GRAPH PIECE: EFFICIENTLY GENERATING HIGH-QUALITY MOLECULAR GRAPHS WITH SUBSTRUCTURES

Anonymous authors
Paper under double-blind review

ABSTRACT

Molecular graph generation is a fundamental but challenging task in various applications such as drug discovery and material science, which requires generating valid molecules with desired properties. Auto-regressive models, which usually construct graphs following sequential actions of adding nodes and edges at the atom-level, have made rapid progress in recent years. However, these atom-level models ignore high-frequency subgraphs that not only capture the regularities of atomic combination in molecules but also are often related to desired chemical properties. In this paper, we propose a method to automatically discover such common substructures, which we call *graph pieces*, from given molecular graphs. Based on graph pieces, we leverage a variational autoencoder to generate molecules in two phases: piece-level graph generation followed by bond completion. Experiments show that our graph piece variational autoencoder achieves better performance over state-of-the-art baselines on property optimization and constrained property optimization tasks with higher computational efficiency.

1 INTRODUCTION

Molecule generation is fundamental to a variety of applications such as drug discovery and material science. It requires producing chemically valid molecules with optimized properties. Recent years have seen impressive progress of deep graph generative models on this task (Jin et al., 2018; Li et al., 2018a; You et al., 2018; Kwon et al., 2019; De Cao & Kipf, 2018; Shi et al., 2020; Jin et al., 2020b). While relatively low validity remains a bottleneck for non-autoregressive models (Simonovsky & Komodakis, 2018; Flam-Shepherd et al., 2020; Kwon et al., 2019), autoregressive models solve this by fusing step-wise validity check into the generation process (Jin et al., 2018; You et al., 2018).

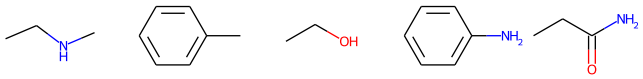
Autoregressive models usually decompose molecular graphs into sequential actions of generating atoms and bonds. However, this fine-grained decomposition ignores the existence of common substructures in molecule graphs, as illustrated in Figure 1. We indicate two benefits of using substructures instead of atoms for generating molecular graphs:

1. *Ability to capture the regularities and their correlation with chemical properties*. As shown in Figure 2, the occurrence of the same substructure in the molecules with top and last 10 % Penalized logP[1] in the ZINC250K dataset (Irwin et al., 2012; Kusner et al., 2017) differs significantly, which indicates that substructures and chemical properties are correlated.
2. *Efficient training and inference*. Using substructures to represent molecule graphs can result in much shorter sequences, and therefore accelerates the training and inference process.

As a result, we believe that models using substructures to represent molecule graphs can generate more realistic molecules with better-optimized properties.

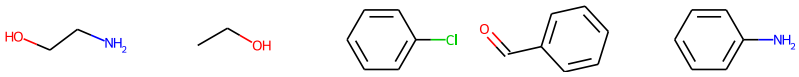
Based on the above intuition, we propose a straightforward algorithm to automatically discover common substructures in molecules, which we call *graph pieces*. The algorithm works iteratively. We first initialize all graph pieces with single atoms (i.e., decomposing molecules into atoms). Then,

[1]Penalized logP is a commonly used chemical property (You et al., 2018; Shi et al., 2020)



83.9% 60.5% 41.3% 32.6% 27.5%

Figure 1: Five high-frequency substructures in the standard ZINC250K dataset (Irwin et al., 2012; Kusner et al., 2017). The percentage means the ratio of molecules which have the substructure.



low PlogP	27.6%	48.7%	2.8%	5.0%	10.7%
high PlogP	8.0%	27.3%	27.6%	32.0%	61.4%

Figure 2: Five substructures which may be related to chemical properties because they have distinct frequency of occurrence in molecules with low and high PlogP (Penalized logP). Each substructure is labeled with its frequency of occurrence in molecules with low PlogP and high PlogP.

for each piece, our method attaches its neighboring pieces to form new graph pieces and calculates their frequency of occurrence. For each iteration, the most frequent graph piece is merged and added to the vocabulary. Since substructures can be seen as small molecules, we use SMILES (Weininger, 1988), a text-based representation for molecules, to judge whether two graph pieces are identical. Each iteration yields one graph piece and larger graph pieces tend to appear in later iterations.

To efficiently utilize these substructures, we also propose a graph piece variational autoencoder. Our model uses a graph neural network (Scarselli et al., 2008, GNN) as an encoder and generates a molecule in two phases: the autoregressive piece-level sequence generation followed by the atom-level bond completion. The second step can be formalized as a link prediction task which can be done simply by GNNs such as GCN (Kipf & Welling, 2016). The two-step generation decouples the nested sequence of alternately generating nodes and edges. As a result, the training process is more computationally efficient, which is about three times faster than the fastest state-of-the-art method (Shi et al., 2020). We conduct extensive experiments on the ZINC250K dataset (Irwin et al., 2012). Results demonstrate that our graph piece variational autoencoder outperforms state-of-the-art methods on both property optimization and constrained property optimization tasks.

2 Related Work

Molecular Representation There are two popular kinds of molecular representations. The simplified molecular-input line-entry system (SMILES) (Weininger, 1988) is a text-based representation. It is widely used due to its simplicity (Gómez-Bombarelli et al., 2018; Kusner et al., 2017; Bjerrum & Threlfall, 2017), but it is not robust because a single perturbation in the string can result in significant changes in molecule structure and vice versa (You et al., 2018; Kwon et al., 2019). Recent years have seen rapid development in representing molecules with graphs where atoms and chemical bonds correspond to nodes and edges, which is more natural and robust (De Cao & Kipf, 2018; Shi et al., 2020; Jin et al., 2020b). In this work, we adopt the graph-based representation.

