Interpretable Variational Autoencoder For Molecule Generation And Editing

A thesis submitted in fulfillment of the requirements

for the degree of Master of Technology

by

Gopichand Kotana

14807249

under the guidance of

Dr. Piyush Rai

to the

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

4th June 2019

Page intentionally left blank

Statement of Thesis Preparation

I, **Gopichand Kotana**, declare that this thesis titled, **"Interpretable Variational Autoencoder For Molecule Generation And Editing"**, hereby submitted in partial fulfillment of the requirements for the degree of **Master of Technology** and the work contained herein are my own. I further confirm that:

1. The "Thesis Guide" was referred to for preparing the thesis.
2. Specifications regarding thesis format have been closely followed.
3. The contents of the thesis have been organized based on the guidelines.
4. The thesis has been prepared without resorting to plagiarism.
5. All sources used have been cited appropriately.
6. The thesis has not been submitted elsewhere for a degree.

K. Gopi Chand

Name: Gopichand Kotana

Roll No.: 14807249

Department of Computer Science and Engineering

4th June 2019

Page intentionally left blank

Certificate

It is certified that the work contained in the thesis titled **"Interpretable Variational Autoencoder For Molecule Generation And Editing"** has been carried out under my supervision by **Gopichand Kotana** and that this work has not been submitted elsewhere for a degree.

Dr. Piyush Rai
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur, 208016.

UNDER GRADUATE OFFICE
THESIS SUBMITTED
ON. 04/06/19
I.I.T., KANPUR-208016

4th June 2019

Page intentionally left blank

ABSTRACT

Name of student: **Gopichand Kotana** Roll no: **14807249**

Degree for which submitted: **Master of Technology**

Department: **Department of Computer Science and Engineering**

Thesis title: **Interpretable Variational Autoencoder For Molecule Generation And Editing**

Name of Thesis Supervisor: **Dr. Piyush Rai**

Month and year of thesis submission: $4^{\mathbf{th}}$ **June 2019**

Drug design research is both cost and time extensive. Recent explorations in machine learning have shown promise in making drug design efficient and cost-effective. Molecular structures can be thought of as graphs with atoms representing the nodes and bonds representing the edges of the graph. This has led to a considerable recent interest in graph neural networks for modeling molecular structures, with the goal of generating new candidate molecules for drug design. In this work, we propose a deep generative model with an interpretable latent representation to generate and edit molecules. Our model is built on a variational autoencoder that learns representations of molecules, such that these representations are also directly interpretable and used to control the process of molecular design. This work is a first step in giving control over the generated candidate molecules to take advantage of domain knowledge in drug design research.

Page intentionally left blank

Acknowldgements

I want to express my sincere gratitude to my supervisor, Prof. Piyush Rai of the Department of Computer Science and Engineering, for his patience, motivation, and guidance. I am thankful to him for introducing me to this field in my undergraduate days and guiding me throughout my research, which forged into this thesis. I am grateful to the computer science department of IIT Kanpur for the knowledge bestowed upon me and the excellent facilities provided. Being a student in this department would remain to be one of the greatest experiences of my life.

I thank my fellow labmates: Pawan Kumar Patel, Sunabha Chatterjee, Prateek Samaiya, and Paramkusam Niranjan, for the stimulating discussions and the amazing environment they created that made working fun and seamless. I would also like to thank Gurpreet Singh for his help in implementing the model.

Finally, I would like to thank my parents for their love and support. It is a great comfort to have constant sources of encouragement.

Gopichand Kotana

Contents

Acknowledgements **viii**

1 Introduction **3**

2 Background & Literature Review **5**

2.1 Variational Autoencoders 5

2.2 Representation Learning on Graphs 6

2.3 Deep Generative Models for Molecular Synthesis 10

2.4 Discrete Latent Variables 18

3 Our Model **21**

3.1 DIVA: Model Architecture 21

3.2 Implementation Details 25

4 Experiments & Results **27**

4.1 Dataset 27

4.2 Evaluation Methods 27

5 Conclusion **55**

List of Figures

2.1 The stochasticity is shifted to a random variable independent of parameters of the network through reparametrization. 6

2.2 Hierarchy of structured data decoding space. The figure is taken from [1]. 12

2.3 GraphRNN at inference time. h_i denotes the graph state vector encoded by graph-based RNN. S_i^π is predicted by the edge-level RNN. The figure is taken from [58]. . 14

2.4 GraphVAE model. $G = (A, E, F)$ is the input graph, where, A is the adjacency matrix, E is the edge attribute tensor, and F is the node attribute matrix. $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$ is the probabilistic graph outputted by the decoder. The figure is taken from [48] 17

3.1 Some examples from the vocabulary created out of zinc dataset. 12 out of 780 components pictured. 22

3.2 Junction tree variational autoencoder. The figure is taken from [29]. 23

4.1 Best three molecules in the $20,000$ generated with property scores of 4.27, 4.40, 4.54 respectively 28

4.2 The molecules in valid set that contain the iodine functional group and the identified dimension is active. 31

4.3 The corresponding edited molecules from 4.2 after masking the identified dimension. We can see that the amount of iodine functional group has reduced. 31

4.4 The molecules in test set that contain the iodine functional group. 31

4.5 The corresponding edited molecules from 4.4 after masking 32

4.6 35 random molecules in valid set that contain the nitrile functional group and the identified dimension is active. 32

4.7 35 corresponding edited molecules from 4.6 after masking the identified dimension. We can see that the amount of nitrile functional group has reduced. 33

4.8 35 random molecules in test set that contain the nitrile functional group and the identified dimension is active. 34

4.9 The corresponding edited molecules from 4.8 after masking 35
4.10 40 random molecules in valid set that contain the Ammonia cation functional group and the identified dimension is active. 36
4.11 The corresponding edited molecules from 4.10 after masking the identified dimension. We can see that the amount of Ammonia cation functional group has reduced. 37
4.12 40 random molecules in test set that contain the Ammonia cation functional group and the identified dimension is active. 38
4.13 The corresponding edited molecules from 4.12 after masking 39
4.14 40 random molecules in valid set that contain the sulfone functional group and the identified dimension is active. 40
4.15 The corresponding edited molecules from 4.14 after masking the identified dimension. We can see that the amount of sulfone functional group has reduced. 41
4.16 40 random molecules in test set that contain the sulfone functional group and the identified dimension is active. 42
4.17 The corresponding edited molecules from 4.16 after masking 43
4.18 40 random molecules in valid set that contain the fluorine functional group and the identified dimension is active. 44
4.19 The corresponding edited molecules from 4.18 after masking the identified dimension. We can see that the amount of fluorine functional group has reduced. 45
4.20 40 random molecules in test set that contain the fluorine functional group and the identified dimension is active. 46
4.21 The corresponding edited molecules from 4.20 after masking 47
4.22 40 random molecules in valid set that contain the trifluoromethyl functional group and the identified dimension is active. 48
4.23 The corresponding edited molecules from 4.22 after masking the identified dimension. We can see that the amount of trifluoromethyl functional group has reduced. 49
4.24 40 random molecules in test set that contain the trifluoromethyl functional group and the identified dimension is active. 50
4.25 The corresponding edited molecules from 4.24 after masking 51
4.26 28 random molecules in valid set that contain the triazole ring and the identified dimension is active. 52
4.27 The corresponding edited molecules from 4.26 after masking the identified dimension. We can see that the amount of triazole rings has reduced. 53
4.28 22 random molecules in test set that contain the triazole ring and the identified dimension is active. 54

4.29 The corresponding edited molecules from 4.28 after masking 54

List of Tables

4.1 Best molecule property scores found by each method. The values for other models are taken from [29]. * - without bayesian optimization. 28

4.2 Tanimoto similarity with Morgan fingerprint between the original and edited molecules from the test set. 30

Page intentionally left blank

Chapter 1

Introduction

Graphs are one of the most fundamental data structures in computer science. They are universal as their flexibility allows modeling pairwise information in a lot of domains, such as social networks, molecular structures, protein-protein networks, recommender systems, and question answering [24, 22]. Each entity can be modeled by a node, and interactions between entities are captured by edges. To solve graph based learning tasks, like, discovering new patterns and making predictions, a common approach is to learn a vector representation for each node of the graph. This allows us to use graphs in machine learning by extracting their structural information and not limiting them to knowledge repositories [24].

Deep learning is successfully applied in diverse data domains. A set of models that use deep learning and structured approaches have arisen that allows computations on discrete entities to learn the vector representations and structure of the nodes and edges in a graph [3]. These methods use neighborhood aggregation and message passing, and we shall discuss these in more detail in the next chapter.

Deep generative modeling is the state of the art for most generative tasks showing great promise in producing text [27], images [53] and music [52]. These methods are being explored for combinatorial structures like graphs. Modeling and generating graphs has applications in biology, chemistry, engineering and, social sciences. Generating graphs is difficult because they are discrete structures with non-local dependencies between edges and arbitrary connectivity,i.e., there is no specific way to linearize their construction [58, 48].

In this work, we take up an important application of graph generation which is molecular generation. We study the current approaches which use deep generative models to solve this problem. We identify difficulties of the task and limitations of the current models. We propose a discrete variational autoencoder(DIVA) by extending the current state of the art VAE based model, JTVAE [29], to make it interpretable and allow molecular generation and editing.

Drug discovery is a costly and time taking process, automating it (or, parts of it) saves tremendous time and resources. Molecular discovery is a difficult task because of the colossal amount of potential drug-like compounds which is estimated to be in the range of 10^{23} and 10^{60} [16]. Classical methods take a combinatorial approach, and that can lead to the generation of a large number of unstable and invalid molecules [16]. The two popular and successful deep generative models are variational autoencoder(VAE) [32] and generative adversarial network(GAN) [19]. Latent representation is low dimensional representation of data which is learned by the model. VAEs have explicit and meaningful latent representation as opposed to GANs, which have implicit latent representation [10]. Hence, we prefer models based on VAEs in this work to impose structure on the generative process.

Chapter 2

Background & Literature Review

Our work on molecular generation is based on applying deep generative models designed for graphs on graphical representations of molecules. We design an interpretable model by discretizing the low dimensional latent embeddings learned from the data. We divide this chapter into four sections to review the prior work. In the first section, we give a brief description of variational autoencoders. In the second section, we study the advancements in representation learning on graphs, especially models based on deep neural networks. These models are used as building blocks in the generative models to generate graphs. In the third section, we review the recent progress in graph generation and generation of molecules. We look at different molecular representations and model architectures and discuss their pros and cons while justifying our model choice. In the fourth section, we study how non-differentiable discrete variables can be used in neural networks.

2.1 Variational Autoencoders

A standard autoencoder compresses and decompresses data, consequently learning a low dimensional representation of the data. While autoencoders produce a good low dimensional representation for the data, the decoder of a standard autoencoder cannot produce a meaningful output from a random low dimensional vector.

Variational Autoencoders(VAEs) [32] allow for a meaningful generation of data from arbitrary noise. They achieve this by imposing a prior on the latent representation. VAEs consist of an encoder, $q_\phi(z|x)$, and a decoder, $p_\theta(x|z)$. Both the encoder and decoder are modeled by neural networks designed to operate on a specific input. The loss function minimized is,

$$\mathcal{L} = \sum_{i=1}^{N} \mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + \mathbb{KL}(q_\phi(z|x_i)||p_\theta(z))$$

The first term of $\mathcal{L}$ is reconstruction loss. The second term is, a regularization term which forces $q_\phi(z|x)$ to be close to the prior $p_\theta(z)$. In a standard VAE, uniform normal distribution is chosen as the prior. Differentiating $\mathcal{L}$ with respect to both ϕ and θ for optimization is problematic because of the stochasticity present in the network.

Reparameterization trick is used to dissociate the stochasticity from the parameters being estimated. The relationship between the parameters μ and σ with the Gaussian random variable Z is converted to a deterministic dependency by using a random variable which independent of the parameters of the network. This allows us to take derivates with respect to μ and σ, allowing for gradient based optimization through backpropagation.

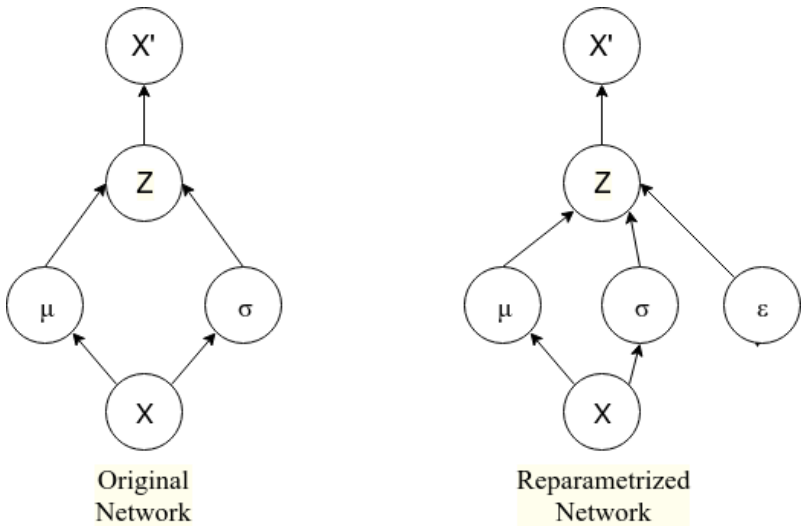

Figure 2.1: The stochasticity is shifted to a random variable independent of parameters of the network through reparametrization.

2.2 Representation Learning on Graphs

Relations between objects can be represented by graphs. Some noteworthy examples are social networks, internet, citation networks, and molecules. Node classification, link prediction, community detection, computing network similarity, are some of the classical machine learning tasks on graphs.

To solve a machine learning task on a graph, we need vector representations of a graph and it's attributes. In this section, we see several methods that use topological information, node and edge feature vectors of the graph, to learn these representations.

We discuss methods of node embedding that represent nodes as low dimensional vectors. For a graph $G = (V, E)$, V represents the nodes, and E represents the edges. A graph is represented by an adjacency matrix A. [2] proposed a simple inner product based model. The inner product

between node vectors of two nodes n, m, $< z_n, z_m >$, is expected to capture the presence of an edge, A_{nm}. The objective function minimizes the mean squared error between the inner product and edge presence over all the edges. This method can be naturally extended to weighted graphs where A_{nm} is the weight of the edge. This model only considers direct local connections. GraRep [9] circumvents this problem by using different transition probability matrices A^k and concatenating the node vectors obtained for each k. This way the node embedding vectors, Zs, capture multi-hop similarity. HOPE [42] is another method aimed at preserving higher order proximity. They use existing proximity measures to count statistics on the graph, and utilize them to obtain embedding vectors that preserve these statistics. LINE [50] is another approach that preserves first order and second order proximity. It does so by minimizing the KL divergence between the empirical distribution and joint probability defined between vertices. The first and second similarities are explicitly factorized.

A class of models based on random walks have been proposed for this task. These methods are easily scalable. DeepWalk [43] is one of the earliest works that introduce truncated random walks to find structural similitude. The algorithm generates a set of random walks by randomly choosing the initial node for each walk. The algorithm then treats these walks as sentences in natural language processing and apply skipgram model [41] on these walks to obtain embeddings for each node. Node2vec [21] uses biased random walks to control the balance between exploring the local neighborhood and moving to farther parts in the graph. They do this by employing breadth first search(BFS), and depth first search(DFS) style random walks. A BFS kind of walk leads to a local view of the neighborhood, and a DFS kind of walk leads to a macroscopic view over the graph. The choice of which node embedding to use is application specific.

Deep learning is largely successful in a number of diverse applications. Recently, new architectures for graph structured data based on deep learning have surged, borrowing ideas from convolution networks, recurrent networks, and autoencoders. First mention of graph neural network(GNN) was in [20, 47]. They propose a superviesed recurrent neural network model. Each node n has a state vector $h_n \in \mathbb{R}^d$ which represents n. If we know the labels of it's neighboring nodes and edges, the state vector can be updated using a local trasition function f_w. $h_n = f_w(l_n, l_{co[n]}, h_{ne[n]}, l_n e[n])$ where, l denotes the label, h denotes the representation vector, $co[n]$ denotes the connections of the node n, $ne[n]$ denotes the neighbors of the node n. Using h_n and l_n we can update the output at the node using a local output function, g_w. $o_n = g_w(h_n, j_n)$. They employ a iterative scheme for computing the state. It can be compactly written for all nodes at once as, $h(t+1) = F_w(h(t), l)$. f_w and g_w are implemented by feed forward networks. F_w is assumed to be a contraction map which ensures that h_n converges to a fixed point. Weights are updated by resilient back propagation until

the output reaches a desired accuracy or, if some other stopping criterion is met.

First variant of graph convolution was discussed in [8]. [8] extends the convolutions defined on grid like data to graphs. They develop graph convolution network(GCN) based on spectral graph theory. [15] improves upon this work by approximating filters in the spectral domain using Chebyshev polynomials.

[33] takes an approach which is similar to [15], but approximates the filters linearly instead of explicitly parameterizing using Chebyshev polynomials. The propogation rule they obtain for activations in the $(l+1)^{th}$ layer is,

$$H^{(l+1)} = \tau(\tilde{D}^{-\frac{1}{2}} A \tilde{D}^{\frac{1}{2}} H^{(l)} W^{(l)}) \text{ where, } \tilde{A} = A + \mathcal{I}_N,\ \mathcal{I}_N \text{ is the identity matrix}$$

$\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, W^l is the weight matrix of lth layer, and τ is an activation function. The activations of the last layer can be considered as the node vectors. This work tackles node classification in a semi-supervised setting. The outputs are computed by applying softmax function on the final activations. The loss function optimizes a cross entropy error over all the labeled examples. Variational Graph Autoencoder(VGAE) [34] is a variational autoencoder to obtain node embeddings of a single graph. The encoder is a GCN [33] and the decoder is a simple inner product between the node embeddings. FastGCN [11] has been recently proposed to cut down the computational cost of the training phase of GCNs.

GraphSAGE [23] is a node embedding method with inductive capability. It can produce embeddings for unseen nodes by aggregating activations of nodes in its local neighborhood. For each node $v \in V$, the propagation rule for activations in the $(l+1)^{th}$ layer is,

$$h_{N(v)}^{(l+1)} = AGGREGATE_{l+1}(h_u^l, \forall v \in N(v))$$

$$h_v^{(l+1)} = \tau(W^{(l+1)} CONCAT(h_v^l, h_{N(v)}^{(l+1)}))$$

Where the notation is similar to GCNs and $AGGREGATE$ function is any aggregator function that can operate on a set of unordered vectors. They use mean, pooling, and LSTM aggregators. The information is aggregated from the local neighborhood and as the iterations progress, nodes gain more information from further reaches of the graph. They employ a graph based loss function that encourages nearby nodes to have similar representations. GCNs require the complete adjacency matrix to be present which can be a memory bottleneck, GraphSAGE uses mini-batching to mitigate this issue. In this work, they also show that GCN [33] can be formulated as a node aggregation method to convert it to an inductive framework.

Although GraphSAGE and GCNs present an effective way to learn node embeddings through neighborhood aggregation, they are only a few layers deep. Each layer has a separate weight matrix to be learned. Hence, training deep models of these architectures is difficult [24] as they suffer from overfitting or, vanishing or exploding gradients. Gated graph neural network[36] is based on the recurrent neural network presented by [47] for graph structured data. [36] uses Gated Recurrent Units(GRUs) and backpropagation to remove the constraints imposed in [47] to ensure convergence. The propagation of information in [36] happens in two steps. First a message is computed based on the activations of the neighbors of a node, $m_v^t = W \sum_{v \in N(v)} h_v^{t-1}$. Then, the activation of a node is updated by passing the computed message vector of the node along with its previous activation value, $h_v^t = GRU(h_v^{t-1}, m_v^t)$. We use this architecture for node and graph embeddings in the encoder of our model. [36] also defines a gated graph sequential neural network, which is sever gated graph neural networks operating in sequence to produce an output sequence.

[17] demonstrates that, all the node embedding methods that use message passing and node aggregation can be formulated into a single framework. They call these neural networks, Message Passing Neural Networks (MPNNs). Each node's messages are updated as,

$$m_v^{(t+1)} = \sum_{u \in N(v)} M_t(h_v^t, h_u^t, e_{vu})$$

where M_t is a message passing function, h_v, h_u, are the activations of nodes v and u respectively, and e_{uv} is the edge feature of the edge (u, v). The messages are used along with previous hidden state, h_v^t, to compute the new hidden state, $h_v^{(t+1)}$, as,

$$h_v^{(t+1)} = U_t(h_v^t, m_v^{(t+1)})$$

where U_t is a vertex update function. The representation for the entire graph can be computed using a readout function R. The graph representation, $\tilde{y}$, is, $\tilde{y} = R(\{h_v^T | v \in V\})$.
Different choices of message function, M_t, vertex update function, U_t, and readout function, R, leads to different state of the art neural network models on graphs.

