MOLER: Incorporate Molecule-Level Reward to Enhance Deep Generative Model for Molecule Optimization

Tianfan Fu, Cao Xiao, Lucas M. Glass, and Jimeng Sun

Abstract—The goal of molecular optimization is to generate molecules similar to a target molecule but with better chemical properties. Deep generative models have shown great success in molecule optimization. However, due to the iterative local generation process of deep generative models, the resulting molecules can significantly deviate from the input in molecular similarity and size, leading to poor chemical properties. The key issue here is that the existing deep generative models restrict their attention on substructure-level generation without considering the entire molecule as a whole. To address this challenge, we propose Molecule-Level Reward functions (MOLER) to encourage (1) the input and the generated molecule to be similar, and to ensure (2) the generated molecule has a similar size to the input. The proposed method can be combined with various deep generative models. Policy gradient technique is introduced to optimize reward-based objectives with small computational overhead. Empirical studies show that MOLER achieves up to 20.2% relative improvement in success rate over the best baseline method on several properties, including QED, DRD2 and LogP.

Index Terms—Automatic molecule optimization, generative models, molecule generation, drug discovery

1 Introduction

Designing molecules with desirable properties is a fundamental task in drug discovery. Traditional methods such as high throughput screening (HTS) tests large compound libraries to identify molecules with desirable properties, which are inefficient and costly [1], [2]. Unlike HTS, *molecule optimization* takes an input molecule X and aims to find molecule Y with improved drug properties such as drug-likeness and biological activity. However, standard molecule optimization is still costly and time-consuming due to the long cycles of labor-intensive synthesis and analysis [3].

The challenges mentioned above motivate a series of works that apply machine learning techniques on molecule optimization, with the ultimate goal to generate new molecules that *maximize desirable drug properties while maintaining similarity* to the original molecule. Existing works mainly can be categorized as generative models [4], [5], [6] and reinforcement learning (RL) methods [7], [8]. *Generative models for molecule optimization* map an input molecule to a latent space and perform optimization in the latent space to search for new molecules. For example, Gómez-Bombarelli *et al.* [6], Blaschke *et al.* [9] utilized SMILES strings as molecule representations to generate molecules; however, they are known to create many invalid molecules. To improve the validity of the generated molecules, various approaches were proposed: Kusner *et al.* [4] and Dai *et al.* [5] designed grammar constraints, while MolGAN [10], CGVAE (Constrained Graph VAE) [11], JTVAE (Junction Tree VAE)-based approaches [12], [13], [14] focus on graph representations of molecules. *Reinforcement learning for molecule optimization* are often developed on top of molecule generators for optimizing desirable properties. For example, Olivecrona *et al.* [15], Putin *et al.* [16], and Popova *et al.* [17] applied RL techniques on top of a string generator to produce SMILES strings. You *et al.* [7], Zhou *et al.* [8] proposed to generate molecular graphs, which achieved perfect validity.

Despite the initial success, existing methods often rely on iterative local expansion to acquire the target molecule, potentially creating molecules of arbitrary sizes. Without a global fitness metric, the generated molecules can significantly deviate from the target molecule in molecular similarity and size. Also, these methods focus on patterns that map on substructures. However, during testing, the evaluation metrics for molecule quality are often based on the entire molecule. These discrepancies cause two challenges. We also empirically demonstrate them using Junction Tree Variational Auto-Encoder (JTVAE) [12] and Variational Junction Tree encoder-decoder (VJTNN) [13] models, as shown in Table 1 and Fig. 1.

- *Tianfan Fu is with the Department of Computer Science and Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA. E-mail: futianfan@gmail.com.*
- *Cao Xiao is with the Analytics Center of Excellence, IQVIA, Cambridge, MA 02139 USA. E-mail: danicaxiao@gmail.com.*
- *Lucas M. Glass is with the Analytics Center of Excellence, IQVIA, Plymouth Meeting, PA 19462 USA, and also with Temple University, Philadelphia, PA 19122 USA. E-mail: Lucas.Glass@iqvia.com.*
- *Jimeng Sun is with the Computer Science Department, University of Illinois, Urbana-Champaign, Champaign, IL 61820 USA. E-mail: jimeng.sun@gmail.com.*

Manuscript received 27 June 2020; revised 18 Nov. 2020; accepted 3 Jan. 2021. Date of publication 21 Jan. 2021; date of current version 6 Oct. 2022.
This work was supported by NSF award SCH-2014438, PPoSS 2028839, IIS-1838042, NIH award R01 1R01NS107291-01 and OSF Healthcare.
(Corresponding author: Tianfan Fu.)
Recommended for acceptance by Z. Cai.
Digital Object Identifier no. 10.1109/TKDE.2021.3052150

TABLE 1
Empirical Results of Two Deep Generative Models on Two Molecule Optimization Tasks, Improving QED and DRD2 Score

Dataset	Method	All Test Set		Small Molecule		Large Molecule	
		Similarity	Property	Similarity	Property	Similarity	Property
QED	JTVAE [12]	0.298	0.804	0.142	0.542	0.231	0.792
	VJTNN [13]	0.311	0.885	0.178	0.602	0.283	0.818
DRD2	JTVAE	0.299	0.709	0.164	0.501	0.243	0.621
	VJTNN	0.315	0.765	0.170	0.579	0.296	0.744

QED and DRD2 are important chemical properties for drug molecules. Small molecule includes the generated molecules with less than eight substructures while large molecule includes the ones generated with more than 18 substructures. "Similarity" is the similarity between input and generated molecule. "Property" (either QED or DRD2) is the average property score of the generated molecules. Both scores are basic metrics for molecule optimization, ranging from 0 to 1, and higher is better. We report the average results on test set. We find that (1) similarity score has more improvement space compared with property scores; (2) performance degradation on either small or large molecules.

- *Difficulty in Maintaining Similarity While Optimizing Drug Properties*. Similarity scores between input and generated molecules (defined in Equation (11)) are relatively low compared with property improvement. For example, when property scores are higher than 0.7 in the range [0,1], the similarity score will only be low (around 0.3 in the range [0,1]), as shown in Table 1.
- *Difficulty in Maintaining Molecule Sizes Which Affect Property Improvement*. When generated molecules are of very different sizes compared to the target molecules, they will have poor performance in both similarity and property. Fig. 1 show the property improvement as a function of the size difference between the generated and input molecules. We can see that a large size difference in either positive or negative direction usually corresponds to smaller property improvement.

To address these challenges, we introduce Molecule-level Rewards (MOLER) to enhance the molecule level properties. MOLER is a general and flexible approach that can be incorporated into various deep generative models for improved performance. MOLER is enabled by the following technical contributions.

- *Similarity Reward* (Section 3.2): We formulate the similarity between input and generated molecules as a reward function to alleviate the *similarity gap* between them.
- *Size deviation penalty* (Section 3.3): To explicitly control the size of molecule, we design a size deviation penalty that penalizes large size deviations to reduce the *size difference* between target and generated molecules.

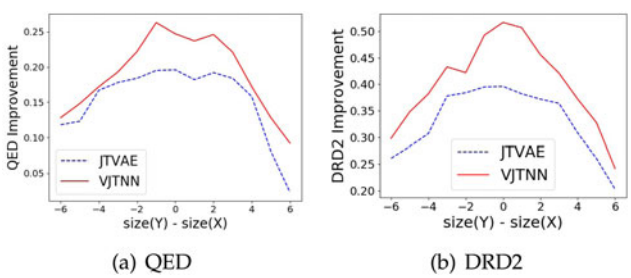


Fig. 1. Property Improvement as a function of size difference between source molecule X and target molecule Y.

We evaluated MOLER by incorporating it into JTVAE [12], VJTNN [13] and CORE [18]. We compared the MOLER-enhanced models with several strong baselines on three real-world molecule optimization tasks. Results show MOLER-enhanced models can achieve up to 16%, 14%, and 20% relatively improvement over the best baselines on similarity score, property improvement, and success rate, respectively.

2 Related Works

We review related works in molecule generation. From the methodology perspective, there are two research lines in automatic molecule generation, namely the maximum likelihood estimation (MLE) via deep generative models (DGM), and the reinforcement learning (RL)-based approaches.

Maximum Likelihood Estimation (MLE) methods include Sequence-to-Sequence (Seq2Seq) models and graph-to-graph ones. The Seq2Seq models focus on maximizing the likelihood of the target sequence and is proven effective in various natural language processing tasks [19]. Since molecule can be also represented as SMILES sequences, it is straightforward to apply the Seq2Seq model. Sequence-based approaches are very brittle since small changes to the sequence may completely violate chemical validity. Thus special constraints were developed to ensure the validity of generated SMILES string [4], [5], [6]. However, due to the limited representation power of SMILES and complex syntactic constraints on SMILES string, these sequence-based approaches still suffer from generating invalid SMILES string. To address this issue, graph-based methods were explored, where graph representation of molecules can capture the structural information of molecules. Specifically, [12] represented molecule graph where nodes are substructures such as rings and atoms, and then applied a two-phase generation procedure to create a scaffolding tree. The generation is to assemble the tree into a new molecular graph. More recently, the graph-to-graph translation model [13] was proposed as an enhancement of [12]. It extends the junction variational autoencoder via attention mechanism and generative adversarial networks. One potential problem for MLE methods often requires paired training data (i.e., input and target molecule pairs), which are not naturally available for molecule optimization.

