w}^d$ and W_i^d ($i=1, 2$) are the learnable parameter vector and matrices, respectively. The node with the largest disconnection score is predicted as the disconnection site n_d. Intuitively, Modof considers the neighbouring or local structures of n_u (in $\mathbf{n}_u$) and 'how likely' edit operations (represented by $\mathbf{z}$) can be applied at n_u. To learn f_d, Modof uses the negative log-likelihood of the ground-truth disconnection site in tree $\mathcal{T}_x$ as the loss function.

Step 2 Removal fragment prediction. Modof predicts which fragments separated by n_d should be removed from $\mathcal{T}_x$. For each node n_u connected to n_d, Modof predicts a removal score as follows:

$$f_r(n_u) = \sigma((\mathbf{w}^r)^T \mathrm{ReLU}(W_1^r \mathbf{n}_u + W_2^r \mathbf{z}^-)), \forall e_{ud} \in \mathcal{E}_x, \quad (6)$$

where $\sigma(\cdot)$ is the sigmoid function, and $\mathbf{w}^r$ and W_i^r ($i=1, 2$) are the learnable parameter vector and matrices, respectively. The fragment with a removal score greater than 0.5 is predicted to be removed. Thus, there could be multiple or no fragments removed. Intuitively, Modof considers the local structures of the fragment (that is, $\mathbf{n}_u$) and 'how likely' this fragment should be removed (represented by $\mathbf{z}^-$). To learn f_r, Modof minimizes the binary cross-entropy loss to maximize the predicted scores of ground-truth removed fragments in $\mathcal{T}_x$.

Step 3 Intermediate representation. After fragment removal, Modof merges the remaining fragments together with the disconnection site n_d into an intermediate representation $M^* = (\mathcal{G}^*, \mathcal{T}^*)$. M^* may not be a valid molecule after some fragments are removed (some bonds are broken). It represents the scaffold of M_x that should remain unchanged during optimization. Modof first removes a fragment so as to identify such a scaffold and then adds a fragment to the scaffold to modify the molecule.

Step 4 New fragment attachment. Modof modifies M^* into the optimized M_y by attaching a new fragment (algorithm 3 in Supplementary Section 14). Modof uses the following four predictors to sequentially attach new nodes to $\mathcal{T}^*$. The predictors will be applied iteratively, starting from n_d, on each newly attached node in $\mathcal{T}^*$. The attached new node in the tth step is denoted $n^{*(t)}$ ($n^{*(0)} = n_d$), and the corresponding molecular graph and tree are denoted $\mathcal{G}^{*(t)}$ ($\mathcal{G}^{*(0)} = \mathcal{G}^*$) and $\mathcal{T}^{*(t)}$ ($\mathcal{T}^{*(0)} = \mathcal{T}^*$), respectively.

Step 4.1 Child connection prediction (NFA-cp). Modof first predicts whether $n^{*(t)}$ should have a new child node attached to it, with the probability calculated as follows:

$$f_c(n^{*(t)}) = \sigma((\mathbf{w}^c)^T \mathrm{ReLU}(W_1^c \mathbf{n}^{*(t)} + W_2^c \mathbf{z}^+)), \quad (7)$$

where $\mathbf{n}^{*(t)}$ is the embedding of node $n^{*(t)}$ learned over $(\mathcal{T}^{*(t)}, \mathcal{G}^{*(t)})$ (equation (3)), $\mathbf{z}^+$ (equation (4)) indicates how much $\mathcal{T}^{*(t)}$ should be expanded, and $\mathbf{w}^c$ and W_i^c ($i=1, 2$) are the learnable parameter vector and matrices. If $f_c(n^{*(t)})$ is above 0.5, Modof predicts that $n^{*(t)}$ should have a new child node and thus child node type prediction will follow; otherwise, the optimization process stops at $n^{*(t)}$. To learn f_c, Modof minimizes a binary cross-entropy loss to maximize the probabilities of ground-truth child nodes. Note that $n^{*(t)}$ may have multiple children, so, once a child is generated as in the following steps and attached to $\mathcal{T}^{*(t)}$, another child connection prediction will be conducted at $n^{*(t)}$ with the updated embedding $\mathbf{n}^{*(t)}$) over the expanded $(\mathcal{T}^{*(t)}, \mathcal{G}^{*(t)})$. The above process is iterated until $n^{*(t)}$ is predicted to have no more children.