2.3 Deep Generative Models for Molecular Synthesis

2.3.1 Molecular representation

A molecule can be considered as a molecular graph with atoms as nodes and bonds of the molecule as edges of the graph. This representation is a success in chemical property prediction and generative tasks. By representing molecules in this way, we lose bond length information and 3D conformation of the molecules that we would get from the 3D structure, which could be critical for some tasks [16].

SMILES

The simplified molecular-input line-entry system (SMILES) [56] is a textual representation of molecular graphs with ASCII characters as the alphabet. Each SMILES string corresponds to a unique molecular graph but each molecule can have more than one SMILES representation. For example, ethanol can have any of the following representations, CCO, OCC and, C(O)C. We can use a canonicalization algorithm to make them unique, these are called canonical SMILES. SMILES is the most common representation as most datasets store molecules as SMILES strings. Since SMILES is a character by character text encoding, it allows us to leverage the extensive research done on sequence generation for generating molecules.

Graph based representation

The surge of deep learning methods that can directly work on graphs has allowed researches to use molecular graphs directly as input for their models. A molecular graph is defined as $G = (V, E)$ where the set of vertices V denote atoms and the set of edges E denote the bonds between atoms. Graphs are usually represented by adjacency matrices. We can also incorporate node and edge features into models which directly work on graphs. The node features can consist of the type of atoms whereas the edge features can represent the type of bond [46].

There are several limitations to SMILES, which led research in molecular generation to be mainly focused on using graph based representations. Generative models, trained on SMILES for molecular generation, suffer from producing a large number of invalid molecules. This is partly due to the SMILES syntax which contains elements that are required to occur in pairs(like, parenthesis) [14]. DeepSMILES [14] tries to circumvent this problem by using only closing parentheses and, a single symbol at the ring closing location instead of ring closure symbols which are also paired.

SMILES strings are not intended to capture molecular similarity,i.e, two molecules with similar

structure can have very different looking canonical SMILES [29]. This makes it difficult for models that use variational autoencoders to learn smooth latent molecular embeddings. Molecular validity and other important chemical properties of a molecule are easier to represent on a molecular graph than in a line notation. It is shown in [5] that latent representations learned using canonical SMILES seem to be closely related to the grammar rules of the canonical smiles than the molecular graphs. [6] suggests training models on all possible SMILES strings of a molecule, to mitigate this issue.

The two representations discussed above are the most widely used but are not the only representations possible. We can also represent molecules in 3D, using nuclear coordinates [16]. Researchers also tried using images of 2D representation of molecules to exploit the recent successes of CNNs on images. IUPAC Chemical Identifier (InChI) is another line notation similar to SMILES [25].

2.3.2 Models for Molecular Generation

In this section, we discuss different model architectures that are used to generate molecular graphs.

Deep Generative models based on SMILES

CVAE

Chemical VAE described in [18] is one of the earliest attempts to automate molecule generation using deep generative models. The data is formalized into canonical SMILES strings which serve as an input to a character based variational autoencoder with recurrent neural networks for encoding and decoding. CVAE is similar to the RNN-based variational autoencoder proposed by [7] for sentence modeling. The model is trained jointly with a property prediction task to correlate the latent representations with target properties.

The objective of this work was to convert discrete representations of molecules into smooth continuous representations. The molecular space is large, discrete, and unstructured which makes optimization difficult in that space. The continuous latent space allows for Bayesian optimization to explore molecular representations with desired properties.

This model produces a large number of invalid SMILES because of the nature of decoding(character by character) and the inherent problems present in the SMILES syntax. The variational autoencoder isn't explicitly forced to decode only valid strings. Enforcing this condition would force the variational autoencoder to learn what is a valid SMILES, which makes the problem significantly difficult. The invalid strings are handled in a post-processing step. They also tried using InChI representations, but they give significantly worse results. They hypothesize that the complex syntax

of InChI is the reason behind the bad results.

GVAE

Grammar variational autoencoder [35] resolves some of the problems present in CVAE. Instead of forcing the model to learn what a valid SMILES string is, it is better to present it with an explicit grammar. They achieve this by encoding the discrete structures into a parse tree of a context free grammar. The encoder and decoder of the variational autoencoder act directly on the parse trees. Besides an improvement in the number of valid SMILES strings generated, it learns a better, more coherent latent space,i.e, two points close by in the latent space produce similar molecules.

Parse trees are generated by the decoder which makes every output valid within the grammar specified. While this ensures syntactic validity, the output produced can be semantically invalid. Hence, the model doesn't produce 100% valid molecules, some molecules produced are unstable or chemically invalid. The SMILES language has non context free features which can not be handled by this architecture.

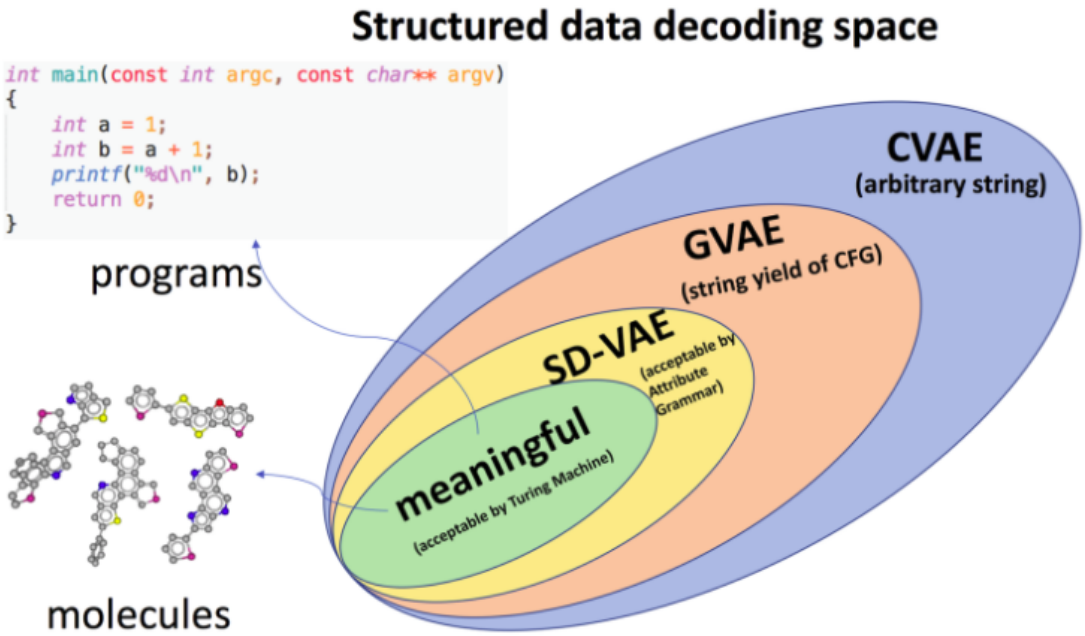

Figure 2.2: Hierarchy of structured data decoding space. The figure is taken from [1].

SD-VAE

Syntax-Directed variational autoencoder [13] uses attribute grammars to ensure semantic validity. Attribute grammars allow us to attach semantics to parse trees generated from context free grammar. This is inspired by the theory in compiler design. Instead of doing a semantic check after

the generation they propose to add online guidance to the generator. They achieve this with the same computational cost as CVAE and GVAE.

Sequential models for molecular graph generation

A graph with n nodes has $n!$ arbitrary node orderings. Each ordering leads to a different adjacency matrix representing the same graph. Modeling is hard because of the high representation complexity[58]. Every graph can have a different number of nodes and edges; generative models should be competent to include this variability.

GraphRNN

As the name suggests, GraphRNN [58] is a deep autoregressive model that learns the distribution of graphs. The model treats the graph generation process to be a series of node and edge formations conditioned on the graph generated up to that point in time. [58] also introduces a new evaluation method to measure distances between

GraphRNN is a hierarchical model which utilizes a 'graph-level RNN' to generate a new node conditioned on the current state of the graph, and an 'edge-level RNN' connects this newly generated node to the existing graph by generating edges. A mapping is defined from graphs to sequences. For a graph $G \sim P(G)$ with n nodes and ordering π,

$$S^{\pi} = f_s(G, \pi) = (S_1^{\pi}, \ldots, S_n^{\pi}), \text{ where, } S_i^{\pi} \in \{0,1\}^{i-1}, i \in \{1, \ldots, n\}$$

$$\text{And, } S_i^{\pi} = (A_{1,i}^{\pi}, \ldots, A_{i-1,i}^{\pi}), \forall i \in \{2, \ldots, n\}$$

where A^{π} is an adjacency matrix from the set of adjacency matrices, A^{Π}($A^{\Pi} = \{A^{\pi} | \pi \in \Pi\}$) corresponding to graph G. S^{π} maps to a unique graph G for an undirected graph.

Instead of learning the graph space, $P(G)$, we sample π and learn $P(S^{\pi})$,

$$P(S^{\pi}) = \prod_{i=1}^{n+1} P(S_i^{\pi} | S_1^{\pi}, \ldots, S_{i-1}^{\pi}), \text{ where } S_{n+1}^{\pi} \text{ is set to EOS}$$

$$P(S_i^{\pi} | S_{<i}^{\pi}) \text{ is further decomposed to } P(S_i^{\pi} | S_{<i}^{\pi}) = \prod_{j=1}^{i-1} P(S_{i,j}^{\pi} | S_{i,<j}^{\pi}, S_{<i}^{\pi})$$

Each component of the products is parameterized by a recurrent neural network (RNN) that consists of a state transition function and an output function. Instead of training the model to generate graphs for all possible node orderings, they use breadth first search(BFS) orderings,

$S^{\pi} = f_s(G, BFS(G, \pi))$. This significantly reduces the number of sequences because the BFS function is many to one. They also hypothesize that BFS ordering makes learning easier.

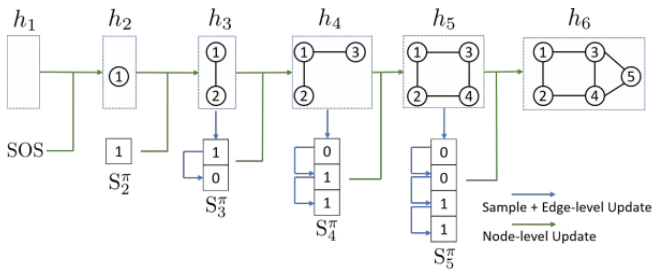

Figure 2.3: GraphRNN at inference time. h_i denotes the graph state vector encoded by graph-based RNN. S_i^{π} is predicted by the edge-level RNN. The figure is taken from [58].

While this method is suitable for producing large graphs and incorporate features like community structure, we need methods that are custom made for small molecular graphs where validity is not pre-established. We need models which allow us to search for molecular graphs of required target properties and produce a large amount of chemically valid and synthesizable molecules. The models that build molecules sequentially have an undesirable property of going through chemically invalid intermediaries and leave the chemical validation check until the end [29].

Sequences like, a sequence of decisions in the graph generation process has no natural ordering. [54] show that the sequence order matters when the underlying model is learned. They also show that while some sequences may not have a natural ordering, there could exist an optimal ordering of sequences for which the underlying model performs the best.
The joint probability is learned by decomposing it using the chain rule. The chain rule can be used without making any assumptions but it imposes an ordering. Hence, if we have a powerful model which can successfully learn long range correlations any ordering is expected to work well. [54] show that models like Long short-term memory(LSTM) which are universal approximators and successful at retaining long range correlations are also susceptible to the ordering of sequences.

A modular sequential model for graph generation

The model proposed by [37] that defines a probability distribution over every possible outcome at each step of the graph generating decisions. The process of graph generation is decomposed into three stages; determining if a new node is to be added to the current graph, determining if an edge has to be added from the generated node to the current graph, deciding the node in the current graph to which this edge is added. This model performs better than models that do not use graph structured representations.

They make use of neural networks that operate on graph structured data to obtain node vectors, and from them, the graph representation. This graph representation is used in different modules which make the decisions required for the graph generation. A simple multilayer perceptron (MLP) model along with softmax is used to obtain the type of node that is generated in the current iteration. Another MLP along with the new node's initialized representation is used to decide if an edge has to be added. Then we provide a different module with all the node embeddings, for it to return a distribution on the edges. The new node's representation is initialized using the current graph state to disambiguate between two nodes of the same type. This model is conditioned on the full history of the generation sequence.

The model defines a joint probability $P(G, \pi)$ over graphs and edge orderings. We are in interested in the marginal, $P(G)$. It requires us to sum over all the permutations which isn't intractable for any reasonable sized graphs. One way to evaluate this marginal is by using Monte Carlo estimates based on importance sampling.

$$P(G) = \sum_{\pi} P(G, \pi) = \mathbb{E}_q(\pi|G)\left[\frac{P(G, \pi)}{q(\pi|G)}\right], q(\pi|G) \text{ is any proposal distribution over permutations}$$

In [37] they always use a fixed ordering or uniform random ordering in their experiments. They also experiment on small graphs where training can be done on all the permutations. By the experiments on small graphs, they show that the canonical orderings need not be the ordering that gives the lowest negative log likelihood. Hence, there is a room for learning an ordering.

NeVAE

To impose semantic validity, GVAE makes use of masking. NeVAE [46] employs this for general graph construction. The model is defined as follows,

For a graph G, V is the set of nodes, E is the set of edges, F denotes the node features, and Y denotes the edge weights.

prior: $p(z_1, \ldots, z_n); p(z_u) \sim \mathcal{N}(0, I) \forall u \in V \ |V| = |F| = n \sim Poisson(\lambda_n)$

Encoder: $q_\phi(z_1, \ldots, z_n|V, E, F, Y)$

Each node has one latent variable, hence the latent representation of graphs can alter in size.

For each node u, $q_\phi(z_u|V, E, F, Y) \sim \mathcal{N}(\mu_u, diag(\sigma_u)); [\mu_u, diag(\sigma_u)] = \phi^{enc}(c_u(1), \ldots, c_u(K))$ $c_u(k)$ is an aggregator function of u from all the nodes that are less than k hops away. It is parameterized similar to the message passing neural networks.

Decoder: $p_\theta(E, F, Y|z_1, \ldots, z_n)$

First, the number of nodes are sampled from a Poisson distribution. Then for each node, a latent vector is drawn from the prior. Each latent vector corresponding to a node is passed through a

neural network to obtain the node feature using softmax. All the node latent representations are passed through another neural network to sample the number of edges. After that, unnormalized log probability vectors(logits) are generated for each possible edge. Edges are sampled from a softmax distribution based on logits and a mask based on the previously generated edges. And for the generated edge, we learn it's weight through another mask and logits conditioned on the weights of the previously generated edges.
Training this model by minimizing the evidence lower bound(ELBO) on a collection of molecular graphs requires specifying an edge ordering. They use breadth first search to obtain the orderings.

We have seen that the choice of ordering matters to the underlying models. Any ordering we choose will impose an artificial ordering. It has been shown that restricting equivalence classes helps in the scalability of models [55]. GraphRNN and NeVAE choose BFS orderings. The BFS ordering scheme is many to one and this drastically improves the scalability of GraphRNN by reducing the number of equivalence classes. All this evidence suggests that it might be possible to treat the ordering as a random variable whose distribution is learned.

CGVAE

Constrained Graph Variational Autoencoders for molecule design [38] builds on the ideas of [37]. The generation involves initialization of a set of possible nodes which jointly specify the generated graph. The generation involves two steps, *focus* and *expand*. A node is chosen as a focus node and edges are added from it to other nodes. The focus nodes are selected from a BFS queue (Random initial node) deterministically. The *expand* step produces a distribution on possible edges and ensures that constraints are satisfied by using binary masks. The generation is conditioned on the current graph state computed by a standard gated graph neural network. By conditioning only on the current graph state, they make the generation independent of its history, and hence, circumvent the problems of generation trace replication and training very deep architectures. The encoder is a standard gated graph neural network which embeds each node of the graph into a latent space parameterized by the mean and standard deviation of a diagonal normal distribution.

VAE based models for full molecular graph generation

GraphVAE

We have seen that the methods we discussed in the previous section require us to linearize the graph generation process. [48] proposes a variational autoencoder model which operates on the

entire graph and generates a probabilistic graph in a single step from the latent representation. They use the adjacency matrix as the representation for graphs. This model can be used only to generate small graphs and does not scale up. The encoder can be any graph embedding method. They use edge conditioned graph convolutions in the encoder. The decoder produces a probabilistic fully connected graph. Nodes and edges are modeled as Bernoulli variables and attributes as Multinoulli variables.

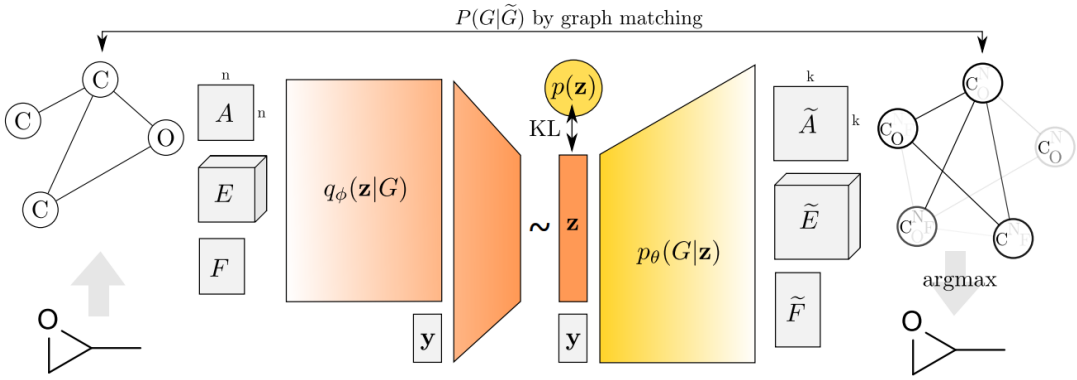

Figure 2.4: GraphVAE model. $G = (A, E, F)$ is the input graph, where, A is the adjacency matrix, E is the edge attribute tensor, and F is the node attribute matrix. $\tilde{G} = (\tilde{A}, \tilde{E}, \tilde{F})$ is the probabilistic graph outputted by the decoder. The figure is taken from [48]

Given a graph G and it's probabilistic reconstruction $\tilde{G}$, we need to compute the likelihood $P(G|\tilde{G})$ for us to evaluate the reconstruction loss in the VAE loss. The adjacency representations are not permutation invariant as the model doesn't impose any order of nodes. Hence, they use an approximate graph matching algorithm to find the correspondence between nodes of G and $\tilde{G}$. The graph matching algorithm is of order $O(k^4)$ where k is the maximum number of nodes in a graph that this model can produce. The computationally expensive graph matching algorithm along with the memory requirements restrict this model to be used for the generation of small graphs. This method requires us to fix the maximum number of nodes that it can generate to a small number (experiments up to $k = 38$ are performed in [48]).

In an approach like GraphVAE, it's difficult to preserve certain global constraints like graph connectivity. [39] proposes a regularization framework to ensure the semantic validity of the graphs produced. They use adjacency matrices for graph representation and add additional penalty terms to loss function to ensure the preservation of all the constraints. In this work, they impose constraints to make sure there are no nodes without edges, to meet the valence criterion of the atoms, to ensure graph connectivity, and for only nodes that are compatible with each other to have an edge between them.

2.4 Discrete Latent Variables

Neural network based models require parameters relationship with an objective function to be smooth and continuous to obtain exact gradients for training. Typically gradient descent methods, especially backpropagation, is used universally to tackle the optimization problem in neural networks. Backpropagation through discrete variables is usually not possible due to their non-differentiability.

The idea of having discrete representations in a neural network is not new. Boltzmann machines [26] are a popular class of models that used discrete latent variables. The advent of backpropagation has led to abandoning the discrete representations with hard threshold outputs and moving to continuous representations whose gradients with respect to the parameters can be exactly computed [4].

Discrete representation in a probabilistic model naturally captures discrete classes in a dataset [45]. Discrete embeddings explicitly deconstruct the data to provide us with concepts present in the data. There are several advantages of using stochastic neural networks with binary stochastic neurons. They give rise to sparse representations and act as a regularizer [4]. Sparse representations can be useful in conditional computation; we can use trained sparse gating units to select the part of a model that needs to be activated for a given example. This greatly reduces the number of computations required, allowing us to train large networks. Discrete variables are also used in recurrent neural network models to take decisions about temporal events at different time scales [12].

