Protein-Ligand Binding Affinity Directed Multi-Objective Drug Design Based on Fragment Representation Methods

Muhetaer Mukaidaisi

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Abstract

Drug discovery is a challenging process with a vast molecular space to be explored and numerous pharmacological properties to be appropriately considered. Among various drug design protocols, fragment-based drug design is an effective way of constraining the search space and better utilizing biologically active compounds. Motivated by fragment-based drug search for a given protein target and the emergence of artificial intelligence (AI) approaches in this field, this work advances the field of *in silico* drug design by (1) integrating a graph fragmentation-based deep generative model with a deep evolutionary learning process for large-scale multi-objective molecular optimization, and (2) applying protein-ligand binding affinity scores together with other desired physicochemical properties as objectives. Our experiments show that the proposed method can generate novel molecules with improved property values and binding affinities.

Acknowledgements

The COVID-19 pandemic has had a profound impact on all aspects of our lives, including the pursuit of higher education. Having to shift to remote learning is especially challenging. The lack of in-person instruction and access to campus resources has made it difficult to fully engage with our academic purposes.

As we move into the post-pandemic world, many of us are embracing the new normal and are excited about the opportunities it presents. For me personally, I am grateful to have been able to achieve my research goals, and even exceed my own expectations. I am thrilled to be moving forward with my studies and am looking forward to what the future holds.

I would like to express my deepest gratitude to my supervisor, Dr. Yifeng Li, for his constant guidance, encouragement, and support throughout the entire program. Dr. Li has always been the inspiration for this research. The enthusiasm and expertise he brought to the field have benefited me tremendously in my academic and research endeavors.

My sincere thanks go out to my co-supervisor Dr. Dividino for her consultation and prompt review. Her insights and suggestions were invaluable in shaping the final composition of my work. It would be remiss of me not to thank the committee members, Dr. Sheridan Houghten and Dr. Naser Ezzati-Jivan, and the external examiner, Dr. Ping Liang, for their participation in my research and for providing valuable feedback that assisted me improve my research.

I would like to extend my appreciation to my family, especially my dad, mom, and my sister, for their mental and financial support during this challenging time. The love they have shown me has been my constant source of motivation and strength throughout my life.

In closing, I'd like to thank my partner, Uchqun, for always being there for me, and helping me through the difficult transition of moving to another country and adjusting to a completely new environment. I am grateful for his unwavering support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Drugs are essentially molecules with a pharmacological profile that compromises numerous relevant objectives such as potency, selectivity, pharmacokinetics, and toxicity [68, 26, 25]. Drug discovery is the process of finding new therapeutically useful compounds or repurposing existing ones, with desirable pharmacological properties. After identification of a drug target (often a protein), the traditional wet-lab approach to drug discovery involves preparing a set of molecules with specific properties, studying their structure-property relationships, and optimizing the compound structure. This trial-and-error approach is often costly and ineffective due to the significant number of potential compounds estimated at up to 10^{60} [123, 25]. Typically, developing novel drugs requires billions of dollars in investment and takes decades to complete [84]. In contrast to the traditional approaches that require iterative and collective involvement of domain experts to identify, ennumerate, select possible compounds molecules, and focus on deriving properties from their structures, modern AI-driven drug discovery methods aim at efficiently searching through a vast space of molecules for promising candidates, this can be viewed as either a molecular generation problem or a molecular optimization problem.

Among many different types of machine learning approaches, generative methods have been shown to be very successful in solving molecular generation problems. It involves a generative model that can capture the regularities or patterns of the given set of molecules using a probabilistic distribution, and then generates new plausible molecules following a sampling mechanism. The recent application of distributed representation methods (e.g., word or graph embedding) and deep generative models (DGMs) in drug design [22, 41, 18] enables the modelling of molecular data via a parameterized distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$ where $\boldsymbol{x}$ corresponds to a molecule, $\boldsymbol{z}$ describes the continuous latent representation, and $\boldsymbol{\theta}$ is the set of neural network parameters.

However, difficulties in producing high-quality novel candidates by prior generative methods arise because of the discrete nature of chemical space and the large number of molecules therein.

While generative models generate molecules in consistence with the training data distribution, molecular optimization on the other hand is the process of designing new molecules with the desired properties, rather than naively enumerating the entire molecular space. Molecular optimization problems can be grouped into conditional and unconditional optimization tasks. A conditional optimization task relies on the principle of local optimization given a molecule as a start point and aims at finding structurally similar molecules with better properties, whereas an unconditional optimization task employs global optimization/search techniques. Since drug candidate needs to quantitatively fulfill multiple desiderata, molecular optimization problems are essentially multi-objective optimization problems. This research advances the field of drug design by integrating the deep generative modelling with the molecular optimization methodology to improve future discoveries.

1.1 Motivations

In our previous work, we developed the deep evolutionary learning (DEL) framework [40] for multi-objective molecular optimization, which proposes innovation in extending metaheuristic multi-objective global optimization methods (e.g., multi-objective evolutionary computation [19, 24]) to their corresponding deep versions through the latent representation space of DGMs. DEL achieves the co-evolution of both molecular data and molecular generative models across multiple generations guided by multiple properties concerned with drug design. To optimize the search of the chemical space, DEL adopted the DGM FragVAE for sequential modeling of SMILES (Simplified molecular input line entry specification) [122] string fragments for molecular generation. Although it achieves interesting results in drug design, its fragmentation strategy usually splits a molecule into only 2-4 fragments, which may pose problems in language models due to this level of coarse granularity. While we continue looking for SMILES fragmentation methods at finer granularity, we are interested in investigating graph fragmentation methods for molecular representation in DGMs and DEL.

Molecular docking is the study of how two or more molecular structures (e.g., drug and enzyme or protein) fit together [29], which has been an imperative task since the majority of drug compounds take effect by specifically binding to active sites of the specified protein targets responsible for diseases (such as COVID-19,

AIDS, cancer, autism, Alzheimer's, etc.). When the target protein structure becomes accessible, molecular docking is more frequently involved in *in silico* drug design to assess potential protein-ligand binding interactions [119, 86], where the ligand is usually a molecule that forms a complex with its receptor to modify pathways associated with diseases. Although molecular generation has gained extensive success in constraining the search space, it still remains challenging to identify true ligands among the candidates or generate molecules with high binding affinity to certain targets. There are many computational tools available for analyzing ligand docking on the given binding site and offering quantified evaluation of the docking solution, such as AutoDock [80], AutoDock Vina [113, 23], QVina [5], GOLD [53] and FlexX [93]. Considering the inherent advantages of the evolutionary feature in the DEL framework, this thesis integrates protein-ligand docking evaluation into DEL and evolves the affinity of protein-ligand binding to molecules as an objective in multi-objective optimization.

1.2 Contributions

In this paper, encouraged by the past success of target-specific fragment-based drug design methods and the recent development of AI techniques for drug design, we present an improvement of DEL using the graph-fragment based method, JTVAE, as the deep generative model for multi-objective molecular optimization. Specifically, our work has three major contributions.

In our first contribution, we introduced the representation (i.e., embedding) and generation of molecular graph fragments to DEL through incorporating the graph-fragmentation deep generative model, JTVAE. The original DEL utilizing FragVAE was implemented as a benchmark in order to compare this graph fragment-based DEL with the SMILES fragment-based DEL. As a result, the JTVAE-based DEL demonstrated superior performance in terms of molecular quality to the FragVAE-based DEL.

As our second contribution, we apply the protein-ligand binding affinity score as one of the molecular optimization objectives in DEL, while the original DEL in [40] entails drug-likeness and synthesizability. Based on molecular mechanism simulation, protein-ligand binding scores are calculated for molecules to estimate their binding affinity toward a preeminent target protein surface area. Aiming at identifying active drug candidates, the binding score is viewed as a desired property along with other choices of objectives to be optimized in DEL.

Third, in recognition of the fact that some drugs are expected to bind to more than one therapeutic target when treating human diseases such as several types of tumors and cancer, we further experimented with the concurrent optimization of protein-ligand binding affinity scores on two drug targets, CA9 [107, 30] and GPX4 [28, 45]. Optimization aims to achieve lower scores, which will lead to higher binding affinity. In addition, we screened novel molecules that had druggable characteristics and the potential to be active inhibitors of both protein targets.

1.3 Structure Overview

This thesis is outlined as follows. Chapter 2 explains the foundational concepts of this work including generative modelling, graph theory and evolutionary algorithms. Chapter 3 offers an extensive review of the related research in drug design, more specifically in molecular generation and molecular optimization tasks. Chapter 4 presents the methodology for building JTVAE-based DEL and incorporating ligand docking on different protein targets. Subsequently, Chapter 5 describes the experiment settings, data preparation, results evaluation and virtual screening of the novel samples. Chapter 6 deduces the conclusion, extends the research's impacts on the community and reflects the applicable future work.

Chapter 2

Foundations

2.1 Generative Modeling

In the field of machine learning, discriminative models is a method of decision making by modeling the relation between unknown data y and known data x. It is an approach based on probability theory, knowing the input variables, discriminative models predict y by constructing a conditional probability distribution. A simple example would be upon acquiring a trained discriminative model to classify images containing pixel-level information ($\boldsymbol{x} \in \mathbb{Z}^D$) into classes of certain objectives ($y \in \mathcal{Y}, \mathcal{Y} = \{car, tree, house, dog\}$). To make decision, the model is formulated to parameterize the conditional distribution $p(y|\boldsymbol{x})$, which has later demonstrated drawback in [110] that adding noise to input images will drastically reduce classification accuracy. This is because without knowledge of the full observed variables, the discriminative model couldn't understand the environment, in this case, the semantic information of images.

As a resort, if we consider the joint distribution $p(\boldsymbol{x}, y)$ and apply the product rule to factorize it as $p(\boldsymbol{x}, y) = p(y|\boldsymbol{x})p(\boldsymbol{x})$, it introduces a generative perspective to estimate the distribution over objective $p(\boldsymbol{x})$. Generative models can be used to simulate (i.e., generate) the distribution of any variable in the model, thus are more suitable for unsupervised tasks such as classification and clustering. With the help of neural networks, deep generative modeling is used in a number of areas, including text analysis, audio generation, and image generation. This thesis is most relevant to drug design applications, where deep generative modeling has enabled more computationally affordable exploration of the chemical space, as the probabilistic model can learn from flexible representations of the input data (molecules) and construct molecules with desired physiochemical properties.

The content of the generative model is rich and colorful, and we can classify the techniques according to the processing method of the probability density function: autoregressive models, flow-based models, latent variable models and energy-based models [112]. In this section, we will first introduce basic methods of maximum likelihood estimation, and then present a detailed look at a few representative models in more detail, such as variational autoencoders, generative adversarial networks, and flow-based models.

2.1.1 Autoregressive Models

The autoregressive model (ARM) uses itself as a regression variable, that is, it is a linear regression model that describes random variables at a certain time in the future by using a linear combination of random variables at several previous moments, which is a common concept in time series analysis. There are certain conditions that need to be met in order for autoregressive models to model data with multiple dimensions and features. First, the input space $\mathcal{X}$ requires a deterministic ordering of its features. They can be used for images by defining, for example, that the pixels on the left come before the pixels on the right, and the pixels on the top come before the pixels on the bottom. Second, given the values of previous features autoregressive methods treat $p(\boldsymbol{x})$ as the product of conditional distributions in order to tractably model the joint distribution of features in the data observations. For instance, the probability that a pixel in an image has a particular intensity value depends on the values of all previous pixels; whereas the probability for an image (the joint distribution of all pixels) is the combination of all its pixels. ARMs use the chain rule to decompose the likelihood of a data sample x into a product of one-dimensional distributions as shown in Equation 2.1. The decomposition process transforms the joint modeling problem into a sequence problem which requires the prediction of the next pixel based on the previous pixels.

$$p(\boldsymbol{x}) = p(x_0) \prod_{i=1}^{D} p(x_i|x_1, x_2, ..., x_{i-1}). \tag{2.1}$$

Now, the biggest challenge is to compute these conditional distributions $p(x_i|x_1, x_2, ..., x_{i-1})$. Is it possible to represent these complex distributions in a tractable and scalable manner? A solution would be to use deep neural networks, most commonly convolutional neural networks (CNNs) which is a means of analyzing audio [114] and generating images [115].

Autoregressive models have many limitations:

1. Prediction is sequential thus slow sampling. ARMs perform forward propagation in parallel while obtaining the probabilities of input x_i ($x \in \mathcal{X}, \mathcal{X} = x_0, x_1, ..., x_i$. However, given the probabilities, sampling new values means iteratively sample at every position until we generate the final objective.

2. Autoregressive model uses its own data to make predictions, hence only suitable for predicting phenomena related to its own previous state.

3. Absence of latent representation of the data, resulting the lack of manipulation over the internal data representation.

2.1.2 Flow-based Models

To directly model the likelihood function (i.e., the joint distribution of the high-dimensional input variable $p(\boldsymbol{x})$), flow-based generative models employ the change of variables formula. Assuming a random variable z has a known probability density function $z \sim \pi(z)$, we would like to construct a new random variable using an injective function $x = f(z)$, which will result in an invertible function $z = f^{-1}(x)$ (a bijection). Consequently, the question arises as to how to infer the unknown probability density function of the new variable? Applying the change of variables formula, it could be calculated using the known bijective transformation f:

$$p(x) = \pi(z = f^{-1}(x))|\frac{\partial f^{-1}(x)}{\partial x}|, \tag{2.2}$$

where the change of volume $|\frac{\partial f^{-1}(x)}{\partial x}|$ is to normalize the known probability distribution $z \sim \pi(z)$ after transformation f. Further, if objectives are high-dimensional variables $\boldsymbol{z}, \boldsymbol{x} \in \mathbb{R}^D$, we can derive a similar form in Equation 2.2 [112]:

$$p(\boldsymbol{x}) = \pi(\boldsymbol{z} = f^{-1}(\boldsymbol{x}))|det\mathbf{J}_{f^{-1}}(\boldsymbol{x})|, \tag{2.3}$$

where $det\mathbf{J}_{f^{-1}}(\boldsymbol{x})$ is the Jacobian matrix of transformation f, which is defined as:

$$\mathbf{J}_{f^{-1}}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1^{-1}}{\partial x_1} & \frac{\partial f_1^{-1}}{\partial x_2} & \cdots & \frac{\partial f_1^{-1}}{\partial x_D} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_D^{-1}}{\partial x_1} & \frac{\partial f_D^{-1}}{\partial x_2} & \cdots & \frac{\partial f_D^{-1}}{\partial x_D} \end{bmatrix}. \tag{2.4}$$

Flow-based modeling involves applying a series of reversible transformation functions to transform a simple distribution into a complex distribution, and updating it repeatedly with new variables in accordance with the change of variables theorem to obtain a complex multimodal distribution in the end. As we expand the equation of the output step by step, we can invert back to the initial simple distribution using the relationship between each pair of continuous variables. Therefore, after K transformations, the logarithm of $p(\boldsymbol{x})$ will be:

$$\log p(\boldsymbol{x}) = \log \pi_0(\boldsymbol{z}_0 = f^{-1}(\boldsymbol{x})) - \sum_{i=1}^{K} |\mathbf{J}_{f_i}(\boldsymbol{z}_i - 1)|, \tag{2.5}$$

where the path traversed by a random variable $\boldsymbol{z}_i$ is a flow, and the full chain formed by a continuous distribution π_i is called a normalized flow.

Modeling invertible transformations with neural networks can be considered a suitable solution. However, it must satisfy two properties as specified above. (1) The neural network has to be invertible. (2) The calculation of $\sum_{i=1}^{K} |\mathbf{J}_{f_i}(\boldsymbol{z}_i - 1)|$ has to be tractable. An exemplary flow-based model is RealNVP (Real-valued Non-Volume Preserving) [21] that implements normalizing flow by stacking a sequence of reversible bijective transformation functions. With each bijective function, an affine coupling layer divides the input dimension into two parts by dividing the input high-dimensional variable $\boldsymbol{x}$ into $\boldsymbol{x}_{1:d}$ and $\boldsymbol{x}_{d+1:D}$. After the first d dimensions remain unchanged, RealNVP performs an affine transformation to the $d+1$ to D dimensions, with the coefficients and intercept parameters being functions of the first d dimensions:

$$\boldsymbol{y}_{1:d} = \boldsymbol{x}_{1:d}, \tag{2.6}$$

$$\boldsymbol{y}_{d+1:D} = \boldsymbol{x}_{d+1:D} \cdot exp(s(\boldsymbol{x}_{1:d})) + t(\boldsymbol{x}_{1:d}), \tag{2.7}$$

where $s(.)$ and $s(.)$ are mapped scaling functions and transition functions, and $\cdot$ is dot product.

RealNVP generalizes the coupling layer and introduces a convolutional layer, allowing for a better handling of graph problems. Further, it also proposes the design of multi-scale layers, which can reduce the amount of computation and improve the generation quality by also providing a strong regularization effect. The general framework of the flow model emerges at this point.

2.1.3 Variational Autoencoders

The third family of generative models is latent variable models, which introduce latent variables $\boldsymbol{z}$ as a solution to the factorized joint distribution $p(\boldsymbol{x}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})$. And after marginalize out $\boldsymbol{z}$, the likelihood function yields

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}. \tag{2.8}$$

Unlike autoregressive models and flow-based models that directly model the likelihood function, latent variable models approximate the likelihood function via variational inference, and one principled approach is the variantional autoencoder, or VAE [54].

VAE is closely related to the auto-Encoder (AE) architecture [46] in terms of structure. AE is a deterministic neural network that is designed to learn an identity function unsupervised and reconstruct the original input while compressing the data. This is done to discover an efficient and compressed representation of the original input. AE can reconstruct the input $\boldsymbol{x}$ (to the encoder) using the decoder output $\tilde{\boldsymbol{x}}$. The hidden layer $\boldsymbol{z}$ between the encoder and decoder generates a set of codes to represent the input. Thus, the network can be seen as a harmony of two parts: an encoder represented by the function $\boldsymbol{z} = f(\boldsymbol{x})$ that converts the original high-dimensional input to a low-dimensional latent variable encoding, and a decoder $\tilde{\boldsymbol{x}} = g(\boldsymbol{z})$ that generates reconstructions. The objective is to decode $\tilde{\boldsymbol{x}}$ as close to $\boldsymbol{x}$ as possible, so a common loss function for the autoencoder is $L = \| \boldsymbol{x} - \tilde{\boldsymbol{x}} \|^2$. After training the model, the encoding should retain most of the information from the input data to be able to directly represent the original data in reconstructions, as to achieve the goal of dimensionality reduction.

An AE is not particularly useful if it simply learns to set $g(f(\boldsymbol{x})) = \boldsymbol{x}$ everywhere since this imposes some constraints or regularizations on the AE so that it produces approximately similar data. How come? Because we would not know which *factors* are crucial when generating an object without explicitly modeling the distribution of $\boldsymbol{z}$. The training data is limited, thus $g(\boldsymbol{z})$ may only respond to the limited set of $\boldsymbol{z}$. If we only randomly sample in this large space $\mathbb{R}^D$, we cannot expect to always sample just enough to generate useful information.

Instead of mapping the input to a fixed vector, VAE maps it to a distribution, which is defined as $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ parameterized by $\boldsymbol{\phi}$. If we consider a known Gaussian distribution $\boldsymbol{\phi} = \{\mu, \sigma^2\}$. By multiplying $1 = \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}$, the logarithm of Equation 2.8

can be derived into:

$$\log p(\boldsymbol{x}) \geq \mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z})], \tag{2.9}$$

where $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is the amortized variational posterior, and this lower bound of log-likelihood function is called Evidence Lower Bound (ELBO).

VAEs use deep neural networks to parameterize the generative component and inference component as a family of deep generative models. Even though the latent vector (i.e., the bottleneck of the architecture) is stochastic, the reparameterization trick enables gradient descent to be used for model learning [95]. Appropriately, the encoder (inference) network can be defined as $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ parameterized by $\boldsymbol{\phi}$ and decoder (generative) network $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ with learnable parameters $\boldsymbol{\theta}$. In the training process, the encoder maps input $\boldsymbol{x}$ to the stochastic latent vector $\boldsymbol{z}$ which is then passed to the decoder to generate the reconstruction $\tilde{\boldsymbol{x}}$. Based on this, the following negated variational loss is formulated from Equation 2.9 for minimization:

$$L(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] + \mathrm{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\big). \tag{2.10}$$

The first term above is referred to as reconstruction error, which measures the difference between the original input and the reconstructed output generated by the decoder. It reflects the loss in the process $\boldsymbol{x} \sim \boldsymbol{z} \sim \tilde{\boldsymbol{x}}$, and can also be described as the expected negated log-likelihood [55]. The second term is a regularization term using Kullback-Leibler (KL) divergence [60] in order to measure the proximity between posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and prior $p(\boldsymbol{z})$ [59], which can be formulated as:

$$\mathrm{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})\big) = \int_{\mathcal{X}} q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})} \mathrm{d}\boldsymbol{x}. \tag{2.11}$$

The latent variable $\boldsymbol{z}$ at the bottleneck should approximate the standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$ after model training. Thus, disparate data can be generated by sampling from this distribution and passing them through the decoder.

2.1.4 Generative Adversarial Networks

Another relevant approach to latent variable models is what is known as generative adversarial networks (GANs) by [37], which have a different structure that consists of a generator G and a discriminator D. Generator is responsible for generating new plausible examples from the problem domain, while discriminator works on classify-

ing examples as *real* (from the domain) or *fake* (generated). The two networks are trained as competitors. Generators produce sample data directly. Its adversary, the discriminator, attempts to distinguish between samples drawn from the training data and samples drawn from the generator. During training, this competitive process continues until the discriminator model fails to judge true or false more than half of the time, indicating the generator model is producing very realistic data. That is, given the empirical distribution $p_{data}(\boldsymbol{x})$, the objective is to learn G that generates $p_g = p_{data}$ that D cannot correctly distinguish from.

In GAN architecture, the generator can be any neural network structure $G_{\boldsymbol{\beta}}$ the input is random noise $\boldsymbol{z} \sim p_{\boldsymbol{z}(\boldsymbol{z})}$ (Gaussian distribution, uniform distribution, etc.), and the training goal is to make the output $G_{\boldsymbol{\beta}}(\boldsymbol{z})$ obtain a greater probability in the subsequent discrimination stage. Similar to VAE, GAN is based on a latent space that reflects the compressed representation of the data distribution. G will average the selected points in the latent space. The new points extracted from the latent space can be used as inputs to the generator model for generating new and varied output examples. After training, a generator model is retained for generating new samples.

The discriminator also is a neural network $D_{\boldsymbol{\alpha}}$ but with two sources of data: $\boldsymbol{x} \sim p_{data}(\boldsymbol{x})$ and $\boldsymbol{x} \sim p_\theta(\boldsymbol{x}) = \int G_{\boldsymbol{\beta}}(\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z}$, and the training goal is to accurately distinguish between those two sources. It is a normal classification model that predicts a binary label *real* or *fake*. The two networks are trained alternately, and their capabilities are improved synchronously.

The training phase of GAN is a minmax two-player game [37], and the corresponding objective function is as follows:

$$\min_G \max_D V(D, G) = \mathrm{E}_{\boldsymbol{x} \sim p_{data}(x)}\big[\log D_{\boldsymbol{\alpha}}(\boldsymbol{x})] + \mathrm{E}_{\boldsymbol{z} \sim p(\boldsymbol{z})}\big[\log(1 - D_{\boldsymbol{\alpha}}(G_{\boldsymbol{\beta}}(\boldsymbol{z}))], \quad (2.12)$$

where D targets to minimize the probability of assigning the correct label to both training examples and samples from G, which is to maximize $\log D_{\boldsymbol{\alpha}}(\boldsymbol{x})$ and $\log(1 - D_{\boldsymbol{\alpha}}(G_{\boldsymbol{\beta}}(\boldsymbol{z}))$. While the goal of G is to minimize the probability of generated sample being correctly labeled by D, that is to minimize $\log(1 - D_{\boldsymbol{\alpha}}(G_{\boldsymbol{\beta}}(\boldsymbol{z}))$.

There are still some existing problems of GAN, such as:

- Vanishing Gradient. The discriminator becoming too powerful may result in the gradient vanishing of G. To address this problem, Wasserstein GAN (WGAN) introduces Wasserstein distance as a solution [6].

- A tendency for mode collapse, leading to a lack of diversity in the generated

samples. WGAN and unrolled GAN are two common resorts [73].

- High computational costs. Due to the model's complex structure, it presents high parameter complexity for resource-constrained real-world applications. GAN compression is useful for reducing the cost, which often involves pruning (removing some parameters), quantization (reducing the bit width) and distillation.

2.2 Graph Neural Network

With the advancement of machine learning and deep learning, significant breakthroughs have been made in speech, image, and natural language processing. Despite this, speech, images, and text are merely sequences or grids of information, and deep learning approaches are adept at handling this type of quantified data. In the real world, however, not everything can be represented as a sequence or a grid, for example social networks, biological networks, molecules or complex file systems. Unlike images, graphs have complex topology, are arbitrary in size, and are difficult to model due to their lack of spatial locality. Can deep learning be extended to model this type of data? These problems have prompted the emergence and development of graph neural networks (GNNs). GNN refers to the use of neural networks to learn graph-structured data, extract and explore features and patterns within graph-structured data, and perform tasks such as clustering, classification, prediction, segmentation, and generation.

2.2.1 Graph Theory

Definition of Graph

As a general definition, an undirected graph is an ordered pair $G = (V, E)$ consisting of a set of vertices or nodes $v_i \in V$ or a set of $(v_i, v_j) \in E$ as edges or lines. E is a subset of the 2-element V, in which an edge is associated with two vertices, and its association consists of unordered pairs of the two vertices. A directed graph, on the other hand, is an ordered pair $G = (V, E)$ where the edge set $(v_i, v_j) \in E$ are ordered pairs of vertices $v_i \in V$. When a simple undirected graph consists of exactly one edge connecting each pair of distinct vertex points, it is considered a complete graph. Whether it is an undirected graph or a directed graph, the sequence of all vertices (including these two vertices) passing from one vertex to another is referred to as a path. A path is called a *loop* (or *cycles*) if it joins a vertex to itself.

In graph theory, *weight* refers to a numerical value assigned to each edge in a graph. This value represents the cost, distance, or any other relevant metric that describes the relationship between the two vertices connected by the edge. A weighted graph is a graph in which the edges are marked with weights. Weights are used to calculate various algorithms for graphs, such as shortest paths and minimum spanning trees [38], in order to determine the most efficient solution. Graphs are considered unweighted if their edges do not bear a specific weight or cost. An unweighted graph serves primarily to describe relationships between objects without considering any quantitative aspects of those relationships.

Graphs can be used to represent a variety of relationships, including those in the fields of medicine [116], biology [72], social media [39], or information systems [2]. These real-world applications have led to the definition of *network* as a graph in which attributes (e.g. names) are associated with vertices and edges.

Spanning Tree

Graph theory defines a spanning tree as a subset of a graph G with the fewest possible edges connecting all vertices. This means that there are no cycles in the spanning tree and it cannot be disconnected. Accordingly, every connected and undirected graph G contains at least one spanning tree. Since a disconnected graph cannot span all vertices, it does not have a spanning tree. Figure 2.1 illustrates that we can construct three spanning trees from a complete graph with three nodes. A complete undirected graph can have a maximum of $n \times n - 2$ spanning trees, where n is the number of nodes. In this example, n =3 forwarding exists three possible spanning trees.

In general, mathematical properties of spanning trees includes:

- A spanning tree has $n - 1$ edges, where n is the number of nodes (vertices).
- A spanning tree can be built by removing the largest $e - n + 1$ edges in the corresponding complete graph.
- A complete graph can have up to $n \times n - 2$ spanning trees.

Spanning trees are mainly used to find the shortest path between all nodes in a graph. There are many applications of spanning trees, including civil network planning, computer network routing protocols, and cluster analysis, etc.

In a weighted graph, a minimum spanning tree is a spanning tree whose sum of edge weights is less than that of all other spanning trees in the graph [38]. There can

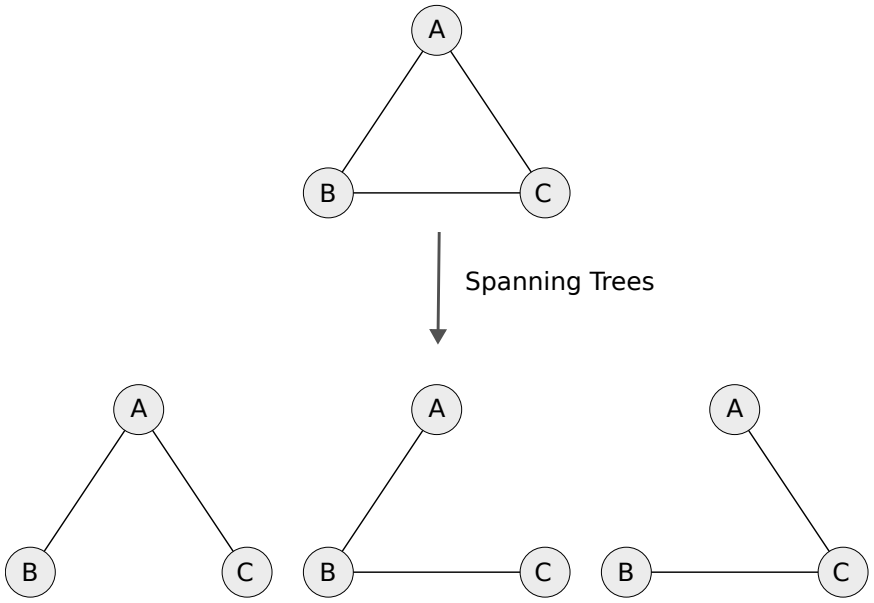


Figure 2.1: An instance of a complete graph and its spanning trees.

be one or more minimum spanning trees in one graph, and the number of minimum spanning tree edges equals the number of vertices minus one.

Adjacency Matrix

It is possible to represent the graph by storing the adjacency matrix in two arrays. A one-dimensional array stores information about vertices in the graph, and a two-dimensional array (called an adjacency matrix) stores information about edges in the graph.

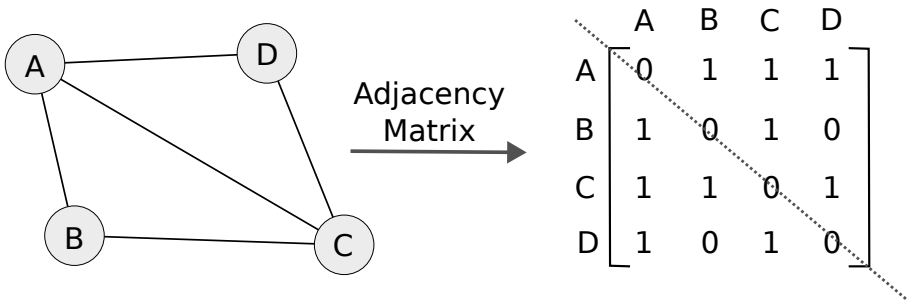


Figure 2.2: The depiction of an undirected graph using adjacency matrix.

When storing a undirected graph, as showing in Figure 2.2, we set up two arrays, the vertex array is $vertex[4] = \{A, B, C, D\}$, and the edge array $edge[4][4]$ is actually a matrix. For the values of the main diagonal of the matrix, i.e. $edge[0][0]$, $edge[1][1]$

, $edge[2][2]$, $edge[3][3]$ are all 0s since there is no loop. This example states that the adjacency matrix of an undirected graph is a symmetric matrix.

2.2.2 Graph Embedding

Graph/Network embedding and graph neural networks are two related areas of research. Graph embedding represents the graph as a low-dimensional vector space, while maintaining the topology and node information of the network, so that machine learning algorithms can be applied directly to graph analysis tasks in the future. It should be noted that some deep learning-based graph embeddings also belong to graph neural networks, such as graph autoencoders and graph convolutional neural networks.

The network representation using the adjacency matrix has the problem of computational efficiency. The adjacency matrix A uses the storage space of $|\mathcal{V}| \times |\mathcal{V}|$ to represent a graph. As the number of nodes grows, the space required for this representation grows exponentially. At the same time, the sparseness of the data makes it difficult to apply fast and effective learning methods.

Real graphs (networks) are often high-dimensional and intractable. Graph embedding is essentially a dimensionality reduction algorithm that maps the high-dimensional features of each node to low-dimensional features. The graph embedding process involves first constructing a $\mathbb{R}^D$ space graph in accordance with the actual problem, and then embedding the nodes of the graph into $\mathbb{R}^d (d \in D)$ space.

The idea of embedding is to keep connected nodes close to each other in a vector space. Local linear embeddings and Laplacian feature maps are algorithms that are based on this principle. However, the scalability of this approach is a significant concern, and it has a time complexity of $O(|V|)$.

The focus of recent graph embedding research has been on scalable embedding techniques that address network sparsity. For example, graph factorization uses an approximate factorization of the adjacency matrix as the embedding. LINE (large-scale information network embedding) [111] extends this approach and attempts to preserve first- and second-order approximations. In an effort to preserve high-order proximity, HOPE (high order proximity preserved embedding) [85] applies generalized singular value decomposition (SVD) to decompose the similarity matrix, rather than the adjacency matrix. SDNE (structural deep network embedding) [120] uses autoencoders to embed graph nodes and capture highly nonlinear dependencies. The time complexity of these novel scalable methods is $O(|E|)$.

2.2.3 Graph Neural Network Models

An adjacency matrix is employed in the graph neural network as opposed to the fully connected layer (MLP), where the feature matrix is multiplied by the weight matrix. The input to the more common application mode of a graph neural network is a graph. After various operations such as multi-layer graph convolution and activation functions, the representation of each node can be obtained to facilitate node classification, link prediction and the generation of graphs and subgraphs.

Graph Convolution Networks

Graph convolutional networks (named as CGNs) proposed by Kipf et al. [57] generalize convolution operations from traditional data (such as images) to graphs. Essentially, the goal is to learn a function map f, through which node v_i can aggregate its own feature x_v and its neighbors' feature $x_u, u \in N(v)$ ($N(v)$ denotes the set of neighbor nodes v) to generate a new representation of node v_i. Thus, GCN is about the flow and dissemination of features and messages. GCNs form the basis of many complex graph neural network models, such as autoencoder-based models, generative models, and spatiotemporal networks.

GCN methods fall into two categories: spectral-based and spatial-based. From a graph signal processing perspective, spectral-based approaches introduce filters to define graph convolution, where graph convolution operations are interpreted as removing noise from graph signals. Spatial-based methods represent graph convolutions as aggregations of neighborhood features. When the algorithm of graph convolutional networks operates at the node level, graph pooling modules can be interleaved with graph convolutional layers to coarsen the graph into high-level substructures.

Bruna et al. [11] adapted convolutional neural networks to graph data for the first time, proposing two parallel graph convolution models: spectrally decomposed graph convolution and spatial graph convolution. Spatial graph convolution addresses the spatial characteristics of graph structure data, as well as the representation of neighbor nodes, so that neighbor nodes of each node are uniform and regular, which is convenient for convolution operations. Several key problems exist with the spatial graph convolution method, primarily (1) choosing the central node, (2) selecting the size of the receptive field, that is, determining the number of neighbor nodes, and (3) how to handle the characteristics of the neighbor nodes, i.e. how to construct a suitable aggregation of neighbor node features.

Graph Auto-encoders

GNN based on autoencoders is known as graph autoencoders (GAEs) [56], unsupervised learning frameworks that encode nodes/graphs into latent vector spaces and reconstruct graph data from that information. GAE is used for learning network embeddings and graph generation distributions. For network embeddings, GAE normally learns latent node representations by reconstructing graph structural information in the adjacency matrix [56, 120]. When it comes to graph generation, some methods produce nodes and edges incrementally, while others generate the graph all at once.

Graph Attention Networks

Attention mechanisms allow neural networks to focus only on the information required for task learning, and they can select specific inputs. By incorporating an attention mechanism into GNN, the neural network is able to focus on nodes and edges that are most relevant to the task, improving the effectiveness of training and the accuracy of testing, resulting in the formation of a graph attention network (GAT) [117]. There are other GNN models built on attention mechanisms such as gated attention network (GaAN) that employs a multi-head attention mechanism to update the hidden states of nodes [133], and graph attention model (GAM) which processes graph information by adaptively visiting a sequence of critical nodes containing target information [63].

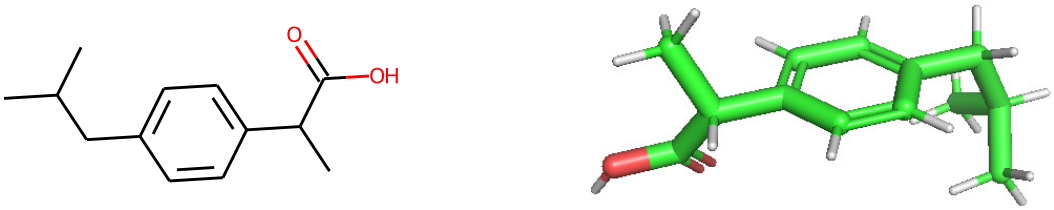
2.3 Molecular Representation in Drug Design

Molecular generation tasks require input information in the form of molecules, and with molecular structures to be learned, it raises a question: what molecular representation scheme should be employed to enable a better understanding of the chemical structure? For machine learning algorithms, the numerical variable can be directly used as input, whereas for categorical variables, the most common way is to represent it by one-hot encoding. Similarly, chemical structures can be represented by their original graph features, or by a set of numbers, such as fingerprints or other descriptors. They exhibit a variety of molecular properties, such as LogP, molecular weight, hydrogen bond donors, acceptors, and rotatable bonds, among others. Different molecular representations have their own advantages and disadvantages. Most often, one or more of them is selected based on the structure of the machine learning model. In general, the higher the descriptor's dimension, the greater its information

content. It consists essentially of three categories: graphs, molecular fingerprints, and SMILES notation.

2.3.1 Graph Representation

Molecules are essentially atoms and chemical bonds connecting atoms to each other, so they can naturally be represented using the mathematical structure of graphs, which exhibit rich structural and spatial information. Given a two-dimensional (2D) graph $G = (V, E)$, the typical procedure of molecular graph representation is to map the atoms into nodes $v_i \in V$ and treat the bonds as edge $(v_i, v_j) \in E$. Graphs typically have nodes defined as circles or junctions with letters indicating their atom types. Figure 2.3 **(A)** depicts an example of 2D graph representation in terms of the ibuprofen molecular structure [126].