Molecular Graph Generation Models We divide modern molecular graph generation models into two categories according to their granularity of building blocks:

- **Atom-level**: Li et al. (2018b) proposed a generative model of graphs and demonstrated it performed better than SMILES-based generative models on molecular generation. You et al. (2018) used reinforcement learning to fuse rewards of chemical validity and property scores into each step of generating a molecule. Popova et al. (2019) proposed a Molecular-RNN to autoregressively generate nodes and edges based on the generated graph. Shi et al. (2020) proposed a flow-based autoregressive model and use reinforcement learning for goal-directed molecular graph generation. However, using atom-level representation will result in very long sequences, therefore the training process is typically time-consuming.
- **Substructure-level**: Jin et al. (2018) proposed to generate molecules in the form of junction trees where each node is a ring or edge. Jin et al. (2020a) decomposed molecules

by breaking all the bridge bonds to extract substructures. It used a complex hierarchical model for polymer generation and graph-to-graph translation. Jin et al. (2020b) proposed to extract the smallest substructure which maintains the original chemical property. The extracted substructures usually include most atoms of the original molecules, which are too coarse-grained and exert limitations on the search space.The models proposed by Jin et al. (2020a) and Jin et al. (2020b) are not suitable for the experiments in this paper, since they either need graph-to-graph supervised data or are incompatible with continuous properties.

Our model use substructures as building blocks. Different from Jin et al. (2018; 2020a) which use manual rules to extract substructures, we automatically extract common substructures which better capture the regularities in molecules. Moreover, our model improves computational efficiency with a two-step decoding method that avoids nested sequences of alternately generating nodes and edges.

Genetic Algorithms Developing better searching algorithms instead of better generative models is another aspect for the task of molecular property optimization, and genetic algorithm shows great potential (Jensen, 2019; Nigam et al., 2020; Ahn et al., 2020). Jensen (2019) defined rules of mutations and crossovers on the graph representation of molecules. Nigam et al. (2020) designed a neural network to model the fitness in the selection of the subsequent population. Ahn et al. (2020) proposed a framework for applying genetic algorithms to generative models. While searching algorithms is also crucial, in this paper we mainly focus on the generative side.

3 Approach

We first give the definition of *graph piece* and formalize the molecular representation in Section 3.1. Then, we illustrate our algorithm for extracting *graph pieces* and decomposing molecules into *graph pieces* in Section 3.2. We propose a graph piece variational autoencoder, in which a molecule is encoded into a latent variable $\boldsymbol{z}$ using an encoder (Section 3.3). Our approach decodes $\boldsymbol{z}$ in two phases: it first autoregressively generates graph pieces and then completes the inter-piece connections in a non-autoregressive fashion (Section 3.4).

3.1 Definition of Graph Piece

A molecule can be represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of atoms and $\mathcal{E}$ is a set of edges that correspond to chemical bonds. We use substructures, which we call *graph pieces*, as building blocks. Therefore, the molecules can be alternatively represented as $\mathcal{G} = (\mathcal{S}, \tilde{\mathcal{E}})$. $\mathcal{S} = \{\mathcal{S}_i = (\mathcal{V}_i, \mathcal{E}_i) | i = 0, 1, 2, ...\}$ contains all the graph pieces in the molecule. $\tilde{\mathcal{E}}$ contains all the connections between the atoms in different graph pieces. Graph pieces in $\mathcal{S}$ satisfy the following three constraints: (1) The union of all atoms in the graph pieces equals to all atoms in the molecule, namely $\cup_i \mathcal{V}_i = \mathcal{V}$; (2) The union of all connections within and between graph pieces equals to all bonds in the molecule, namely $\cup_i \mathcal{E}_i \cup \tilde{\mathcal{E}} = \mathcal{E}$; (3) There is no intersection between all graph pieces, namely $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset \ (\forall i \neq j)$.

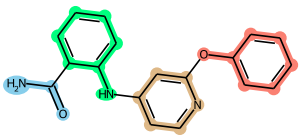


Figure 3: Four graph pieces in an example molecule. Different graph pieces are highlighted in different colors.

3.2 Graph Piece Extraction

We draw inspiration from Byte Pair Encoding (Gage, 1994, BPE) which is a simple data compression technique that iteratively replaces the most frequent pair of bytes in a sequence with a single, unused byte. Our graph piece extraction algorithm iteratively merges the most frequent pair of connected graph pieces into a new graph piece, which is then added to the vocabulary. We initialize the vocabulary with single atoms and decompose molecules as sequences of atoms with the canonical SMILES order (Weininger, 1988). Each graph piece has a set of graph pieces that are connected to it. A pair of connected graph pieces can be represented as $(A, \tilde{\mathcal{E}}, B)$ where A and B are two graph pieces and $\tilde{\mathcal{E}}$ contains all bonds connecting two atoms in A and B. $(A, \tilde{\mathcal{E}}, B)$ is equivalent to $(B, \tilde{\mathcal{E}}, A)$ due to the disordered nature of graphs. Since each graph piece can be seen as a small molecule, we use SMILES (Weininger, 1988) which is unique for different molecules to represent

(a) Initialization (b) Iteration 1 (c) Iteration 2

Figure 4: Two iterations of our graph piece extraction algorithm on {C=CC=C, CC=CC, C=CCC}. (a) The vocabulary is initialized with atoms. (b) Graph piece CC is the most frequent and added to the vocabulary. All CC are merged and highlighted in red. (c) Graph piece C=CC is the most frequent and added to the vocabulary. All C=CC are merged and highlighted in green (molecules 1 and 3). After 2 iterations the vocabulary is {C, CC, C=CC}. Note that due to the uniqueness of SMILES, graph piece C=CC will not be translated into CC=C.

graph pieces as strings, avoiding complicated graph matching process. For each iteration, we count the frequency of each pair of graph pieces and merge the most frequent pair $(A, \tilde{\mathcal{E}}, B)$ into a new graph piece. Then each $(A, \tilde{\mathcal{E}}, B)$ is replaced with the new graph piece. Note that if two pairs $(A, \tilde{\mathcal{E}}, B)$ shares the same A or B in the graph, only the pair traversed first will be merged. Each iteration produces a new graph piece, therefore the vocabulary size is equal to the number of atomic types plus the number of iterations. We use N as a hyperparameter to represent the final vocabulary size. Larger N indicates more coarse-grained decomposition. The pseudo code is in Appendix B.

Figure 4 shows an example of our graph piece extraction algorithm on a toy dataset containing three molecules. During test time, we first decompose a molecule into atoms. Then we apply the same algorithm to iteratively merge the graph piece with the highest frequency recorded in the vocabulary. Note that a limitation of JT-VAE (Jin et al., 2018) is that some molecules cannot be represented as a junction tree while our algorithm guarantees there is a decomposition for an arbitrary molecule.