*Reinforcement Learning (RL) methods*were also adopted in sequence-based molecule generation [15], [20] and graph-

TABLE 2
Notations Used in the Paper

Notations	short explanation
(X, Y)	input-target molecule pair
S	substructure set S (a.k.a. vocabulary)
V/E	set of vertex(atom) / edge(bond)
$G = (V, E)$	molecular graph
$\mathcal{T}_G = (\mathcal{V}, \mathcal{E})$	scaffolding tree of graph G, junction tree [12]
$\mathbb{T}_G$	scaffolding tree of G without substructure, output of topological prediction, as shown in Fig. 2
$N(v)$	set of neighbor nodes of vertex v
$\mathbf{e}_v/\mathbf{e}_{uv}$	feature vector for node v / edge (u, v)
$\mathbf{m}_{uv}^{(t)}$	message for edge (u, v) at the tth iteration
T	Depth of Message Passing Network
$\mathbf{z}_i^G/\mathbf{z}_i^{\mathcal{T}}$	embedding of node i in G / $\mathcal{T}$, $\mathbf{Z}^G = \{\mathbf{z}_1^G, \ldots\}$
$\mathbf{h}_{i_t,j_t}$	message vector for edge (i_t, j_t)
$\mathbf{d}_G$	Embedding of Graph G
p_t^{topo}	topological prediction score at the tth step
$\mathbf{q}_t^{\text{sub}}$	substructure prediction distribution at the tth step
$f_i(\cdot),\ i = 1, \ldots, 6$	parameterized neural networks
$\text{sim}(X, Y) \in [0, 1]$	Similarity between X and Y
$\text{size}(X)$	number of substructure in X
θ	all learnable parameters in generative model
$\pi_\theta(Y\|X)$	generative model parameterized by θ
$g_1(\cdot), g_2(\cdot)$	fully-connected neural network

based generation [7], [8]. The learning objective is based on molecule level reward, e.g., directly introducing chemical property of the generated molecule as reward. However, RL-based methods usually ignore substructure-level supervision and suffer from poor exploration efficiency [21] and are usually hard to train [13], [22].

Between the two, the MLE/DGM methods focus on substructure-level translation and ignore the molecule-level reward [12], [13], [18], [23]. Also, MLE methods train on paired molecules, which are not naturally available for molecule optimization and thus limit the performance. In contrast, for RL methods, the learning objective is based on molecule-level reward, e.g., directly introduce the chemical property of the generated molecule or molecular similarity as reward. RL-based methods usually ignore substructure-level supervision and suffer from poor exploration efficiency [21] and are generally hard to train [13], [22]. To alleviate these issues, our method bridges the two approaches by incorporating two molecule-level rewards into MLE/DGM models.

3 MOLER METHOD

In this section, we will first present deep generative models as background for molecule optimization in Section 3.1, particularly the scaffolding tree[1] based graph generation approaches [12], [13], [14], [18].

1. Such a scaffolding tree is also named as a junction tree in [12], [13].

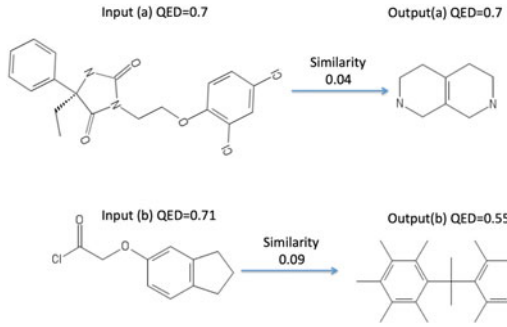


Fig. 2. MOLER Framework. (1) First we describe the generative model. A molecular graph is transformed into a scaffolding tree. Both of them are encoded. Then, tree decoder and graph decoder are applied one after another. During the tree decoder, topological prediction generates the structure of scaffolding tree, while substructure prediction is to predict the substructure for the label of each node in the scaffolding tree. After that, during the graph decoder, scaffolding tree is assembled into molecules. SGD is applied to optimize generative models. (2) Then in MOLER, we use the generative model to generate molecules and evaluate their reward, where the two reward functions (similar reward and size deviation penalty) will assign high reward to the molecule that is (a) similar to input molecule X; (b) close to X in molecular size. Policy gradient is used to optimize the reward.

After introducing the background on scaffolding tree-based generative model and its limitations, we present the two molecule reward functions individually in Sections 3.2 and 3.3 followed by the optimization procedure (Section 3.4). We list some important mathematical notations and their short explanations in Table 2 for clarity, and also show a flow chart in Fig. 2 to illustrate deep generative models and MOLER in an intuitive way.

3.1 Deep Generative Models for Molecule Generation

In this subsection, we summarize the scaffolding tree-based generative model for molecule generation, which achieves the state of the art performance [12], [13], [14], [18]. For ease of exposition, we first define molecular graph, substructure and scaffolding tree. Note that this subsection provides the background to understand these types of generative models, which were originally proposed in [12] then subsequently enhanced by [13], [14], [18]. The main contribution of this paper will be discussed in next subsection.

Definition 1 (Molecular Graph G). *Molecular graph G represents the graph structure for a molecule. In molecule graph G, each node corresponds to an atom of a molecule, and each edge corresponds to a bond connecting atoms.*

Definition 2 (Substructure). *The set of substructures includes all possible rings, bonds, and atoms. A substructure is a basic unit in the scaffolding tree. Each substructure corresponds to a node in the scaffolding tree.*

Definition 3 (Scaffolding Tree $\mathcal{T}_G$). *Given a molecule and its corresponding molecular graph G, the scaffolding tree $\mathcal{T}$ is generated by partitioning the graph G into substructures and connecting substructures into a tree. The main purpose of using scaffolding trees is to simplify the generation procedure. The scaffolding tree $\mathcal{T}_G$ is the skeleton of the molecular graph.*

These models for generating valid molecules comprise of two parts: (1) the encoder-decoder for scaffolding tree $\mathcal{T}_G$, and 2) the encoder-decoder for molecular graph G.

For scaffolding tree-based graph generation methods, the training set consists of paired data (X, Y). The input is molecule X while the target is another molecule Y similar to X but with better chemical property, e.g., biological activity (measured by DRD) [24], drug-likeness (measured by QED) [25]. The neural architecture can be divided into two parts: (i) encoder and (ii) decoder. Both encoder and decoder have molecular graph phase and scaffolding tree phase.

3.1.1 Encoder

The goal of the encoder is to yield an embedding vector for each node in the scaffolding tree $\mathcal{T}_G$ or the input molecular graph G. Specifically, both input molecular graphs and scaffolding trees are encoded via graph Message Passing Networks (MPN) [12], [26]. It encodes both nodes and edges in a graph. First, on the node level, each node v has a feature vector denoted $\mathbf{e}_v$. For example, node v in a molecular graph G is an atom, $\mathbf{e}_v$ includes the atom type, valence, and other atomic properties. In the corresponding scaffolding tree $\mathcal{T}_G$, node v refers to a substructure, then $\mathbf{e}_v$ is a one-hot vector indicating the substructure's index. On the edge level, $\mathbf{e}_{uv}$ is the feature vector for edge $(u, v) \in E$. $N(u)$ represents the set of all the neighbor nodes of the node u. $\mathbf{m}_{uv}$ and $\mathbf{m}_{vu}$ are the hidden variables representing the message from node u to node v and vice versa. They are iteratively updated as

$$\mathbf{m}_{uv}^{(t)} = f_1\big(\mathbf{e}_u, \mathbf{e}_{uv}, \sum_{w \in N(u) \setminus v} \mathbf{m}_{wu}^{(t-1)}\big), \quad t = 1, \ldots, T, \tag{1}$$

where $f_1(\cdot)$ is a fully-connected neural network, $\mathbf{m}_{uv}^{(t)}$ is the message vector from node u to node v at the tth iteration, whose initialization is all-0 vector, i.e., $\mathbf{m}_{uv}^{(0)} = \mathbf{0}$, following the rule of thumb [26]. After T steps of iteration,[2] another fully-connected neural network $f_2(\cdot)$ is used to aggregate these messages. Each vertex has a latent vector as

$$\mathbf{z}_u = f_2\big(\mathbf{e}_u, \sum_{v \in N(u)} \mathbf{m}_{vu}^{(T)}\big). \tag{2}$$

In a nutshell, the encoder module yields embedding vectors for nodes in graph G and scaffolding tree $\mathcal{T}_G$, denoted $\mathbf{Z}^G = \{\mathbf{z}_1^G, \mathbf{z}_2^G, \ldots\}$ and $\mathbf{Z}^{\mathcal{T}_G} = \{\mathbf{z}_1^{\mathcal{T}_G}, \mathbf{z}_2^{\mathcal{T}_G}, \ldots\}$, respectively.

3.1.2 Decoder

Like encoder, molecule generation can be also divided into two parts: A. (scaffolding) tree decoder; B. (molecular) graph decoder. The goal of the scaffolding tree decoder is to generate a new scaffolding tree. Then graph decoder converts the scaffolding tree into the correct molecular graph.

*A. Tree Decoder.*To generate a new scaffolding tree, the idea is to generate one substructure at a time from an empty tree. In each step, first it needs to decide/predict whether to expand the current node or backtrack to its parent (*topological prediction*). If the prediction is to expand, (*substructure prediction*) is leveraged to predict which substructure to add to the new node, as shown in Fig. 2. When it backtracks to the starting node and decide not to expand any more, the generation process terminates automatically.