A **B**

Figure 2.3: 2D **(A)** and 3D **(B)** graph representation of ibuprofen molecule.

As shown in Figure 2.3 **(B)**, in three dimensions, molecules are viewed in space with x, y, and z coordinates. Therefore, three-dimensional (3D) representation can contain a high level of information that may be useful for modeling biological endpoints. It accounts for spatial and geometric configuration, shape-based information, conformation-dependent distances, and surface properties (e.g. van der Waals force, solvent-accessible surface area). There can, however, be some complications when it comes to selecting the appropriate optimization method. In addition, generating 3D conformations, evaluating and finalizing optimized conformations, and determining 3D descriptors can take a significant amount of time.

The graph $G = (V, E)$, being a 2D object, can also embody 3D information about molecules in order to access their quantum properties. The 3D graph representation

of molecules is complementary to the 2D topology, and tends to include the geometric distribution of atoms, specifically their positions around stereocenters, axes, and planes. A common solution is to encode the spatial relationships between the nodes as node or edge attributes [17], such as atomic coordinates, bond angles and chirality.

2.3.2 Molecular Fingerprint

The most common form of fingerprint is the use of a series of binary digits (bits) that encode the presence or absence of molecular neutron structures using sparse vectors. Therefore, drugs (small compounds) are described as vectors (arrays) of 0s and 1s that can represent molecules in machine learning models based on the substructure present in a particular molecule. Drugs (small compounds) can therefore be viewed as vectors (arrays) of 0s and 1s for the purpose of representing molecules in machine learning models based on their substructure.

The molecular fingerprint has the advantages of simplicity and speed. Nevertheless, encoding molecules into binary vectors is not a reversible process, but rather a lossy transformation. Namely, we can encode a molecular formula that can represent structural information into a molecular fingerprint, but we cannot infer the structure of the molecule from only the fingerprint.

2.3.3 SMILES Representation

An alternative means to represent molecules is to encode molecular structures by converting graph-structured data into textual content and using text (encoded strings) as input in a machine learning input pipeline. SMILES [122] is a popular specification in the form of a line notation for describing the structure of chemical species using short ASCII strings, thereby facilitating the use of corresponding algorithms in natural language processing (NLP) in processing molecules. To continue with the example of ibuprofen, its chemical formula is $C_{13}H_{18}O_2$, which can be expressed in SMILES string format as "CC(C)CC1=CC=C(C=C1)C(C)C(=O)O" according to its graph structure.

SMILES characterization is usually obtained by expanding molecular graphs using depth-first search. To describe a three-dimensional chemical structure, it must convert the chemical structure into a spanning tree, and adopt the vertical priority tree traversal algorithm. As part of the transformation, hydrogen must be removed first, and the ring must be opened. It is recommended to denote the atoms at the ends of the bonds to be removed by numbers, and the branches by parentheses. The

majority of molecular editing software can import and convert SMILES strings into 2D graphics or 3D models of molecules. One molecule often appears in different forms of SMILES notation as a result of the different starting points and directions in the depth-first search.

2.4 Evolutionary Algorithms

2.4.1 Overview

From Lamarck's theory of evolution to Darwin's theory of evolution, to Mendel's genetics, human research on the phenomenon of life evolution has reached unprecedented levels of significance. Darwin's theory of evolution states that organisms on earth may undergo mutations during reproduction, leading to the development of new species. Due to limited resources, when competition between different species occurs, the fittest survive and the unfit are eliminated. This principle of survival of the fittest constantly governs the evolution of nature. The idea of using evolutionary theory to solve problems was developed into different implementations including genetic algorithms [47, 4], genetic programming [58], evolution strategies [94] and evolution programming [31], later merged into a family of algorithms, evolutionary algorithms (EAs). In comparison to other methods, genetic algorithms (GA) are relatively mature and have been widely applied for a long time. And there is a growing interest in using evolutionary strategies (ES) [125] and evolutionary programming (EP) in scientific research and in solving practical problems.

Evolutionary algorithms are probabilistic search algorithms based on biological evolution mechanisms, which generally involve basic operations such as reproduction, mutation, recombination, and selection [118, 4]. Compared to traditional optimization algorithms such as calculus-based methods and exhaustive methods, evolutionary computing is a mature global optimization method with high robustness and wide applicability, and it is self-organizing, self-adapting, and self-learning in nature. Moreover, EA is not constrained by the nature of the problem, so it can effectively deal with problems that are difficult to solve with traditional optimization algorithms, in particular NP-hard problems [129].

A comprehensive context for evolutionary algorithms based on genetic algorithms often involves an optimization problem, with populations of its candidates (individuals or phenotypes) evolving through generations towards superior solutions. An individual's fitness is estimated based on his or her properties (genotypes) that make

maneuvering easier, and is expressed as the value of the objective function (fitness function) of the original optimization problem. Individuals with higher fitness will be selected and subjected to a series of evolutionary operators, and formulate the new generation for the next iteration of the algorithm. The overall process contains the following steps:

1. Population initialization: design an appropriate initialization operation according to the characteristics of the optimization problem to initialize the individuals in the population.

2. Individual evaluation: calculate the fitness value of individuals in the population according to the optimized objective function.

3. Generation settings: configure the termination conditions and the current state of the generation process.

4. Individual selection: design a suitable selection operator to select individuals in the population, and the selected individuals will enter the mating pool to form the parent population for evolutionary operators to generate new individuals [8, 36]. Selection should be based on individual fitness values. When the optimization problem is a minimization problem, an individual with a lower fitness value should have a greater probability of being chosen. Among the most commonly used selection strategies are roulette selection, tournament selection [79], etc.

5. Crossover operator: determine whether the parental individual needs to be crossed using the crossover probability [75, 124]. The process mimics how two parent individuals exchange chromosome segments to create new individuals. The crossover operator should be designed with respect to the characteristics of the problem to be optimized.

6. Mutation operator: determine whether the parental individual needs to be mutated using the mutation probability [74, 9]. It refers to the process of forming new chromosomes by changing some elements of the chromosomes. As mutation operators are intended to maintain the diversity of the population and prevent the population from falling into a local optimum, they are usually designed as random transformations.

7. Termination. When the crossover and mutation operation is completed, the parent population produces individuals (offsprings) of appropriate size for the

population, and the generation process repeats (back to step 4) until a termination condition has been reached, which may be a maximum or fixed number of generations, a minimum requirement for candidate solutions, or an allocated budget of computation cost.

Selection, crossover and mutation are classic evolutionary operators. Many improved evolutionary algorithms are built around them. While they may differ in name, they are inseparable from their common origin.

2.4.2 Multi-Objective Evolutionary Algorithm

When several conflicting objectives need to be dealt with simultaneously in optimization problems it is called multi-objective optimization problems (MOPs), which have multiple objective functions [76, 33]:

$$\min_{x \in X}(f_1(x), f_2(x), f_3(x), ..., f_k(x)), \tag{2.13}$$

where $X \subset \mathbb{R}^D$ denotes the feasible solution set and there are K objectives to balance.

It is impossible for MOPs to attain optimality of all objectives simultaneously, and only a set of equilibrium solutions can be obtained, which are called non-dominated solutions or Pareto optimal solutions (Pareto is a concept from economics). Equilibrium solution means that in multiple objective functions, none of the objective value can be improved without degrading the value of other objectives. In particular, $x_1 \in X$ is a non-dominated solution, $x_2 \in X$ is the dominant solution if they follows the relation:

$$\forall i \in \{1, 2, ..., k\}, f_i(x_1) \leqslant f_i(x_2) \ \ \& \ \ \exists i \in \{1, 2, ..., k\}, f_i(x_1) < f_i(x_2) \tag{2.14}$$

When the solution dominates x_1 does not exist, x_1 is called the Pareto optimal solution [108, 76]. The set composed of all Pareto optimal solutions is called Pareto optimal set (Pareto set, PS). The mapping of the Pareto optimal set in the target space is the Pareto front (PF).

The complexity of multi-objective optimization makes traditional mathematical methods incapable of providing optimum results, and evolutionary algorithms have been widely used in multi-objective optimization, which has led to many classical multi-objective evolutionary algorithms (MOEAs) [100, 32, 14]. Through continuous evolution, multi-objective evolutionary algorithms can approximate Pareto sets [134]. The collected approximation set can be evaluated by two criteria: convergence to

measure the degree of closeness between the approximation set and the true Pareto set, and distribution to indicate how the approximation front is distributed in Pareto front, including uniformity and diversity. Specifically, the two commonly used indicators are IGD (inverted generational distance) and HV (hypervolume). IGD, for example, requires the Pareto front. It measures how far the elements in the true Pareto front are from those in the nearest approximation front. IGD is defined as $IGD = \frac{\sum_{v \in P} d(v,Q)}{|p|}$. Q represents the approximation front, $|P|$ is the number of individuals of the true Pareto front, and $d(v, Q)$ corresponds to the Euclidean distance between each of these individuals v and the nearest member of the approximation front. IGD and HV can both measure the distribution and convergence of the solution.

NSGA-II (non-dominated sorting genetic algorithm II) [19] is the most representative application of MOEA. A main characteristic of this method is fast non-dominated sorting, which merges the solution generated by each crossover mutation with the individuals from the previous generation. It then uses non-dominated sorting to stratify the individuals. As layers of rank are accumulated until they exceed the population size, crowding distance sorting is implemented to construct the new population. NSGA-II uses fast non-dominated sorting to guarantee convergence and crowding distance to guarantee distribution. At the end of the iteration, most of the solutions are non-dominated, and most of the solutions are in the first layer.

Chapter 3

Related Work

3.1 Deep Learning on Chemical Structures

Substantially, computation methods for evaluating chemical structures rely heavily on molecular representations, i.e., the form in which a molecule structure is seen by the algorithm. Molecular representation typically uses one of two methods: SMILES-based method, graph-based method, or three-dimensional method, which are discussed below.

3.1.1 SMILES-Based Method

SMILES (Simplified Molecular Linear Input Specification) characterization of molecules obtained by the graph-to-text mapping algorithm is a method of encoding the compositional and structural information of molecules that has been widely used to store molecular structures in cheminformatics. Natural language processing (NLP) algorithms can be naturally leveraged in the field of molecule modeling due to the sequence-based embedded representations of SMILES. As part of the SMILES-based approach, chemical species can be described as short ASCII strings in the form of line notations, and a variational autoencoder (VAE) [54] is often used as a character-based language generative model for SMILES strings in order to enable efficient molecular generation and optimization based on the open-ended latent representation space (as chemical spaces) [42].

As a prototypical approach, CVAE [41] converts a discrete SMILES representation of a molecule to a continuous multidimensional representation. This allows for the efficient search and optimization of compounds in chemical space. A high level of fidelity was observed (through the reconstruction of SMILES strings) as well as

the ability to capture the features of the training set. It has been demonstrated that a trained VAE is capable of predicting properties accurately and implementing gradient-based optimization in the resulting smooth latent space. SD-VAE (Syntax-directed VAE) [16] proposes stochastic lazy attributes that integrate the syntactic and semantic checking of SMILES strings into VAE to constrain the decoder directly. In comparison to CVAE, SD-VAE has been proved to produce a more discriminative representation of latent space. [98] further introduced the Generative Topographic Mapping (GTM) to the sequence-to-sequence VAE in order to shape the latent space around the molecular properties of interest. GTM latent spaces contain a grid that maps the latent vectors to the 2D coordinates of the nodes, thereby creating different densities of actives and inactives. They also adopted Bidirectional Long Short-Term Memory (BLTSM) networks in the encoder to process the SMILES sequence from both directions while utilizing the future context.

Other generative models are used in SMILES-based molecular generation tasks, such as Transformer (SMILES Transformer [48]), GAN (LatentGAN [92]), and RNN-based pre-trained sequence-to-sequence models [109]. A statistical chemical language model composed of LSTM networks was originally proposed by [102] to generate focused molecule libraries for drug discovery. With the aid of a scoring function, the pretrained model was fine-tuned to smaller sets of active drug molecules toward a given biological target, producing novel molecules with desired properties for drug development. This approach also incorporates molecular structure optimization in the generation and enables the generation of high score *novo* molecules.

Synthesis-based methods for molecule generation are efficient in terms of training and sampling, yet have the drawback of producing a large number of chemically invalid and repetitive molecules during generation. In recent years, fragment-based drug design has gained attention as an alternative approach to the design of novel compounds. Molecule fragments are available from commercial fragment libraries [106] or can be produced through various fragmentation techniques. As a modification of the SMILES fragment-based drug design approach originally presented in [88], FragVAE [40] utilizes the Breaking of Retro-synthetically Interesting Chemical Substructures algorithm (BRICS) [20] to collect sequences of fragments from molecules. The BRICS methodology integrates more elaborate medicinal chemistry rules and decomposes molecules by breaking strategic bonds that can be used to recombine chemical motifs. In analogy with NLP tasks, the sequences of fragments can be viewed as "sentences", each fragment representing a "word", embedded within a continuous latent space based on a vocabulary of unique words [78, 77]. FragVAE

employs a gated recurrent unit (GRU) [13, 66] encoder to map embeddings into a stochastic latent representation space. A latent vector (either generated by the encoder or sampled from the prior distribution) is used as the initial hidden state of a GRU-based autoregressive decoder to calculate the probability of the next possible fragments. As a result of a greedy selection strategy, the fragment sequence with the highest probability is then reassembled into a molecule.

3.1.2 Graph-Based Method

Molecular graph generation is similar to SMILES string generation in that nodes (atoms) and edges (bonds) are added sequentially to a graph. In addition to conventional molecular generation models, graph-based methods [105, 97] leverage molecular topology information explicitly in the generator to learn a molecular structure as a natural two-dimensional or three-dimensional graph.

The VAE has been used as a base for generating molecule graphs directly from latent vectors. First, the fundamental model for incorporating graph neural network (GNN) into VAE is formulated in [67] by adding new structures in steps to account for the sequential composition of molecular graphs. Specifically, new structure refers to a new node or edge. CGVAE (Constrained Graph VAE) [70] further investigates the VAE architecture with gated graph neural networks (GGNNs), in which gradient ascent is used to optimize the graph properties locally. In order to better shape the VAE for molecular generation, domain-specific constraints are imposed using binary masks. When graph VAEs reconstruct molecular structures, solving the problem of graph isomorphism is computationally expensive. As a result, graph reconstruction is extremely ineffective and inaccurate without imposing constraints.

The maneuvering of other generative models in drug design persists such as GAN-based models [18], flow-based models [103, 132], or RL (Reinforcement Learning)-based models [131]. MolGAN established in [18] utilizes generative adversarial networks (GANs) to directly manipulate graph-structured molecule data. In the framework, reinforcement learning objectives are integrated to promote the production of molecules with specific chemical properties. Based on experiments with the QM9 chemical database, the model was able to generate nearly 100% valid compounds. Shi et al. [103] proposed a state-of-the-art graph generation model, GraphAF, which integrated graph normalizing flow and autoregressive approaches, where they parameterized an invertible autoregressive flow transformation of molecule structure into Gaussian distribution. In comparison with graph convolutional policy network

(GCPN) by [131], GraphAF exhibits twice the training speed while maintaining 68% validation in molecules generated without chemical rules.

At present, the junction tree variational autoencoder (JTVAE) [51], which we view as a graph-fragment based approach [89], is one of the most successful approaches for transforming molecular graphs into meaningful latent vectors. JTVAE decomposes training molecules into a set of molecular substructures including rings, functional groups, and atoms. As opposed to the conventional node-by-node generation of graphs, JTVAE generates these building blocks in two stages: (1) representing the effective brackets and their arrangement as scaffolding trees, and (2) integrating the entire tree into a graph by adding edges between intersecting components. In order to optimize molecular properties, JTVAE was built on top of the junction tree-based encoder-decoder neural network and refined with graph-to-graph transformation [15, 35] and autoregressive techniques [61, 81]. In later work, the authors of JTVAE extended their graph VAE architecture by introducing larger graph motifs instead of small substructures. The method is referred to as motif-based hierarchical encoder-decoder (HierG2G) model [52], which allows for more flexible substructure representation and more efficient molecular graph reconstruction.

Fragment-based approaches to molecular graphs are becoming increasingly popular. Chen et al. [12] developed a novel deep generative model named Modof (modifier with one fragment) that predicts a single cleavage site in a molecule and removes and/or adds fragments at that site to modify molecules. To modify different fragments at multiple disconnection sets, they developed a pipeline of multiple Modof models in Modof-pipe. It is evident that Modof-pipe can preserve major molecular scaffolds, allowing better control over intermediate optimization steps and constraining molecular similarity. FAME [87] proposes an innovative generative model specifically for phenotypic drug design that is capable of learning the conditional distribution $P_\theta(molecule|phenotype)$ on gene expression profiles derived from molecular graphs. Using an encoder-decoder scheme, it consists of a conditional graph encoder and a fragment-based decoder which assembles fragments in an autoregressive manner.

Further developments in graph-based molecular modelling include 3D-graph and geometric deep learning methods to exploit the geometric properties of molecular structures [7, 71]. In the latest research, [91] explored the possibility of expanding molecule fragments with specific physicochemical properties, especially docking on target protein sites, by attaching them to growing seed molecules. Leveraging a 3D atomic point cloud representation and E(3) equivariant neural networks, it was able to produce chemically valid molecules with preferable binding affinity that are 100%

chemically valid.

3.2 Evolutionary Molecular Optimization

The prominent pursuit of designing molecules with desirable properties has achieved many goals in medicinal chemistry with efficient exploration of the molecular space. Various works based on evolutionary algorithms [130, 50, 83] have shown that they outperform traditional generative drug design frameworks.

CReM [90] is a fragment-based framework for the generation of structures that facilitates the use of genetic operators, explicitly, mutations. There are three steps in the workflow: GROW which replaces a hydrogen atom with another fragment in the interchangeable fragment database, MUTATE which changes fragments of choice, and LINK which connects two fragments after their hydrogen atoms have been removed. In addition to the novelty and diversity advantages of fragment-based approaches, CReM sampling also exhibits a higher potential for synthetic feasibility through evolutionary optimization. Genetic expert-guided learning (GEGL) framework [3], which was developed more recently, proposes integrating reinforcement learning and evolutionary iteration in order to prioritize candidates with high reward. By employing genetic operators like graph-based crossover and mutation on the graph structure of the parent molecules, GEGL is able to produce a population of seed molecules for enriching the generative model training.

3.3 Protein-Ligand Interaction

While a drug-like molecule needs to fulfill physicochemical and structural feature requirements, such as Lipinski's rule of five [69] as a rule of thumb for druggability, it is imperative that the molecule specifically binds to the expected binding site of a protein target. Using molecular mechanism simulation, protein-ligand docking is the standard technique for virtual screening. Among many other efficient open-source docking tools, Rosetta [64] and AutoDock Suite [23] are widely adopted by the research community.

Machine learning approaches have been exploited to predict drug-target interaction (DTI) or drug-target binding affinity (DTBA) through learning on heterogeneous biological data of known interactions to understand the mechanism of drug actions. For example, AutoDTI++ [96] uses a denoising autoencoder that reconstructs the drug-target interaction matrix by adopting denoising empirical loss, which emphasizes

interaction prediction while discarding the loss of missing values. The model input is composed by multiplying the drug-target interaction matrix by the fingerprint-drug matrix for additional information on drug fingerprints. [1] introduces the DeepCDA model containing a training encoder and a test encoder for cross-domain binding affinity prediction of novel drug-protein pairs. The adversarial discriminative domain adaptation (ADDA) technique is utilized in the test feature encoder to map the marginal distribution from both training and test domains into one same feature space. They also proposed a combination of convolutional neural network (CNN) and long-short-term memory (LSTM) neural network with the aid of a two-sided attention mechanism for encoding the interactions between the compound substructures and protein subsequences.

Chapter 4

Methods

The idea of our approach is to integrate the graph fragment-based deep generative model, JTVAE, into the DEL framework, whereby JTVAE provides a latent representation space for the multi-objective evolutionary algorithm (MOEA) to explore, while meanwhile elite samples from each generation of MOEA continue to learn JTVAE. The prior FragVAE-based DEL is implemented as a benchmark for the SMILES fragment-based molecular design method. To identify active drug candidates, both approaches incorporate protein-ligand binding affinity scores (denoted by BAS) [26, 68], synthetic accessibility scores (SAS), and water-octanol partition coefficients (LogP).

The subsections below describe the methodology for preliminary works, including the deep evolutionary learning framework (DEL) and multi-objective optimization. We then discuss the integration of FragVAE and JTVAE within the DEL framework. As well as the implementation of the docking module for BAS calculation, as well as the measurement tools for evaluating the generation and optimization processes.

4.1 Deep Evolutionary Learning Framework (DEL)

Proposed in [40], the DEL framework incorporates multi-objective evolutionary computation with the deep generative models for molecular optimization by establishing a data-model co-evolution paradigm. In contrast to traditional evolutionary algorithms that encode genotypes in the original problem space, DEL instead employs evolutionary operations in the latent representation space of molecules to feedback the evolved data with desired properties, leading to the fine-tuning of the deep generative model as well. The DEL framework comprises the following components.

1. For the first evolutionary generation, the DGM (usually a VAE) is parameterized and pretrained on the training data, and the first population is sampled from the original training data; whereas the population samples of the successive generations are obtained from the DGM.

2. The samples are transformed into latent vectors with the help of an encoder. Meanwhile, the samples are processed to assign molecular properties and are subject to a multi-objective sorting method (e.g. non-dominated ranking) and crowding distance computing.

3. Evolutionary operations are applied to the latent representations of population samples considering their Pareto ranks and crowding distances. This is a randomized and stochastic technique that simulates the evolutionary process of selecting high-fit candidates and exerts evolutionary operators entailing "crossover" and "mutation" to evolve better molecules [82].

4. The evolved latent representations are decoded by the DGM to generate new molecules, which are then evaluated on the basis of validity, novelty, and uniqueness using RDKit [62]. Invalid and duplicate individuals are eliminated to form new samples for new population construction.

5. The new population of each generation is constructed of the high-quality valid samples from the previous population and newly generated data in Step (4), and can also be used to fine-tune the DGM.

6. Steps (2-5) are repeated for multiple generations as needed to compose the final population.

The specific interaction between the DEL framework and DGM is illustrated in Figure 4.1.

The selection of high-quality samples for fine-tuning the VAE model is based on the non-dominated ranking result, which underlines the molecules with the most beneficial properties in terms of SAS, LogP, and BAS. It is worth noting that we also take into account other properties along with the protein-ligand binding affinity score, namely, molecules whose SAS, LogP, and BAS are smaller are prioritized in ranks and exploited during evolutionary computations. SAS and LogP can be calculated using RDKit; however, BAS is generated by the docking module utilizing QuickVina [5], as explained in section 4.5.

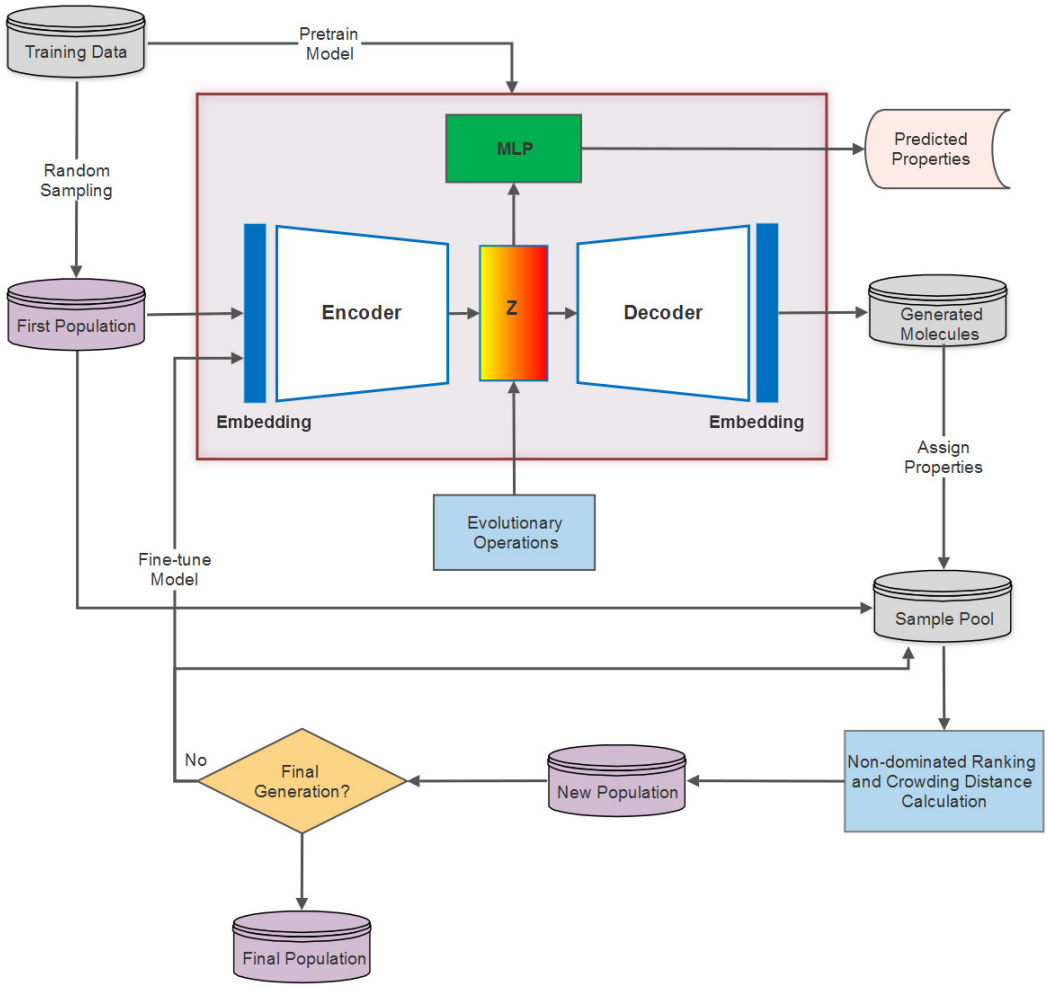


Figure 4.1: Diagram showing the workflow of the DEL framework and its interaction with VAE.

4.2 Multi-objective Optimization

4.2.1 Non-dominated Ranking and Crowding Distance

In preparation for evolutionary operations, DEL analyzes non-dominated rankings and crowding distances of multi-objective solutions for the purpose of identifying advantageous parent solutions in selection for generating offspring, as well as producing distinct populations with competitive and diverse molecules from the molecule pool of previous populations in the subsequent step. Both methods were adapted from the NSGA II algorithm outlined in section 2.4.2. As part of the non-dominated ranking, with k molecular properties as potentially conflicting objectives and multiple objective optimization problem in the latent space: $\min_{\boldsymbol{z}}(f_1(\boldsymbol{z}), f_2(\boldsymbol{z}), f_3(\boldsymbol{z}), ..., f_k(\boldsymbol{z}))$, where

$\boldsymbol{z}$ represents the feasible solutions. $\boldsymbol{z_1} \prec \boldsymbol{z_2}$ denotes $\boldsymbol{z}_1$ dominates $\boldsymbol{z}_2$, and samples from training data or DGMs can be sorted into Pareto fronts $\mathcal{F} = \{\mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, ...\}$. For instance, the first front $\mathcal{F}_1$ dominates $\mathcal{F}_2$ but solutions in $\mathcal{F}_1$ do not dominate each other.

NSGA-II introduced the concept of crowding [19], which is the density value of individuals around a feasible solution within Pareto fronts. It may be intuitively seen as a cuboid surrounding an individual and excluding others. An individual's crowding distance is calculated by summing the differences between the values of the sub-objective functions of two individuals before and after this individual. In general, the crowding distance of individual $\boldsymbol{z}_i$ with K objective is given as:

$$d(\boldsymbol{z}_i) = \sum_{k=1}^{K} \frac{f_k(\boldsymbol{z}_{i+1}) - f_k(\boldsymbol{z}_{i-1})}{f_k^{\max} - f_k^{\min}} \tag{4.1}$$

where $\boldsymbol{z}_{i+1}$ denotes the closest individual above $\boldsymbol{z}_i$ and $\boldsymbol{z}_{i-1}$ is the closest individual below $\boldsymbol{z}_i$. $f_k^{\max}$ and $f_k^{\min}$ are the the maximum and minimum value of the k-th objective, respectively.

Combining non-dominated ranking and crowding distance calculation, the comparison between two feasible solutions can be further defined using partial order $\prec_n$ [40]. Given two solutions, $\boldsymbol{z}_1 \prec_n \boldsymbol{z}_2$ holds if $\boldsymbol{z_1} \prec \boldsymbol{z_2}$, or $\boldsymbol{z_1}$ do not dominate $\boldsymbol{z_2}$ but $d(\boldsymbol{z}_1) > d(\boldsymbol{z}_2)$. This procedure is performed to rank the top M samples for selection with regard to evolutionary operations, or to form the new population for the next generation.

4.2.2 Evolutionary Operators

To produce new samples of the next generation, latent representations of populations undergo evolutionary operations in accordance with classical genetic algorithms, which include individual selection, crossover, and mutation. Individual selection utilizes binary tournament selection to select one individual (parent) from every two random samples by comparing their partial relation, until the drawn mating pool size matches the population size M, namely, if $\boldsymbol{z}_1 \prec_n \boldsymbol{z}_2$ then $\boldsymbol{z}_1$ will be selected as the parent solution. Selected M parents are then paired up as $\boldsymbol{z}_{p1}$ and $\boldsymbol{z}_{p2}$ for crossover operator to generate children $\hat{\boldsymbol{z}}_{c1}$ and $\hat{\boldsymbol{z}}_{c2}$. All experiments use blending

linear crossover (BLX) [121, 43], which formulates

$$\hat{\boldsymbol{z}}_{c1} = \boldsymbol{z}_{p1} + r_1(\boldsymbol{z}_{p2} - \boldsymbol{z}_{p1}), \tag{4.2}$$

$$\hat{\boldsymbol{z}}_{c2} = \boldsymbol{z}_{p1} + r_2(\boldsymbol{z}_{p2} - \boldsymbol{z}_{p1}), \tag{4.3}$$

where

$$r_1 = -d + (1 + 2d)\alpha_1, \tag{4.4}$$

$$r_2 = -d + (1 + 2d)\alpha_2. \tag{4.5}$$

The recommended value of $d = 0.25$ is used in this work and $\alpha_1, \alpha_2 \sim Uniform(0, 1)$. Recombinant samples $\hat{\boldsymbol{z}}_m$ are carried on to the mutation operations over a predefined mutation rate $p_m = 0.01$, meaning for every sample, if a random value l from $Uniform(0, 1)$ is smaller p_m, the l-th position of $\hat{\boldsymbol{z}}_m$ will be overwritten by value drawn from standard Gaussian distribution $\mathcal{N}(0, \mathbf{I})$ [40].

4.3 FragVAE-based DEL

FragVAE has been implemented as a DGM in DEL for drug design that uses the strategy of SMILES fragmentation. As an alternative to atom-and-bond methods, fragment-based drug design (FBDD) [27, 104] has demonstrated constructive results. The FBDD approach appropriates fragmentation of molecular structures as a screening method that breaks molecules into small-weighted components. In general, molecular fragmentation offers three advantages.

- Small organic molecules, corresponding to the fragments, are efficiently synthesizable, hence easier to manipulate chemically.
- Since drug-like molecules may share analogous fragments, fragmentation can help identify components that are possibly responsible for biological activities.
- It can drastically reduce the search space for exploration and characterization.

By incorporating the protein-ligand binding affinity score as another objective, FragVAE-based DEL may assist in discovering highly viable drug candidates. Thus, in this work, FragVAE approach is examined and compared with the graph fragment-based method, JTVAE.

4.3.1 BRICS Fragmentation

BRICS (breaking of retro-synthetically interesting chemical substructures) rules [20] were employed in FragVAE to fragment molecular data, which has proven to outperform the former benchmark RECAP (retrosynthetic combinatorial analysis procedure) [65] in many ways, including the ability to cleave 10% more molecules and produce fragments with more connecting points for enhanced reconstruction flexibility.

BRICS breaks molecules by identifying the 16 chemical environments in which cleavage occurs. It takes into account not only the type of chemical bond but also its surrounding substructures. There are 16 fragment prototypes characterized by a variety of linked (dummy) atoms that correspond to the chemical environments, along with the cleavage site labels from L_1 to L_{16}. To avoid redundant fragmentation, all possible retrosynthetic bonds are cut simultaneously. The cleavage process discards unwanted chemical substructures and leaves small terminal motifs intact [20].

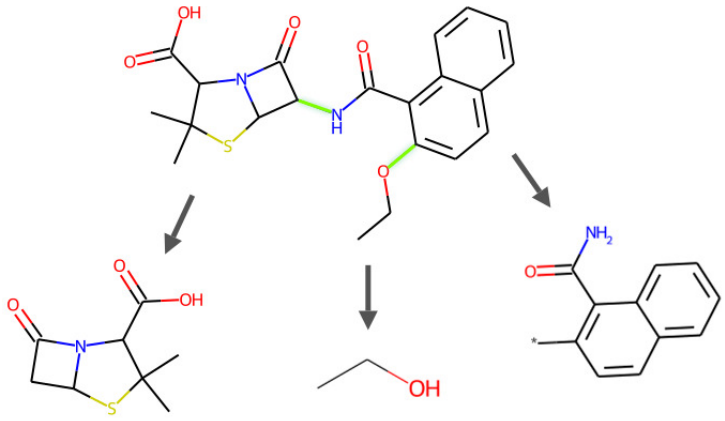


Figure 4.2: Cleavage example of an FDA-approved small-molecule durg Nafcillin (a penicillin derivative antibiotic, DrugBank Access Number DB00607), demonstrating the procedure of BRICS-based fragmentation by producing fragments on breakable bonds. In this example, the SMILES representation is CCOc1ccc2ccccc2c1C(=O)NC1C(=O)N2C1SC(C)(C)C2C(=O)O, and is broken into fragments *0CC, *NC(=O)c1c(*)ccc2ccccc12, and *C1C(=O)N2C1SC(C)(C)C2C(=O)O.

In regards to the fragmentation procedure, atoms in the SMILES string are scanned from left to right, and a fragment is extracted every time a breakable bond

is encountered in accordance with the BRICS rules. This process is repeated until the remaining part cannot be split further. An example of how a SMILES string is fragmented is shown in Figure 4.2. To reconstruct a molecule, fragments can be reassembled starting from the leaves to the root, right to left. In Algorithm 1, a recursive approach is used to perform BRICS-based molecular fragmentation in FragVAE [88].

Algorithm 1 BRICS Fragmentation

Require: list of fragments: F molecule to be fragmented M
  procedure BRICSFragmentation(F, M)
    local variables
      b, chemical bond
      f, potential fragment
      m, remaining molecule after fragmentation
    end local variables
    $b \leftarrow$ GetFirstBRICSBond(M)
    if b is $None$ **then**
      return
    else
      $f, m \leftarrow$ BreakMolAtBond(M, b)
      $M \leftarrow m$
      F.append(f)
      BRICSFragmentation(F, M)
    end if
  end procedure

4.3.2 Modifications

There are two features that impact FragVAE's integration with DEL. First, in contrast with the variational loss of the vanilla VAE in Equation (2.10), a trade-off weight β [44, 127] is added between the KL-divergence and the reconstruction error to balance their discrepancy in minimization. Second, a multi-layer perceptron neural network (MLP) is added to the VAE structure as a property predictor [128]. It intends to regularize the latent representation space by predicting the property values of encoded samples. The variational loss of the VAE model is thus altered by adding a third term for property regression error (see Equation (4.6)).

$$L(\boldsymbol{\phi}, \boldsymbol{\theta}) = -\mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] + \beta \mathrm{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\big) + \alpha \mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\mathrm{SE}\big(f_{\psi}(\boldsymbol{z}), \boldsymbol{y}\big)\big], \tag{4.6}$$

where $f_{\psi}(\boldsymbol{z})$ denotes the predicted property value, $\boldsymbol{\psi}$ is the parameter set of the MLP sub-network, $\boldsymbol{y}$ is the actual property value, and SE stands for squared error.

4.4 JTVAE-based DEL

Despite the proliferation of SMILES-based models in recent years for molecular modelling, it still faces two critical limitations. (1) The SMILES syntax is not robust to small changes or mistakes, which can result in the generation of invalid or drastically divergent structures. (2) The unstructured nature of SMILES notation implies that two structurally similar molecules can have completely different SMILES representations. As a result of these shortcomings, the molecules produced lack diversity and effectiveness. This brings graph-based deep generative models to the fore as an alternative strategy, allowing the search for the topology of molecules and their fragments. It involves a more intuitive way to represent a molecule as a graph built on its Lewis structure. Numerous molecular graph models, such as graph neural network (GNN) [99], graph convolutional network (GCN) [22, 57], message passing neural network (MPNN) [35], and many other methods have been explored and shown outstanding performance in molecular property prediction tasks, and consequently laid the foundation for graph-based molecular generation.

4.4.1 Junction Tree Fragmentation

As one of the most representative VAE-based graph generative models, JTVAE [51] employs a subgraph-by-subgraph manufacturing mode instead of an atom-by-atom mode. This is to avoid revising chemically invalid intermediaries while constructing a molecular graph sequentially atom-by-atom, allowing the model to consistently generate valid molecules since validity is checked at each step following a non-sequential method.

In JTVAE, a cluster vocabulary is first constructed containing simple rings, bonds, and atoms. To begin, the molecular graph G is scanned to pinpoint substructures that appear in the vocabulary and edges that do not belong to any cycles. Two simple rings are merged as bridged compounds if they have three or more overlapping atoms. This step eliminates cycles in molecular structures by considering them as clusters. A graph of clusters is constructed by adding edges between all intersecting clusters and composited into a junction tree by mapping its maximum spanning tree. Tree nodes in the junction tree associate the vertices in the cluster graph, and the connectivity

between nodes corresponds to the chemical bonds between clusters. Through tree decomposition, Algorithm 2 is an implementation of junction tree-based molecular fragmentation as used by JTVAE [51]. Figure 4.3 presents an example of the graph fragmentation procedure using an FDA-approved drug.