3.3 Graph Encoder

As shown in Figure 5, we use a GNN to encode a molecular graph $\mathcal{G}$ into a latent variable $\boldsymbol{z}$. Each node v has a feature vector $\boldsymbol{x_v}$ which indicates its atomic type, the type and generation order of the graph piece it is in. Each edge has a feature vector indicating its bond type. Modern GNNs follow a neighborhood aggregation strategy (Xu et al., 2018), which iteratively update the representations of nodes with AGGREGATE and COMBINE process. The representation of node v in the k-th iteration is calculated as follows:

$$\boldsymbol{a}_v^{(k)} = \text{AGGREGATE}^{(k)}(\{\boldsymbol{h}_u^{(k-1)} : u \in \mathcal{N}(v)\}), \tag{1}$$

$$\boldsymbol{h}_v^{(k)} = \text{COMBINE}^{(k)}(\boldsymbol{h}_v^{(k-1)}, \boldsymbol{a}_v^{(k)}), \tag{2}$$

where AGGREGATE and COMBINE vary in different GNNs. $\mathcal{N}(v)$ denotes neighboring nodes of v. In this work we set $\boldsymbol{h_v}^{(0)} = \boldsymbol{x_v}$ and use GIN with edge feature (Hu et al., 2019) which implements AGGREGATE and COMBINE as follows:

$$\boldsymbol{a}_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{ReLU}(\boldsymbol{h}_u^{(k-1)} + \boldsymbol{e}_{uv}) \tag{3}$$

$$\boldsymbol{h}_v^{(k)} = h_\Theta((1+\varepsilon)\boldsymbol{h}_v^{(k-1)} + \boldsymbol{a}_v^{(k)}), \tag{4}$$

where h_Θ is a neural network and ε is a constant. We implement h_Θ as a 2-layer multilayer perceptron (Gardner & Dorling, 1998, MLP) with ReLU activation and set $\varepsilon = 0$. Since the k-th representation captures k-hop feature of nodes (Xu et al., 2018), we obtain the final representations of nodes as $\boldsymbol{h}_v = [\boldsymbol{h}_v^{(1)}, \boldsymbol{h}_v^{(2)}, ..., \boldsymbol{h}_v^{(t)}]$ so that it contains 1-hop to t-hop contextual information. In this work we set $t = 4$. Then we compute the representation of graph $\mathcal{G}$ through summation $\boldsymbol{h}_\mathcal{G} = \sum_{v \in \mathcal{V}} \boldsymbol{h}_v$. We use $\boldsymbol{h}_\mathcal{G}$ to obtain the mean $\boldsymbol{\mu}_\mathcal{G}$ and log variance $\boldsymbol{\sigma}_\mathcal{G}$ of variational posterior approximation $q(\boldsymbol{z}|\mathcal{G})$ through two separate linear layers and use the reparameterization trick (Kingma & Welling, 2013) to sample from the distribution in the training process.

3.4 Two-step Decoder

With a molecule represented as $\mathcal{G} = (\mathcal{S}, \tilde{\mathcal{E}})$, our model generates $\mathcal{S}$ and $\tilde{\mathcal{E}}$ in two consecutive phases. The two phases move from coarse to fine granularity.

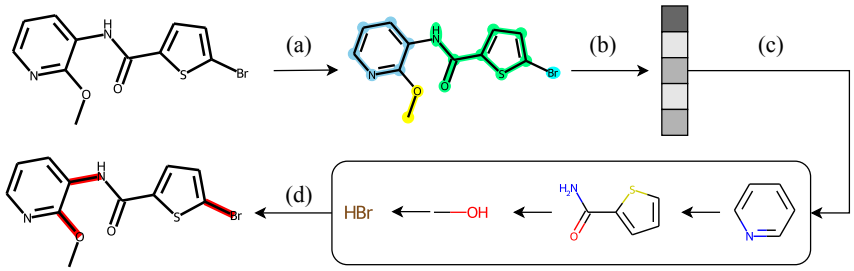


Figure 5: Overview of the *graph piece variational autoencoder*. (a) Piece-level decomposition of molecules. Atoms and bonds which belong to different graph pieces are highlighted in different colors. (b) Latent space encoding. We obtain the latent variable $\boldsymbol{z}$ through the graph encoder. (c) Piece-level sequence generation. A sequence of graph pieces is autoregressively decoded from the latent variable. (d) Bond completion. We highlight the added bonds, which connect two atoms from different graph pieces. After training, we sample from the latent space to generate molecules.

Piece-level Sequence Generation Given a latent variable $\boldsymbol{z}$, our model first uses an autoregressive sequence generation model $P(\mathcal{S}_i|\mathcal{S}_{<i}, \boldsymbol{z})$ to decode a sequence of graph pieces $\mathcal{S} = [\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_n]$[2]. During training, we insert two special tokens $\langle\texttt{bos}\rangle$ and $\langle\texttt{eos}\rangle$ into the begin and the end of the graph piece sequence to indicate the begin and the end, respectively. During testing, the generation stops when a $\langle\texttt{eos}\rangle$ is generated. The sequence model can be RNNs such as LSTM (Hochreiter & Schmidhuber, 1997) and GRU (Cho et al., 2014). In this work, we use a single layer of GRU and project the latent variable $\boldsymbol{z}$ to the hidden size of GRU to initialize the hidden state. The training objective of this stage is to minimize the log-likelihood $\mathcal{L}_{\mathcal{S}}$ of the ground truth graph piece sequence:

$$\mathcal{L}_{\mathcal{S}} = -\sum_{i=1}^{n} \log P(\mathcal{S}_i|\mathcal{S}_{<i}, \boldsymbol{z}) \tag{5}$$

Bond Completion We generate $\tilde{\mathcal{E}}$ non-autoregressively. We use a GNN with the same structure as the graph encoder in Section 3.3 but with different parameters to obtain the representations $\boldsymbol{h}_v$ of each atom v in the graph pieces. Given nodes v and u, we predict their connections as follows:

$$P(\boldsymbol{e}_{uv}|\boldsymbol{z}) = H_\theta([\boldsymbol{h}_v; \boldsymbol{h}_u; \boldsymbol{z}]) \tag{6}$$

where H_θ is a neural network. In this work we adopt a 3-layer MLP with ReLU activation. Apart from the types of chemical bonds, we also add a special type $\langle\texttt{none}\rangle$ to the edge vocabulary which indicates there is no connection between two nodes. During training, we predict both $P(\boldsymbol{e}_{uv})$ and $P(\boldsymbol{e}_{vu})$ to let H_θ learn the undirected nature of chemical bonds. We use negative sampling (Goldberg & Levy, 2014) to balance the ratio of $\langle\texttt{none}\rangle$ and chemical bonds. Since only about 2% pairs of nodes has inter-piece connections, negative sampling significantly improves the computational efficiency and scalability. The training objective of this stage is to minimize the log-likelihood $\mathcal{L}_{\tilde{\mathcal{E}}}$ of the ground truth inter-piece connections:

$$\mathcal{L}_{\tilde{\mathcal{E}}} = -\sum_{u\in\mathcal{S}_i, v\in\mathcal{S}_j, i\neq j} \log P(\boldsymbol{e}_{uv}|\boldsymbol{z}) \tag{7}$$