[4] describes gradient estimators for stochastic binary neurons. A stochastic binary neuron outputs one or zero by sampling from a uniform distribution between zero and one and comparing it to a probability parameter. The probability parameter is obtained from a real valued input, a_i, by passing the real valued input, a_i through the sigmoid function. Hence, the probability parameter is $sigm(a_i)$, where, $sigm(a_i) = \frac{1}{1+exp(-a_i)}$. If the sampled output from uniform distribution, z_i, is greater than $sigm(a_i)$ the stochastic binary neuron outputs one, otherwise, it outputs zero. Compactly we can write it as, $h_i = \mathbb{1}_{z_i > sigm(a_i)}$, where $z_i \sim U[0,1]$ and $\mathbb{1}$ is the indicator function. The loss function, L, is dependent on the output h_i. We would like to estimate how a change in a_i influences L. This isn't straightforward due to the presence of the stochastic transformation of a_i into h_i. Estimating $\frac{\partial L}{\partial a_i}$ is necessary to backpropagate through the network that produces a_i. The REINFORCE algorithm [57] provides an unbiased estimator and [4] provides the constant to be substituted in the expression for a binary stochastic neuron to obtain minimum variance of the estimate. Even the estimate obtained from variance adjusted REINFORCE algorithm is quite

high and training fails when the neural network is deep. [4] proposes a simple biased estimator called straight through estimator which simply replaces $\frac{\partial L}{\partial a_i}$ by $\frac{\partial L}{\partial h_i}$, where $h_i = \mathbb{1}_{z_i > sigm(a_i)}$. They also propose a variant which multiplies the derivative of sigmoid to $\frac{\partial L}{\partial h_i}$ turning the estimate of $\frac{\partial L}{\partial a_i}$ into $\frac{\partial L}{\partial h_i}(1 - sigm(a_i))(sigm(a_i))$.

Consider a computation graph with θ as the parameters and f is the cost function. The objective is to minimize the expected cost through gradient descent. The REINFORCE estimator uses the identity, $\nabla_\theta p_\theta(z) = p_\theta(z)\nabla_\theta \log p_\theta(z)$. Using this we can estimate $\nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)]$ as,

$$\nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)] = \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)\nabla_\theta \log p_\theta(z)]$$

This expectation can be estimated using Monte Carlo estimator. This method does not require $f(x,z)$ to be differentiable.

If a deterministic transformation, $g(\epsilon,\theta)$, exists such that it gives a sample from $p_\theta(z)$ by transforming a sample ϵ from a fixed distribution, $p(\epsilon)$, independent of θ, we can reparametrize the distribution $p(z)$ to simplify the estimation of $\nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)]$. This is known as the reparameterization trick. Using the reparameterization trick we can estimate $\nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)]$ as,

$$\nabla_\theta \mathbb{E}_{z\sim p_\theta(z)}[f(x,z)] = \mathbb{E}_{\epsilon\sim p(\epsilon)}\nabla_\theta[f(x,g(\epsilon,\theta))]$$

The expectation can be estimated using Monte Carlo estimator. This method requires $f(x,z)$ to be differentiable. Also, only a limited number of distributions can be reparameterized.
The variance of the gradients obtained from REINFORCE method is impractically high. There are methods which work on reducing the variance of the REINFORCE estimator. Even with reduced variance, to capture a good estimate of the variance, we require K samples for a K dimensional latent representation [45]. Reparameterization trick can estimate the direction of gradients even by using a simple sample. Hence, reparameterization trick is preferred wherever it is applicable.

Recently, [40] and [28] independently proposed continuous relaxations of discrete distributions and optimizing a consistent objective. Although we use binary variables for our latent representation, we describe the work presented in [40] and [28] in its general form, which is a continuous distribution that anneals into a discrete distribution over categorical variables. The continuous distribution proposed is called Gumbel-Softmax distribution by [28] and concrete distribution by [40]. A simple reparameterization exists for this distribution. This property is important for gradient estimation as we have discussed above.

Below we describe how the Gumbel-Softmax distribution can approximate samples from a categorical distribution.

Consider one-hot representation as the representation of a categorical variable z with k classes. Let $\pi_1, \pi_2, \ldots, \pi_K$ be the class probabilities of z. The Gumbel-Max trick allows sampling from the categorical variable z to be done in a efficient way in the following manner,

$$z = onehot(\arg\max_i [g_i + \log \pi_i])$$

where, $g_1, \ldots, g_k$ are independent and identically distributed samples drawn from Gumbel$(0, 1)$. Sampling g_i i.i.d for each i can be done as follows: sample $u_i \sim$ Uniform$(0, 1)$ and compute $g_i = -\log(-\log u_i))$. The derivative of arg max is undefined at the boundary of state changes and zero everywhere else. Hence, softmax function is used to approximate arg max. Softmax is a continuous and differentiable approximation. To sample a vector y from this approximation at temperature $\tau \in (0, \infty)$, sample $g_i \sim$ Gumbel$(0, 1)$ for each i, and set,

$$y_i = \frac{\exp\left(\frac{\log(\pi_i)+g_i}{\tau}\right)}{\sum_{j=1}^{k} \exp\left(\frac{\log(\pi_j)+g_j}{\tau}\right)} \text{ for } i = 1, \ldots, k$$

The Gumbel-Softmax is a smooth distribution for any temperature that is greater than zero. The softmax distribution smoothly approaches the categorical distribution as $\tau \to 0$. As $\tau \to \infty$ the samples from Gumbel-Softmax become uniform.
The probability density of the Gumbel-softmax distribution is given by,

$$p_{\pi,\tau}(y) = \Gamma(k)\tau^{k-1}\left(\sum_{i=1}^{k} \frac{\pi_i}{y_i^{\tau}}\right)^{-k} \prod_{i=1}^{k} \left(\frac{\pi_i}{y_i^{\tau+1}}\right)$$

When we want to train a variational autoencoder using discrete variables as latent representation, we replace the prior and posterior of the discrete random variable in evidence lower bound(ELBO) with the continous relaxation offered by Gumbel-Softmax distribution. This makes the entire model differentiable and allows gradient based optimization. By limiting the number of classes to two, we can easily train binary latent vectors using this method. During the training phase, the temperature can be fixed or annealed with a certain schedule. Annealing is vital because, at low temperatures, the samples are close to one-hot, but the gradients have large variance, whereas, at a higher temperature, the variance is small, but the samples become smooth.

Chapter 3

Our Model

In this section, we describe the model we propose and explain the motivations behind the architecture choices. We present a discrete interpretable variational autoencoder(DIVA) for molecule generation and editing. The generation of new molecules takes place from a discrete representation. We give partial control over the functional groups being generated and show how this can be useful in translating molecules into molecules without undesired chemical subcomponents. DIVA is an extension of junction tree variational autoencoder [29]. We make part of the latent space discrete to impart interpretability into the model and gain some control over the generation process. In the following text, we describe the encoder and decoder architectures, which are similar to the ones proposed in junction tree variational autoencoder(JTVAE). We also explain why we believe this is ideal for the task at hand.

3.1 DIVA: Model Architecture

Our proposed model is based on the Junction tree variational autoencoder (JTVAE) [29] architecture, which extends the standard variational autoencoder to generate molecular graphs. The model operates in two phases. First, a vocabulary of components is created from the training set of the dataset used. Molecular graphs are constructed by piecing together subgraphs from the vocabulary. The advantage of such a decoding processing is, the generated molecular graph always forms through chemically valid intermediaries. The vocabulary creation process is deterministic. We use the vocabulary created by the tree decomposition method explained in [29]. yThe vocabulary is used to form a junction tree of a molecular graph. We use the algorithm presented by [29] to decompose molecules into trees. More than one molecule can have the same junction tree. The junction tree is composed of overlapping components of atoms and clusters of atoms(rings). The molecular graph and its junction tree are fed into the graph encoder and tree encoder, respectively.

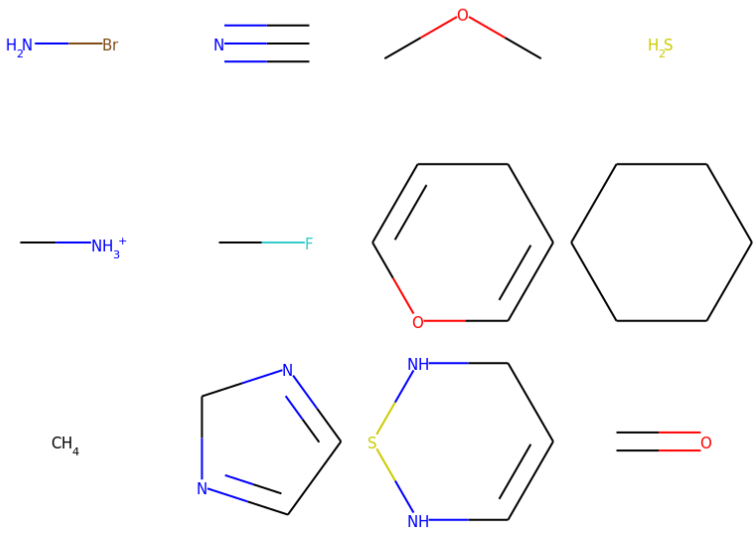

Figure 3.1: Some examples from the vocabulary created out of zinc dataset. 12 out of 780 components pictured.

The graph encoder encodes the molecular graph into latent representation z_G using message passing neural networks [17]. The message vector from node u to v, $m_{uv}^{(t)}$, is computed as,

$$m_{uv}^{(t)} = \tau(W_1^g x_u + W_2^g e_{uv} + W_3^g \sum_{w \in N(u) \setminus v} m_{wu}^{(t-1)})$$

Where, x_u is the node features of node u, e_{uv} is the edge features of the edge (u, v), and τ is the activation function. Here, the node features are the properties of the atom, and edge features the properties of the chemical bond between two atoms. The messages are aggregated after T iterations of message passing to obtain the node embedding of a node. Node embedding of node v, h_v, is computed as,

$$h_v = \tau(U_1^g x_v + \sum_{w \in N(v)} U_2^g m_{wv}^T)$$

The readout function to obtain the graph embedding is simple mean over all the node embeddings. The graph embedding is further passed through a feedforward network to obtain parameters for the Gaussian distribution from which z_G is sampled.

The tree encoder encodes the junction tree into latent representation z_T using gated graph neural network[36]. A node is arbitrarily picked from the tree to be it's root. Message propagation happens from leaf nodes to the root and the root to the leaf nodes alternatively. Message from node u to v, m_{uv}, is updated as,

$$m_{uv} = \text{GRU}(x_u, (m_{wu})_{w \in N(u) \setminus v})$$

The aggregation of messages, $(m_{wu})_{w \in N(u) \setminus v}$ is done as mentioned in [36]. Node embedding of each node is obtained by aggregating the messages coming inwards for each node. For a node, v, node embedding, h_v, is computed as,

$$h_v = \tau(W^o x_v + \sum_{w \in N(v)} U^o m_{wv})$$

The readout function just selects the node embedding of the root as it's graph embedding. The tree embedding h_{root} is passed through a feedforward to obtain parameters for the Gaussian distribution from which z_T is sampled.

Once (z_G, z_T) are obtained from the encoder, they are passed to the decoder. The decoding also happens in two phases. First, a junction tree is generated from z_T. Later, z_G is used to construct the molecular graph from the junction tree.

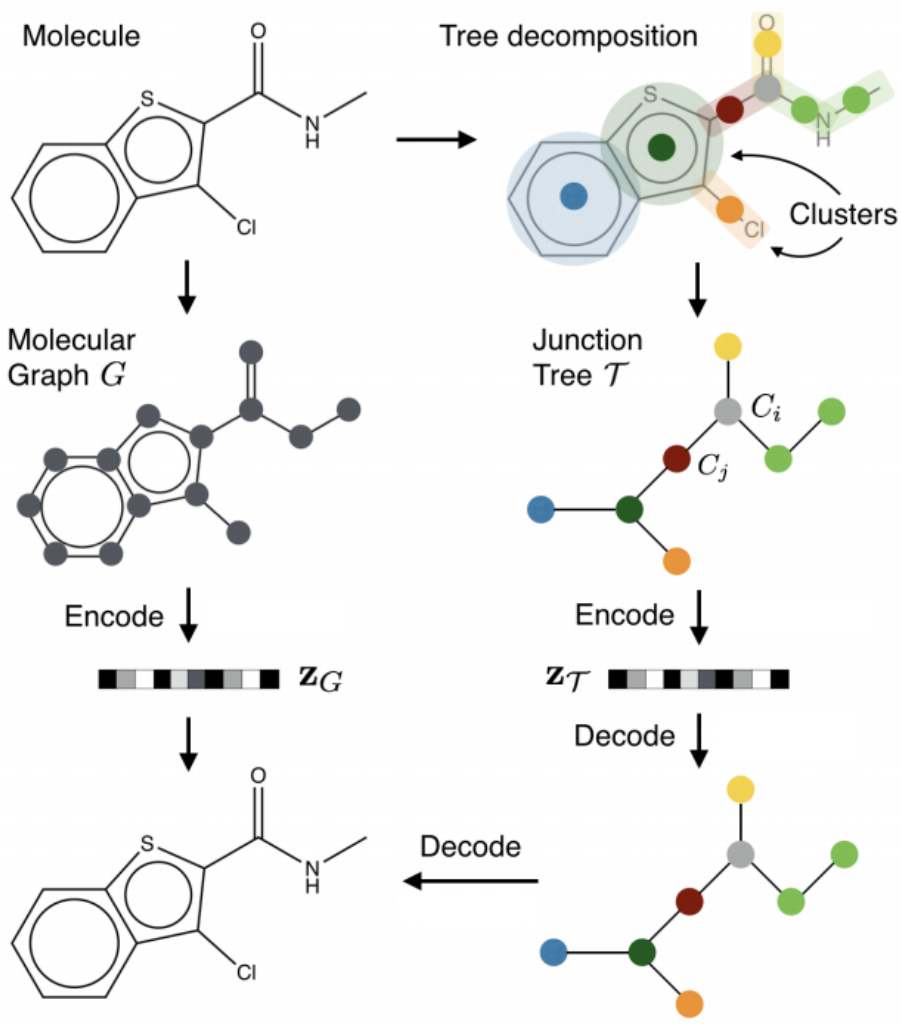

Figure 3.2: Junction tree variational autoencoder. The figure is taken from [29].

The tree decoder constructs the junction tree in a top down manner. When a node is generated, a topological prediction is made to decide if it has any children. If it does, the labels of the children are assigned from the generated vocabulary. Then the control shifts to the child node generated and a similar decision sequence takes place. Hence, the nodes are generated in a depth-first order. To make the topological prediction, the tree decoder uses the messages obtained from the nodes neighbors that have already been generated in this process. The probability of a child nodes existence for given node u is given by,

$$p = \sigma(U^d \cdot \tau(W_1^d x_u + W_2^d z_T + W_3^d \sum_{(v,u)\in E_t} m_{v,u}))$$

where E_t is the edges traversed in the depth-first traversal over the generated tree. The message $m_{v,u}$ for each v is computed as,

$$m_{v,u} = \mathrm{GRU}(x_v, (m_{w,v})_{(w,v)\in E_t, w\neq u})$$

where the message aggregation, $(m_{w,v})_{(w,v)\in E_t, w\neq u})$, is done as mentioned in [36]. If a child node v is generated from a parent node u with probability p, it's label is predicted by using z_t and $m_{u,v}$ as,

$$q_v = \mathrm{softmax}(U^l \tau(W_1^l z_T + W_2^l m_{u,v}))$$

where q_v is a distribution over the vocabulary of components that we generated from the training data. We re-normalize it over the components that are chemically compatible with the parent node. The training for label prediction is done by minimizing a cross entropy loss between the true topological and label values and the generated values. A fixed ordering is used to obtain the ground truth values of the topological and label values.

A junction tree can correspond to multiple molecular graphs. Once the tree is created by the tree decoder, we use z_G to decide which molecular graph corresponds to this tree. For this, a scoring function was designed by [29]. To generate the molecular graph, the components of the junction tree are merged in the same order as their creation. Each merge operation is given a score. The scoring function is a simple dot product between the graph embedding, h_G, of the subgraph created after the merge operation and z_G. The subgraph embedding is obtained in a way similar to the graph embedding in graph encoder.

In the junction tree variational autoencoder, we have seen that the decoder can assemble a valid molecule by learning the interactions between the vocabulary of components instead of an atom by atom construction of the molecule. We wish to capture these interactions and have control over the generation process. To achieve this goal, we employ discrete binary variables for the junction tree latent representation, z_T. If our latent representation learns the interaction between the chemical components in a molecule, a discrete representation makes them explicit and allows us to realize them in a post-hoc analysis.

Functional groups in molecules are responsible for the characteristic reactions of those molecules. Since we work with "drug-like" molecules from the ZINC dataset, we expect these molecules to be chemically active and contain at least one functional group. While the vocabulary created through the tree decomposition algorithm is relatively large, only a few components like functional groups and common rings like benzene, imidazole, etc., frequently occur in the molecules. Hence, we expect the latent representation to be predominantly modelling the interactions between the frequently occurring and important chemical sub components of the molecules.

We train our model by making each dimension of z_T a discrete binary variable. To facilitate for gradient based learning, we use the Gumbel Softmax random variables discussed in section 2.4 as continuous relaxations of discrete random variables. This relaxation allows us to use automatic differentiation to produce gradients of the objective. We use Bernoulli(0.5) as the prior over each dimension of z_T. z_G is only used in the scoring function to determine the molecule that corresponds to a certain junction tree. We can make z_G discrete and interpretable in the same way we have for z_T, but we find z_T to be more suitable to the task as it models interactions between valid chemical components as opposed to z_G which needs to model interactions between atoms. Hence, z_G remains to be a continuous latent representation with standard normal as its prior. This concludes the discussion on the model architecture.

3.2 Implementation Details

Our implementation of DIVA is based on the implementation of [29] in pytorch. The temperature variable for the Gumbel-softmax is annealed to provide for better convergence (for details, refer to [40]). Since the reconstruction is terrible at the beginning, KL annealing is used to ensure convergence to a better optimum. Both the temperature and the KL constant is annealed every 1000 batch steps. The KL constant is incremented by a constant value of 0.001 starting from 0.0 with a maximum value of 1.0. The temperature is decreased exponentially at a rate of 0.9.

Additionally, a warmup of 5000 batch steps is done where the temperature is fixed to 0.3 and the KL constant to 0.

For training, Adam optimizer is used with an initial learning rate of 0.001, annealed every 40000 batch steps at a rate of 0.9. The training is done for 20 epochs, and a batch size of 10 is used. The tree depth used is 20, and the graph depth used is 3. The dimensionality of latent representation in the tree and graph encoding is 56.

Chapter 4

Experiments & Results

4.1 Dataset

We use the standard ZINC drug dataset[49] for all our experiments. ZINC is a publicly available dataset of commercially available compounds. This dataset was developed to provide researchers with access to a library of compounds for virtual screening. It contains 250k drug molecules which are split into the train, test, and valid sets. The dataset details about the biological activity, and properties such as molecular weight, octanol-water partitions coefficient, number of rotatable bonds, and the commercial availability data.

The dataset contains 'drug' or 'lead' molecules. A lead compound in drug discovery is a possible sub-optimal structure which could lead to a drug molecule upon tinkering with its structure to fit the target better. These compounds have biological activity and suggest therapeutical use. The lead compound's structure is the starting point for chemical modifications to make a drug.

4.2 Evaluation Methods

4.2.1 Molecule validity and generative ability

Since we build molecules by assembling from a vocabulary of chemically valid components, we always expect to generate a chemically valid molecule. We generate $20,000$ molecules by decoding latent vectors, $(z_T, z_G)s$, from their prior distributions. DIVA always decodes a valid molecule from the decoder. Hence, we have model with latent representation z_T discrete without sacrificing the ability to always generate valid molecules. Different z_Ts sometimes decode the same molecule, we remove all the duplicates after generation. We consider $20,000$ unique generated molecules for further analysis.

To prove that our model isn't a simple memorization, we check the number of molecules that are generated which do not exist in the training set. Out of the $20,000$ molecules generated, 99.93% of the molecules are new molecules.

4.2.2 Quality of the generated samples

The water-octanol partition coefficient (logP) is a metric that characterizes the drug-likeness of a molecule. The property we measure is penalized logP score. logP is penalized by the synthetic accessibility (SA) score and number of long cycles. Synthetic accessibility score estimates the difficulty of synthesis of drug-like molecules. We report the best three molecules we obtain in the $20,000$ molecules we generated.

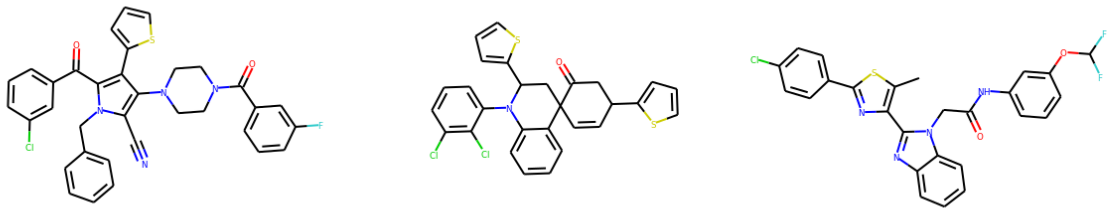

Figure 4.1: Best three molecules in the $20,000$ generated with property scores of 4.27, 4.40, 4.54 respectively

All the other VAE based baseline methods perform optimization in the latent space using Bayesian optimization to finds molecules with high property scores. Since our latent representation z_T is discrete, we can not perform bayesian optimization in the way the baseline models do. Despite this limitation, we can see from the table 4.1 that our model produces molecules with property scores as high as the ones obtained after bayesian optimization on the baseline models. DIVA finds over 20 molecules(in 20,000) with scores over 3.50 (the second best molecule proposed by SD-VAE).