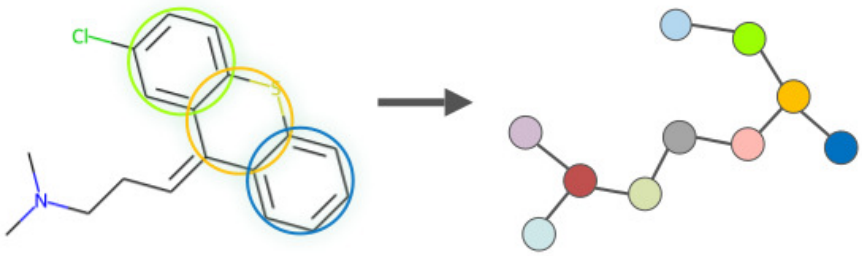


Figure 4.3: Illustration of the graph fragmentation in JTVAE following the subgraph-by-subgraph strategy, representing the process of tree decomposition on an FDA-approved small molecule drug Chlorprothixene (a thioxanthene antipsychotic, DrugBank Access Number DB01239). Clusters in the molecule (left) are identified as substructures and denoted as nodes in the junction tree (right).

4.4.2 Modifications

As with FragVAE for DEL, the same modifications are applied to JTVAE when integrated into the DEL framework. An MLP property predictor is added to the generative model and a SE loss term is appended to the model's total loss. Furthermore, we adopt the two-part latent representation $\boldsymbol{z} = [\boldsymbol{z}_T, \boldsymbol{z}_G]$ introduced in [51] for the evolutionary operations, in which $\boldsymbol{z}_T$ maps the junction tree structure of a molecule at the cluster level and $\boldsymbol{z}_G$ encodes the association within each cluster. The tree-structured latent variable captures the hierarchical relationships between nodes in a graph, while the graph-structured latent variable represents the local graph structures and node attributes, resulting in a more comprehensive and accurate representation of the data.

Considering the tree decomposition of a molecule, the architecture of the modified JTVAE consists of the following five components: (a) a graph encoder $q(\boldsymbol{z}_G|G)$ to encode molecular graph G to its latent representation $\boldsymbol{z}_G$, (b) a tree encoder $q(\boldsymbol{z}_T|T)$ to encode the junction tree T decomposed from molecular graph to acquire the latent

Algorithm 2 Junction Tree Fragmentation

Require: molecule graph $G = (V, E)$
  procedure TREEDECOMPOSITION(G)
    local variables
      L_1, list of simple rings $(v_s, ..., v_e)$
      L_2, list bonds (v_s, v_e) that do not belong to any rings
      V_0, list of vertices that share bonds with three or more clusters
      $G' = (V', E')$, cluster graph after fragmentation
    end local variables
    if LENGTH$(V) \leq 1$ **then**
      return empty graph
    end if
    $L_1 \leftarrow$ CHEM.GETSYMMSSSR(G) ▷ Get the simple rings in G
    for each bond $(v_s, v_e) \in E$ **do**
      L_1.APPEND$((v_s, v_e))$ if not (v_s, v_e).ISINRING
    end for
    for each ring $(v_s, ..., v_e) \in L_1$ **do**
      Merge rings that have three or more overlapping atoms
    end for
    $V_0 \leftarrow$ GETINSTERSECTIONATOM$(L_1 \cup L_2)$
    $V' \leftarrow L_1 \cup L_2 \cup V_0$
    $E' \leftarrow (v_s, v_e) \in V' \times V'$ if $|v_s \cap v_e| > 0$ **and** $(v_s \in V_0$ **or** $v_e \in V_0)$
    return GETMAXIMUMSPANNINGTREE$(G = (V', E'))$
  end procedure

vector $\boldsymbol{z}_T$, (c) a property predictor for the latent space regularization, (d) a tree decoder $p(T|\boldsymbol{z}_T)$ to decode the junction tree from $\boldsymbol{z}_T$, and (e) a graph decoder $p(G|\boldsymbol{z}_G)$ to eventually manifest the molecular graph. The tree message passing neural network (TMPNN) [35] and the graph message passing neural network (GMPNN) [15] with GRU units are used in the transformation of representations. The reconstruction error in JTVAE becomes

$$R = L_c(T) + L_g(G) + L_s, \tag{4.7}$$

where $L_c(T)$, given junction tree T, is the entropy loss of the tree decoder, that is the summed error of topological prediction (binary prediction of the existence probability of child node) and label prediction (label of generated child node); $L_g(G)$ and L_s are, respectively, the negative expectations of log-likelihood of subgraph prediction based on tree nodes, and stereoisomer prediction by comparing the cosine similarity of its graph representation. In view of these factors, the overall loss function of the modified JTVAE can be defined as:

$$\begin{aligned} L(\boldsymbol{\phi}, \boldsymbol{\theta}) &= -\mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[R + \beta \mathrm{KL}\big(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})\big) + \alpha \mathrm{E}_{\boldsymbol{z}\sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\mathrm{SE}(f_{\boldsymbol{\psi}}(\boldsymbol{z}, \boldsymbol{y})\big], \\ \boldsymbol{z} &= [\boldsymbol{z}_T, \boldsymbol{z}_G], \\ q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) &= [q_{\boldsymbol{\phi}}(\boldsymbol{z}_T|T), q_{\boldsymbol{\phi}}(\boldsymbol{z}_G|G)]. \end{aligned} \tag{4.8}$$

4.5 Protein-Ligand Binding Affinity Score (BAS) Calculation

Implementing a reliable and suitable docking score calculation module is essential to integrate protein-ligand BAS within DEL for molecular optimization. We adopted the efficient docking tool QuickVina (QVina) [5] which is an amended and accelerated version of the well-known AutoDock Vina tool [113, 23]. AutoDock Vina is an open-source molecular docking engine in the AutoDock suite, and shows dominant performance in virtual screening by adopting the benchmark data Directory of Useful Decoys (DUD). AutoDock Vina uses a scoring function to calculate the binding affinity of a protein-ligand complex. The scoring function is based on the concept of electrostatic complementarity, which aims to predict the stability of the complex by assessing the attractive and repulsive interactions between the protein and ligand. Based on a global search for the best possible ligand conformation, the program selects the conformation with the lowest energy as the most stable and the one with the greatest affinity for binding. While AutoDock Vina is accurate, it is time-consuming

due to the exhaustiveness of the search on the 3D target surface. QVina on the other hand, has an improved search algorithm that performs faster and yet still accurate BAS calculations. The scoring function of QVina is similar to that of AutoDock Vina providing an equivalent scale of prediction of binding affinity, but enhanced with a global search history. It is possible for adjacent search threads to communicate and share docking information, which greatly accelerates the search. Therefore, QVina is the more appropriate choice for our research.

Applying QVina, our system was first experimented on the single protein target CA9 with three objectives $\{SAS, LogP, CA9\}$ in the evolutionary process. Following that, we also investigated the possibility for optimization on double protein targets (CA9 and GPX4 protein) counting four objective functions $\{SAS, LogP, CA9, GPX4\}$.

4.5.1 Docking on CA9 Protein Target

The carbonic anhydrase IX protein (CA9/CAIX) [107, 30] has been identified as a significant marker of tumour hypoxia and may be used as a diagnostic biomarker, prognostic indicator, and therapeutic target for cancer. Solid tumours rely on the blood supply to deliver oxygen and nutrients. As the tumour grows, these blood vessels are unable to supply oxygen and nutrients to every part of the tumour, leading to regional hypoxia. Over time, this hypoxic environment can result in an accumulation of acid within the tumour cells. Cancer cells respond to this stress by releasing enzymes that neutralize the acidic conditions in their environment, allowing them to survive and eventually spread to other tissues. CA9, one of the major neutralizing enzymes, plays a crucial role in the survival, invasion, and metastasis of cancer cells. Due to the fact that CA9 possesses extracellular active docking sites, ligands with desired affinity for docking to these sites may have the potential to become cancer drugs that act by inhibiting the growth of cancer cells.

The binding site coordinate and constrained search box size in the docking score calculation module are user-specified. For our experiments, generated molecules from both methods are docked to a cubic box of size 20 centred at coordinate $[7.750, -14.556, 6.747]$ in the binding pocket. The current calculation of BAS using QVina is based on molecular mechanism simulations, where the ligand is examined on the protein surface in 3D space. This requires the configuration of target site coordinates as well as the search grid size in order to constrain simulation time. When sufficient experimental data for CA9 become available in the future, machine learning-based approaches will be investigated for protein-ligand binding affinity prediction.

4.5.2 Docking on GPX4 Protein Target

GPX4 [28, 45], also known as phospholipid hydroperoxide glutathione peroxidase, is the fourth member of the selenium-containing GPX family. A variety of signaling pathways can cause cells, specifically with iron-dependent cell death (ferroptosis) in cancer cells. Studies have demonstrated that direct targeting of GPX4 induces ferroptosis in cancer cells, making GPX4 inhibitors effective anticancer agents. Several GPX4 inhibitors have been discovered, such as RSL3, ML162, DPI compounds, FIN56 and FINO2. Most of the available GPX4 inhibitors have, however, been limited in their clinical application due to their inferior pharmacokinetic properties. Whether targeting CA9 can generate drug-like molecules that inhibit cancer development and invasion is unclear, but when used together with GPX4, collected samples with optimal BAS on both CA9 and GPX4 targets may have promising clinical potential.

Proceeding with the BAS calculation module, the center coordinate for GPX4 docking is set to $[-9.67, 7.043, 4.609]$ with grid box size 20. As GPX4 does not have the same depth of pockets as most docking sites, it is an extremely difficult target to dock with. The possibility exists in forming covalent bonds with small molecules or peptidic drugs targeting another active site containing Selenium. Since QVina does not support Selenium-specific parameters, we chose to investigate the docking site $[-9.67, 7.043, 4.609]$.

4.6 1-Wasserstein Distance

In the space $\mathbb{R}$, there are many ways to describe the distance between two probability distributions $q(\boldsymbol{x})$ and $p(\boldsymbol{x})$. One of the more popular ones is Kullback-Leibler (KL) divergence as motioned in 2.11:

$$\mathrm{KL}(p \parallel q) = \int_{\mathbb{R}^n} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \mathrm{d}\boldsymbol{x}. \tag{4.9}$$

As defined above, KL does not measure the geometric properties of $\mathbb{R}^n$, because the comparison between $q(\boldsymbol{x})$ and $p(\boldsymbol{x})$ is made at the same points. This motivates us to utilize Wasserstein distance (WD) as a distance metric for a pair of distributions. For the probability distributions μ and υ defined on $\mathbb{R}$, the p-th is given as:

$$W_p(\mu, \upsilon) := \inf_{\gamma \in \Gamma(\mu, \upsilon)} \Big(\int_{\mathbb{R} \times \mathbb{R}} d(x, y)^p \mathrm{d}\gamma(x, y) \Big)^{\frac{1}{p}}, \tag{4.10}$$

where $\Gamma(\mu, \upsilon)$ is the collection of joint probability measures γ on $\mathbb{R} \times \mathbb{R}$ with marginal distributions μ and υ. A measure with marginals μ and υ is also called the coupling of μ and υ. d can be any distance on $\mathbb{R}$, such as Euclidean distance, l_1 distance, etc. In the case of $p = 1$ and $d(x, y) = |x - y|$, the one-dimensional Wasserstein distance (1-Wasserstein distance) is explicitly formulates as:

$$W_1(\mu, \upsilon) = \int_{\mathbb{R}} |F_\mu(x) - F_\upsilon(x)| \mathrm{d}x, \tag{4.11}$$

where F is a cumulative distribution function.

4.7 Hypervolume Measure

The condition of algorithm convergence is an extremely critical aspect of multi-objective evolutionary algorithms. DEL retains the elite solution set of the previous generation and adds it to the evolutionary process of the new generation. The solution set of the evolutionary population continues to converge to the real Pareto frontier, and reaches a satisfactory optimization solution. Usually, when analyzing the performance of a multi-objective optimization algorithm, we hope that the algorithm can advance in three aspects. (1) The distance between the real Pareto front surface and the one obtained by the algorithm should be as small as possible. (2) Although the obtained individual solution points are only partial solutions, they should be distributed on the Pareto front as uniformly as possible. And, (3) a sufficient number of solution points should be able to cover the entire front, that is each region on the front should be represented by solution points unless this region is missing on the actual optimal Pareto front.

The hypervolume (HV) index measures the volume of the dimensional region in the target space bounded by the non-dominated solution set obtained by a multi-objective optimization algorithm and a pre-specified reference point. The mathematical representation of the HV calculation is given in Equation (4.12):

$$\mathrm{HV} = \delta(\cup_{i=1}^{|S|} v_i), \tag{4.12}$$

where δ represents the Lebesgue measure, which is used to compute volume, $|S|$ represents the number of non-dominated solutions, and v_i represents the hypercube formed by the reference point and the i-th solution in the solution set. HV is an effective quantitative scalar metric, which is strictly monotonic in terms of Pareto

dominance. The larger the value of HV, the the better the set of solutions covers the objective space and trade-offs between objectives. Especially, the calculation of the HV index does not require the ideal Pareto front of the test problem, which greatly facilitates the use of HV in practical applications.

Chapter 5

Experiments

5.1 Data

The experiments were conducted on the ZINC dataset [49] and a variant of ZINC by appending authentic drug molecules from the DrugBank database [126] (named ZINC+DrugBank hereafter). The ZINC dataset is a popular benchmark set for generative tasks that comprises approximately 250K molecules in SMILES notation. DrugBank is a web-based database hosting detailed information on medicines including identification, pharmacology, interactions, properties, and clinical trials. We formed a subset by extracting 1932 small-molecule drugs from DrugBank and fusing them into the original ZINC dataset. Molecular samples drawn from the original ZINC and DrugBank data are visualized in Figure 5.1 and 5.2.

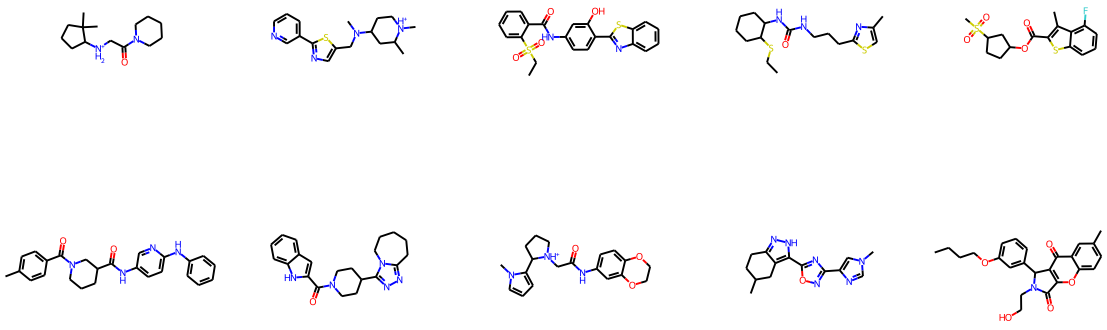


Figure 5.1: 2D graph visualization of randomly chosen molecules from ZINC datasets. RDKit was used for visualization.

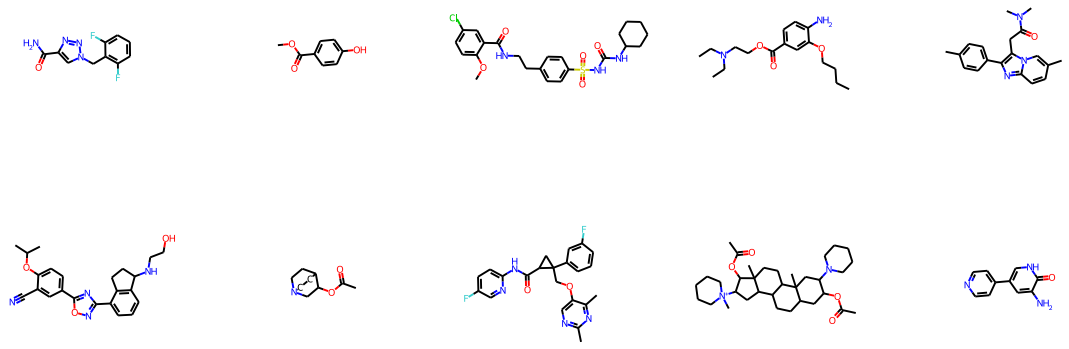


Figure 5.2: 2D graph visualization of randomly chosen molecules from DrugBank dataset. RDKit was used for visualization.

The ZINC and DrugBank datasets were both subjected to a three-fold preprocessing step.

1. For FragVAE, molecules are cleaved into SMILES fragments following the BRICS algorithm, whereas in JTVAE, subgraph enumeration and tree decomposition are performed.

2. Calculation of the molecular properties (including objectives SAS, LogP, and BAS) using RDKit and the protein-ligand binding score calculation module.

3. Removal of duplicated molecules as well as molecules with fewer than 2 fragments for FragVAE.

5.2 Hyperparameter Settings

During the experiments, we used the same evolutionary learning hyperparameters for evaluating the DEL framework. However, for FragVAE and JTVAE, we tuned the hyperparameters differently to optimize their performance. Table 5.1 lists our key hyperparameter settings.

Hyperparameter	DEL	Hyperparameter	DGM	
			FragVAE	JTVAE
Number of generations	10	Embedding size	128	128
Polulation size	20000	Recurrent layers	2	1
Initial epochs	20	Hidden state dimension	128	450
Subsequent epochs	10	Latent space dimension	64	64
Scheduler annealing rate	0.8	Learning rate	0.0001	0.0001
Tournament selection probability	0.95	Batch size	128	32
Mutation rate	0.01	KL divergence weight β	0.1	0.1

Table 5.1: Hyperparameter settings of the DEL process and DGMs respectively.

5.3 Implementation Requirements

The successful execution of this work depends on the proper installation and configuration of various software and hardware components. This section outlines the key requirements, including necessary software libraries and tools, hardware specifications, and dependencies. It is crucial to adhere to the guidelines for a stable and consistent project environment. To specify, this work uses Python 3.7 as the programming language.

- Boost 1.74.0
- Gensim 3.4.0
- PyTorch 1.7.1
- GpyTorch 1.5.1
- Matplotlib 3.2.2
- Joblib 1.1.0
- QVina 2.1.0

- Scikit-learn 1.0.1
- Scipy 1.6.2
- Openbabel 3.1.1
- Numpy 1.19.2
- Pandas 1.3.4
- RDkit 2021.09.2

5.4 FragVAE versus JTVAE in DEL

5.4.1 Single Target

While performing the optimization on objectives $\{SAS, LogP, CA9\}$, the two base DGMs in DEL and two datasets lead to a set of four experiments: (1) FragVAE in DEL framework trained on the original ZINC data, (2) FragVAE in DEL on ZINC+DrugBank data, (3) JTVAE in DEL on ZINC data, and (4) JTVAE in DEL on ZINC+DrugBank data. Whenever DEL was trained on the ZINC+DrugBank data, the actual drug molecules from DrugBank were added to the initial population composing the training data of the subsequent generation.

Due to JTVAE's graph-based nature, training and fine-tuning it for multiple generations is a lengthy process. In addition, docking simulation of the populations is time-consuming, so it would take roughly fourteen days to run the JTVAE+DEL program on the ZINC dataset. Hence, we conducted one JTVAE+DEL experiment (set (3) and (4) above), one FragVAE+DEL trained on the original ZINC data, and five FragVAE+DEL trained on the ZINC+DrugBank dataset for the purpose of gathering more novel samples for use in future drug discovery pipelines based on this particular dataset.

Firstly, the performances of FragVAE-based and JTVAE-based DEL were measured using three metrics:

- Validity: the ratio of chemically valid generated molecules in the population.
- Novelty: the ratio of validly unique generated samples that are not originated from the training dataset.

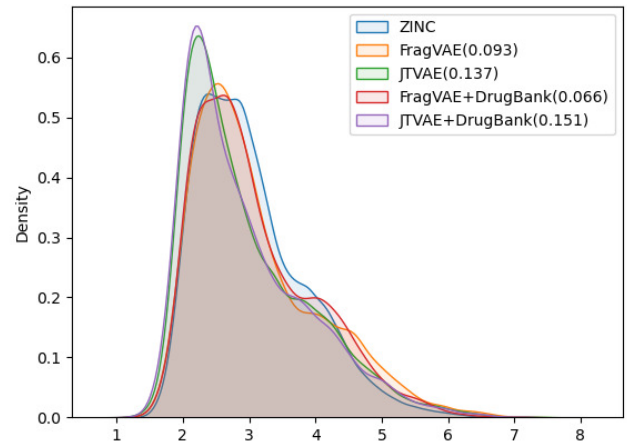
- Uniqueness: the ratio of generated molecules that are not duplicated in the population.

All of these metrics were scored based on SMILES strings and chemical validity checking was performed by RDKit. A comparison of DEL based on these two DGMs is shown in Table 5.2, which conveys that JTVAE produces all valid molecules, and the vast majority of the samples in the last populations generated in reliance on FragVAE and JTVAE, respectively, are novel. Furthermore, both methods are capable of maintaining a highly diverse population of DEL.

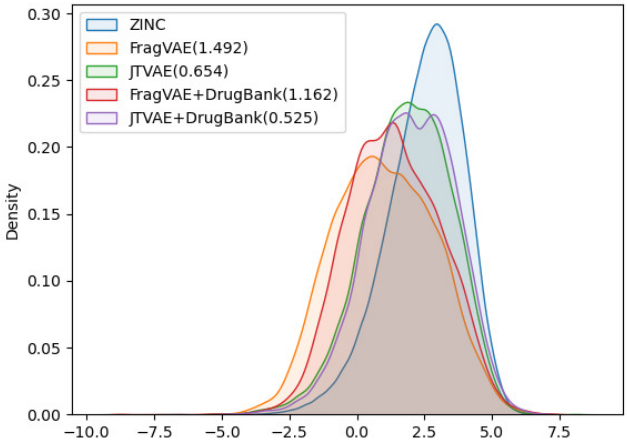
Model	Dataset	Validity	Novelty	Uniqueness
FragVAE	ZINC	0.973	0.999	0.968
JTVAE	ZINC	1	0.987	0.933
FragVAE	ZINC+DrugBank	0.994	0.999	0.960
JTVAE	ZINC+DrugBank	1	0.988	0.940

Table 5.2: Performance metrics of the final (10th) population from DEL using FragVAE and JTVAE, respectively, on two datasets.

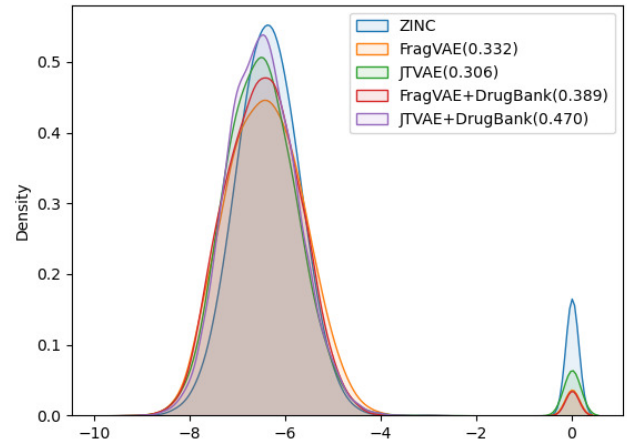
Secondly, the methods were evaluated according to the property-wise distributions of samples in their last populations (See Figure 5.3). Given the goal of expressing an intuitive and quantitative comparison, the 1-Wasserstein distances (WD) from the final population of FragVAE-based and JTVAE-based DEL, respectively, to the original ZINC data were calculated and indicated in the legends. It shows that both methods succeed in improving the properties through generations. While there is a slight difference between the two methods on BAS, JTVAE exhibits superior performance on SAS whereas FragVAE improves LogP the most.



A



B



C

Figure 5.3: Property distributions on SAS **(A)**, LogP **(B)**, and CA9 **(C)** with 1-Wasserstein distances between the final population (10th) of DEL and the original ZINC data.

Thirdly, while the property-wise distributions presented above can only offer a partial comparison, the Pareto-fronts obtained by DEL using different base DGMs are compared in terms of HV (calculated using 4.12) to reflect the overall quality of solutions. As reference points, we selected $[-7.2893, -8.2521, 0]$ (SAS, LogP, CA9) for the ZINC dataset and $[-7.2893, -12.6058, 0]$ for the ZINC+DrugBank dataset. Both points represent the property values of the worst possible samples in each training dataset. The number of generations was set to 10 and the trade-off hyperparameter β was set to 0.1. The obtained HVs in the initial generation, the middle generation, and the final generation are listed in Table 5.3 which demonstrates the following key features. (1) HV increases along with evolutionary generations, showing that DEL can gradually improve the quality of Pareto fronts, and, (2) the JTVAE in the DEL framework results in a significantly higher HV value compared to FragVAE, revealing a better comprehensive performance of the graph fragmentation algorithm in DEL.

Model	Dataset	Hypervolume		
		Generation 1	Generation 5	Generation 10
FragVAE	ZINC	430.93	432.98	433.13
JTVAE	ZINC	496.59	498.18	493.31
FragVAE	ZINC+DrugBank	655.03	656.79	658.30
JTVAE	ZINC+DrugBank	655.50	661.02	671.62

Table 5.3: Hypervolumes of DEL's Pareto fronts ($\beta = 0.1$) targeting CA9 protein with three objectives $\{SAS, LogP, CA9\}$ in the evolutionary process. The results are collected from the populations of Generations 1, 5, and 10, respectively.

5.4.2 Double Targets

The evaluation scheme was extended to DEL targeting both CA9 and GPX4 proteins with four objectives: $\{SAS, LogP, CA9, GPX4\}$. Given the obvious advantages of training on ZINC+DrugBank dataset, and docking simulation of two protein targets over multiple generations being extremely time-consuming, this section focuses on: (1) FragVAE in double-target DEL trained on ZINC+DrugBank data, and (2) JTVAE in double-target DEL trained on ZINC+DrugBank data. Over the course of the experiments, we completed one run of JTVAE+DEL and Frag+VAE on two targets, as it could take approximately one month to fully run JTVAE+DEL and twelve days for Frag+VAE.

Implementing the performance metrics scheme that comprise validity, novelty and uniqueness, Table 5.4 displays the comparison between FragVAE and JTVAE in the DEL framework. In line with the analysis in the single target section, both models presented satisfactory performance in population novelty, where the samples demonstrated distinct structures from those in the training molecules. JTVAE maintained a validity score of 100% without exception, which aligns with results from the single-target approach as evidenced by the advantages of valid reconstruction.

Model	Dataset	Validity	Novelty	Uniqueness
FragVAE	ZINC+DrugBank	0.995	0.999	0.953
JTVAE	ZINC+DrugBank	1	0.988	0.940

Table 5.4: Performance metrics of the final (10th) population from DEL using FragVAE and JTVAE with double protein targets, respectively, on two datasets.

Property-wise distributions of objectives $\{SAS, LogP, CA9, GPX4\}$ across the final populations are illustrated in Figure 5.4. The curves reveal that both models are capable of capturing the dataset sample space and generating samples that fit the distribution of the training data. In conjunction with the WD measuring the quantitative differences between the population distribution and the original ZINC data, we can see that both JTVAE+DEL and FragVAE+DEL have successfully optimized toward the multi-objective goal. With WD results of 0.529 against 0.464 and 0.496 against 0.385, molecular structures produced by JTVAE have favorable binding affinity for both CA9 and GPX4 proteins, whereas FragVAE samples perform better in LogP.

Finally, we calculated the hypervolume of the collected final populations using reference point of $[-7.2893, -12.6058, 0, 0]$ (SAS, LogP, CA9, GPX4) to evaluate the solutions to the target multi-objective problem in a scalar manner, and our goal to obtain higher hypervolume scores. In Table 5.5, if compared horizontally, we can see that DEL framework is able to consistently evolve towards better solutions even considering more objectives. By factoring in the information pertaining to the topology of the molecules, graph based model JTVAE achieved a higher overall hypervolume value in DEL throughout the generations as opposed to SMILES-based models.

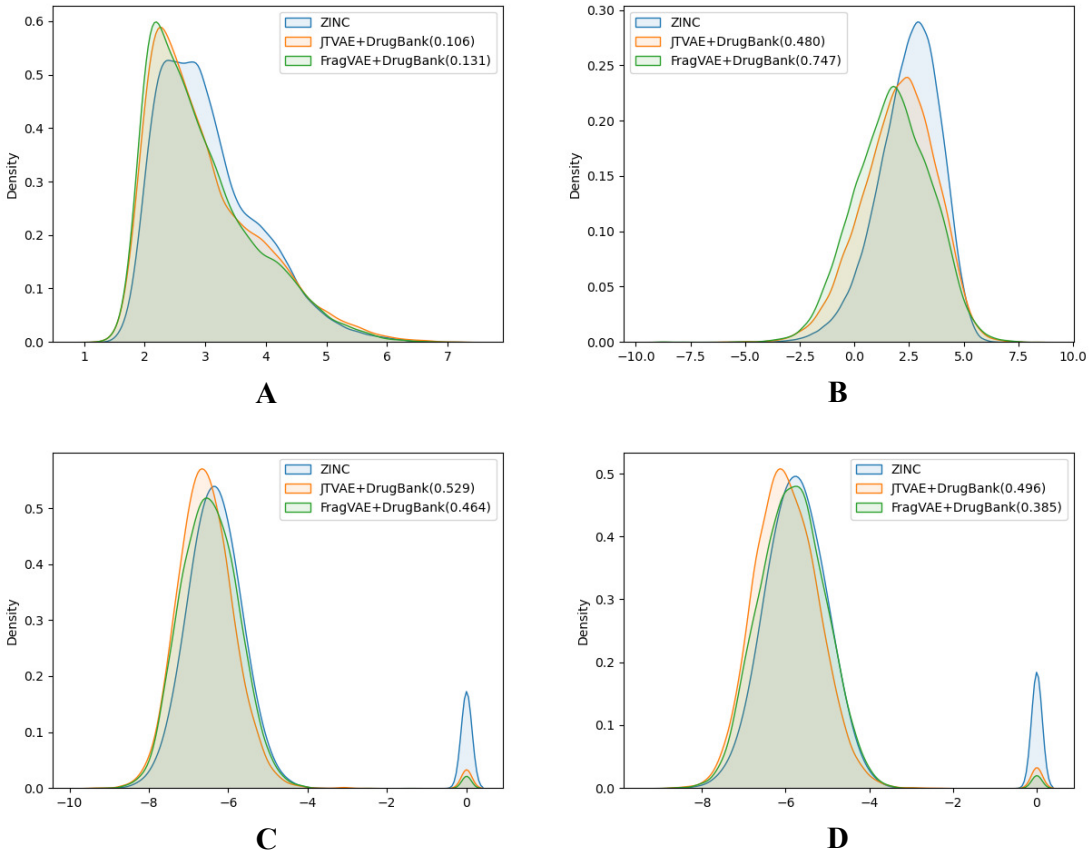


Figure 5.4: Property distributions on SAS **(A)**, LogP **(B)**, CA9 **(C)** and GPX4 **(D)** with 1-Wasserstein distances between the final population (10th) of DEL and the original ZINC data.

Model	Dataset	Hypervolume		
		Generation 1	Generation 5	Generation 10
FragVAE	ZINC+DrugBank	5294.69	5294.18	5305.49
JTVAE	ZINC+DrugBank	5351.80	5351.86	5439.90

Table 5.5: Hypervolumes of DEL's Pareto fronts ($\beta = 0.1$) involving CA9 and GPX4 protein targets, optimizing on four objectives (SAS, LogP, CA9 and GPX4). The results are collected from the populations of Generations 1, 5, and 10, respectively.

5.5 Virtual Screening

Generated samples on the first Pareto front of the final population are viewed as high-rank molecules. As we set the population size M to 20,000 in our experiments, DEL has typically resulted in up to 30 Pareto ranks. This section examines only the first fronts.

5.5.1 Single Protein Target

For DEL with single CA9 protein target, the following threefold criteria were applied to identify high-quality novel samples in the the first front:

- SAS ≤ 3
- $-0.4 \leq$ LogP ≤ 5.6
- CA9 ≤ -5.9

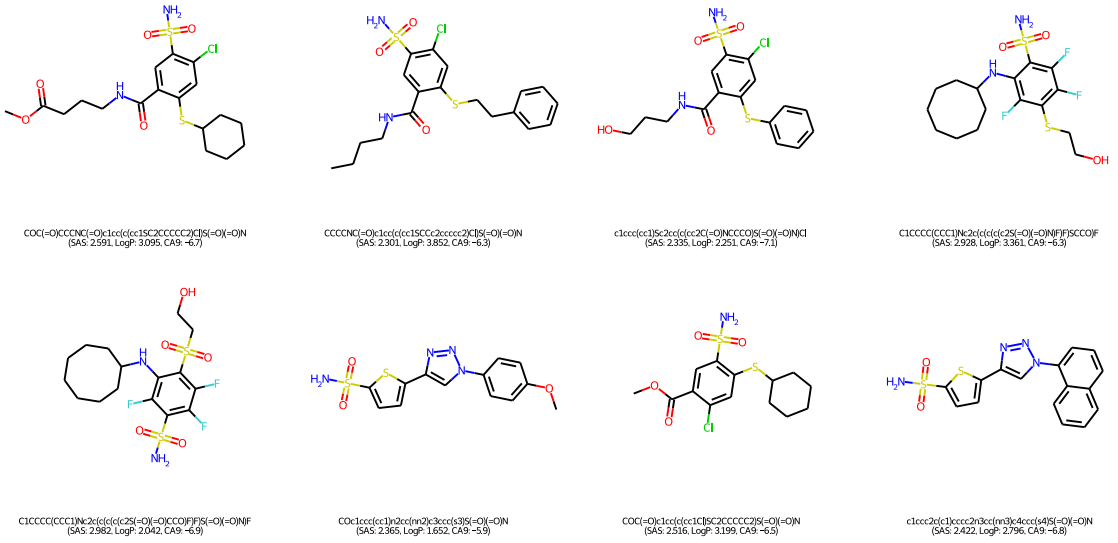


Figure 5.5: 2D graph visualization of 8 unique CA9 ligands from PDB.

Regarding the qualitative characterization of LogP, the Ghose filter rules [34] were considered to benefit the prediction of drug-likeness. We then identified the BAS threshold by assessing 8 unique real ligands of the target protein CA9. These ligands were collected from Protein Data Bank (PDB) [10] and processed using the scoring module as our framework to isolate the impact of docking configurations. In Figure 5.5, we observe a maximum BAS of -5.9 and a minimum of -7.1, thus

setting the upper bound for filtering the high-quality novel molecules from the first fronts as one of our objectives is to minimize BAS. Respectively, 78 molecules and 109 molecules (after merging five runs, average 72 molecules per run) were retrieved from FragVAE+DEL trained on the ZINC and ZINC+DrugBank data; 81 and 92 molecules from JTVAE+DEL were trained on the ZINC and ZINC+DrugBank data. Figure 5.6 and 5.7 illustrate the results obtained from the ZINC+DrugBank dataset when sorted by CA9 binding affinity scores.

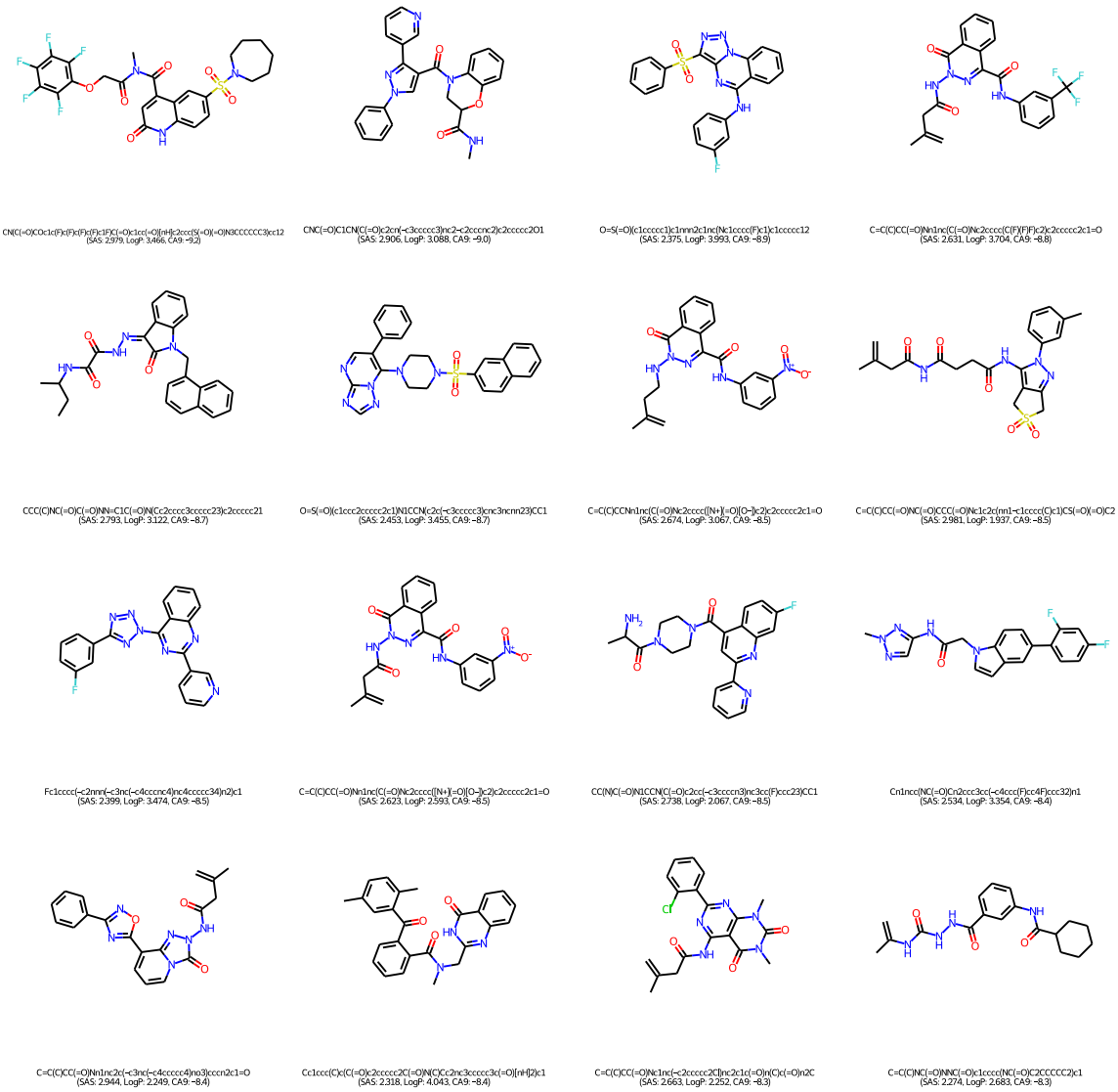


Figure 5.6: 2D graph visualization of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9\}$ applying **FragVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with advantageous BAS, the samples are sorted descendingly by the CA9 binding score. Due to space limitations, only the top 16 molecules are shown. A.1 displays the complete list of samples.