The training objective of the decoder it to minimize the reconstruction loss:

$$\mathcal{L}_{\text{rec}} = \mathcal{L}_{\mathcal{S}} + \mathcal{L}_{\tilde{\mathcal{E}}} \tag{8}$$

3.5 Inference

To decode $\tilde{\mathcal{E}}$ in inference time, the decoder first assigns all possible inter-piece connections with a bond type and a corresponding confidence level. We try to add bonds which has a confidence level higher than $\delta_{\text{th}} = 0.5$ to the molecule in order of confidence level from high to low. For each attempt,

[2]The ordering of graph pieces is not unique. We train with one order and leave this problem for future work.

we perform a valency check and a cycle check to reject the connections that will cause violation of valency or form unstable rings which are too small or too large. Note that this procedure may form unconnected graphs. To solve this problem, we find the maximal connected component as the result of the generation. We present the pseudo code for inference in Appendix C.

4 EXPERIMENTS

4.1 SETUP

Evaluation Tasks First we report three common metrics *Validity*, *Uniqueness*, and *Novelty* for molecular generation, which evaluate the model's ability to generate realistic and diverse molecules. Then we validate our model on two common tasks. **Property Optimization** requires generating molecules with optimized properties. **Constrained Property Optimization** concentrates on improving the chemical properties of given molecules with a restricted degree of modification.

Dataset We use the ZINC250K (Irwin et al., 2012) dataset for training. The dataset contains 250,000 drug-like molecules that are commercially available with a maximum atom number of 38. All molecules are represented in kekulized form.

Baselines We compare our graph piece variational autoencoder (**GP-VAE**) with the following state-of-the-art models. **JT-VAE** (Jin et al., 2018) is a variational autoencoder that represents molecules as junction trees. It performs Bayesian optimization on the latent variable for property optimization. We also compare our model with two baselines that utilize reinforcement learning. **GCPN** (You et al., 2018) combines reinforcement learning and graph representation for goal-directed molecular graph generation. **GraphAF** (Shi et al., 2020) is a flow-based autoregressive model for graph generation which achieves state-of-the-art performance on property optimization and constrained property optimization. It is first pretrained for likelihood modeling, then fine-tuned with reinforcement learning for property optimization. All the models are trained on the ZINC250K dataset with the same train/valid/test split as in (Kusner et al., 2017). **GA** (Nigam et al., 2020) adopts genetic algorithms for molecular property optimization and uses a neural network to model the selection of the subsequent population.

Implementation Details We choose $N = 300$ for property optimization and $N = 500$ for constrained property optimization. We train our GP-VAE on the ZINC250K dataset for 6 epochs with a batch size of 32 and a learning rate of 0.001. We jointly train a 2-layer MLP from $\boldsymbol{z}$ to predict the property scores. Therefore the loss function is $\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{rec}} + (1 - \alpha) \cdot \mathcal{L}_{\text{prop}} + \beta \cdot \text{D}_{\text{KL}}$. We set $\alpha = 0.1$ and initialize $\beta = 0$. We adopt a warm-up method that increases β by 0.002 every 1000 steps to a maximum of 0.01. We use Adam (Kingma & Ba, 2014) to optimize our model. For a fair comparison, we train all the models on a machine with 1 Tesla V100 GPU and 32 CPU cores. For the training process, JT-VAE, GCPN, and GraphAF take around 24, 8, and 4 hours respectively, while our GP-VAE only takes 1.2 hours. For property optimization, we use gradient search (Luo et al., 2018) on the latent variable and perform a grid search on the hyperparameters and select the setting with the best property scores. More details can be found in Appendix D.

4.2 RESULTS

Common Metrics We follow previous works (Jin et al., 2018; You et al., 2018; Shi et al., 2020) to report *Validity*, *Uniqueness*, and *Novelty* of 10,000 generated molecules. *Validity* is the ratio of valid molecules among all generated graphs. *Uniqueness* is the ratio of unique molecules among all generated molecules. *Novelty* is the ratio of the generated molecules that are not in the training set.

Table 1 shows the common metrics of different methods. Since all the methods incorporate validity check in each step of generation, they can always generate valid molecules. GA suffers from very low uniqueness since the genetic algorithms tend to unify the population into molecules with the highest fitness in each iteration. Our model achieves competitive results in *Uniqueness* and *Novelty*, which means our model can generate diverse molecules and is not overfitted on the training set. We present some molecules sampled from the prior distribution in Appendix F.

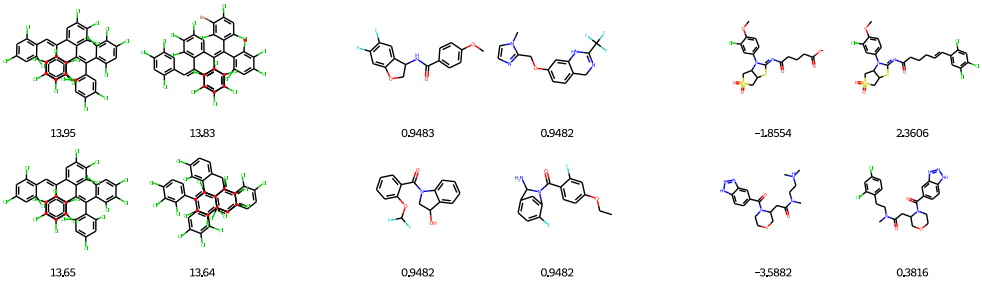
Table 1: Validity, Uniqueness and Novelty of the generated molecules by different models.