Method	1st	2nd	3rd
CVAE	1.98	1.42	1.19
GVAE	2.94	2.89	2.80
SD-VAE	4.04	3.50	2.96
JTVAE	5.30	4.93	4.49
DIVA*	4.54	4.40	4.27

Table 4.1: Best molecule property scores found by each method. The values for other models are taken from [29]. * - without bayesian optimization.

4.2.3 Interpretability and Editing molecules

As mentioned in section 3, we use discrete binary variables for the latent representation z_T with Bernoulli(0.5) as a prior on each dimension. z_G is a continuous latent variable with a standard normal distribution as its prior. All the results we report are from a model trained with a 56 dimensional z_T. Since z_T is discrete and is expected to capture the interactions between the chemical components in the vocabulary created, we can devise posthoc analysis methods to identify the interactions being modeled by a specific dimension or a set of dimensions of z_T. In this section, we present how the posthoc analysis can be done and demonstrate the results.

The prior forces for 50% of all the dimensions to be active(one). We find empirically that 40% of the 56 latent dimensions are active on average. If we obtain disentangled embeddings as we expect, the interactions of major components of the vocabulary would be modeled by a latent dimension or a set of latent dimensions. If we can identify the set of dimensions controlling specific interactions of a chemical subcomponent, we can mask these dimensions in the latent space of molecules in which the chemical subcomponent is unwanted and produce new molecules without the chemical subcomponent by decoding from the masked latent representation.
We try to identify such interactions only on single latent dimensions and not their sets. We do this because masking too many dimensions will produce molecules that are very dissimilar from the original molecules. We do the identification analysis in two ways:

- We encode a set of molecules that haven't been seen by the model during training, using the encoder of the trained DIVA model. For each dimension, we identify a set of molecules that have this particular dimension active and has the least number of total latent dimensions active. We visually inspect for any specific functional group interactions in these molecules. If we observe a similarity, we test our hypothesis. We test our hypothesis by encoding the molecules in the valid set of the dataset and selecting the molecules with the particular functional group. We mask the particular dimension while decoding to see if the functional group disappears in the outputted molecules. The rationale behind visualizing the molecules with the least number of total latent dimensions active is, we except an active dimension's influence to be more when the total number of active dimensions is low.

- While we identify a few dimensions and their role in generation by the method described above, visually analyzing isn't a principled way for analysis. We propose a more disciplined way for our posthoc analysis. Choose the chemical component which you would like to edit from the molecules. Encode a set of molecules that have this chemical component and weren't seen by the model during training, using the encoder of the trained DIVA model.

Look at the global statistics of the encoded z_T to notice which dimensions are mostly active for the encoded molecules. Use the valid set of the dataset to figure out the dimension that represents the interactions of this particular chemical component. Note that we might not find such a dimension in which case we will have to look for all the subsets of the active dimensions. We do not put any constraints while training, so it's not necessary for every chemical component to have a corresponding dimension(s) modeling it's interactions.

It's important to note that a dimension does not exclusively represent a certain chemical component's interaction, and a chemical component's interaction can be modelled by more than one set of dimensions. Hence, even after we mask a certain dimension that corresponds to a chemical component, the chemical component can still appear in the decoded molecules because of the other dimensions that are still active. We provide partial and not total control over editing.

A molecule obtained by removing the unwanted chemical component from a drug-like molecule may not give rise to a drug-like molecule. Naively editing a molecule by removing the unwanted chemical component does not serve our purpose as molecules function is greatly determined by its structure. For example, (S)-(+)-naproxen is used to treat arthritis pain, but (R)-(−)-naproxen causes liver poisoning with no analgesic effect. The S enantiomer of thalidomide causes birth defects, while the R enantiomer is effective against morning sickness.
We want the molecules generated to be drug-like. The process we propose for editing circumvents this problem because the latent representation in DIVA learns a distribution over the drug-like molecules and the molecules decoded from the masked representation are expected to be drug-like and edited.

Below we present some examples of editing. We provide the editing performance on valid set from which identification was performed, and the test set. We also provide the average similarity scores between the original and edited molecules from the test set. We use Tanimoto similarity with Morgan fingerprint [44] as the similarity metric.

Chemical component	**Similarity**
Iodine	0.35
Nitriles	0.22
Ammonia	0.22
Sulfones	0.27
Flourine	0.26
CF3	0.26
Triazoles	0.26

Table 4.2: Tanimoto similarity with Morgan fingerprint between the original and edited molecules from the test set.

Iodine(I) (in purple): Figures 4.2,4.3,4.4,4.5

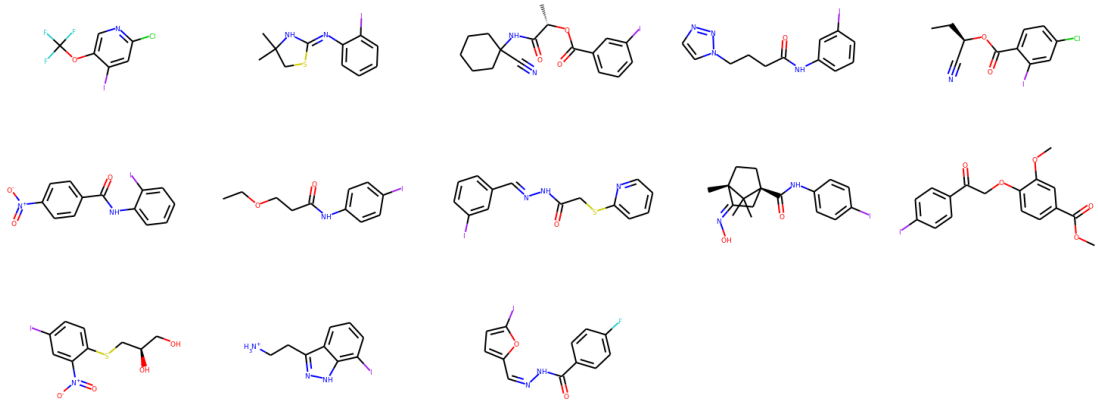

Figure 4.2: The molecules in valid set that contain the iodine functional group and the identified dimension is active.

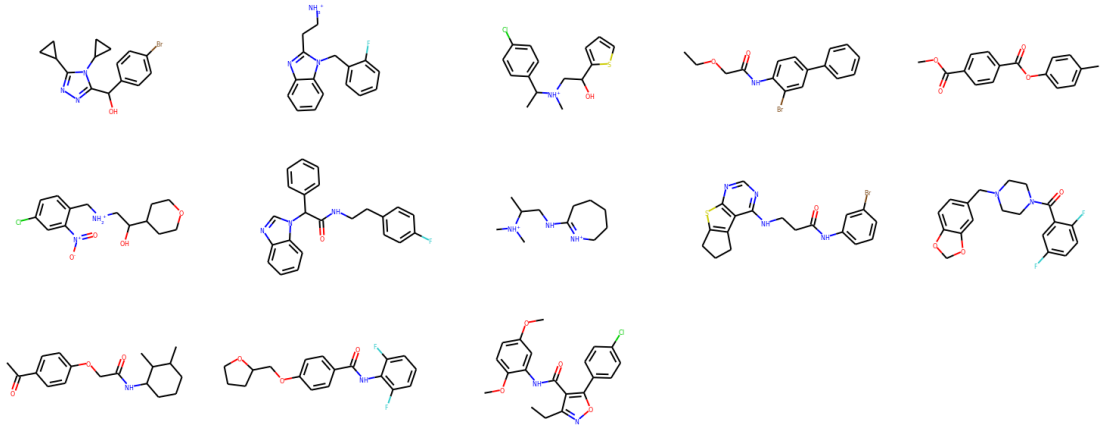

Figure 4.3: The corresponding edited molecules from 4.2 after masking the identified dimension. We can see that the amount of iodine functional group has reduced.

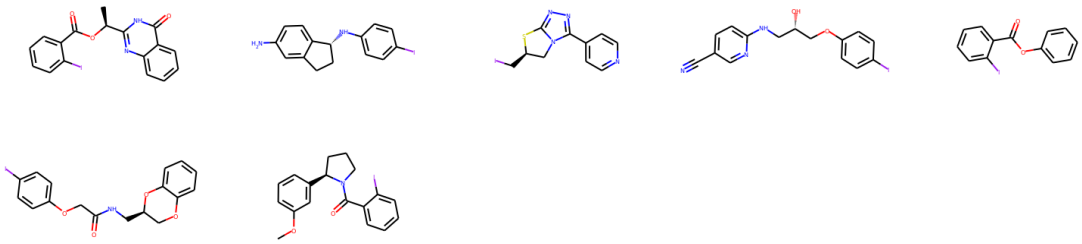

Figure 4.4: The molecules in test set that contain the iodine functional group.

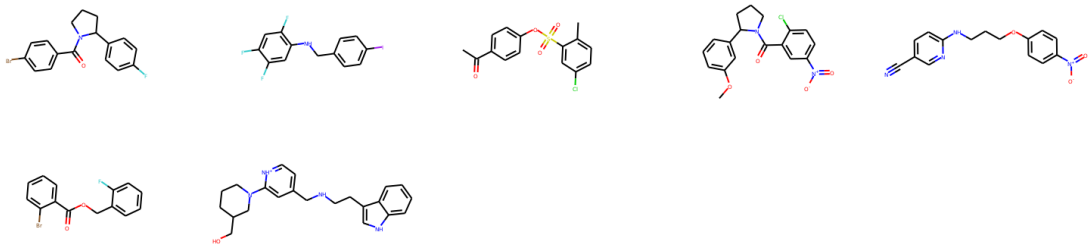

Figure 4.5: The corresponding edited molecules from 4.4 after masking

Nitriles(CN) (C triple bond N in blue): Figures 4.6, 4.7,4.8,4.9

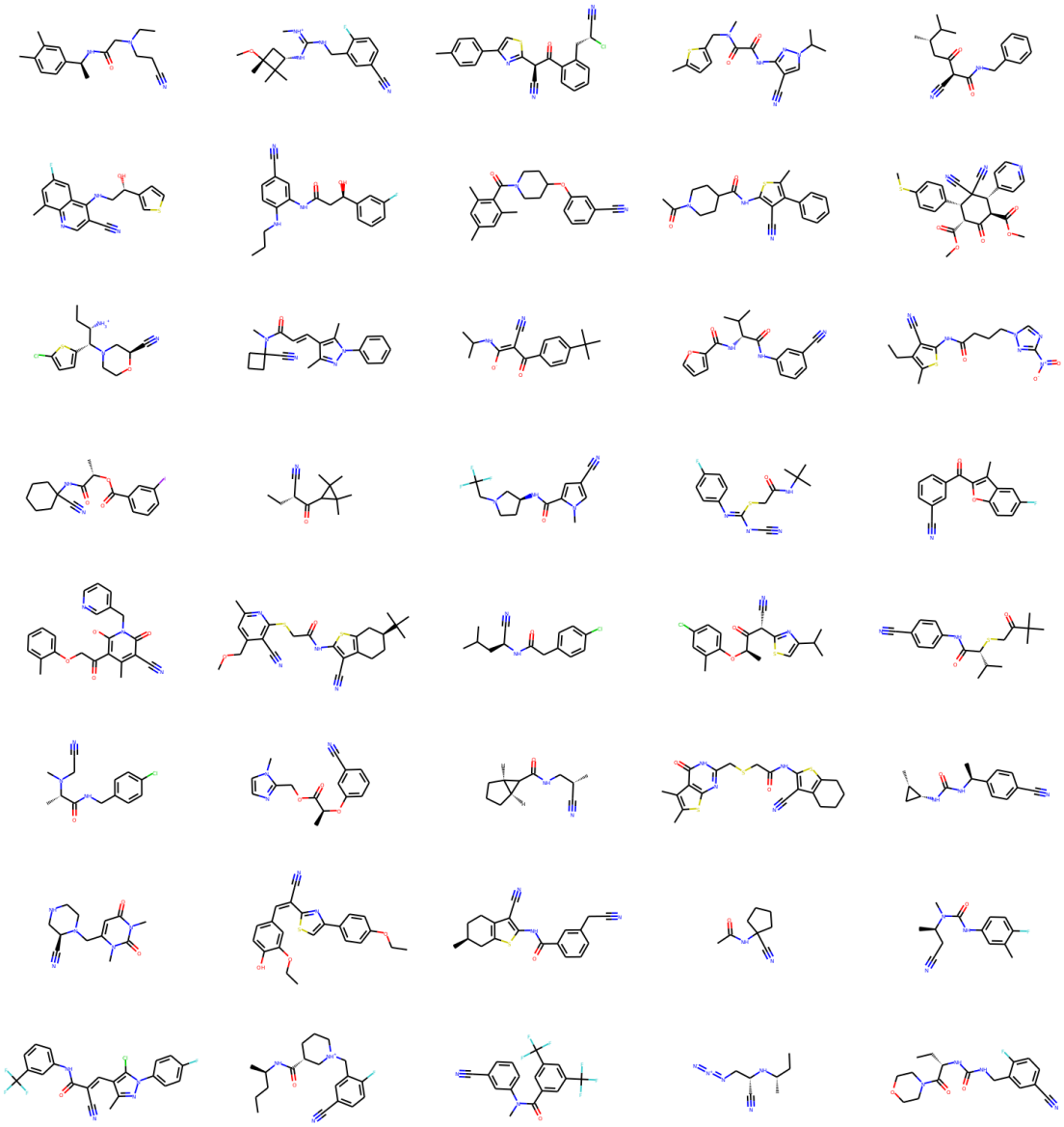

Figure 4.6: 35 random molecules in valid set that contain the nitrile functional group and the identified dimension is active.

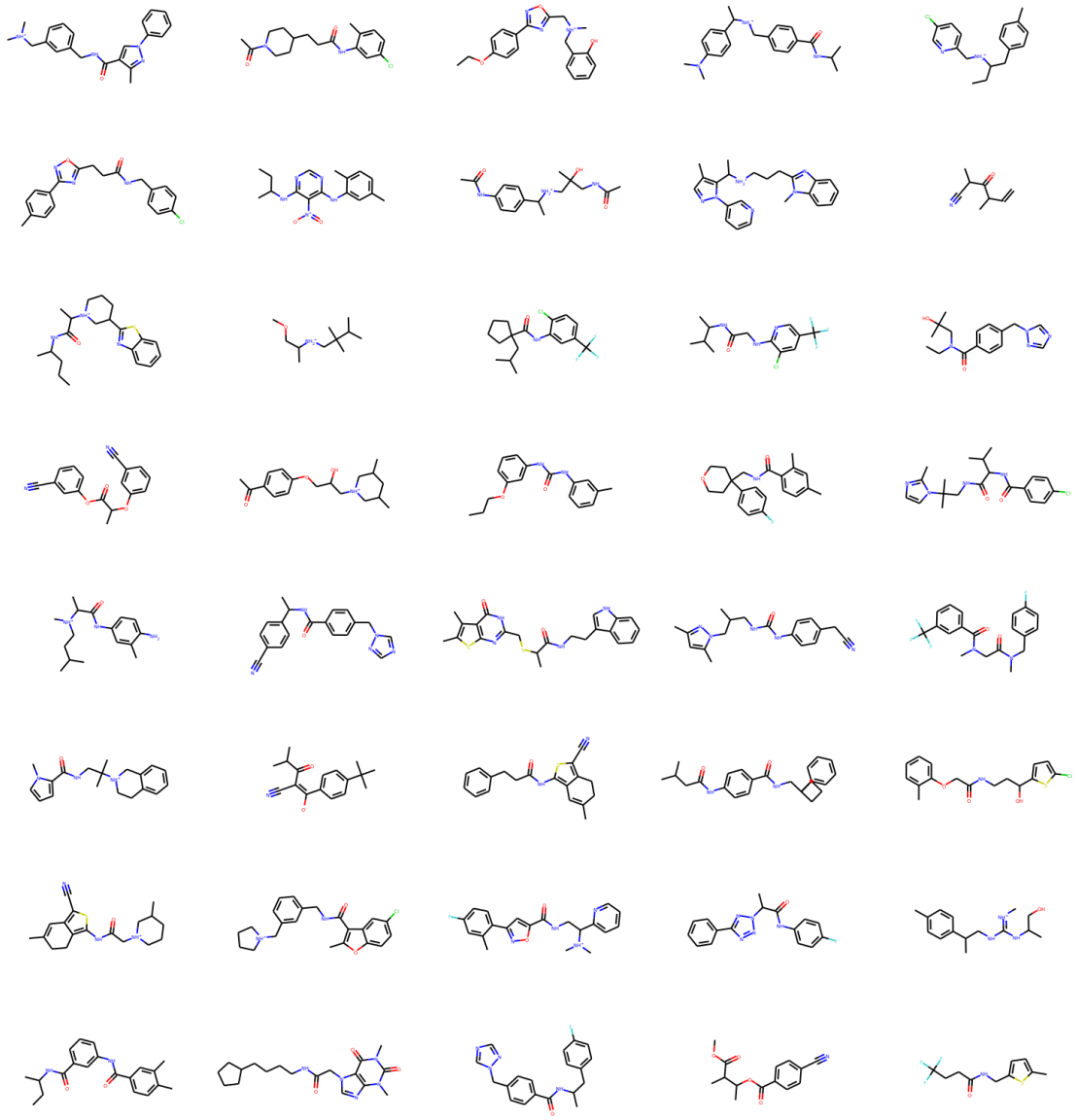

Figure 4.7: 35 corresponding edited molecules from 4.6 after masking the identified dimension. We can see that the amount of nitrile functional group has reduced.

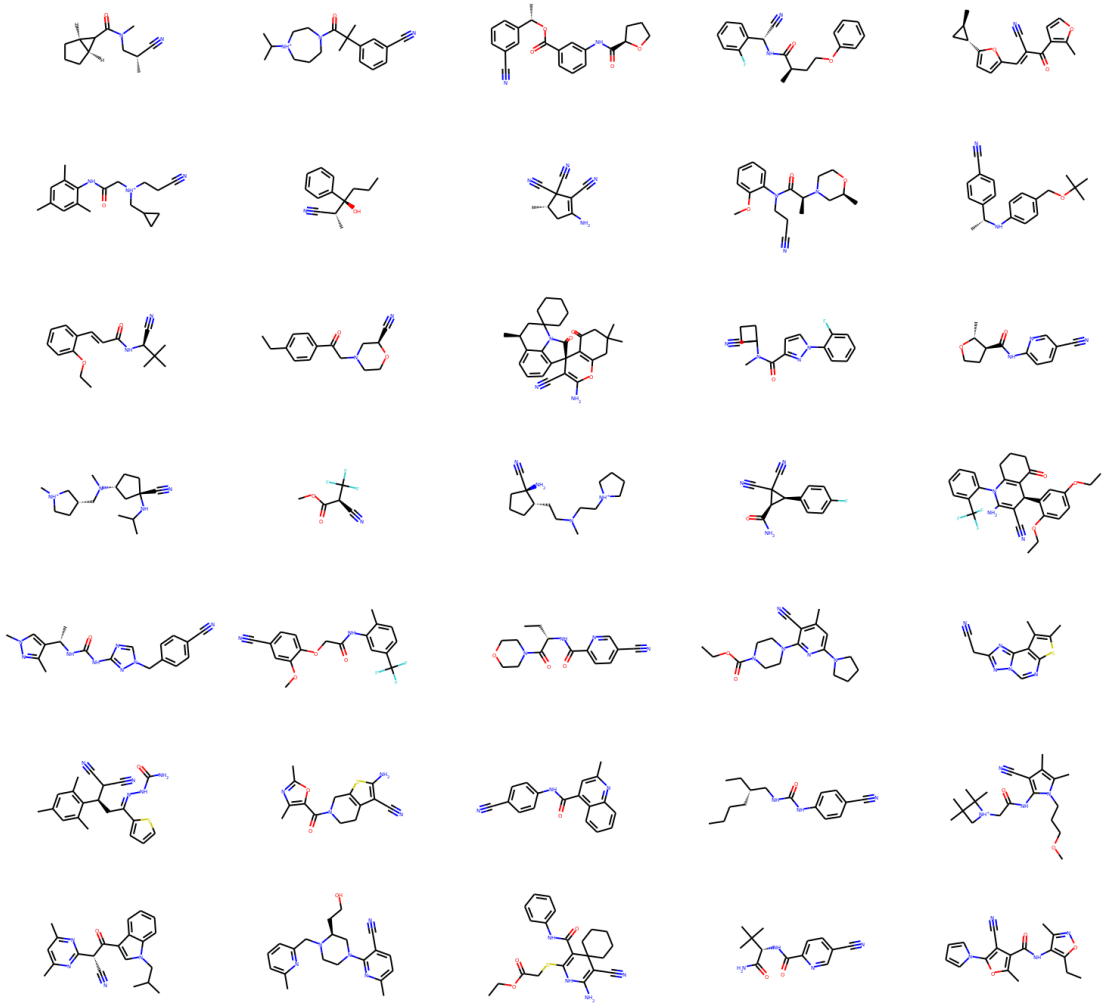

Figure 4.8: 35 random molecules in test set that contain the nitrile functional group and the identified dimension is active.