Step 4.2 Child node type prediction (NFA-ntp). The new child node of $n^{*(t)}$ is denoted n_c. Modof predicts the type of n_c by calculating the probabilities of all types of node that can be attached to $n^{*(t)}$ as follows:

$$f_l(n_c) = \mathrm{softmax}(U^l \times \mathrm{ReLU}(W_1^l \mathbf{n}^{*(t)} + W_2^l \mathbf{z}^+)), \quad (8)$$

where softmax$(\cdot)$ converts a vector of values into probabilities, and U^l and W_i^l ($i=1, 2$) are learnable matrices. Modof assigns the new child n_c the node type x_c corresponding to the highest probability. Modof learns f_l by minimizing the cross-entropy to maximize the likelihood of true child node types.

Step 4.3 Attachment point prediction (NFA-app). If node $n^{*(t)}$ is predicted to have a child node n_c, the next step is to connect $n^{*(t)}$ and n_c. If $n^{*(t)}$ and n_c share one or multiple atoms (for example, $n^{*(t)}$ and n_c form a fused ring and thus share two adjacent atoms) that can be unambiguously determined as the attachment point(s) based on chemical rules, Modof will connect $n^{*(t)}$ and n_c via the atom(s). Otherwise, if $n^{*(t)}$ and n_c have multiple connection configurations, Modof predicts the attachment atoms at $n^{*(t)}$ and n_c, respectively.

Step 4.3.1 Attachment point prediction at the parent node (NFA-app-p). Modof scores each candidate attachment point at parent node $n^{*(t)}$, denoted a_p^*, using

$$g_p(a_p^*) = (\mathbf{w}^p)^T \tanh(W_1^p \mathbf{a}_p^* + W_2^p \mathbf{x}_c + W_3^p \times \mathrm{ReLU}(U_2^n[\mathbf{x}^{*(t)}; \tilde{\mathbf{s}}^{*(t)}]) + W_4^p \mathbf{z}^+), \quad (9)$$

where $\mathbf{a}_p^* = \sum_{a_i \in a_p^*} \tilde{\mathbf{a}}_i$ represents the embedding of a_p^* (a_p^* could be an atom or a bond), $\tilde{\mathbf{a}}_i$ is calculated by GMPN over $\mathcal{G}^{*(t)}$, U_2^n is as in equation (3), $\tilde{\mathbf{s}}^{*(t)}$ is the sum of the embeddings of all atoms in $n^{*(t)}$ (equation (1)), and $\mathbf{w}^p$ and W_i^p ($i=1$, 2, 3, 4) are the learnable and matrices. Modof intuitively measures how likely a_p^* can be attached to n_c by looking at a_p^* its own (that is, $\mathbf{a}_p^*$), its context in $n^{*(t)}$ (that is, $\mathbf{x}^{*(t)}$ and neighbours $\tilde{\mathbf{s}}^{*(t)}$), its connecting node n_c (that is, $\mathbf{x}_c$) and how much $n^{*(t)}$ should be expanded (represented by $\mathbf{z}^+$). The candidate with the highest score is selected as the attachment point in $n^{*(t)}$. Modof learns g_p by minimizing the negative log-likelihood of ground-truth attachment points.