Method	Validity	Uniqueness	Novelty
JT-VAE	100%	100.00%	100%
GCPN	100%	99.97%	100%
GraphAF	100%	99.10%	100%
GA	100%	13.96%	100%
GP-VAE	100%	99.65%	100%

Property Optimization This task focuses on generating molecules with optimized Penalized logP (Kusner et al., 2017) and QED (Bickerton et al., 2012). Penalized logP is logP penalized by synthesis accessibility and ring size which has an unbounded range. QED measures the drug-likeness of molecules with a range of $[0, 1]$. Both properties are calculated by empirical prediction models (Wildman & Crippen, 1999; Bickerton et al., 2012), and are widely used in previous works (Jin et al., 2018; You et al., 2018; Shi et al., 2020). We adopt the scripts of Shi et al. (2020) to calculate property scores so that the results are comparable. We jointly train a 2-layer MLP to predict Penalized logP and QED from the latent variable $\boldsymbol{z}$. For simplicity, we directly perform gradient ascending on the latent variable (Jin et al., 2018; Luo et al., 2018). Hyperparameters can be found in Appendix D. Following previous works (Jin et al., 2018; You et al., 2018; Shi et al., 2020), we generate 10,000 optimized molecules from the latent space and report the top-3 scores found by each model. Results in Table 2 show that our model surpasses the state-of-the-art models consistently.

Table 2: Comparison of the top-3 property scores found by each model

Method	Penalized logP ↑				QED ↑			
	1st	2nd	3rd	Validity	1st	2nd	3rd	Validity
ZINC250K	4.52	4.30	4.23	100.0%	0.948	0.948	0.948	100.0%
JT-VAE	5.30	4.93	4.49	100.0%	0.925	0.911	0.910	100.0%
GCPN	7.98	7.85	7.80	100.0%	**0.948**	0.947	0.946	100.0%
GraphAF	12.23	11.29	11.05	100.0%	**0.948**	**0.948**	0.947	100.0%
GA[3]	12.25	12.22	12.20	100.0%	0.946	0.944	0.932	100.0%
GP-VAE	**13.95**	**13.83**	**13.65**	100.0%	**0.948**	**0.948**	**0.948**	100.0%



(a) Penalized logP optimization (b) QED optimization (c) Constrained optimization of Penalized logP

Figure 6: Samples of property optimization and constrained property optimization. In (c) the first and the second columns are the original and modified molecules labeled with their Penalized logP.

Constrained Property Optimization This task concentrates on improving the property scores of given molecules with the constraint that the similarity between the original molecule and the modi-

[3]Results are obtained by running the scripts from the original paper and use the scripts of Shi et al. (2020) to evaluate property scores.

fied molecule is above a threshold δ. Following Jin et al. (2018); You et al. (2018); Shi et al. (2020), we choose 800 molecules with lowest Penalized logP in the test set of ZINC250K for optimization and Tanimoto similarity with Morgan fingerprint (Rogers & Hahn, 2010) as the similarity metric.

Similar to the property optimization task, we perform gradient ascending on the latent variable with a max step of 80. We collect all latent variables which have better-predicted scores than the previous iteration and decode each of them 5 times, namely up to 400 molecules. Following Shi et al. (2020), we initialize the generation with sub-graphs sampled from the original molecules. Then we choose the one with the highest property score from the molecules that meet the similarity constraint. Table 3 shows our model can generate molecules with a higher Penalized logP score while satisfying the similarity constraint. Since our model uses graph pieces as building blocks, the degree of modification tends to be greater than atom-level models, leading to a lower success rate. Nevertheless, our model still manages to achieve high success rates close to atom-level models.

Table 3: Comparison of the mean and standard deviation of improvement of each model on constrained property optimization.

Model	$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$	
	Improvement	Success	Improvement	Success	Improvement	Success
JT-VAE	1.68±1.85	97.1%	0.84±1.45	83.6%	0.21±0.71	46.4%
GCPN	4.12±1.19	100%	2.49±1.30	100%	0.79±0.63	100%
GraphAF[4]	4.99±1.38	100%	3.74±1.25	100%	1.95±0.99	98.4%
GA[5]	3.04±1.60	100%	2.34±1.34	100%	1.35±1.06	95.9%
GP-VAE	**6.42±1.86**	99.9%	**4.19±1.30**	98.9%	**2.52±1.12**	90.3%

4.3 Graph Piece Statistics

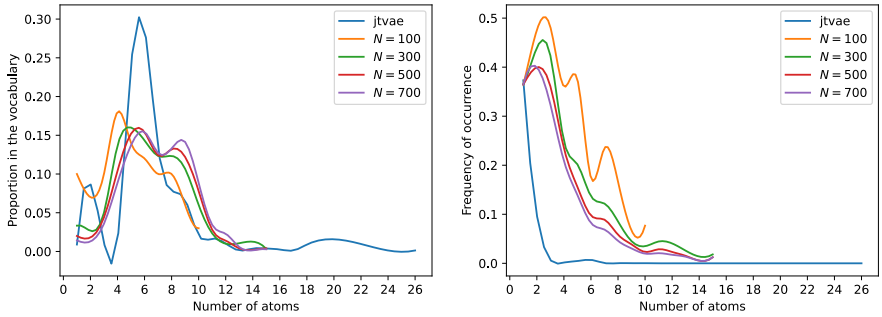


Figure 7: The left figure shows the proportion of substructures with different number of atoms in the vocabulary. The right figure shows the average frequency of occurrence of substructures with different number of atoms.

We compare the statistical characteristics of the vocabulary of JT-VAE (Jin et al., 2018) that contains 780 substructures and the vocabulary of graph pieces with a size of 100, 300, 500, and 700. Figure 7 shows the proportion of substructures with different numbers of atoms in the vocabulary and their frequencies of occurrence in the ZINC250K (Irwin et al., 2012) dataset. The substructures in the vocabulary of JT-VAE mainly concentrate on 5 to 8 atoms with a sharp distribution. However, starting from substructures with 3 atoms, the frequency of occurrence is already close to zero. Therefore, the majority of substructures in the vocabulary of JT-VAE are actually not common substructures. On the contrary, the substructures in the vocabulary of graph pieces have a relatively smooth distribution

[4]We rerun GraphAF on the same 800 molecules using its script to obtain the results since the original paper chooses a different set of 800 molecules whose results are not comparable.