Topological Prediction. The core idea is first leveraging tree-based Recurrent Neural Network (tree-RNN) [12] to enhance the embedding for node i_t, then predict whether to expand or backtrack. Given scaffolding tree $\mathcal{T}_G = (\mathcal{V}, \mathcal{E})$, the tree decoder uses the tree-based RNN with attention mechanism to further improve embedding information learned from the original message-passing embeddings $\mathbf{Z}^{\mathcal{T}}$. The tree-RNN converts graph into a sequence of nodes and edges via depth-first search. Specifically, let $\tilde{\mathcal{E}} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\}$ be the edges traversed in depth first search, each edge is visited twice in both directions, so we have $m = |\tilde{\mathcal{E}}| = 2|\mathcal{E}|$. Suppose $\tilde{\mathcal{E}}_t$ is the first t edges in $\tilde{\mathcal{E}}$, message vector $\mathbf{h}_{i_t,j_t}$ is updated as

$$\mathbf{h}_{i_t,j_t} = \mathrm{GRU}(\mathbf{e}_{i_t}, \{\mathbf{h}_{k,i_t}\}_{(k,i_t) \in \tilde{\mathcal{E}}_t, k \neq j_t}). \tag{3}$$

The expanding probability at node i_t is computed by combining the embeddings $\mathbf{Z}^{\mathcal{T}}$, $\mathbf{Z}^G$ and the current state $\mathbf{e}_{i_t}$, $\sum_{(k,i_t) \in \tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}$ using a neural network $f_3(\cdot)$

$$p_t^{\text{topo}} = f_3(\mathbf{e}_{i_t}, \sum_{(k,i_t) \in \tilde{\mathcal{E}}_t} \mathbf{h}_{k,i_t}, \mathbf{Z}^{\mathcal{T}}, \mathbf{Z}^G), \quad \text{where } t = 1, \ldots, m. \tag{4}$$

Specifically, we first compute context vector $\mathbf{c}_t^{\text{topo}}$ using attention mechanism (similar to the process in Equations (5) and (6)), then concatenate $\mathbf{c}_t^{\text{topo}}$ and $\mathbf{e}_{i_t}$, and feed into a fully-connnected neural network with sigmoid function to produce the expanding probability.

Substructure Prediction. After the decision of node expansion, we focus on finding what substructures to add by either selecting from the global set of substructures [12], [13] or copying from original input [18]. Every time after expanding a new node, the model predicts the substructure from the vocabulary.

We will summarize the key steps next. First we use attention mechanism to compute context vector based on current message vector $\mathbf{h}_{i_t,j_t}$ and node embedding $\mathbf{Z}^{\mathcal{T}}, \mathbf{Z}^G$

$$\mathbf{c}_t^{\text{sub}} = \mathrm{Attention}(\mathbf{h}_{i_t,j_t}, \mathbf{Z}^{\mathcal{T}}, \mathbf{Z}^G), \tag{5}$$

Specifically, the attention weight is computed as

$$\begin{aligned} \alpha_j^{\mathcal{T}} &= f_4(\mathbf{h}_{k,i_t}, \mathbf{z}_j^{\mathcal{T}}), \quad \alpha_i^{\mathcal{T}} \in \mathbb{R}, \\ [\alpha_1^{\mathcal{T}}, \alpha_2^{\mathcal{T}}, \ldots] &= \mathrm{Softmax}([\alpha_1^{\mathcal{T}}, \alpha_2^{\mathcal{T}}, \ldots]), \end{aligned} \tag{6}$$

where $f_4(\cdot)$ is the dot-product function [28]. $\{\alpha^G\}$ are generated in the same way. Then we concatenate tree-level and graph-level context vector to generate the context vector

$$\mathbf{c}_t^{\text{sub}} = \Big[\sum_i \alpha_i^{\mathcal{T}} \mathbf{z}_i^{\mathcal{T}}, \sum_j \alpha_j^G \mathbf{z}_j^G\Big]. \tag{7}$$

Then based on attention vector $\mathbf{c}_t^{\text{sub}}$ and message vector $\mathbf{h}_{i_t,j_t}$, $f_5(\cdot)$, a fully-connected neural network with softmax activation, is added to predict the substructure

$$\mathbf{q}_t^{\text{sub}} = f_5(\mathbf{h}_{i_t,j_t}, \mathbf{c}_t^{\text{sub}}), \tag{8}$$

2. T is a hyperparameter, similar to that in Graph Convolutional Network [27].

where $\mathbf{q}_t^{\text{sub}}$ is a probability distribution over all substructures [12], [13]. In [18], $\mathbf{q}_t^{\text{sub}}$ is enhanced by a COPY strategy that copies substructures from the input molecule using attention weight in Equation (6).

B. Graph Decoder. Based on the scaffolding tree generated by the tree decoder, graph decoder is used to convert nodes in the scaffolding tree together into the correct molecular graph [12]. During the learning procedure, we enumerate all the possible molecular structures $\{G_i\}$. Then we formulate it as a multi-classification problem, where the learning objective is the log-likelihood of the right structure G_o

$$\mathcal{L}_g = s^a(G_o) - \log \sum_{G_i} \exp(s^a(G_i)), \tag{9}$$

where $s^a(G_i) = \mathbf{h}_{G_i}^\top \mathbf{d}_{G_o}$ is a scoring function that measures the likelihood of the current structure G_i; $\mathbf{d}_{G_o}$ is the embedding of the original graph G_o.

Limitation and Extensions. The problem of these generative models is that they only focus on the local (substructure level) mapping while ignore the global (molecule level) metric, which may lead a significant deviation from input molecule in similarity. Next, we will present RL-based reward functions to enhance the overall loss function towards more desirable molecules.

3.2 Similarity Reward

Both similarity with input molecule X and property of Y (denoted $\text{sim}(X, Y)$) are essential metrics to evaluate the quality of generated Y. The similarity between two molecules ranges from 0 to 1. As mentioned, the similarity value is relatively low (around 0.3 for both QED and DRD2 dataset) compared with property value (approximately 0.7-0.8) in numerical value. We consider adding "similarity reward" to explicitly enhance the similarity constraint, i.e., maximize

$$\mathcal{L}_{\text{sim}}(\theta) = \mathbb{E}_{Y \sim \pi_\theta(\cdot|X)}\big[\text{sim}(X, Y)\big], \tag{10}$$

where θ represents all the parameters for generative models; $\mathcal{L}_{\text{sim}}(\theta)$ is objective function for similarity reward; hyperparameter $w_{\text{sim}} \in \mathbb{R}_+$ is weight of the reward. $\text{sim}(X, Y)$ represents Tanimoto similarity between molecule X and Y over Morgan fingerprints [29]. Morgan fingerprints is a binary vector where each bit records the presence ("1") or absence ("0") of a binary fragment descriptor in the molecule. For example, a fragment descriptor can be "if Benzene ring exists in the molecule". It is assumed that two fingerprints with many bits in common represent similar molecules. Let $\mathcal{S}_X$ and $\mathcal{S}_Y$ denote the set of fragment descriptors that exist in molecule X and Y, respectively (i.e., presence of fragment descriptor). Tanimoto similarity is defined as

$$\text{sim}(X, Y) = \frac{|\mathcal{S}_X \cap \mathcal{S}_Y|}{|\mathcal{S}_X \cup \mathcal{S}_Y|} \in [0, 1], \tag{11}$$

where $\cap, \cup$ represent the intersection and union of two binary vectors respectively; $|\cdot|$ denotes the cardinality of a set. The value ranges from 0 to 1. Higher value means more similarity. We use Rdkit package [30] for both generations of Morgan fingerprints and computation of Tanimoto similarity. When optimizing Equation (10), the computation of its gradient estimator requires the numerical value of the probability density function $\pi_\theta(Y|X)$ explicitly. Now we discuss how to evaluate $\pi_\theta(Y|X)$ explicitly.

We know from Section 3.1, the molecule generation is mainly divided into three prediction tasks: (i) topological prediction, which is a binary classification task; (ii) substructure prediction; (iii)Assembling prediction (in graph decoder). Since there are two phases (scaffolding tree and molecular graph) in graph generation, $\pi_\theta(Y|X)$ can be written as the following joint distribution that incorporate the generation of both the scaffolding tree and the molecular graph, which includes three prediction tasks,

$$\begin{aligned}&\pi_\theta(Y|X)\\ &= \underbrace{\prod_{t=1}^{2|\mathcal{E}|} p_t^{\text{topo}}}_{\text{topological prediction}} \cdot \underbrace{\prod_{t \in \mathcal{S}} \mathbf{q}_t^{\text{sub}}}_{\text{substructure prediction}} \cdot \underbrace{\frac{\exp(s^a(G_Y))}{\sum_{G_i} \exp(s^a(G_i))}}_{\text{assembling prediction}}.\end{aligned} \tag{12}$$

First, p_t^{topo} is the probability for the tth topological prediction, defined in Equation (4). $\mathcal{E}$ is edge set of the generated scaffolding tree. Since the generation procedure would terminate until backtracking to root node, each edge is visited twice and there are totally $2|\mathcal{E}|$ topological predictions. Second, regarding substructure prediction, $\mathbf{q}_t^{\text{sub}}$ is a distribution over all substructures defined Equation (8), $\mathcal{S} \subseteq \{1, 2, \ldots, 2|\mathcal{E}|\}$ represents the set of the indexes of the edges who expands to a new node in a scaffolding tree, because only when topological prediction p_t^{topo} predict to expand to a new node, we need to make substructure prediction. Third, regarding assembling prediction, G_Y is the assembling structure for Y, selected from all the possible molecular structure $\{G_i\}$ based on the scaffolding tree, $s^a(G_i) = \mathbf{h}_{G_i} \cdot \mathbf{d}_{G_o}$ is a scoring function defined in Equation (9).