Step 4.3.2 Attachment point prediction at the child node (NFA-app-c). Modof scores each candidate attachment point at the child node n_c, denoted a_c^*, using

$$g_c(a_c^*) = (\mathbf{w}^o)^T \tanh(W_1^o \mathbf{a}_c^* + W_2^o \mathbf{x}_c + W_3^o \mathbf{a}_p^* + W_4^o \mathbf{z}^+), \quad (10)$$

where $\mathbf{a}_c^* = \sum_{a_i \in a_c^*} \tilde{\mathbf{a}}_i$ represents the embedding of a_c^* (a_c^* could be an atom or a bond) and $\tilde{\mathbf{a}}_i$ is the embedding of a_i calculated over n_c via GMPN, and $\mathbf{w}^o$ and W_i^o ($i=1, 2, 3, 4$) are the learnable parameters. Modof intuitively measures how likely candidate a_c^* can be attached to a_p^* at $n^{*(t)}$ by looking at a_c^* its own (that is, $\mathbf{a}_c^*$), the features of a_p^* (that is, $\mathbf{a}_p^*$), its context in n_c (that is, $\mathbf{x}_c$) and how much $n^{*(t)}$ should be expanded (that is, $\mathbf{z}^+$). The candidate with the highest score is selected as the attachment point in n_c. Modof learns g_c by minimizing the negative log-likelihood of ground-truth attachment points.

Valence checking. In NFA-app, Modof incorporates a valence check to only generate and predict legitimate candidate attachment points that do not violate valence laws.

Molecule size constraint. Following You et al.[9], for plog$P$ optimization, we limit the size of the optimized molecules to at most 38 (38 is the maximum number of atoms in the molecules in the ZINC dataset[23]). With this molecule size constraint, Modof can avoid increasing plog$P$ by trivially increasing the molecule size, which may have the effect of improving plog$P$ (ref. [50]).

Sampling schemes. In the decoding process, for each M_x, Modof samples 20 times from the latent space of $\mathbf{z}$ and optimizes M_x accordingly. Among all decoded molecules satisfying the similarity constraint with M_x, Modof selects the one with the best property as its output.

Modof pipelines. A pipeline of Modof models, denoted Modof-pipe (algorithm 4 in Supplementary Section 14), is constructed with a series of identical Modof models, with the output molecule from one Modof model as the input to the next. Given an input molecule $M^{(t)}$ to the tth Modof model ($M^{(0)}=M$), Modof first optimizes $M^{(t)}$ into $M^{(t+1)}$ as the output of this model. $M^{(t+1)}$ is then fed into the $(t+1)$th model if it satisfies the similarity constraint $\mathrm{sim}(M^{(t+1)}, M) > \delta$ and the property constraint plog$P(M^{(t+1)}) >$ plog$P(M^{(t)})$. Otherwise, $M^{(t)}$ is output as the final result and Modof-pipe stops. In addition to Modof-pipe, which outputs one optimized molecule for each input molecule, Modof-pipe^m has been developed to output multiple optimized molecules for each input molecule. Details about Modof-pipe^m are provided in Supplementary Section 2.

The advantages of this iterative, one-fragment-at-one-time optimization process include the following: (1) it is easier to control intermediate optimization steps so as to result in optimized molecules of desired similarities and properties; (2) it is easier to optimize multiple fragments in a molecule that are far apart; (3) it follows a rational molecule design process[11] and thus could enable more insights and inform in vitro lead optimization.

Model training. During model training, we apply teacher forcing to feed the ground truth instead of the prediction results to the sequential decoding process. Following the idea of VAE, we minimize the following loss function to maximize the likelihood $P(M_y|\mathbf{z}, M_x)$. Thus, the optimization problem is formulated as

$$\min_{\Theta} -\beta D_{\mathrm{KL}}(q_\phi(\mathbf{z}|M_x, M_y) \parallel p_\theta(\mathbf{z})) + E_{q_\phi(\mathbf{z}|M_x, M_y)}[\log p_\theta(M_y|\mathbf{z}, M_x)], \quad (11)$$

where Θ is the set of parameters, $q_\phi()$ is an estimated posterior probability function (Modof-encoder), $p_\theta(M_y|\mathbf{z}, M_x)$ is the probabilistic decoder representing the likelihood of generating M_y given the latent embedding $\mathbf{z}$ and M_x, and the

prior $p_\theta(\mathbf{z})$ follows $\mathcal{N}(0, \mathbf{I})$. In the above problem, $D_{\mathrm{KL}}()$ is the Kullback–Leibler divergence between $q_\phi()$ and $p_\theta()$. Specifically, the second term represents the prediction or empirical error, defined as the sum of all the loss functions in the above six predictions (equations (5) to (10)). We use AMSGRAD[51] to optimize the learning objective.