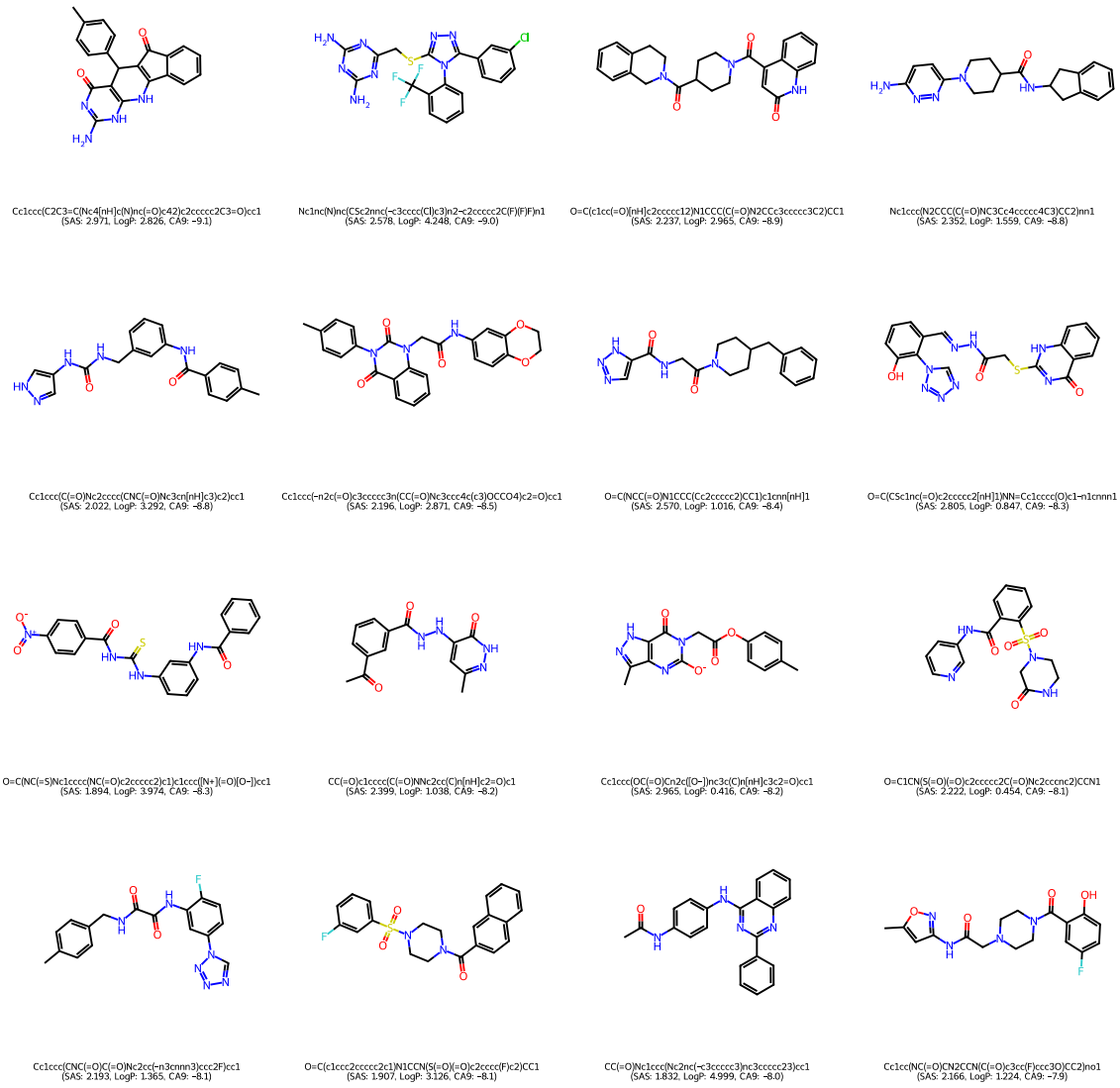
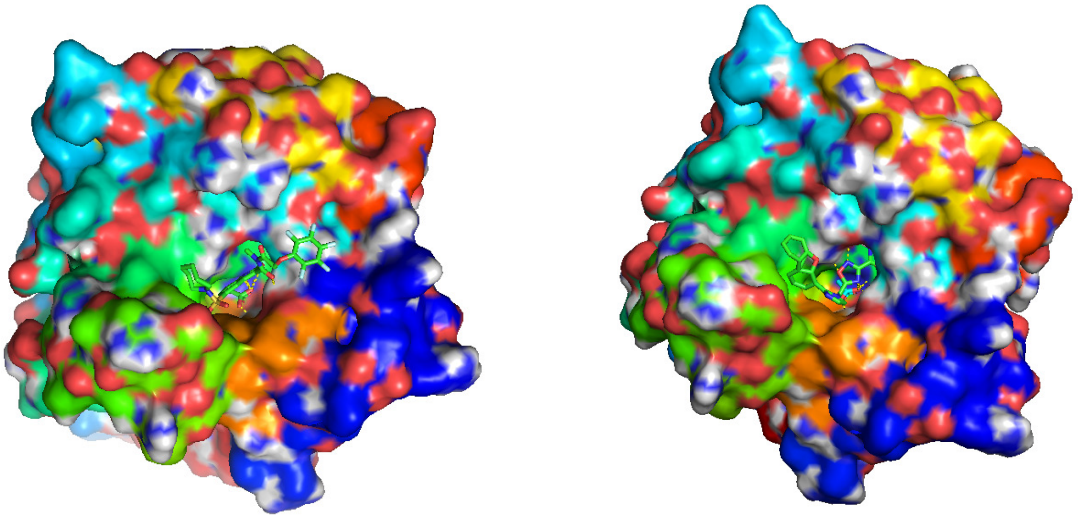


Figure 5.7: 2D graph examples of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9\}$ in combination with **JTVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with advantageous BAS, the samples are sorted descendingly by the CA9 binding score. Due to space limitations, only the top 16 molecules are shown. The complete list of samples can be found in A.2.

Different from the traditional procedure in virtual screening which investigates the binding affinity scores of given molecules in a fixed library, our approach integrates virtual screening in the generation and optimization process through the use of binding affinity score along with other concerned objectives. The benefit of our approach is that it can find novel molecules which potentially satisfy all applied criteria. As a case study to demonstrate, we ranked the high-quality molecules descendingly

by BAS and selected four novel molecules from the top, two were obtained using FragVAE+DEL and two from JTVAE+DEL. Figure 5.8 and 5.9 display, thanks to PyMOL [101], the corresponding protein-ligand complexes. As compared to the existing ligands in Figure 5.5, both molecules have no violation of the criteria and excel in all three objectives. For further validations, all novel molecules from the DEL result in preferred ranges of properties and binding scores can be promoted.



A **B**

Figure 5.8: Docking visualization of two novel molecules binding on CA9 protein surface. Both molecules ranked top on BAS in the high-quality samples of their final population. **(A)** shows the molecule CN(C(=O)COc1c(F)c(F)c(F)c(F)c1F)C(=O)c1cc(=O)[nH]c2ccc(S(=O)(=O)N3C-CCCCC3)cc12 binding to the binding site of CA9 protein. It was generated by FragVAE+DEL trained on ZINC+DrugBank dataset and has a binding affinity score of -9.2 (SAS: 2.979, LogP: 3.466). **(B)** shows the molecule Cc1cccc(C(=O)Nc2cccc3nonc23)c1Nc1cccc2ccccc12 binding to the binding site of CA9. It was discovered by FragVAE+DEL on ZINC data and has a binding affinity score of -9.0 (SAS: 2.668, LogP: 2.979).

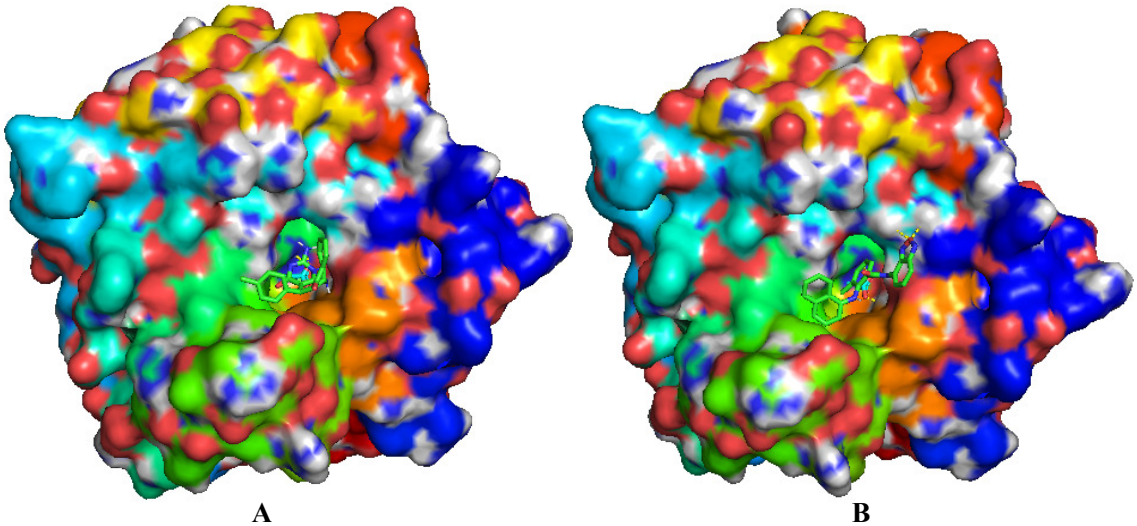


Figure 5.9: Docking visualization of two novel molecules binding on CA9 protein surface. Both molecules ranked top on BAS in the high-quality samples of their final population. **(A)** shows the molecule Cc1ccc(C2C3=C(Nc4[nH]c(N)nc(=O)c42)c2ccccc2C3=O)cc1 binding to the binding site of CA9 protein. It was generated by JTVAE+DEL trained on ZINC+DrugBank dataset and has a binding affinity score of -9.1 (SAS: 2.971, LogP: 2.826). **(B)** shows the molecule O=C1Nc2cc(C(=O)NCCc3nnc(-c4ccccc4)o3)ccc2C1=O binding to the binding site of CA9. It was discovered by JTVAE+DEL on ZINC data and has a binding affinity score of -9.0 (SAS: 2.245, LogP: 5.580).

5.5.2 Double Protein Targets

We established the following screening criteria to the four corresponding objectives in DEL for both protein targets:

- SAS ≤ 3
- $-0.4 \leq$ LogP ≤ 5.6
- CA9 ≤ -5.9
- GPX4 ≤ -6.3

In addition to the threshold for SAS, LogP, and CA9 in Section 5.4.1, we determined the upper bound of the GPX4 binding affinity score using the same methodology as for CA9. By screening the GPX4 ligands available in PDB and analyzing their docking ability under our experiment setting, we pinpointed six real ligands with a considerable degree of complexity in their structure and docking affinity score (Figure 5.10). It shows that the BAS of the real ligands ranges between -3.4 and -6.3. Unlike

CA9, GPX4 is a challenging target to dock, and potential drug molecules are prone to benefit from a relatively high BAS. To that end, we strictly adhered to the minimum BAS -6.3 threshold for the GPX4 score. After applying the screening criteria to the first fronts of FragVAE+DEL and JTVAE+DEL trained on ZINC+DrugBank data, we obtained 240 and 196 high-quality novel molecules, respectively. Graph structures of the high-quality samples are visualized in Figure 5.11 and 5.12. We ranked the molecules descendingly via the sum of ranks method to present an intuitive comparison of their docking affinities towards both targets. Binding affinity scores of CA9 and GPX4 are ranked individually, yielding the final rank of each molecule by summing its CA9 and GPX4 ranks.

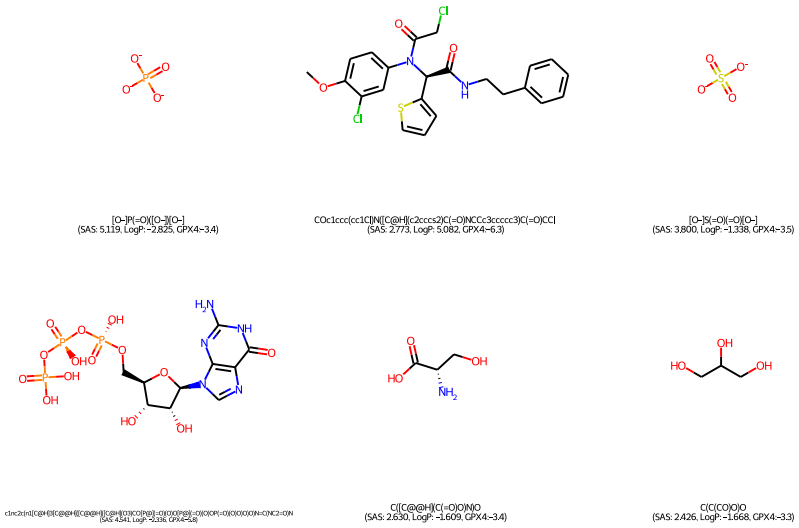


Figure 5.10: 2D graph visualization of 6 unique GPX4 ligands from PDB.

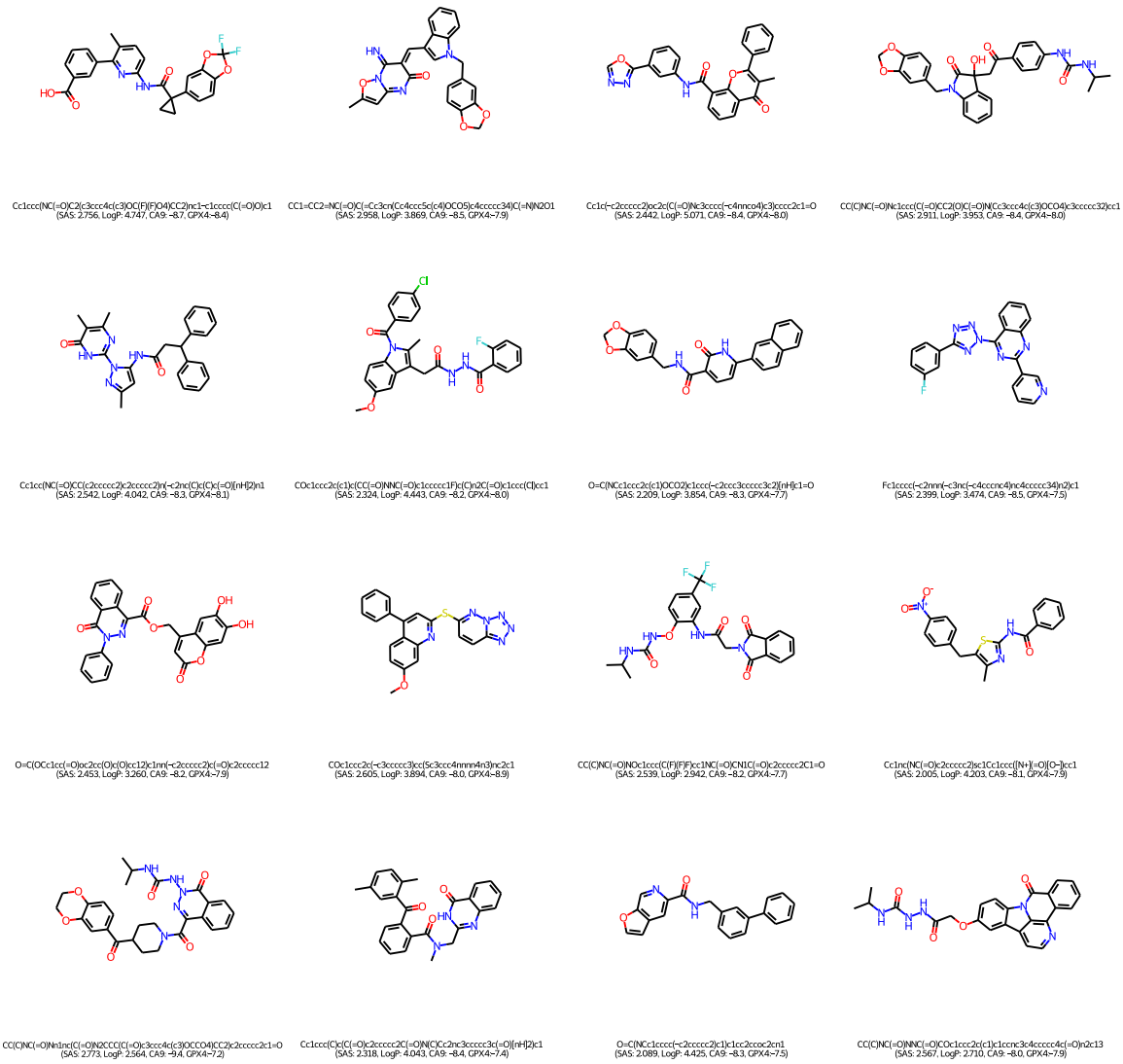


Figure 5.11: 2D graph visualization of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9, GPX4\}$ in combination with **FragVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with favorable BAS, the results are ranked based on CA9 and GPX4 binding scores respectively, then sorted by the summation of both ranks. Only the top 16 molecules are shown due to space limitations. A.3 displays the complete list of samples.

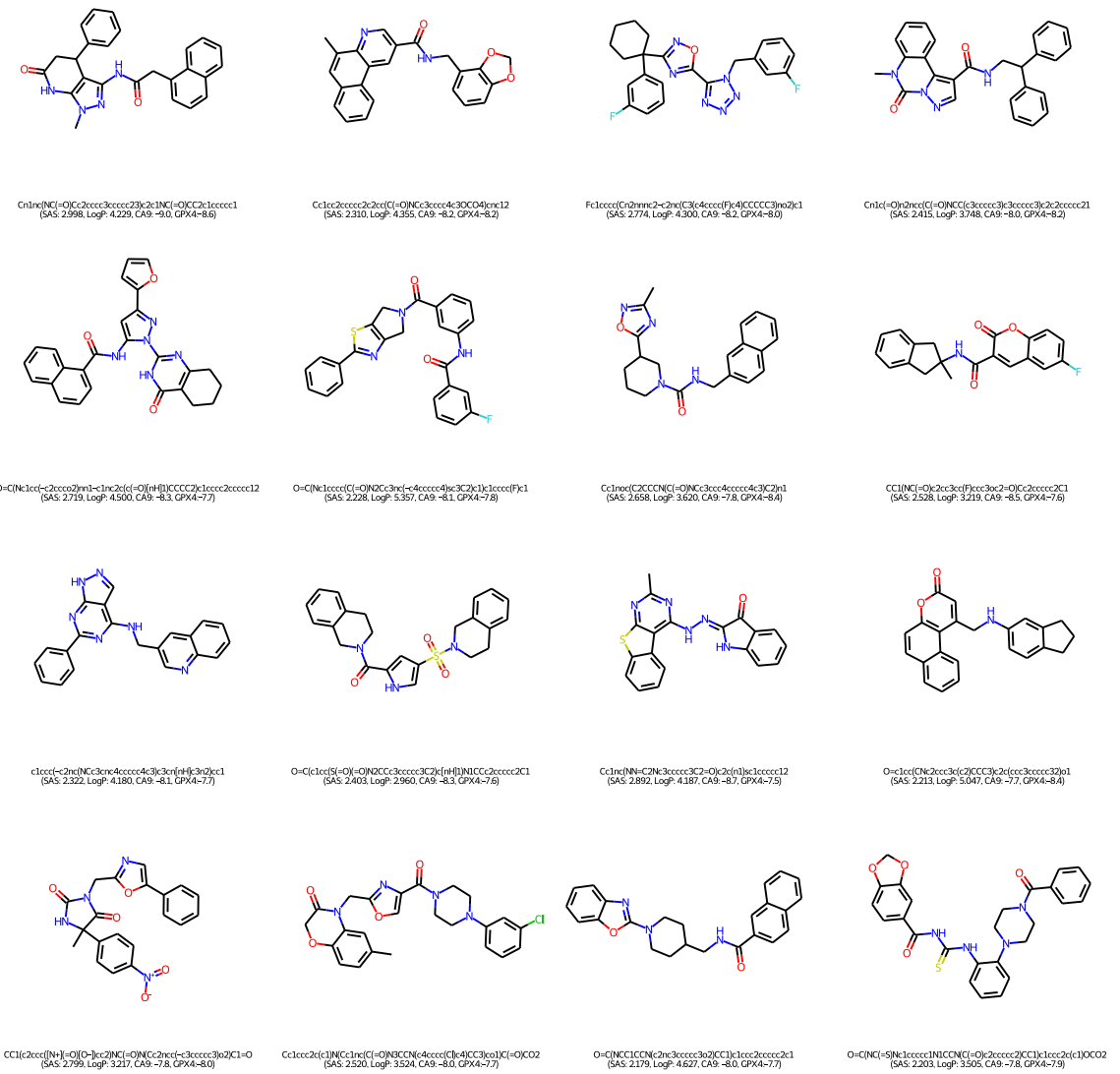
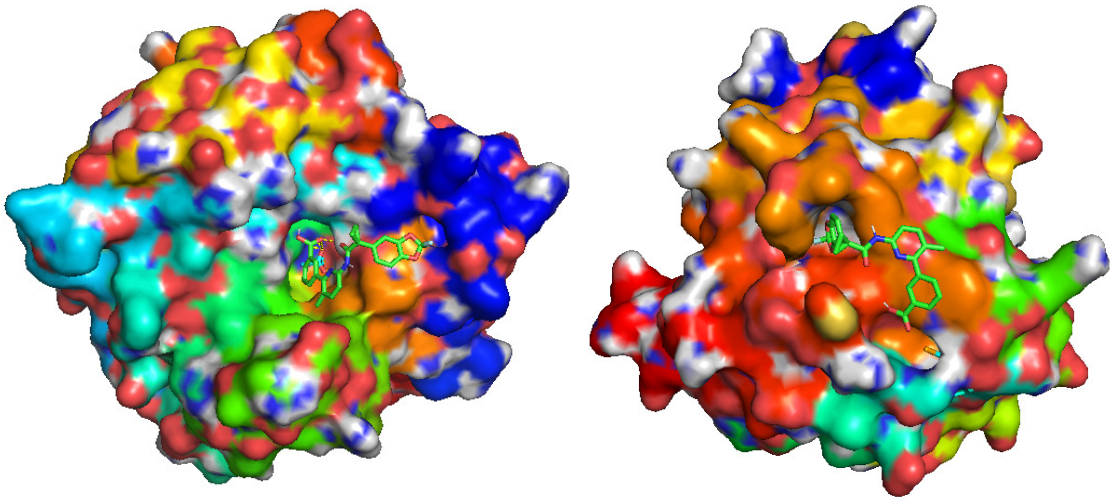


Figure 5.12: 2D graph visualization of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9, GPX4\}$ in combination with **JTVAE**, trained on the ZINC+DrugBank data. The results are ranked based on CA9 and GPX4 binding scores respectively, then sorted by the summation of two ranks. Only the top 16 molecules are shown due to space limitations. The complete list of samples can be found in A.4.

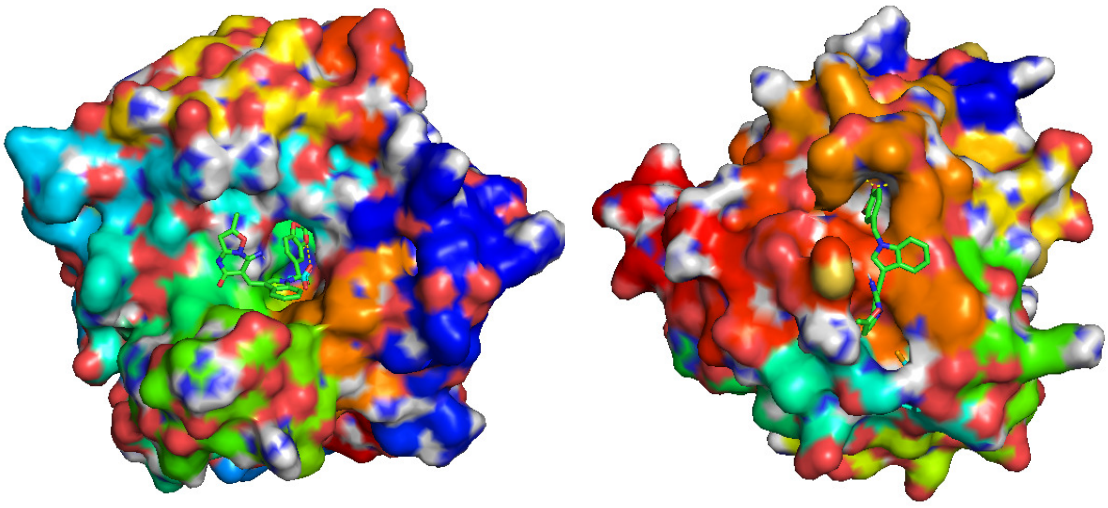
Among these sorted novel samples, the top three molecules generated by each method (FragVAE+DEL and JTVAE+DEL) have been subjected to virtual screening to assess their performance on the target 3D surfaces of CA9 and GPX4 proteins. Figure 5.13 displays the protein-ligand complex of molecule Cc1ccc(NC(=O)C2(c3ccc4c-(c3)OC(F)(F)O4)CC2)nc1-c1cccc(C(=O)O)c1 with the most favorable BAS (CA9: -8.7, GPX4: -8.4) generated by FragVAE+DEL, while Figure 5.14 and 5.15 shows

the other two promising samples actively docking on CA9 and GPX4 surface respectively. Figure 5.16, Figure 5.17 and Figure 5.18 demonstrate the binding interaction of top three samples from JTVAE+DEL associated with both protein targets, in particular the molecule Cn1nc(NC(=O)Cc2-cccc3ccccc23)c2c1NC(=O)CC2c1ccccc1 has an optimal binding affinity for both CA9 (-9.0) and GPX4 (-8.6). The novel samples examined are capable of adhering tightly to the docking sites and forming non-covalent bonds with the available resources. Our research suggests that these compounds could have the potential to be converted into potent drug molecules through wet-lab validation and synthesis.



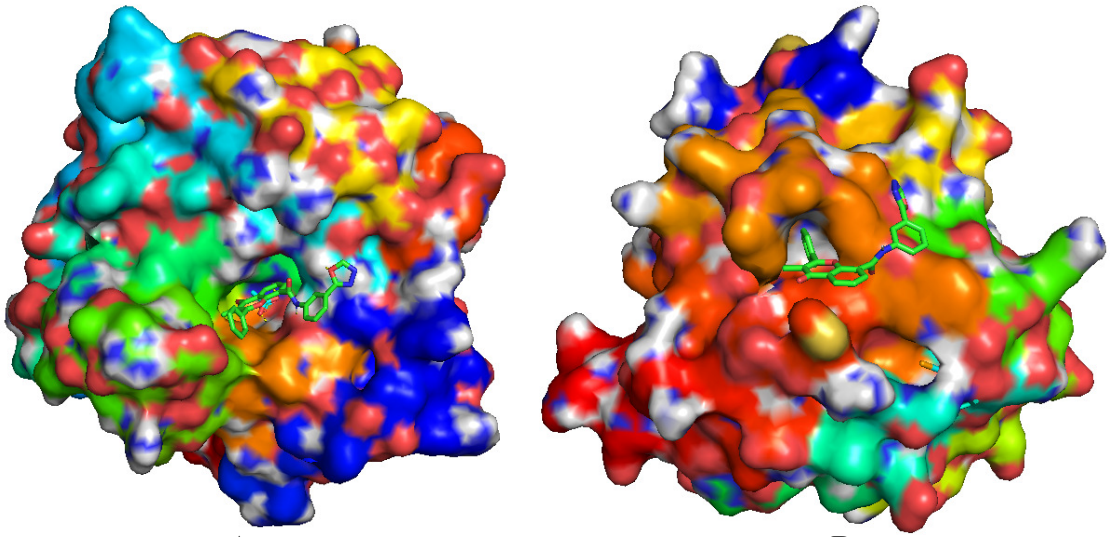
A **B**

Figure 5.13: Docking visualization of novel molecule Cc1ccc(NC(=O)C2(c3ccc4c(c3)-OC(F)(F)O4)CC2)nc1-c1cccc(C(=O)O)c1 docking to both CA9 protein and GPX4 protein surface, which achieved highest rank amongst the novel samples generated by FragVAE+DEL trained on ZINC+DrugBank data. This molecule has a CA9 binding affinity score of -8.7 and GPX4 score of -8.4 (SAS: 2.756, LogP: 4.747). **(A)** shows the molecule binding to the target binding site of CA9 protein, while **(B)** shows the molecule docking to the target site of GPX4.



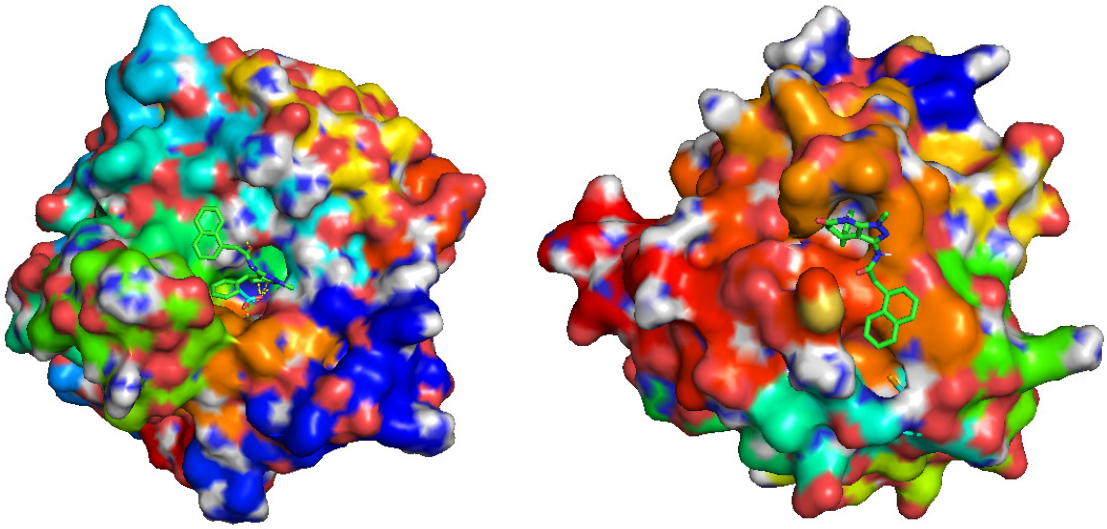
A **B**

Figure 5.14: Docking visualization of novel molecule CC1=CC2=NC(=O)C(=Cc3cn(Cc4ccc5c(c4)OCO5)c4ccccc34)C(=N)N2O1 (CA9: -8.5, GPX4: -7.9, SAS: 2.958, 3.868) docking to CA9 protein **(A)** and GPX4 protein **(B)** surface.



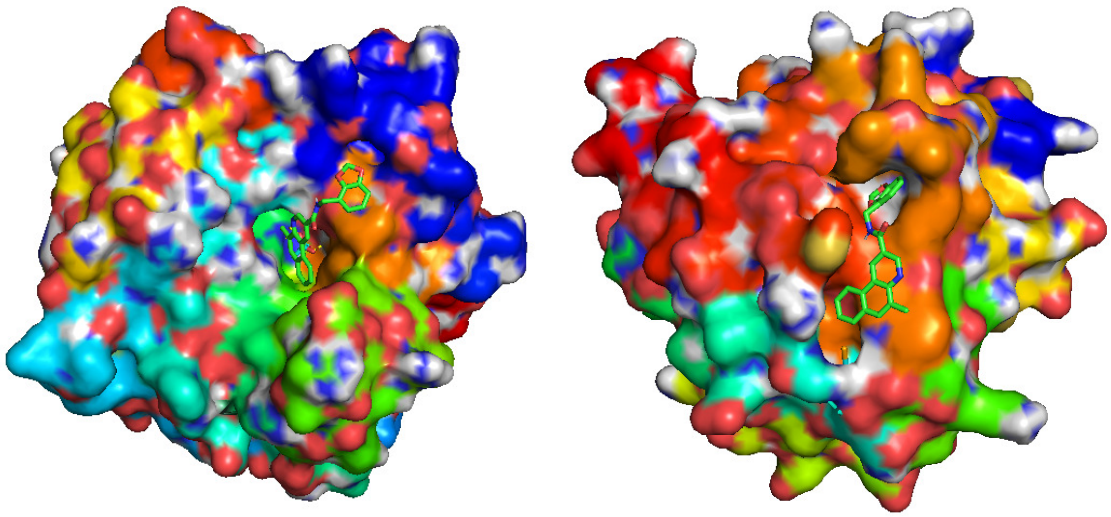
A **B**

Figure 5.15: Docking visualization of novel molecule Cc1c(-c2ccccc2)oc2c(C(=O)Nc3cccc(-c4nnco4)c3)cccc2c1=O (CA9: -8.4, GPX4: -8.0, SAS: 2.442, GPX4: 5.071) docking to both CA9 protein **(A)** and GPX4 protein **(B)** surface.



A **B**

Figure 5.16: Docking visualization of novel molecule Cn1nc(NC(=O)Cc2cccc3ccccc23)c2c1NC(=O)CC2c1ccccc1 docking to both CA9 protein and GPX4 protein surface, which achieved highest rank amongst the novel samples generated by JTVAE+DEL trained on ZINC+DrugBank data. This molecule has a CA9 binding affinity score of -9.0 and GPX4 score of -8.6 (SAS: 2.997, LogP: 4.229). **(A)** shows the molecule binding to the target binding site of CA9 protein, while **(B)** shows the molecule docking to the target site of GPX4.



A **B**

Figure 5.17: Docking visualization of novel molecule Cc1cc2ccccc2c2cc(C(=O)NCc3cccc4c3OCO4)cnc12 (CA9: -8.2, GPX4: -8.2, SAS: 2.310, LogP: 4.355) docking to both CA9 protein **(A)** and GPX4 protein **(B)** surface.

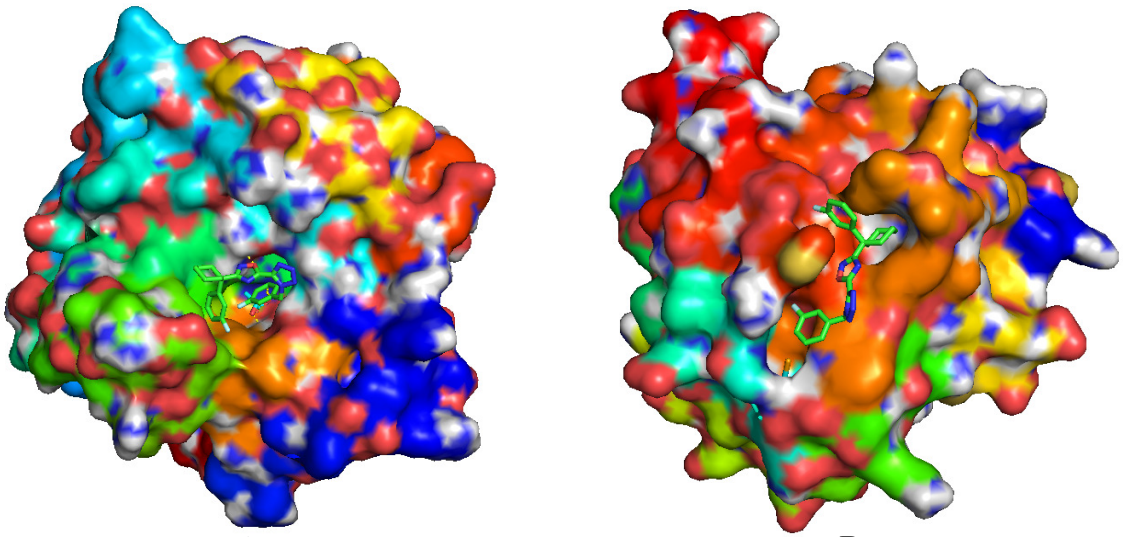


Figure 5.18: Docking visualization of novel molecule Cn1nc(NC(=O)Cc2cccc3ccccc23)c2c1NC(=O)CC2c1ccccc1 (CA9: -8.2, GPX4: -8.0, SAS: 2.774, LogP: 4.300) docking to both CA9 protein **(A)** and GPX4 protein **(B)** surface.

Chapter 6

Conclusion

6.1 Discussion

Drug discovery can be modelled as a multi-objective optimization problem over a vast search space. The advantages of target-aimed fragment-based drug design, as well as the powerful representation and modelling capacity of deep learning methods were the driving forces behind our research.

We propose using graph fragment-based deep generative models in the deep evolutionary learning process and incorporating the protein-ligand binding affinity score as one of the optimization objectives. This approach involves generating molecular structures via a graph-based model, evolving those structures with evolutionary techniques, and using the protein-ligand binding affinity score as a metric to guide the evolution towards generating molecules with desirable properties. By combining these processes, we aim to optimize the generation of molecules with specific docking properties to proteins of interest.

As part of the graph fragment-based approach, the junction tree-based deep generative model JTVAE was implemented in DEL to enhance its ability to extract structural information from molecules during training. SMILES fragment-based DEL was compared with graph fragment-based DEL using FragVAE as a benchmark. In a series of experiments, it has been demonstrated that our graph-based approach is capable of generating novel molecules possessing better quality in terms of Pareto front hypervolume and the number of novel samples satisfying the screening criteria when compared to a SMILES fragment-based deep generative model previously employed in the deep evolutionary learning framework. Since both approaches use VAE and DEL, it is likely that junction-tree-based graph fragmentation contributes to the improvement in performance. Unfortunately, due to the inability of the graph fragment-based

framework to accelerate training, fewer than optimal data were available for analysis.

Protein-ligand binding affinity score (BAS) is used as an optimization objective in DEL, along with solubility (LogP) and synthesizability (SAS). This enables us to identify and prioritize novel molecules that have specific binding affinity for a protein target, while also optimizing other pharmacological properties. Molecular mechanism simulations are used to calculate BAS values, which estimate a molecule's binding affinity towards a target protein surface area. Our BAS calculation module allows us to customize target information. Two protein targets, CA9 and GPX4, were studied separately and simultaneously. Through the ensuing screening process, we were able to acquire high-quality samples that had the potential to inhibit both cancer targets and to promote synthesis for cancer treatment.