[5]We set max iteration to 5 and population to 80, namely traversing up to 400 molecules, for fair comparison.

over 4 to 10 atoms. Moreover, these substructures also have a much higher frequency of occurrence compared to those in the vocabulary of JT-VAE. The distribution of the proportion of graph pieces with different numbers of atoms also shifts right with larger N, indicating the shift towards more coarse granularity. We present some graph pieces sampled from our vocabulary in Appendix E.

4.4 Ablation Study

To further validate the importance of graph pieces, we conduct an ablation study on GP-VAE with atom-level vocabulary and piece-level vocabulary. Atom-level GP-VAE has the same model structure as GP-VAE except that its vocabulary only includes single atoms. We evaluate atom-level and piece-level GP-VAEs on property optimization and constrained property optimization. Table 4 and Table 5 demonstrate that the use of graph pieces significantly improves the generation quality and that the state-of-the-art performance of our model is mainly credited to the use of graph pieces.

Table 4: The top-3 property scores found by our atom-level and piece-level GP-VAEs

Method	Penalized logP				QED			
	1st	2nd	3rd	Validity	1st	2nd	3rd	Validity
atom-level	6.91	5.50	5.12	100.0%	0.870	0.869	0.869	100.0%
piece-level	**13.95**	**13.83**	**13.65**	100.0%	**0.948**	**0.948**	**0.948**	100.0%

Table 5: Comparison of atom-level and piece-level GP-VAEs on constrained property optimization

Model	$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$	
	Improvement	Success	Improvement	Success	Improvement	Success
atom-level	2.33±1.46	74.8%	2.12±1.36	50.9%	1.87±1.12	27.0%
piecel-level	**6.42±1.86**	99.9%	**4.19±1.30**	98.9%	**2.52±1.12**	90.3%

4.5 Fused Rings Generation

We conduct an additional experiment to validate the ability of GP-VAE to generate molecules with fused rings (cycles with shared edges), because at first thought it seems difficult for GP-VAE to handle these molecules due to the non-overlapping nature of graph pieces. We train atom-level and piece-level GP-VAEs on all 4,431 structures consisting of fused rings from ZINC250K. Then we sample 1,000 molecules from the latent space to calculate the proportion of molecules with fused rings. The results are 94.5% and 97.2% for the atom-level model and the piece-level model, respectively. The experiment demonstrates that the introduction of graph pieces as building blocks will not hinder the generation of molecules with fused rings.

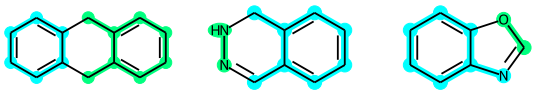


Figure 8: Decomposition of three molecules with fused rings (cycles that share edges).

5 Conclusion

We propose an algorithm to automatically discover the regularity in molecules and extract them as *graph pieces*. We also propose a graph piece variational autoencoder utilizing the graph pieces and generate molecules in two phases. Our model consistently outperforms state-of-the-art models on property optimization and constrained property optimization with higher computational efficiency. *Graph piece* has the potential to universally adapt atom-level models to their piece-level counterparts for better generation quality, which we leave for future work. Our work provides insights into the selection of granularity on molecular graph generation and can inspire future search in this direction.

REFERENCES

Sungsoo Ahn, Junsu Kim, Hankook Lee, and Jinwoo Shin. Guiding deep molecular optimization with genetic exploration. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12008–12021. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/8ba6c657b03fc7c8dd4dff8e45defcd2-Paper.pdf.

G Richard Bickerton, Gaia V Paolini, Jérémy Besnard, Sorel Muresan, and Andrew L Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4(2):90–98, 2012.

Esben Jannik Bjerrum and Richard Threlfall. Molecular generation with recurrent neural networks (rnns). *arXiv preprint arXiv:1705.04612*, 2017.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

Daniel Flam-Shepherd, Tony Wu, and Alan Aspuru-Guzik. Graph deconvolutional generation. *arXiv preprint arXiv:2002.07087*, 2020.

Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

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

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

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pp. 2323–2332. PMLR, 2018.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *International Conference on Machine Learning*, pp. 4839–4848. PMLR, 2020a.

Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Multi-objective molecule generation using interpretable substructures. In *International Conference on Machine Learning*, pp. 4849–4859. PMLR, 2020b.

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

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pp. 1945–1954. PMLR, 2017.

Youngchun Kwon, Jiho Yoo, Youn-Suk Choi, Won-Joon Son, Dongseon Lee, and Seokho Kang. Efficient learning of non-autoregressive graph variational autoencoders for molecular graph generation. *Journal of Cheminformatics*, 11(1):1–10, 2019.

Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):1–24, 2018a.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018b.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018.

AkshatKumar Nigam, Pascal Friederich, Mario Krenn, and Alán Aspuru-Guzik. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=H1lmyRNFvr`.

Mariya Popova, Mykhailo Shvets, Junier Oliva, and Olexandr Isayev. Molecularrnn: Generating realistic molecular graphs with optimized properties. *arXiv preprint arXiv:1905.13372*, 2019.

David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 50(5):742–754, 2010.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

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

Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

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

Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Jiaxuan You, Bowen Liu, Rex Ying, Vijay Pande, and Jure Leskovec. Graph convolutional policy network for goal-directed molecular graph generation. *arXiv preprint arXiv:1806.02473*, 2018.

A DISCUSSION

Universal Granularity Adaption The concept and extraction algorithm of *graph pieces* resemble those of subword units (Sennrich et al., 2015) in machine translation. Though subword units are designed for the out-of-vocabulary problem of machine translation, they also improve the translation quality (Sennrich et al., 2015). In this work, we demonstrate the power of graph pieces and are curious about whether there is a universal way to adapt atom-level models into piece-level counterparts to improve their generation quality. The key challenge is to find an efficient and expressive way to encode inter-piece connections into feature vectors. We leave this for future work.

Searching in Continuous Space In recent years, reinforcement learning (RL) is becoming dominant in the field of optimization of molecular properties (You et al., 2018; Shi et al., 2020). These RL models usually suffer from reward sparsity when applied to multi-objective optimization (Jin et al., 2020b). However, most scenarios that incorporate molecular property optimization have multi-objective constraints (e.g.,drug discovery). In this work, we show that with graph pieces, even simple searching method like gradient ascending can surpass RL methods on single-objective optimization. It is possible that with better searching methods in continuous space our model can achieve competitive results on multi-objective optimization.