Data availability

The data used in this manuscript are available publicly from Chen et al.[52] and https://github.com/ziqi92/Modof. Source data are provided with this paper.

Code availability

The code for Modof, Modof-pipe and Modof-pipe^m is publicly available from Chen et al.[52] and https://github.com/ziqi92/Modof.

Received: 27 December 2020; Accepted: 4 October 2021;
Published online: 9 December 2021

References

1. Jorgensen, W. L. Efficient drug lead discovery and optimization. *Acc. Chem. Res.* **42**, 724–733 (2009).
2. Verdonk, M. L. & Hartshorn, M. J. Structure-guided fragment screening for lead discovery. *Curr. Opin. Drug Discov. Dev.* **7**, 404–410 (2004).
3. de Souza Neto, L. R. et al. In silico strategies to support fragment-to-lead optimization in drug discovery. *Front. Chem* **8**, 93 (2020).
4. Hoffer, L. et al. Integrated strategy for lead optimization based on fragment growing: the diversity-oriented-target-focused-synthesis approach. *J. Med. Chem.* **61**, 5719–5732 (2018).
5. Gerry, C. J. & Schreiber, S. L. Chemical probes and drug leads from advances in synthetic planning and methodology. *Nat. Rev. Drug Discov.* **17**, 333–352 (2018).
6. Sattarov, B. et al. De novo molecular design by combining deep autoencoder recurrent neural networks with generative topographic mapping. *J. Chem. Inf. Model.* **59**, 1182–1196 (2019).
7. Sanchez-Lengeling, B. & Aspuru-Guzik, A. Inverse molecular design using machine learning: generative models for matter engineering. *Science* **361**, 360–365 (2018).
8. Jin, W., Barzilay, R. & Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. In *Proc. Machine Learning Research* Vol. 80 (eds Dy, J. & Krause, A.), 2323–2332 (PMLR, 2018).
9. You, J., Liu, B., Ying, Z., Pande, V. & Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in Neural Information Processing Systems* Vol. 31 (eds Bengio, S. et al.) 6410–6421 (Curran Associates, 2018).
10. Murray, C. & Rees, D. The rise of fragment-based drug discovery. *Nat. Chem.* **1**, 187–192 (2009).
11. Hajduk, P. J. & Greer, J. A decade of fragment-based drug design: strategic advances and lessons learned. *Nat. Rev. Drug Discov.* **6**, 211–219 (2007).
12. Shi, C. et al. Graphaf: a flow-based autoregressive model for molecular graph generation. In *Proc. 8th International Conference on Learning Representations* (OpenReview.net, 2020).
13. Zang, C. & Wang, F. Moflow: an invertible flow model for generating molecular graphs. In *Proc. 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (eds Gupta, R. et al.) 617–626 (ACM, 2020).
14. Jin, W., Yang, K., Barzilay, R. & Jaakkola, T. S. Learning multimodal graph-to-graph translation for molecule optimization. In *Proc. 7th International Conference on Learning Representations* (2019).
15. Jin, W., Barzilay, R. & Jaakkola, T. S. Hierarchical generation of molecular graphs using structural motifs. In *Proc. 37th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* Vol. 119 (eds Daumé, H. III & Singh, H.) 4839–4848 (PMLR, 2020).
16. Podda, M., Bacciu, D. & Micheli, A. A deep generative model for fragment-based molecule generation. In *Proc. Twenty Third International Conference on Artificial Intelligence and Statistics*, *Proc. Machine Learning Research* Vol. 108 (eds Chiappa, S. & Calandra, R.) 2240–2250 (PMLR, 2020).
17. Ji, C., Zheng, Y., Wang, R., Cai, Y. & Wu, H. Graph Polish: a novel graph generation paradigm for molecular optimization. Preprint at https://arxiv.org/abs/2008.06246 (2021).
18. Lim, J., Hwang, S.-Y., Moon, S., Kim, S. & Kim, W. Y. Scaffold-based molecular design with a graph generative model. *Chem. Sci.* **11**, 1153–1164 (2020).
19. Ahn, S., Kim, J., Lee, H. & Shin, J. Guiding deep molecular optimization with genetic exploration. In *Advances in Neural Information Processing Systems* Vol. 33 (eds Larochelle, H. et al.) (Curran Associates, 2020).
20. Nigam, A., Friederich, P., Krenn, M. & Aspuru-Guzik, A. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. In *Proc. 8th International Conference on Learning Representations* (OpenReview.net, 2020).
21. Wildman, S. A. & Crippen, G. M. Prediction of physicochemical parameters by atomic contributions. *J. Chem. Inf. Comput. Sci.* **39**, 868–873 (1999).
22. Ertl, P. & Schuffenhauer, A. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *J. Cheminf.* **1**, 8 (2009).
23. Sterling, T. & Irwin, J. J. Zinc 15—ligand discovery for everyone. *J. Chem. Inf. Model.* **55**, 2324–2337 (2015).