6.2 Limitations

Despite the promising results achieved by our proposed approach, there are several limitations that must be considered when interpreting and extending the findings of this work.

1. Computational expense of molecular graph generative model JTVAE. In comparison to other methods such as SMILES-based generation, molecule graph generation is more computationally intensive because it involves more complex data structures and operations. In a molecular graph, atoms and bonds are represented as nodes and edges, respectively. This graph-based representations can incorporate more detailed information about molecules, such as stereochemistry and hydrogens, which increases computational complexity. Additionally, the current implementation of JTVAE is not amenable to multi-thread training. This limitation may hinder the scalability of the proposed approach and limit its applicability to large datasets.

2. High cost of protein-ligand binding score calculation. Protein-ligand binding score calculations require 3D docking simulations for all generated molecules, which is a computationally intensive process. As the simulation must be performed for both targets, this is especially challenging when two protein targets are considered. As a result, the overall optimization process incurs a substantial computational cost.

3. Overhead of evolutionary optimization process. As part of the evolutionary optimization process, non-dominated sorting and crowding distance calculations

for populations are performed, which can pose a computational challenge for the DEL framework. The non-dominated sorting algorithm is particularly demanding since it involves sorting a large number of solutions based on multiple criteria. As well as requiring significant computational resources, the crowding distance calculation can also increase the overhead of the evolutionary optimization process.

It is worth noting that these limitations are not unique to the proposed approach, but are common to many computational drug design methods using deep generative models and evolutionary optimization. To overcome these limitations, we intend to use alternative methods such as distributed computing, reducing the complexity of the generative model, or devising more efficient optimization algorithms.

6.3 Research Impact

This research was supported by the Artificial Intelligence for Design Challenge Program at the National Research Council Canada (NRC). Further, we have gained considerable attention in the field of drug design, and have been able to forge collaborations with the Digital Technologies Research Centre at NRC, as well as Dr. Michael Organ's Synthetic Chemistry Lab at University of Ottawa, and Dr. Shoukat Dehar's Group at BC Cancer Research Centre. In the context of our collaboration, we are facilitating the wet-lab validation and synthesis of the discovered novel compounds.

6.4 Future Work

As we move forward with our work, we will look for opportunities to improve our framework. One of the main priorities will be to address the computational expense of the JTVAE molecular graph generation component. One possible solution to this could be to explore more efficient architectures for the JTVAE network. Additionally, we will investigate the use of parallel computing techniques and distributed training methods to speed up the model training process. Another promising approach could be to use more powerful computational resources such as GPUs, which have been shown to be highly effective for deep learning tasks.

To address the computational bottleneck in protein-ligand binding simulation, one potential approach is to utilize more powerful computational resources. AutoDock

GPU is a highly efficient and optimized implementation of the popular AutoDock protein-ligand docking algorithm that utilizes the parallel processing capabilities of GPUs. This implementation has been shown to significantly speed up the docking process compared to traditional CPU-based implementations, such as QVina. By incorporating AutoDock GPU into our protein-ligand binding score calculation module, we can potentially reduce the computational cost of this step and increase the overall scalability of our framework.

Another area of future research will be to improve the efficiency and effectiveness of the molecular fragmentation methods used in our framework. We will explore different fragmentation strategies and evaluate their performance in terms of molecular quality and computational cost. Additionally, we will investigate the use of more advanced graph-based representations for molecular structures, such as those based on topological fingerprints or other descriptors.

In order to evaluate the performance of the framework, we will also investigate the use of statistical metrics to assess the statistical significance of the differences between the probability distributions of the generated molecules. This will allow us to better understand the performance of the framework and identify areas for improvement. Examples of such metrics could be t-test, standardized mean difference, Mann–Whitney U test. Furthermore, we plan to extend the evaluation scheme by incorporating more diverse objectives. And we are interested in applying our approach to other multi-objective design problems.

Bibliography

[1] K. Abbasi, P. Razzaghi, A. Poso, M. Amanlou, J. B. Ghasemi, and A. Masoudi-Nejad. DeepCDA: deep cross-domain compound-protein affinity prediction through LSTM and convolutional neural networks. *Bioinformatics*, 36(17):4633–4642, 2020.

[2] T. Adali and A. Ortega. Applications of graph theory [scanning the issue]. *Proceedings of the IEEE*, 106(5):784–786, 2018.

[3] S. Ahn, J. Kim, H. Lee, and J. Shin. Guiding deep molecular optimization with genetic exploration. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12008–12021. Curran Associates, Inc., 2020.

[4] N. M. AL-Salami. Evolutionary algorithm definition. *American Journal of Engineering and Applied Sciences*, 2(4):789–795, 2009.

[5] A. Alhossary, S. D. Handoko, Y. Mu, and C.-K. Kwoh. Fast, accurate, and reliable molecular docking with QuickVina 2. *Bioinformatics*, 31(13):2214–2216, 2015.

[6] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017.

[7] K. Atz, F. Grisoni, and G. Schneider. Geometric deep learning on molecular representations. *Nature Machine Intelligence*, 3:1023–1032, 2021.

[8] T. Back. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence.* IEEE, 1994.

[9] T. Baeck, D. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation*. CRC Press, Jan. 1997.

[10] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 01 2000.

[11] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs, 2013.

[12] Z. Chen, M. R. Min, S. Parthasarathy, and X. Ning. A deep generative model for molecule optimization via one fragment modification. *Nature Machine Intelligence*, 3(12):1040–1049, Dec. 2021.

[13] J. Chung, Çaglar Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv*, abs/1412.3555, 2014.

[14] C. A. Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys (CSUR)*, 32(2):109–143, 2000.

[15] H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *International Conference on Machine Learning*, 2016.

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

[17] L. David, A. Thakkar, R. Mercado, and O. Engkvist. Molecular representations in AI-driven drug discovery: a review and practical guide. *Journal of Cheminformatics*, 12(1), Sept. 2020.

[18] N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. *ArXiv*, abs/1805.11973, 2018.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[20] J. Degen, C. Wegscheid-Gerlach, A. Zaliani, and M. Rarey. On the art of compiling and using 'drug-like' chemical fragment spaces. *ChemMedChem*, 3(10):1503–1507, 2008.

[21] L. Dinh, J. N. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *ArXiv*, abs/1605.08803, 2017.

[22] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *International Conference on Neural Information Processing Systems*, 2015.

[23] J. Eberhardt, D. Santos-Martins, A. F. Tillack, and S. Forli. Autodock vina 1.2.0: New docking methods, expanded force field, and python bindings. *Journal of Chemical Information and Modeling*, 61(8):3891–3898, 2021.

[24] A. Eiben and J. Smith. From evolutionary computation to the evolution of things. *Nature*, 521(2014):476–482, 2015.

[25] T. Engel and J. Gasteiger. *Chemoinformatics: Achievements and Future Opportunities*. Wiley-VCH, 2018.

[26] T. Engel and J. Gasteiger. *Chemoinformatics: Basic Concepts and Methods*. Wiley-VCH, 2018.

[27] D. Erlanson. Introduction to fragment-based drug discovery. *Topics in Current Chemistry*, 317:1–32, 2011.

[28] R. Esworthy, K. Doan, J. H. Doroshow, and F.-F. Chu. Cloning and sequencing of the cDNA encoding a human testis phospholipid hydroperoxide glutathione peroxidase∗. *Gene*, 144(2):317–318, 1994.

[29] T. J. A. Ewing and I. D. Kuntz. Critical evaluation of search algorithms for automated molecular docking and database screening. *Journal of Computational Chemistry*, 18(9):1175–1189, 1997.

[30] Z. Fisher and K. Koruza. X-ray crystal structure of h/d exchanged (h/d) small monoclinic unit cell CA IX SV., 2019.

[31] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence Through Simulated Evolution*. Wiley, Chichester, WS, UK, 1966.

[32] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995.

[33] T. Gal. Multiple objective decision making - methods and applications: A state-of-the art survey: Ching-lai hwang and abu syed md. masud springer, berlin, 1979, xii + 351 pages, dfl. 46.90. soft cover. *European Journal of Operational Research*, 4:287–288, 1980.

[34] A. K. Ghose, V. N. Viswanadhan, and J. J. Wendoloski. A knowledge-based approach in designing combinatorial or medicinal chemistry libraries for drug discovery. 1. a qualitative and quantitative characterization of known drug databases. *Journal of Combinatorial Chemistry*, 1(1):55–68, Jan 1999.

[35] J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, 2017.

[36] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, pages 69–93. Elsevier, 1991.

[37] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. arxiv e-prints. *ArXiv:1406.2661*, 1406, 2014.

[38] R. Graham and P. Hell. On the history of the minimum spanning tree problem. *IEEE Annals of the History of Computing*, 7(1):43–57, 1985.

[39] M. Grandjean. A social network analysis of twitter: Mapping the digital humanities community. *Cogent Arts & Humanities*, 3(1):1171458, 2016.

[40] K. Grantham, M. Mukaidaisi, H. K. Ooi, M. S. Ghaemi, A. Tchagang, and Y. Li. Deep evolutionary learning for molecular design. *IEEE Computational Intelligence Magazine*, 17(2):14–28, 2022.

[41] R. Gómez-Bombarelli, D. Duvenaud, J. Hernández-Lobato, J. Aguilera-Iparraguirre, T. Hirzel, R. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[42] D. Hebb. *The Organization of Behavior*. John Wiley & Sons, New York, 1949.

[43] F. Herrera, M. Lozano, and A. Sanchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18:309–338, 2003.

[44] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[45] R. Hillig, D. Moosmayer, A. Hilpmann, J. Hoffmann, L. Schnirch, J. Eaton, V. Badock, and S. Gradl. Wildtype form (apo) of human GPX4 with se-cys46, 2020.

[46] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and Helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 3–10. Morgan Kaufmann Publishers Inc., 1993.

[47] J. H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, 1992.

[48] S. Honda, S. Shi, and H. R. Ueda. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery, 2019.

[49] J. Irwin and B. Shoichet. ZINC - A free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling*, 45(1):177–182, 2005.

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

[51] W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *International Conference on Machine Learning*, pages 2323–2332, 2018.

[52] W. Jin, R. Barzilay, and T. Jaakkola. Hierarchical generation of molecular graphs using structural motifs. In *ICML*, 2020.

[53] G. Jones, P. Willett, R. C. Glen, A. R. Leach, and R. Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology*, 267(3):727–748, 1997.

[54] D. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

[55] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013.

[56] T. N. Kipf and M. Welling. Variational graph auto-encoders, 2016.

[57] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[58] J. Koza and R. Poli. *Genetic Programming*. MIT Press, Cambridge, MA, USA, 1992.

[59] S. Kullback. Letters to the editor. *The American Statistician*, 41(4):338–341, 1987.

[60] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[61] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. Grammar variational autoencoder. In *International Conference on Machine Learning*, pages 1945–954, 2017.

[62] G. Landrum. RDKit: Open-source cheminformatics. 2006.

[63] J. B. Lee, R. Rossi, and X. Kong. Deep graph attention model. 2017.

[64] J. K. Leman, B. D. Weitzner, S. M. Lewis, J. Adolf-Bryfogle, N. Alam, R. F. Alford, M. Aprahamian, D. Baker, K. A. Barlow, P. Barth, B. Basanta, B. J. Bender, K. Blacklock, J. Bonet, S. E. Boyken, P. Bradley, C. Bystroff, P. Conway, S. Cooper, B. E. Correia, B. Coventry, R. Das, R. M. De Jong, F. DiMaio, L. Dsilva, R. Dunbrack, A. S. Ford, B. Frenz, D. Y. Fu, C. Geniesse, L. Goldschmidt, R. Gowthaman, J. J. Gray, D. Gront, S. Guffy, S. Horowitz, P.-S. Huang, T. Huber, T. M. Jacobs, J. R. Jeliazkov, D. K. Johnson, K. Kappel, J. Karanicolas, H. Khakzad, K. R. Khar, S. D. Khare, F. Khatib, A. Khramushin, I. C. King, R. Kleffner, B. Koepnick, T. Kortemme, G. Kuenze,

B. Kuhlman, D. Kuroda, J. W. Labonte, J. K. Lai, G. Lapidoth, A. Leaver-Fay, S. Lindert, T. Linsky, N. London, J. H. Lubin, S. Lyskov, J. Maguire, L. Malmström, E. Marcos, O. Marcu, N. A. Marze, J. Meiler, R. Moretti, V. K. Mulligan, S. Nerli, C. Norn, S. Ó'Conchúir, N. Ollikainen, S. Ovchinnikov, M. S. Pacella, X. Pan, H. Park, R. E. Pavlovicz, M. Pethe, B. G. Pierce, K. B. Pilla, B. Raveh, P. D. Renfrew, S. S. R. Burman, A. Rubenstein, M. F. Sauer, A. Scheck, W. Schief, O. Schueler-Furman, Y. Sedan, A. M. Sevy, N. G. Sgourakis, L. Shi, J. B. Siegel, D.-A. Silva, S. Smith, Y. Song, A. Stein, M. Szegedy, F. D. Teets, S. B. Thyme, R. Y.-R. Wang, A. Watkins, L. Zimmerman, and R. Bonneau. Macromolecular modeling and design in rosetta: recent methods and frameworks. *Nature Methods*, 17(7):665–680, Jul 2020.

[65] X. Q. Lewell, D. B. Judd, S. P. Watson, and M. M. Hann. RECAPRetrosynthetic combinatorial analysis procedure: a powerful new technique for identifying privileged molecular fragments with useful applications in combinatorial chemistry. *Journal of Chemical Information and Computer Sciences*, 38(3):511–522, 1998.

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

[67] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs. 2018.

[68] C. Lipinski, F. Lombardo, B. Dominy, and P. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews*, 46(1-4):3–26, 2001.

[69] C. A. Lipinski, F. Lombardo, B. W. Dominy, and P. J. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews*, 23(1):3–25, 1997.

[70] Q. Liu, M. Allamanis, M. Brockschmidt, and A. L. Gaunt. Constrained graph variational autoencoders for molecule design. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 7806–7815, 2018.

[71] S. Luo, J. Guan, J. Ma, and J. Peng. A 3D generative model for structure-based drug design. In *Conference on Neural Information Processing Systems*, 2021.

[72] A. R. Mashaghi, A. Ramezanpour, and V. Karimipour. Investigation of a protein complex network. *The European Physical Journal B*, 41(1):113–121, 2004.

[73] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. 2016.

[74] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Berlin Heidelberg, 1992.

[75] Z. Michalewicz, R. Hinterding, and M. Michalewicz. Evolutionary algorithms. In *Fuzzy Evolutionary Computation*. Springer US, 1997.

[76] K. Miettinen. *Nonlinear multiobjective optimization*. Springer Science & Business Media, 1999.

[77] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

[78] T. Mikolov, M. Karafiát, L. Burget, J. H. Cernocký, and S. Khudanpur. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

[79] B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Syst.*, 1995.

[80] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[81] J. Mueller, D. Gifford, and T. Jaakkola. Sequence to better sequence: Continuous revision of combinatorial structures. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[82] C. A. Nicolaou, J. Apostolakis, and C. S. Pattichis. De novo drug design using multiobjective evolutionary graphs. *Journal of Chemical Information and Modeling*, 49(2):295–307, 2009.

[83] A. Nigam, P. Friederich, M. Krenn, and A. Aspuru-Guzik. Augmenting genetic algorithms with deep neural networks for exploring the chemical space. *ArXiv*, 2019.

[84] A. Oduguwa, A. Tiwari, R. Roy, and C. Bessant. An overview of soft computing techniques used in the drug discovery process. In *Applied Soft Computing Technologies: The Challenge of Complexity*. Springer Berlin Heidelberg, 2006.

[85] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1105–1114, 2016.

[86] N. Pagadala, K. Syed, and J. Tuszynski. Software for molecular docking: A review. *Biophysical Reviews*, 9(2):91–102, 12 2016.

[87] T.-H. Pham, L. Xie, and P. Zhang. FAME: Fragment-based conditional molecular generation for phenotypic drug discovery. In *Proceedings of the 2022 SIAM International Conference on Data Mining (SDM)*, pages 720–728. Society for Industrial and Applied Mathematics, 2022.

[88] M. Podda, D. Bacciu, and A. Micheli. A deep generative model for fragment-based molecule generation. In *International Conference on Artificial Intelligence and Statistics*, pages 2240–2250, 2020.

[89] P. Pogány, N. Arad, S. Genway, and S. D. Pickett. De novo molecule design by translating from reduced graphs to SMILES. *Journal of Chemical Information and Modeling*, 59(3):1136–1146, 2019.

[90] P. Polishchuk. CReM: chemically reasonable mutations framework for structure generation. *Journal of Cheminformatics*, 12(1), 2020.

[91] A. S. Powers, H. H. Yu, P. Suriana, and R. O. Dror. Fragment-based ligand generation guided by geometric deep learning on protein-ligand structure. *BioRxiv*, 2022.

[92] O. Prykhodko, S. V. Johansson, P.-C. Kotsias, J. Arús-Pous, E. J. Bjerrum, O. Engkvist, and H. Chen. A de novo molecular generation method using latent vector based generative adversarial network. *Journal of Cheminformatics*, 11(1), 2019.

[93] M. Rarey, B. Kramer, T. Lengauer, and G. Klebe. A fast flexible docking method using an incremental construction algorithm. *Journal of Molecular Biology*, 261(3):470–489, 1996.

[94] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, 1973.

[95] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[96] S. Z. Sajadi, M. A. Zare Chahooki, S. Gharaghani, and K. Abbasi. AutoDTI++: deep unsupervised learning for DTI prediction by autoencoders. *BMC Bioinformatics*, 22(1):204, Apr. 2021.

[97] B. Samanta, A. De, G. Jana, P. K. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez. NeVAE: A deep generative model for molecular graphs. *Journal of Machine Learning Research*, 21, 2020.

[98] B. Sattarov, I. I. Baskin, D. Horvath, G. Marcou, E. J. Bjerrum, and A. Varnek. De novo molecular design by combining deep autoencoder recurrent neural networks with generative topographic mapping. *Journal of Chemical Information and Modeling*, 59 3:1182–1196, 2019.

[99] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[100] J. D. Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pages 93–100. Psychology Press, 2014.

[101] L. Schrödinger. The PyMOL molecular graphics system, version 1.8. 2015.

[102] M. H. S. Segler, T. Kogej, C. Tyrchan, and M. P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS Central Science*, 4(1):120–131, 2017.

[103] C. Shi, M. Xu, Z. Zhu, W. Zhang, M. Zhang, and J. Tang. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations*, 2020.

[104] S. Shuker, P. Hajduk, R. Meadows, and S. Fesik. Discovering high-affinity ligands for proteins: SAR by NMR. *Science*, 274(5292):11531–1534, 1996.

[105] M. Simonovsky and N. Komodakis. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pages 412–422, 2018.

[106] M. Singh, B. Tam, and B. Akabayov. Nmr-fragment based virtual screening: A brief overview. *Molecules*, 23(2), 2018.

[107] P. Span, J. Bussink, P. Manders, L. Beex, and C. Sweep. Carbonic anhydrase-9 expression levels and prognosis in human breast cancer: association with treatment outcome. *British Journal of Cancer*, 89(2):271–276, 2003.

[108] W. Stadler. A survey of multicriteria optimization or the vector maximum problem. *Journal of Optimization Theory and Applications*, 29(1):1–52, 1979.

[109] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[110] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *ArXiv*, 2013.

[111] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, 2015.

[112] J. M. Tomczak. *Deep Generative Modeling*. Springer International Publishing, 2022.

[113] O. Trott and A. Olson. AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010.

[114] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[115] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016.

[116] F. Vecchio, F. Miraglia, F. Piludu, G. Granata, R. Romanello, M. Caulo, V. Onofrj, P. Bramanti, C. Colosimo, and P. M. Rossini. "small world" architecture in brain connectivity and hippocampal volume in alzheimer's disease: a study via graph theory from EEG data. *Brain Imaging and Behavior*, 11(2):473–485, 2016.

[117] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

[118] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265, 2016.

[119] W. Walters, M. T. Stahl, and M. A. Murcko. Virtual screening - an overview. *Drug Discovery Today*, 3(4):160–178, 1998.

[120] D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

[121] X. Wang and L. Cao. *Genetic Algorithms - Theory, Applications, and Software Implementations*. Xi'an Jiaotong University Press, 2002.

[122] D. 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.

[123] C. G. Wermuth. *The practice of medicinal chemistry*. Academic Press, 2011.

[124] D. Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.

[125] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(2014):949–980, 2014.

[126] D. S. Wishart, Y. D. Feunang, A. C. Guo, E. J. Lo, A. Marcu, J. R. Grant, T. Sajed, D. Johnson, C. Li, Z. Sayeeda, N. Assempour, I. Iynkkaran, Y. Liu, A. Maciejewski, N. Gale, A. Wilson, L. Chin, R. Cummings, D. Le, A. Pon,

C. Knox, and M. Wilson. DrugBank 5.0: A major update to the DrugBank database for 2018. *Nucleic Acids Research*, 46:D1074–D1082, 2018.

[127] C. Yan, S. Wang, J. Yang, T. Xu, and J. Huang. Re-balancing variational autoencoder loss for molecule sequence generation. In *ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, page Article Number 54, 2020.

[128] K. Yang, K. Swanson, W. Jin, C. Coley, P. Eiden, H. Gao, A. Guzman-Perez, T. Hopper, B. Kelley, M. Mathea, A. Palmer, V. Settels, T. Jaakkola, K. Jensen, and R. Barzilay. Analyzing learned molecular representations for property prediction. *Journal of Chemical Information and Modeling*, 59(8):3370–3388, 2019.

[129] X. Yao. Global optimisation by evolutionary algorithms. In *Proceedings of IEEE International Symposium on Parallel Algorithms Architecture Synthesis*, pages 282–291, 1997.

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

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

[132] C. Zang and F. Wang. Moflow: An invertible flow model for generating molecular graphs. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 617–626, 2020.

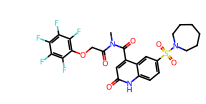
[133] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *ArXiv*, abs/1803.07294, 2018.

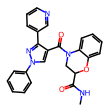
[134] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization. *Metaheuristics for Multiobjective Optimisation*, 535(535):21–40, 2004.

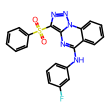
Appendix A

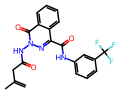
Additional Experimental Analysis

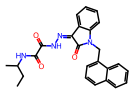
A.1 Single Protein Target

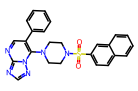


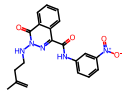


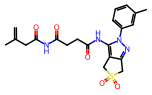


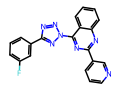


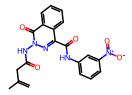


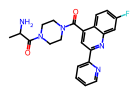


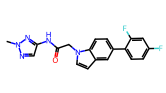


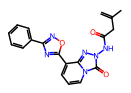


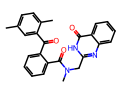


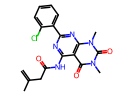


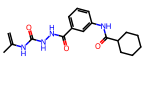


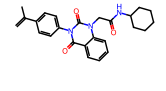


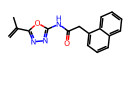


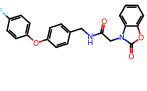


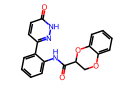


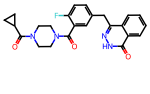


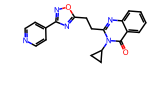


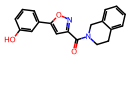


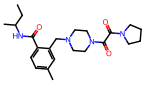


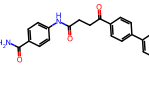


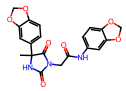


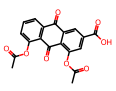


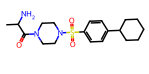


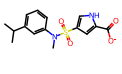


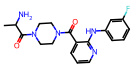



CC1(c2ccc3c(c2)OCO3)NC(=O)N(CC(=O)Nc2ccc3c(c2)OCO3)C1=O
(SAS: 2,808, LogP: 1,550, CA9: -8,2)

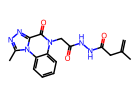

CC(=O)Oc1ccc2c1C(=O)c1c(OC(C)=O)ccc(C(=O)O)cc1C2=O
(SAS: 2,576, LogP: 2,011, CA9: -8,1)

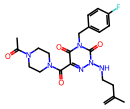

CC(N)C(=O)N1CCN(S(=O)(=O)c2ccc(C3CCCCC3)cc2)CC1
(SAS: 2,548, LogP: 1,914, CA9: -8,1)

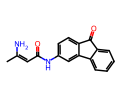

CC(C)c1ccc(N(C)S(=O)(=O)c2cnn(C)c2C(=O)O)cc1
(SAS: 2,808, LogP: 1,327, CA9: -8,1)

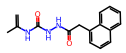

CC(N)C(=O)N1CCN(C(=O)c2ccccc2N2CCCC2)CC1
(SAS: 2,607, LogP: 1,596, CA9: -8,1)

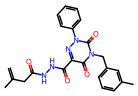

O=C(Nc1ccccc1)C1CC(O)CN1S(=O)(=O)c1ccc([N+](=O)[O-])cc1
(SAS: 2,805, LogP: 1,357, CA9: -8,1)

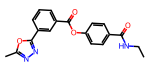

C=C(C)CC(=O)NNC(=O)Cn1c(=O)c2ccn(C)c2n2ccccc21
(SAS: 2,737, LogP: 0,466, CA9: -8,1)

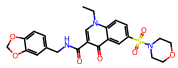

C=C(C)CCNn1nnc(C(=O)N3CCN(C(=O)CC2)c(=O)n(C)c2ccc(F)cc2)c1=O
(SAS: 2,876, LogP: 0,406, CA9: -8,1)

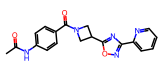

CC(N)=CC(=O)Nc1ccc2c(c1)-c1ccccc1C2=O
(SAS: 2,175, LogP: 2,699, CA9: -8,1)

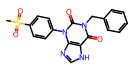

C=C(C)NC(=O)NNC(=O)Cc1cccc2ccccc12
(SAS: 2,259, LogP: 2,246, CA9: -8,0)

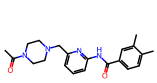

C=C(C)CC(=O)NNC(=O)c1nn(-c2ccccc2)c(=O)n(Cc2ccccC)c2)cc1=O
(SAS: 2,499, LogP: 1,478, CA9: -8,0)

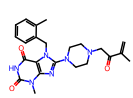

CCNC(=O)c1ccc(OC(=O)c2ccnc(-c3nnc(C)o3)c2)cc1
(SAS: 2,051, LogP: 3,014, CA9: -8,0)

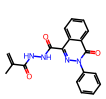

CCn1cc(C(=O)NCc2ccc3c(c2)OCO3)c(=O)c2cc(S(=O)(=O)N3CCOCC3)ccc21
(SAS: 2,496, LogP: 1,701, CA9: -8,0)

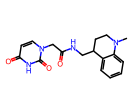

CC(=O)Nc1ccc(C(=O)N2CC(c3nc(-c4cccnc4)no3)C2)cc1
(SAS: 2,204, LogP: 2,330, CA9: -8,0)

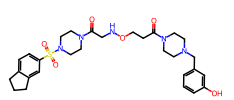

CS(=O)(=O)c1ccc(-n2c(=O)n(Cc3ccccc3)c(=O)c3[nH]cnc32)cc1
(SAS: 2,312, LogP: 1,327, CA9: -7,9)

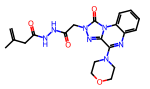

CC(=O)N1CCN(Cc2ccccNC(=O)c3ccc(Cl)c(Cl)c3)c2)CC1
(SAS: 2,056, LogP: 2,615, CA9: -7,9)

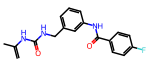

C=C(C)C(=O)CN1CCN(c2nc3c(c(=O)[nH]c(=O)n3C)n2Cc2ccccc2C)CC1
(SAS: 2,696, LogP: 1,047, CA9: -7,9)

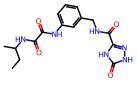

C=C(C)C(=O)NNC(=O)c1nn(-c2ccccc2)c(=O)c2ccccc12
(SAS: 2,207, LogP: 1,723, CA9: -7,9)

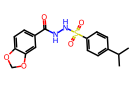

CN1CCC(CN(C(=O)Cn2ccc(=O)[nH]c2=O)c2ccccc21
(SAS: 2,929, LogP: 0,277, CA9: -7,9)

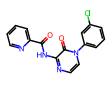

O=C(CCONCC(=O)N1CCN(S(=O)(=O)c2ccc3c(c2)CCC3)CC1)N1CCN(Cc2cccc(O)c2)CC1
(SAS: 2,795, LogP: 0,970, CA9: -7,8)

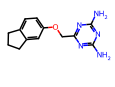

C=C(C)CC(=O)NNC(=O)Cn1nc2c(N3CCOCC3)nc3ccccc3n2c1=O
(SAS: 2,819, LogP: -0,006, CA9: -7,8)

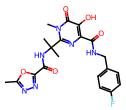

C=C(C)NC(=O)NC(=O)Cc1ccc(NC(=O)c2ccc(F)cc2)cc1
(SAS: 1,992, LogP: 3,411, CA9: -7,8)

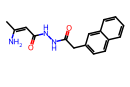

CCC(C)NC(=O)C(=O)Nc1ccc(CNC(=O)c2n[nH]c(=O)[nH]2)c1
(SAS: 2,987, LogP: -0,119, CA9: -7,8)

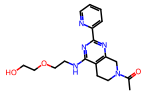

CC(C)c1ccc(S(=O)(=O)NNC(=O)c2ccc3c(c2)OCO3)cc1
(SAS: 2,082, LogP: 2,162, CA9: -7,8)

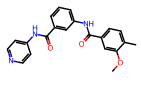

O=C(Nc1nccnn1-c2ccccc(Cl)c2)c1=O)c1ccccn1
(SAS: 2,154, LogP: 2,533, CA9: -7,8)

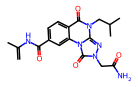

Nc1nc(N)nc(OCc2ccc3c(c2)CCC3)n1
(SAS: 2,252, LogP: 1,104, CA9: -7,8)

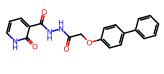

Cc1nnn(C(=O)NC(C)(C)c2nc(C(=O)NCc3ccc(F)cc3)c(O)n(C)n2C)o1
(SAS: 2,871, LogP: 0,912, CA9: -7,8)

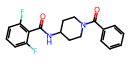

CC(N)=CC(=O)NNC(=O)Cc1ccc2ccccc2c1
(SAS: 2,226, LogP: 1,392, CA9: -7,7)

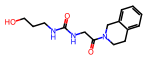

CC(=O)N1CCc2c(nc(-c3cccn3)nc2NCCOCCO)C1
(SAS: 2,611, LogP: 0,864, CA9: -7,7)

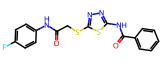

COc1cc(C(=O)Nc2cccc(C(=O)Nc3ccncs3)c2)ccc1C
(SAS: 1,808, LogP: 3,903, CA9: -7,7)

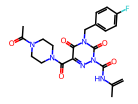

C=C(C)NC(=O)c1ccc2c(=O)n(CC(C)C)c3nn(CC(N)=O)c(=O)n3c2c1
(SAS: 2,857, LogP: 0,216, CA9: -7,7)

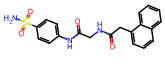

O=C(COc1ccc(-c2cccc2)cc1)NNC(=O)c1ccc[nH]c1=O
(SAS: 1,986, LogP: 1,882, CA9: -7,7)

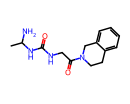

O=C(NC1CCN(C(=O)c2ccccc2)CC1)c1c[F]ccccc1F
(SAS: 1,863, LogP: 2,999, CA9: -7,6)

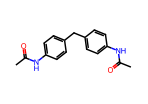

O=C(NCCCO)NCC(=O)N1CCc2ccccc2C1
(SAS: 2,081, LogP: 0,253, CA9: -7,6)

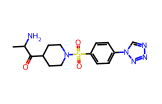

O=C(CSc1nnc(NC(=O)c2ccccc2)s1)Nc1ccc(F)cc1
(SAS: 1,854, LogP: 3,660, CA9: -7,5)


C=C(C)NC(=O)n1nnc(C(=O)N2CCN(C(C)=O)CC2)c(=O)n(Cc2ccc(F)cc2)c1=O
(SAS: 2,747, LogP: -0,012, CA9: -7,5)


NS(=O)(=O)c1ccc(NC(=O)CNC(=O)Cc2ccnc3ccccc23)cc1
(SAS: 1,902, LogP: 1,785, CA9: -7,5)


CC(N)NC(=O)NCC(=O)N1CCc2ccccc2C1
(SAS: 2,720, LogP: 0,175, CA9: -7,5)


CC(=O)Nc1ccc(Cc2ccc(NC)C)=O)nc2)cc1
(SAS: 1,573, LogP: 3,194, CA9: -7,4)


CCN(C(=O)C1CCCN(S(=O)(=O)c2ccc(-n3cnnn3)cc2)CC1
(SAS: 2,880, LogP: -0,021, CA9: -7,4)

C=C(C)NS(=O)(=O)c1ccc2[nH]c(=O)c(=O)[nH]c2c1
(SAS: 2,837, LogP: 0,028, CA9: -7,3)

O=C1c2c(O)ccc(O)c2C(=O)c2c1NCCNCCO)ccc[NCCNCCO)c21
(SAS: 2,814, LogP: -0,139, CA9: -7,3)

CC(=O)Nc1ccc[C(=O)COc2ccc(C(N)=O)cc2)cc1
(SAS: 1,693, LogP: 2,006, CA9: -7,3)

CC(N)CN(C(=O)Cn1nnc2ccc(-c3cccc3)nn2)c1=O
(SAS: 2,806, LogP: 0,921, CA9: -7,3)

O=C([O-])C[NC(=O)C1CCC(Cn2nnc3ccccc3c2=O)CC1
(SAS: 2,553, LogP: 0,069, CA9: -7,3)

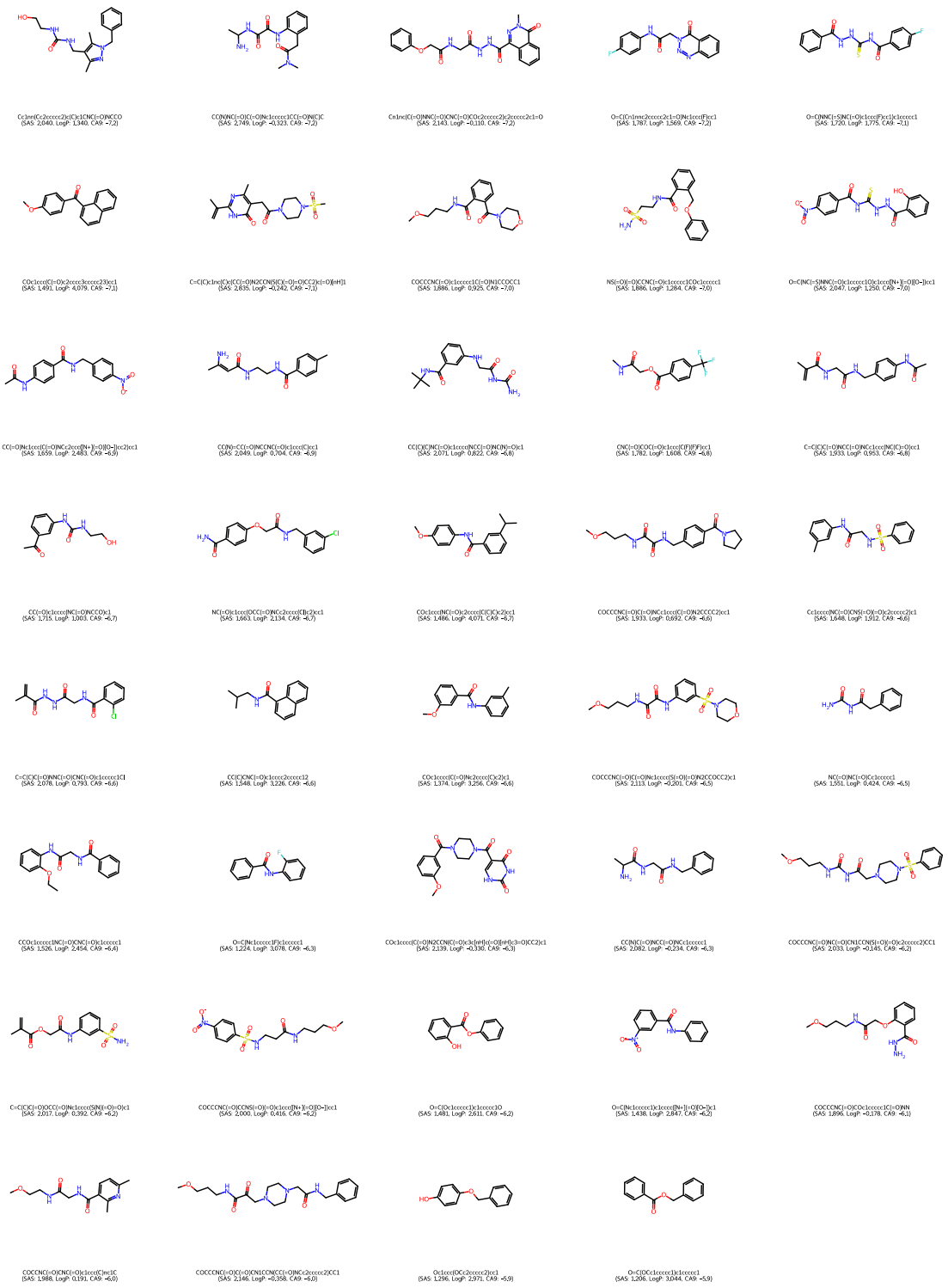
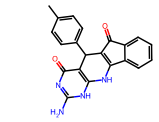
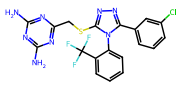
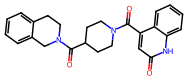
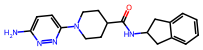


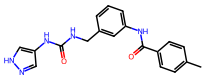
Figure A.1: Full list of the 2D graph examples of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9\}$ in combination with **FragVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with advantageous BAS, the samples are sorted descendingly by the CA9 binding score.