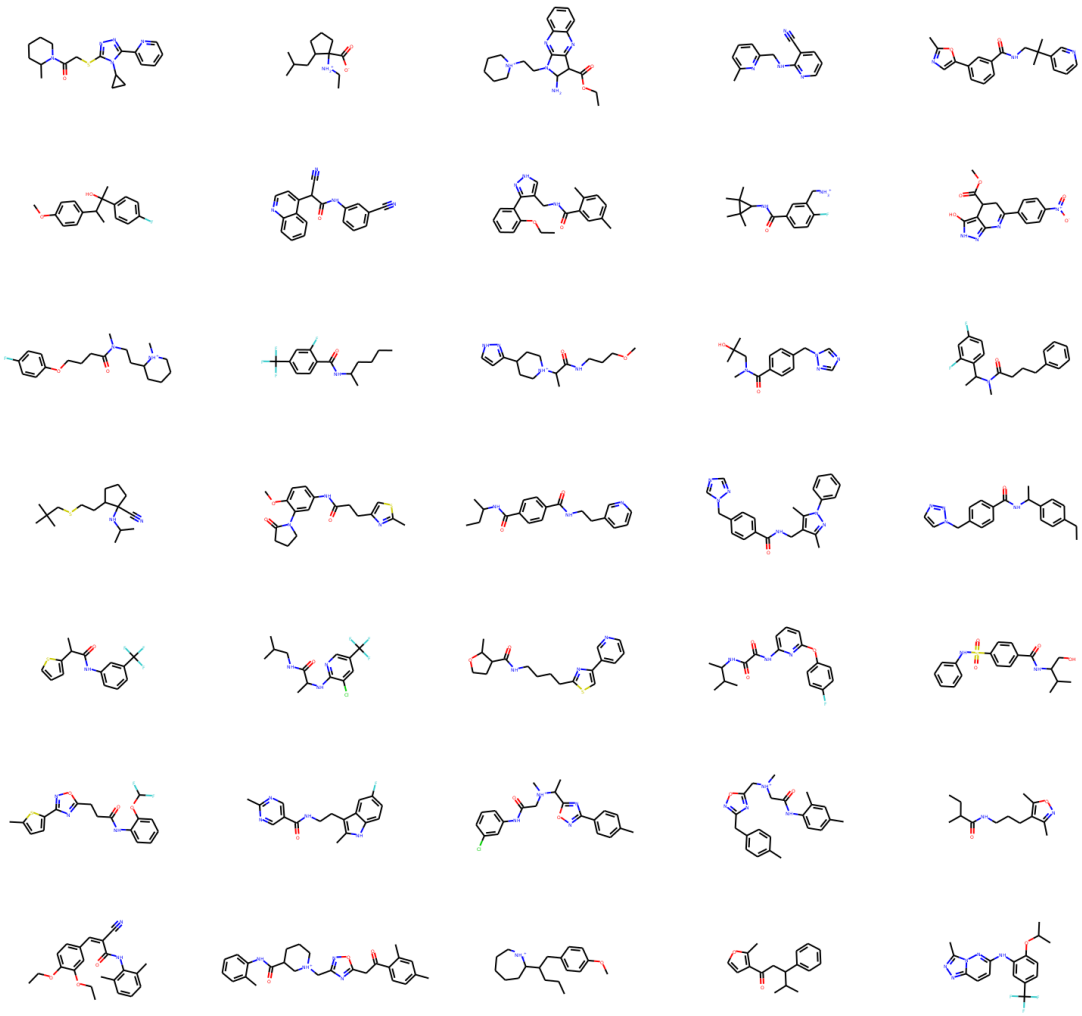

Figure 4.9: The corresponding edited molecules from 4.8 after masking

Ammonia cation(NH3+) (in blue): The identified dimension turns NH3+ into NRH_2+ or NR_2H+ or NRH_2 or NR_2H or NR_3 N. Figures 4.10, 4.11, 4.12, 4.13

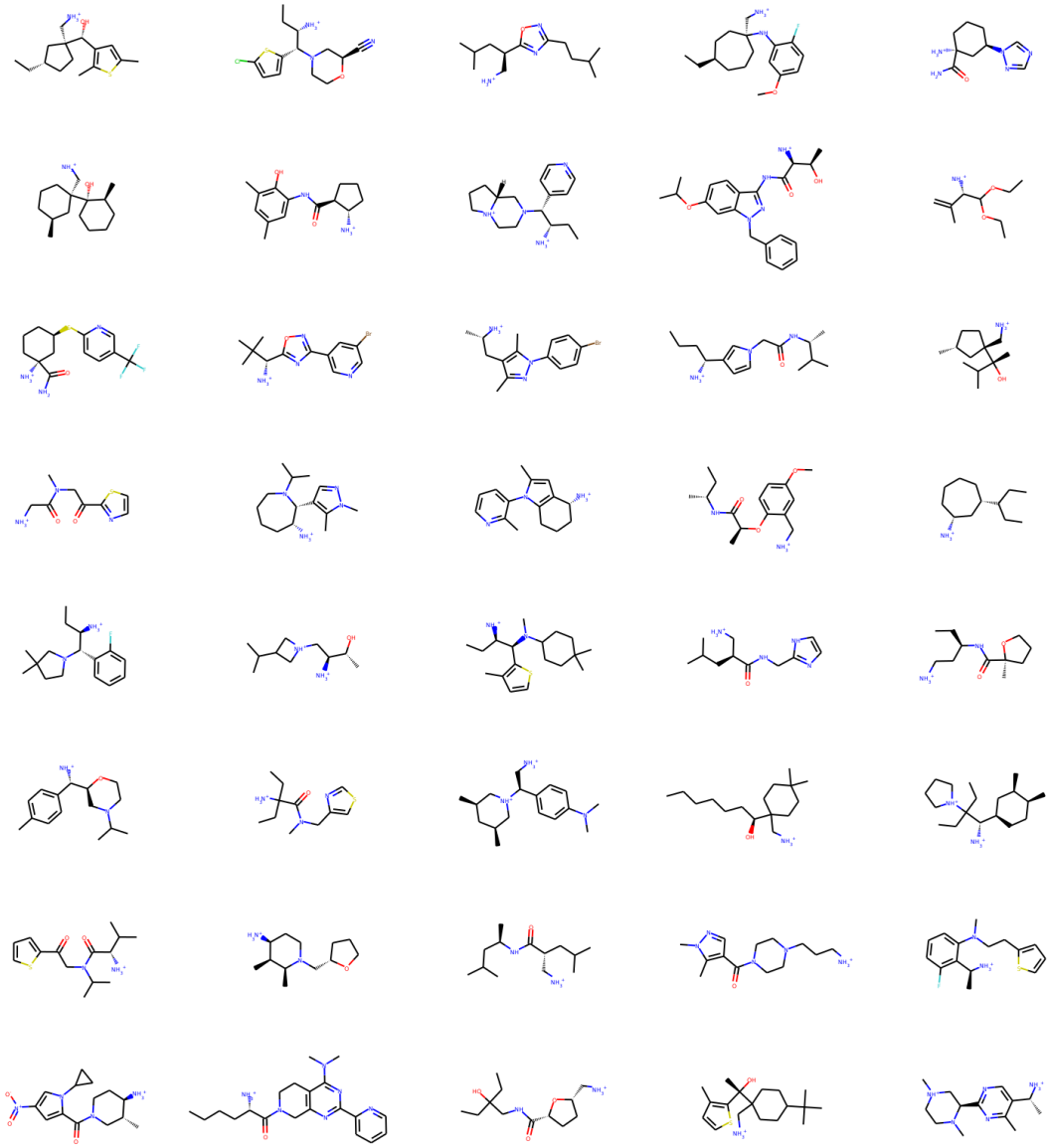

Figure 4.10: 40 random molecules in valid set that contain the Ammonia cation functional group and the identified dimension is active.

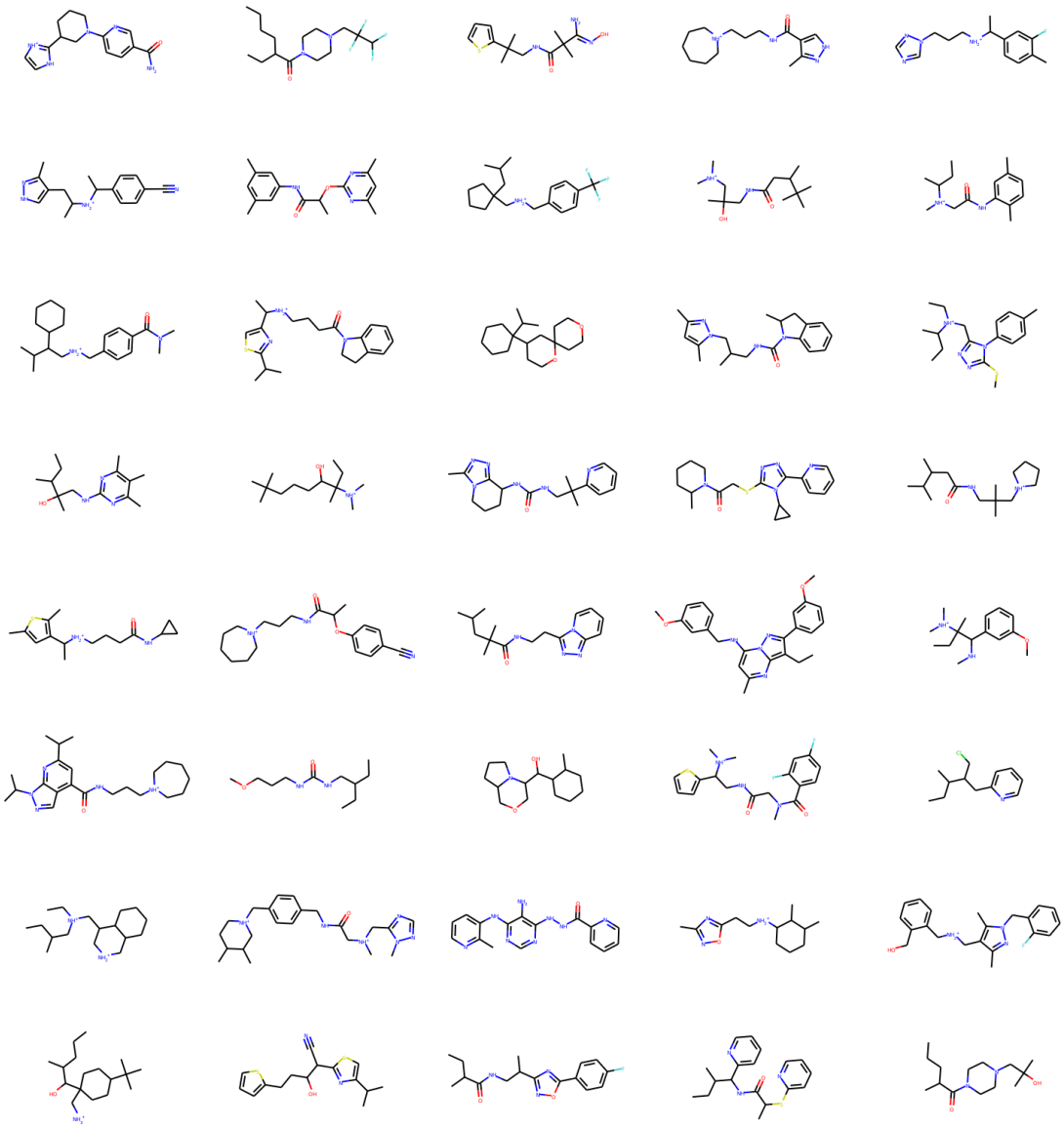

Figure 4.11: The corresponding edited molecules from 4.10 after masking the identified dimension. We can see that the amount of Ammonia cation functional group has reduced.

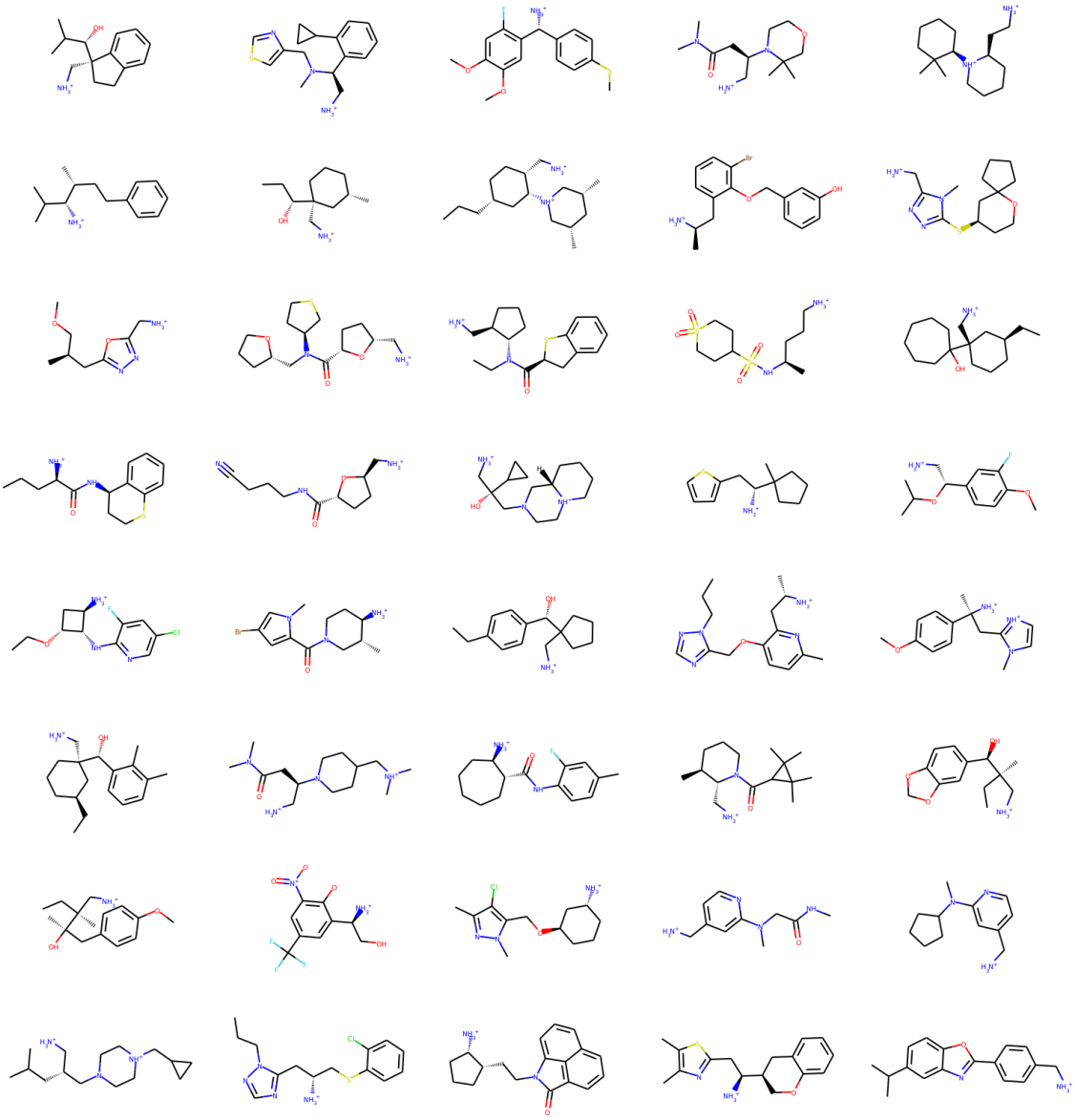

Figure 4.12: 40 random molecules in test set that contain the Ammonia cation functional group and the identified dimension is active.