3.3 Size Deviation Penalty

Size deviation between input and target molecules is another reason that the target molecule is dissimilar to the input. Therefore, we design a penalty score to constrain this deviation.

As mentioned in Section 3.1, in graph generation there are three key prediction tasks: (i) topological prediction; (ii) substructure prediction; (iii) assembling prediction. Since scaffolding tree-based approaches use substructure as the basic component in molecule generation, we use the number of substructure as the surrogate for molecule size, which is much more efficient to compute since it is only related to topological prediction.

To design a reasonable size deviation penalty, we empirically investigate the correlation between the size of X and Y in training data pairs, and find they are positively correlated. We want to minimize the following size deviation penalty,

$$\mathcal{L}_{\text{size}}(\theta) = \mathbb{E}_{\pi_\theta^{\mathbb{T}}(\mathbb{T}_Y|X)}\Big[g\big(\text{size}(\mathbb{T}_X), \text{size}(\mathbb{T}_Y)\big)\Big], \tag{13}$$

where $\mathcal{L}_{\text{size}}(\theta)$ is objective function of size deviation penalty, hyperparameter $w_{\text{size}} \in \mathbb{R}_+$ is weight for size deviation penalty, $\text{size}(X)$ denotes the number of substructure in scaffolding tree of X. $g(\cdot, \cdot)$ is the reward function of molecule size

defined as

$$g(x,y) = \begin{cases} |x-y| - \epsilon, & |x-y| > \epsilon, \\ 0, & |x-y| \leq \epsilon, \end{cases} \tag{14}$$

where $\epsilon \in \mathbb{N}_+$ is a positive integer. We set $\epsilon = 3$, which is empirically validated. Note that $g(\cdot,\cdot)$ is commonly known as ϵ-insensitive loss function in the context of SVM regression [31]. The intuition behind the design of $g(x,y)$ is to give a high penalty when the size of the input and generated molecule differs significantly. When minimizing $\mathcal{L}_{\text{size}}(\theta)$, we need to evaluate $\pi_\theta^{\mathbb{T}}$ explicitly, which is a joint distribution for all topological prediction,

$$\pi_\theta^{\mathbb{T}}(Y|X) = \prod_{t=1}^{2|\mathcal{E}|} p_t^{\text{topo}}. \tag{15}$$

3.4 Learning and Optimization

Now we discuss the optimization procedure. Without loss of generalization, we consider maximizing a general objective as follows,

$$\mathcal{L}(\theta) = \mathbb{E}_{Y\sim\pi_\theta(\cdot|X)}\big[R(X,Y)\big], \tag{16}$$

where $\mathcal{L}(\theta)$ can be either $\mathcal{L}_{\text{sim}}$ or $\mathcal{L}_{\text{size}}$ in this paper, reward function $R(X,Y) \in \mathbb{R}$ can be either $\text{sim}(X,Y)$ or $g(\text{size}(X), \text{size}(Y))$. $R(X,Y)$ usually doesn't involve θ, so it's seen as a constant if we optimize w.r.t. θ. $\pi_\theta(Y|X) \in \mathbb{R}_+$ corresponds the probability density function for generative model. $\nabla_\theta \mathcal{L}(\theta)$, the gradient of objective function with regard to θ, can be expanded as

$$\begin{aligned} \nabla_\theta \mathcal{L}(\theta) &= \nabla_\theta \int \pi_\theta(Y|X) R(X,Y) dY \\ &= \int \nabla_\theta \pi_\theta(Y|X) R(X,Y) dY \\ &= \int \pi_\theta(Y|X) \nabla_\theta \log \pi_\theta(Y|X) R(X,Y) dY \\ &= \mathbb{E}_{Y\sim\pi_\theta(\cdot|X)}\Big[\nabla_\theta \log \pi_\theta(Y|X) R(X,Y)\Big], \end{aligned} \tag{17}$$

where in the third equality, "log-derivative trick" is applied. It is a well-known technique in policy gradient-based reinforcement learning [32], [33] and stochastic variational inference [34]. Then, the unbiased estimator for gradient of $\mathcal{L}(\theta)$ with regard to parameters θ can be obtained using Monte Carlo samples,

$$\nabla_\theta \mathcal{L}(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log \pi_\theta(\tilde{Y_i}|X_i) R(X_i, \tilde{Y_i}), \tag{18}$$

where $\tilde{Y_i} \sim \pi_\theta(\cdot|X_i)$. That is, $\tilde{Y_i}$ is a molecule generated by generative model given the input molecule X_i. Then gradient ascent is used to maximize the objective function with regard to θ. Worth to mention that the proposed method does not require extra parameters for the model compared with the deep generative models.

In summary, the deep generative model and two reward-based objectives are optimized alternatively and individually. Stopping criteria is checked every epoch on the validation set. When the success rate (a metric, which would be discussed in Section 4.5) of the current epoch is lower than the previous epoch, we terminate the learning procedure. We show the complete algorithm in Algorithm 1, which has more details.

Algorithm 1. `MOLER`

```
# training
while Convergence criteria is not met do
    Sample a minibatch M = {(X_1, Y_1), ..., (X_m, Y_m)}, m = |M|.
    Optimize L_gen(θ) w.r.t. θ using {(X_i, Y_i)}_{i=1}^m using SGD, where L_gen(θ) is loss of generative models.
    Generate molecule via Ỹ_i ~ π_θ(·|X_i) for i = 1, ..., m.
    Maximize L_sim(θ) + L_size(θ) w.r.t. θ using {(X_i, Ỹ_i)}_{i=1}^m based on gradient estimator in Equation (18).
end while
# test, e.g., QED task
for X_i ∈ Test Set do
    generate Y_i ~ π_θ(·|X_i).
    Evaluate and record sim(X_i, Y_i) and QED(Y) − QED(X)
end for
Evaluate average similarity, property improvement and success rate on the whole test set.
```

3.5 Choosing Weight for `MOLER`

During learning procedure, two reward values (related to similarity and size) are integrated in the learning objective in order to minimize

$$\mathcal{L} = \mathcal{L}_{\text{gen}} - w_{\text{sim}} \mathcal{L}_{\text{sim}} + w_{\text{size}} \mathcal{L}_{\text{size}}, \tag{19}$$

where $w_{\text{sim}}, w_{\text{size}} \in \mathbb{R}_+$ are hyperparameters that control the strength of `MOLER`, $\mathcal{L}_{\text{gen}}$ is loss of generative model . It is time-consuming to use grid search/random search to find the near-optimal weight combination $(w_{\text{sim}}, w_{\text{size}})$. To address this issue, we resort to Gaussian process (GP) [35], [36]. Gaussian process is used to approximate the validation accuracy as a function of these weight combination. The validation accuracy can be success rate, which is described in Section 4.5) Then via searching for the optimum of the approximated function, we obtain the appropriate weight combinations. Concretely, we denote the function to approximate as $h(w_1, w_2)$, where $w_1 = w_{\text{sim}}, w_2 = w_{\text{size}}$. The search range for w_i, $i = 1, 2$ is bounded by $\text{LB}_i \leq w_i \leq \text{UB}_i$. We draw m weight combinations, denoted $\mathbf{w}^{(1)} = (w_1^1, w_2^1); \mathbf{w}^{(2)} = (w_1^2, w_2^2); \ldots; \mathbf{w}^{(m)} = (w_1^m, w_2^m)$. For the ith weight combination $\mathbf{w}^{(i)}$, we evaluate the task metric (e.g., success rate) on validation set r^i, which can be seen as a noisy evaluation of the true function $h(w_1^i, w_2^i)$ we want to approximate. The noise comes from sampling bias of validation set. The true objective function $h(w_1, w_2)$ is unknown, a zero-mean Gaussian prior is specified,

$$h(\cdot) \sim \text{GP}(\cdot, k(\cdot,\cdot)), \tag{20}$$

where $k(\cdot,\cdot)$ is the covariance function. Given m points $\{\mathbf{w}^{(i)}\}_{i=1}^m$ (weight combination) and their evaluation $\{r^i\}_{i=1}^m$. We assume r is a noisy evaluation of the true unknown function that we are interested, i.e., $r(\mathbf{w}) = h(\mathbf{w}) + \epsilon$, $\mathbf{w} = (w_1, w_2)$. According to Bayes rule, the posterior distribution

of the true objective function is

$$\begin{aligned} &h|\{\mathbf{w}^{(i)}, r^i\}_{i=1}^m \sim \mathcal{N}(\mu(\mathbf{w}), \sigma^2(\mathbf{w})), \\ &\mu(\mathbf{w}) = \mathbf{k}^\top (\mathbf{K} + \sigma^2 I)^{-1} r, \\ &\sigma(\mathbf{w}) = k(\mathbf{w}, \mathbf{w}) - \mathbf{k}^\top (\mathbf{K} + \sigma_n^2 I)^{-1} k, \end{aligned} \tag{21}$$

where

$$\begin{aligned} &\mathbf{K} = \begin{bmatrix} k(\mathbf{w}_1, \mathbf{w}_1) & \cdots & k(\mathbf{w}_1, \mathbf{w}_m) \\ \vdots & \ddots & \vdots \\ k(\mathbf{w}_m, \mathbf{w}_1) & \cdots & k(\mathbf{w}_m, \mathbf{w}_m) \end{bmatrix}, \\ &\mathbf{k} = [k(\mathbf{w}, \mathbf{w}_1), \ldots, k(\mathbf{w}, \mathbf{w}_m)]^\top, \\ &r = [r_1, \ldots, r_m]^\top. \end{aligned} \tag{22}$$

Regarding covariance function we set $k(\mathbf{w}, \mathbf{v}) = \exp\big(-\frac{1}{2}(\mathbf{w}-\mathbf{v})^\top \Sigma^{-1}(\mathbf{w}-\mathbf{v})\big)$, where $\Sigma = \mathrm{diag}\big(\alpha(\mathrm{UB}_1 - \mathrm{LB}_1)^2, \alpha(\mathrm{UB}_2 - \mathrm{LB}_2)^2\big)$. α is empirically set to 0.2 [36].