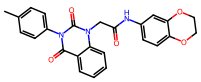

Cc1ccc(C2C3=C(Nc4[nH]c(N)nc(=O)c42)c2ccccc2C3=O)cc1
(SAS: 2,971, LogP: 2,826, CA9: -9,1)

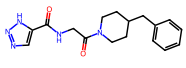

Nc1nc(N)nc(C5c2nn(-c3ccccc3[1]c3n2-c2ccccc2C(F)(F)F)n1
(SAS: 2,578, LogP: 4,248, CA9: -8,9)

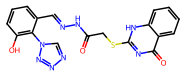

O=C(c1ccc(=O)[nH]c2ccccc12)N1CCC(C(=O)N2CCc3ccccc3C2)CC1
(SAS: 2,237, LogP: 2,965, CA9: -8,9)

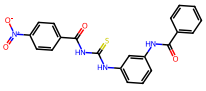

Nc1ccc(N2CCC(C(=O)NC3Cc4ccccc4C3)CC2)nn1
(SAS: 2,352, LogP: 1,559, CA9: -8,8)

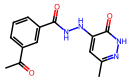

Cc1ccc(C(=O)Nc2cccc(CNC(=O)Nc3cc[nH]c3)c2)cc1
(SAS: 2,022, LogP: 3,292, CA9: -8,8)

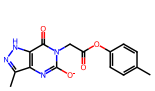

Cc1ccc(-n2c(=O)c3ccccc3n(CC(=O)Nc3ccc4c(c3)OCCO4)c2=O)cc1
(SAS: 2,196, LogP: 2,871, CA9: -8,5)

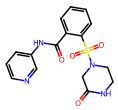

O=C(NCC(=O)N1CCC(Cc2ccccc2)CC1)c1cnn[nH]1
(SAS: 2,570, LogP: 1,016, CA9: -8,4)

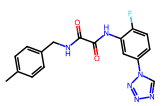

O=C(CSc1nc(=O)c2ccccc2[nH]1)NN=Cc1ccccc1O)c1-n1cnnn1
(SAS: 2,805, LogP: 0,847, CA9: -8,3)

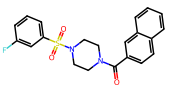

O=C(NC(=S)Nc1ccccc1N(C(=O)c2ccccc2)c1ccccc1[N+](=O)[O-])cc1
(SAS: 1,894, LogP: 3,974, CA9: -8,3)

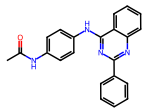

CC(=O)c1ccc(C(=O)NNc2cc(C)c[nH]c2=O)c1
(SAS: 2,399, LogP: 1,038, CA9: -8,2)

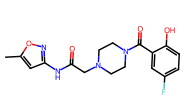

Cc1ccc(OC(=O)Cn2c[O-]nc3c(C)n[nH]c3c2=O)cc1
(SAS: 2,965, LogP: 0,416, CA9: -8,2)

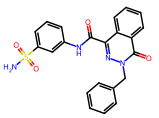

O=C1CN(S(=O)(=O)c2ccccc2C(=O)Nc2ccccc2)CCN1
(SAS: 2,222, LogP: 0,454, CA9: -8,1)

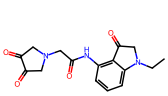

Cc1ccc(CNC(=O)C(=O)Nc2cc(-n3cnnn3)ccc2F)cc1
(SAS: 2,193, LogP: 1,265, CA9: -8,1)

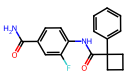

O=C(c1ccc2ccccc2c1)N1CCN(S(=O)(=O)c2ccc(F)c2)CC1
(SAS: 1,907, LogP: 3,126, CA9: -8,1)

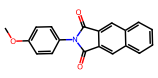

CC(=O)Nc1ccc(Nc2nc(-c3ccccc3)nc3ccccc23)cc1
(SAS: 1,832, LogP: 4,999, CA9: -8,0)

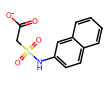

Cc1nc(NC(=O)CN2CCN(C(=O)c3cnc(F)ccc3O)CC2)no1
(SAS: 2,166, LogP: 1,224, CA9: -7,9)

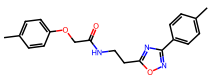

NS(=O)(=O)c1ccccc(NC(=O)c2nn(CC3ccccc3)c(=O)c3ccccc23)c1
(SAS: 2,050, LogP: 2,345, CA9: -7,9)

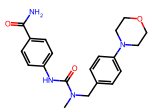

CCN1CC(=O)c2ccNC(=O)CN3CC(=O)C(=O)C3)cccc21
(SAS: 2,874, LogP: 0,102, CA9: -7,9)

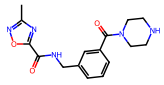

NC(=O)c1ccc(NC(=O)C2(c3ccccc3)CCC2)c(F)c1
(SAS: 2,025, LogP: 2,985, CA9: -7,9)

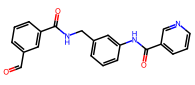

COc1ccc(N2C(=O)c3cc4ccccc4cc3C2=O)cc1
(SAS: 1,799, LogP: 3,649, CA9: -7,8)

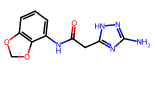

O=C([O-])CS(=O)(=O)Nc1ccc2ccccc2c1
(SAS: 2,376, LogP: 0,331, CA9: -7,8)

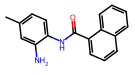

Cc1ccc(OCC(=O)NCCc2nc(-c3ccc(C)cc3)no2)cc1
(SAS: 1,918, LogP: 3,091, CA9: -7,8)

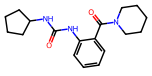

CN(Cc1ccc(N2CCOCC2)cc1)C(=O)Nc1ccc(C)N)=O)cc1
(SAS: 1,989, LogP: 2,286, CA9: -7,8)

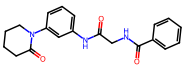

Cc1noc(C(=O)NCc2ccccc(C(=O)N3CCNCC3)c2)n1
(SAS: 2,317, LogP: 0,353, CA9: -7,7)

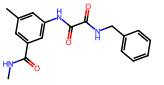

O=C1ccccc1C(=O)NCc2ccccNC(=O)c3ccnnc3)c2)c1
(SAS: 1,945, LogP: 3,076, CA9: -7,7)

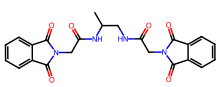

Nc1nl[nH]c(CC(=O)Nc2ccnc3c2OCO3)n1
(SAS: 2,554, LogP: 0,297, CA9: -7,6)

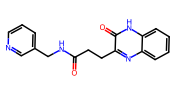

Cc1ccc(NC(=O)c2ccnc3ccccc23)c(N)c1
(SAS: 1,676, LogP: 3,983, CA9: -7,6)

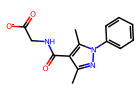

O=C(Nc1ccccc1C(=O)N1CCCCC1)NC1CCCC1
(SAS: 1,831, LogP: 3,377, CA9: -7,6)

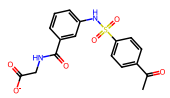

O=C(CNC(=O)c1ccccc1)Nc1ccccN2CCCCC2=O)c1
(SAS: 1,833, LogP: 2,572, CA9: -7,6)

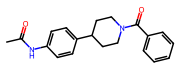

CNC(=O)c1cc(C)cc(NC(=O)C(=O)NCc2ccccc2)c1
(SAS: 1,858, LogP: 1,610, CA9: -7,6)

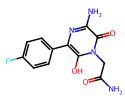

CC(CNC(=O)CN1C(=O)c2ccccc2C1=O)NC(=O)CN1C(=O)c2ccccc2C1=O
(SAS: 2,703, LogP: 0,200, CA9: -7,5)

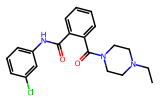

O=C(CCc1nc2ccccc2[nH]1)c1=O)NCc1ccncn1
(SAS: 2,077, LogP: 1,567, CA9: -7,4)

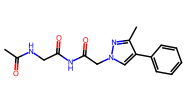

Cc1nn(-c2ccccc2)c(C)c1C(=O)NCC(=O)O[O-]
(SAS: 2,409, LogP: -0,031, CA9: -7,4)

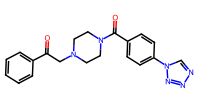

CC(=O)c1ccc(S(=O)(=O)Nc2ccc(C(=O)NCC(=O)[O-])c2)cc1
(SAS: 2,249, LogP: 0,170, CA9: -7,4)

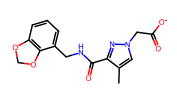

CC(=O)Nc1ccc(C2CCN(C(=O)c3ccccc3)CC2)cc1
(SAS: 1,716, LogP: 3,665, CA9: -7,3)


NC(=O)Cn1c(O)c(-c2ccc(F)cc2)nc(N)c1=O
(SAS: 2,485, LogP: -0,177, CA9: -7,3)


CCN1CCN(C(=O)c2ccccc2C(=O)Nc2ccc(Cl)c2)CC1
(SAS: 1,828, LogP: 3,370, CA9: -7,3)


CC(=O)NCC(=O)NC(=O)Cn1cc(-c2ccccc2)c(C)n1
(SAS: 2,216, LogP: 0,637, CA9: -7,3)


O=C(CN1CCN(C(=O)c2ccc(-n3cnnn3)cc2)CC1)c1ccccc1
(SAS: 2,072, LogP: 1,303, CA9: -7,3)


Cc1cnc(CC(=O)[O-]nc1C(=O)NCc1ccoc2c1OCO2
(SAS: 2,894, LogP: -0,400, CA9: -7,2)

CC(=O)c1ccc(OCc2ccn(NC(N)=O)cc2)cc1
(SAS: 1,627, LogP: 2,959, CA9: -7,2)

Cc1cccc(CC(=O)Nc2c(O)cccc20)c1
(SAS: 1,825, LogP: 2,587, CA9: -7,2)

Cc1ccc(S(=O)(=O)N2CCCCC2)cc1NC(=O)C(=O)CCC1=O
(SAS: 2,130, LogP: 1,257, CA9: -7,2)

O=C(Cc1ccccc2ccccc12)Nc1ccccc1
(SAS: 1,442, LogP: 4,021, CA9: -7,1)

NS(=O)(=O)c1ccc(CNC(=O)CCc2ccc(F)c2)cc1
(SAS: 1,785, LogP: 1,722, CA9: -7,1)

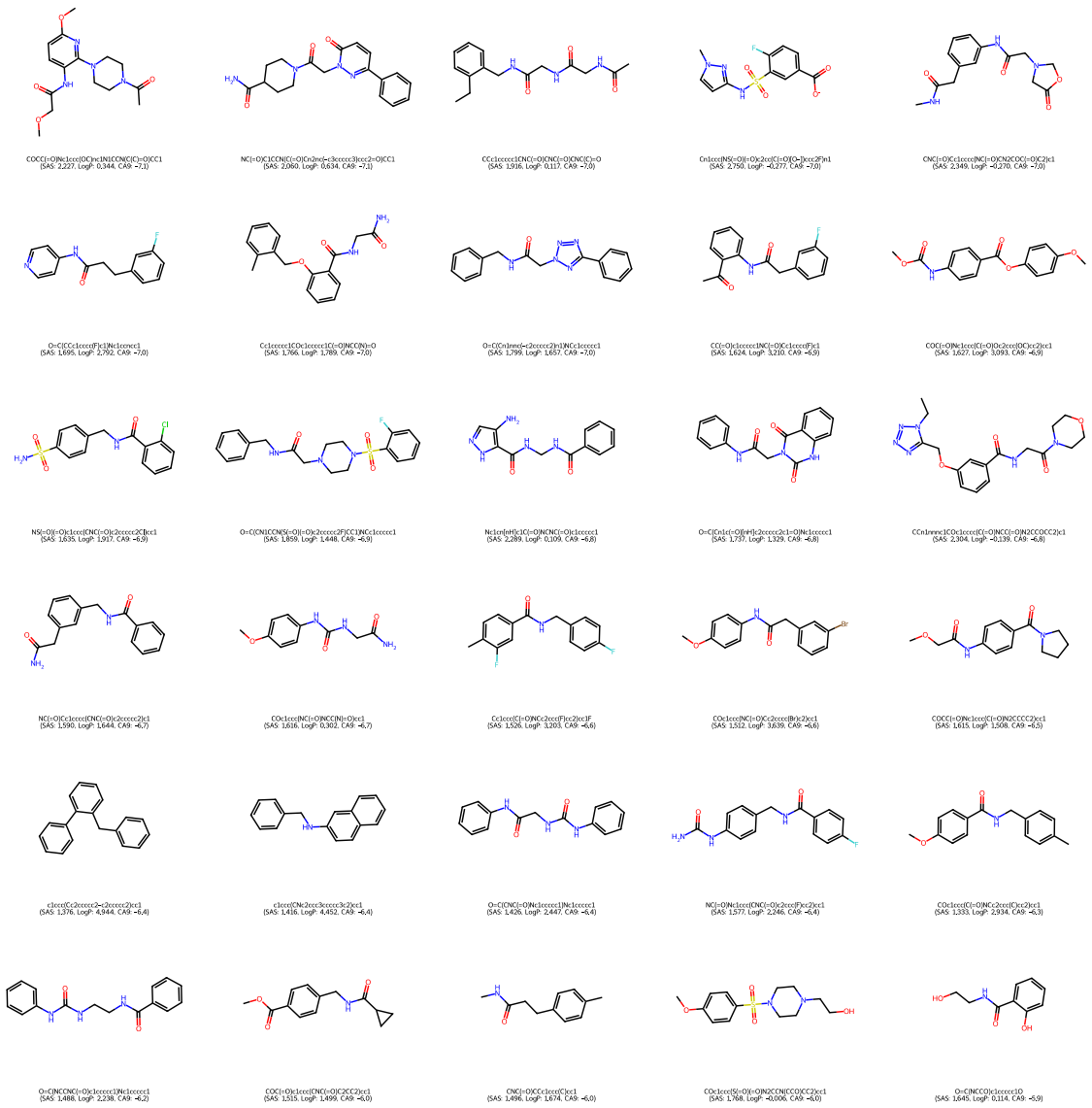
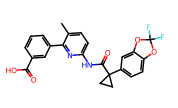
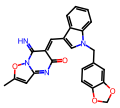
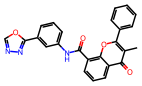


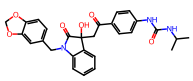
Figure A.2: Full list of the 2D graph examples of high-quality novel samples of the final (10th) population of DEL with objectives $\{SAS, LogP, CA9\}$ in combination with **JTVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with advantageous BAS, the samples are sorted descendingly by the CA9 binding score.

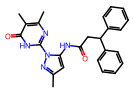
A.2 Double Protein Targets

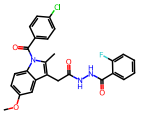


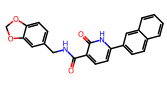


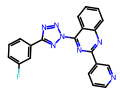


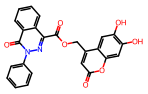


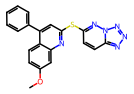


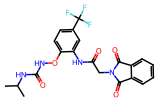


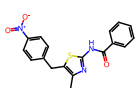


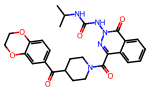


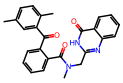


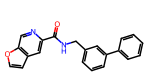


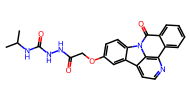


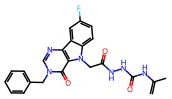


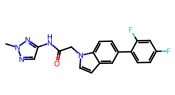


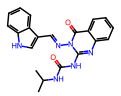


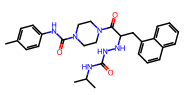


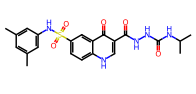


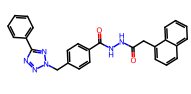


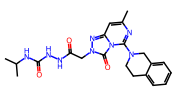


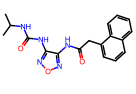


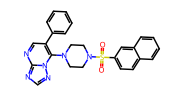


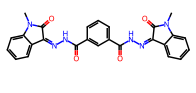


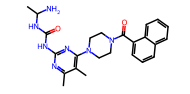


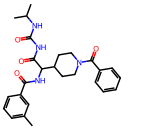


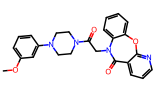


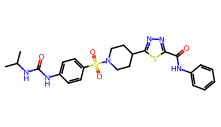


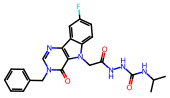


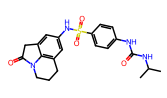


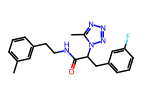


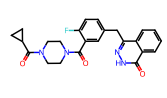


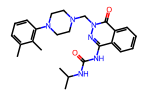


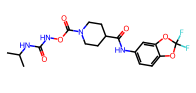


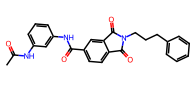


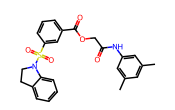


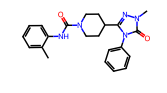


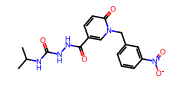


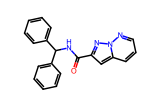


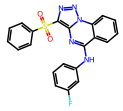


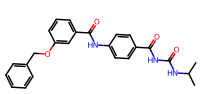


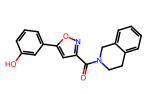


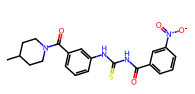


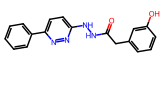

O=C(NC(c1ccccc1)c1ccccc1)c1cc2ccnnn2n1
(SAS: 2,268, LogP: 3,249, CA9: -7,8, GPX4=-7,4)

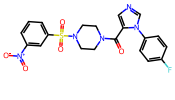

O=S(=O)(c1ccccc1Cl)n1nnn2c1nc(Nc1ccccc(F)c1)c1ccccc12
(SAS: 2,375, LogP: 3,993, CA9: -8,9, GPX4=-6,8)

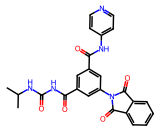

CC(C)NC(=O)NC(=O)c1ccc(NC(=O)c2ccc(OCc3ccccc3)c2)cc1
(SAS: 1,904, LogP: 4,366, CA9: -7,9, GPX4=-7,2)

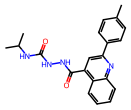

O=C(c1cc(-c2ccc(O)c2)on1)N1CCc2ccccc2C1
(SAS: 2,169, LogP: 3,246, CA9: -8,2, GPX4=-7,0)

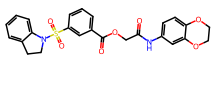

CC1CCN(C(=O)c2ccccNC(=S)NC(=O)c3ccc([N+](=O)[O-])c3)c2)CC1
(SAS: 2,131, LogP: 3,594, CA9: -7,8, GPX4=-8,0)

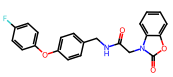

O=C(Cc1ccccc1O)c1NNc1ccc(-c2ccccc2)nn1
(SAS: 2,122, LogP: 2,535, CA9: -7,6, GPX4=-7,6)

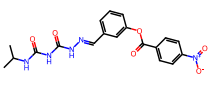

O=C(c1cnccn1-c1ccc(F)cc1)N1CCN(S(=O)(=O)c2ccc([N+](=O)[O-])c2)CC1
(SAS: 2,399, LogP: 2,066, CA9: -7,3, GPX4=-8,1)

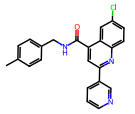

CC(C)NC(=O)NC(=O)c1ccc(C(=O)Nc2ccncc2)cc1N2C(=O)c3ccccc3C2=O)c1
(SAS: 2,381, LogP: 2,982, CA9: -7,5, GPX4=-7,7)

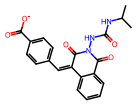

Cc1ccc(-c2cc(C(=O)NNC(=O)NC(C)C)c3ccccc3n2)cc1
(SAS: 2,024, LogP: 3,563, CA9: -7,7, GPX4=-7,4)

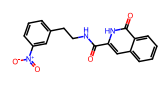

O=C(COC(=O)c1ccccc(S(=O)(=O)N2CCc3ccccc32)c1)Nc1ccc2c(c1)OCCO2
(SAS: 2,300, LogP: 3,005, CA9: -7,7, GPX4=-7,4)

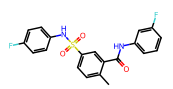

O=C(Cn1c(=O)oc2ccccc21)NCc1ccc(Oc2ccc(F)cc2)cc1
(SAS: 2,002, LogP: 3,842, CA9: -8,3, GPX4=-6,9)

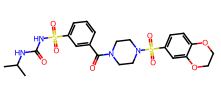

CC(C)NC(=O)NC(=O)NN=Cc1cccc(OC(=O)c2ccc([N+](=O)[O-])cc2)c1
(SAS: 2,399, LogP: 2,565, CA9: -7,4, GPX4=-7,8)

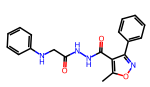

Cc1ccc(CNC(=O)c2cc(-c3ccccn3)nc3ccc(Cl)cc23)cc1
(SAS: 1,978, LogP: 5,189, CA9: -7,6, GPX4=-7,5)

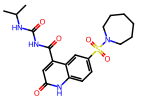

CC(C)NC(=O)NN1C(=O)C(=Cc2ccc(C(=O)[O-])cc2)c2ccccc2C1=O
(SAS: 2,782, LogP: 1,196, CA9: -8,1, GPX4=-7,0)

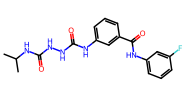

O=C(NCCc1ccccc[N+](=O)[O-])c1)c1cc2ccccc2o[nH]1
(SAS: 2,088, LogP: 2,409, CA9: -7,3, GPX4=-8,0)

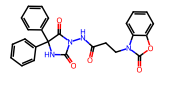

Cc1ccc(S(=O)(=O)Nc2ccc(F)cc2)cc1C(=O)Nc1ccc(F)c1
(SAS: 1,854, LogP: 4,326, CA9: -7,8, GPX4=-7,2)

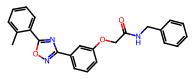

CC(C)NC(=O)NS(=O)(=O)c1ccccc1C(=O)N2CCN(S(=O)(=O)c3ccc4c(c3)OCCO4)CC2)c1
(SAS: 2,492, LogP: 1,001, CA9: -7,5, GPX4=-7,1)

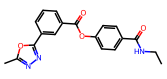

Cc1ccc(-c2ccccc2)c1C(=O)NNC(=O)CNc1ccccc1
(SAS: 2,055, LogP: 2,523, CA9: -7,7, GPX4=-7,3)

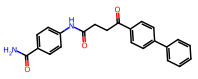

CC(C)NC(=O)NC(=O)c1cc(=O)[nH]c2ccc(S(=O)(=O)N3CCCCC3)cc12
(SAS: 2,380, LogP: 1,941, CA9: -8,4, GPX4=-6,8)

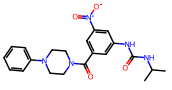

CC(C)NC(=O)NNC(=O)Nc1ccc(C(=O)Nc2ccc(F)c2)c1
(SAS: 2,028, LogP: 2,822, CA9: -8,4, GPX4=-6,8)

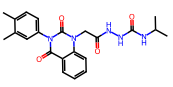

O=C(CCn1c(=O)oc2ccccc21)NN1C(=O)NC(c2ccccc2)(c2ccccc2)C1=O
(SAS: 2,579, LogP: 2,511, CA9: -7,2, GPX4=-8,2)

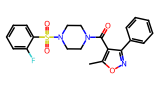

Cc1ccccc1-c1nnc(-c2ccccc2OCC(=O)NCc3ccccc3)o1
(SAS: 1,980, LogP: 4,407, CA9: -7,4, GPX4=-7,7)

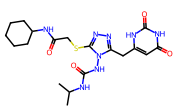

CCNC(=O)c1ccccc1OC(=O)c2ccc(-c3nnn(C)c3)cc2)cc1
(SAS: 2,051, LogP: 3,014, CA9: -8,0, GPX4=-7,0)

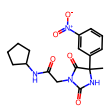

NC(=O)c1ccc(NC(=O)CCC(=O)c2ccc(-c3ccccc3)cc2)cc1
(SAS: 1,729, LogP: 4,054, CA9: -8,2, GPX4=-6,9)

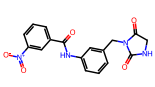

CC(C)NC(=O)Nc1ccc(S(=O)N2CCN(c3ccccc3)CC2)cc[N+](=O)[O-])c1
(SAS: 2,117, LogP: 3,087, CA9: -7,6, GPX4=-7,4)

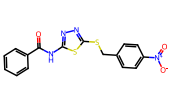

Cc1ccc(-c2c(=O)c3ccccc3n(CC(=O)NNC(=O)NC(C)C)c2=O)cc1C
(SAS: 2,339, LogP: 1,508, CA9: -8,6, GPX4=-6,7)

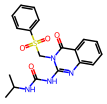

Cc1ccc(-c2cccc2)c1C(=O)N1CCN(S(=O)(=O)c2ccccc2F)CC1
(SAS: 2,170, LogP: 2,936, CA9: -7,8, GPX4=-7,1)

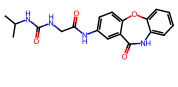

CC(C)NC(=O)Nn1c(Cc2cc(=O)[nH]c(=O)[nH]2)snc1SCC(=O)NC1CCCCC1
(SAS: 2,940, LogP: 0,448, CA9: -7,8, GPX4=-7,1)

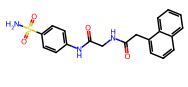

CC1(c2cccc[N+](=O)[O-])c2)NC(=O)NC(=O)C2CCC2)C1=O
(SAS: 2,716, LogP: 1,421, CA9: -7,4, GPX4=-7,6)

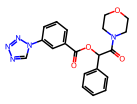

O=C(Nc1cccc(CN2C(=O)CNC2=O)c1)c1ccc([N+](=O)[O-])c1
(SAS: 2,122, LogP: 1,899, CA9: -7,3, GPX4=-7,7)

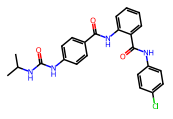

O=C(Nc1nnn(SCc2ccc([N+](=O)[O-])cc2)s1)c1ccccc1
(SAS: 1,969, LogP: 3,991, CA9: -7,3, GPX4=-7,7)

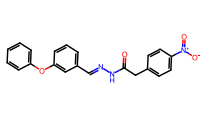

CC(C)NC(=O)Nc1nc2ccccc2c(=O)n1CSc(=O)c(=O)c1ccccc1
(SAS: 2,327, LogP: 2,358, CA9: -7,9, GPX4=-7,0)

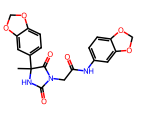

CC(C)NC(=O)NCC(=O)NNc1ccc2c(c1)C(=O)Nc1ccccc1O2
(SAS: 2,265, LogP: 2,691, CA9: -7,9, GPX4=-7,0)

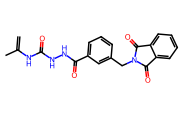

NS(=O)(=O)c1ccc(NC(=O)CNC(=O)Cc2ccc3ccccc23)cc1
(SAS: 1,902, LogP: 1,785, CA9: -7,5, GPX4=-7,4)

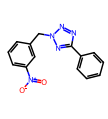

O=C(OC(C(=O)N1CCOCC1)c1ccccc1)c1ccccc1-n1cnnn2)c1
(SAS: 2,762, LogP: 1,419, CA9: -7,5, GPX4=-7,4)


CC(C)NC(=O)Nc1ccc(C(=O)Nc2ccccc2C(=O)Nc2ccc(Cl)cc2)cc1
(SAS: 1,849, LogP: 5,375, CA9: -7,4, GPX4=-7,5)


O=C(Cc1ccc([N+](=O)[O-])cc1)NN=Cc1ccc(Oc2ccccc2)c1
(SAS: 2,006, LogP: 4,080, CA9: -7,6, GPX4=-7,2)


CC1(c2ccc3c(c2)OCO3)NC(=O)NC(C)=O)Nc2ccc3c(c2)OCO3)C1=O
(SAS: 2,808, LogP: 1,550, CA9: -8,2, GPX4=-6,8)


C=C(C)NC(=O)NNC(=O)c1ccccC(CN2C(=O)c3ccccc3C2=O)c1
(SAS: 2,598, LogP: 1,960, CA9: -8,2, GPX4=-6,8)


O=[N+]([O-])c1ccccc1Cn2nnn(-c3ccccc3)n2c1
(SAS: 2,002, LogP: 2,297, CA9: -7,3, GPX4=-7,6)

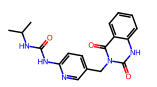
C=C(C)c1ccc(NC(=O)c2cc3ccccc3oc2=O)cc1
(SAS: 1,965, LogP: 4,078, CA9: -7,7, GPX4=-7,1)

CC(C)NC(=O)Nn1cnc2c1c(=O)n(C)c(=O)n2C)c1ccccc1
(SAS: 2,960, LogP: 2,366, CA9: -8,7, GPX4=-6,6)

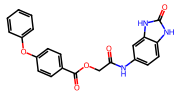
CC(C)NC(=O)Nc1ccc(S(=O)(=O)NNC(=O)c2ccc3c(c2)OCO3)c1
(SAS: 2,243, LogP: 1,568, CA9: -8,0, GPX4=-6,9)

O=C(c1ccccc1)C1CCN(C(=O)CS(=O)(=O)c2ccc([n+]([O-])cc2)N1CC1
(SAS: 2,547, LogP: 1,215, CA9: -7,2, GPX4=-7,7)

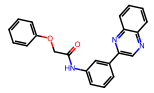
CC(C)NC(=O)NOCC(=O)Nc1ccc(-c2ccccc2)n2nc(=3ccccc3)sc2n1
(SAS: 2,630, LogP: 1,073, CA9: -7,5, GPX4=-7,2)



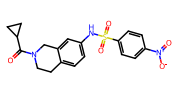
CC[C]NC(=O)Nc1ccc(Cn2c(=O)[nH]c3ccccc3c2=O)cn1
(SAS: 2,195, LogP: 1,663, CA9: -8,4, GPX4=-6,6)



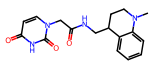
O=C(COC(=O)c1ccc(Oc2ccccc2)cc1)Nc1ccc2[nH]c(=O)[nH]c2c1
(SAS: 2,037, LogP: 3,444, CA9: -7,6, GPX4=-7,1)



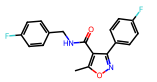
O=C(COc1ccccc1)Nc1cccc(-c2cnc3ccccc3n2)c1
(SAS: 1,827, LogP: 4,314, CA9: -7,6, GPX4=-7,1)



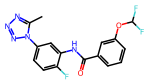
O=C(C1CC1)N1CCc2ccc(NS(=O)(=O)c3ccc([N+](=O)[O-])cc3)cc2C1
(SAS: 2,198, LogP: 2,690, CA9: -7,9, GPX4=-6,9)



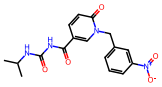
CN1CCC(CNC(=O)Cn2ccc(=O)[nH]c2=O)c2ccccc21
(SAS: 2,929, LogP: 0,277, CA9: -7,9, GPX4=-6,9)



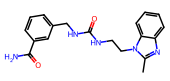
Cc1onc(-c2ccc(F)cc2)c1C(=O)NCc1ccc(F)cc1
(SAS: 1,902, LogP: 3,858, CA9: -7,1, GPX4=-7,7)



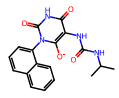
Cc1nnnn1-c1ccc(F)c(NC(=O)c2ccc(OC(F)F)cc2)c1
(SAS: 2,254, LogP: 2,964, CA9: -6,9, GPX4=-8,1)



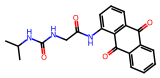
CC(C)NC(=O)NC(=O)c1ccc(-o)n(Cc2ccc([N+](=O)[O-])c2)c1
(SAS: 2,255, LogP: 1,653, CA9: -7,3, GPX4=-7,4)



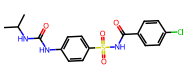
Cc1nc2ccccc2n1CCNC(=O)NCc1ccccc1[N+](=O)[O-]
(SAS: 2,067, LogP: 1,943, CA9: -7,1, GPX4=-7,6)



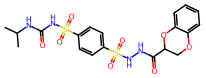
CC(C)NC(=O)Nc1cc[O-]n(-c2ccc3ccccc23)c(=O)[nH]c1=O
(SAS: 2,639, LogP: 1,282, CA9: -8,1, GPX4=-6,7)



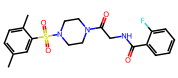
CC(C)NC(=O)NCC(=O)Nc1cccc2c1C(=O)c1ccccc1C2=O
(SAS: 2,105, LogP: 2,108, CA9: -8,1, GPX4=-6,7)



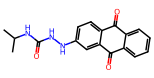
CC(C)NC(=O)Nc1ccc(S(=O)(=O)NC(=O)c2ccc(Cl)cc2)cc1
(SAS: 1,881, LogP: 2,989, CA9: -8,1, GPX4=-6,7)



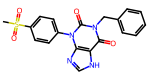
CC(C)NC(=O)NS(=O)(=O)c1ccc(S(=O)(=O)NNC(=O)C2COc3ccccc3O2)cc1
(SAS: 2,981, LogP: 0,232, CA9: -7,5, GPX4=-7,1)



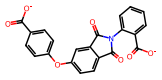
Cc1ccc(C)c(S(=O)(=O)N2CCN(C(=O)CNC(=O)c3ccccc3F)CC2)c1
(SAS: 2,082, LogP: 1,705, CA9: -7,5, GPX4=-7,1)



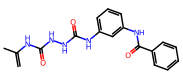
CC(C)NC(=O)NNc1ccc2c(c1)C(=O)c1ccccc1C2=O
(SAS: 2,203, LogP: 2,497, CA9: -7,8, GPX4=-6,9)



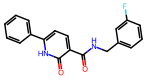
CS(=O)(=O)c1ccn(-c2c(=O)n(Cc3ccccc3)c(=O)c3[nH]cnc32)cc1
(SAS: 2,312, LogP: 1,327, CA9: -7,9, GPX4=-6,8)



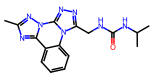
O=C([O-])c1ccc(Oc2ccc3c(c2)C(=O)N(c2ccccc2C(=O)[O-])C3=O)cc1
(SAS: 2,723, LogP: 1,006, CA9: -7,2, GPX4=-7,4)



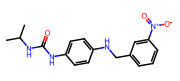
C=C(C)NC(=O)NNC(=O)Nc1ccccc(NC(=O)c2ccccc2)c1
(SAS: 2,187, LogP: 2,808, CA9: -8,9, GPX4=-6,4)



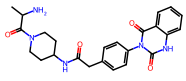
O=C(NCc1ccccc(F)c1)c1ccc(-c2ccccc2)[nH]c1=O
(SAS: 1,886, LogP: 3,111, CA9: -7,1, GPX4=-7,5)



Cc1nc2c3ccccc3n3c(CNC(=O)NC(C)C)nnc3n2n1
(SAS: 2,633, LogP: 1,441, CA9: -8,2, GPX4=-6,6)

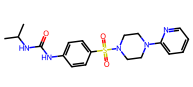


CC(C)NC(=O)Nc1ccc(NCc2cccc([N+](=O)[O-])c2)cc1
(SAS: 1,898, LogP: 3,737, CA9: -7,2, GPX4=-7,3)

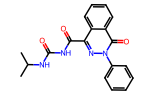


CC(N)C(=O)N1CCCN(C(=O)Cc2ccc(-n3c(=O)[nH]c4ccccc4c3=O)cc2)CC1
(SAS: 2,783, LogP: 0,670, CA9: -7,4, GPX4=-7,1)

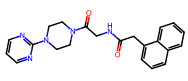
CC(C)NC(=O)NNC(=O)c1ccccc1NC(=O)C(c1ccc(=O)c2ccccc21
(SAS: 2,321, LogP: 1,388, CA9: -8,3, GPX4=-6,5)



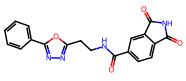
CC(C)NC(=O)NS(=O)(=O)N2CCN(c3ccccn3)CC2)cc1
(SAS: 2,039, LogP: 2,122, CA9: -7,8, GPX4=-6,8)



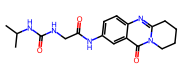
CC(C)NC(=O)NC(=O)c1nn(-c2ccccc2)c(=O)c2ccccc12
(SAS: 2,062, LogP: 2,234, CA9: -8,1, GPX4=-6,6)



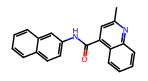
O=C(Cc1ccnc2ccccc12)NCC(=O)N1CCN(c2nccnn2)CC1
(SAS: 2,131, LogP: 1,637, CA9: -7,5, GPX4=-7,0)



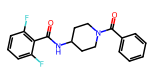
O=C(NCCc1nnc(-c2ccccc2)o1)c1ccc2c(c1)C(=O)NC2=O
(SAS: 2,221, LogP: 1,593, CA9: -7,5, GPX4=-7,0)



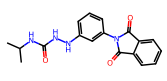
CC(C)NC(=O)NCC(=O)Nc1ccc2nc3nh(c(=O)c2c1)CCCC3
(SAS: 2,284, LogP: 1,379, CA9: -7,9, GPX4=-6,7)



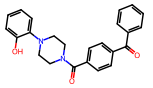
Cc1ccc(C)c(=O)Nc2ccc3ccccc3c2)c2ccccc2n1
(SAS: 1,713, LogP: 4,949, CA9: -7,0, GPX4=-7,5)



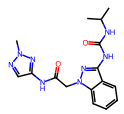
O=C(NC1CCN(C(=O)c2ccccc2)CC1)c1cFccc1F
(SAS: 1,863, LogP: 2,999, CA9: -7,6, GPX4=-6,9)



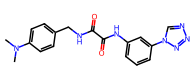
CC(C)NC(=O)NNc1ccccN2C(=O)c3ccccc3C2=O)c1
(SAS: 2,160, LogP: 2,522, CA9: -7,6, GPX4=-6,9)



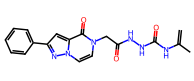
O=C(c1ccccc1)c1ccc(C(=O)N2CCN(c3ccccc3O)CC2)cc1
(SAS: 1,851, LogP: 3,586, CA9: -6,8, GPX4=-7,6)