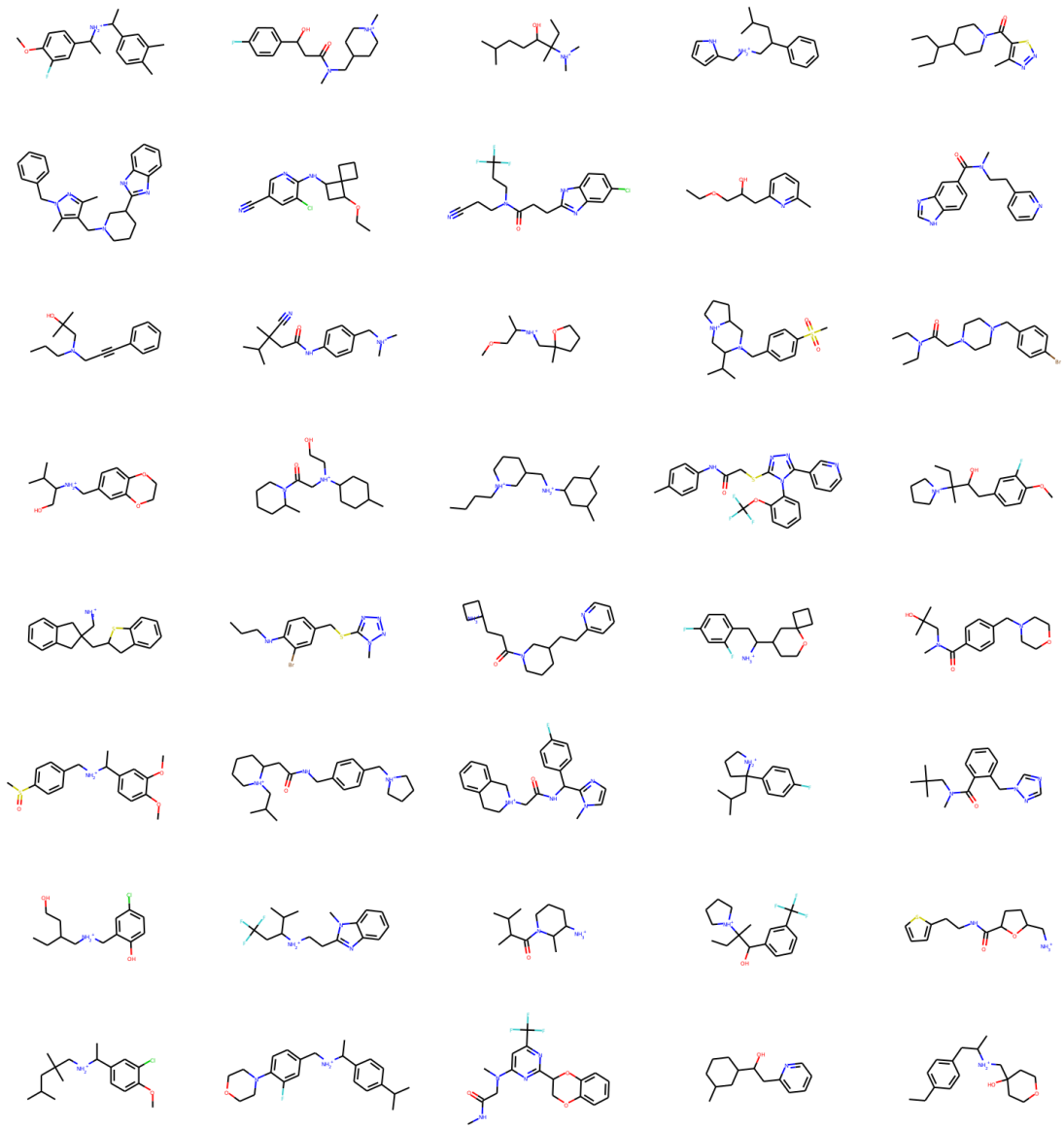

Figure 4.13: The corresponding edited molecules from 4.12 after masking

Sulfone (SO_2R_2) (S in yellow, oxygens in red): Figures 4.14, 4.15, 4.16, 4.17

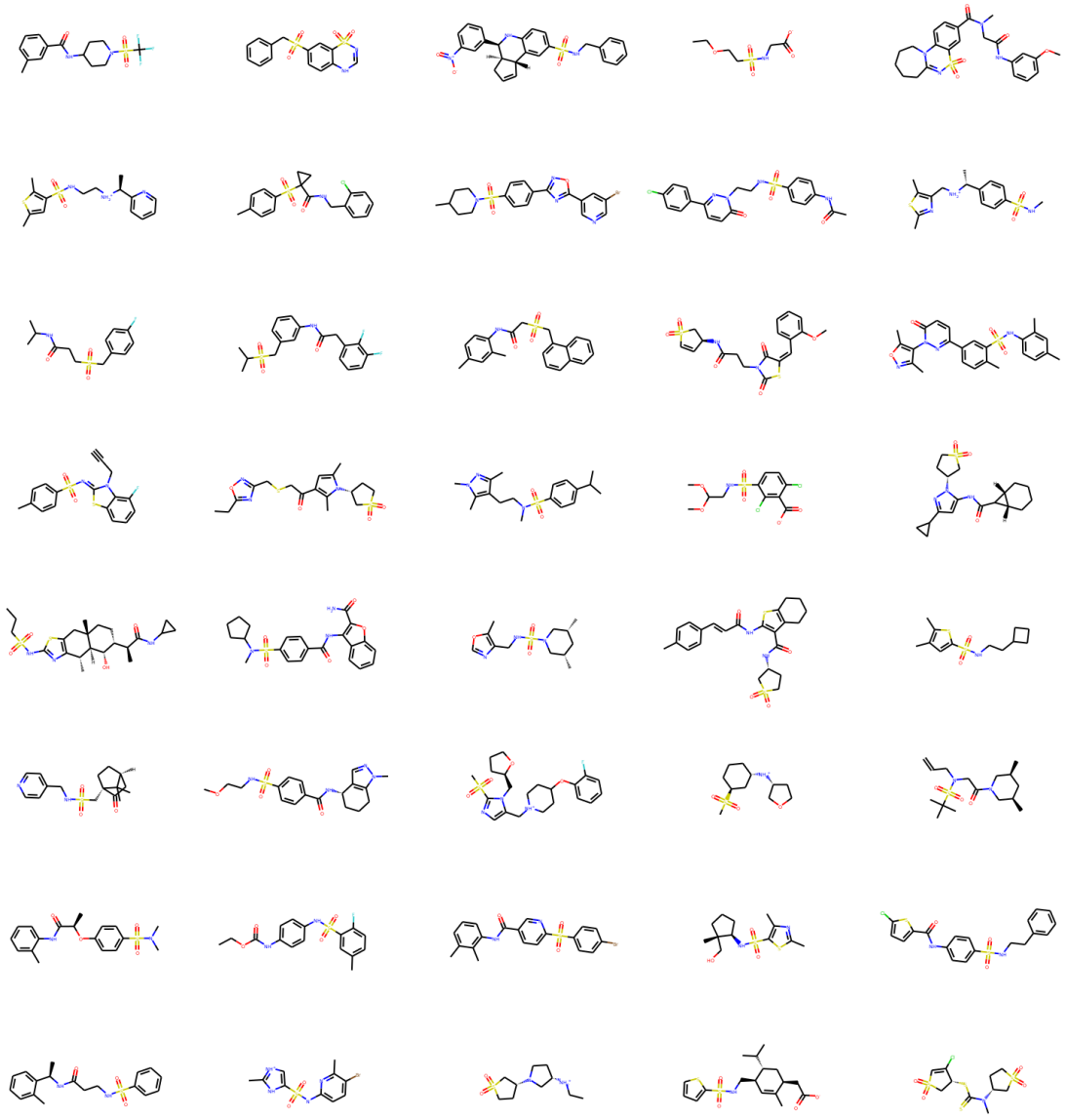

Figure 4.14: 40 random molecules in valid set that contain the sulfone functional group and the identified dimension is active.

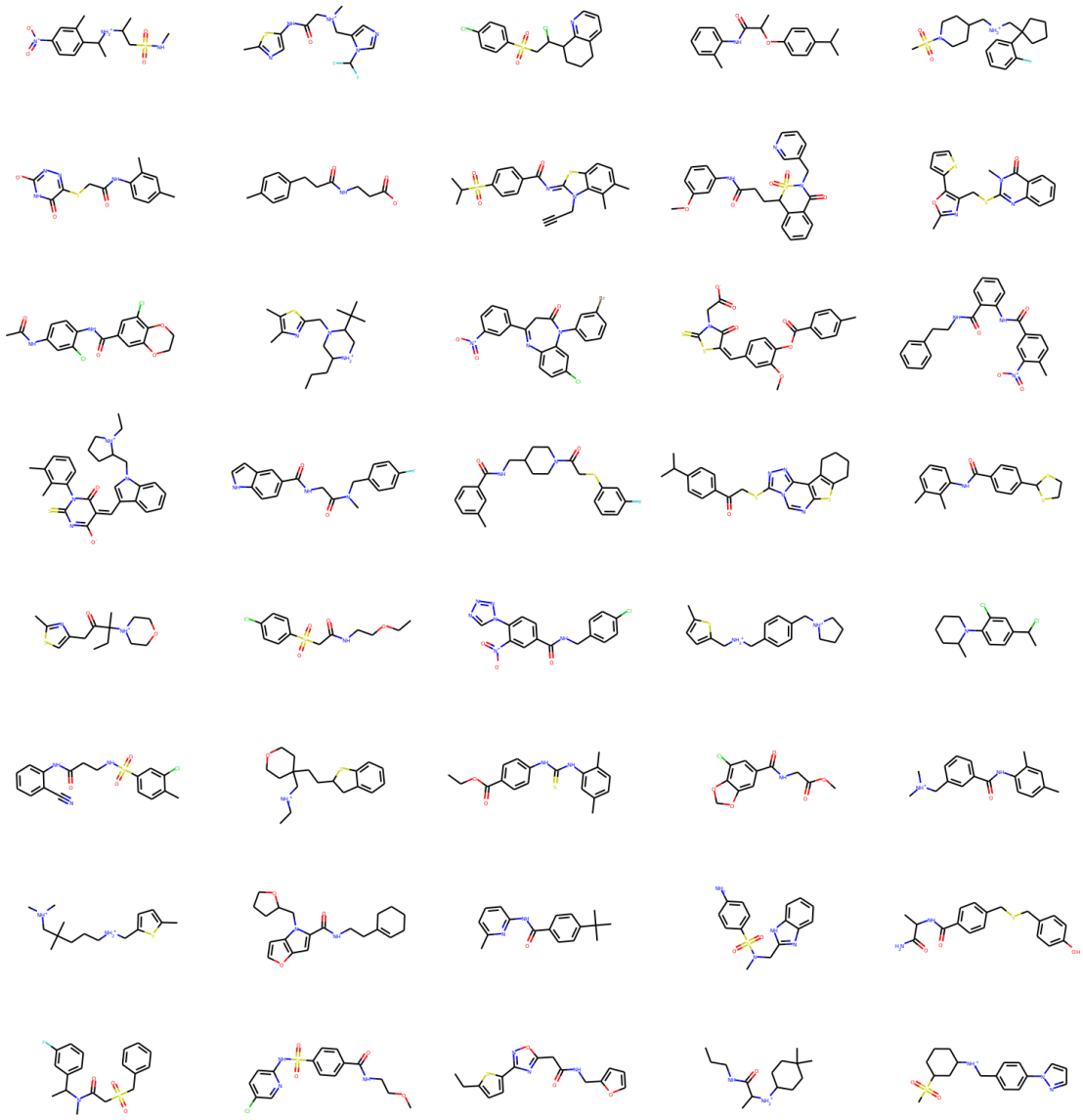

Figure 4.15: The corresponding edited molecules from 4.14 after masking the identified dimension. We can see that the amount of sulfone functional group has reduced.

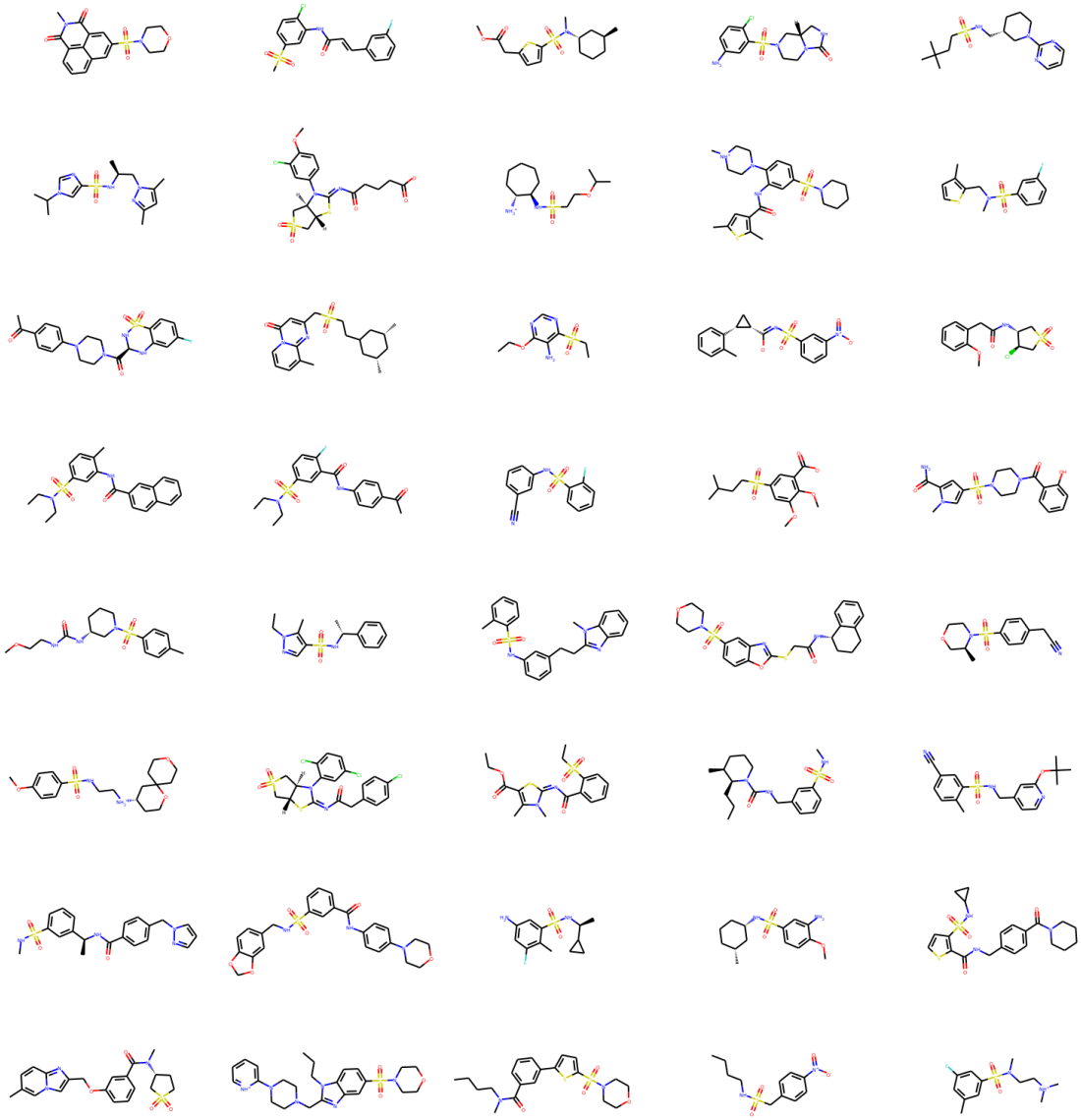

Figure 4.16: 40 random molecules in test set that contain the sulfone functional group and the identified dimension is active.

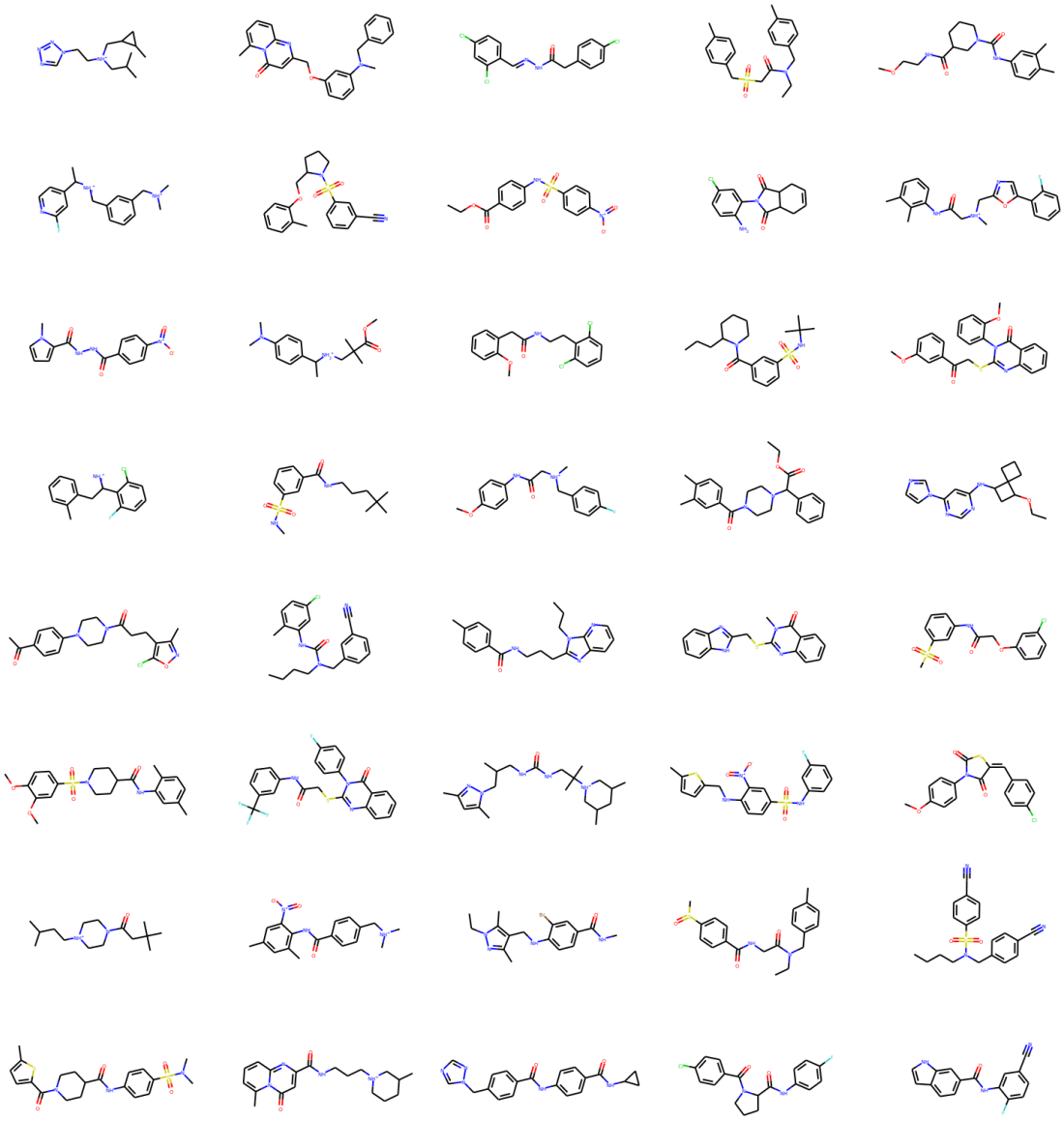

Figure 4.17: The corresponding edited molecules from 4.16 after masking

Fluorine(F) (in light blue): Figures 4.18, 4.19, 4.20, 4.21

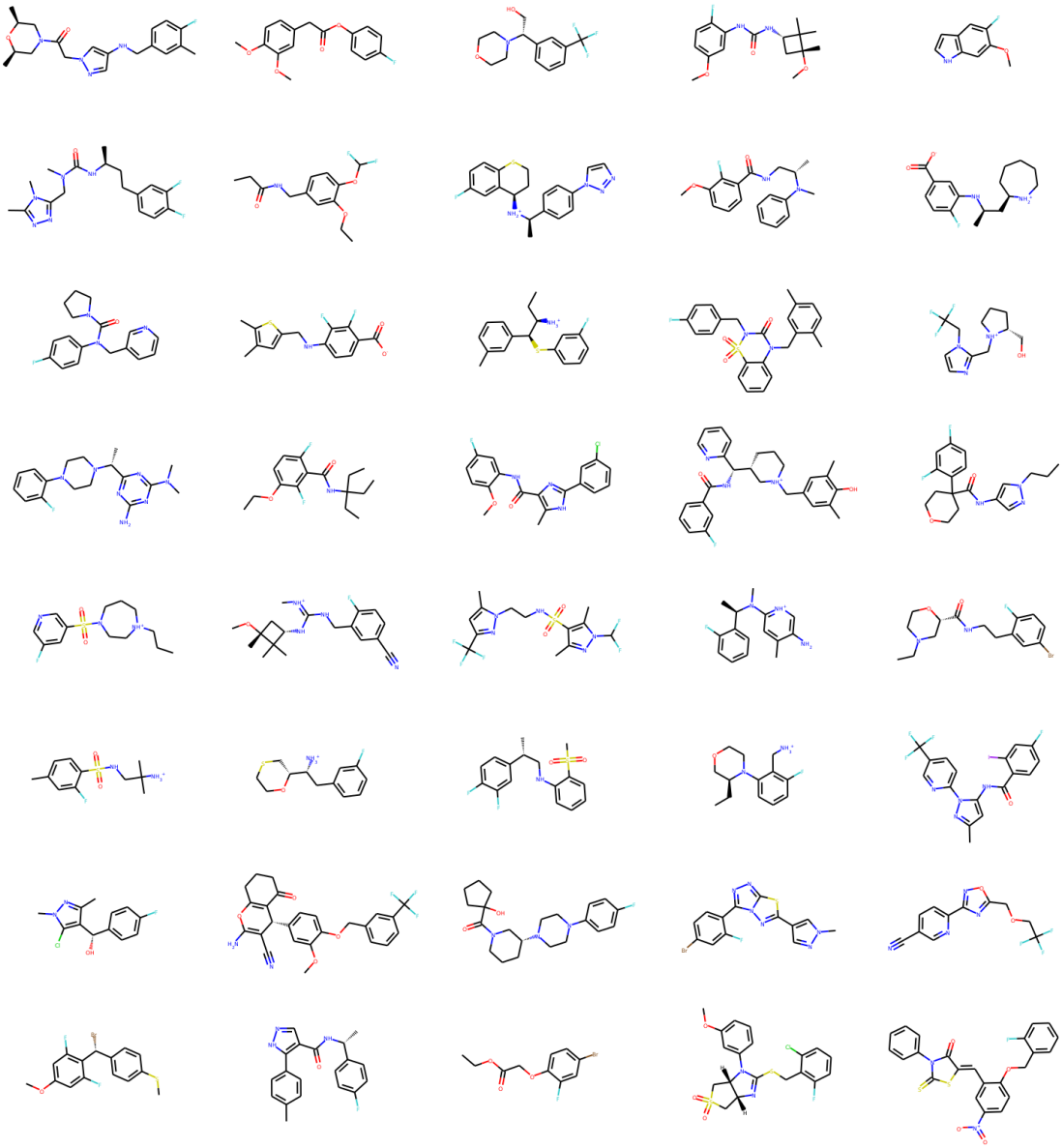

Figure 4.18: 40 random molecules in valid set that contain the fluorine functional group and the identified dimension is active.