To summarize, Gaussian process provides a surrogate function to approximate the true objective $h(w_1, w_2)$, the validation accuracy as a function of weights in MOLER (w_{sim} and w_{size}). The surrogate can be used to search, efficiently, for the optimum of the objective function, i.e., optimal weights in MOLER. Grid search is leveraged here to explore the surrogate and get the optimal weights.

4 Experiment

The experiment section answers the following questions.

Q1: Can MOLER improve deep generative models for molecular optimization?

Q2: What is the effect of similarity reward alone?

Q3: What is the effect of size deviation penalty alone?

4.1 Molecular Properties

We are usually interested in some desired chemical properties of molecule in drug discovery, which are natural metrics for evaluation. Following [7], [8], [13], [14], [23], [37], we also focus on the following molecular properties:

- *Quantitative Estimate of Drug-likeness (QED)*. QED is an indicator of drug-likeness [25], and the QED score of a compound ranges from 0 to 1.
- *Dopamine Receptor (DRD2)*. DRD2 score measures a molecule's biological activity against a biological target named the dopamine type 2 receptor (DRD2) [24]. DRD2 score ranges from 0 to 1, and a high score means better property.
- *Penalized LogP*. Penalized LogP is a logP score that also accounts for ring size and synthetic accessibility [38]. LogP score ranges from $-\infty$ to $+\infty$. For most molecules, the score ranges from -20 to 20.

For any chemically valid molecule, QED, DRD2, and LogP scores can be evaluated via Rdkit package [30]. For all these three scores, a higher score is better. Thus, for the training data pairs (X, Y), X is the molecule with lower scores, while Y is a paraphrase of X with a higher score.

4.2 Setup

First, we describe the experimental setup, including the chemical properties of molecules and the generation of training data. The molecular properties include Drug likeness (QED), Dopamine Receptor (DRD2), and Penalized LogP.

TABLE 3
Statistics of all the Three Datasets: DRD2, QED, LogP

Dataset	# Training Pairs	# Valid Pairs	# Test	Vocab Size	Avg # Substructures	η_1
QED	84,306	4,000	800	780	14.99	0.4
DRD2	32,404	2,000	1,000	967	13.85	0.4
LogP	94909	5,000	800	785	14.30	0.4

Here vocabulary is the set of all substructures. "Avg # Substructures" is the average number of substructures per molecule. η_1 is the threshold for similarity constraint defined in Equation (23). QED, DRD2, LogP are not only the chemical properties in this paper. They are also used to name the datasets, e.g., QED dataset is used to learn a model for improving the QED score of a molecule.

Dataset.Data pairs of input and target molecules are extracted from the ZINC dataset. ZINC contains 250K drug molecules, which are extracted from the ZINC database.[3]

We use 3 public datasets publicly available in [13], including (1) QED dataset for improving QED; (2) DRD2 dataset for improving DRD2; (3) LogP dataset for improving Penalized LogP. In this paper, QED, DRD2, LogP are both the target chemical properties to improve and the name of datasets, e.g., QED dataset is used to learn a model for improving the QED score of a molecule. Some basic statistics of all the dataset are listed in Table 3.

4.3 Dataset Construction

The training datasets contain data pairs, which are extracted from ZINC dataset[4] based on following two criteria:

- *Simlarity Constraint*. X and Y are similar enough, i.e.,

$$\mathrm{sim}(X, Y) \geq \eta_1. \tag{23}$$

  Following [13], [23], [37], for all the three datasets, $\eta_1 = 0.4$.
- *Property Constraint*. Y has a higher score than X on a certain property. For QED dataset, QED score of X are in the range of [0.7,0.8] while QED score of Y is in the range of [0.9,1]. For DRD2 dataset, a well-trained model in [40] is applied to assess the probability of biological activity, where probability of X is lower than 0.05 and the probability for Y is greater than 0.5. The property constraint for LogP dataset is described in [13], [39]. The settings are the same as [13].

4.4 Baseline Methods

Now we briefly introduce the baseline methods.

- *JTVAE* (Junction Tree Variational Auto-Encoder) [12].
- *VJTNN* (Variational Junction Tree Encoder-Decoder, also called graph-to-graph translation) [13].
- *CORE* (VJTNN with Copy and Refine Strategy) [18].

3. ZINC database (https://zinc.docking.org) is a curated collection of commercially available chemical compounds prepared especially for virtual screening [39].

4. ZINC database is publicly available at https://zinc.docking.org.

- *GCPN* (Graph Convolutional Policy Network) [7]. GCPN is state-of-the-art reinforcement learning-based method.
- *ReLeaSE*. Reinforcement Learning for Structural Evolution (ReLeaSE) [20] exhibits state-of-the-art performance on SMILES representation.

MOLER can be applied to enhance deep generative models such as JTVAE, VJTNN, and CORE.

4.5 Evaluation Metrics

Next, we provide some key metrics when evaluating the effectiveness of the generated molecules. The task is to generate Y, a paraphrase molecule of X, with better desired property. Here are the evaluation metrics used in our experiments:

- *Similarity*. We evaluate the molecular similarity between the input molecule and the generated molecule, measured by Tanimoto similarity over Morgan fingerprints [29], defined in Equation (11). Similarity between X and Y (denoted $\mathrm{sim}(X, Y)$) ranges from 0 to 1. Morgan fingerprint generation and Tanimoto similarity calculation can be done by Rdkit package [30].
- *Property of generated molecules*, i.e., $\mathrm{Property}(Y)$, includes QED-score, DRD2-score, and LogP-score, evaluated using Rdkit package [30] or chemical rule.
- *Success Rate (SR)* considers both similarity constraint and property improvement. Following [13], [14], [23], we design a criteria to judge whether it satisfied these two aspects: (a) Input and generated molecules are similar enough, $\mathrm{sim}(X, Y) \geq \lambda_1$; (b) improvement are big enough, i.e., $\mathrm{property}(Y) - \mathrm{property}(X) \geq \lambda_2$ or $\mathrm{property}(Y) \geq \lambda_3$. The selection of λ_1 and λ_2 (or λ_3) depend on datasets. Following [13], [14], [37], [41], for QED and DRD2 dataset, when the similarity between input and generated molecule is greater than 0.3 and chemical property (QED, DRD2) of generated molecule is greater than 0.6, we regard the target molecule as a success. For LogP dataset, when the similarity between input and generated molecule is greater than 0.4 and LogP improvement is greater than 0.8, we regard the target as a success.
- *Novelty* is defined as the percentage of generated molecules that does not appear in training set. We also need to do canonical operation to convert the generated molecule into canonical SMILES.
- *Run time & Model Size*. We also report training run time in hours and the model size. Worth to mention that during the inference (generating molecule) procedure, MOLER do not require extra computation, so inference time is about the same as the generative model baselines. MOLER does not require extra parameters compared with original generative models, so JTVAE + MOLER and JTVAE have the same model size, VJTNN + MOLER and VJTNN have the same size, CORE + MOLER and CORE have the same model size.

4.6 Experimental Details for Reproducibility

In this section, we provide the implementation details required for reproduction of experimental results.

Features. First we discuss the features used as input for encoders, including both node and edge features. Following [12], [13], [18], in the (molecular) graph message passing network, the initial atom features include its atom type, degree, its formal charge and its chiral configuration. Bond feature is a concatenation of its bond type, whether the bond is in a ring, and its cis-trans configuration. In the scaffolding tree-based encoder, we represent each substructure with a neural embedding vector, which is similar to word embedding. Both tree and graph decoder leverage the same feature setting as encoders.