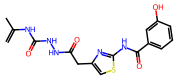
CC(C)NC(=O)Nc1nn[nH]CC(=O)Nc2cnn(C)n2)c2ccccc12
(SAS: 2,646, LogP: 1,333, CA9: -8,3, GPX4=-6,4)



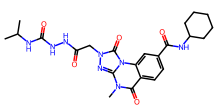
CN(C)c1ccc(CNC(=O)C(=O)Nc2ccccn(=s3nnn3)c2)cc1
(SAS: 2,224, LogP: 0,983, CA9: -7,4, GPX4=-7,0)



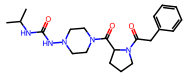
C=C(C)NC(=O)NNC(=O)C(=O)c1ccn2nc(-c3cccc3)cc2c1=O
(SAS: 2,741, LogP: 1,027, CA9: -8,3, GPX4=-6,4)



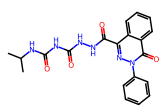
C=C(C)NC(=O)NNC(=O)C(Lcsc(NC(=O)c2ccccO)c2)n1
(SAS: 2,564, LogP: 1,507, CA9: -8,5, GPX4=-6,3)



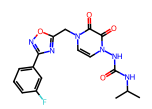
CC(C)NC(=O)NNC(=O)Cn1nc2n(C)c1=O)c3ccc(C(=O)NC4CCCC4)cc3n2c1=O
(SAS: 2,766, LogP: 0,149, CA9: -7,1, GPX4=-7,2)



CC(C)NC(=O)NNS(CCN(C(=O)C2CCCN2C(=O)C(c2ccccc2)CC1
(SAS: 2,803, LogP: 0,987, CA9: -7,9, GPX4=-6,7)



CC(C)NC(=O)NC(=O)NNC(=O)c1nn(-c2ccccc2)c(=O)c2ccccc12
(SAS: 2,319, LogP: 1,448, CA9: -7,2, GPX4=-7,1)



CC(C)NC(=O)Nn1ccn(Cc2nc(-c3cccc(F)c3)no2)c(=O)c1=O
(SAS: 2,767, LogP: 0,909, CA9: -8,4, GPX4=-6,3)

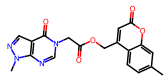
C=C(C)NC(=O)NNCC(=O)Nc1ccccc1C(=O)NCc1ccccc1
(SAS: 2,052, LogP: 2,388, CA9: -7,9, GPX4=-6,6)

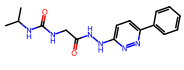
Cc1ccccc(Cn2c(=O)c(C(=O)NNC(=O)NC(C)C)nn(-c3ccccc3)c2=O)c1
(SAS: 2,355, LogP: 1,103, CA9: -7,9, GPX4=-6,6)

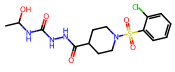
CC(C)NC(=O)NCCC(=O)Nc1ccc(C(=O)NCc2ccccc2)cc1
(SAS: 1,792, LogP: 2,653, CA9: -7,3, GPX4=-7,0)

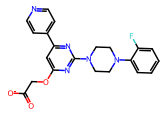
CC(C)NC(=O)NNC(=O)c1ccccc1C(=O)NNc2ccccO)c2)c1
(SAS: 2,004, LogP: 1,997, CA9: -7,6, GPX4=-6,8)

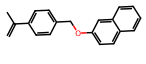
CC(C)NC(=O)NCC(=O)Oc1ccccN2C(=O)c3ccccc3C2=O)c1
(SAS: 2,180, LogP: 2,100, CA9: -8,2, GPX4=-6,4)

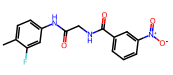

Cc1ccc2c(C(OC[=O)Cn3ncn4c(nnn4C)c3=O)ccl=O)oc2c1
(SAS: 2,508, LogP: 1,288, CA9: −6,8, GPX4=−7,6)

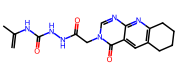

CC(C)NC(=O)NCC(=O)NNc1ccnc(−c2ccccc2)nn1
(SAS: 2,230, LogP: 1,294, CA9: −8,0, GPX4=−6,5)

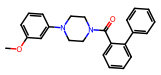

CC(O)NC(=O)NNC(=O)C1CCN(S(=O)(=O)c2ccccc2Cl)CC1
(SAS: 2,805, LogP: 0,409, CA9: −7,7, GPX4=−6,7)

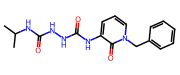

O=C([O−])COc1cc(−c2ccnc2)nn(N2CCN(c3cccc3F)CC2)n1
(SAS: 2,664, LogP: 1,133, CA9: −6,8, GPX4=−7,4)

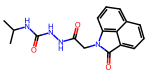

C=C(C)c1ccc(COc2ccc3ccccc3c2)cc1
(SAS: 1,738, LogP: 5,452, CA9: −7,1, GPX4=−7,1)

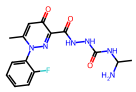

Cc1ccc(NC(=O)CNC(=O)c2ccc([N+](=O)[O−])c2)cc1F
(SAS: 1,832, LogP: 2,411, CA9: −7,1, GPX4=−7,1)

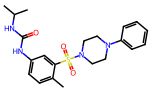

C=C(C)NC(=O)NNC(=O)Cn1cnc2nn3c(cc2c1=O)CCCC3
(SAS: 2,917, LogP: 0,534, CA9: −8,3, GPX4=−6,3)

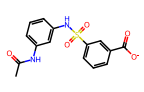

COc1cccc(N2CCN(C(=O)c3ccccc3−c3cccc3)CC2)c1
(SAS: 1,777, LogP: 4,325, CA9: −7,2, GPX4=−7,0)

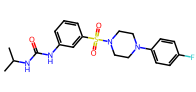

CC(C)NC(=O)NNC(=O)Nc1ccccc1C(=O)Nc1ccccc2c1c1=O
(SAS: 2,201, LogP: 1,641, CA9: −7,6, GPX4=−6,7)

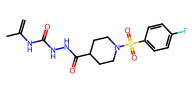

CC(C)NC(=O)NNC(=O)CN1C(=O)c2ccc3cccc1c23
(SAS: 2,231, LogP: 1,539, CA9: −7,6, GPX4=−6,7)

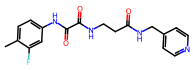

Cc1ccc(=O)c(C(=O)NNC(=O)NC(C)N)nn1−c1ccccc1F
(SAS: 2,994, LogP: −0,071, CA9: −8,2, GPX4=−6,3)

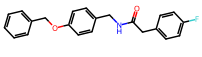

Cc1ccc(NC(=O)NC(C)C)cc1S(=O)(=O)N1CCN(c2ccccc2)CC1
(SAS: 2,022, LogP: 3,036, CA9: −8,2, GPX4=−6,3)

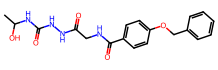

CC(=O)Nc1ccccc1NS(=O)(=O)c2ccc(C(=O)[O−])c2)c1
(SAS: 2,237, LogP: 0,809, CA9: −7,0, GPX4=−7,1)

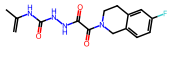

CC(C)NC(=O)Nc1ccccc1S(=O)(=O)N2CCN(c3ccc(F)cc3)CC2)c1
(SAS: 2,016, LogP: 2,867, CA9: −8,0, GPX4=−6,4)

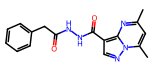

C=C(C)NC(=O)NNC(=O)C1CCN(S(=O)(=O)c2ccc(F)cc2)CC1
(SAS: 2,413, LogP: 1,090, CA9: −8,0, GPX4=−6,4)

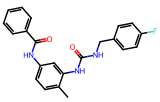

Cc1ccc(NC(=O)C(C)CCC(=O)NCc2ccccc2)cc1F
(SAS: 2,039, LogP: 1,290, CA9: −6,8, GPX4=−7,2)

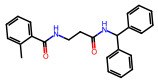

O=C(Cc1ccc(F)cc1)NCc1ccc(OCc2ccccc2)cc1
(SAS: 1,604, LogP: 4,264, CA9: −7,1, GPX4=−7,0)

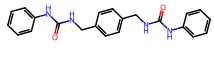

CC(O)NC(=O)NNC(=O)CNC(=O)c1ccc(OCc2ccccc2)cc1
(SAS: 2,525, LogP: 0,664, CA9: −7,1, GPX4=−7,0)

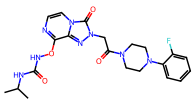

C=C(C)NC(=O)NNCn1=O)C(=O)N1CCc2cc(F)ccc2C1
(SAS: 2,745, LogP: 0,574, CA9: −7,8, GPX4=−6,5)

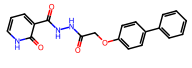

Cc1cc(C)c2ncc(C(=O)NNC(=O)Cc3ccccc3)c2n1
(SAS: 2,145, LogP: 1,350, CA9: −7,2, GPX4=−6,9)

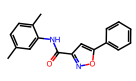

Cc1ccc(NC(=O)c2ccccc2)cc1NC(=O)NCc1ccc(F)cc1
(SAS: 1,706, LogP: 4,708, CA9: −7,2, GPX4=−6,9)

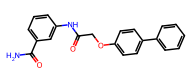

Cc1ccccc1C(=O)NCCC(=O)NC(c1ccccc1)c1ccccc1
(SAS: 1,789, LogP: 4,021, CA9: −6,6, GPX4=−7,2)

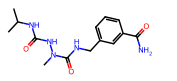

O=C(NC(c1ccc(CNC(=O)Nc2ccccc2)cc1)Nc1ccccc1
(SAS: 1,596, LogP: 4,330, CA9: −7,0, GPX4=−7,0)

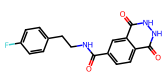

CC(C)NC(=O)NNOc1nccn2c(=O)n(CC(=O)N3CCN(c4ccccc4F)CC3)nc12
(SAS: 2,875, LogP: 0,380, CA9: −7,7, GPX4=−6,5)

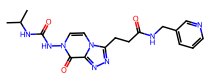

O=C(CCOc1ccc(−c2ccccc2)cc1)NNC(=O)c1ccc[nH]c1=O
(SAS: 1,986, LogP: 1,882, CA9: −7,7, GPX4=−6,5)

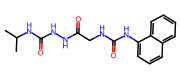

Cc1ccc(C)n(NC(=O)c2cc(−c3ccccc3)on2)c1
(SAS: 1,806, LogP: 4,211, CA9: −6,7, GPX4=−7,1)

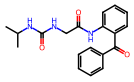

NC(=O)c1ccccc1NC(=O)COc2ccc(−c3ccccc3)cc2)c1
(SAS: 1,634, LogP: 3,470, CA9: −7,1, GPX4=−6,9)

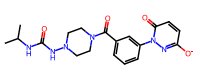

CC(C)NC(=O)NN1C(=O)NC(=O)C1ccccc1C(N)=O)c1
(SAS: 2,408, LogP: 0,550, CA9: −7,2, GPX4=−6,8)

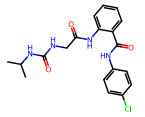

O=C(NCCc1ccc(F)cc1)c1ccc2c(=O)[nH]c(=O)c2c1
(SAS: 2,120, LogP: 1,328, CA9: −6,9, GPX4=−7,0)

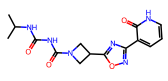

CC(C)NC(=O)Nn1ccnc2c(CCC(=O)NCc3ccccc3)nnc2c1=O
(SAS: 2,815, LogP: 0,197, CA9: −7,9, GPX4=−6,3)

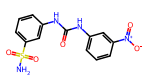

CC(C)NC(=O)NNC(=O)NC(=O)Nc1ccc2ccccc12
(SAS: 2,063, LogP: 1,700, CA9: −7,3, GPX4=−6,7)

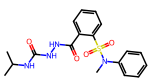

CC(C)NC(=O)NCC(=O)Nc1ccccc1C(=O)c1ccccc1
(SAS: 1,821, LogP: 2,564, CA9: −7,0, GPX4=−6,9)

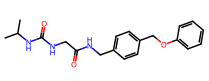

CC(C)NC(=O)NN1CCN(C(=O)c2ccc(−n3nc[O−])ccc3=O)c2)CC1
(SAS: 2,881, LogP: −0,313, CA9: −7,0, GPX4=−6,9)

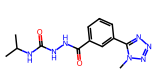

CC(C)NC(=O)NCC(=O)Nc1ccccc1C(=O)Nc1ccc(Cl)cc1
(SAS: 1,830, LogP: 3,238, CA9: −7,4, GPX4=−6,6)

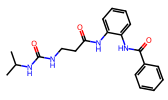

CC(C)NC(=O)NC(=O)N1CC(c2nc[−c3ccc[nH]c3=O)no2)C1
(SAS: 2,797, LogP: 0,652, CA9: −7,4, GPX4=−6,6)


NS(=O)(=O)c1ccccc(NC(=O)Nc2ccc([N+](=O)[O−])c2)c1
(SAS: 1,897, LogP: 1,886, CA9: −6,4, GPX4=−7,1)


CC(C)NC(=O)NNC(=O)c1ccccc1S(=O)(=O)N(C)c1ccccc1
(SAS: 2,189, LogP: 1,864, CA9: −7,8, GPX4=−6,3)


CC(C)NC(=O)NCC(=O)NCc1ccc(COc2ccccc2)cc1
(SAS: 1,812, LogP: 2,589, CA9: −7,2, GPX4=−6,7)


CC(C)NC(=O)NNC(=O)Nc1ccccc1−c2nnnn2)C)c1
(SAS: 2,283, LogP: 0,229, CA9: −7,5, GPX4=−6,5)


CC(C)NC(=O)NCCC(=O)Nc1ccccc1NC(=O)c1ccccc1
(SAS: 1,823, LogP: 2,975, CA9: −7,5, GPX4=−6,5)

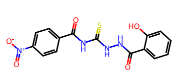
O=C([O−])Cn1nnnc1SCc1ccc([N+](=O)[O−])c(F)c1
(SAS: 2,875, LogP: −0,237, CA9: −8,6, GPX4=−7,0)

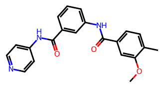
CC(C)NC(=O)NNC(=O)C1CCN(S(=O)(=O)c2ccc3c(c2)CCC(=O)N3)CC1
(SAS: 2,498, LogP: 0,711, CA9: −7,6, GPX4=−6,4)

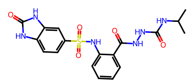
C=C(C)NC(=O)NS(=O)(=O)c1ccc2[nH]c(=O)c(=O)[nH]c2c1
(SAS: 2,711, LogP: −0,262, CA9: −7,6, GPX4=−6,6)

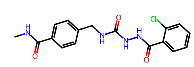
Cc1ccc(NC(=O)C(=O)NCC2CCN(c3nccn3)CC2)c1
(SAS: 2,161, LogP: 1,756, CA9: −7,6, GPX4=−6,4)

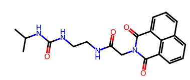
CC(C)NC(=O)NCC(=O)NNC(=O)c1ccc2ccccc2oc1=O
(SAS: 2,199, LogP: 0,262, CA9: −7,3, GPX4=−6,6)

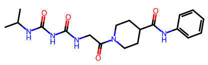

O=C(NC(=S)NNC(=O)c1ccccc1O)c1ccc([N+](=O)[O-])cc1
(SAS: 2.047, LogP: 1.250, CA9: -7.0, GPX4:-6.8)

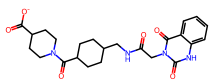

COc1cc(C(=O)Nc2cccc(C(=O)Nc3ccncc3)c2)ccc1C
(SAS: 1.808, LogP: 3.903, CA9: -7.7, GPX4:-6.3)

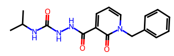

CC(C)NC(=O)NNC(=O)c1ccccc1NS(=O)(=O)c1ccc2[nH]c(=O)[nH]c2c1
(SAS: 2.416, LogP: 1.009, CA9: -7.7, GPX4:-6.3)

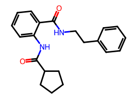

CNC(=O)c1ccc(CNC(=O)NNC(=O)c2ccccc2Cl)cc1
(SAS: 1.880, LogP: 1.844, CA9: -6.8, GPX4:-6.9)

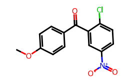

CC(C)NC(=O)NCCNC(=O)CN1C(=O)c2cccc3cccc(c23)C1=O
(SAS: 2.203, LogP: 1.260, CA9: -7.1, GPX4:-6.7)

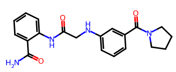

CC(C)NC(=O)NC(=O)NCC(=O)N1CCC(C(=O)Nc2ccccc2)CC1
(SAS: 2.114, LogP: 1.281, CA9: -7.4, GPX4:-6.5)

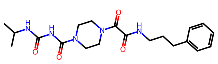

O=C(Cn1c(=O)[nH]c2ccccc2c1=O)NCC1CCC(C(=O)N2CCC(C(=O)[O-])CC2)CC1
(SAS: 2.716, LogP: -0.399, CA9: -6.7, GPX4:-6.9)

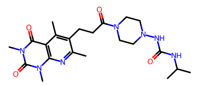

CC(C)NC(=O)NNC(=O)c1cccn(Cc2ccccc2)c1=O
(SAS: 2.076, LogP: 1.249, CA9: -7.2, GPX4:-6.6)

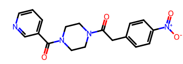

O=C(NCCc1ccccc1)c1ccccc1NC(=O)C1CCCC1
(SAS: 1.673, LogP: 3.788, CA9: -7.2, GPX4:-6.6)

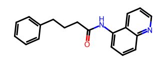

COc1ccc(C(=O)c2cc([N+](=O)[O-])ccc2Cl)cc1
(SAS: 1.727, LogP: 3.488, CA9: -6.3, GPX4:-7.0)

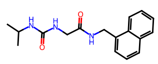

NC(=O)c1ccccc1NC(=O)CNc1cccc(C(=O)N2CCCC2)c1
(SAS: 1.901, LogP: 2.072, CA9: -7.0, GPX4:-6.7)

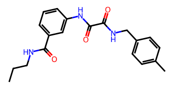

CC(C)NC(=O)NC(=O)N1CCN(C(=O)C(=O)NCCCc2ccccc2)CC1
(SAS: 2.292, LogP: 0.707, CA9: -7.0, GPX4:-6.7)

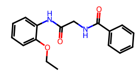

Cc1nc2c(c(C)c1CCC(=O)N1CCN(NC(=O)NC(C)C)CC1)c(=O)n(C)c(=O)n2C
(SAS: 2.810, LogP: -0.052, CA9: -6.2, GPX4:-7.0)

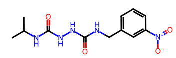

O=C(Cc1ccc([N+](=O)[O-])cc1)N1CCN(C(=O)c2cccnc2)CC1
(SAS: 1.987, LogP: 1.517, CA9: -6.8, GPX4:-6.8)

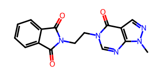

O=C(CCCc1ccccc1)Nc1cccc2ncccc12
(SAS: 1.643, LogP: 4.196, CA9: -6.0, GPX4:-7.0)

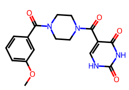

CC(C)NC(=O)NCC(=O)NCc1cccc2ccccc12
(SAS: 1.849, LogP: 2.164, CA9: -7.4, GPX4:-6.4)

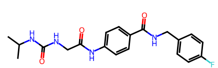

CCCNC(=O)c1cccc(NC(=O)C(=O)NCc2ccc(C)cc2)c1
(SAS: 1.794, LogP: 2.390, CA9: -7.1, GPX4:-6.6)

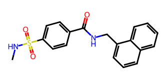

CCOc1ccccc1NC(=O)CNC(=O)c1ccccc1
(SAS: 1.526, LogP: 2.454, CA9: -6.4, GPX4:-6.9)

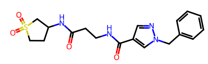

CC(C)NC(=O)NNC(=O)NCc1cccc([N+](=O)[O-])c1
(SAS: 2.163, LogP: 1.017, CA9: -6.9, GPX4:-6.7)

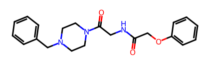

Cn1ncc2c(=O)n(CCN3C(=O)c4ccccc4C3=O)cnc21
(SAS: 2.279, LogP: 0.426, CA9: -6.9, GPX4:-6.7)

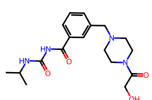

COc1cccc(C(=O)N2CCN(C(=O)c3c[nH]c(=O)[nH]c3=O)CC2)c1
(SAS: 2.139, LogP: -0.330, CA9: -6.3, GPX4:-6.9)

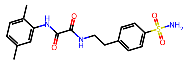

CC(C)NC(=O)NCC(=O)Nc1ccc(C(=O)NCc2ccc(F)cc2)cc1
(SAS: 1.849, LogP: 2.402, CA9: -6.3, GPX4:-6.9)

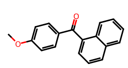

CNS(=O)(=O)c1ccc(C(=O)NCc2cccc3ccccc23)cc1
(SAS: 1.787, LogP: 2.678, CA9: -7.0, GPX4:-6.6)

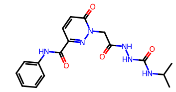

O=C(CCNC(=O)c1cnn(Cc2ccccc2)c1)NC1CCS(=O)(=O)C1
(SAS: 2.709, LogP: 0.355, CA9: -6.5, GPX4:-6.8)

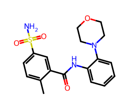

O=C(COc1ccccc1)NCC(=O)N1CCN(Cc2ccccc2)CC1
(SAS: 1.770, LogP: 1.526, CA9: -6.5, GPX4:-6.8)

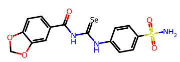

CC(C)NC(=O)NC(=O)c1cccc(CN2CCN(C(=O)CO)CC2)c1
(SAS: 2.121, LogP: 0.171, CA9: -6.7, GPX4:-6.7)

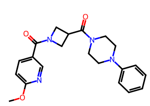

Cc1ccc(C)c(NC(=O)C(=O)NCCc2ccc(S(N)(=O)=O)cc2)c1
(SAS: 1.954, LogP: 1.248, CA9: -6.7, GPX4:-6.7)

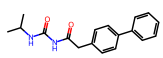

COc1ccc(C(=O)c2cccc3ccccc23)cc1
(SAS: 1.491, LogP: 4.079, CA9: -7.1, GPX4:-6.5)

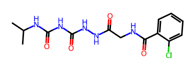

CC(C)NC(=O)NNC(=O)Cn1nc(C(=O)Nc2ccccc2)ccc1=O
(SAS: 2.189, LogP: 0.234, CA9: -7.2, GPX4:-6.4)

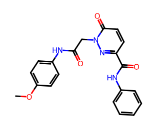

Cc1ccc(S(N)(=O)=O)cc1C(=O)Nc1ccccc1N1CCOCC1
(SAS: 2.004, LogP: 1.731, CA9: -6.9, GPX4:-6.6)

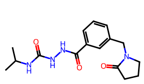

NS(=O)(=O)c1ccc(NC(=[Se])NC(=O)c2ccc3c(c2)OCO3)cc1
(SAS: 2.519, LogP: 0.160, CA9: -6.6, GPX4:-6.7)

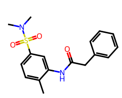

COc1ccc(C(=O)N2CC(C(=O)N3CCN(c4ccccc4)CC3)C2)cn1
(SAS: 2.111, LogP: 1.511, CA9: -7.3, GPX4:-6.3)

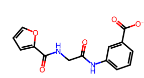

CC(C)NC(=O)NC(=O)Cc1ccc(-c2ccccc2)cc1
(SAS: 1.704, LogP: 3.130, CA9: -6.8, GPX4:-6.6)

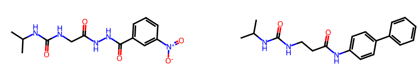

CC(C)NC(=O)NC(=O)NNC(=O)CNC(=O)c1ccccc1Cl
(SAS: 2.173, LogP: 0.518, CA9: -6.8, GPX4:-6.6)


COc1ccc(NC(=O)Cn2nc(C(=O)Nc3ccccc3)ccc2=O)cc1
(SAS: 1.853, LogP: 2.143, CA9: -7.0, GPX4:-6.5)

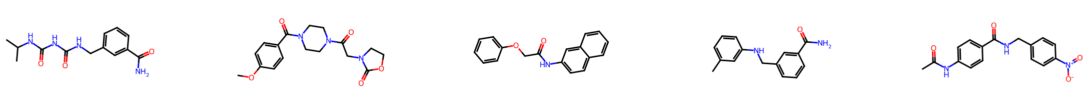

CC(C)NC(=O)NNC(=O)c1cccc(CN2CCCC2=O)c1
(SAS: 2.131, LogP: 1.161, CA9: -7.0, GPX4:-6.5)


Cc1ccc(S(=O)(=O)N(C)C)cc1NC(=O)Cc1ccccc1
(SAS: 1.729, LogP: 2.427, CA9: -7.0, GPX4:-6.5)


O=C(CNC(=O)c1ccco1)Nc1cccc(C(=O)[O-])c1
(SAS: 2.293, LogP: 0.012, CA9: -7.0, GPX4:-6.5)


CC(C)NC(=O)NCC(=O)NNC(=O)c1cccc([N+](=O)[O-])c1
(SAS: 2.075, LogP: 0.063, CA9: -7.2, GPX4:-6.3)

CC(C)NC(=O)NCCC(=O)Nc1ccc(-c2ccccc2)cc1
(SAS: 1.689, LogP: 3.390, CA9: -7.2, GPX4:-6.3)


CC(C)NC(=O)NC(=O)NCc1cccc(C(N)=O)c1
(SAS: 1.988, LogP: 0.703, CA9: -6.9, GPX4:-6.5)

COc1ccc(C(=O)N2CCN(C(=O)CN3CCOC3=O)CC2)cc1
(SAS: 2.070, LogP: 0.432, CA9: -6.2, GPX4:-6.7)

O=C(COc1ccccc1)Nc1ccc2ccccc2c1
(SAS: 1.461, LogP: 3.857, CA9: -6.5, GPX4:-6.6)

Cc1cccc(NCc2cccc(C(N)=O)c2)c1
(SAS: 1.628, LogP: 2.706, CA9: -6.5, GPX4:-6.6)

CC(=O)Nc1ccc(C(=O)NCc2ccc([N+](=O)[O-])cc2)cc1
(SAS: 1.659, LogP: 2.483, CA9: -6.9, GPX4:-6.4)

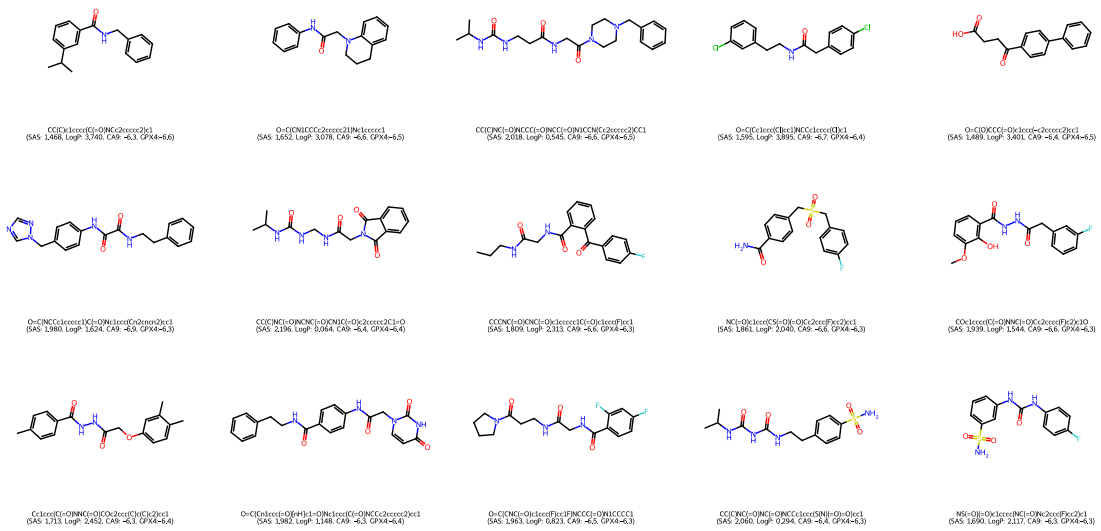
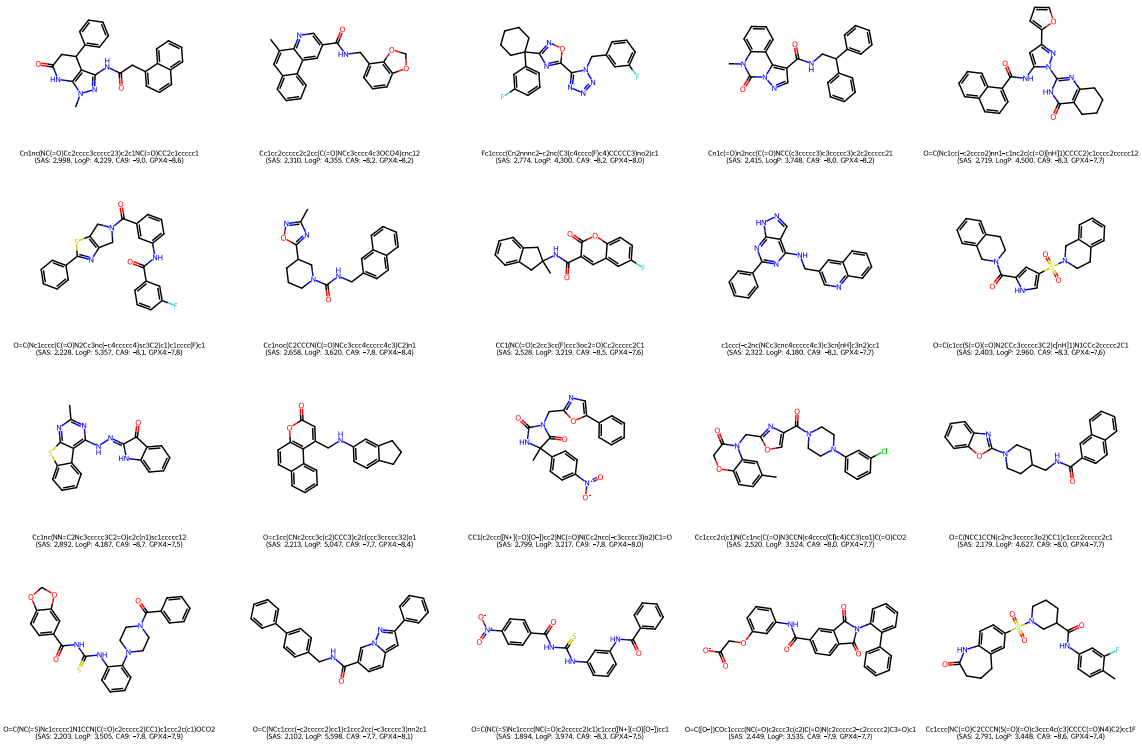
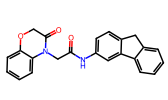
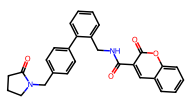
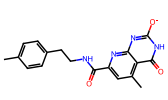


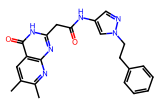
Figure A.3: Full list of the 2D graph visualization of high-quality novel samples in the final (10th) population of DEL with objectives $\{SAS, LogP, CA9, GPX4\}$ in combination with **FragVAE**, trained on the ZINC+DrugBank data. To prioritize molecules with favorable BAS, the results are ranked based on CA9 and GPX4 binding scores respectively, then sorted by the summation of both ranks.

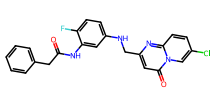


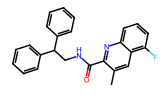

O=C(CN1C(=O)COc2ccccc21)Nc1ccc2c(c1)-c1ccccc1C2
(SAS: 2,170, LogP: 3,622, CA9: -7,6, GPX4=-8,2)

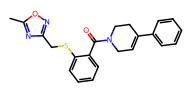

O=C(NCc1ccccc1-c1ccc(ON2CCCC2=O)cc1)c1cc2ccccc2oc1=O
(SAS: 2,241, LogP: 4,512, CA9: -8,4, GPX4=-7,4)

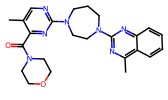

Cc1ccc(CCNC(=O)c2cc(C)c3c(=O)[nH]n([O-])nc3n2)cc1
(SAS: 2,761, LogP: 0,981, CA9: -7,8, GPX4=-7,7)

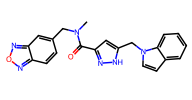

Cc1cc2c(=O)[nH]c(CC(=O)Nc3nnc(CC(=O)Nc4ccccc4)s3)n2nc1C
(SAS: 2,607, LogP: 2,553, CA9: -7,8, GPX4=-7,7)

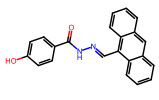

O=C(Cc1ccccc1)Nc1cc(NC(=O)c2cc(=O)n3ccc(Cl)ccc3n2)ccc1F
(SAS: 2,262, LogP: 4,280, CA9: -7,9, GPX4=-7,6)

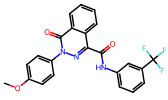

Cc1cc2c(F)cccc2nc1C(=O)NCC(c1ccccc1)c1ccccc1
(SAS: 2,159, LogP: 5,244, CA9: -7,5, GPX4=-8,2)

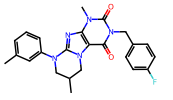

Cc1nc(CSc2cccc2C(=O)N2CC=C(c3ccccc3)CC2)no1
(SAS: 2,432, LogP: 4,600, CA9: -8,5, GPX4=-7,3)

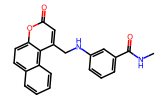

Cc1ccn(N2CCCN(c3nc(C)c4ccccc4n3)CC2)nc1C(=O)N1CCOCC1
(SAS: 2,572, LogP: 2,226, CA9: -7,6, GPX4=-7,8)

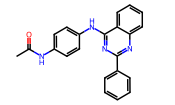

CN(C1ccc2nsnc2c1)C(=O)c1cc(C)n2ccc3ccccc32)[nH]n1
(SAS: 2,709, LogP: 3,221, CA9: -7,5, GPX4=-8,0)

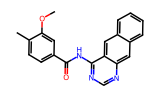

O=C(NN=Cc1c2ccccc2nc2ccccc12)c1ccc(O)cc1
(SAS: 1,994, LogP: 4,463, CA9: -7,4, GPX4=-8,4)

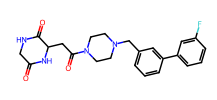

COc1ccc(-n2nc(C(=O)Nc3cccc(C(F)(F)F)c3)c3ccccc3c2=O)cc1
(SAS: 2,088, LogP: 4,665, CA9: -7,8, GPX4=-7,6)

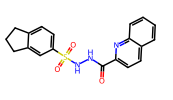

Cc1cccc(N2CC(C)Cn3c2nc2c3c(=O)n(Cc3ccc(F)cc3)c(=O)n2C)c1
(SAS: 2,959, LogP: 3,180, CA9: -7,5, GPX4=-7,9)

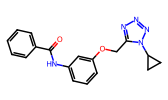

CNC(=O)c1ccccc(NCc2cc(=O)oc3ccc4ccccc4c23)c1
(SAS: 2,144, LogP: 3,918, CA9: -8,7, GPX4=-7,2)

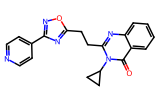

CC(=O)Nc1ccc(Nc2nc(-c3ccccc3)nc3ccccc23)cc1
(SAS: 1,832, LogP: 4,999, CA9: -8,0, GPX4=-7,4)

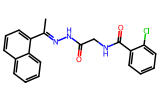

COc1cc(C(=O)Nc2ncnc3cc4ccccc4cc23)ccc1C
(SAS: 2,035, LogP: 4,352, CA9: -7,7, GPX4=-7,6)

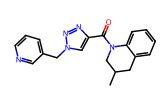

O=C1CN(C[n+]2ccn(C(=O)N2CCN(Cc3ccccc3-c4cccc(F)c4)c3)CC2)N1
(SAS: 2,806, LogP: 1,142, CA9: -8,4, GPX4=-7,2)

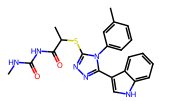

O=C(NNS(=O)(=O)c1ccc2c(c1)CCC2)c1ccc2ccccc2n1
(SAS: 2,193, LogP: 2,347, CA9: -8,1, GPX4=-7,3)

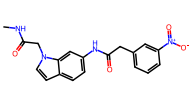

O=C(Nc1cccc(OCc2nnnn2C2CC2)c1)c1ccccc1
(SAS: 2,017, LogP: 2,839, CA9: -7,8, GPX4=-7,5)

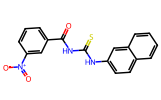

O=c1c2ccccc2nc(CCc2nc(-c3ccnc3)no2)n1C1CC1
(SAS: 2,354, LogP: 2,962, CA9: -8,2, GPX4=-7,2)

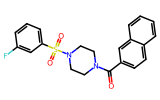

CC(=NNC(=O)CNC(=O)c1ccccc1Cl)c1ccc2ccccc12
(SAS: 1,991, LogP: 3,763, CA9: -8,2, GPX4=-7,2)

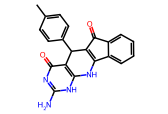

CC1Cc2ccccc2N(C(=O)c2cn(Cc3ccnc3)nn2)C1
(SAS: 2,840, LogP: 2,560, CA9: -7,3, GPX4=-8,0)

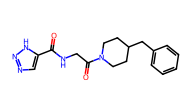

CNC(=O)NC(=O)C(C)Sc1nnc(-c2c[nH]c3ccccc23)n1-c1cccc(C)c1
(SAS: 2,895, LogP: 3,660, CA9: -8,7, GPX4=-7,1)

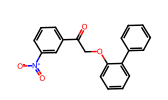

CNC(=O)Cn1ccc2ccc(NC(=O)Cc3ccc([N+](=O)[O-])cc3)cc21
(SAS: 2,232, LogP: 2,477, CA9: -7,2, GPX4=-8,4)

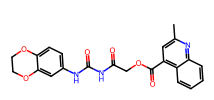

O=C(NC(=S)Nc1ccc2ccccc2c1)c1ccc([N+](=O)[O-])c1
(SAS: 1,898, LogP: 3,875, CA9: -7,7, GPX4=-7,5)