24. Gómez-Bombarelli, R. et al. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Cent. Sci.* **4**, 268–276 (2018).
25. Abu-Aisheh, Z., Raveaux, R., Ramel, J.-Y. & Martineau, P. An exact graph edit distance algorithm for solving pattern recognition problems. In *Proc. International Conference on Pattern Recognition Applications and Methods* Vol. 1, 271–278 (SciTePress, 2015).
26. Sanfeliu, A. & Fu, K. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.* **SMC-13**, 353–362 (1983).
27. Lipinski, C. A. Lead- and drug-like compounds: the rule-of-five revolution. *Drug Discov. Today Technol.* **1**, 337–341 (2004).
28. Ghose, A. K., Viswanadhan, V. N. & Wendoloski, J. J. A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. A qualitative and quantitative characterization of known drug databases. *J. Comb. Chem.* **1**, 55–68 (1999).
29. Whiteson, S., Tanner, B., Taylor, M. E. & Stone, P. Protecting against evaluation overfitting in empirical reinforcement learning. In *Proc. 2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning* (eds Sarangapani, J. et. al.) 120–127 (IEEE, 2011).
30. Zhang, C., Vinyals, O., Munos, R. & Bengio, S. A study on overfitting in deep reinforcement learning. Preprint at https://arxiv.org/abs/1804.06893 (2018).
31. Lipinski, C. A., Lombardo, F., Dominy, B. W. & Feeney, P. J. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug Deliv. Rev.* **46**, 3–26 (2001).
32. Rokitskaya, T. I., Luzhkov, V. B., Korshunova, G. A., Tashlitsky, V. N. & Antonenko, Y. N. Effect of methyl and halogen substituents on the transmembrane movement of lipophilic ions. *Phys. Chem. Chem. Phys.* **21**, 23355–23363 (2019).
33. Bickerton, G. R., Paolini, G. V., Besnard, J., Muresan, S. & Hopkins, A. L. Quantifying the chemical beauty of drugs. *Nat. Chem.* **4**, 90–98 (2012).
34. Olivecrona, M., Blaschke, T., Engkvist, O. & Chen, H. Molecular de-novo design through deep reinforcement learning. *J. Cheminf* **9**, 48 (2017).
35. Kusner, M. J., Paige, B. & Hernández-Lobato, J. M. Grammar variational autoencoder. In *Proc. 34th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* Vol. 70 (eds Precup, D. & Teh, Y. W.) 1945–1954 (PMLR, 2017).
36. De Cao, N. & Kipf, T. MolGAN: an implicit generative model for small molecular graphs. In *ICML 2018 Workshop on Theoretical Foundations and Applications of Deep Generative Models* (2018).
37. Zhou, Z., Kearnes, S., Li, L., Zare, R. N. & Riley, P. Optimization of molecules via deep reinforcement learning. *Sci. Rep.* **9**, 10752 (2019).
38. Wainberg, M., Merico, D., Delong, A. & Frey, B. J. Deep learning in biomedicine. *Nat. Biotechnol.* **36**, 829–838 (2018).
39. Kim, S. et al. PubChem in 2021: new data content and improved web interfaces. *Nucleic Acids Res.* **49**, D1388–D1395 (2020).
40. Gao, W. & Coley, C. W. The synthesizability of molecules proposed by generative models. *J. Chem. Inf. Model.* **60**, 5714–5723 (2020).
41. Segler, M. H. S., Preuss, M. & Waller, M. P. Planning chemical syntheses with deep neural networks and symbolic AI. *Nature* **555**, 604–610 (2018).
42. Kishimoto, A., Buesser, B., Chen, B. & Botea, A. Depth-first proof-number search with heuristic edge cost and application to chemical synthesis planning. In *Advances in Neural Information Processing Systems* Vol. 32 (eds Wallach, H. M. et al.) 7224–7234 (Curran Associates, 2019).
43. Stokes, J. M. et al. A deep learning approach to antibiotic discovery. *Cell* **180**, 688–702 (2020).
44. Jumper, J. et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583–589 (2021).
45. Liu, J. & Ning, X. Multi-assay-based compound prioritization via assistance utilization: a machine learning framework. *J. Chem. Inf. Model.* **57**, 484–498 (2017).
46. Liu, J. & Ning, X. Differential compound prioritization via bidirectional selectivity push with power. *J. Chem. Inf. Model.* **57**, 2958–2975 (2017).
47. Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O. & Dahl, G. E. Neural message passing for quantum chemistry. In *Proc. 34th International Conference on Machine Learning* Vol. 70 (eds Precup, D. & Teh, Y. W.) 1263–1272 (PMLR, 2017).
48. Xu, K., Hu, W., Leskovec, J. & Jegelka, S. How powerful are graph neural networks? In *Proc. 7th International Conference on Learning Representations* (OpenReview.net, 2019).
49. Kingma, D. P. & Welling, M. Auto-encoding variational Bayes. In *Proc. 2nd International Conference on Learning Representations* (eds. Bengio, Y. & LeCun, Y.) (OpenReview.net, 2014).