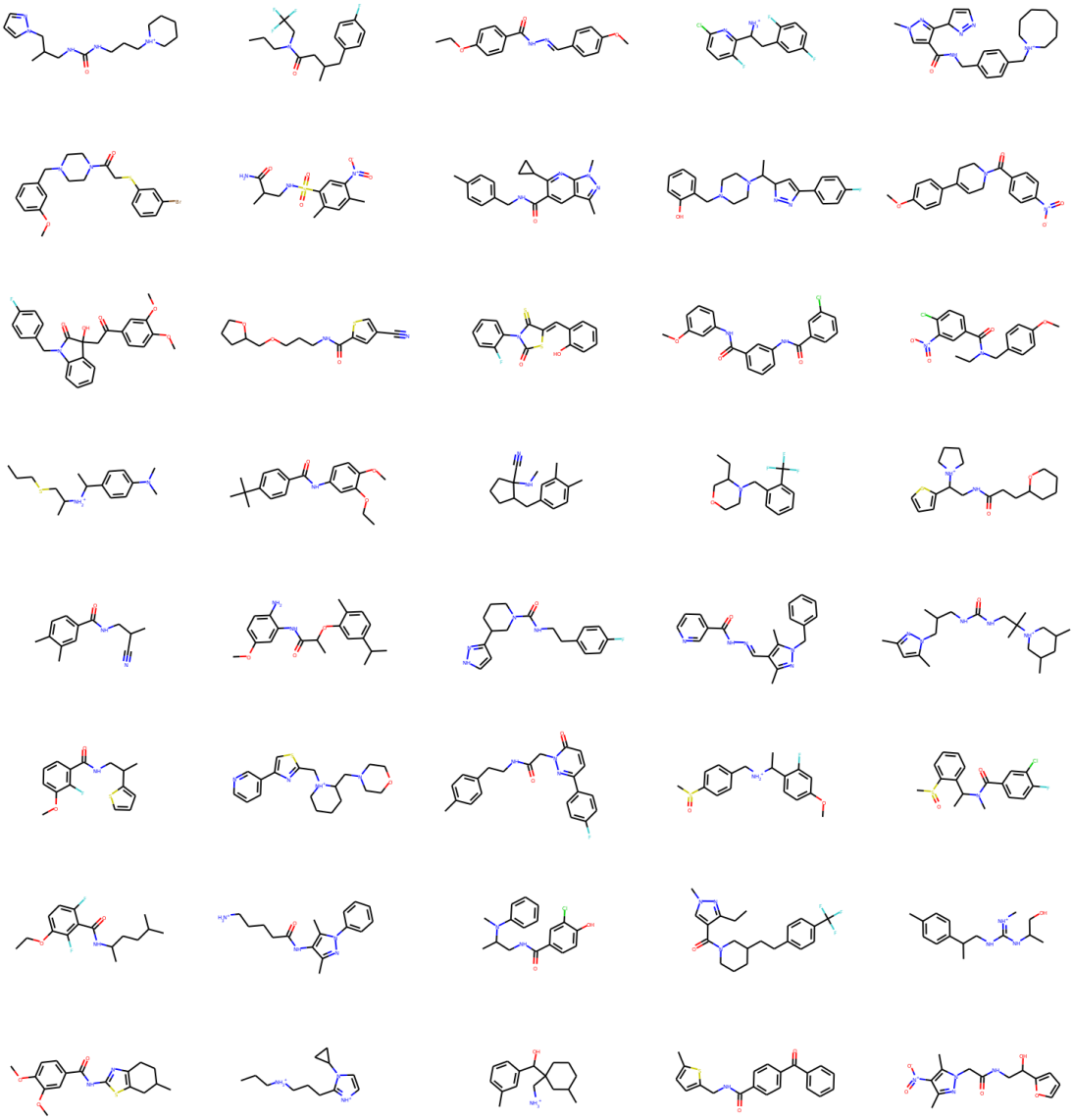

Figure 4.19: The corresponding edited molecules from 4.18 after masking the identified dimension. We can see that the amount of fluorine functional group has reduced.

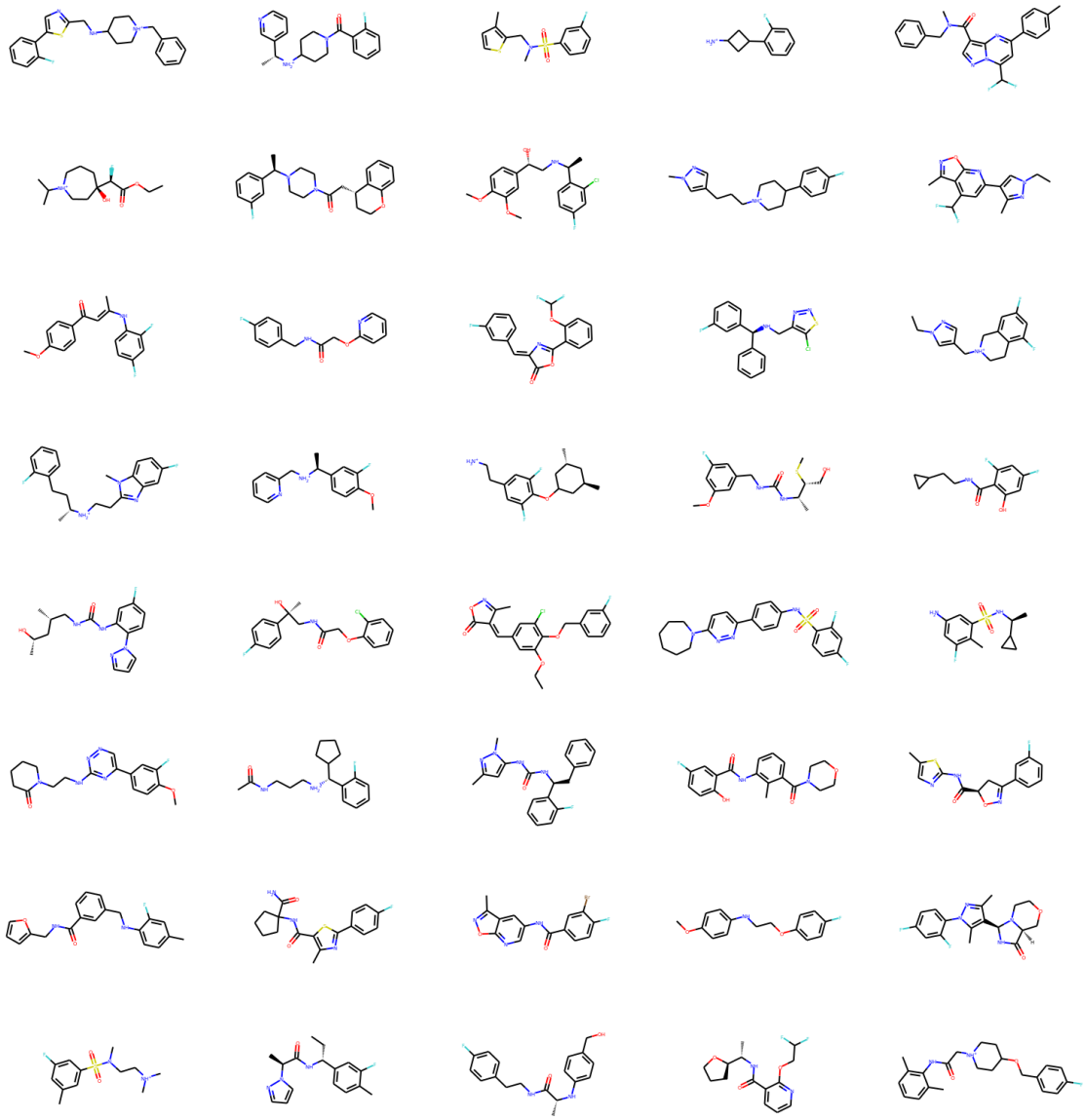

Figure 4.20: 40 random molecules in test set that contain the fluorine functional group and the identified dimension is active.

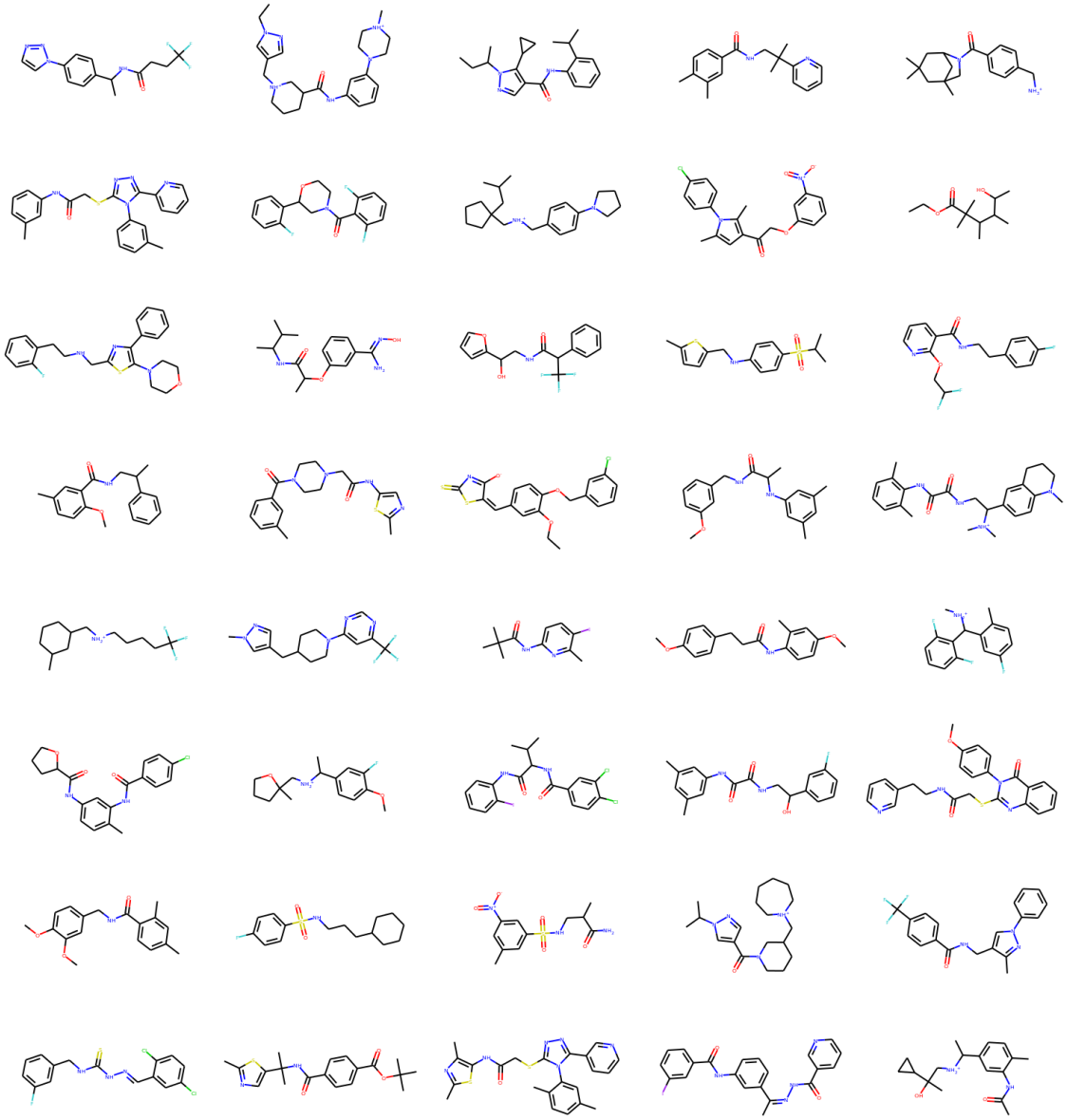

Figure 4.21: The corresponding edited molecules from 4.20 after masking

Trifluoromethyl(CF3): Figures 4.22, 4.23, 4.24, 4.25

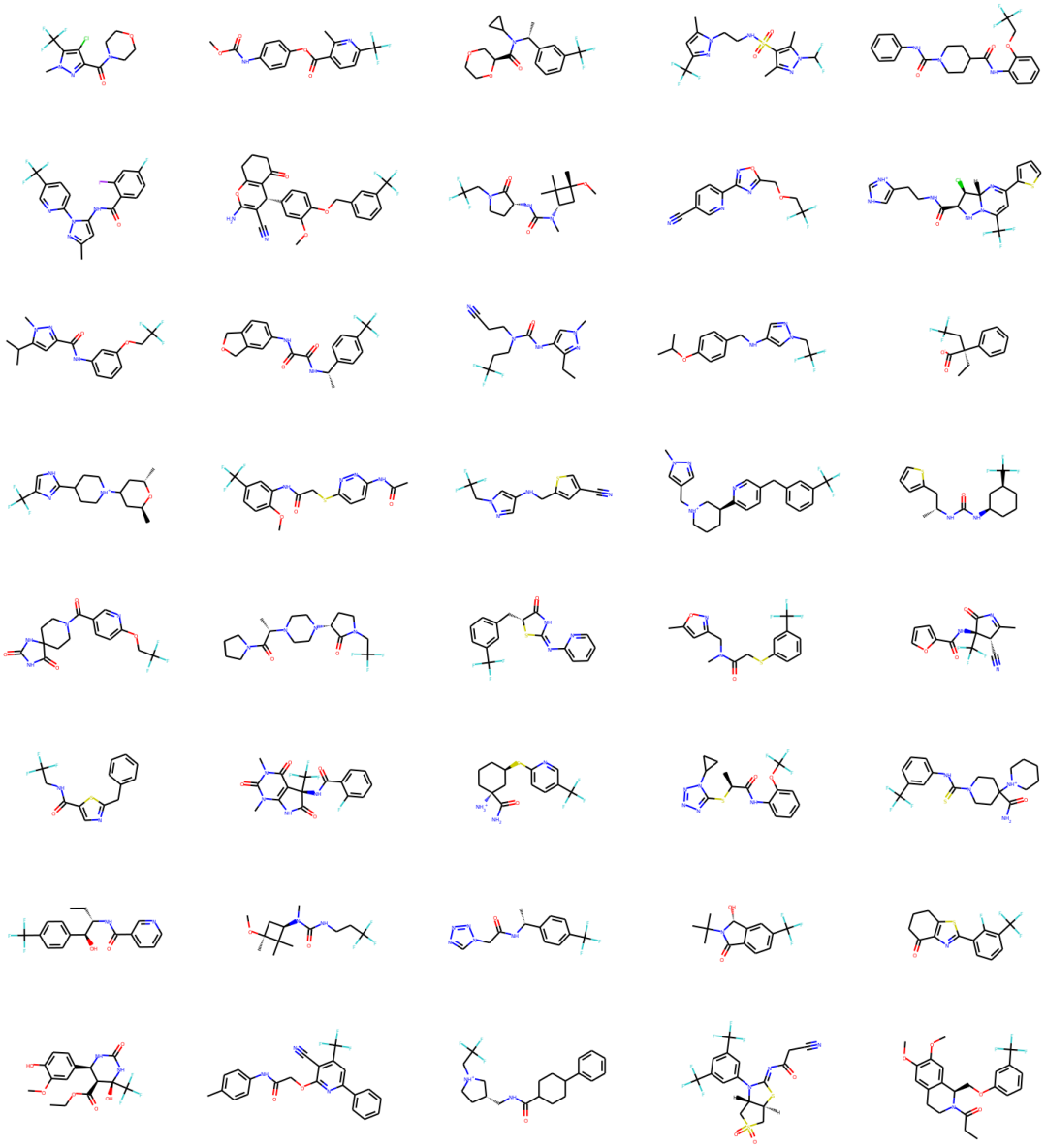

Figure 4.22: 40 random molecules in valid set that contain the trifluoromethyl functional group and the identified dimension is active.

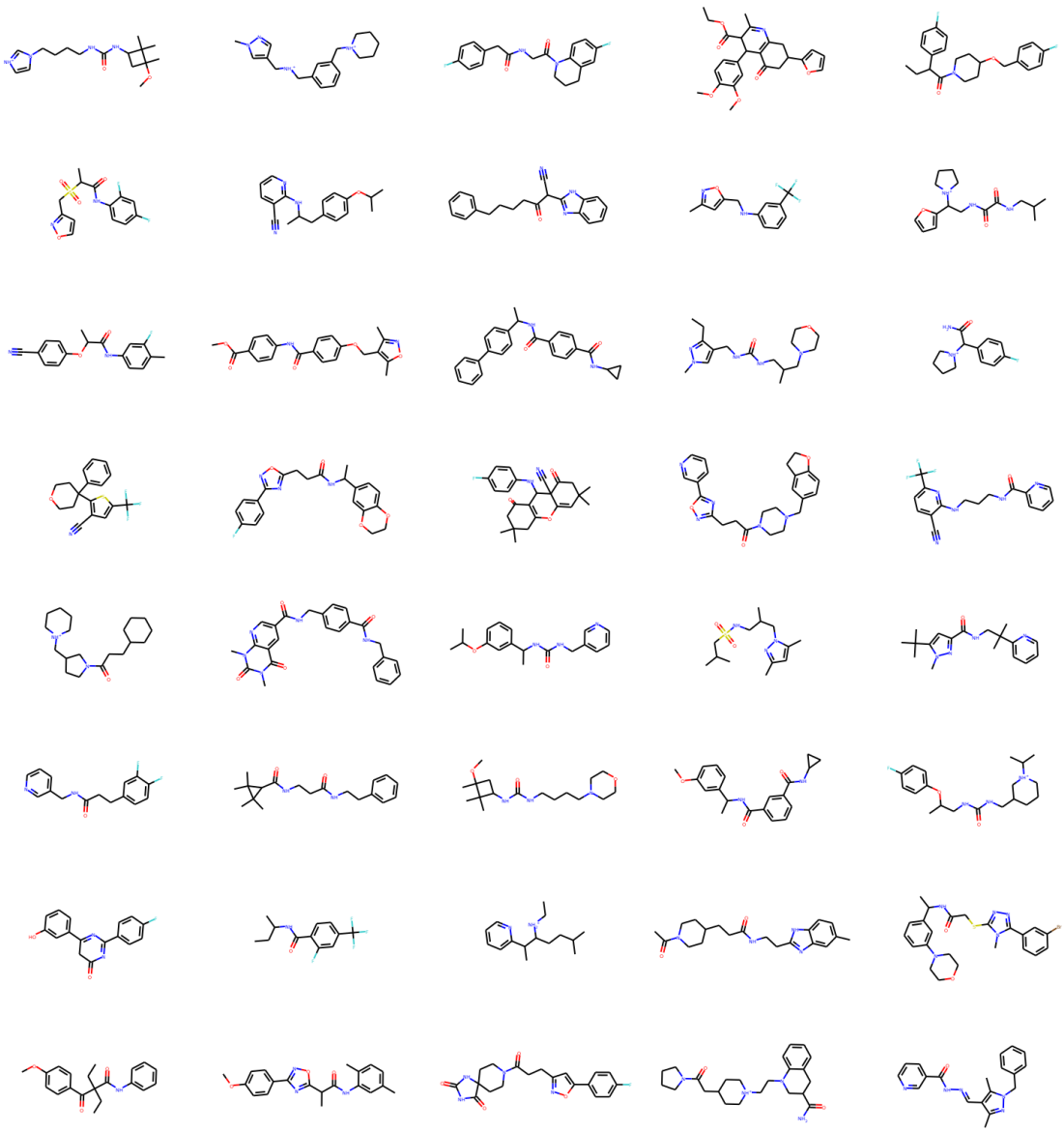

Figure 4.23: The corresponding edited molecules from 4.22 after masking the identified dimension. We can see that the amount of trifluoromethyl functional group has reduced.

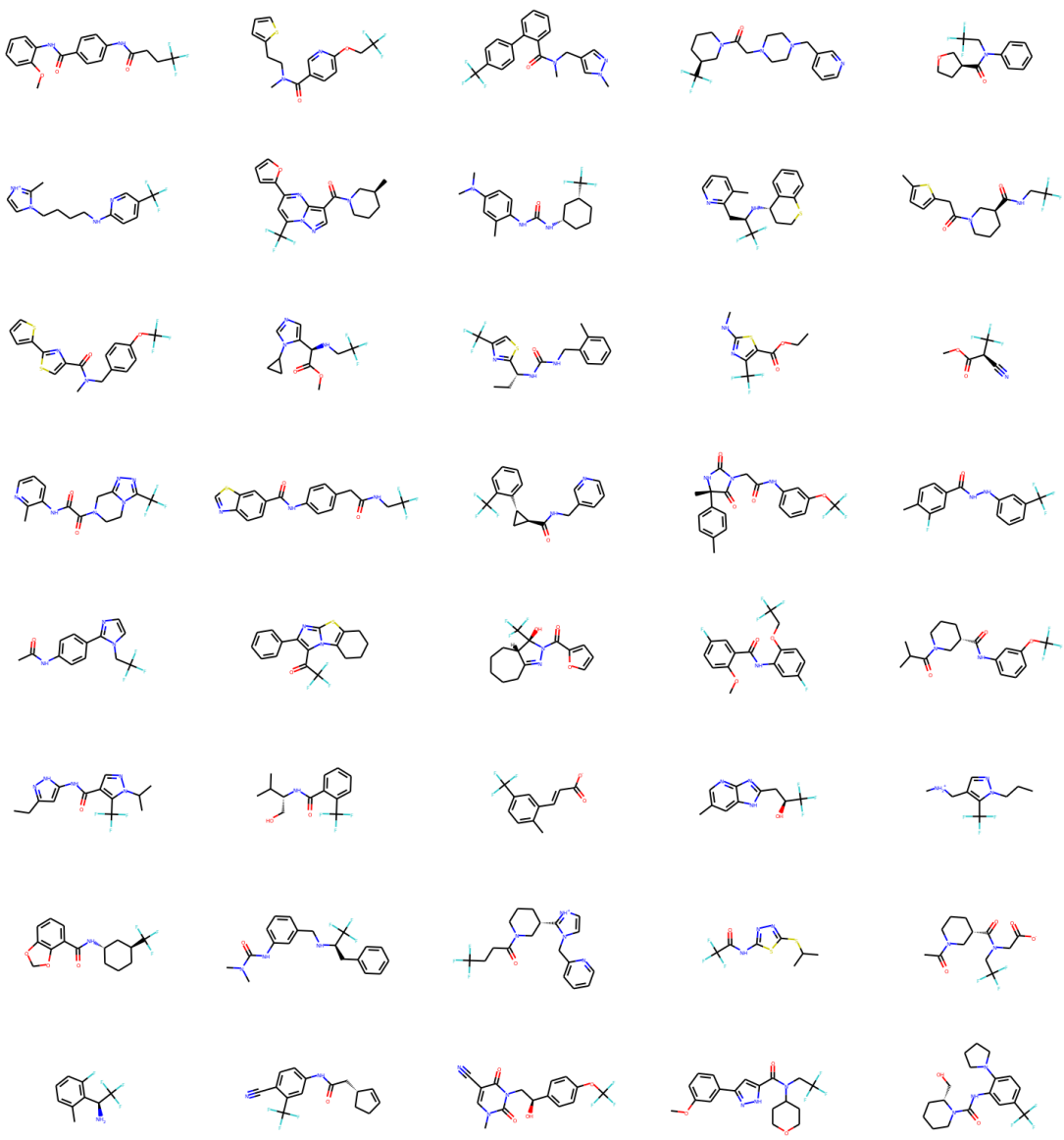

Figure 4.24: 40 random molecules in test set that contain the trifluoromethyl functional group and the identified dimension is active.