Hyperparameter. We follow most of the hyperparameter setting of JTVAE [12], VJTNN [13] and CORE [18] when running JTVAE+MOLER, VJTNN+MOLER and CORE +MOLER, respectively. For all these baseline methods and datasets, the maximal epoch number is set to 10; the batch size is set to 32. The initial learning rate is set to $1e^{-3}$ with the Adam optimizer. Every epoch learning rate is annealed by 0.8. Convergence criteria is checked every epoch measured by success rate on the validation set. When the success rate of the current epoch is lower than the previous epoch, we claim that the model converges and terminate the learning procedure. Following VJTNN and CORE, the discriminator in adversarial training is a three-layer fully-connected network with latent dimension 300 and LeakyReLU function is leverage as activation. For JTVAE, VJTNN, CORE, and their MOLER variants, the depth of tree encoder and graph encoder are set to 6 and 3, respectively. The dimension of hidden state for both (molecular) graph message passing network (GMPN) and junction tree message passing network (JTMPN) are set to 300. All the MPN uses tanh activation and mean function as readout function. In decoder network, all the hidden layer uses ReLU function as activation. MOLER does not bring extra learnable parameters. According to empirical study, the weight of individual MOLER plays a particularly important role in performance. It's regarded as a gradient-free optimization problem, we resort to Gaussian Process [35], as described in Section 3.5. Regarding both similarity reward and size deviation penalty, the search space of weight ranges from $1e^{-4}$ to $1e^{-2}$.

Hardware and Software Configuration. All the experiments are run on an Intel Xeon E5-2690 machine with 256G RAM and 8 NVIDIA Pascal Titan X GPUs. Our method is implemented by Python 2.7 and Pytorch 0.4.0. RDKit version has to be later than 2017.09.

4.7 Q1: MOLER can Improve Deep Generative Models

First, we demonstrate that MOLER can improve the state-of-the-art generative model methods on all the three datasets. The results (in terms of similarity, property improvement, and success rate) are reported in Table 4. We found that MOLER variants (JTVAE+MOLER, VJTNN +MOLER, CORE+MOLER) provides significant and consistent improvement across different generative models up to 19%. For other metrics, in Fig. 3, we use bar charts to show the results of various methods on various datasets. MOLER along with other methods can maintain high novelty in the generated molecules in Fig. 3d; MOLER will

TABLE 4
MOLER-Based Methods Outperform Deep Generative Methods (JTVAE [12], VJTNN [13] and CORE [18]) and RL Methods (GCPN [7], ReLeaSE [20])

Method	Similarity			Property Improvement			Success Rate (%)		
	QED	DRD2	LogP	QED	DRD2	LogP	QED	DRD2	LogP
GCPN	0.308	0.309	0.360	0.877	0.745	3.041	47.71	44.81	55.43
ReLeaSE	0.293	0.284	0.302	0.783	0.643	2.465	38.02	31.21	32.38
JTVAE	0.299	0.300	0.285	0.791	0.693	2.532	38.74	35.43	38.43
JTVAE+MOLER	0.302	0.314	0.315	0.858	0.732	3.015	43.20	40.01	45.24
improvement	+1.07%	+4.77%	+10.41%	+8.47%	+5.63%	+19.08%	+11.51%	+12.93%	+17.72%
VJTNN	0.315	0.316	0.358	0.886	0.764	2.972	48.16	45.38	55.15
VJTNN+MOLER	0.351	0.334	0.372	0.904	0.778	3.182	56.32	47.39	57.01
improvement	+11.61%	+5.56%	+3.94%	+2.03%	+1.83%	+7.07%	+16.94%	+4.43%	+3.37%
CORE	0.321	0.333	0.369	0.895	0.769	3.100	50.26	47.91	57.64
CORE+MOLER	0.360	0.352	0.371	0.910	0.782	3.199	57.32	49.47	57.93
improvement	+12.11%	+5.58%	+.41%	+1.68%	+1.69%	+3.19%	+14.05%	+3.26%	+.50%

MOLER-based methods achieved the best performance in all settings, especially CORE+MOLER, which seems the most competitive. MOLER provides significant and consistent improvement across different generative models up to 19%. In each column, we highlight the best performance using bold font.

not cost many extra computational costs in training time and model size as shown in Figs. 3e and f; Regarding running time, we observe that JTVAE-based methods (JTVAE and JTVAE+MOLER) takes a longer time to converge when optimizing LogP score. There are mainly three reasons. (i) When we use JTVAE+MOLER to optimize LogP, we observe during the early iterations, the generated molecules differ a lot from the target ones, the MOLER objective is not stable. Thus it costs more time. (ii) LogP is more sensitive to the local structure [38] compared QED and DRD2 and hard to optimize. We observe that all the methods cost a longer time when optimizing LogP. (3) At the same time, compared with VJTNN, the neural architecture of JTVAE is less finer-grained. Specifically, it does not have adversarial learning module and attention mechanism in the decoder, which may cause a longer time to converge. Finally, as shown in Fig. 3g, MOLER is able to reduce the size deviation, measured by the variance of size(X)-size(Y) on the test set.

Case Study. Moreover, we provide an example in Fig. 4, where VJTNN generate a small molecule with poor performance. In contrast, JTVAE+MOLER and VJTNN+MOLER generate target molecules that achieve high similarity and much more improved QED score for the same input molecule.

4.8 Q2: The Positive Effect of Similarity Reward

Next, we present the effect of similarity reward defined in Equation (10) (Section 3.2), and show how we set the similarity reward.

In practice, to accelerate the optimization procedure [32], [33], we want to make the average $R(X, Y)$ to be close to 0,

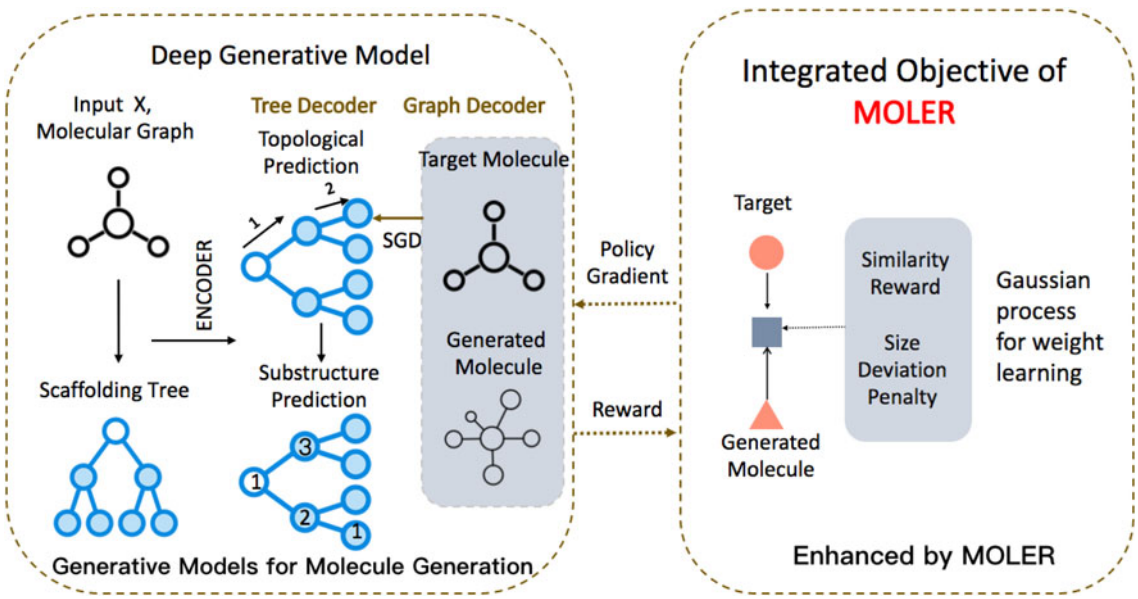
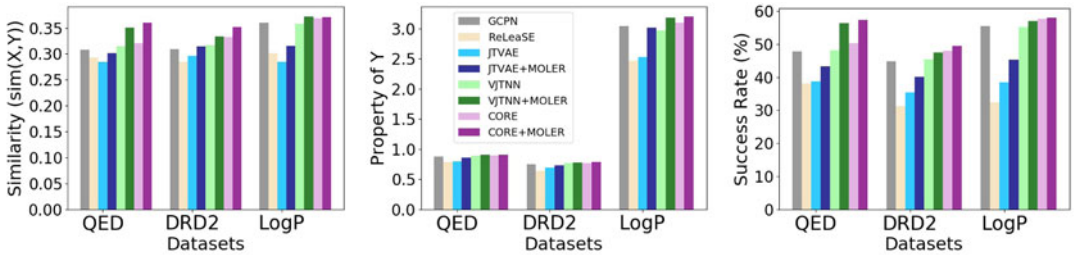
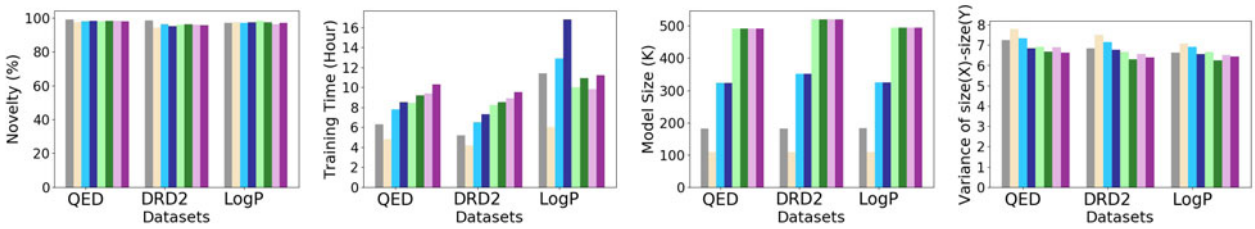


Fig. 3. Results measured by novelty, training time, model size and size deviation of generated molecule. (a) All methods have $\sim 100\%$ novelty scores, and (b-c) MOLER variants do not cost much more computational cost in training time, and their model sizes are compared to the original generative models. (d) MOLER can reduce the size deviation of generated molecules.