B GRAPH PIECE EXTRACTION

Algorithm 1 shows the pseudo code of our graph piece extraction algorithm.

Algorithm 1 Graph Piece Extraction

Initialize: $V = \mathcal{A}$ where $\mathcal{A}$ includes all single atoms
while $|V| < N$ **do**
  Initialize: `freq = defaultdict(int)`
  for `mol` in `dataset` **do**
    for $(A, \tilde{\mathcal{E}}, B)$ in `mol` **do**
      `smi = MolToSmiles(`$(A, \tilde{\mathcal{E}}, B)$`)`
      `freq[smi] += 1`
    end for
  end for
  `piece = max(freq.keys(), key=lambda smi: freq[smi])`
  for `mol` in `dataset` **do**
    `mol.merge(piece)`
  end for
  $V \leftarrow V \cup \{$`piece`$\}$
end while

C INFERENCE ALGORITHM

Algorithm 2 shows the pseudo code of our inference algorithm.

Algorithm 2 Inference algorithm of graph piece variational autoencoder

Initialize: $\tilde{\mathcal{E}} = \emptyset$
Sort bond by confidence level: $\mathcal{B} = \texttt{sorted}(\{b_{uv} | u \in \mathcal{S}_i, v \in \mathcal{S}_j, i \neq j\}, \texttt{key} = b_{uv}.\texttt{confidence})$
for b_{uv} in $\mathcal{B}$ **do**
  if $b_{uv}.\texttt{confidence} < \delta_{th}$ **then**
    break
  else if `valence_check`(b_{uv}) and `cycle_check`(b_{uv}) **then**
    $\tilde{\mathcal{E}} \leftarrow \tilde{\mathcal{E}} \cup \{b_{uv}\}$
  end if
end for

D EXPERIMENT DETAILS

Model and Training Hyperparameters We present the choice of model parameters in Table 6 and training parameters in Table 7. We represent an atom with three features: atom embedding, piece embedding and position embedding. Atom embedding is a trainable vector of size e_{atom} for each type of atoms. Similarly, piece embedding is a trainable vector of size $\mathrm{e}_{\text{piece}}$ for each type of pieces. Positions indicate the order of generation of pieces. We jointly train a 2-layer MLP from the latent variable to predict property scores. The training loss is represented as $\mathcal{L} = \alpha \cdot \mathcal{L}_{\text{rec}} + (1 - \alpha) \cdot \mathcal{L}_{\text{prop}} + \beta \cdot \mathrm{D}_{\text{KL}}$ where α balances the reconstruction loss and prediction loss. For β, we adopt a warm-up method that increase it by β_{stage} every fixed number of steps to a maximum of β_{max}. We found a β higher than 0.01 often causes KL vanishing problem and greatly harm the performance.

Table 6: Parameters in the graph piece variational autoencoder

Model	Param	Description	Value
Common	e_{atom}	Dimension of embeddings of atoms.	50
	$\mathrm{e}_{\text{piece}}$	Dimension of embeddings of pieces.	100
	e_{pos}	Dimension of embeddings of postions. The max position is set to be 50.	50
Encoder	d_{h}	Dimension of the node representations $\boldsymbol{h}_v$	300
	$\mathrm{d}_{\mathcal{G}}$	The final representaion of graphs are projected to $d_{\mathcal{G}}$.	400
	$\mathrm{d}_{\boldsymbol{z}}$	Dimension of the latent variable.	56
Decoder	d_{GRU}	Hidden size of GRU.	200
Predictor	d_{p}	Dimension of the hidden layer of MLP.	200

Table 7: Training hyperparameters

Param	Description	Value
lr	Learning rate	0.001
α	Weight for balancing reconstruction loss and predictor loss	0.1
β_{init}	Initial weight of KL Divergence	0
β_{max}	Max weight of KL Divergence	0.01
kl_{warmup}	The number of steps for one stage up in β	1000
β_{stage}	Increase of β every stage	0.002

Property Optimization We use gradient ascending to search in the continuous space of latent variable. For simplicity, we set a target score and optimize the mean square error between the score given by the predictor and the target score just as in the training process. The optimization stops if the mean square error does not drop for 3 iterations or it has been iterated to the $maxstep$. We normalize the Penalized logP in the training set to $[0, 1]$ according to the statistics of ZINC250K. By setting a target value higher than 1 the model is supposed to find molecules with better property than the molecules in the training set. To acquire the best performance, we perform a grid search with $lr \in \{0.001, 0.01, 0.1, 1, 2\}$, $maxstep \in \{20, 40, 60, 80, 100\}$ and $target \in \{1, 2, 3, 4\}$. For optimization of QED, we choose $lr = 0.01, maxstep = 100, target = 2$. For optimization of Penalized logP, we choose $lr = 0.1, maxstep = 100, target = 2$.

Constrained Property Optimization We use the same method as property optimization to optimize the latent variable. We also perform a grid search with $lr \in \{0.1, 0.01\}$ and $target \in \{2, 3\}$. We select $lr = 0.1, maxstep = 80$ and $target = 2$. For decoding, we first initialize the generation with a submol sampled from the original molecule by teacher forcing. We follow Shi et al. (2020) to first sample a BFS order of all atoms and then randomly drop out the last m atoms with m up to 5. We collect all latent variables which have better predicted scores than the previous iteration and decode each of them 5 times, namely up to 400 molecules. Then we choose the one with the highest property score from the molecules that meet the similarity constraint. For the baseline GA (Ahn et al., 2020), we adjust the number of iterations to 5 and the size of population to 80, namely traversing up to 400 molecules, for fair comparison.

E GRAPH PIECE SAMPLES

We present 50 graph pieces found by our extraction algorithm in Figure 9.