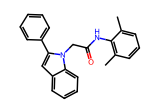

O=C(c1ccc2ccccc2c1)N1CCN(S(=O)(=O)c2ccc(F)c2)CC1
(SAS: 1,907, LogP: 3,126, CA9: -8,1, GPX4=-7,2)

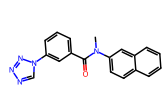

Cc1ccc(C2C3=C(Nc4[nH]c(N)nc(=O)c42)c2ccccc2C3=O)cc1
(SAS: 2,971, LogP: 2,826, CA9: -9,1, GPX4=-7,0)

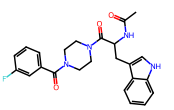

O=C(NCC(=O)N1CCC(Cc2ccccc2)CC1)c1cnn[nH]1
(SAS: 2,570, LogP: 1,016, CA9: -8,4, GPX4=-7,1)

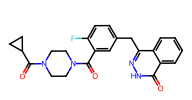

O=C(COc1ccccc1-c1ccccc1)c1ccc([N+](=O)[O-])c1
(SAS: 1,782, LogP: 4,524, CA9: -7,4, GPX4=-7,7)

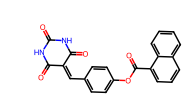

Cc1ccc(C(=O)OCC(=O)NC(=O)Nc2ccc3c(c2)OCCO3)c2ccccc2n1
(SAS: 2,214, LogP: 2,820, CA9: -7,4, GPX4=-7,7)

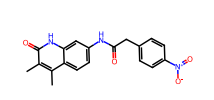

Cc1ccccc(C)c1NC(=O)Cn1nc(-c2ccccc2Cl)cc2ccccc21
(SAS: 1,923, LogP: 5,564, CA9: -7,3, GPX4=-7,8)

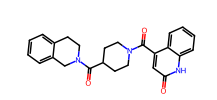

CN(C(=O)c1ccccc1-n2nnn2)c1c1ccc2ccccc2c1
(SAS: 2,175, LogP: 3,092, CA9: -7,3, GPX4=-7,8)

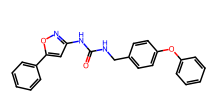

CC(=O)NC(Cc1c[nH]c2ccccc12)C(=O)N1CCN(C(=O)c2ccccF)c2)CC1
(SAS: 2,620, LogP: 2,539, CA9: -8,0, GPX4=-7,2)

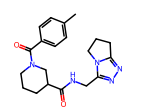

O=C(c1ccc(Cc2n[nH]c(=O)c3ccccc23)ccc1F)N1CCN(C(=O)C2CC2)CC1
(SAS: 2,369, LogP: 2,347, CA9: -8,3, GPX4=-7,1)

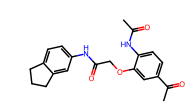

O=C1NC(=O)C(=Cc2ccc(OC(=O)c3ccccc4ccccc34)cc2)C(=O)N1
(SAS: 2,236, LogP: 2,808, CA9: -7,7, GPX4=-7,4)

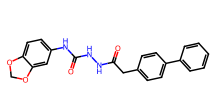

Cc1c(C)c2ccc(NC(=O)Cc3ccc([N+](=O)[O-])cc3)cc2[nH]c1=O
(SAS: 2,135, LogP: 3,234, CA9: -7,3, GPX4=-7,7)


O=C(c1ccc(=O)[nH]c2ccccc12)N1CCC(C(=O)N2CCc3ccccc3C2)CC1
(SAS: 2,237, LogP: 2,965, CA9: -8,3, GPX4=-6,9)


O=C(NCc1ccc(Oc2ccccc2)cc1)Nc1ccc(-c2ccccc2)on1
(SAS: 1,970, LogP: 5,456, CA9: -7,2, GPX4=-7,8)


Cc1ccc(C(=O)N2CCCC(C(=O)NCc3nnc4n3CCC4)C2)cc1
(SAS: 2,670, LogP: 1,701, CA9: -7,4, GPX4=-7,6)


CC(=O)Nc1ccc(C(C)=O)cc1OCC(=O)Nc1ccc2o[c1)CCC2
(SAS: 2,033, LogP: 3,354, CA9: -7,4, GPX4=-7,6)


O=C(Cc1ccc(-c2ccccc2)cc1)NNC(=O)Nc1ccc2o[c1)OCO2
(SAS: 2,008, LogP: 3,478, CA9: -7,6, GPX4=-7,4)

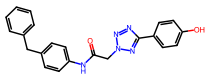
Cn1ncc2c(N3CCN(C(=O)CCCn4c(=O)occ5ccccc54)CC3)nccc21
(SAS: 2,524, LogP: 1,400, CA9: -7,6, GPX4=-7,4)

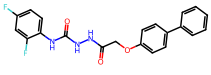
Cc1ccc(Nc2nc(N)nc(CN3CCN(c4ccccc4CC3)n2)cc1
(SAS: 2,088, LogP: 2,828, CA9: -7,7, GPX4=-7,3)

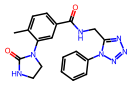
O=C(NC(=S)NNC(=O)c1cc2ccccc2cc1O)c1ccc(F)cc1
(SAS: 2,064, LogP: 2,634, CA9: -7,8, GPX4=-7,2)

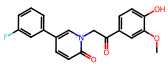
Nc1[nH+]nc(C(=O)NC(=O)c2cc3ccccc3o2)c1-c1ccccc1
(SAS: 2,516, LogP: 2,125, CA9: -7,8, GPX4=-7,2)

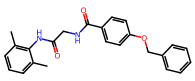
Cc1ccn(C(=O)[O-])c1C(=O)NCc1ccc(-n2ccnc2)cc1
(SAS: 2,660, LogP: 1,067, CA9: -7,0, GPX4=-8,4)

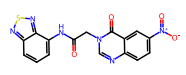

O=C(Cn1nnnc1-c2ccc(O)cc2)Nc1ccc(Cc2ccccc2)cc1
(SAS: 2,022, LogP: 3,275, CA9: −7,2, GPX4=−7,6)

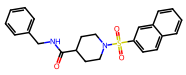

O=C(COc1ccc(-c2ccccc2)cc1)NNC(=O)Nc1ccc(F)cc1F
(SAS: 1,930, LogP: 3,864, CA9: −7,5, GPX4=−7,3)

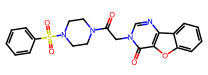

Cc1ccc(C(=O)NCc2nnnn2-c2ccccc2)cc1N1CCNC1=O
(SAS: 2,343, LogP: 1,430, CA9: −7,3, GPX4=−7,5)

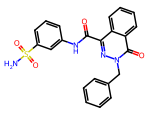

COc1cc(C(=O)Cn2cc(-c3ccc(F)c3)ccc2=O)ccc1O
(SAS: 2,120, LogP: 3,252, CA9: −7,0, GPX4=−7,9)

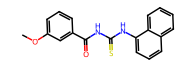

Cc1ccc(C)c(NC(=O)CNC(=O)c1ccc(OCc2ccccc2)cc1
(SAS: 1,757, LogP: 4,251, CA9: −7,0, GPX4=−7,8)

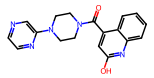

O=C(Cn1cnc2ccc([N+](=O)[O-])cc2c1=O)Nc1nccs1
(SAS: 2,440, LogP: 1,948, CA9: −8,0, GPX4=−6,9)

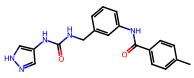

O=C(NCc1ccccc1)C1CCN(S(=O)(=O)c2ccc3ccccc3c2)CC1
(SAS: 1,869, LogP: 3,557, CA9: −7,4, GPX4=−7,3)

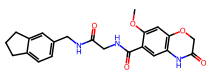

O=C(Cn1cnc2c(oc3ccccc32)c1=O)N1CCN(S(=O)(=O)c2ccccc2)CC1
(SAS: 2,309, LogP: 1,676, CA9: −7,8, GPX4=−7,0)

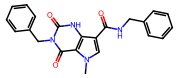

NS(=O)(=O)c1ccccc1NC(=O)c2nn(Cc3ccccc3)c(=O)c3ccccc23)c1
(SAS: 2,050, LogP: 2,345, CA9: −7,9, GPX4=−6,9)

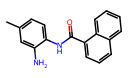

COc1ccccc1C(=O)NC(=S)Nc2ccc3ccccc23)c1
(SAS: 1,756, LogP: 3,975, CA9: −7,1, GPX4=−7,5)

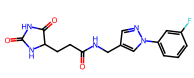

O=C(c1ccc(O)nc2ccccc12)N1CCN(c2cnccn2)CC1
(SAS: 2,305, LogP: 1,693, CA9: −7,7, GPX4=−7,0)

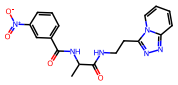

Cc1ccc(C(=O)Nc2ccccc2CNC(=O)Nc3cn[nH]c3)c2)cc1
(SAS: 2,022, LogP: 3,292, CA9: −8,8, GPX4=−6,5)

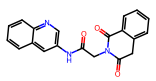

COc1cc2c(cc1C(=O)NCC(=O)NCc1ccc3c(c1)CCC3)NC(=O)CO2
(SAS: 2,428, LogP: 1,561, CA9: −7,5, GPX4=−7,1)

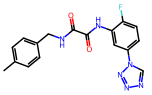

Cn1cc(C(=O)NCc2ccccc2)c2[nH]c(=O)n(Cc3ccccc3)c(=O)c21
(SAS: 2,193, LogP: 2,007, CA9: −7,2, GPX4=−7,3)

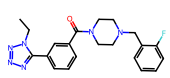

Cc1ccc(NC(=O)Cc2ccc3ccccc23)cc1
(SAS: 1,676, LogP: 3,983, CA9: −7,6, GPX4=−7,0)

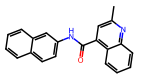

O=C(CCC1NC(=O)NC1=O)NCc1cnn(-c2ccc(F)c2)c1
(SAS: 2,898, LogP: 0,816, CA9: −7,9, GPX4=−6,7)

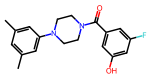

CC(NC(=O)c1ccccc[N+](=O)[O-])c1)C(=O)NCCc1nnc2ccccn12
(SAS: 2,718, LogP: 1,115, CA9: −7,1, GPX4=−7,4)

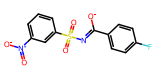

O=C(CN1C(=O)Cc2ccccc2C1=O)Nc1cnc2ccccc2c1
(SAS: 2,091, LogP: 2,398, CA9: −7,3, GPX4=−7,2)

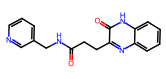

Cc1ccc(CNC(=O)C(=O)Nc2ccf(-c3nnn3)ccc2F)cc1
(SAS: 2,193, LogP: 1,365, CA9: −8,1, GPX4=−6,6)

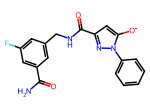

CCn1nnnc1-c1cccc(C(=O)N2CCN(Cc3ccccc3F)CC2)c1
(SAS: 2,191, LogP: 2,457, CA9: −7,0, GPX4=−7,5)

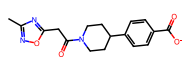

Cc1cc(C(=O)Nc2ccc3ccccc3c2)c2ccccc2n1
(SAS: 1,713, LogP: 4,949, CA9: −7,0, GPX4=−7,5)

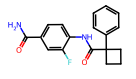

Cc1cc(C)cc(N2CCN(C(=O)c3cc(O)ccc(F)c3)CC2)c1
(SAS: 2,093, LogP: 3,111, CA9: −6,9, GPX4=−7,6)

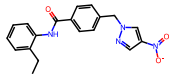

O=[N+]([O-])c1ccccc1S(=O)(=O)N=C([O-])c2ccc(F)cc2)c1
(SAS: 2,598, LogP: 1,230, CA9: −6,7, GPX4=−7,7)

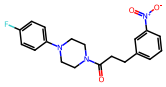

O=C(CCc1nc2ccccc2[nH]c1=O)NCc1ccccc1
(SAS: 2,077, LogP: 1,567, CA9: −7,4, GPX4=−7,1)

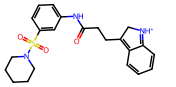

NC(=O)c1ccf(F)cc(CNC(=O)c2ccf[O-])n(-c3ccccc3)n2)c1
(SAS: 2,546, LogP: 1,114, CA9: −7,2, GPX4=−7,2)

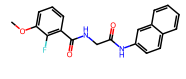

Cc1nnnc(CC(=O)N2CCC(c3ccc([O-])cc3)CC2)n1
(SAS: 2,654, LogP: 0,690, CA9: −7,9, GPX4=−6,6)

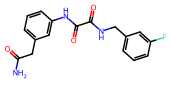

NC(=O)c1ccc(NC(=O)C2(c3ccccc3)CCC2)cf]Fc1
(SAS: 2,025, LogP: 2,985, CA9: −7,9, GPX4=−6,6)

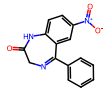

CCc1ccccc1NC(=O)c1ccc(Cn2cc[N+](=O)[O-])cn2)cc1
(SAS: 2,008, LogP: 3,654, CA9: −6,5, GPX4=−7,8)

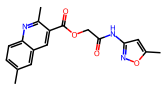

O=C(CCc1ccccc[N+](=O)[O-])c1)N1CCN(c2ccf[Fcc2)CC1
(SAS: 1,932, LogP: 3,015, CA9: −6,9, GPX4=−7,5)

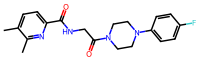

O=C(CCC1=c2ccccc2=[NH+]C1)Nc1ccccc(S(=O)(=O)N2CCCCC2)c1
(SAS: 2,994, LogP: 0,145, CA9: −7,5, GPX4=−6,9)

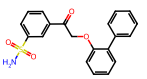

COc1ccccc(C(=O)NCC(=O)Nc2ccc3ccccc3c2)c1F
(SAS: 1,883, LogP: 3,356, CA9: −7,3, GPX4=−7,1)

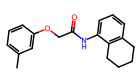

NC(=O)c1ccccc(NC(=O)C(=O)NCc2ccc(F)c2)c1
(SAS: 1,924, LogP: 1,108, CA9: −7,5, GPX4=−6,9)

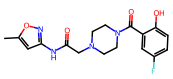

O=C1CN=C(c2ccccc2)c2cdN+](=O)[O-])ccc2N1
(SAS: 2,282, LogP: 2,384, CA9: −6,7, GPX4=−7,6)

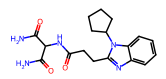

Cc1ccc2nc(C)c(C(=O)OCC(=O)Nc3ccc(C)on3)cc2c1
(SAS: 2,212, LogP: 2,944, CA9: −8,2, GPX4=−6,4)

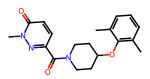

Cc1ccc(C(=O)NCC(=O)N2CCN(c3ccc(F)cc3)CC2)nc1C
(SAS: 2,116, LogP: 1,916, CA9: −7,8, GPX4=−6,7)


NS(=O)(=O)c1ccccc(C(=O)COc2ccccc2-c2ccccc2)c1
(SAS: 1,830, LogP: 3,263, CA9: −6,9, GPX4=−7,4)


Cc1ccccc1OCC(=O)Nc2ccc3c2CCC3)c1
(SAS: 1,848, LogP: 3,891, CA9: −6,7, GPX4=−7,5)


Cc1cc(NC(=O)CN2CCN(C(=O)c3cc(F)ccc3O)CC2)no1
(SAS: 2,166, LogP: 1,224, CA9: −7,9, GPX4=−6,5)


NC(=O)C(NC(=O)CC1c1ne2ccccc2n1C1CCCC1)C(N)=O
(SAS: 2,463, LogP: 0,539, CA9: −7,7, GPX4=−6,6)


Cc1ccccC)c1OC1CCN(C(=O)c2ccc(-o)n(C)n2)CC1
(SAS: 2,317, LogP: 2,081, CA9: −6,8, GPX4=−7,4)

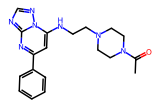
O=C(NCc1ccccc1)Nc1ccc2ncccc2c1
(SAS: 1,848, LogP: 2,952, CA9: −6,8, GPX4=−7,4)

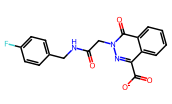
O=C(c1ccccF)c1N1CCN(c2ccc([N+](=O)[O-])c2)CC1
(SAS: 1,869, LogP: 2,696, CA9: −6,5, GPX4=−7,6)

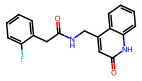
Cc1nnnc1CNC(=O)NCc1ccc2c1OCO2
(SAS: 2,525, LogP: 1,106, CA9: −7,0, GPX4=−7,2)

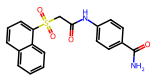
O=C(NC(=O)N1CCC(OCc2ccccc2)C1)NCc1ccccc1
(SAS: 1,961, LogP: 2,694, CA9: −7,0, GPX4=−7,2)

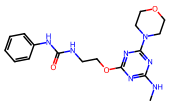
O=C(Cc1ccc(Cl)c1N)N1CCN(c2ccccc2)CC1
(SAS: 1,769, LogP: 3,668, CA9: −7,3, GPX4=−6,9)

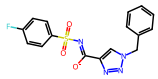

CC(=O)N1CCN(CCNc2nc(-c3ccccc3)nc3ncnn23)CC1
(SAS: 2,378, LogP: 1,367, CA9: -7,0, GPX4=-7,2)

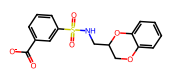

O=C(Cn1nc(C(=O)[O-])c2ccccc2c1=O)NCc1ccc(F)cc1
(SAS: 2,314, LogP: 0,215, CA9: -7,1, GPX4=-7,1)

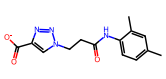

O=C(Cc1ccccc1F)NCc1cc(=O)[nH]c2ccccc12
(SAS: 1,955, LogP: 2,526, CA9: -7,1, GPX4=-7,1)

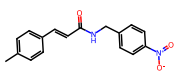

NC(=O)c1ccc(NC(=O)CS(=O)(=O)c2ccc3ccccc23)cc1
(SAS: 1,949, LogP: 2,351, CA9: -7,9, GPX4=-6,4)

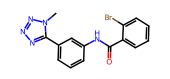

CNc1nc(OCCNC(=O)Nc2ccccc2)nc(N2CCOCC2)n1
(SAS: 2,339, LogP: 0,950, CA9: -7,6, GPX4=-6,6)

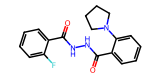

O=S(=O)(N=C([O-])c1cn(Cc2ccccc2)nn1)c1ccc(F)cc1
(SAS: 2,599, LogP: 0,961, CA9: -7,4, GPX4=-6,7)

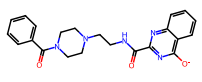

O=C([O-])c1cccc(S(=O)(=O)NCC2COc3ccccc3O2)c1
(SAS: 2,958, LogP: 0,168, CA9: -6,5, GPX4=-7,5)

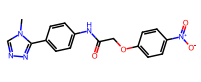

Cc1ccn(NC(=O)CCn2cc([O-])nn2)c(C)c1
(SAS: 2,491, LogP: 0,287, CA9: -7,2, GPX4=-6,9)

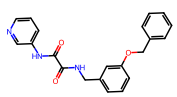

Cc1ccc(C=CC(=O)NCc2ccc([N+](=O)[O-])cc2)cc1
(SAS: 1,834, LogP: 3,233, CA9: -7,2, GPX4=-6,9)

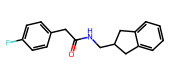

Cn1nnnc1-c1ccc(NC(=O)c2ccccc2Br)c1
(SAS: 2,002, LogP: 2,892, CA9: -6,6, GPX4=-7,4)

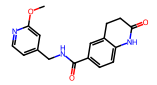

O=C(NNC(=O)c1ccccc1N1CCCC1)c1ccccc1F
(SAS: 1,842, LogP: 2,501, CA9: -7,5, GPX4=-6,6)

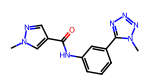

O=C(NCCN1CCN(C(=O)c2ccccc2)CC1)c1nc([O-])c2ccccc2n1
(SAS: 2,454, LogP: 0,891, CA9: -7,5, GPX4=-6,6)

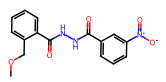

Cn1ccnc1-c1ccc(NC(=O)COc2ccc([N+](=O)[O-])cc2)cc1
(SAS: 2,073, LogP: 2,408, CA9: -7,1, GPX4=-7,0)

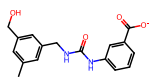

O=C(NCc1ccccc1OCc2ccccc2Cl)C(=O)Nc1ccccn1
(SAS: 1,874, LogP: 2,916, CA9: -7,0, GPX4=-7,1)

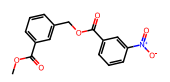

O=C(Cc1ccc(F)cc1)NCC1Cc2ccccc2C1
(SAS: 1,903, LogP: 2,899, CA9: -7,2, GPX4=-6,8)

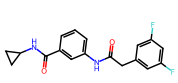

COc1cc(CNC(=O)c2ccc3c(c2)CCC(=O)N3)ccn1
(SAS: 2,180, LogP: 1,905, CA9: -6,8, GPX4=-7,3)

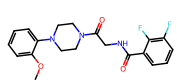

Cn1cc(C(=O)Nc2ccccc2-c3nnnn3C)c2ccn1
(SAS: 2,240, LogP: 0,863, CA9: -7,4, GPX4=-6,6)

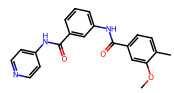

COCc1ccccc1C(=O)NNC(=O)c1ccc([N+](=O)[O-])c1
(SAS: 1,962, LogP: 1,816, CA9: -6,7, GPX4=-7,2)

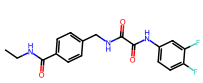

Cc1ccc(CO)cc(CNC(=O)Nc2ccc(C(=O)[O-])c2)c1
(SAS: 2,367, LogP: 1,173, CA9: -6,7, GPX4=-7,2)

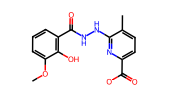

COC(=O)c1ccccc(COC(=O)c2ccc([N+](=O)[O-])c2)c1
(SAS: 1,808, LogP: 2,738, CA9: -6,7, GPX4=-7,2)

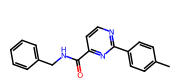

O=C(C1CC1F)Nc1ccc(F)c1Nc1cccc(C(=O)NC2CC2)c1
(SAS: 1,876, LogP: 3,038, CA9: -7,2, GPX4=-6,7)

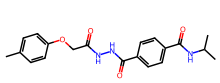

COc1ccccc1N1CCN(C(=O)CNC(=O)c2ccccc2F)CC1
(SAS: 2,031, LogP: 2,052, CA9: -7,6, GPX4=-6,4)

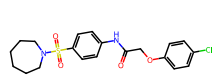

COc1ccc(C(=O)Nc2ccccc2C(=O)Nc3ccccc3Cl)cc1C
(SAS: 1,808, LogP: 3,903, CA9: -7,7, GPX4=-6,3)

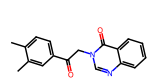

CCNC(=O)c1ccc(CNC(=O)C(=O)Nc2ccc(F)cF)c2)cc1
(SAS: 1,897, LogP: 1,969, CA9: -7,3, GPX4=-6,6)

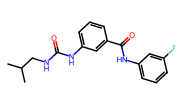

COc1ccccc(C(=O)NNc2nc(Cl)c(O[C-]3ccc2C)c3O)
(SAS: 2,724, LogP: 0,225, CA9: -6,6, GPX4=-7,2)

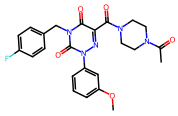

Cc1ccc(-c2ncc(C(=O)NCc3ccccc3)n2)cc1
(SAS: 1,777, LogP: 3,382, CA9: -7,0, GPX4=-6,9)

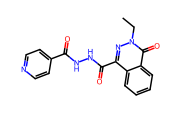

Cc1ccc(OCC(=O)NNC(=O)c2ccc(C(=O)NC(C)C)cc2)cc1
(SAS: 1,805, LogP: 1,973, CA9: -7,0, GPX4=-6,9)

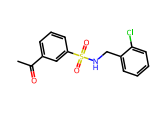

O=C(COc1ccccc1Cl)Nc1ccccS(=O)(=O)N2CCCCCC2)cc1
(SAS: 1,815, LogP: 3,922, CA9: -7,4, GPX4=-6,5)

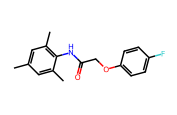

Cc1ccc(C(=O)Cn2nc3ccccc3c2=O)cc1C
(SAS: 1,855, LogP: 2,896, CA9: -7,1, GPX4=-6,7)

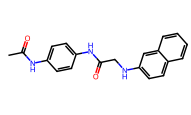

CC(C)CNC(=O)Nc1ccccc1C(=O)Nc2ccc(F)c2)c1
(SAS: 1,745, LogP: 3,856, CA9: -7,5, GPX4=-6,3)

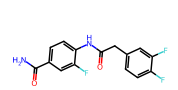

COc1cccc(-n2nc(C(=O)N3CCN(C)C)c2c(=O)c(CC3)c(=O)n(Cc3ccc(F)cc3)c2=O)c1
(SAS: 2,351, LogP: 0,894, CA9: -6,4, GPX4=-7,2)

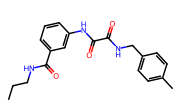

CCn1nc(C(=O)NNC(=O)c2cnccc2)c2ccccc2c1=O
(SAS: 2,122, LogP: 0,886, CA9: -6,7, GPX4=-7,0)

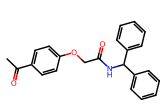

CC(=O)c1ccc(S(=O)(=O)NCc2ccccc2Cl)cc1
(SAS: 1,776, LogP: 3,021, CA9: -7,0, GPX4=-6,7)

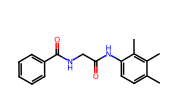

Cc1cc(Cl)c(NC(=O)COc2ccc(F)cc2)c(C)c1
(SAS: 1,685, LogP: 3,768, CA9: -7,0, GPX4=-6,7)

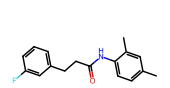

CC(=O)Nc1ccc(NC(=O)CNc2cc3ccccc3c2)cc1
(SAS: 1,725, LogP: 3,849, CA9: -7,1, GPX4=-6,6)


NC(=O)c1ccc(NC(=O)Cc2ccc(F)c(F)c2)cf)Cc1
(SAS: 1,858, LogP: 2,364, CA9: -7,1, GPX4=-6,8)


CCCNC(=O)c1ccc(NC(=O)C(=O)NNCc2ccc(C)cc2)c1
(SAS: 1,794, LogP: 2,390, CA9: -7,1, GPX4=-6,6)


CC(=O)c1ccc(OCC(=O)NC(c2ccccc2)c2ccccc2)cc1
(SAS: 1,749, LogP: 4,174, CA9: -6,3, GPX4=-7,2)


Cc1ccc(NC(=O)CNC(=O)c2ccc2)c(C)c1C
(SAS: 1,674, LogP: 2,980, CA9: -6,5, GPX4=-7,1)


Cc1ccc(NC(=O)CCc2ccc(F)c2)c(C)c1
(SAS: 1,590, LogP: 4,014, CA9: -6,5, GPX4=-7,1)

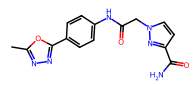
NC(=O)NC(=O)C(NC(=O)N1CCCC1)c1ccccc1
(SAS: 2,539, LogP: 0,135, CA9: -7,4, GPX4=-6,3)

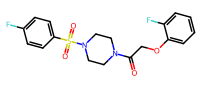
O=C(Nc1ccccc1N1CCCc2ccccc21
(SAS: 1,675, LogP: 3,671, CA9: -7,2, GPX4=-6,5)

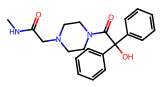
NC(=O)c1cc(-c2ccc(Br)cc2)nc2ccccc12
(SAS: 1,760, LogP: 3,763, CA9: -6,8, GPX4=-7,0)

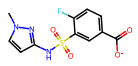
COc1nnnc1COc1ccc(C(=O)NCC(=O)N2CCOCC2)c1
(SAS: 2,304, LogP: -0,139, CA9: -6,8, GPX4=-6,8)

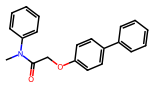
NS(=O)(=O)c1ccc(NC(=O)Nc2ccccc[N+](=O)[O-])c2)c1
(SAS: 1,897, LogP: 1,886, CA9: -6,4, GPX4=-7,1)

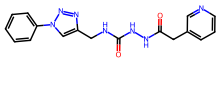

Cc1nnc(=2ccc(NC(=O)Cn3ccc(C(N)=O)n3)cc2)o1
(SAS: 2,275, LogP: 0,979, CA9: −6,9, GPX4−6,7)

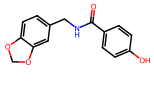

O=C(CO1ccccc1F)N1CCN(S(=O)(=O)c2ccc(F)cc2)CC1
(SAS: 1,911, LogP: 1,877, CA9: −7,2, GPX4−6,4)

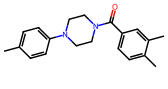

CNC(=O)CN1CCN(C(=O)C(O)(c2ccccc2)c2ccccc2)CC1
(SAS: 2,187, LogP: 0,813, CA9: −7,0, GPX4−6,8)

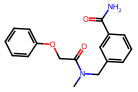

Cn1nccN(S(=O)(=O)c2cc(C(=O)[O-])ccc2F)n1
(SAS: 2,750, LogP: −0,277, CA9: −7,0, GPX4−6,6)

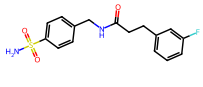

CN(C(=O)COc1ccc(-c2ccccc2)cc1)c1ccccc1
(SAS: 1,663, LogP: 4,395, CA9: −7,1, GPX4−6,5)

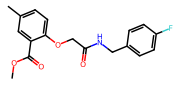

O=C(Cc1ccccc1)NNC(=O)NNc1nn(-c2ccccc2)nn1
(SAS: 2,303, LogP: 0,735, CA9: −7,1, GPX4−6,5)

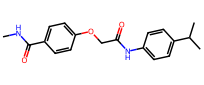

O=C(NCc1ccc2c(c1)OCO2)c1ccc(O)cc1
(SAS: 1,758, LogP: 2,051, CA9: −6,4, GPX4−7,0)

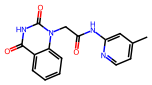

Cc1ccnc(N2CCN(C(=O)c3ccc(Cl)c(Cl)c3)CC2)cc1
(SAS: 1,660, LogP: 3,574, CA9: −6,4, GPX4−7,0)

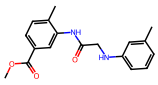

CN(Cc1ccccc1OCC(=O)N(C)C)C(=O)c1ccccc1
(SAS: 1,746, LogP: 1,823, CA9: −6,2, GPX4−7,1)

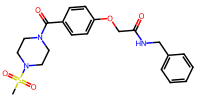

NS(=O)(=O)c1ccc(CNC(=O)CCc2ccc(F)c2)cc1
(SAS: 1,785, LogP: 1,722, CA9: −7,1, GPX4−6,4)

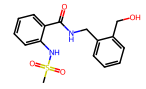

COC(=O)c1ccc(C)ccc1OCC(=O)NCc1ccc(F)cc1
(SAS: 1,764, LogP: 2,616, CA9: −6,5, GPX4−6,8)

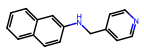

CNC(=O)c1ccc(OCC(=O)Nc2ccc(C(C)C)cc2)cc1
(SAS: 1,673, LogP: 3,187, CA9: −7,1, GPX4−6,3)

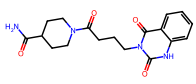

Cc1ccnc(NC(=O)Cn2c(=O)n(H)c(=O)c3ccccc32)c1
(SAS: 2,123, LogP: 1,032, CA9: −6,9, GPX4−6,5)

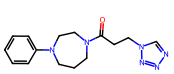

COC(=O)c1ccc(C)c(NC(=O)CNc2ccccc2C)c2)c1
(SAS: 1,751, LogP: 3,141, CA9: −6,9, GPX4−6,5)

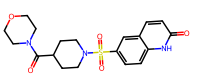

CS(=O)(=O)N1CCN(C(=O)c2ccc(OCC(=O)NCc3ccccc3)cc2)CC1
(SAS: 1,930, LogP: 1,099, CA9: −6,3, GPX4−6,8)

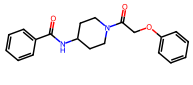

CS(=O)(=O)Nc1ccccc1C(=O)NCc1ccccc1CO
(SAS: 1,895, LogP: 1,480, CA9: −6,3, GPX4−6,8)

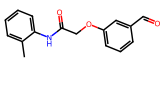

c1ccc2ccNCc3ccncc3)ccc2c1
(SAS: 1,674, LogP: 3,847, CA9: −6,8, GPX4−6,5)

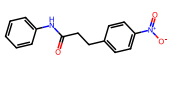

NC(=O)C3CCN(C(=O)CCCn2c(=O)[nH]c3ccccc3c2=O)CC1
(SAS: 2,175, LogP: 0,194, CA9: −6,6, GPX4−6,6)

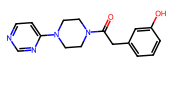

O=C(CCn1cnnn1)N1CCCN(c2ccccc2)CC1
(SAS: 2,139, LogP: 0,802, CA9: −6,6, GPX4−6,6)

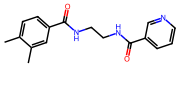

O=C(C1CCN(S(=O)(=O)c2cc3[nH]c(=O)ccc3c2)CC1)N1CCOCC1
(SAS: 2,282, LogP: 0,788, CA9: −7,0, GPX4−6,3)

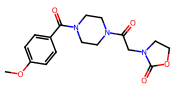

O=C(NC1CCN(C(=O)COc2ccccc2)CC1)c1ccccc1
(SAS: 1,699, LogP: 2,486, CA9: −6,9, GPX4−6,4)

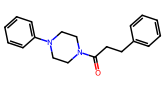

Cc1ccccc1NC(=O)COc1ccc(C=O)c1
(SAS: 1,685, LogP: 2,825, CA9: −6,0, GPX4−6,8)

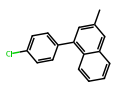

O=C(CCc1ccc([N+](=O)[O-])cc1)Nc1ccccc1
(SAS: 1,530, LogP: 3,166, CA9: −6,7, GPX4−6,5)

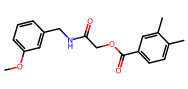

O=C(Cc1ccccc1O)c1ccN(c2ccccn2)CC1
(SAS: 2,107, LogP: 1,073, CA9: −6,3, GPX4−6,7)

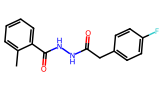

Cc1cccc(C(=O)NCCNC(=O)c2ccccn2)cc1C
(SAS: 1,770, LogP: 1,858, CA9: −6,3, GPX4−6,7)

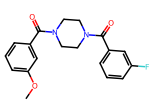

COc1cccc(C(=O)N2CCN(C(=O)CN3CCOC3=O)CC2)cc1
(SAS: 2,070, LogP: 0,432, CA9: −6,2, GPX4−6,7)

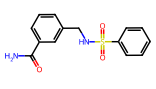

O=C(CCc1ccccc1)N1CCN(c2ccccc2)CC1
(SAS: 1,557, LogP: 2,968, CA9: −6,2, GPX4−6,7)

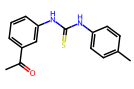

Cc1cc(-c2ccc(Cl)cc2)c2ccccc2c1
(SAS: 1,545, LogP: 5,469, CA9: −6,9, GPX4−6,3)

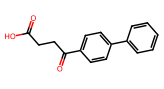

COc1cccc(CNC(=O)COC(=O)c2ccc(Cl)cc2)c1
(SAS: 1,703, LogP: 2,785, CA9: −6,1, GPX4−6,7)

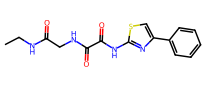

Cc1ccccc1C(=O)NNC(=O)Cc1ccc(F)cc1
(SAS: 1,662, LogP: 2,138, CA9: −6,6, GPX4−6,5)


COc1cccc(C(=O)N2CCN(C(=O)c3ccc(F)c3)CC2)c1
(SAS: 1,734, LogP: 2,433, CA9: −6,4, GPX4−6,6)


NC(=O)c1ccccc1CNS(=O)(=O)c2ccccc2)c1
(SAS: 1,668, LogP: 1,264, CA9: −6,7, GPX4−6,4)


CC(=O)c1ccc(NC(=S)Nc2ccc(C)cc2)c1
(SAS: 1,638, LogP: 4,007, CA9: −6,3, GPX4−6,6)


O=C(O)CCC(=O)c1ccc(-c2ccccc2)cc1
(SAS: 1,489, LogP: 3,401, CA9: −6,4, GPX4−6,5)


CCNC(=O)CNC(=O)C(=O)NNc1nc(-c2ccccc2)cs1
(SAS: 2,022, LogP: 1,001, CA9: −6,5, GPX4−6,4)

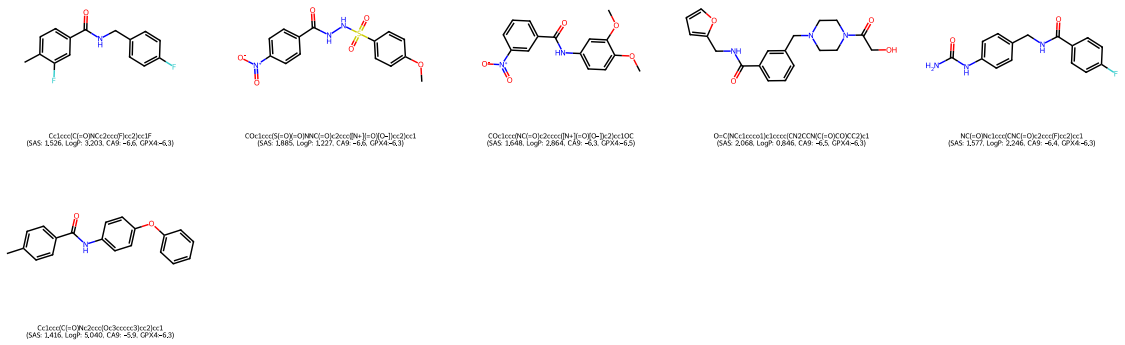


Figure A.4: Full list of the 2D graph visualization of high-quality novel samples in the final (10th) population of DEL with objectives $\{SAS, LogP, CA9, GPX4\}$ in combination with **JTVAE**, trained on the ZINC+DrugBank data. The results are ranked based on CA9 and GPX4 binding scores respectively, then sorted by the summation of two ranks.