50. Wildman, S. A. & Crippen, G. M. Prediction of physicochemical parameters by atomic contributions. *J. Chem. Inf. Comput. Sci.* **39**, 868–873 (1999).
51. Reddi, S. J., Kale, S. & Kumar, S. On the convergence of Adam and beyond. In *Proc. 6th International Conference on Learning Representations* (OpenReview.net, 2018).
52. Chen, Z. A deep generative model for molecule optimization via one fragment modification. *Zenodo* https://doi.org/10.5281/zenodo.4667928 (2021).

Acknowledgements

This project was made possible, in part, by support from the National Science Foundation grant nos. IIS-1855501 (X.N.), IIS-1827472 (X.N.), IIS-2133650 (X.N. and S.P.) and OAC-2018627 (S.P.), the National Library of Medicine grant nos. 1R01LM012605-01A1 (X.N.) and 1R21LM013678-01 (X.N.), an AWS Machine Learning Research Award (X.N.) and The Ohio State University President's Research Excellence programme (X.N.). Any opinions, findings and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies. We thank X. Wang and X. Cheng for their constructive comments.

Author contributions

X.N. conceived the research. X.N. and S.P. obtained funding for the research and co-supervised Z.C. Z.C., M.R.M., S.P. and X.N. designed the research. Z.C. and X.N. conducted the research, including data curation, formal analysis, methodology design and implementation, result analysis and visualization. Z.C. drafted the original manuscript. M.R.M. provided comments on the original manuscript. Z.C., X.N. and S.P. conducted the manuscript editing and revision. All authors reviewed the final manuscript.

Competing interests

M.R.M. was employed by NEC Labs America. The remaining authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s42256-021-00410-2.

Correspondence and requests for materials should be addressed to Xia Ning.

Peer review information *Nature Machine Intelligence* thanks Michael Withnall and Benjamin Sanchez-Lengeling for their contribution to the peer review of this work.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© The Author(s), under exclusive licence to Springer Nature Limited 2021

Reproduced with permission of copyright owner. Further reproduction prohibited without permission.