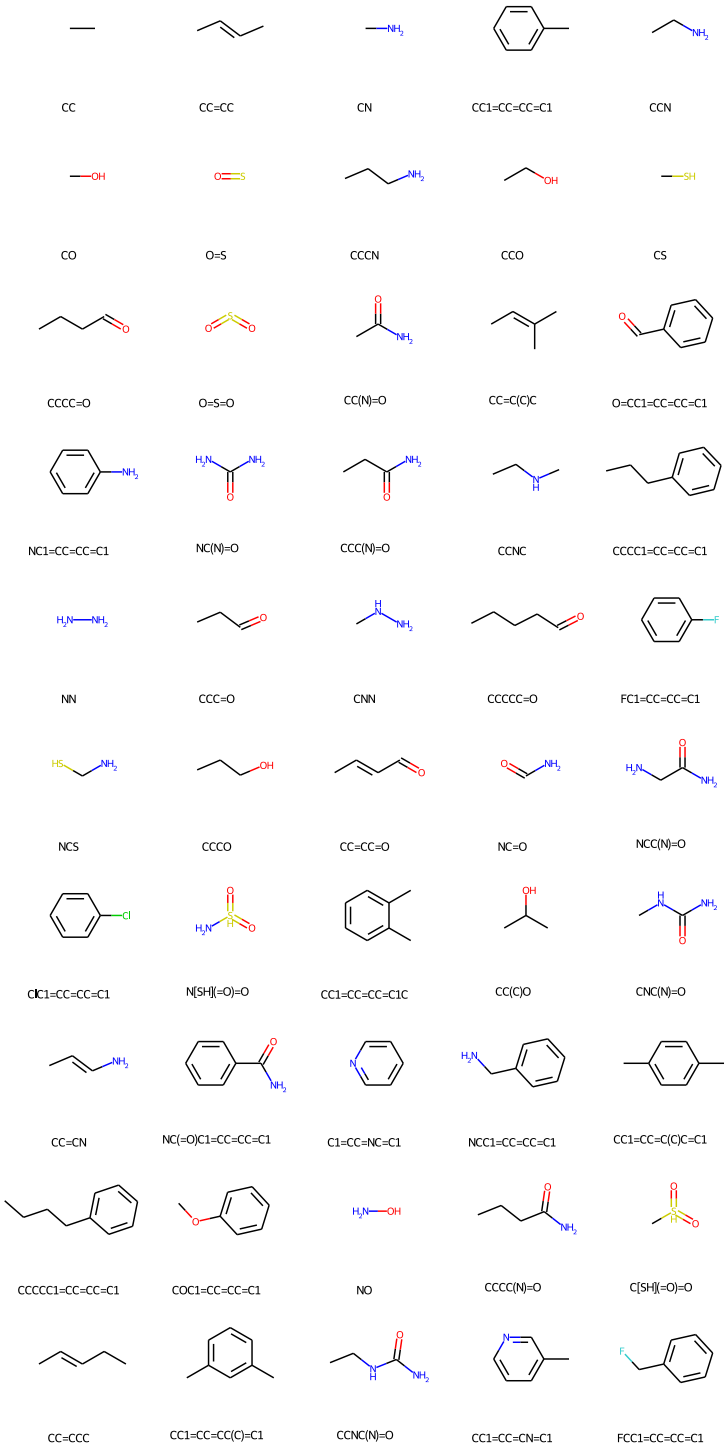


Figure 9: 50 Samples of graph pieces from the vocabulary with 100 graph pieces in total. Each graph piece is labeled with its SMILES representation.

F MORE MOLECULE SAMPLES

We further present 50 molecules sampled from the prior distribution in Figure 10.

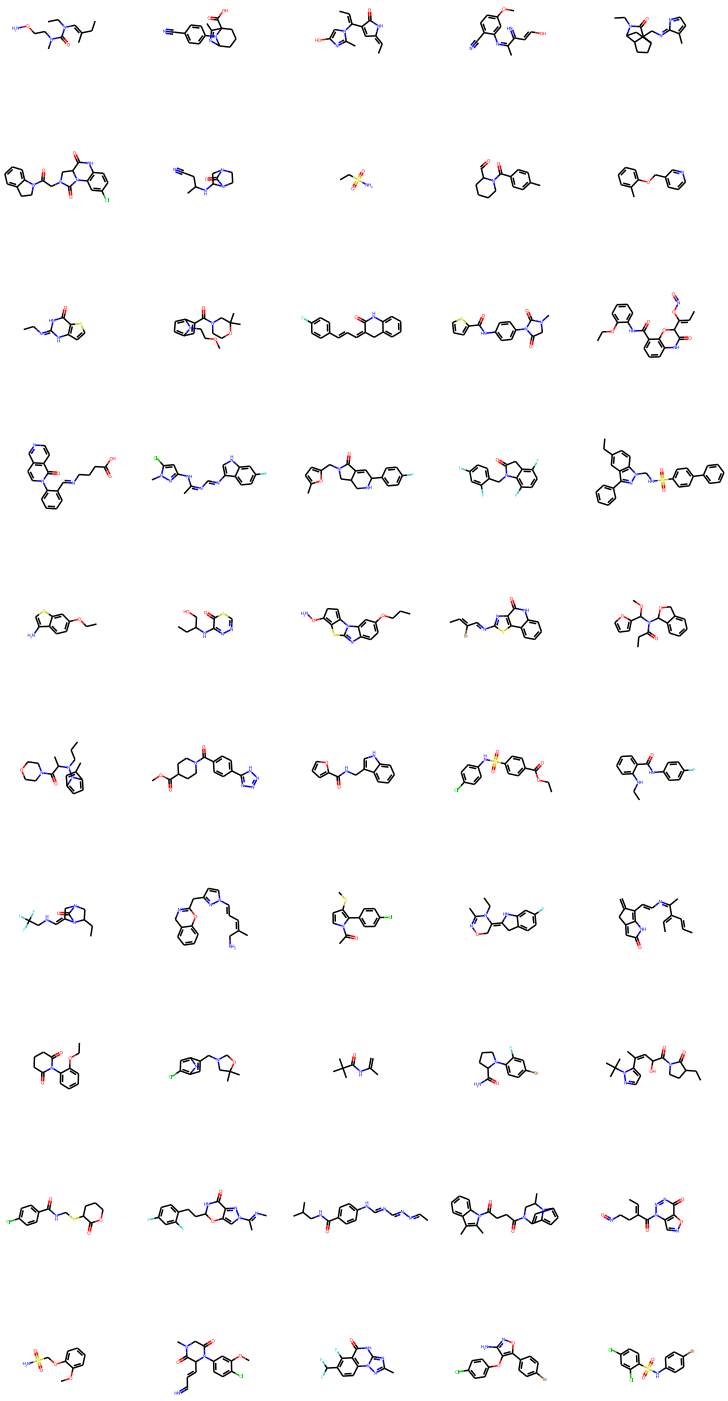


Figure 10: 50 molecules sampled from the prior distribution $\mathcal{N}(0, \boldsymbol{I})$