(a) Similarity is significantly higher. (b) Improvement on Property of generated Y (c) Higher success rate.



(d) High novelty for all methods. (e) Comparable training time. (f) Same model size with baseline. (g) MOLER reduces Size deviation.

Fig. 4. A case study on QED dataset, where VJTNN generate a small molecule with poor performance, VJTNN+MOLER generates a molecule that achieves much better similarity and QED score.

so we subtract its average from the original value. For example, in similarity reward, the reward is

$$R(X,Y) = \text{sim}(X,Y) - C, \tag{24}$$

where $C \in \mathbb{R}$ is a hyperparameter, we want it to be close to the average of all the similarity value.

Two hyperparameters play crucial role in the empirical performance of similarity reward: (1) the weight of similarity reward in the whole objective $w_{\text{sim}} \in \mathbb{R}_+$ in Equation (10); (2) hyperparameter in similarity reward $C \in \mathbb{R}$ in Equation (24). We search the weight of similarity reward w_{sim} from $\{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}\}$. For hyperparameter in similarity reward C, we want it to be close to the average of similarity value. During the learning procedure, the similarity would increase from 0 to about 0.3-0.4. So we search C from $\{0, 0.1, 0.2, 0.3, 0.4\}$. We use grid search to find the optimal combination of hyperparameters. For all the possible combinations, similarity and property of generated molecules are shown in Table 5. $w_{\text{sim}} = 0$ corresponds to baseline method (VJTNN) that don't use similarity reward. We can find that most of the (w_{sim}, C) combinations can outperform baseline method, validating the effectiveness of adding similarity reward.

In addition, we show more visualization results in Fig. 5. For each weight $w_{\text{sim}} \in \{1e^{-4}, 3e^{-4}, 1e^{-3}, 3e^{-3}, 1e^{-2}\}$, we show the change of performance (both similarity and property) with different Cs (in Equation (24)) in Figs. 5a and 5b. Also, for each $C \in \{0, 0.1, 0.2, 0.3, 0.4\}$, we show the change of performance with various w_{sim} in Figs. 5c and 5d. We find that (i) $w_{\text{sim}} = 1e^{-3}$ performs best among all the hyperparameters, especially on improvement of similarity; (ii) When $C = 0.3$, MOLER achieve the best performance; (iii) within a reasonable range, the similarity would increase as we increase the weight w_{sim}; (iv) too large w_{sim} would degrade the performance, even worse than baseline method (VJTNN).

For QED dataset, the optimal combination of hyperparameters is $w_{\text{sim}} = 1e^{-3}$ and $C = 0.3$, as mentioned above. It is also validated to be optimal in DRD2 dataset. For LogP dataset, the optimum is $w_{\text{sim}} = 1e^{-3}$ and $C = 0.4$.

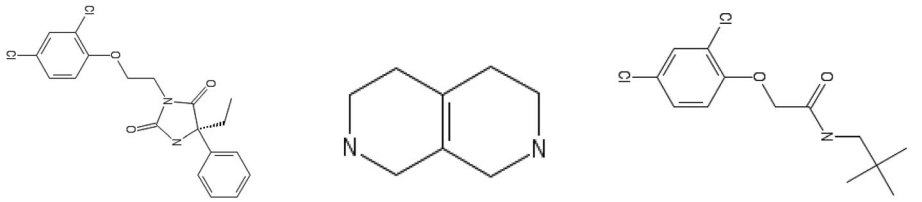
4.9 Q3: The Positive Impact of Size Deviation Penalty

Next, we demonstrate the effect of the size deviation penalty on the quality of generated molecules. The selection of

TABLE 5
Empirical Performance of Adding Only Similarity Reward for Different Hyperparameter on QED Dataset

weight w_{sim} / C in reward	0	$1e^{-4}$	$3e^{-4}$	$1e^{-3}$	$3e^{-3}$	$1e^{-2}$
0	0.314, 0.886	0.307,0.881	0.315,0.882	0.322,0.886	0.312,0.873	0.243,0.819
0.1		0.318,0.887	0.321,0.889	0.328,0.886	0.319,0.871	0.273,0.781
0.2		0.328,0.884	0.336,0.885	0.342,0.885	0.332,0.879	0.262,0.801
0.3		0.336,0.892	0.342,0.898	**0.347,0.902**	0.325,0.863	0.251,0.812
0.4		0.320,0.884	0.317,0.887	0.328,0.889	0.320,0.862	0.219,0.832
ADAPT		0.338,0.888	0.340,0.893	0.345,0.897	0.335,0.863	0.298,0.813

Average $\text{Sim}(X,Y)$, $\text{QED}(Y)$ *are reported. We use grid search to find the best hyperparameter combination for similarity reward. The optimal selection is* $w_{\text{sim}} = 1e^{-3}, C = 0.3$, *reaching the best similarity and property improvement at the same time.*



(a) Input X, QED:0.75. (b) Generated(VJTNN), Sim:0.04, QED:0.56. (c) Generated(VJTNN+MOLER), Sim:0.31, QED:0.92

Fig. 5. Selection of (w_{sim}, C) on QED dataset for similarity reward (Section 3.2). We find (i) $w_{\text{sim}} = 1e^{-3}$ and $C = 0.3$ performs best among all the hyperparameters, especially on improvement of similarity; (ii) within a reasonable range, the similarity would increase when weight w_{sim} increase. However, too large w_{sim} would degrade the performance, even worse than baseline method (VJTNN, dashed line).

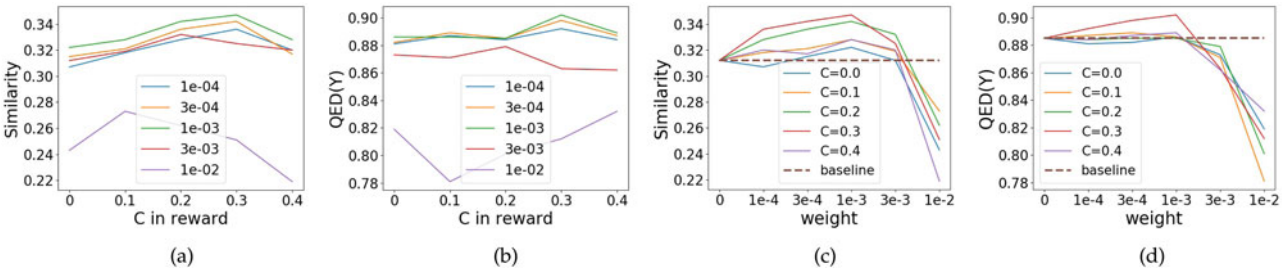


Fig. 6. Empirical effect of different size deviation penalty function (from g_1 to g_5, Section 4.9) for size deviation penalty on QED using VJTNN+MOLER. The dash lines correspond to baseline VJTNN (weight of reward is set to 0). g_5 and $w_{\text{size}} = 1e^{-3}$ obtain the best performance among all selections.

reward weight w_{size} and the reward function g (Equation (14)) are important to the performance. We also compare the performance of different size deviation penalty functions, where x, y represent the size of input X and generated Y, respectively.

- g_1, reward function that discourage generating small molecule. It set a global threshold S for all the molecules, so it doesn't depend on the input molecule X. It is defined as

$$g_1(x, y) = \begin{cases} 0, & y \geq S, \\ S - y, & y < S, \end{cases} \tag{25}$$

  it assigns large penalty when size of Y is small, We set $S = 10$ because it achieve best performance. The average number of substructures in training set is around 14, as shown in Table 3.
- g_2, reward function that discourage generating large molecule. Similar with g_1, it set a global threshold T for all the molecules and is defined as

$$g_2(x, y) = \begin{cases} 0, & y \leq T, \\ y - T, & y > T, \end{cases} \tag{26}$$

  which returns large penalty for large molecule. $T = 17$ achieve best performance.
- g_3, reward function that discourage generating small molecule compared with the size of input molecule X. It is defined as

$$g_3(x, y) = \begin{cases} 0, & y \geq x - \epsilon, \\ |x - y| - \epsilon, & y < x - \epsilon, \end{cases} \tag{27}$$

  it returns a big penalty when the size of Y is much smaller than the size of X.
- g_4, reward function that discourage generating large molecule. Similar with g_3, It depends on the size of input molecule X and is defined as

$$g_4(x, y) = \begin{cases} 0, & y - x \leq \epsilon, \\ |x - y| - \epsilon, & y - x > \epsilon, \end{cases} \tag{28}$$

- g_5 ($g_5 = g$ in Equation (14) in Section 3.3) can be seen as a combination of g_3 and g_4. It will generate penalty when the size of generated Y deviate significantly from the input molecule X. It is defined as

$$g_5(x, y) = \begin{cases} |x - y| - \epsilon, & |x - y| > \epsilon, \\ 0, & |x - y| \leq \epsilon. \end{cases} \tag{29}$$

For g_3, g_4 and g_5, $\epsilon = 3$ perform best among $\epsilon \in \{1, 2, 3, 4, 5\}$ on QED, so it's fixed to 3.