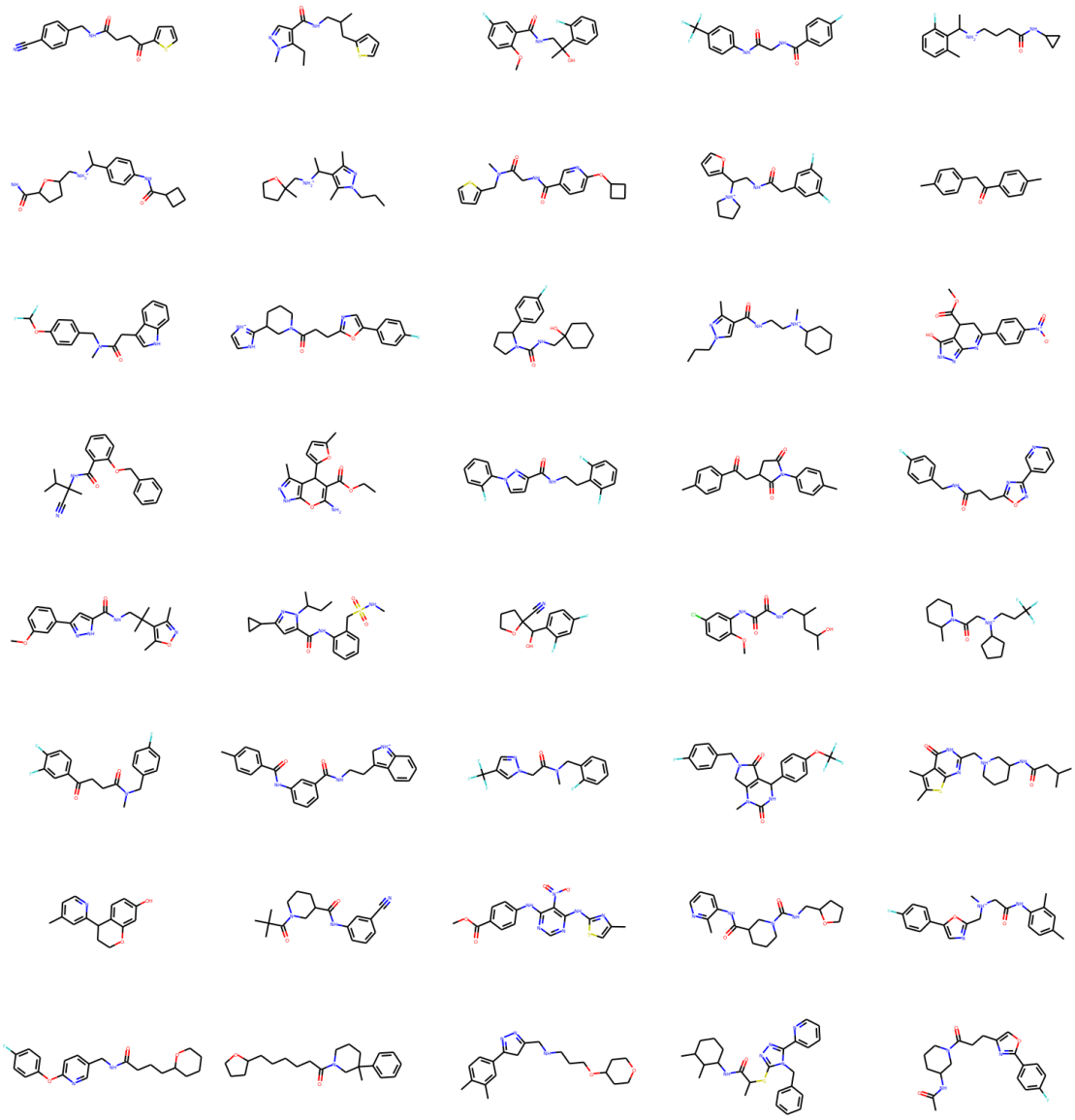

Figure 4.25: The corresponding edited molecules from 4.24 after masking

Triazole: Figures 4.26, 4.27, 4.28, 4.29

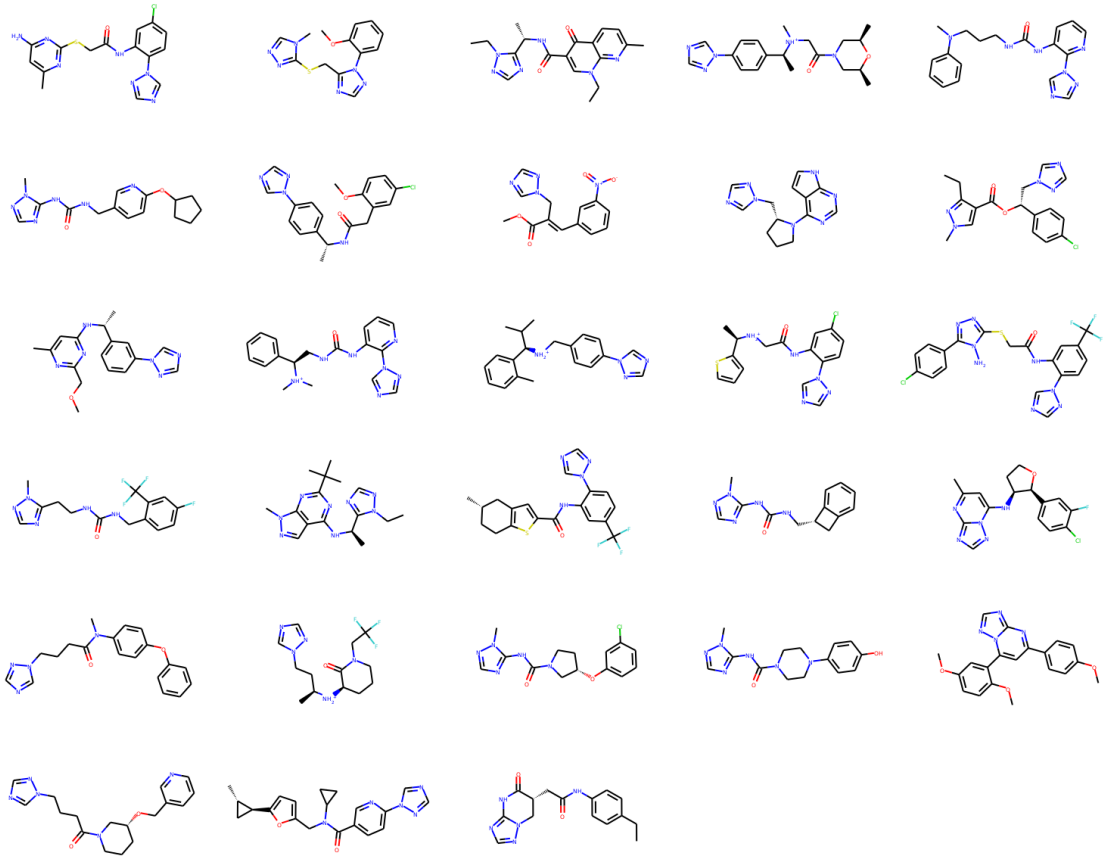

Figure 4.26: 28 random molecules in valid set that contain the triazole ring and the identified dimension is active.

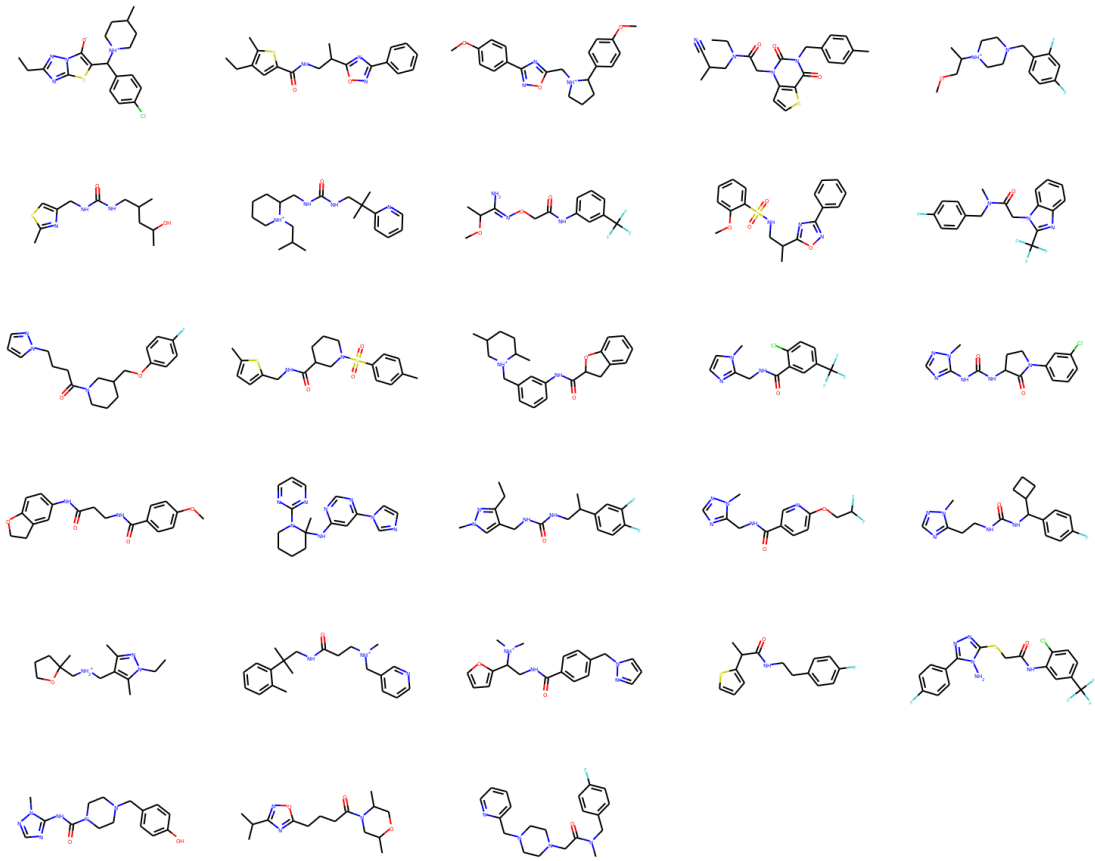

Figure 4.27: The corresponding edited molecules from 4.26 after masking the identified dimension. We can see that the amount of triazole rings has reduced.

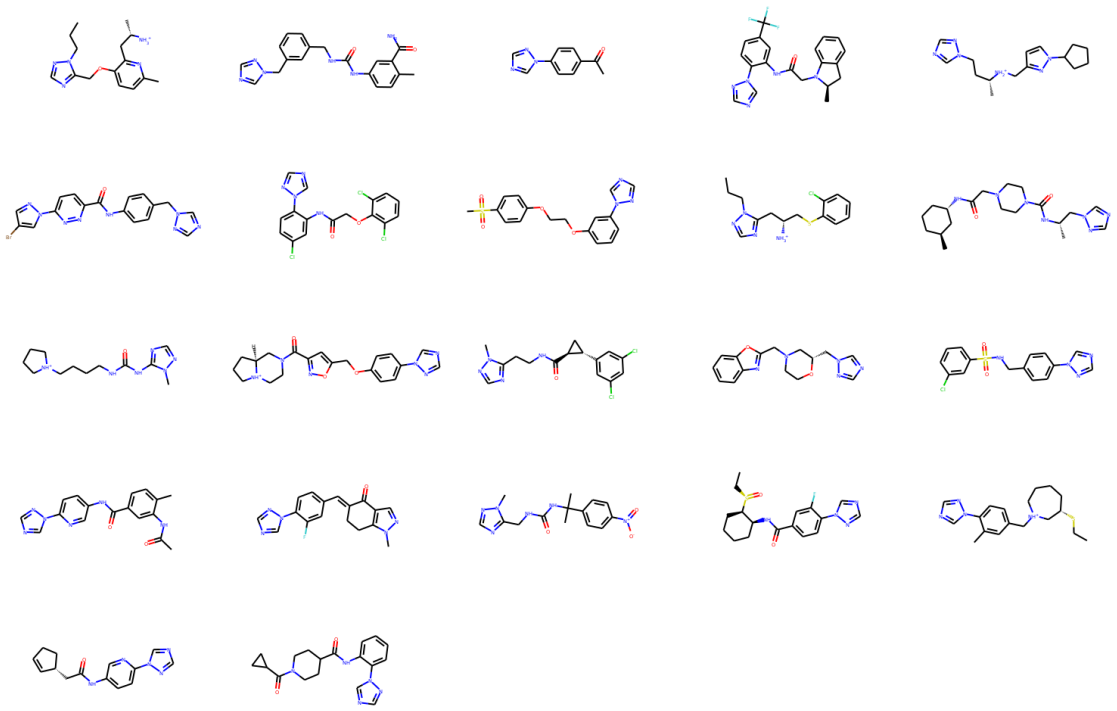

Figure 4.28: 22 random molecules in test set that contain the triazole ring and the identified dimension is active.

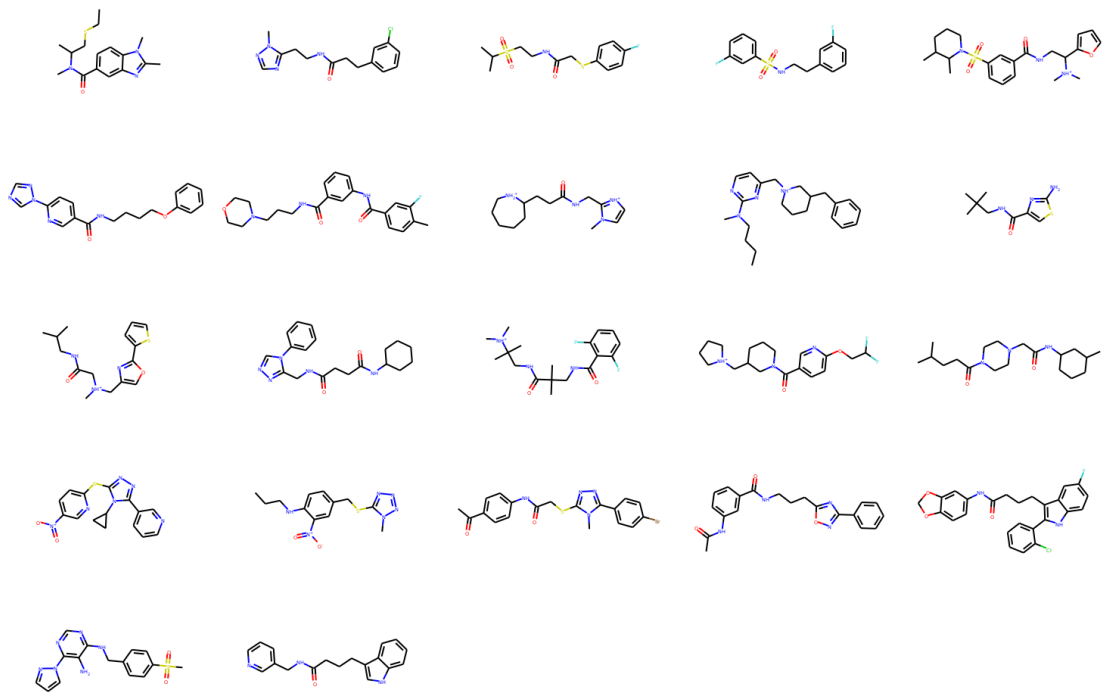

Figure 4.29: The corresponding edited molecules from 4.28 after masking

Chapter 5

Conclusion

In this work, we propose a discrete interpretable variational autoencoder for molecule generation and editing. Despite the model's simplicity, the model learns surprisingly effective interpretable latent representation. We use the interpretability in latent space to provide explicit control over molecular editing. To the best of our knowledge, this work is first of its kind in molecule generation. Current methods perform molecular graph-to-graph translation by optimizing for the betterment of a specific property [30, 29].

Our model is a first step towards using discrete latent variables to perform molecule generation and editing. It could be improved upon in several ways. We can use more powerful priors such as Boltzmann machines on our discrete latent representation [31, 51]. A powerful prior might lead to more disentanglement in the latent space. The qualitative analysis was done with a limited understanding of drug design research. A domain expert in drug design will greatly improve the analysis. Constraints could be placed in latent representation to force the model to give control over specific chemical components. This model can easily be extended to use more expressive categorical random variables instead of binary random variables, but that makes qualitative analysis very complicated.

Bibliography

[1] Syntax-directed variational autoencoder for structured data. `https://mlatgt.blog/2018/02/08/syntax-directed-variational-autoencoder-for-structured-data/`.

[2] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48. ACM, 2013.

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[5] Esben Bjerrum and Boris Sattarov. Improving chemical autoencoder latent space and molecular de novo generation diversity with heteroencoders. *Biomolecules*, 8(4):131, 2018.

[6] Esben Jannik Bjerrum. Smiles enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:1703.07076*, 2017.

[7] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[8] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 891–900. ACM, 2015.

[10] Daniel T Chang. Probabilistic generative deep learning for molecular design. *arXiv preprint arXiv:1902.05148*, 2019.

[11] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[12] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[13] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for structured data. *arXiv preprint arXiv:1802.08786*, 2018.

[14] Andrew Dalke. Deepsmiles: An adaptation of smiles for use in machine-learning of chemical structures. 2018.

[15] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[16] Daniel C Elton, Zois Boukouvalas, Mark D Fuge, and Peter W Chung. Deep learning for molecular generation and optimization-a review of the state of the art. *arXiv preprint arXiv:1903.04388*, 2019.

[17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[18] 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.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[21] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[22] Xiaojie Guo, Lingfei Wu, and Liang Zhao. Deep graph translation. *arXiv preprint arXiv:1805.09980*, 2018.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[24] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[25] Stephen Heller, Alan McNaught, Stephen Stein, Dmitrii Tchekhovskoi, and Igor Pletnev. Inchi-the worldwide chemical structure identifier standard. *Journal of cheminformatics*, 5(1):7, 2013.

[26] Geoffrey E Hinton, Terrence J Sejnowski, and David H Ackley. *Boltzmann machines: Constraint satisfaction networks that learn.* Carnegie-Mellon University, Department of Computer Science Pittsburgh, 1984.

[27] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[28] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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

[30] Wengong Jin, Kevin Yang, Regina Barzilay, and Tommi Jaakkola. Learning multimodal graph-to-graph translation for molecular optimization. *arXiv preprint arXiv:1812.01070*, 2018.

[31] Amir H Khoshaman and Mohammad Amin. Gumbolt: Extending gumbel trick to boltzmann priors. In *Advances in Neural Information Processing Systems*, pages 4061–4070, 2018.

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

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

[34] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[35] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1945–1954. JMLR. org, 2017.

[36] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

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

[38] Qi Liu, Miltiadis Allamanis, Marc Brockschmidt, and Alexander Gaunt. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, pages 7795–7804, 2018.

[39] Tengfei Ma, Jie Chen, and Cao Xiao. Constrained generation of semantically valid graphs via regularizing variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 7113–7124, 2018.

[40] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[41] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[42] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM, 2016.

[43] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

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

[45] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[46] Bidisha Samanta, Abir De, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez-Rodriguez. Nevae: A deep generative model for molecular graphs. *arXiv preprint arXiv:1802.05283*, 2018.

[47] 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.

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

[49] Teague Sterling and John J Irwin. Zinc 15–ligand discovery for everyone. *Journal of chemical information and modeling*, 55(11):2324–2337, 2015.

[50] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[51] Arash Vahdat, Evgeny Andriyash, and William Macready. Dvae#: Discrete variational autoencoders with relaxed boltzmann priors. In *Advances in Neural Information Processing Systems*, pages 1864–1874, 2018.

[52] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[53] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[54] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[55] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[56] 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.

[57] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[58] Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.