For each size deviation penalty g, we plot the change of performance (both similarity and property) with various weight w_{size} in Fig. 6 on the QED dataset. We find that (i) g_5 (which is represented in Section 3.3) and $w_{\text{size}} = 1e^{-3}$ achieve the best performance; (ii) most of the hyperparameter setting would outperform VJTNN (baseline). Therefore, the effect of size deviation penalty is positive. Moreover, this selection is also validated to be optimal on both DRD2 and LogP datasets.

5 Conclusion

In this paper, we have proposed to incorporate molecule level reward function (MOLER) into deep generative models for molecule optimization. Specifically, we have designed two molecule reward functions motivated by some empirical observations. The first one is the similarity reward to encourage the generated molecule to be similar to the input

one. Another reward is to control the size of the generated molecule explicitly. MOLER provides a general and flexible framework that can incorporate various reward functions to specify different aspects of generated molecules based on any deep generative models for molecule optimization. Policy gradient is applied to optimize the reward objective, and it wouldn't cause too much extra computational cost compared with deep generative models. Thorough empirical studies have been conducted on several real molecule optimization tasks to validate the effectiveness of MOLER.

In the future, we want to explore the following directions: (1) designing more reward functions that can improve the molecule optimization procedure; (2) decomposing molecule level reward into substructure level to make the learning procedure easier and more efficient.

References

[1] P. G. Polishchuk, T. I. Madzhidov, and A. Varnek, "Estimation of the size of drug-like chemical space based on GDB-17 data," *J. Comput.-Aided Mol. Des.*, vol. 27, no. 8, pp. 675–679, 2013.

[2] K. Huang, T. Fu, L. Glass, M. Zitnik, C. Xiao, and J. Sun, "DeepPurpose: A deep learning library for drug-target interaction prediction," *Bioinformatics*, vol. 36, pp. 5545–5547, 2020.

[3] K. Huang *et al.*, "MolDesigner: Interactive design of efficacious drugs with deep learning," *NEURIPS Demo*, 2020.

[4] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 1945–1954.

[5] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, "Syntax-directed variational autoencoder for structured data," in *Proc. 6th Int. Conf. Learn. Representations*, 2018.

[6] R. Gómez-Bombarelli *et al.*, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS Central Sci.*, vol. 4, no. 2, pp. 268–276, 2018.

[7] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, "Graph convolutional policy network for goal-directed molecular graph generation," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 6412–6422.

[8] Z. Zhou, S. Kearnes, L. Li, R. N. Zare, and P. Riley, "Optimization of molecules via deep reinforcement learning," *Sci. Rep.*, vol. 9, no. 1, 2019, Art. no. 10752.

[9] T. Blaschke, M. Olivecrona, O. Engkvist, J. Bajorath, and H. Chen, "Application of generative autoencoder in de novo molecular design," *Mol. Informat.*, vol. 37, 2018, Art. no. 1700123.

[10] N. D. Cao and T. Kipf, "MolGAN: An implicit generative model for small molecular graphs," 2018.

[11] Q. Liu, M. Allamanis, M. Brockschmidt, and A. Gaunt, "Constrained graph variational autoencoders for molecule design," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 7795–7804.

[12] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 2323–2332.

[13] W. Jin, K. Yang, R. Barzilay, and T. Jaakkola, "Learning multimodal graph-to-graph translation for molecular optimization," in *Proc. Int. Conf. Learn. Representations*, 2019.

[14] W. Jin, R. Barzilay, and T. Jaakkola, "Multi-resolution autoregressive graph-to-graph translation for molecules," 2019, *arXiv: 1907.11223*.

[15] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," *J. Cheminformatics*, vol. 9, no. 1, Sep. 2017, Art. no. 48.

[16] E. Putin, "Reinforced adversarial neural computer for de novo molecular design," *J. Chem. Inf. Model.*, vol. 58, pp. 1194–1204, 2018.

[17] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Sci. Advances*, vol. 4, no. 7, 2018, Art. no. eaap7885.

[18] T. Fu, C. Xiao, and J. Sun, "CORE: Automatic molecule optimization using copy & refine strategy," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, pp. 638–645.

[19] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. 27th Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[20] M. Popova, O. Isayev, and A. Tropsha, "Deep reinforcement learning for de novo drug design," *Sci. Advances*, vol. 4, no. 7, Jul. 2018, Art. no. eaap7885.

[21] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms," 2018, *arXiv: 1802.05054*.

[22] B. Tan, Z. Hu, Z. Yang, R. Salakhutdinov, and E. Xing, "Connecting the dots between MLE and RL for sequence generation," in *Proc. Int. Conf. Learn. Representations*, 2019.

[23] T. Fu, C. Xiao, M. L. Glass, and J. Sun, "α-MOP: Molecule optimization with α-divergence," in *Proc. IEEE Int. Conf. Bioinf. Biomedicine*, 2020, pp. 240–244.

[24] D. Comings, D. Muhleman, and R. Gysin, "Dopamine D2 receptor (DRD2) gene and susceptibility to posttraumatic stress disorder: A study and replication," *Biol. Psychiatry*, vol. 40, no. 5, pp. 368–372, 1996.

[25] G. R. Bickerton, G. V. Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nat. Chem.*, vol. 4, no. 2, 2012, Art. no. 90.

[26] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 2702–2711.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Representations*, 2017.

[28] A. Vaswani *et al.*, "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[29] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *J. Chem. Inf. Model.*, vol. 50, no. 5, pp. 742–754, 2010.

[30] G. Landrum *et al.*, "RDKit: Open-source cheminformatics," 2006.

[31] I. Steinwart and A. Christmann, "Sparsity of SVMs that use the epsilon-insensitive loss," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1569–1576.

[32] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[34] R. Ranganath, S. Gerrish, and D. Blei, "Black box variational inference," in *Proc. 17th Int. Conf. Artif. Intell. Statist.*, 2014, pp. 814–822.

[35] C. M. Bishop, *Pattern Recognition and Machine Learning*. Berlin, Germany: Springer, 2006.

[36] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Berlin, Germany: Springer, 2003, pp. 63–71.

[37] T. Fu, C. Xiao, X. Li, M. L. Glass, and J. Sun, "MIMOSA: Multi-constraint molecule sampling for molecule optimization," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 125–133.

[38] P. Ertl and A. Schuffenhauer, "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions," *J. Cheminformatics*, vol. 1, no. 1, 2009, Art. no. 8.

[39] T. Sterling and J. J. Irwin, "ZINC 15–ligand discovery for everyone," *J. Chem. Inf. Model.*, vol. 55, no. 11, pp. 2324–2337, 2015.

[40] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, "Molecular de-novo design through deep reinforcement learning," *J. Cheminformatics*, vol. 9, no. 1, 2017, Art. no. 48.

[41] W. Wei, Q. Zhang, and L. Liu, "Bitcoin transaction forecasting with deep network representation learning," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1359–1371, Third Quarter 2021.

Tianfan Fu received the BS and MS degrees from Shanghai Jiao Tong University, China, in 2015 and 2018, respectively. He is currently working toward the third-year PhD degree in the College of Computing, Georgia Tech, Atlanta, Georgia. His research interests are machine learning and data mining for healthcare, especially deep learning methods for drug discovery.

Cao Xiao received the PhD degree from the University of Washington, Seattle, Washington, in 2016, and was a research staff member in the AI for Healthcare team at IBM Research from 2017 to 2019 and served as member of the IBM Global Technology Outlook Committee from 2018 to 2019. She is the director of machine learning at Analytics Center of Excellence of IQVIA, Cambridge, Massachusetts. She is leading IQVIA's North America machine learning teams to drive next generation healthcare AI. Her research focuses on developing machine learning and deep learning models to solve diverse real world healthcare challenges. Particularly, she is interested in deep phenotyping on electronic health records, graph neural networks for in-silico drug modeling, patient segmentation for neurodegenerative diseases. The results of her research have been published in leading AI conferences including NIPS, ICLR, KDD, AAAI, IJCAI, SDM, ICDM, WWW and top health informatics journals such as the *Nature Scientific Reports* and the *Journal of the American Medical Informatics Association*.

Lucas M. Glass received the master's degree in biostatistics from Drexel University, Philadelphia, Pennsylvania, and is currently working toward the PhD degree at Temple University, Philadelphia, Pennsylvania, in statistics. He is the Global Analytics Lead within the Analytics Center of Excellence at IQVIA, Plymouth Meeting, Pennsylvania. His teams build data science and artificial intelligence microservices to make the design, planning, and execution of clinical research more efficient. Prior to IQVIA, he worked on healthcare fraud analytics at the Department of Justice.

Jimeng Sun received the BS and MPhil degrees in computer science from the Hong Kong University of Science and Technology, Hong Kong, in 2002 and 2003, respectively, and the PhD degree in computer science from Carnegie Mellon University, Pittsburgh, Pennsylvania, in 2007, advised by Christos Faloutsos. He is a professor at Computer Science Department, University of Illinois, Urbana-Champaign, Champaign, Illinois. Prior to UIUC, he was a researcher at IBM TJ Watson Research Center and an associate professor at Georgia Tech, Atlanta, Georgia. His research focuses on data mining and health analytics, especially in tensor factorizations, deep learning, and large-scale predictive modeling systems. He has been collaborating with many healthcare organizations. He published more than 120 papers and filed more than 20 patents (five granted). He has received SDM/IBM Early Career Research Award 2017, ICDM Best Research Paper Award in 2008, SDM Best Research Paper Award, in 2007, and KDD Dissertation Runner-up Award, in 2008.